ITP 449

Decision Trees

Lecture 11

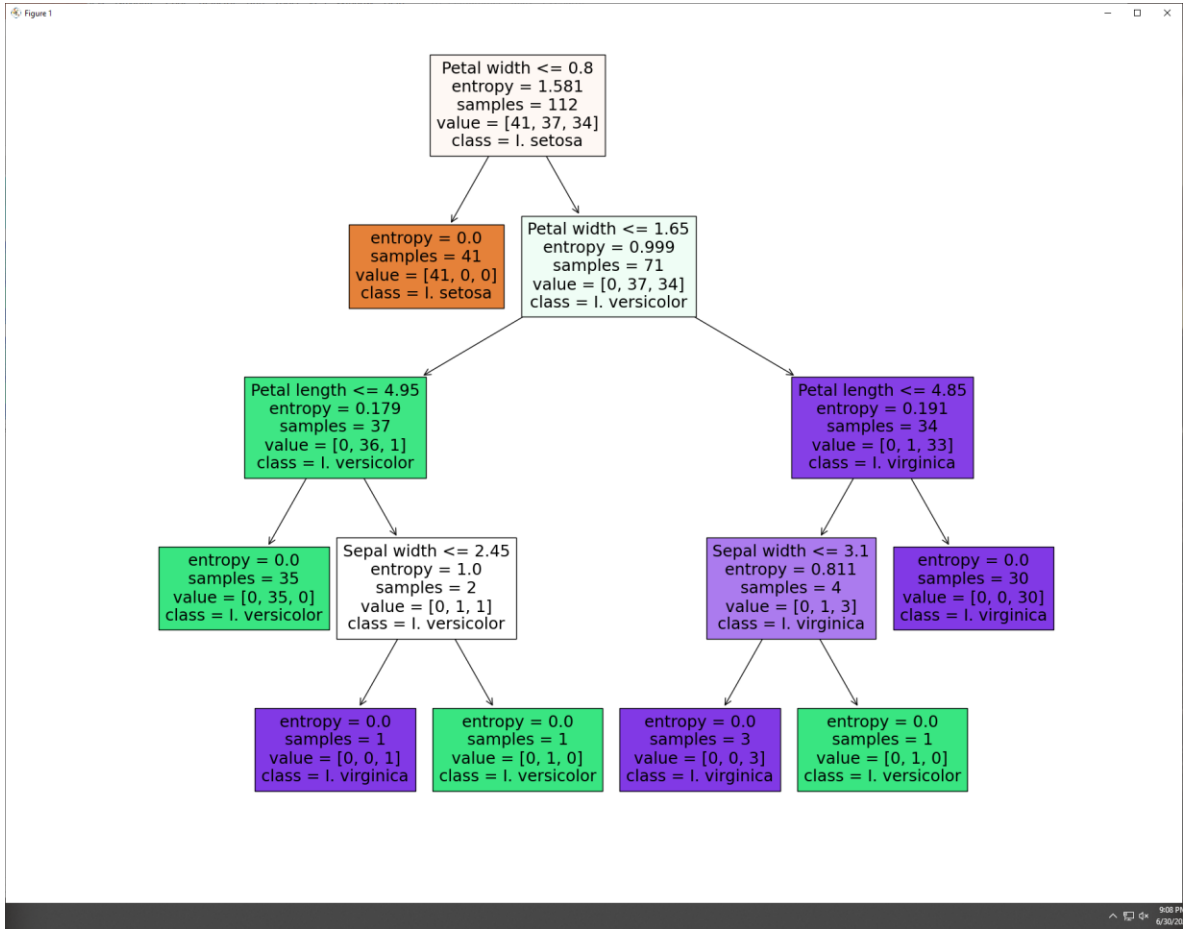


Figure 1
Petal width <= 0.8
entropy = 1.581
samples = 112
value = [41, 37, 34]
class = I. setosa
entropy = 0.0
samples = 41
value = [41, 0, 0]
class = I. setosa
Petal width <= 1.65
entropy = 0.999
samples = 71
value = [0, 37, 34]
class = I. versicolor
Petal length <= 4.95
entropy = 0.179
samples = 37
value = [0, 36, 1]
class = I. versicolor
Petal length <= 4.85
entropy = 0.191
samples = 34
value = [0, 1, 33]
class = I. virginica
entropy = 0.0
samples = 35
value = [0, 35, 0]
class = I. versicolor
Sepal width <= 2.45
entropy = 1.0
samples = 2
value = [0, 1, 1]
class = I. versicolor
Sepal width <= 3.1
entropy = 0.811
samples = 4
value = [0, 1, 3]
class = I. virginica
entropy = 0.0
samples = 30
value = [0, 0, 30]
class = I. virginica
entropy = 0.0
samples = 1
value = [0, 0, 1]
class = I. virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = I. versicolor
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = I. virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = I. versicolor

CART

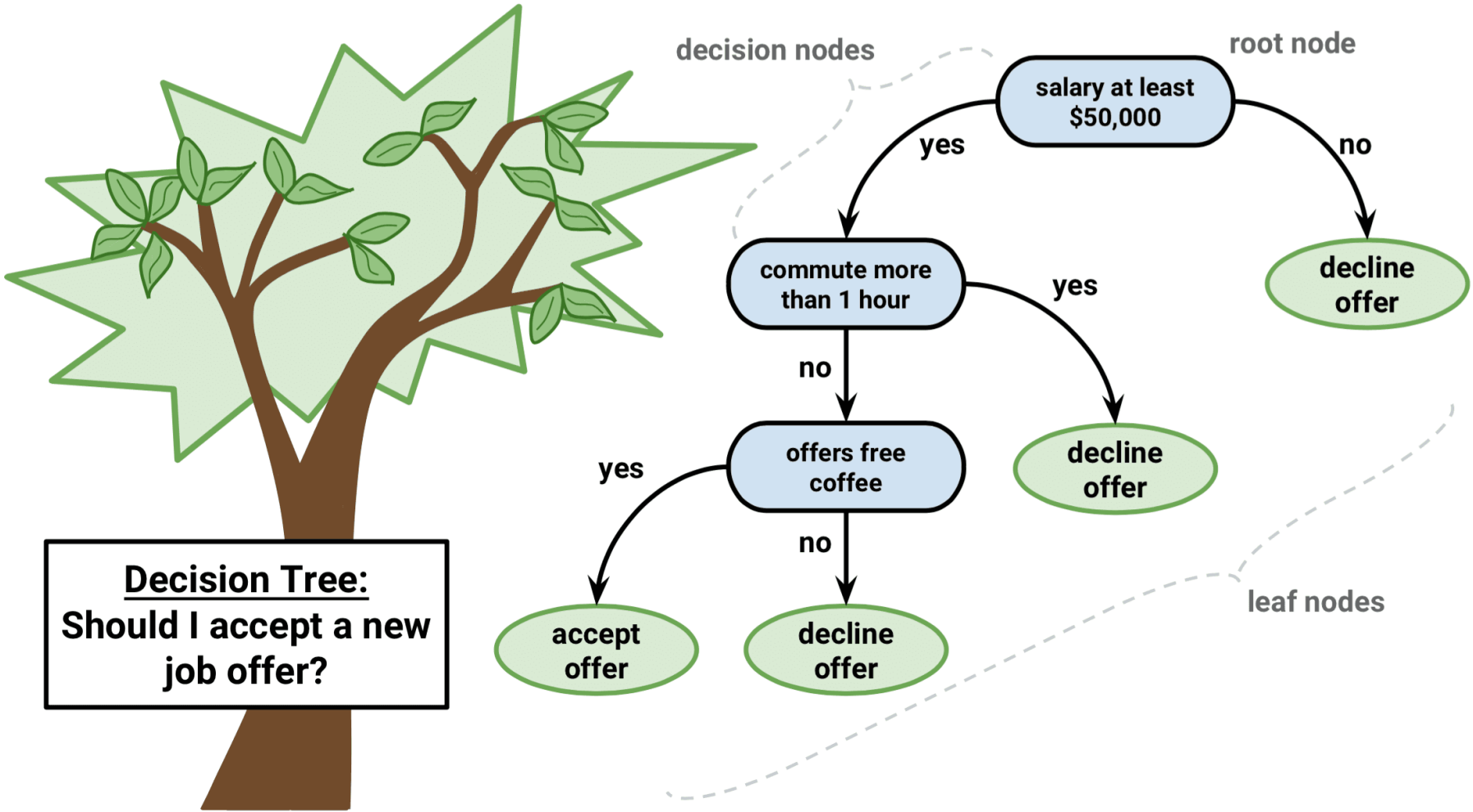
Classification and Regression Trees

Difference: **Classification** and **Regression**

- In **classification** we classify cases into specific **classes**. For example, we might take a set of people and classify them as potential customers or non-customers (based on some predictor attributes)
- In **regression**, we are trying to predict a **numerical** value – like the annual income of a person, the price of a car, the sales volume or GPA or whatever.
- As the name implies, CART can be used for both purposes!
- In this lecture we will only look at the use of CART for **classification**.

Decision Trees

- *Supervised* classification algorithm that is used when the target variable is a categorical variable and the predictor variables are either categorical or numerical
- Utilizes a set of *if-then* rules, represented as a tree, for classification
- *Visual* output of the model makes it easier to understand and implement compared to other predictive models


decision nodes
root node
salary at least $50,000
yes
no
commute more than 1 hour
yes
decline offer
no
offers free coffee
yes
decline offer
no
accept offer
decline offer
leaf nodes
Decision Tree:
Should I accept a new job offer?

Homogeneity

- The main goal in a decision tree algorithm is to identify a *variable* and *classification* that results in a more homogeneous distribution with reference to the target variable
- The homogeneous distribution means that similar values of the target variable are grouped together so that a *concrete* decision can be made

Income	Education	Family size	Type of car
200,000	1	4	Luxury
30,000	1	4	Compact
75,000	2	3	High-end
90,000	3	6	Economy
30,000	2	2	Compact
125,000	2	2	Luxury
130,000	3	6	Economy
300,000	3	2	Luxury
250,000	2	3	Luxury
128,000	2	5	Economy
35,000	3	1	Compact
…	…	…	…

Rule Archeology

Income >= 200,000 → Luxury

Income <= 35,000 → Compact

(100,000 <= Income <= 200,000) and family size < 3 → Luxury

(100,000 <= Income <= 200,000) and family size >= 5 → Economy

- Looking at the data on the type of car that people own along with other demographic information (made up for illustration), we might be able to infer some underlying “rules” about the kinds of cars that people own.

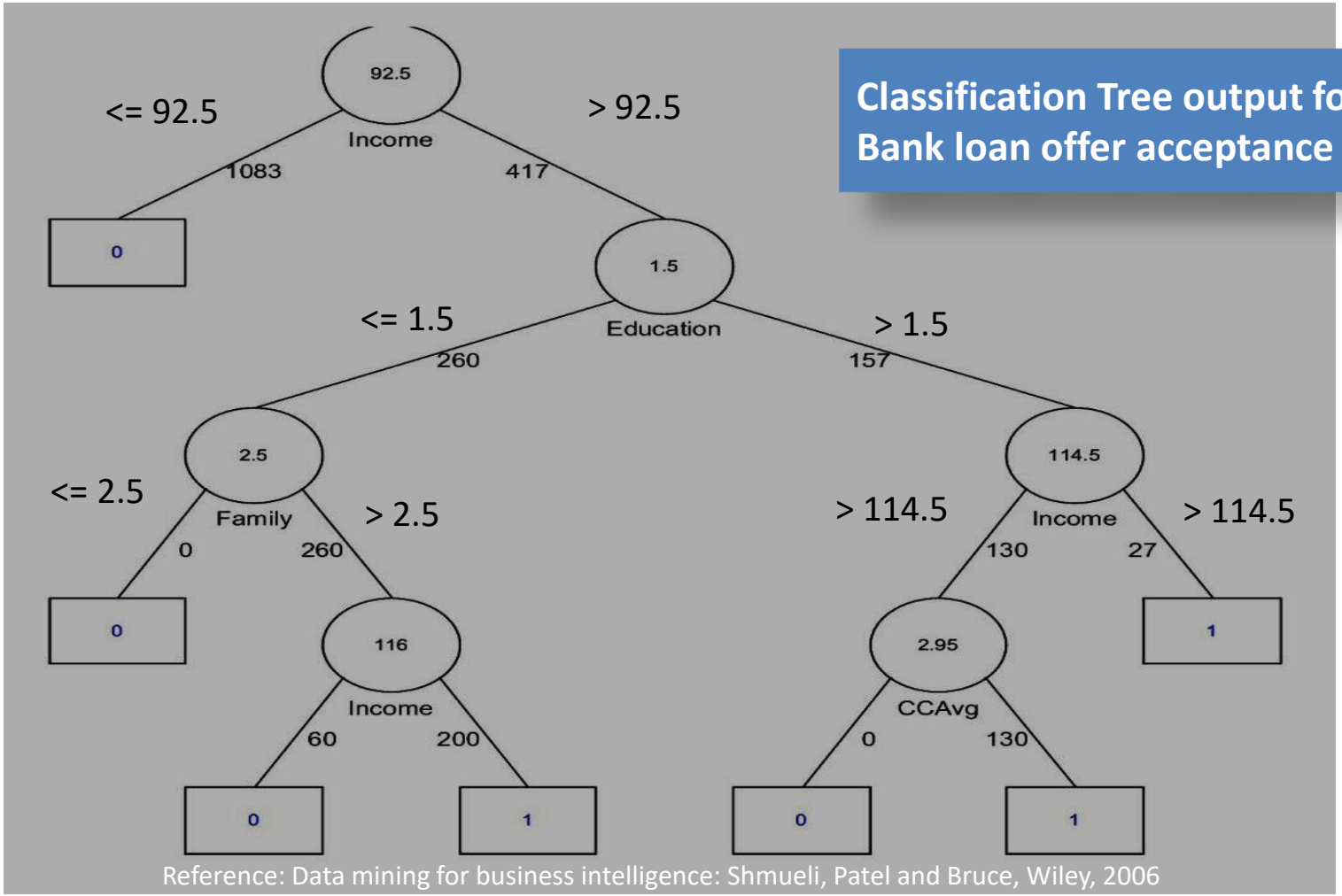
- With the caveat that we have very little data above and cannot really make any serious inferences, the discussion that follows is only intended to illustrate what we will be studying this week. To really make serious inferences we need much more data.

- From the data we might observe some patterns:

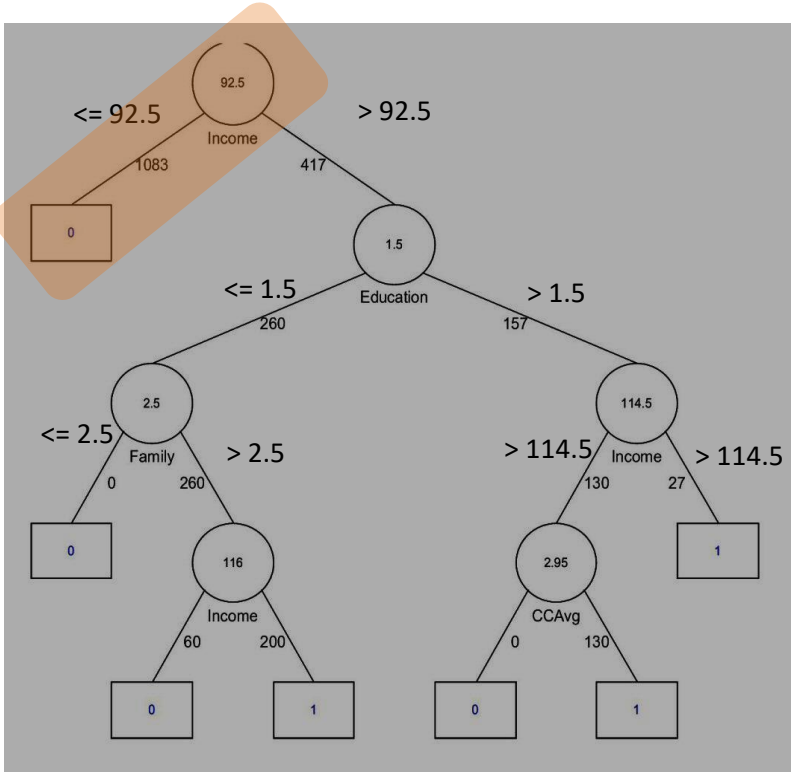
  - People making $200,000 or more seem to own a luxury car independent of other factors.
  - People making $35,000 or less seem to be owning a compact car, independent of other factors.
  - For people making between $100,000 and $200,000, it seems that they own luxury cars when the family size is 2 or less and own an economy car when the family size is 5 or more.

LOAN EXAMPLE

Classification Tree output for Bank loan offer acceptance data

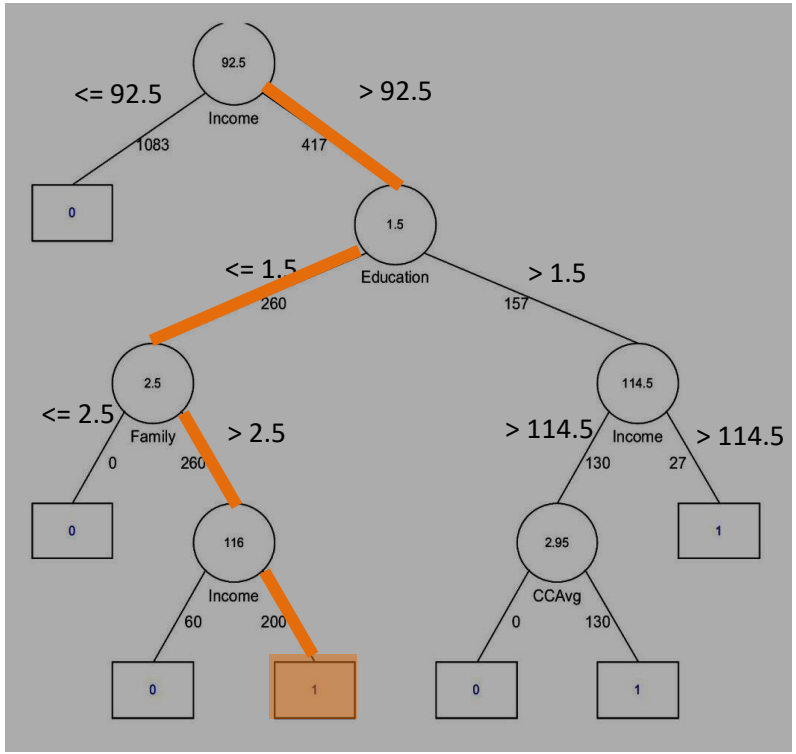


Reference: Data mining for business intelligence: Shmueli, Patel and Bruce, Wiley, 2006



The tree implies rules. The simplest rule is highlighted.

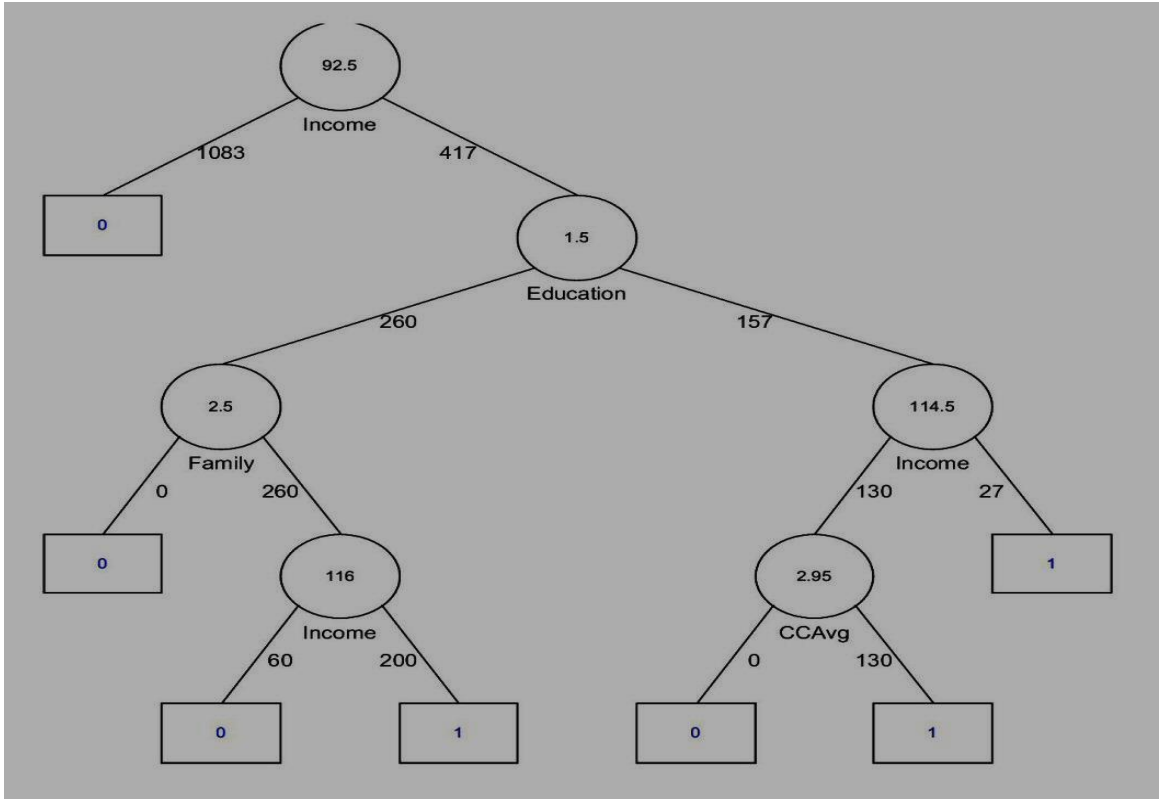
If income <= 92.5 then Non acceptor



A more complex rule that the tree shows.

If income > 92.5 and education <= 1.5 and family size > 2.5 and income > 116 then Acceptor

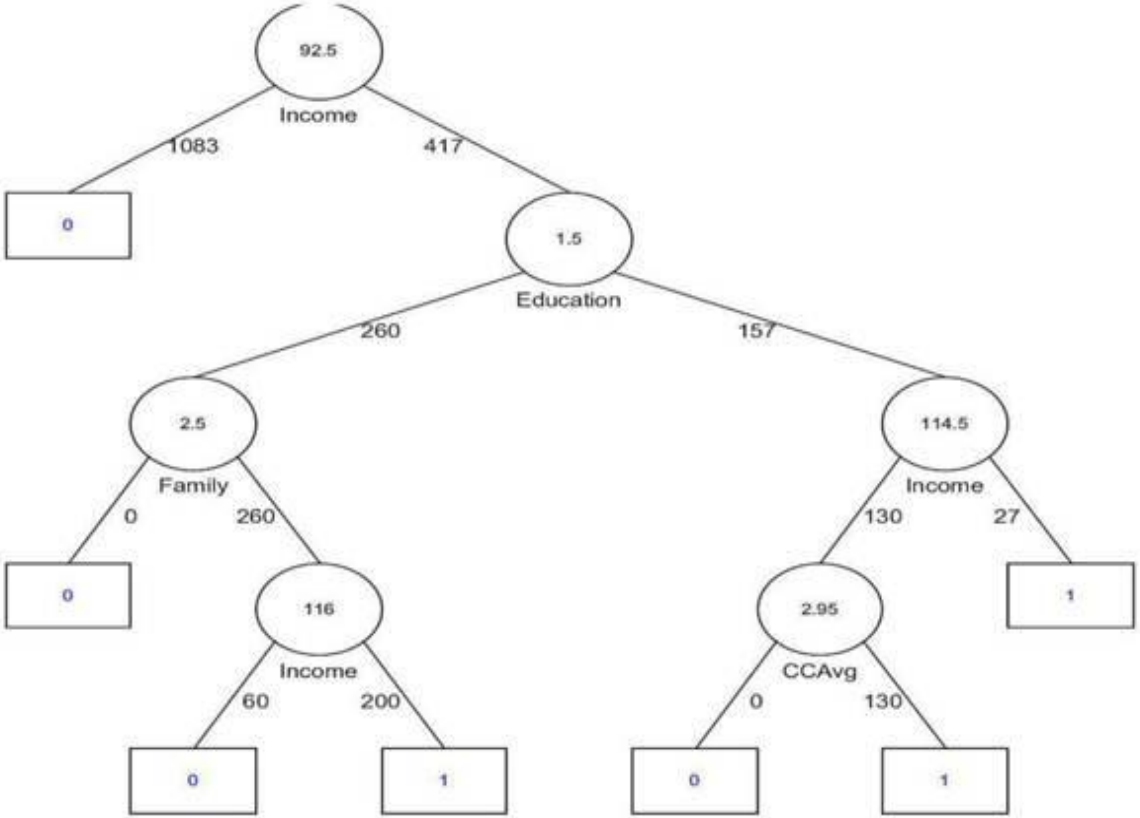
Can you condense this rule?

