Tree Methods

IOE 373 Lecture 24

Topics

- Classification and Regression Trees
- Tree Impurities
- Ensembles

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 >= 106) 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

Classification Trees

- Let us denote the dependent (response) variable by y and the predictor variables by $x_1, x_2, x_3, \ldots, x_p$.
  - In classification, the outcome variable will be a categorical variable.
  - Recursive partitioning divides up the p-dimensional space of the x variables into non-overlapping multidimensional rectangles.
  - The X variables here are considered to be continuous, binary, or