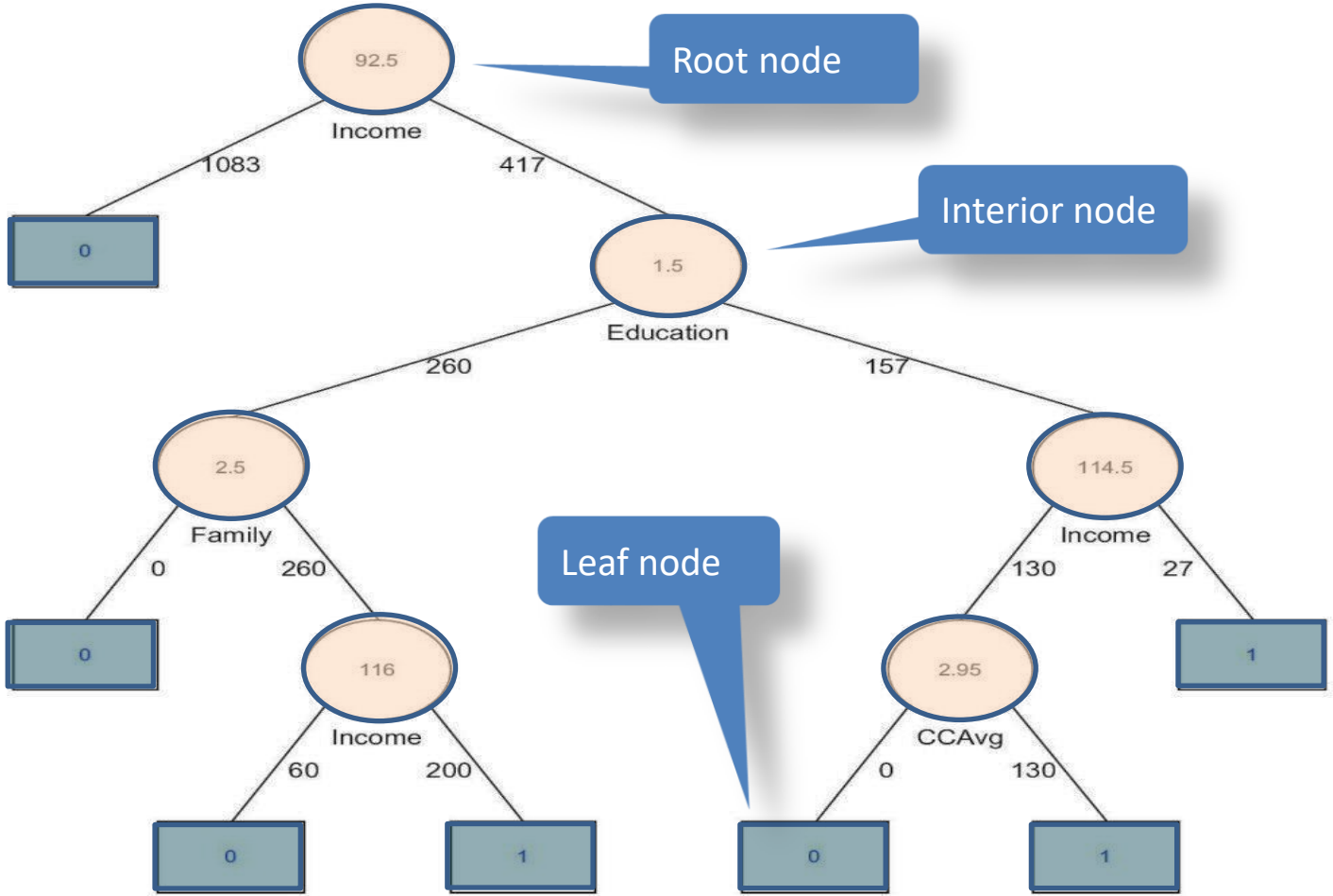


Find two more rules.

How many rules in all?

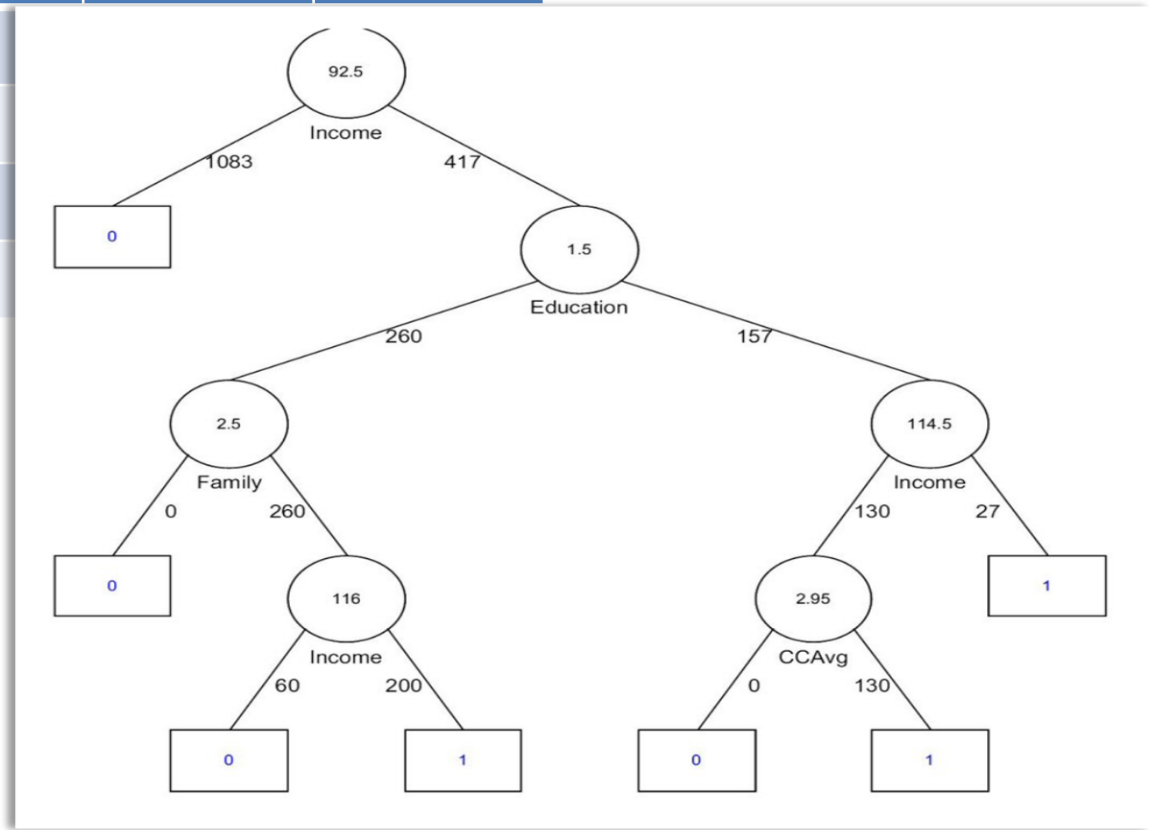
Tree?


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


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

Training Partition

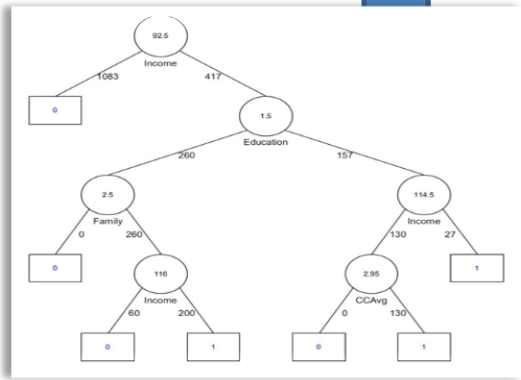
Income	Education	Family Sz	CC Avg	Acceptor?
…	…	…		
…	…	…		
…	…	…		
…	…	…		

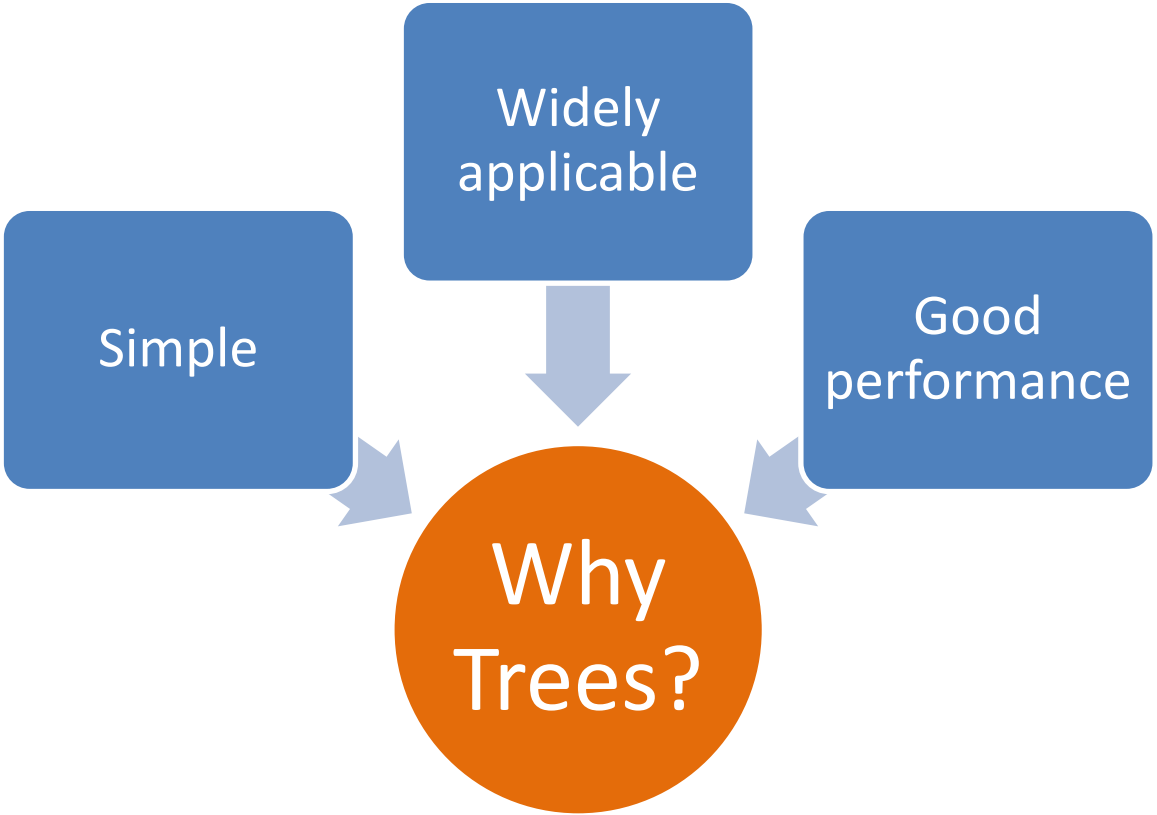


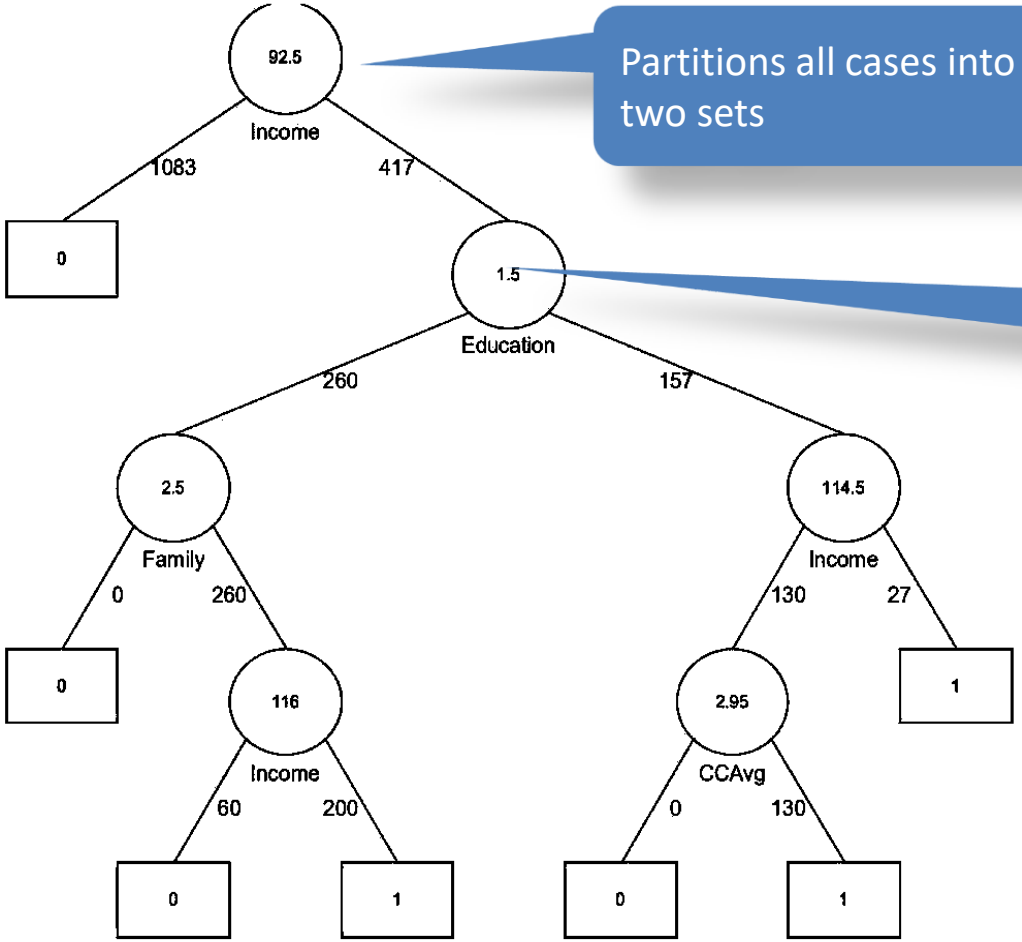
Test Data

Income	Education	Family Sz	CC Avg	Acceptor?	Model says
…	…	…	…	Yes	No
…	…	…	…	No	Yes
…	…	…	…	No	No
…	…	…	…	…	…

How good is the model?




Widely applicable
Simple
Good performance
Why Trees?

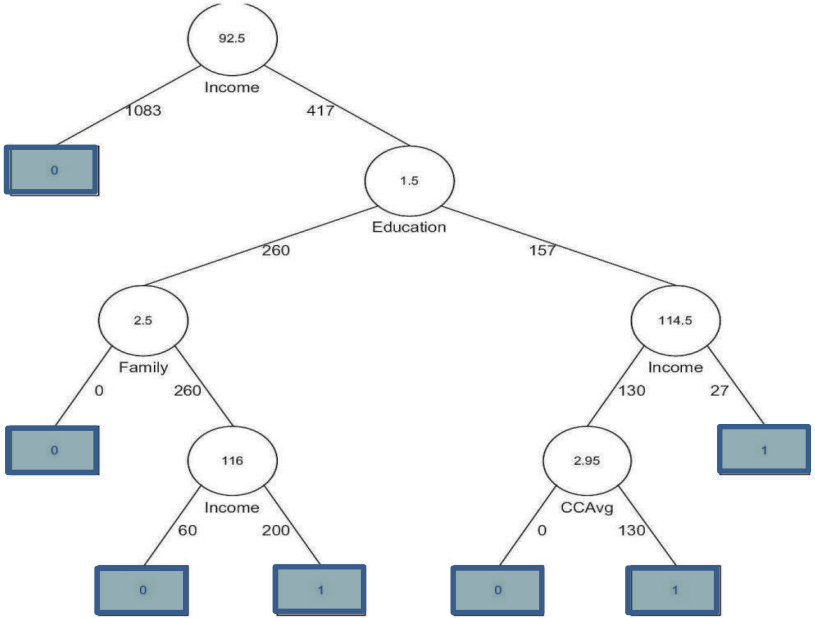


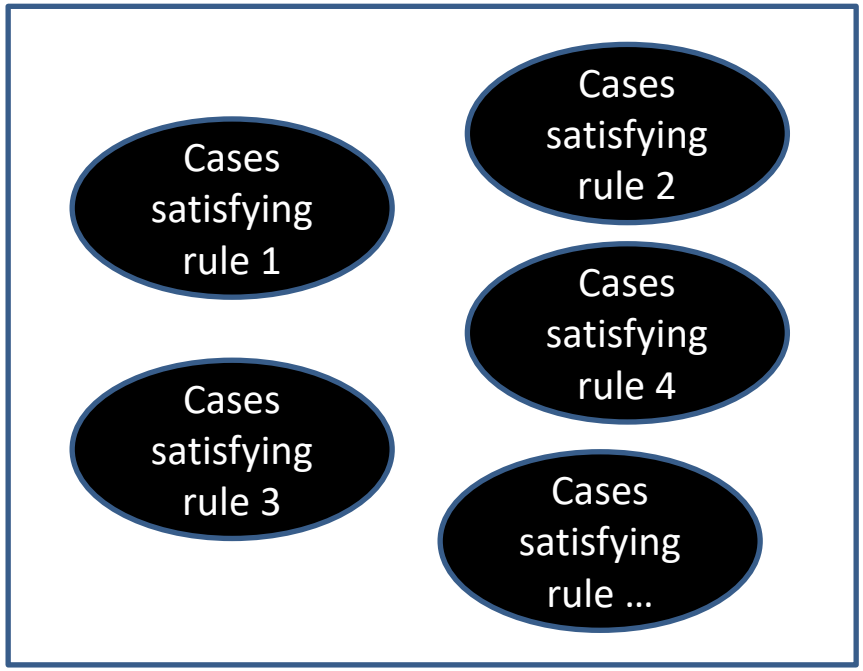
Partitions all cases into two sets

Partitions all cases with income > 92.5 into two sets

Each “decision node” partitions its base cases into two sets based on some criterion

What does a set of rules do?





A complete set of rules **divides** all the cases into disjoint sets such that each set of cases matches a **single** rule.

The above statement is not entirely accurate. It represents an idealized situation that will seldom be applicable in the real world.

It might initially appear that we can always achieve a set of rules that provide perfect matches – after all can we not create a rule for every case? If we do that we will have a huge number of rules, but they will all be perfect. Does this seem to be valid?

In reality the above approach will not generally be valid. We could have two cases with identical values for all the predictor variables, but having different values for the target variable. Concretely, we might have two cases with the same values for income, education, family size and credit card average and yet one of them accepted the loan offer and the other did not. In this case, no matter what rule matches these two cases, the rule can **never** be correct for both cases because they differ in the value of the target variable.

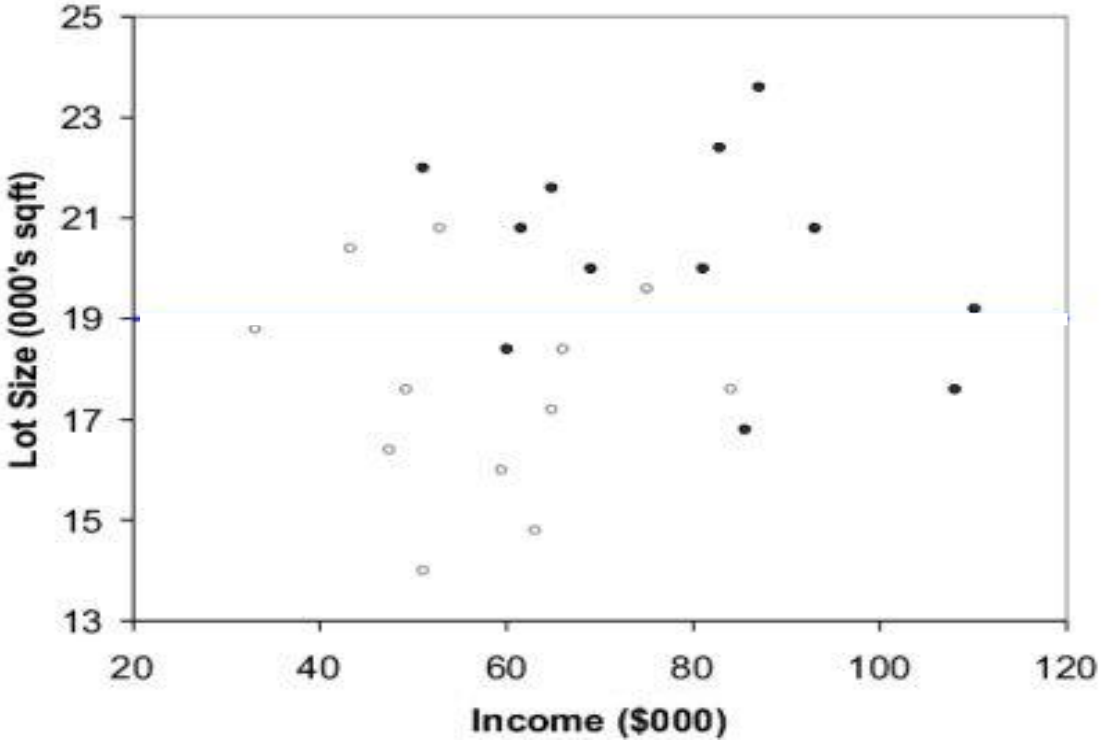
From the above example, we can see clearly that we will not pursue perfect rules. In machine learning we should not expect perfection. Instead we are looking for patterns that apply in enough cases to be of some use.

We are looking for **lift** – that is to beat the averages by a reasonable margin so that applying our models is better than doing things randomly or using more or less obvious ideas.

We see that the set of rules partitions the cases, with each partition consisting of the result of applying a specific rule.

Although we cannot strive for perfection, we still want the partitions to be as homogeneous as possible. Suppose that a particular rule applies to 100 cases in our data set and these cases belong to several different target groups (say, many acceptors and many non-acceptors as well), then the rule might not be as good as another that applies to a majority of a single target group – for example, most of the applicable cases are acceptors. In this case, the rule is perhaps doing a good job of finding an underlying pattern.

Ownership of riding mower example



This previous picture shows the all the data points. Owners are shown with dark dots.

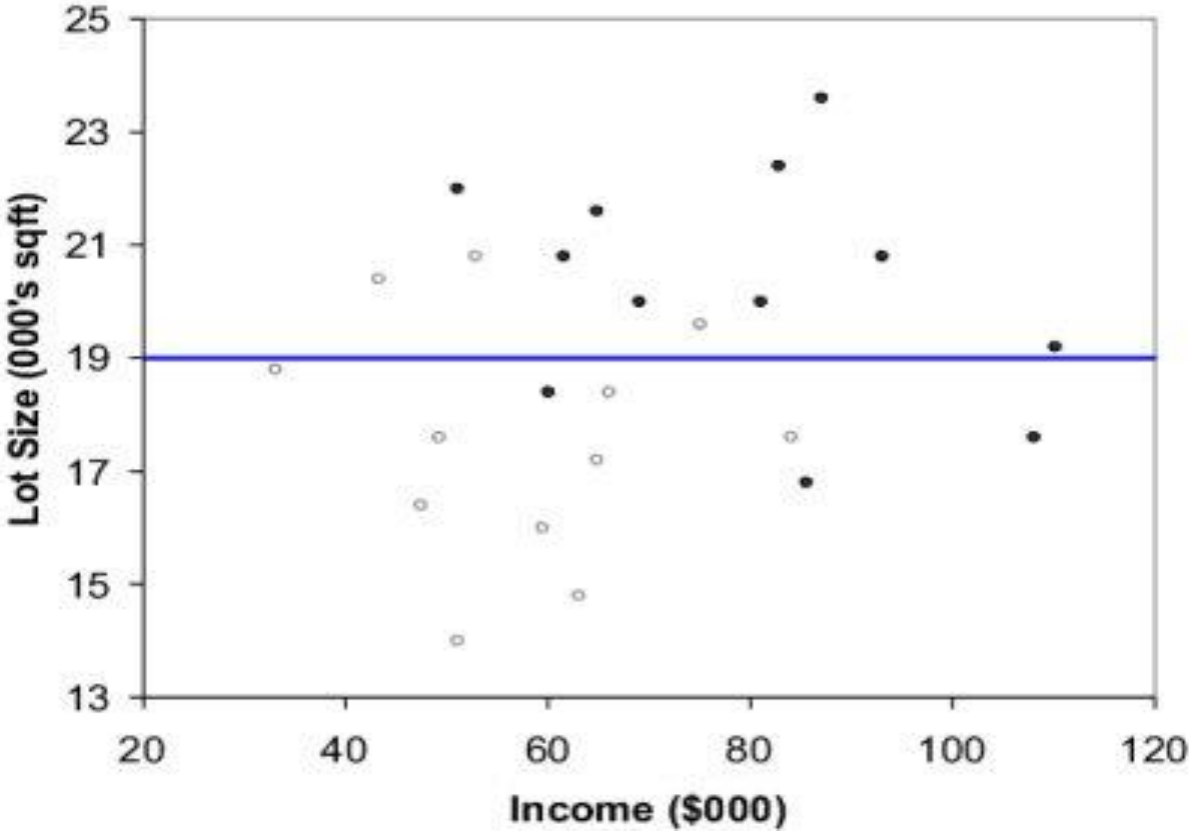
As we observed earlier, while owners generally have high incomes and lot sizes, there are owners with somewhat low incomes and somewhat low lot sizes too.

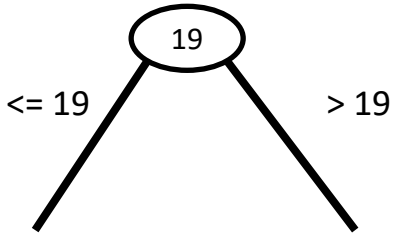
We have seen that rules effectively partition the data set into a sets of disjoint cases with each set being predominantly homogeneous (say acceptors or non-acceptors). Our procedure for finding rules therefore revolves around building these homogeneous partitions. Once we have that, the rules simply follow.

We want to divide this whole space into regions that contain predominantly owners or non-owners, and derive rules from there.

This procedure is called “**recursive partitioning**”

Partitioning





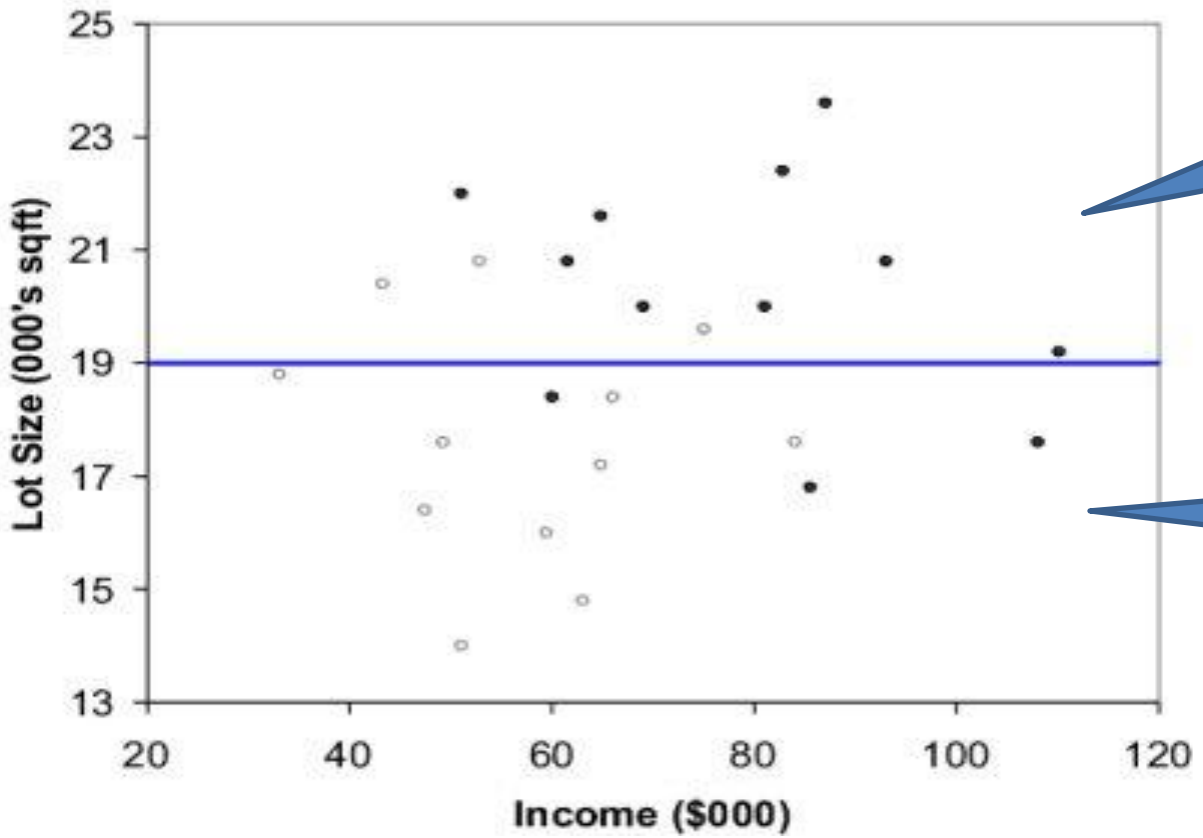
In the above case, our overall region is a rectangle (because we have two variables and therefore have a two-dimensional space) and we have shown above a partition based on lot size. We have shown a line which divides the whole region into two parts. All values above 19 are in one partition and all values <= 19 are in the other.

We could have chosen to partition by income rather than lot size. Furthermore, having chosen lot size, we could have partitioned at several places.

Why lot size? Why 19?

Before we look at those issues, take a look at the figure – do you think that the partition is “good?” What would be your criterion o for deeming a partition to be “good?”

“Good” Partition Decision?



Owners predominate

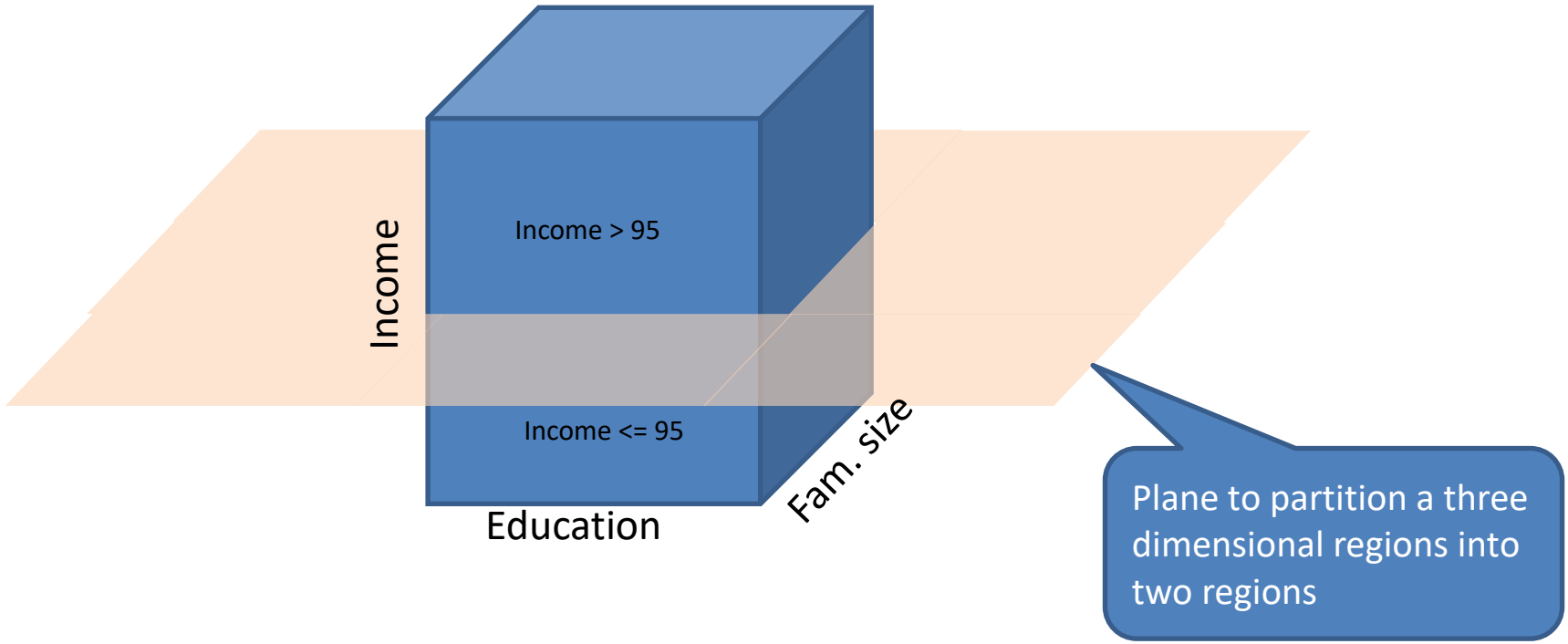
Non-owners predominate

Let us think back to what we are trying to achieve with partitioning – we are trying to create regions with a lot of homogeneity. That is, we want to create regions that have predominantly owners or non-owners.

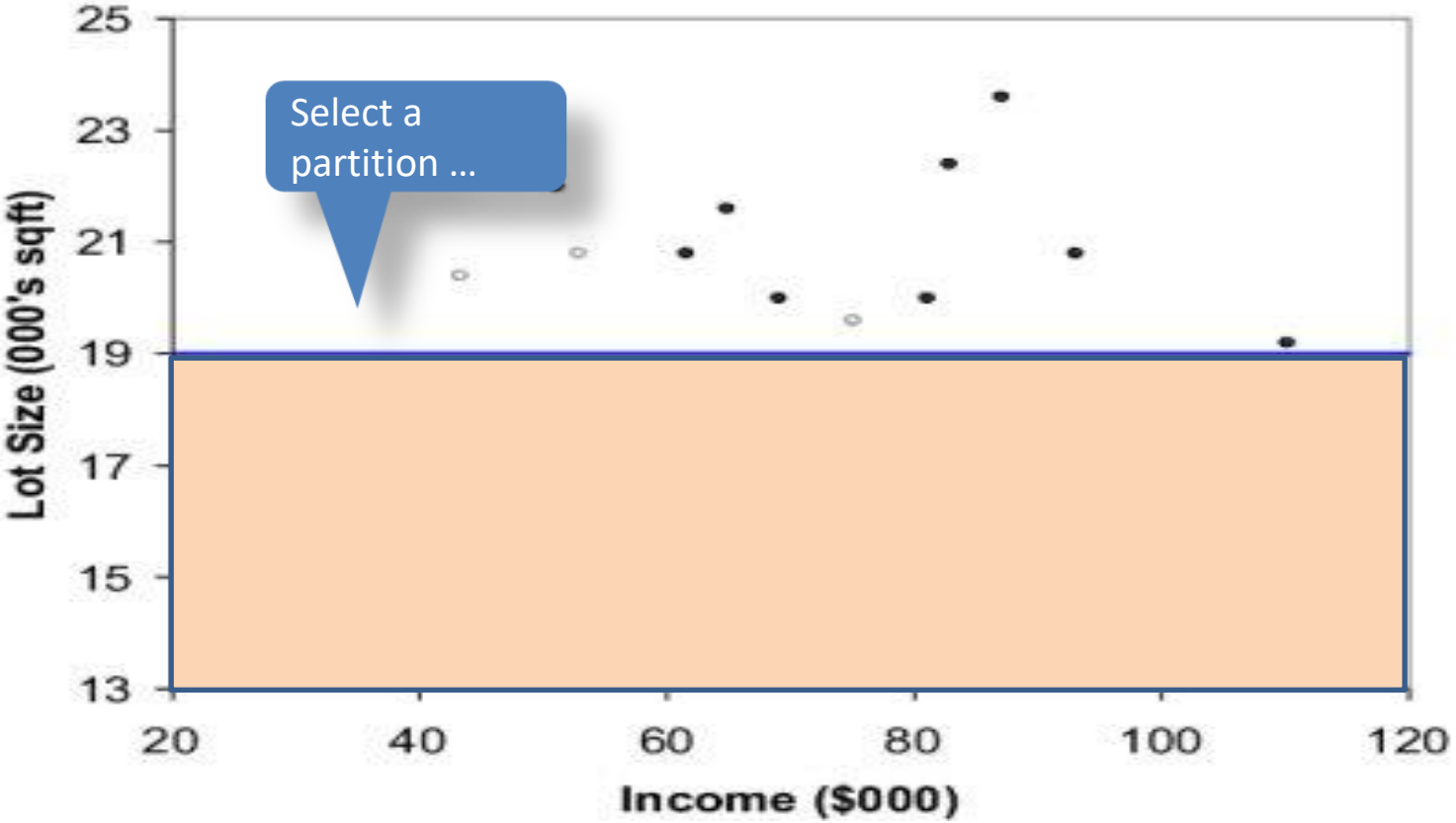
The partition at 19 seem like a good one because the resulting two regions are much more homogeneous than the original partitioned space was. We now have a majority of owners in the top region and a majority of non-owners in the bottom partition. Visually observe that drawing the partition at any other value of lot size would not have helped us to do better because the two resulting regions would have been less homogeneous.

If we had three variables instead of two, then each partition will be a three dimensional space and partitioning is achieved by means of a plane as shown in the next slide.

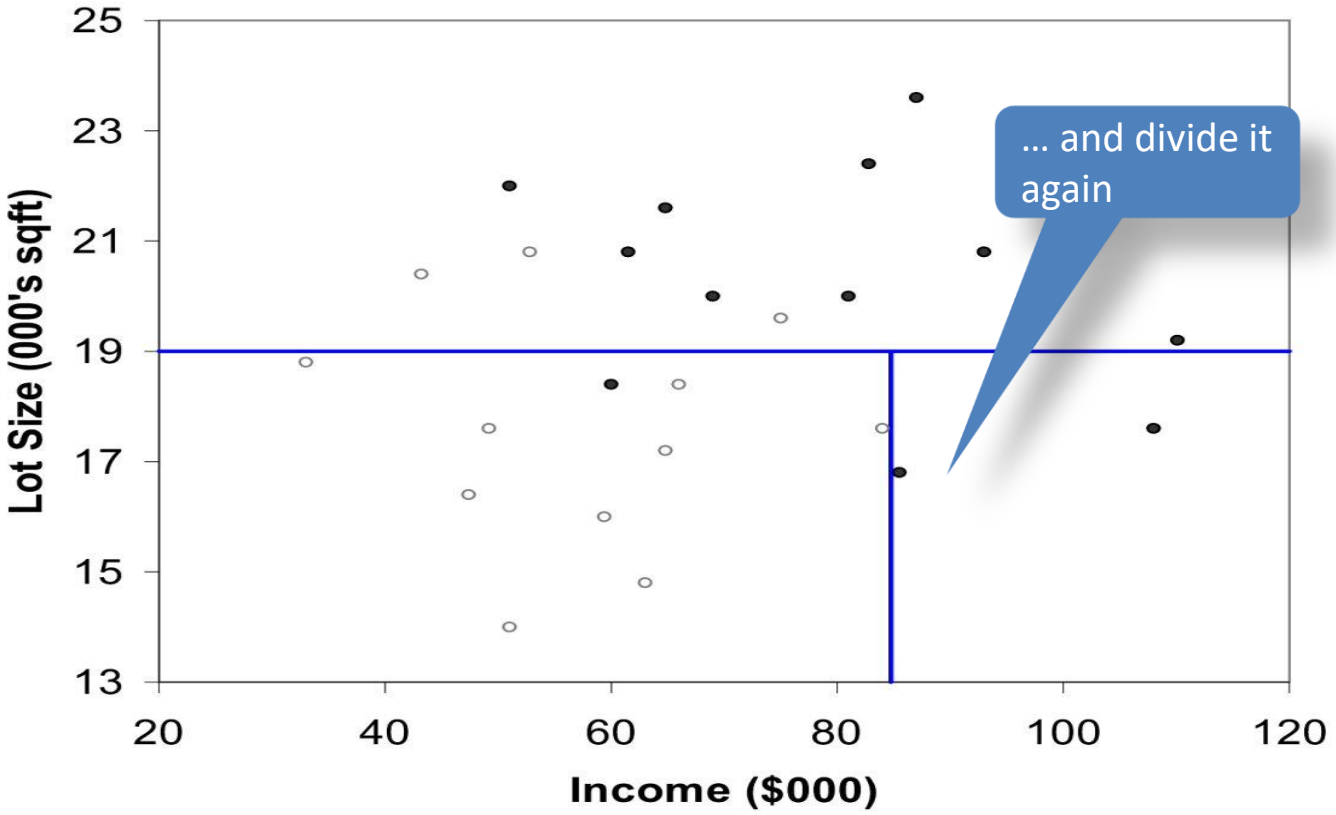
Partitioning in Three Dimensions

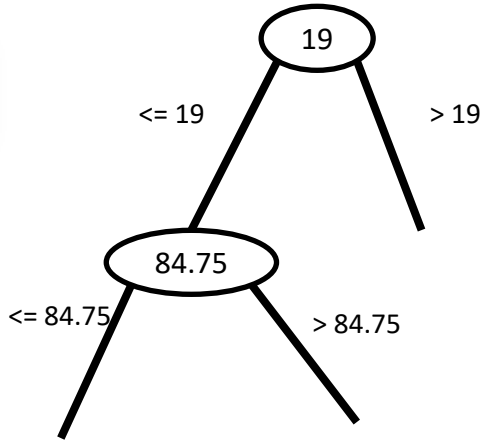


Recursive Partitioning



Recursive Partitioning





... and again divided into two, this time based on income. Those cases with income less than or equal to 92.5 went to one partition and the rest went into another.

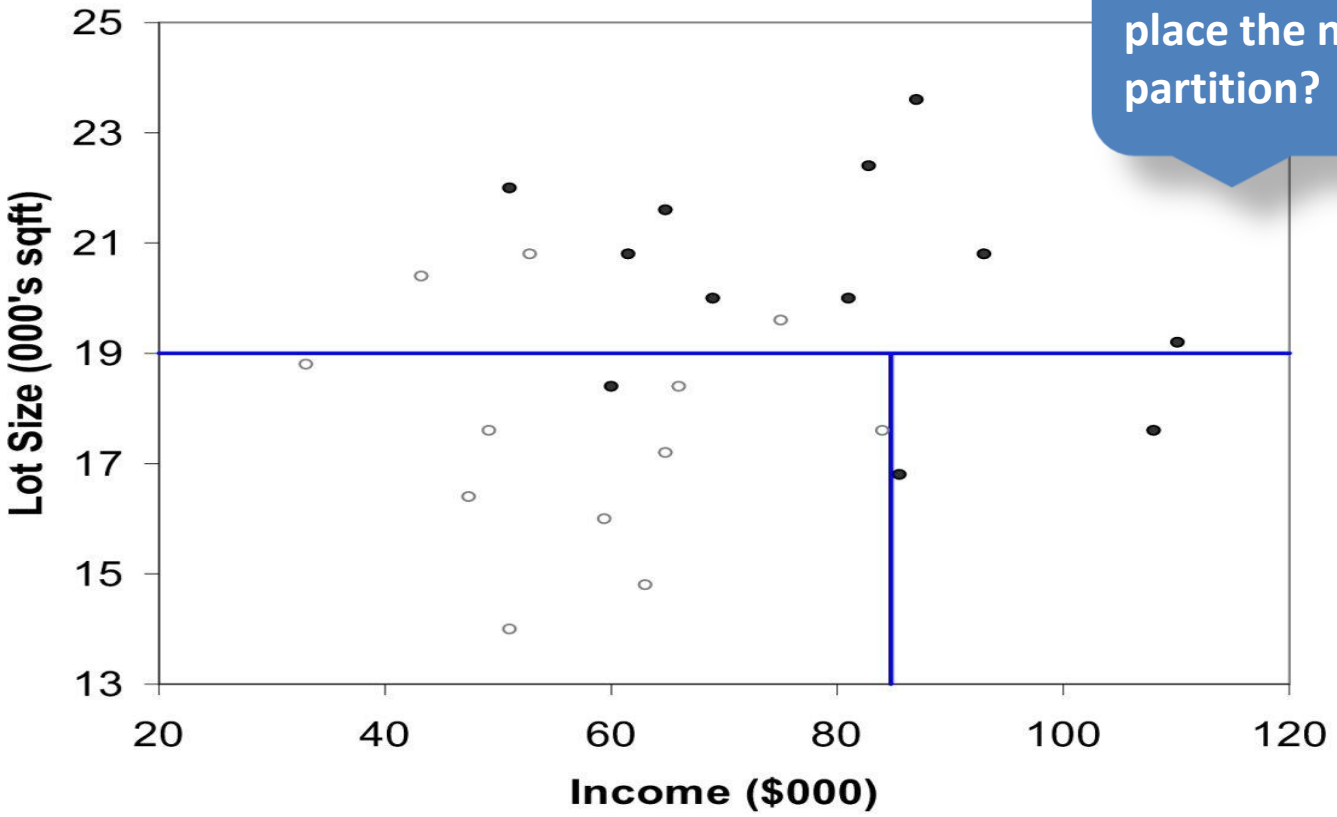
Note however that we are only partitioning the selected region. The top portion remains as is for now.

Several big questions remain:

- Why did I pick the variable lot size selected for the prior partition?
- Why did I choose to partition at 19 and not some other value?
- Why did we select the bottom region?
- Why did we select Income as the variable for partitioning?
- Why did we select 92.5 as the specific value to partition on?

We have already hinted that “homogeneity” of the regions resulting from a partitioning step guides our choices.

Your Turn


Where would you place the next partition?
25
23
21
19
17
15
13
Lot Size (000's sqft)
20
40
60
80
100
120
Income ($000)

Take a look at the situation and consider where you might choose to place the next partition. Remember, you have to select a region (from among the three above) and also select where exactly to draw the line.

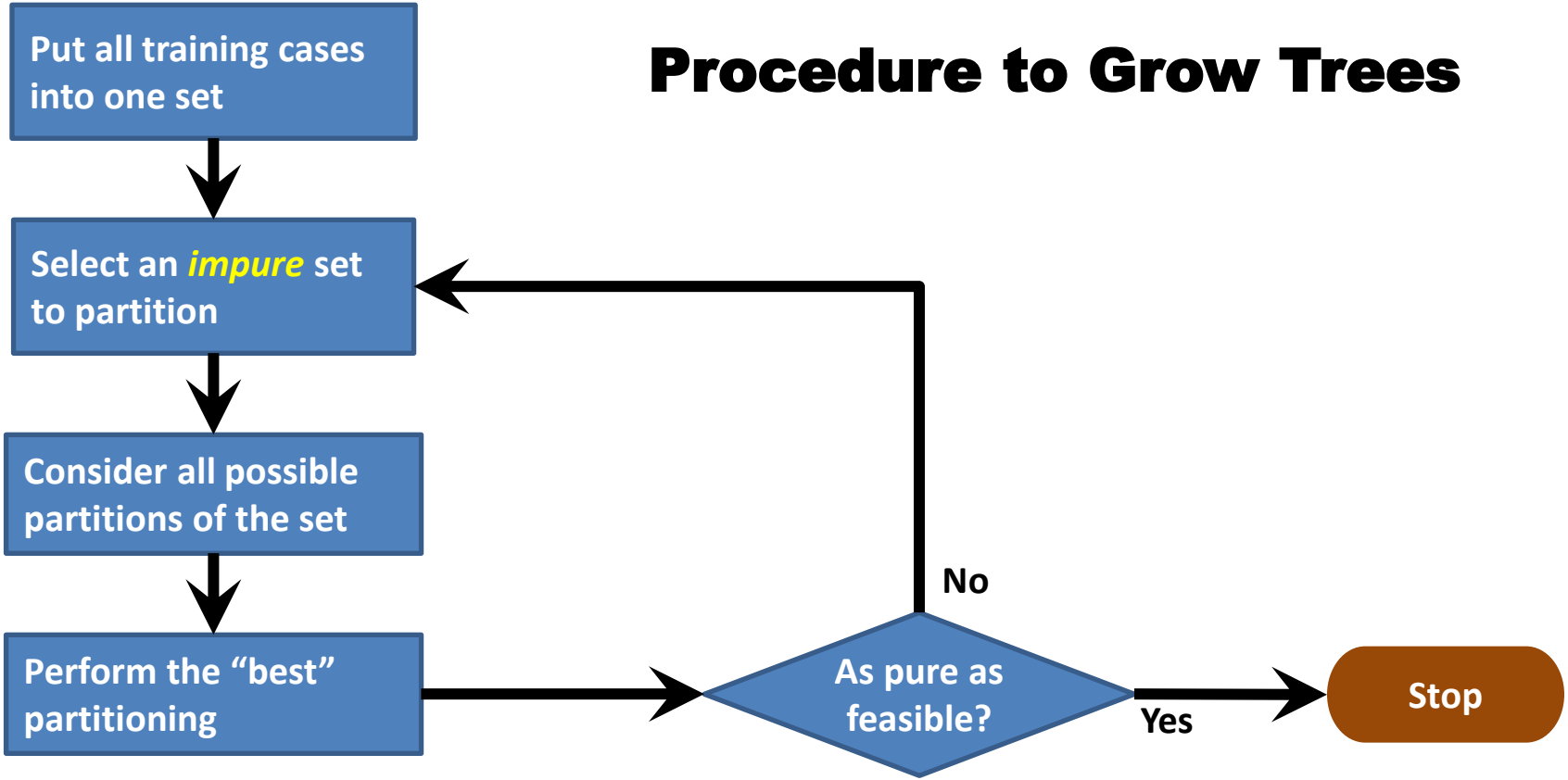
For example, suppose you chose the topmost region, you then have to decide whether to partition on lot size or income. If you choose lot size, then you will select the value to partition on. Again suppose you choose 23 (just for illustration only, might not be a good choice), then you are effectively drawing a horizontal line running through the selected region at 23 on the Y axis

Perfection not always possible.

Under what conditions might obtaining perfect partitioning be impossible?

Answer: If two cases have the same values for the predictor variables and differ only in their values for the target variable, then getting a set of rules that performs perfectly is impossible.

Procedure to Grow Trees



We want each group to consist only of one type of case (Owner or Non-owner).

Therefore, we need a measure of the extent to which a group is impure (or lacks homogeneity). Clearly, if all the cases in a group are owners or all are non-owners, then the group is pure. On the other hand the maximum amount of impurity occurs when the group is perfectly heterogeneous. In the present example that would mean that the group consists of 50% owners and 50% non-owners. This is the maximum possible deviation that we can get from absolute purity when every member is of the same type. (A 60-40 or 70-30 mix of owners and non-owners is purer than a 50-50 mix because there is a tendency for the predominance of some group.)

When we partition, we change the measure of impurity.

At each stage, among all possible partitions, we select the one that results in the best improvement in the measure of purity. This is the same as saying that we choose the split that results in the best possible improvement in overall purity.

Homogeneity

- The main goal in a decision tree algorithm is to identify a variable and classification that results in a more homogeneous distribution with reference to the target variable

- The homogeneous distribution means that similar values of the target variable are grouped together so that a concrete decision can be made

Entropy

A measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i)$$

Where H is the entropy. X is the dataset. P(x) is the proportion of x in X, n is the number of classes in the target variable

Information Gain

The amount of information gained about a random variable from another random variable. The difference between the entropy of *parent* node and weighted average entropy of *child* nodes.

$$IG(S, A) = H(S) - H(S, A)$$

Alternatively,

$$IG(S, A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

Where IG(S,A) is the difference in entropy.
H(S) is the entropy of S
H(S,A) is the entropy of S given A.

Gini Index: A Measure of Impurity of a Set

m: number of classes → 2 – Owner and Non-Owner

p_k: proportion of elements in class k → 12 Owners and 12 Non-Owners p_1 and p_2 equal 0.5

$$Gini(x) = 1 - \sum_{k=1}^{m} p_k^2$$

$$0 \text{ to } \frac{(m-1)}{m}$$

Gini Calculation for Initial Set

m	2 – Owner and Non-Owner
p_1	0.5: 12 Owners out of 24
p_2	0.5: 12 Non-owners out of 24

$$Gini(x) = 1 - \sum_{k=1}^{m} p_k^2$$

$$1 - (0.5^2 + 0.5^2) = 0.5$$

Entropy: Another Impurity Measure of a set

m: number of classes → 2 – Owner and Non-Owner

p_k: proportion of elements in class k → 12 Owners and 12 Non-Owners p_1 and p_2 equal 0.5

$$E(s) = -\sum_{k=1}^{m} p_k \log_2 p_k$$

0 to 1

Entropy Calculation for Initial Set

m

2 – Owner and Non-Owner

$$E(s) = -\sum_{k=1}^{m} p_k \log_2 p_k$$

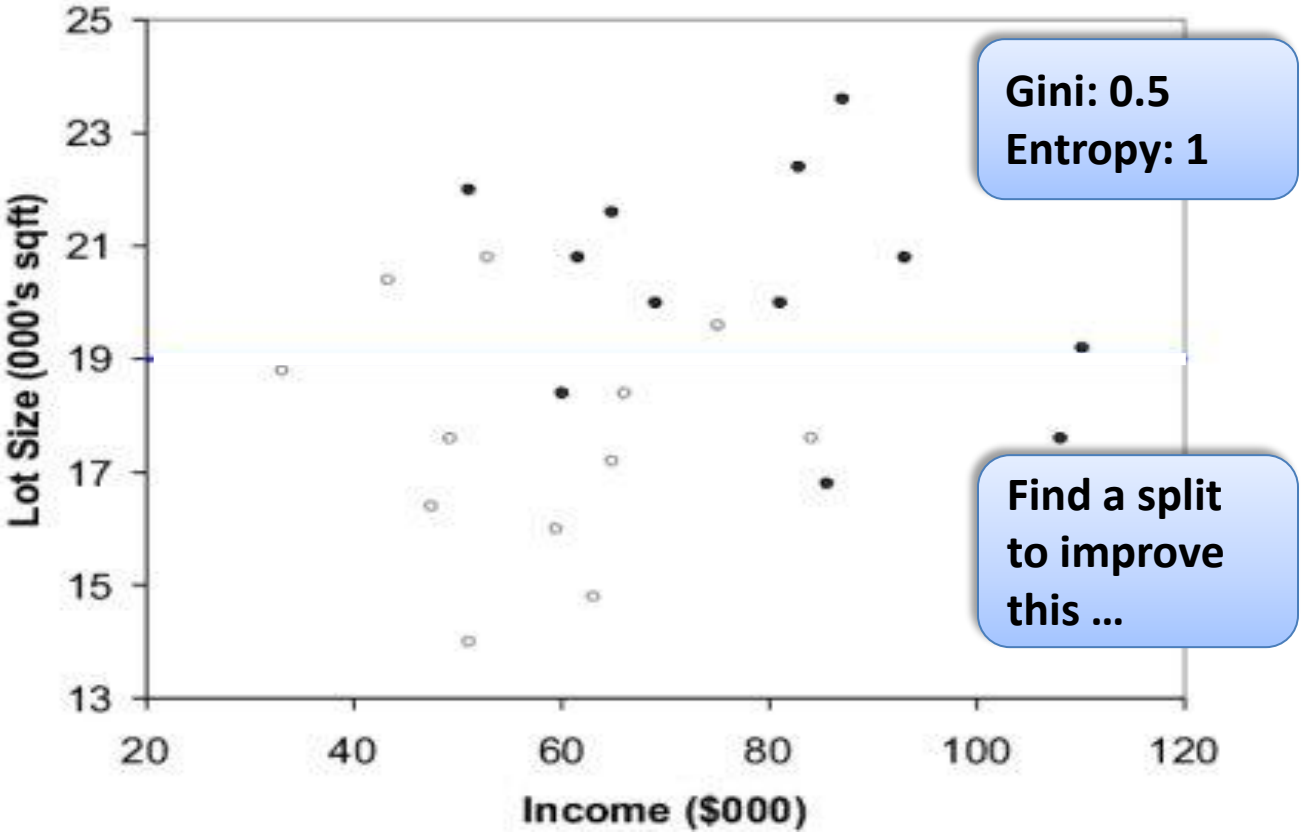
p_1

0.5: 12 Owners out of 24

p_2

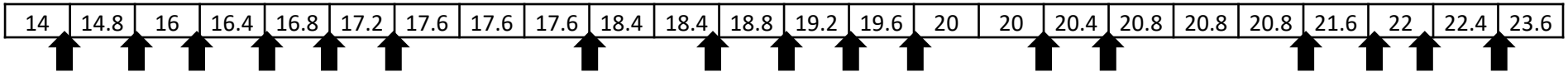
0.5: 12 Non-owners out of 24

$$-[0.5*(-1)+0.5*(-1)]=1$$

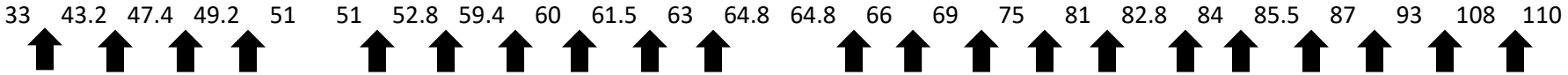

25
23
21
19
17
15
13
Lot Size (000's sqft)
20
40
60
80
100
120
Income ($000)
Gini: 0.5
Entropy: 1
Find a split to improve this …

Finding the First Split

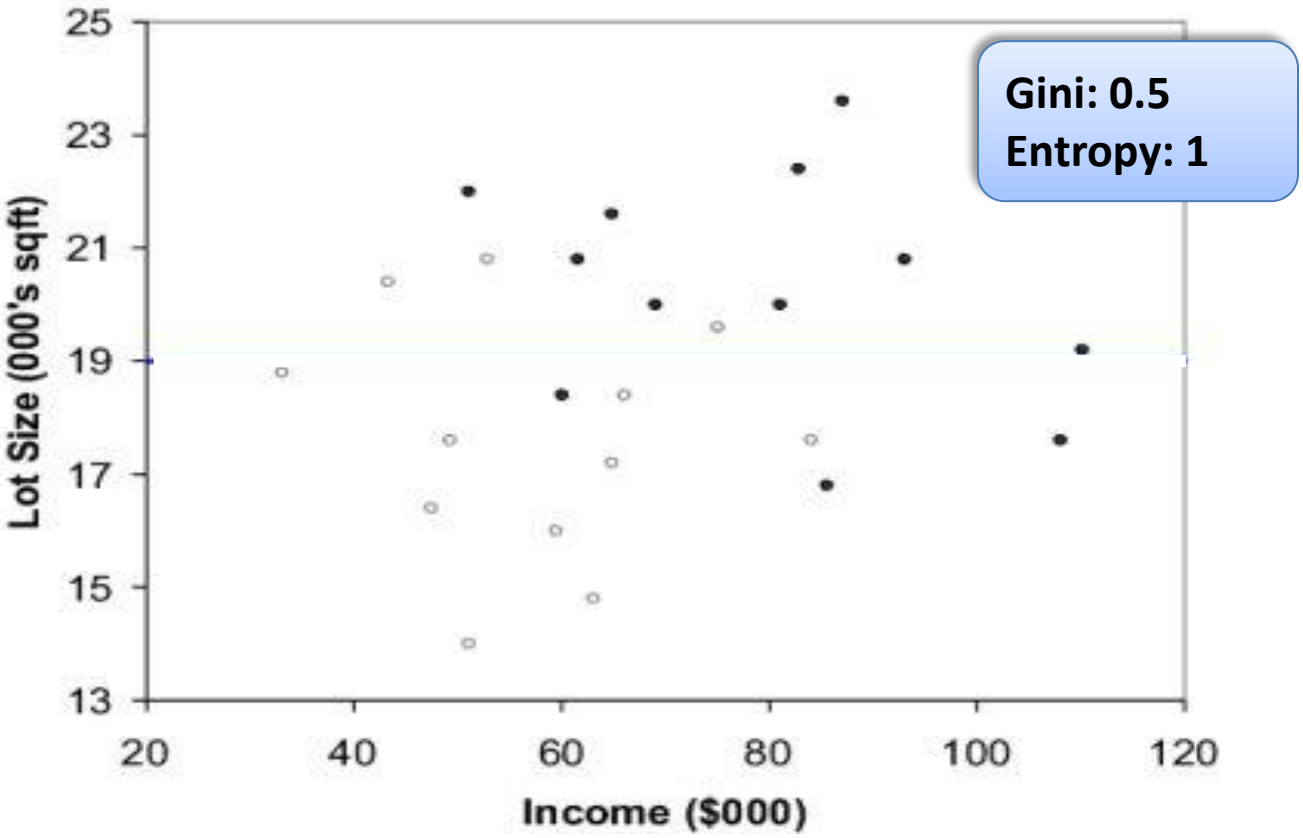
Lot Sizes -- sorted **Can split between any two successive values that are different**



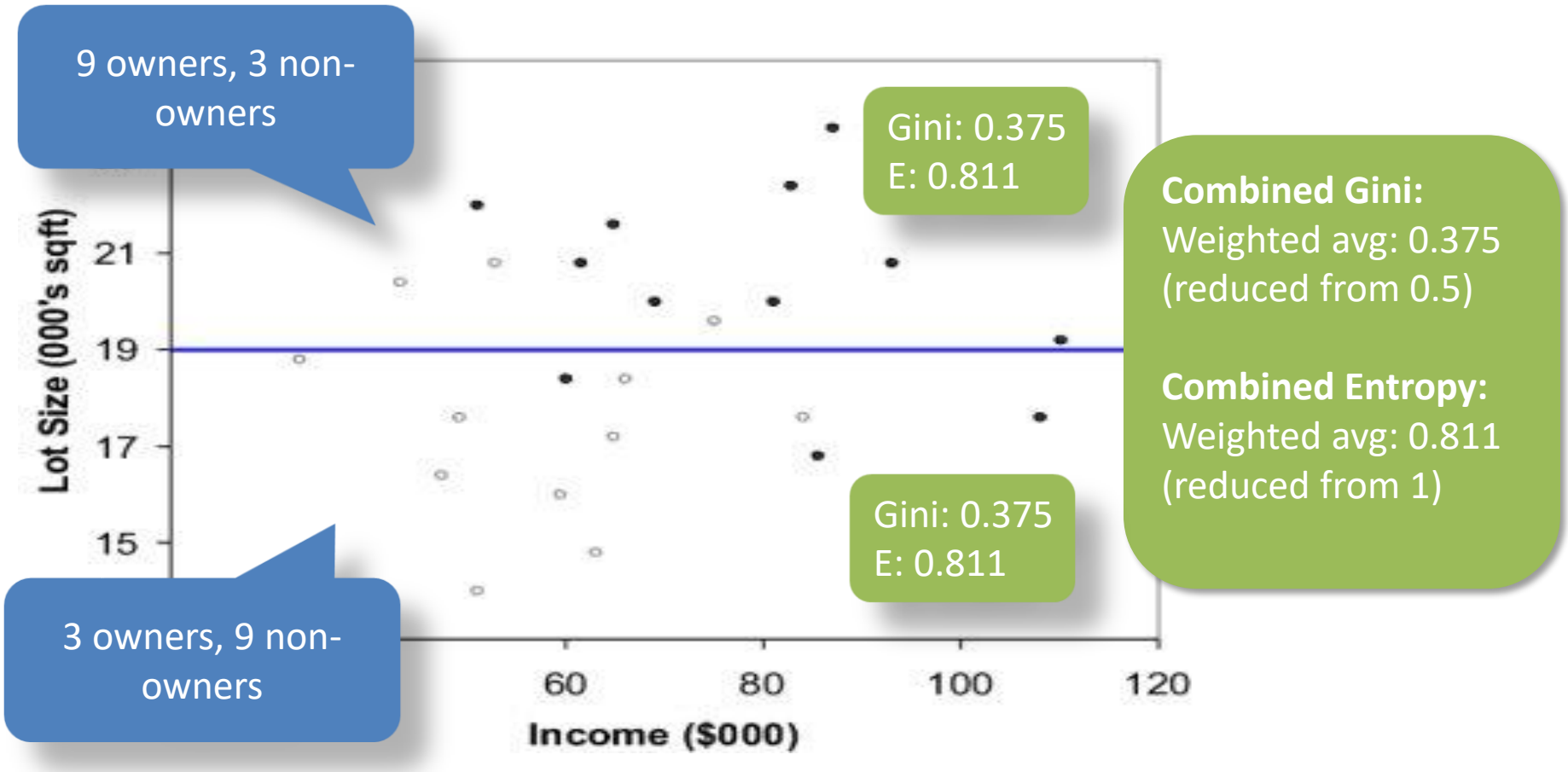
Incomes -- sorted



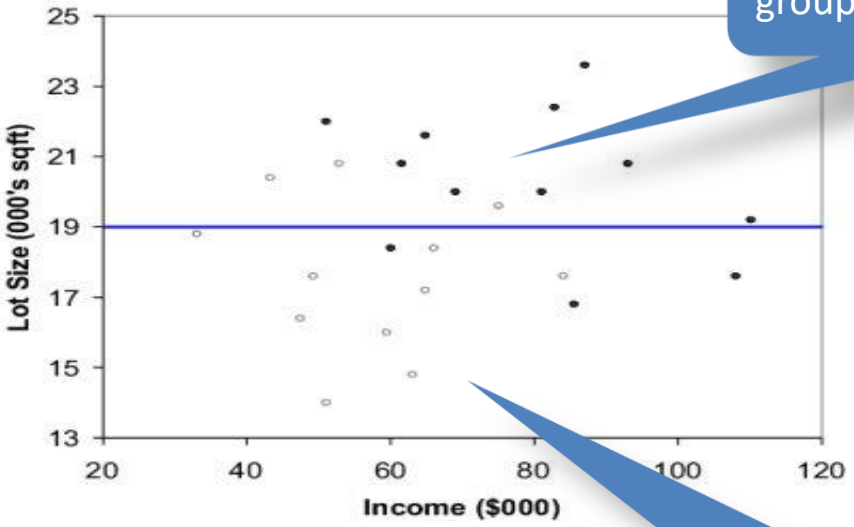
38 -- Try all and pick the one that reduces overall impurity the most


Gini: 0.5
Entropy: 1
Lot Size (000's sqft)
25
23
21
19
17
15
13
20
40
60
80
100
120
Income ($000)

First Split – Lot Size 19



Choosing Subsequent Splits

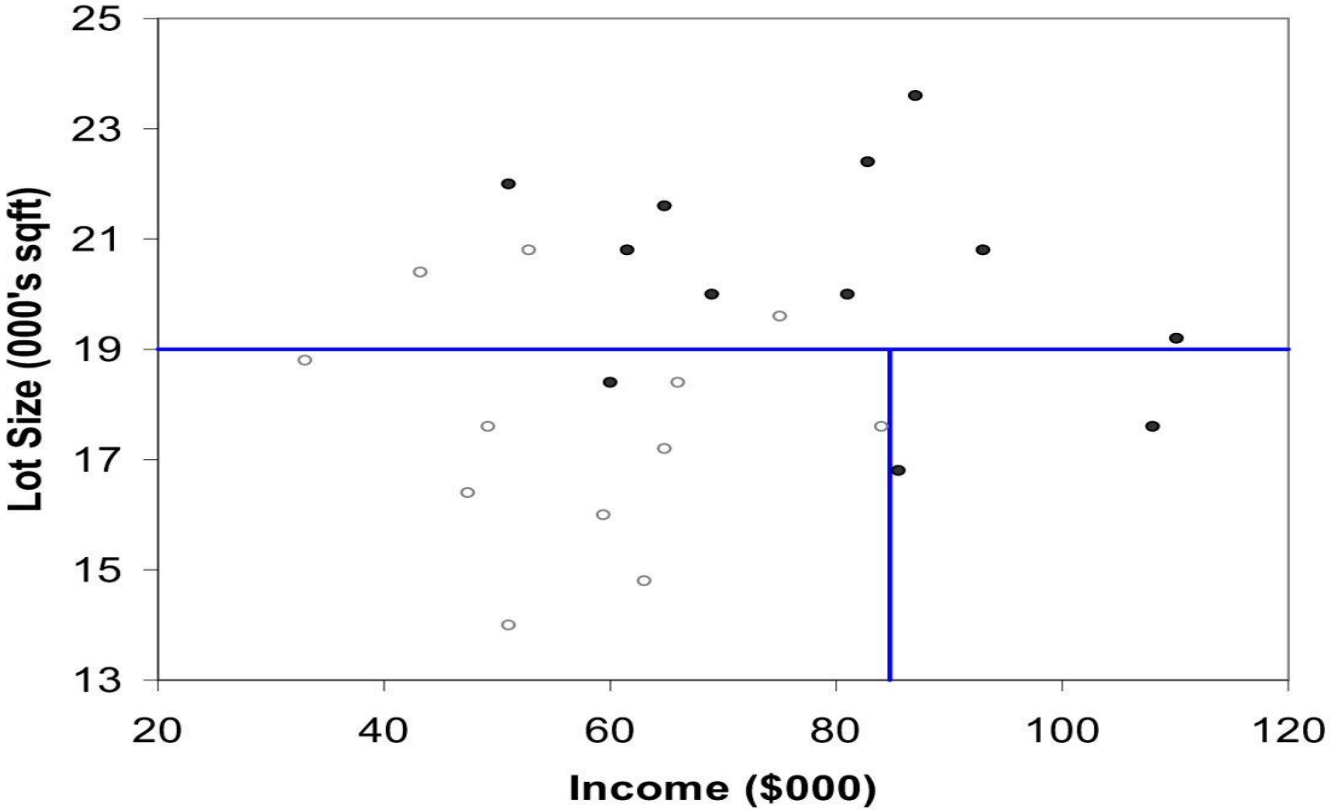


Several splits possible on this group (income and lot size)

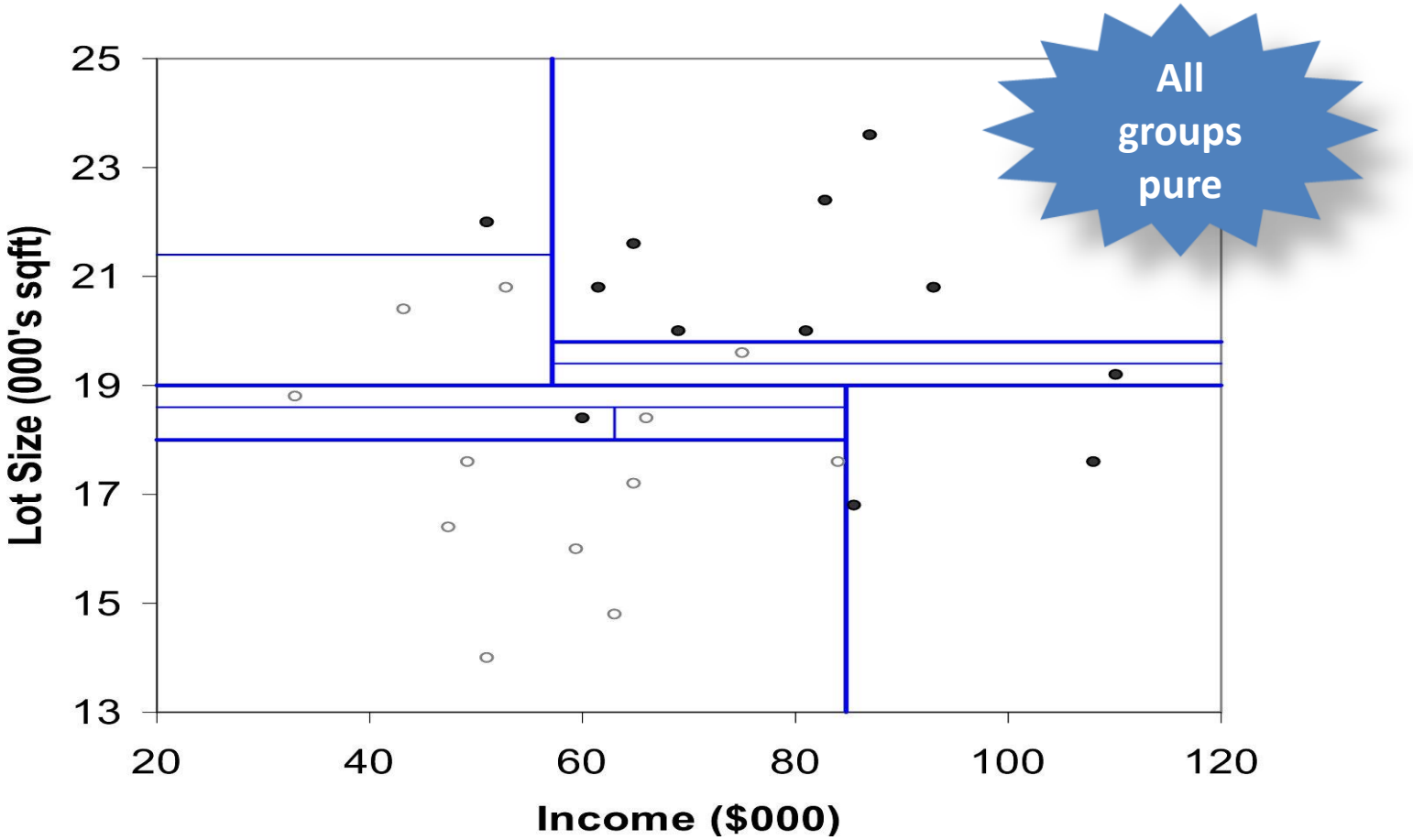
Consider all possibilities and calculate the overall Gini/Entropy under each. Choose the best.

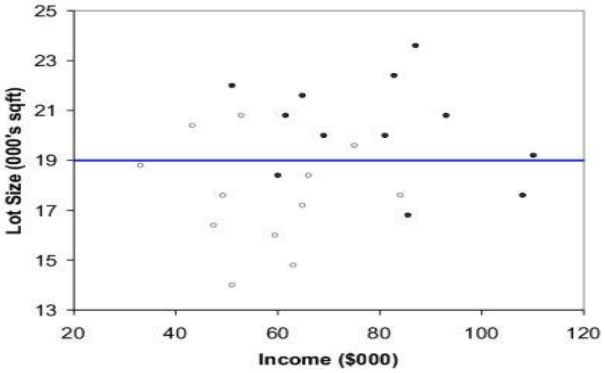
Several splits possible on this group (income and lot size)

Second Split – Income $84K



After All Splits





Tree Representation

Split value

No of cases <= 19

No of cases > 19

19

Lot_Size

12

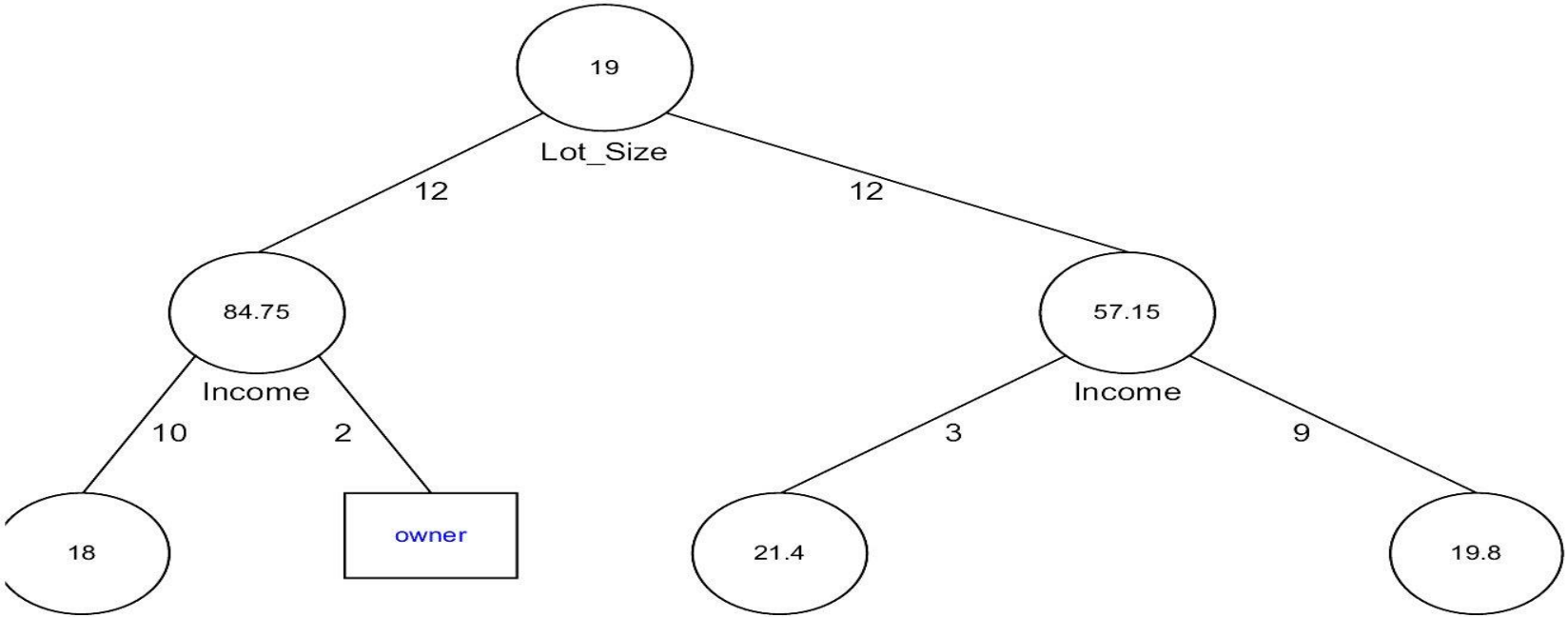
12

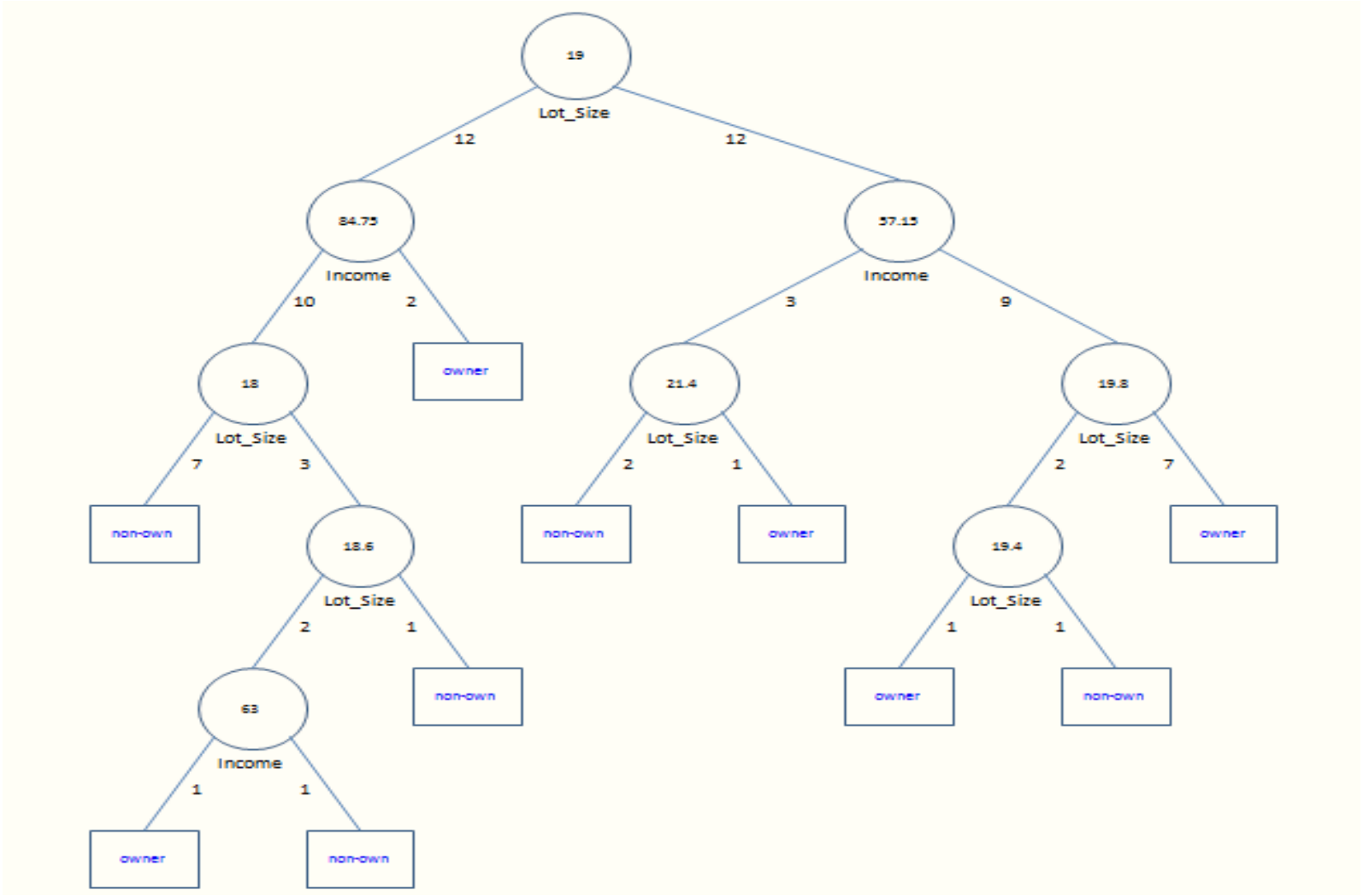
84.75

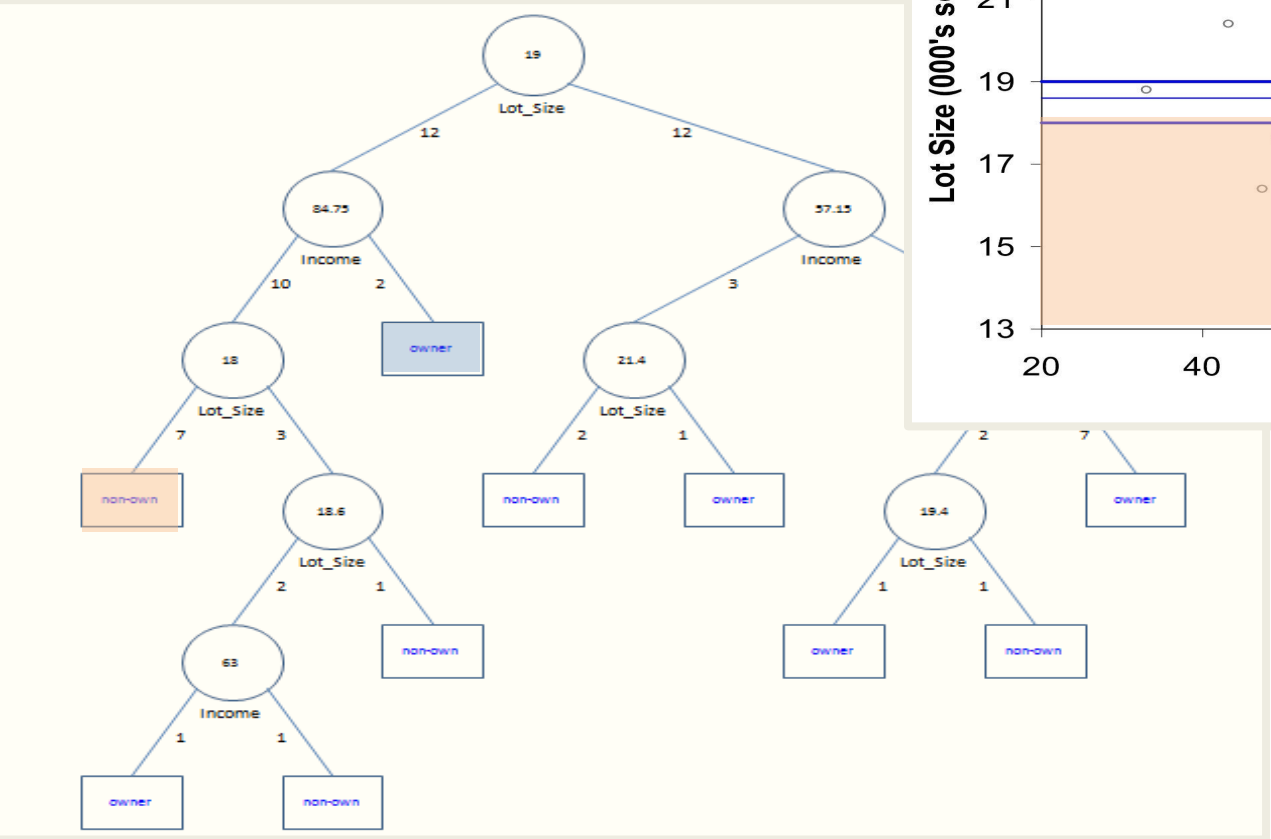
57.15

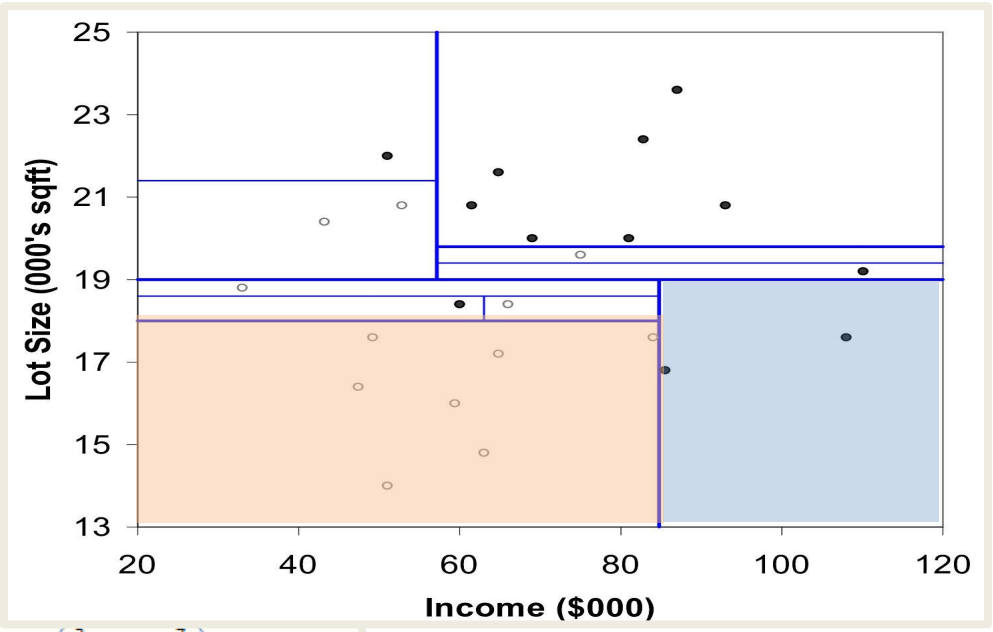
Variable being used to split

Tree -- After 3 Splits



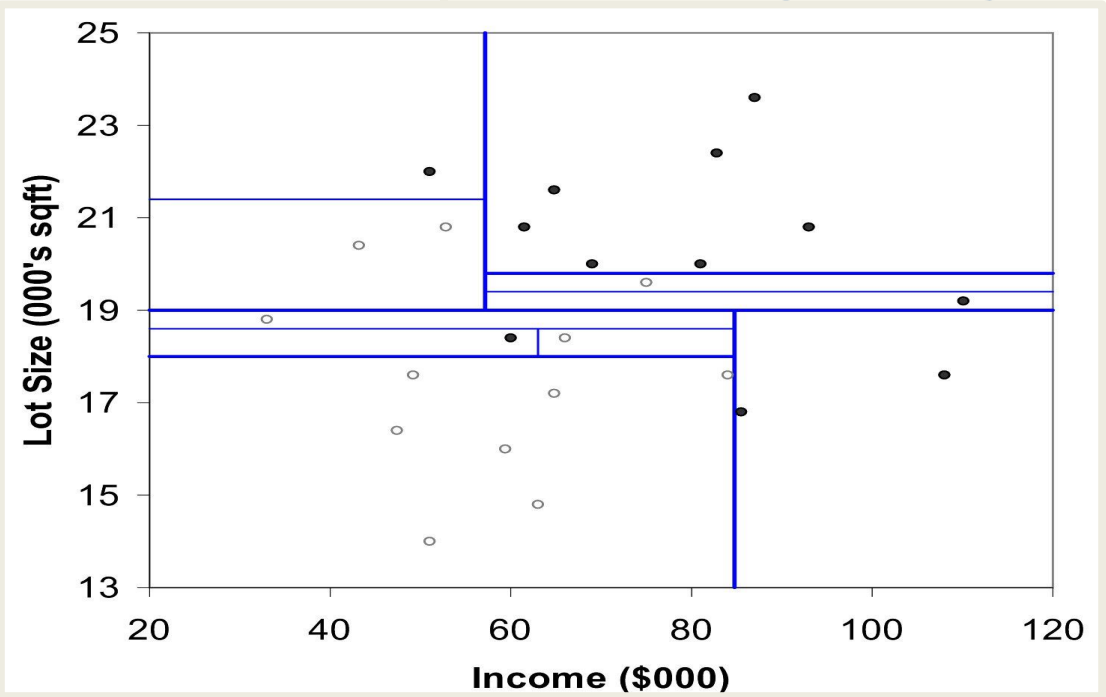

19
Lot_Size
12
12
84.75
Income
57.15
Income
10
2
3
9
18
owner
21.4
19.8
Lot_Size
Lot_Size
Lot_Size
7
3
2
1
2
7
non-own
18.6
non-own
owner
19.4
owner
Lot_Size
Lot_Size
2
1
1
1
63
non-own
owner
non-own
Income
1
1
owner
non-own

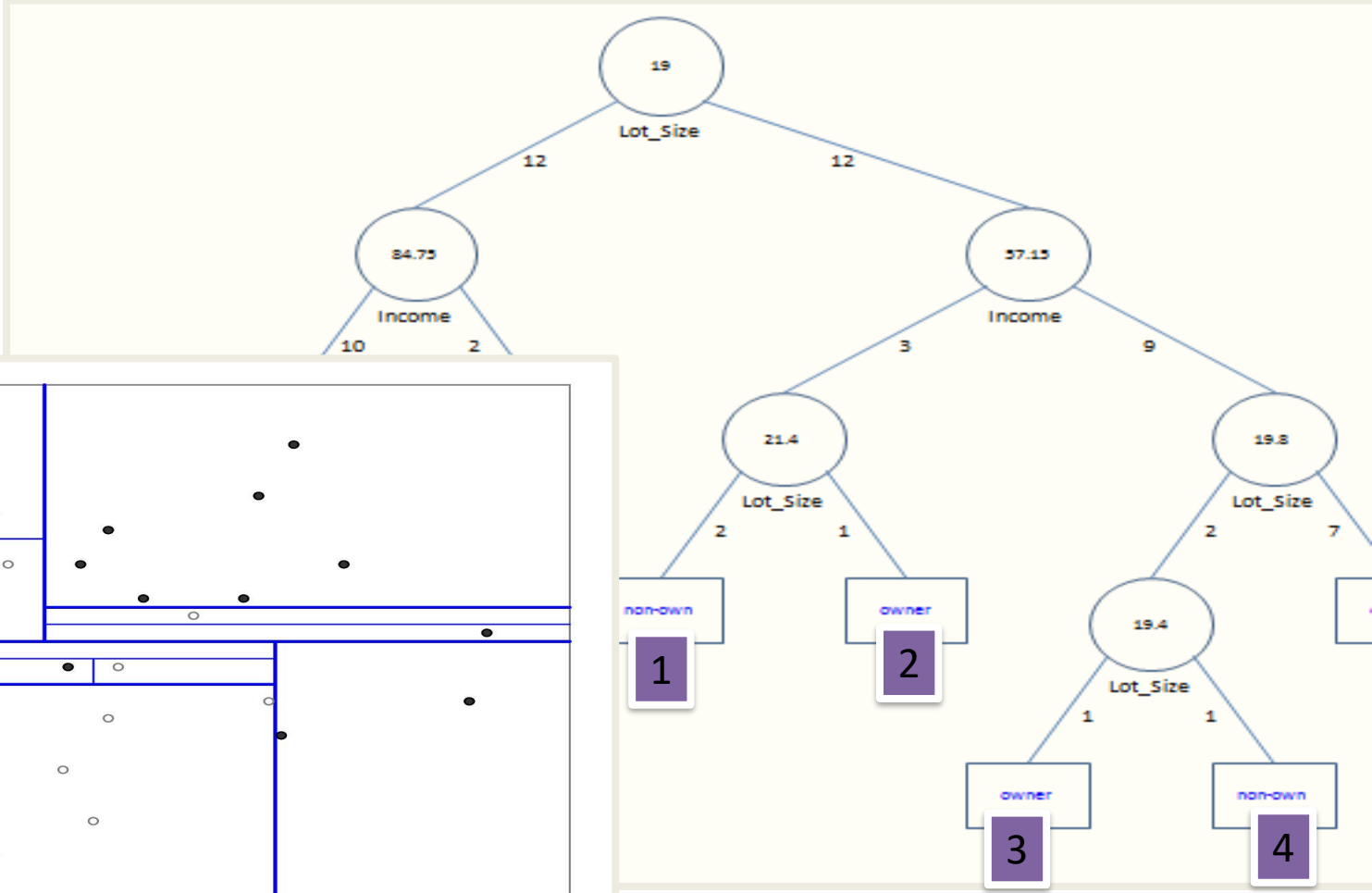




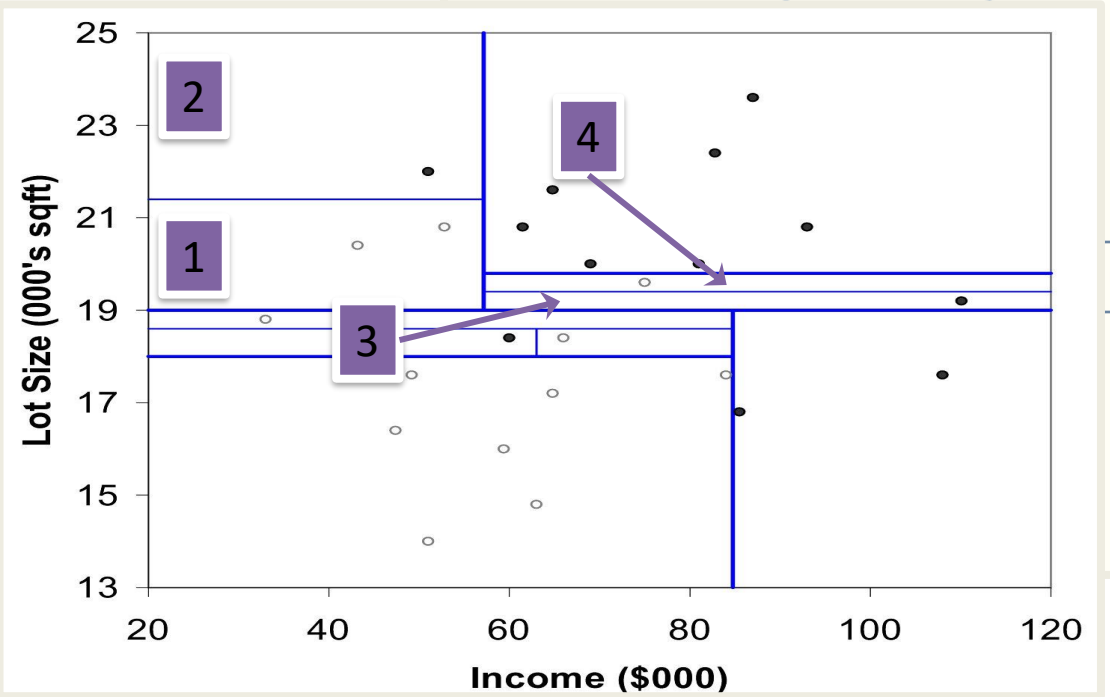
After All Splits

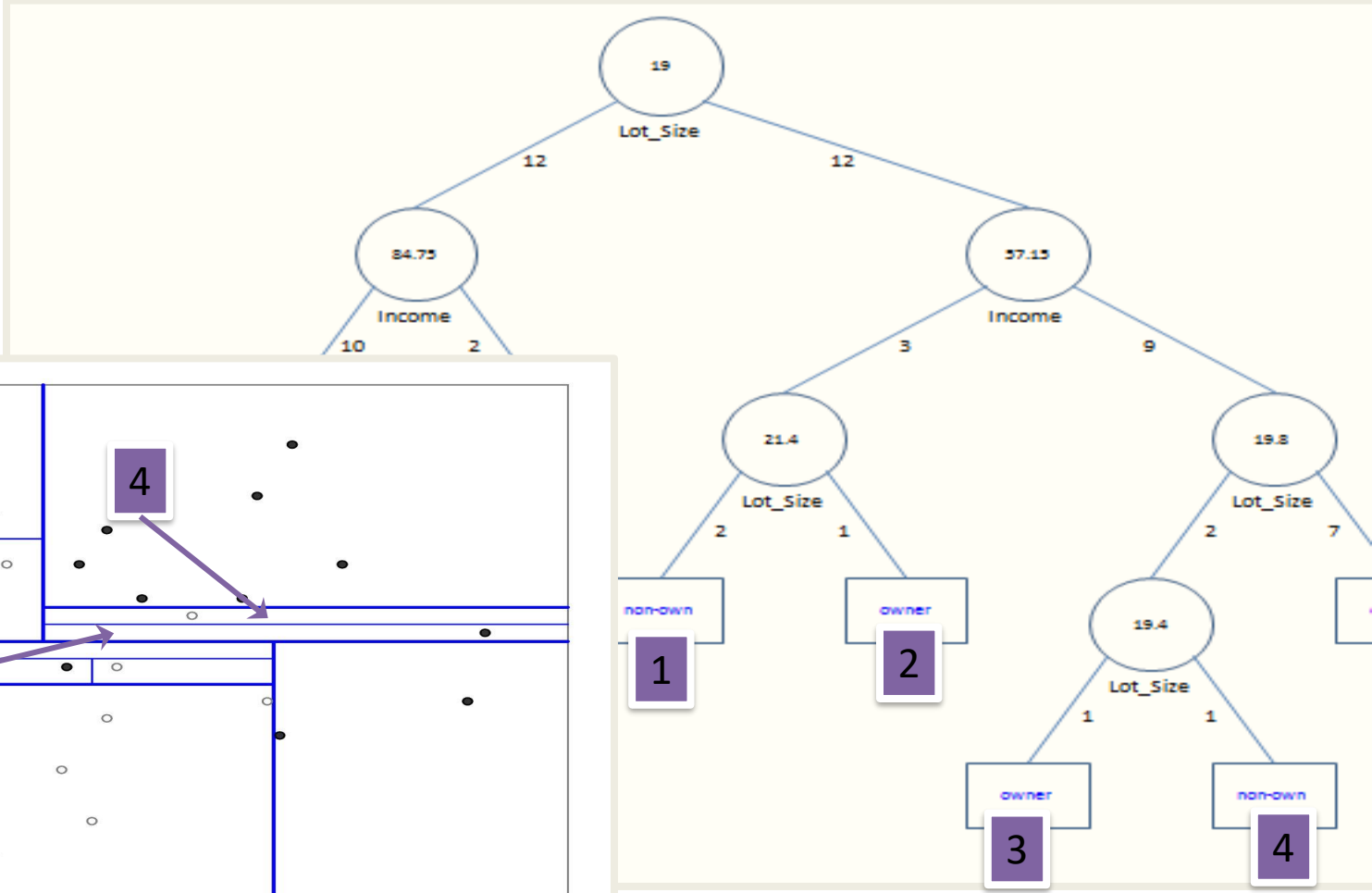
Your Turn


Lot Size (000's sqft)
25
23
21
19
17
15
13
20
40
60
80
100
120
Income ($000)


19
Lot_Size
12
12
84.75
Income
10
2
57.15
Income
3
9
21.4
Lot_Size
2
1
19.8
Lot_Size
2
7
non-own
1
owner
2
19.4
Lot_Size
1
1
owner
3
non-own
4

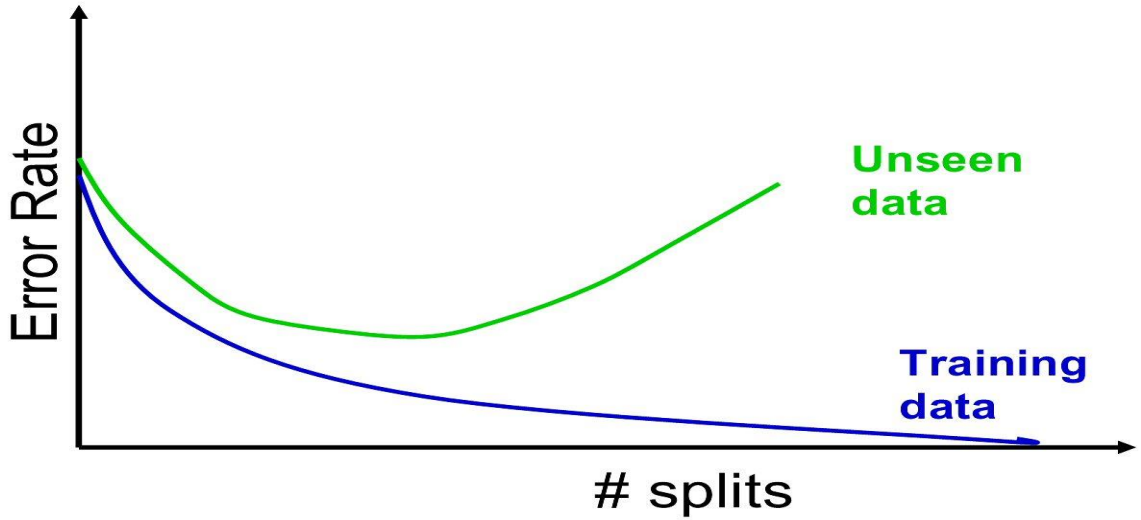
Answer


25
23
21
19
17
15
13
Lot Size (000's sqft)
20
40
60
80
100
120
Income ($000)
1
2
3
4


19
Lot_Size
12
12
84.75
57.15
Income
Income
10
2
3
9
21.4
19.8
Lot_Size
Lot_Size
2
1
2
7
non-own
owner
19.4
Lot_Size
1
1
owner
non-own
1
2
3
4

Beyond a point, fitting noise -- Over fitting
Reduces performance on validation data

Impact of Number of Splits



Iterative Dichotomizer 3 (ID3) algorithm steps (uses entropy)

1. Calculate the initial *entropy* of the system based on the target variable.
2. Calculate the *information gains* for each candidate variable for a node. Select the variable that provides the maximum information gain as a decision node.
3. Repeat step 2 for each branch (value) of the node (variable) identified in step 2. The newly identified node is termed as *leaf* node.
4. Check whether the leaf node classifies the entire data perfectly. If not, repeat the steps from step 2 onwards. If yes, stop.

ID3 algorithm example

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Candidate Variable: *Outlook*

		play		
		yes	no	total
	sunny	3	2	5
Outlook	overcast	4	0	4
	rainy	2	3	5
				14

$$E(S) = -[(9/14)\log(9/14) + (5/14)\log(5/14)] = 0.94$$

$$E(S, \text{outlook}) = (5/14)*E(3,2) + (4/14)*E(4,0) + (5/14)*E(2,3) =$$

$$(5/14)(-(3/5)\log(3/5)-(2/5)\log(2/5)) + (4/14)(0) + (5/14)$$

$$((2/5)\log(2/5)-(3/5)\log(3/5)) = 0.693$$

Compare Info Gain for all variables

$$IG(S, outlook) = 0.94 - 0.693 = 0.247$$

$$IG(S, Temperature) = 0.940 - 0.911 = 0.029$$

$$IG(S, Humidity) = 0.940 - 0.788 = 0.152$$

$$IG(S, Windy) = 0.940 - 0.8932 = 0.048$$

Highest Information Gain is for *outlook*. *Outlook* is first decision node

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

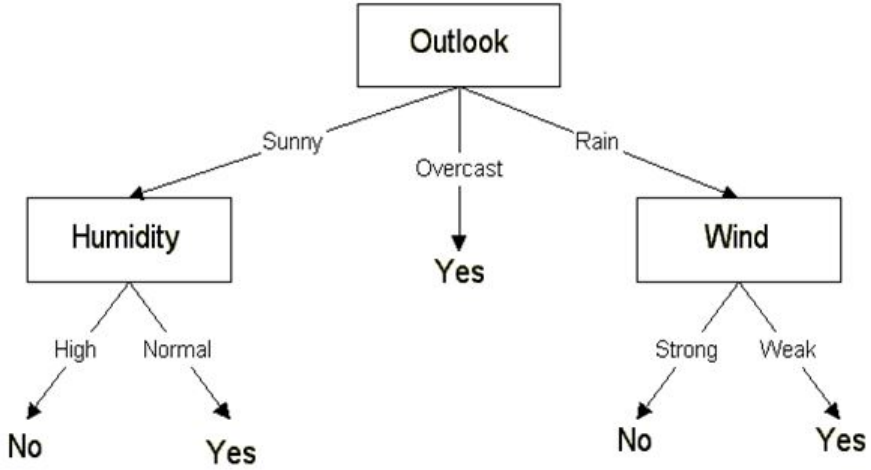
Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

		play		
		yes	no	total
	hot	0	2	2
Temperature	cool	1	1	2
	mild	1	0	1
				5

$$E(\text{sunny}) = (-(3/5)\log(3/5)-(2/5)\log(2/5)) = 0.971$$

$$E(\text{sunny, Temperature}) = (2/5)*E(0,2) + (2/5)*E(1,1) + (1/5)*E(1,0) = 2/5 = 0.4$$



$$IG(\text{sunny, Temperature}) = 0.971–0.4 = 0.571$$

$$IG(\text{sunny, Humidity}) = 0.971$$

$$IG(\text{sunny, Windy}) = 0.020$$

Humidity has the highest IG. Continue building out the decision tree

Gini Impurity

A measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

CART algorithm steps (uses Gini)

1. Calculate the *Gini* index based on the target variable.
2. Calculate the *Gini* gains for each candidate variable for a node. Select the variable that provides the maximum Gini gain as a decision node.
3. Repeat step 2 for each branch (value) of the node (variable) identified in step 2. The newly identified node is termed as leaf node.
4. Check whether the leaf node classifies the entire data perfectly. If not, repeat the steps from step 2 onwards. If yes, stop.

CART algorithm example

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

		play		
		yes	no	total
	sunny	3	2	5
Outlook	overcast	4	0	4
	rainy	2	3	5
				14

$$Gini(S) = 1 - [(9/14)^2 + (5/14)^2] = 0.4591$$

$$Gini(S, outlook) = (5/14)gini(3,2) + (4/14)*gini(4,0) + (5/14)*gini(2,3)$$
$$= (5/14)(1 - (3/5)^2 - (2/5)^2) + (4/14)*0 + (5/14)(1 - (2/5)^2 - (3/5)^2) =$$
$$0.171 + 0 + 0.171 = 0.342$$

$$\text{Gini gain (S, outlook)} = 0.459 - 0.342 = 0.117$$

$$\text{Gini gain(S, Temperature)} = 0.459 - 0.4405 = 0.0185$$

$$\text{Gini gain(S, Humidity)} = 0.459 - 0.3674 = 0.0916$$

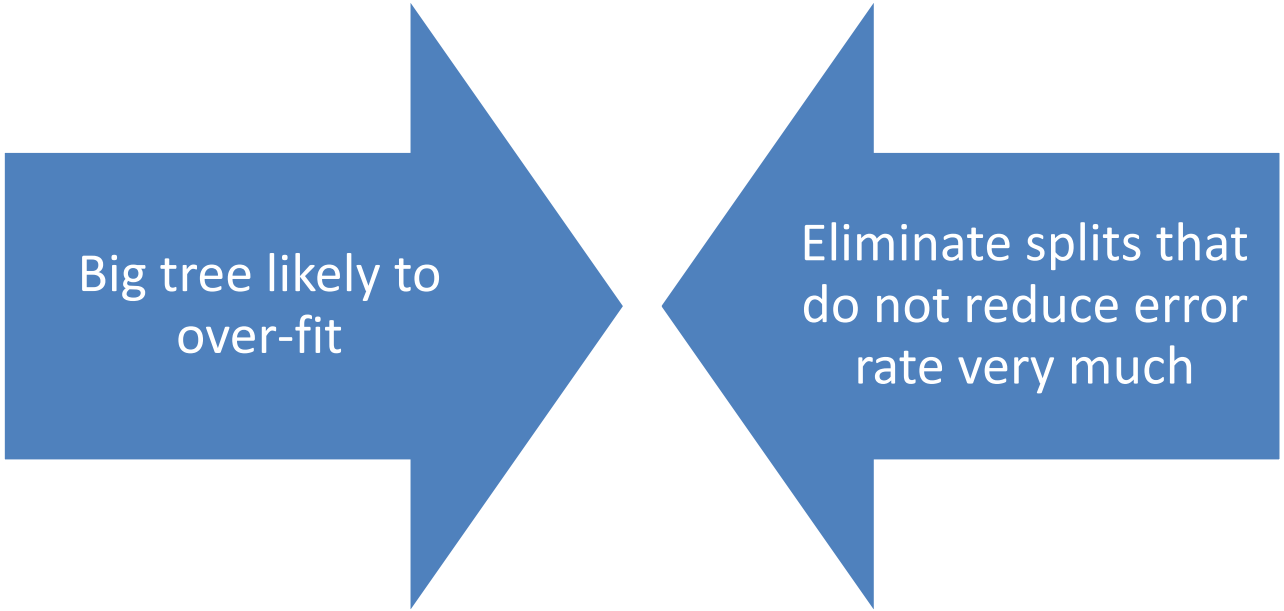
$$\text{Gini gain(S, windy)} = 0.459 - 0.4286 = 0.0304$$

Highest Gini Gain is for *outlook*. *Outlook* is first decision node

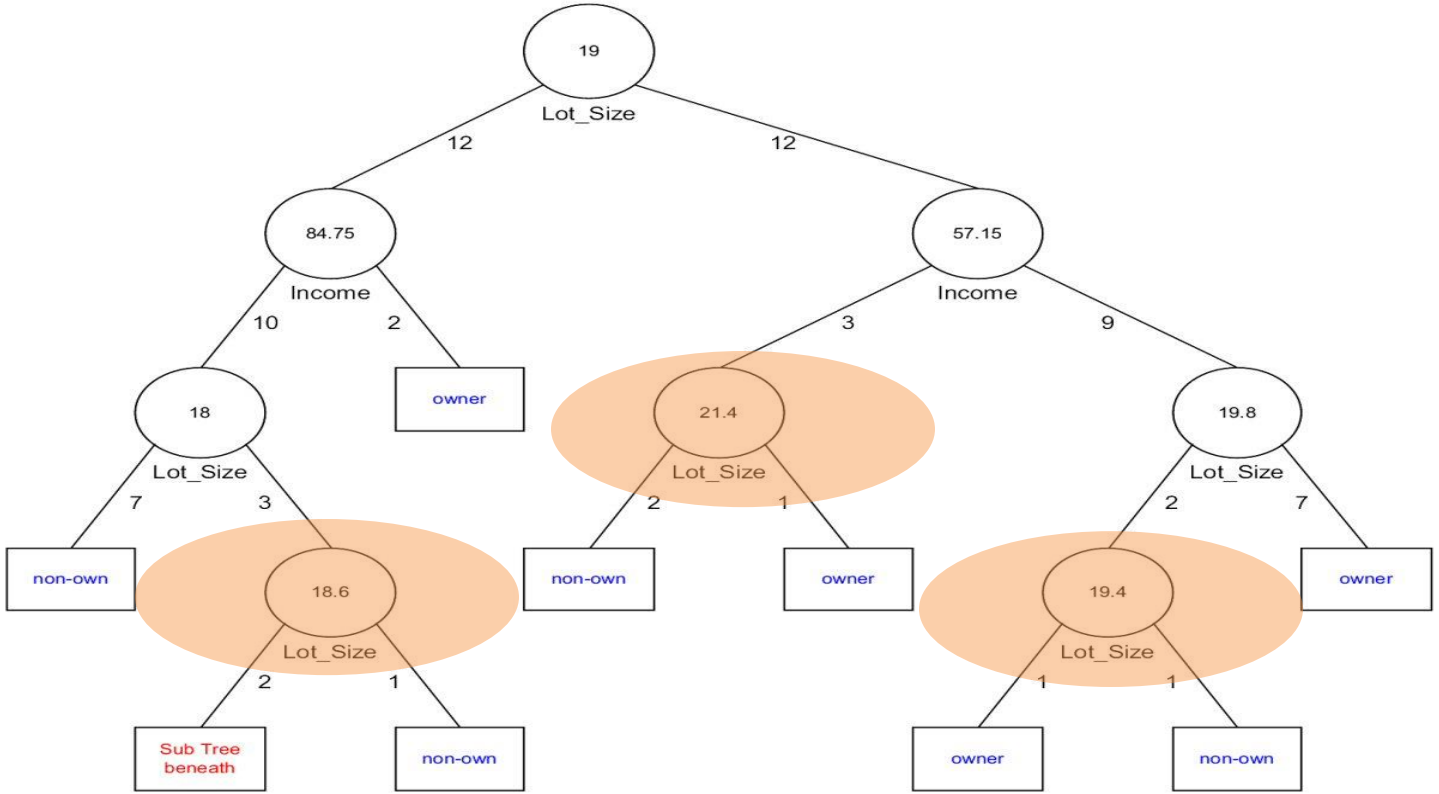
Handling numerical variables

1. Sort the dataset based on the numerical variable in ascending order.
2. Mark the numerical variable ranges where the target variable transitions from one category to another.
3. Create a separate categorical variable with values corresponding to the thresholds for the transitions from step 2.

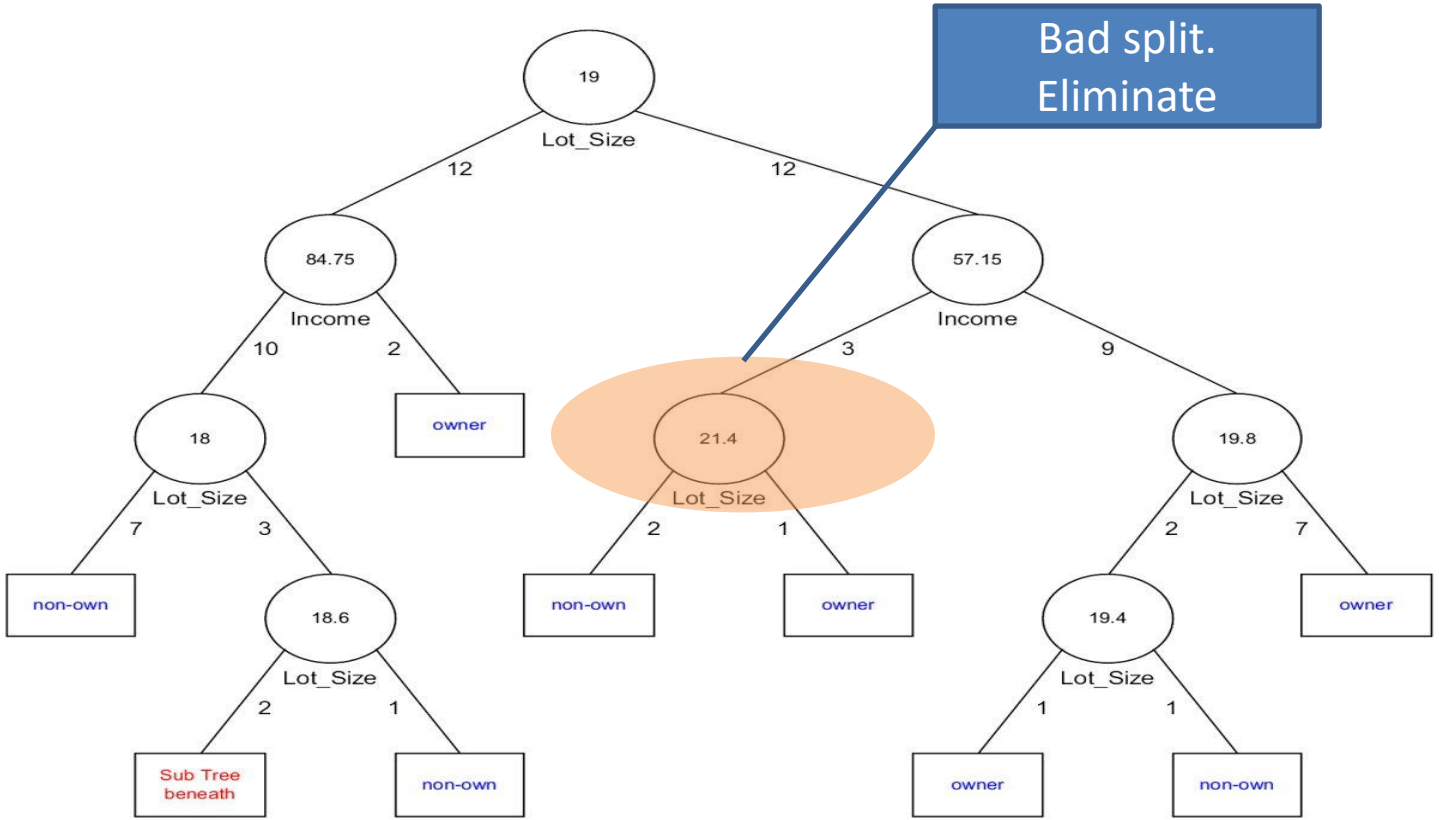
Pruning


Big tree likely to over-fit
Eliminate splits that do not reduce error rate very much

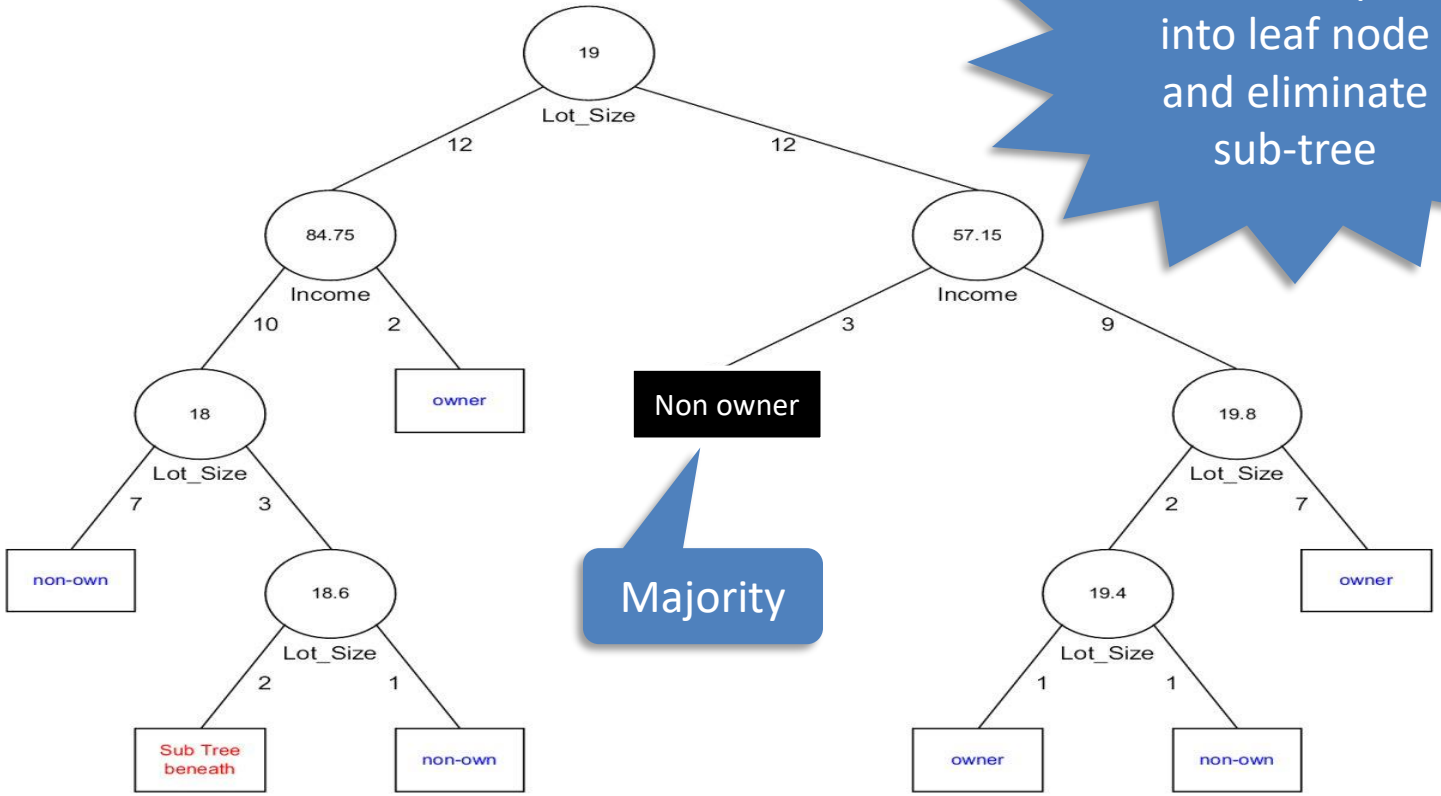
Splits Creating Small Groups



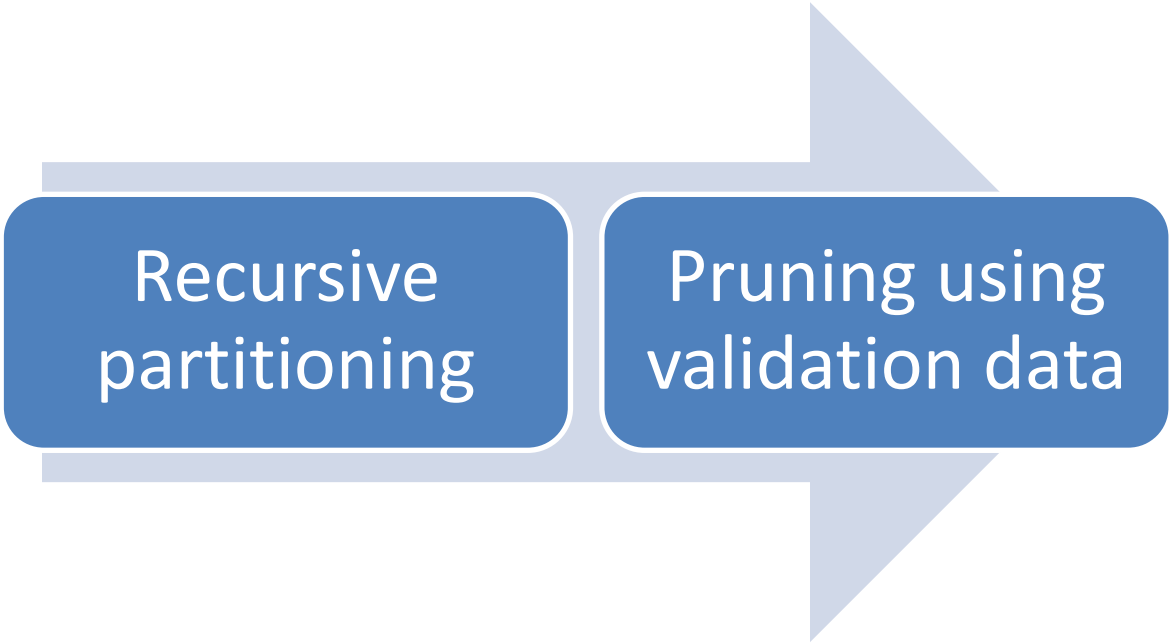
Pruning Example



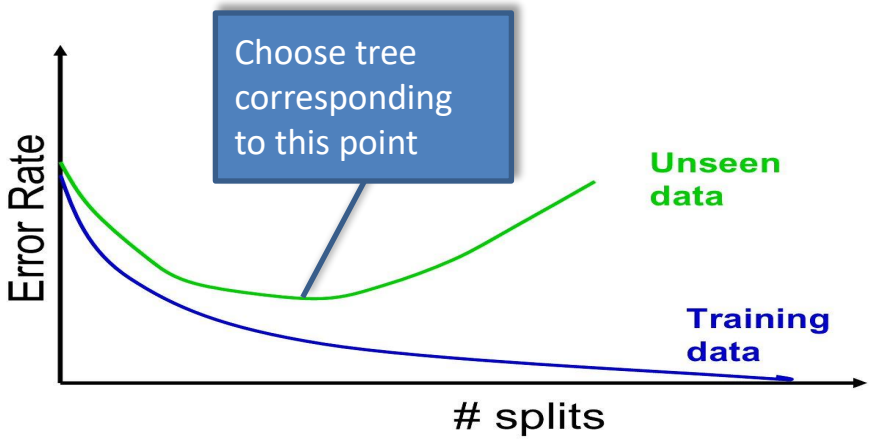
Pruning Example



Convert split into leaf node and eliminate sub-tree


Recursive partitioning
Pruning using validation data

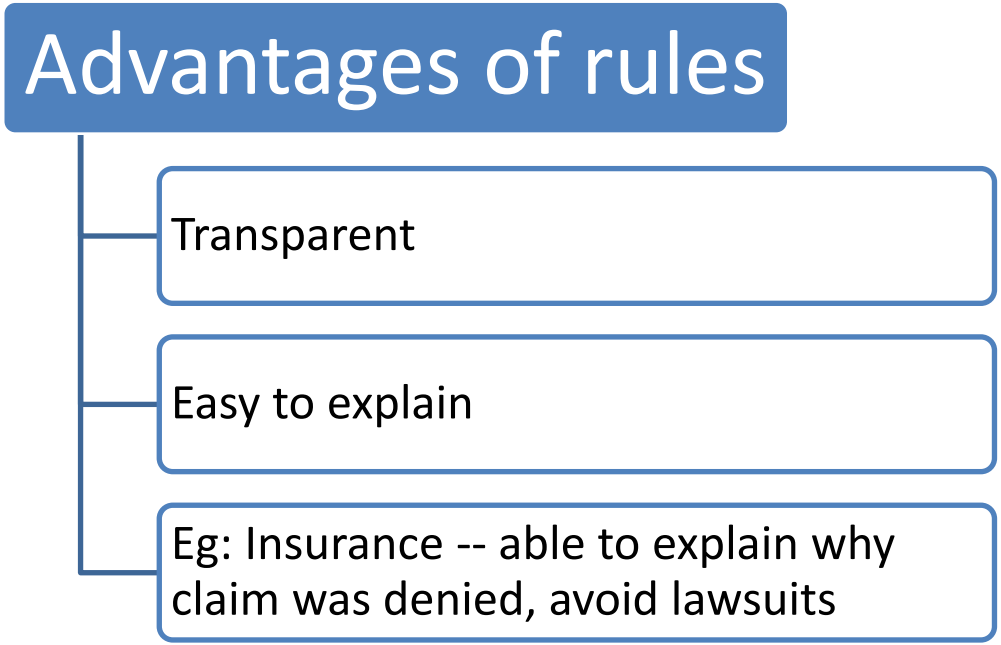
Pruning Approach

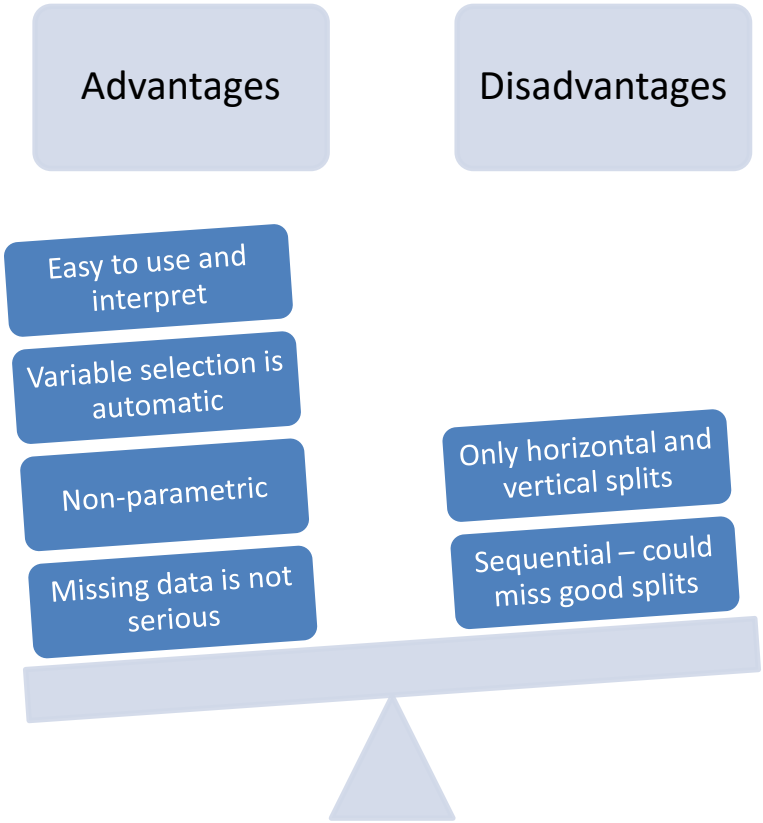


Generate **best** trees of each size

Select tree with minimum **error rate on test data**

Tree Implies Rules




Advantages
Disadvantages
Easy to use and interpret
Variable selection is automatic
Non-parametric
Missing data is not serious
Only horizontal and vertical splits
Sequential – could miss good splits

Demo IRIS

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

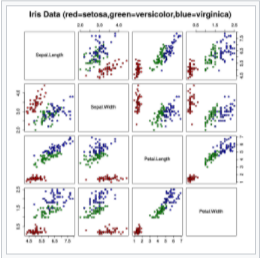
Languages
العربية
Español
فارسی
Français
Italiano
Русский

Iris flower data set

From Wikipedia, the free encyclopedia

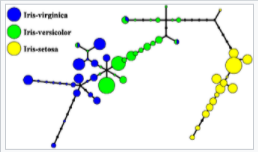
The ***Iris* flower data set** or **Fisher's *Iris* data set** is a multivariate data set introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis.[1] It is sometimes called **Anderson's *Iris* data set** because Edgar Anderson collected the data to quantify the morphologic variation of *Iris* flowers of three related species.[2] Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".[3] Fisher's paper was published in the journal, the Annals of Eugenics, creating controversy about the continued use of the Iris dataset for teaching statistical techniques today.

The data set consists of 50 samples from each of three species of *Iris* (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.



Scatterplot of the data set

Use of the data set [edit]



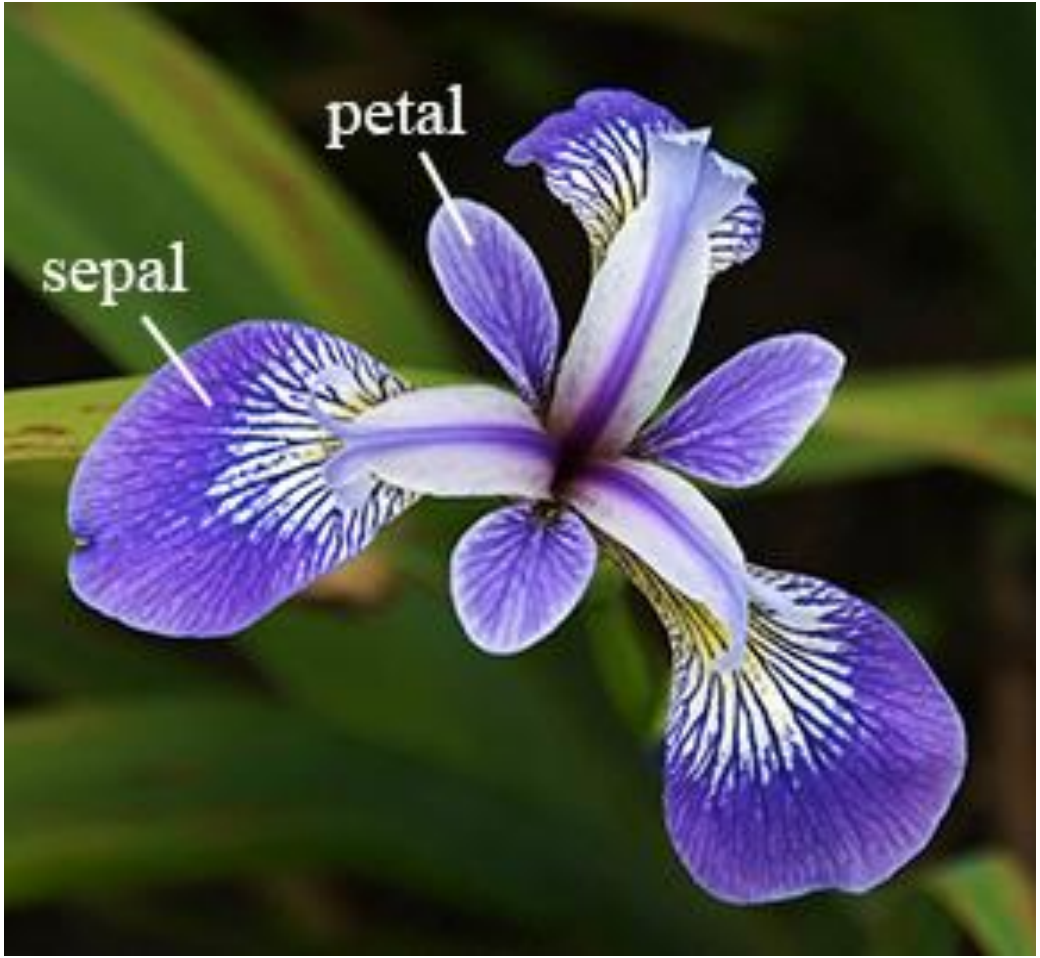
An example of the so-called "metro map" for the *Iris* data set[4] Only a small fraction of *Iris-virginica* is mixed with *Iris-versicolor*. All other samples of the different *Iris* species belong to the different nodes.

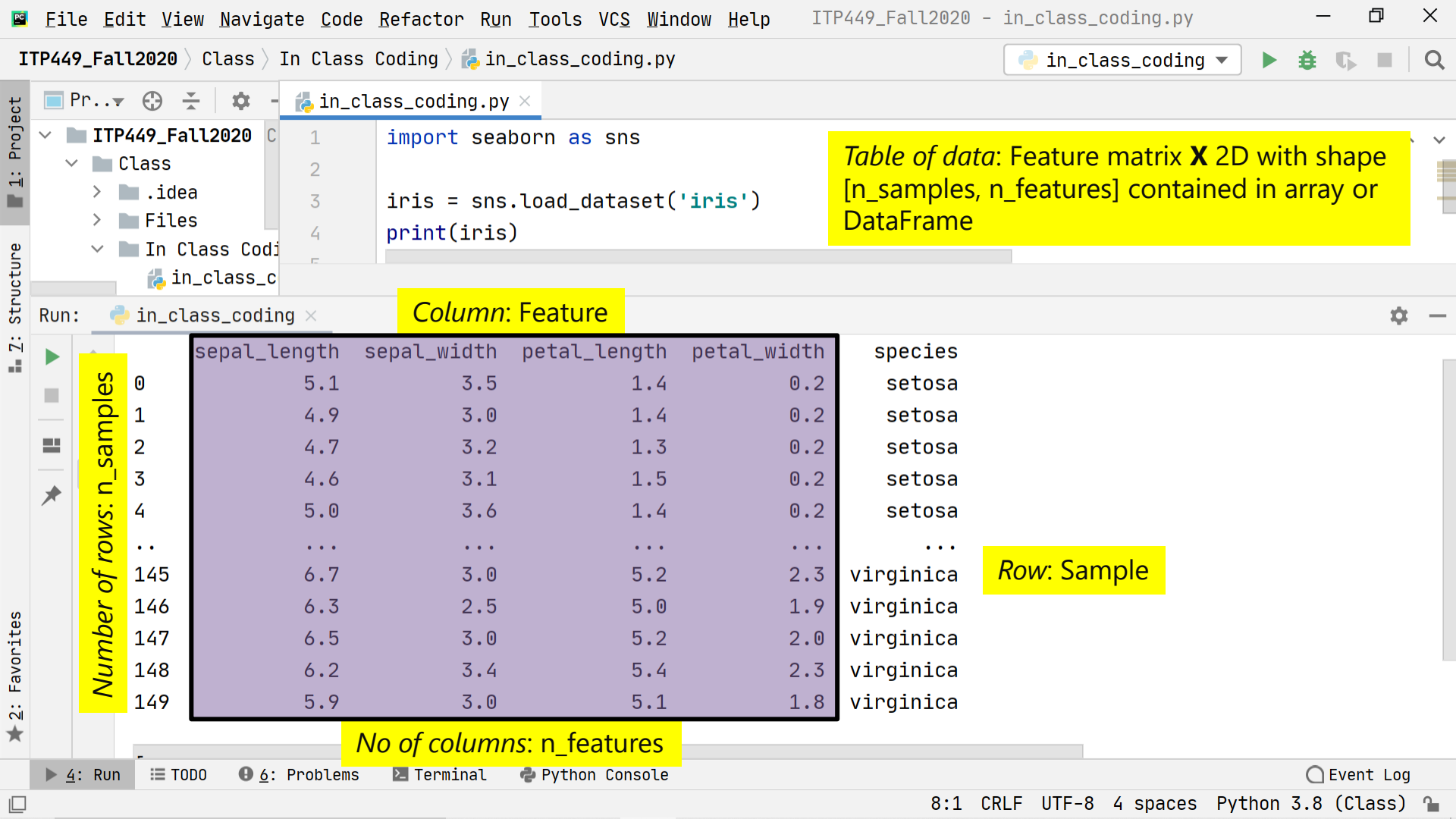
Based on Fisher's linear discriminant model, this data set became a typical test case for many statistical classification techniques in machine learning such as support vector machines.[5]

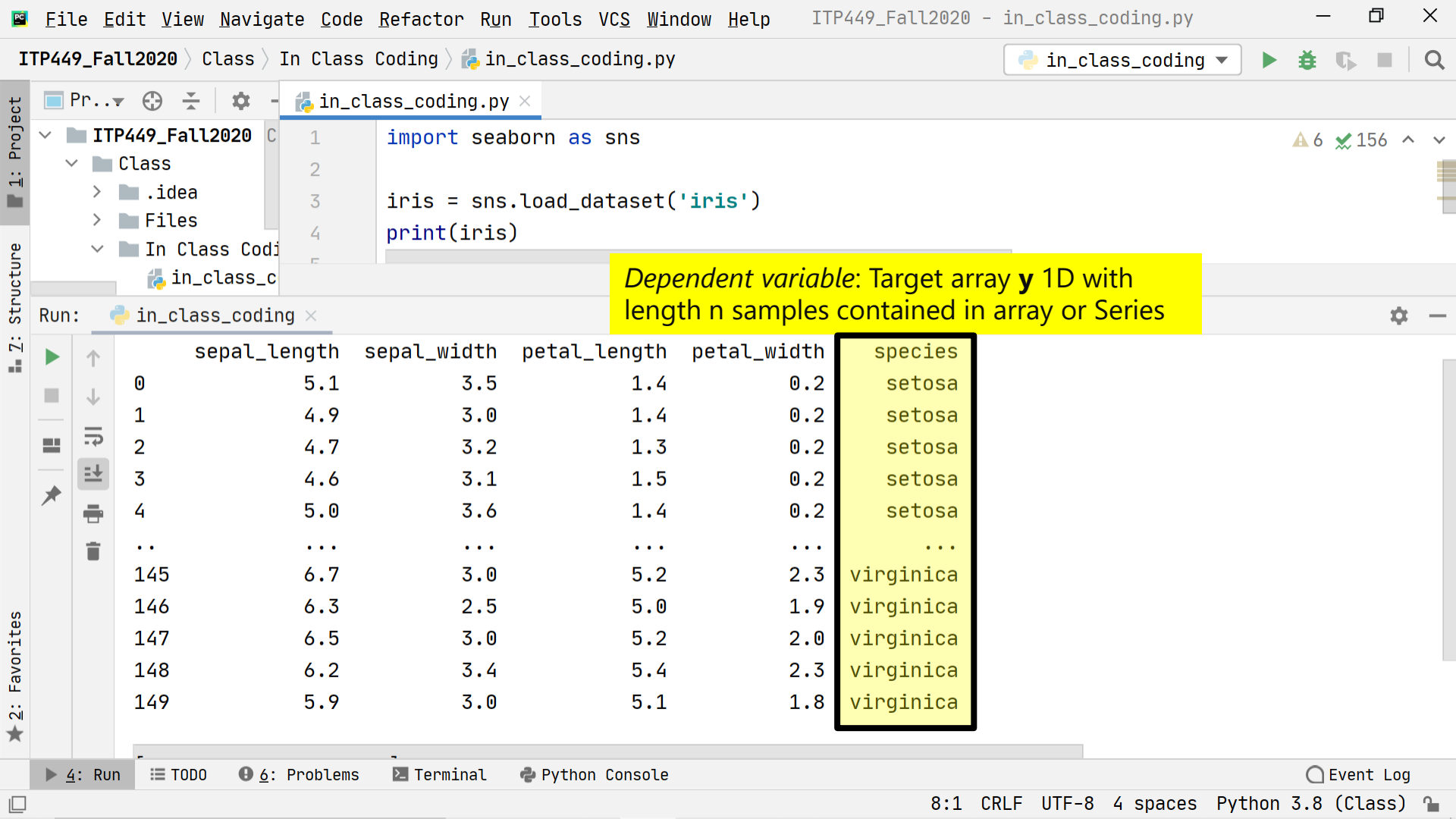
The use of this data set in cluster analysis however is not common, since the data set only contains two clusters with rather obvious separation. One of the clusters contains *Iris setosa*, while the other cluster contains both *Iris virginica* and *Iris versicolor* and is not separable without the species information Fisher used. This makes the data set a good example to explain the difference between supervised and unsupervised techniques in data mining: Fisher's linear discriminant model can only be obtained when the object species are known: class labels and clusters are not necessarily the same.[6]

Unsatisfactory k-means clustering (the data cannot be clustered into the known classes) and actual species visualized using ELKI

Nevertheless, all three species of *Iris* are separable in the projection on the nonlinear and branching principal component.[7] The data set is approximated by the closest tree with some penalty for the excessive number of nodes, bending and stretching. Then the so-called "metro map" is constructed.[4] The data points are projected into the closest node. For each node the pie diagram of the projected points is prepared. The area of the pie is proportional to the number of the projected points. It is clear from the diagram (left) that the absolute majority of the samples of the different *Iris* species belong to the different nodes. Only a small fraction of *Iris-virginica* is mixed with *Iris-versicolor* (the mixed blue-green nodes in the diagram). Therefore, the three species of Iris (*Iris setosa*, *Iris virginica* and *Iris versicolor*) are separable by the


petal
sepal


File Edit View Navigate Code Refactor Run Tools VCS Window Help
ITP449_Fall2020 - in_class_coding.py
ITP449_Fall2020 〉 Class 〉 In Class Coding 〉 in_class_coding.py
in_class_coding
in_class_coding.py
1 import seaborn as sns
2
3 iris = sns.load_dataset('iris')
4 print(iris)
Table of data: Feature matrix X 2D with shape [n_samples, n_features] contained in array or DataFrame
Run: in_class_coding
Column: Feature
sepal_length sepal_width petal_length petal_width species
0 5.1 3.5 1.4 0.2 setosa
1 4.9 3.0 1.4 0.2 setosa
2 4.7 3.2 1.3 0.2 setosa
3 4.6 3.1 1.5 0.2 setosa
4 5.0 3.6 1.4 0.2 setosa
.. ..
145 6.7 3.0 5.2 2.3 virginica
146 6.3 2.5 5.0 1.9 virginica
147 6.5 3.0 5.2 2.0 virginica
148 6.2 3.4 5.4 2.3 virginica
149 5.9 3.0 5.1 1.8 virginica
Number of rows: n_samples
Row: Sample
No of columns: n_features
4: Run
TODO
6: Problems
Terminal
Python Console
Event Log
8:1 CRLF UTF-8 4 spaces Python 3.8 (Class)


File Edit View Navigate Code Refactor Run Tools VCS Window Help
ITP449_Fall2020 - in_class_coding.py
ITP449_Fall2020 〉 Class 〉 In Class Coding 〉 in_class_coding.py
in_class_coding
in_class_coding.py
ITP449_Fall2020
Class
.idea
Files
In Class Codi
in_class_c
1 import seaborn as sns
2
3 iris = sns.load_dataset('iris')
4 print(iris)
6 156
Dependent variable: Target array y 1D with length n samples contained in array or Series
Run: in_class_coding
sepal_length sepal_width petal_length petal_width species
0 5.1 3.5 1.4 0.2 setosa
1 4.9 3.0 1.4 0.2 setosa
2 4.7 3.2 1.3 0.2 setosa
3 4.6 3.1 1.5 0.2 setosa
4 5.0 3.6 1.4 0.2 setosa
.. ..
145 6.7 3.0 5.2 2.3 virginica
146 6.3 2.5 5.0 1.9 virginica
147 6.5 3.0 5.2 2.0 virginica
148 6.2 3.4 5.4 2.3 virginica
149 5.9 3.0 5.1 1.8 virginica
4: Run
TODO
6: Problems
Terminal
Python Console
Event Log
8:1 CRLF UTF-8 4 spaces Python 3.8 (Class)

Do the following:

Import iris dataset from iris.csv and store as a dataframe

Print the dataframe

in_class_coding.py

```python
import pandas as pd
import os

os.chdir('C:/Users/Reza/Desktop/ITP449_Fall2020/Class/Files')
pd.set_option('display.max_columns', None)

irisDf = pd.read_csv('iris.csv')

print(irisDf.head())
```

Run: in_class_coding

```
C:\Users\Reza\Desktop\ITP449_Fall2020\Class\venv\Scripts\python.exe "C:/Users/Reza/Desktop/ITP449_Fall2020/Class/In Class Coc
   Sepal length  Sepal width  Petal length  Petal width    Species
0           5.1          3.5           1.4          0.2  I. setosa
1           4.9          3.0           1.4          0.2  I. setosa
2           4.7          3.2           1.3          0.2  I. setosa
3           4.6          3.1           1.5          0.2  I. setosa
4           5.0          3.6           1.4          0.3  I. setosa

Process finished with exit code 0
```

Do the following:

Create feature matrix and response vector: X, y

in_class_coding.py

```
1  import pandas as pd
2  import os
3
4  os.chdir('C:/Users/Reza/Desktop/ITP449_Fall2020/Class/Files')
5  pd.set_option('display.max_columns', None)
6
7  irisDf = pd.read_csv('iris.csv')
8
9  X = irisDf[['Sepal length', 'Sepal width', 'Petal length', 'Petal width']]
10 y = irisDf['Species']
11
```

3 258

Run: in_class_coding

```
C:\Users\Reza\Desktop\ITP449_Fall2020\Class\venv\Scripts\python.exe "C:/Users/Reza/Desktop/ITP449_Fall2020/Class/In Class Coc

Process finished with exit code 0
```

Do the following:

Partition the X and y into train and test partitions - 70/30.

in_class_coding.py

```python
import pandas as pd
import os

os.chdir('C:/Users/Reza/Desktop/ITP449_Fall2020/Class/Files')
pd.set_option('display.max_columns', None)

irisDf = pd.read_csv('iris.csv')

X = irisDf[['Sepal length', 'Sepal width', 'Petal length', 'Petal width']]
y = irisDf['Species']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2020, stratify=y)
```

⚠ 7 ✓ 259

Run: in_class_coding

```
C:\Users\Reza\Desktop\ITP449_Fall2020\Class\venv\Scripts\python.exe "C:/Users/Reza/Desktop/ITP449_Fall2020/Class/In Class Cod
```

Do the following:

Train the decision tree classifier

in_class_coding.py

```
1  import pandas as pd
2  import os
3
4  os.chdir('C:/Users/Reza/Desktop/ITP449_Fall2020/Class/Files')
5  pd.set_option('display.max_columns', None)
6
7  irisDf = pd.read_csv('iris.csv')
8
9  X = irisDf[['Sepal length', 'Sepal width', 'Petal length', 'Petal width']]
10 y = irisDf['Species']
11
12 from sklearn.model_selection import train_test_split
13
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2020, stratify=y)
15
16 from sklearn.tree import DecisionTreeClassifier
17
18 dt = DecisionTreeClassifier(criterion='entropy')
19 dt.fit(X_train, y_train)
20
```

⚠ 5 ✓ 258

criterion: *gini* or *entropy*

Do the following:

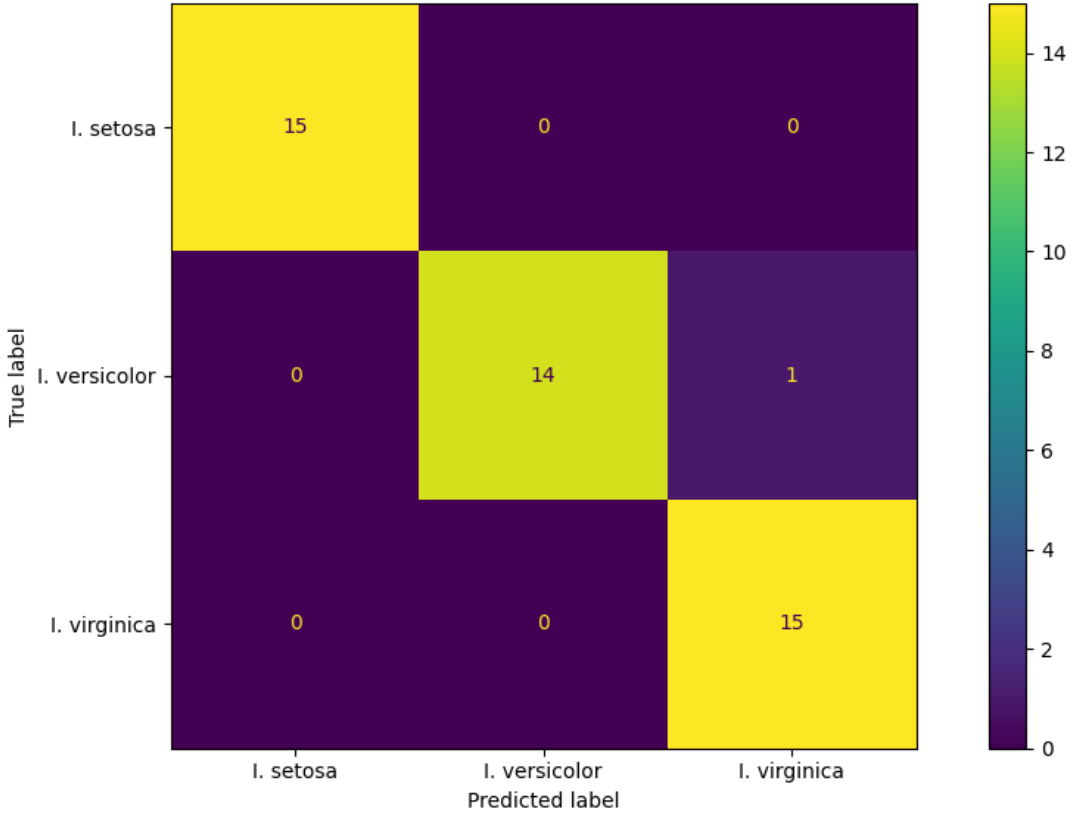
- Predict the species using the test partition.
- Plot the confusion matrix
- Print the accuracy

in_class_coding.py

```
dt = DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

from sklearn import metrics

cf = metrics.confusion_matrix(y_test, y_pred)
print(cf)

print('Accuracy =', metrics.accuracy_score(y_test, y_pred))

import matplotlib.pyplot as plt

metrics.plot_confusion_matrix(dt, X_test, y_test)
plt.show()
```

Run: in_class_coding

```
[[15  0  0]
 [ 0 14  1]
 [ 0  0 15]]
Accuracy = 0.9777777777777777
```


I. setosa
I. versicolor
I. virginica
True label
15
0
0
0
14
1
0
0
15
I. setosa
I. versicolor
I. virginica
Predicted label
14
12
10
8
6
4
2
0

Do the following:

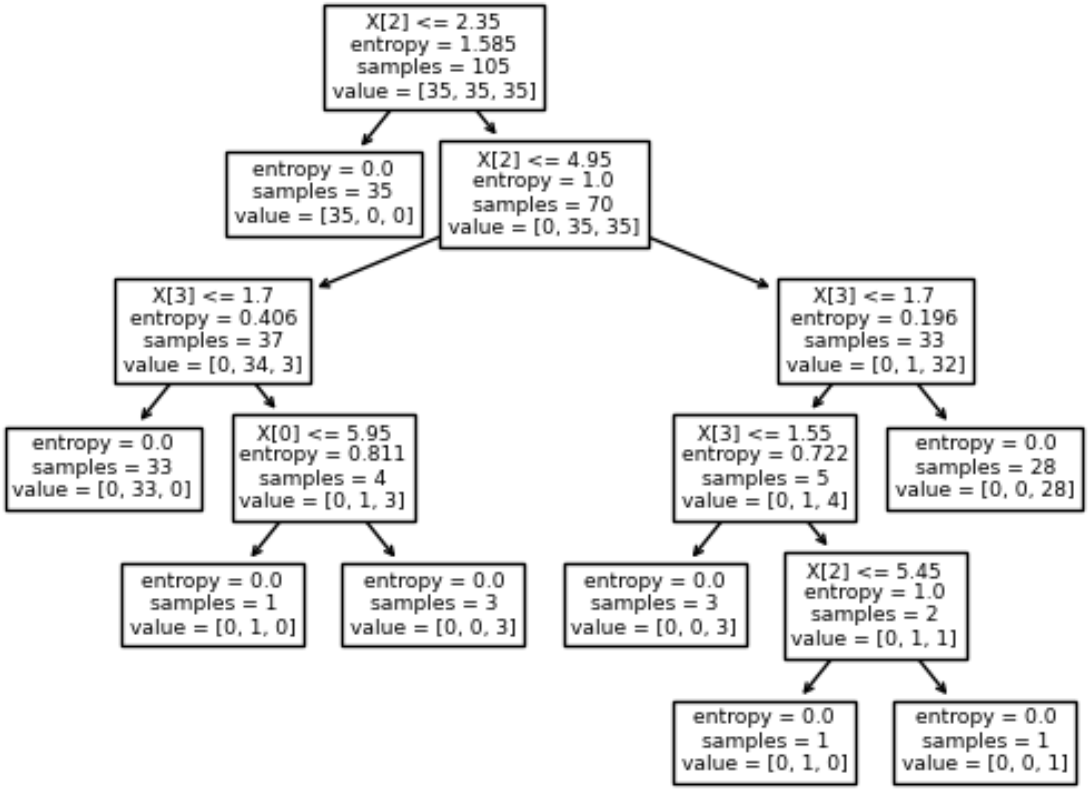
Display the decision tree

in_class_coding.py

```
16  from sklearn.tree import DecisionTreeClassifier
17
18  dt = DecisionTreeClassifier(criterion='entropy')
19  dt.fit(X_train, y_train)
20
21  y_pred = dt.predict(X_test)
22
23  from sklearn import metrics
24
25  cf = metrics.confusion_matrix(y_test, y_pred)
26  print(cf)
27
28  print('Accuracy =', metrics.accuracy_score(y_test, y_pred))
29
30  import matplotlib.pyplot as plt
31
32  metrics.plot_confusion_matrix(dt, X_test, y_test)
33  plt.show()
34
```

8 259

Run: in_class_coding

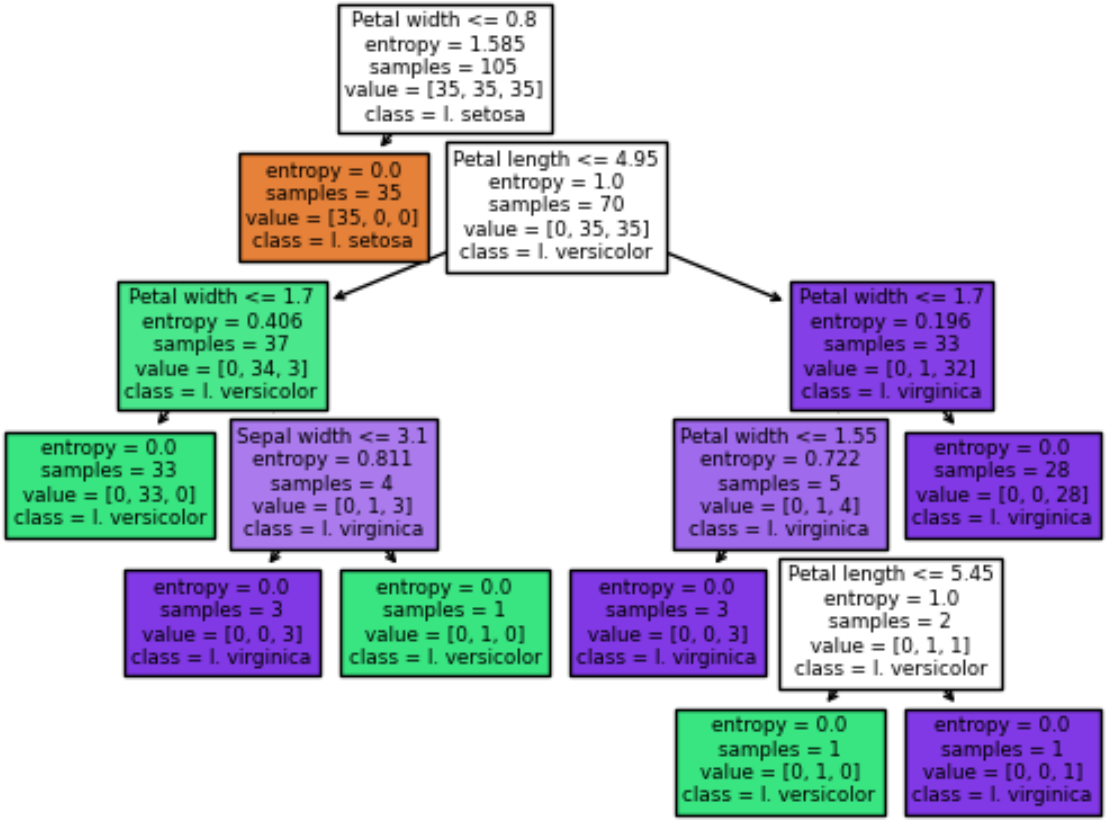

X[2] <= 2.35
entropy = 1.585
samples = 105
value = [35, 35, 35]
entropy = 0.0
samples = 35
value = [35, 0, 0]
X[2] <= 4.95
entropy = 1.0
samples = 70
value = [0, 35, 35]
X[3] <= 1.7
entropy = 0.406
samples = 37
value = [0, 34, 3]
X[3] <= 1.7
entropy = 0.196
samples = 33
value = [0, 1, 32]
entropy = 0.0
samples = 33
value = [0, 33, 0]
X[0] <= 5.95
entropy = 0.811
samples = 4
value = [0, 1, 3]
X[3] <= 1.55
entropy = 0.722
samples = 5
value = [0, 1, 4]
entropy = 0.0
samples = 28
value = [0, 0, 28]
entropy = 0.0
samples = 1
value = [0, 1, 0]
entropy = 0.0
samples = 3
value = [0, 0, 3]
entropy = 0.0
samples = 3
value = [0, 0, 3]
X[2] <= 5.45
entropy = 1.0
samples = 2
value = [0, 1, 1]
entropy = 0.0
samples = 1
value = [0, 1, 0]
entropy = 0.0
samples = 1
value = [0, 0, 1]

Do the following:

Format the tree to be more descriptive and save the decision tree as a png

in_class_coding.py

```
23  from sklearn import metrics
24
25  cf = metrics.confusion_matrix(y_test, y_pred)
26  print(cf)
27
28  print('Accuracy =', metrics.accuracy_score(y_test, y_pred))
29
30  import matplotlib.pyplot as plt
31
32  metrics.plot_confusion_matrix(dt, X_test, y_test)
33  plt.show()
34
35  from sklearn import tree
36
37  plt.figure(2)
38  fn = X.columns
39  cn = y.unique()
40  irisTree = tree.plot_tree(dt, feature_names=fn, class_names=cn, filled=True)
41  plt.savefig('IrisDT.png')
42  plt.show()
43
```


Petal width <= 0.8
entropy = 1.585
samples = 105
value = [35, 35, 35]
class = I. setosa
entropy = 0.0
samples = 35
value = [35, 0, 0]
class = I. setosa
Petal length <= 4.95
entropy = 1.0
samples = 70
value = [0, 35, 35]
class = I. versicolor
Petal width <= 1.7
entropy = 0.406
samples = 37
value = [0, 34, 3]
class = I. versicolor
Petal width <= 1.7
entropy = 0.196
samples = 33
value = [0, 1, 32]
class = I. virginica
entropy = 0.0
samples = 33
value = [0, 33, 0]
class = I. versicolor
Sepal width <= 3.1
entropy = 0.811
samples = 4
value = [0, 1, 3]
class = I. virginica
Petal width <= 1.55
entropy = 0.722
samples = 5
value = [0, 1, 4]
class = I. virginica
entropy = 0.0
samples = 28
value = [0, 0, 28]
class = I. virginica
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = I. virginica
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = I. versicolor
entropy = 0.0
samples = 3
value = [0, 0, 3]
class = I. virginica
Petal length <= 5.45
entropy = 1.0
samples = 2
value = [0, 1, 1]
class = I. versicolor
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = I. versicolor
entropy = 0.0
samples = 1
value = [0, 0, 1]
class = I. virginica

Do the following:

Show Feature importance

in_class_coding.py

```
35  from sklearn import tree
36
37  plt.figure(2)
38  fn = X.columns
39  cn = y.unique()
40  irisTree = tree.plot_tree(dt, feature_names=fn, class_names=cn, filled=True)
41  plt.savefig('IrisDT.png')
42  plt.show()
43
44  # Feature Importance
45
46  print('Feature Importance:', dt.feature_importances_)
47
```

Run: in_class_coding

```
C:\Users\Reza\Desktop\ITP449_Fall2020\Class\venv\Scripts\python.exe "C:/Users/Reza/Desktop/ITP449_Fall2020/Class/In Class Cod
[[15  0  0]
 [ 0 14  1]
 [ 0  0 15]]
Accuracy = 0.9777777777777777
Feature Importance: [0.0096721  0.01949941 0.87089281 0.09993569]
```

Petal Length is the most important feature.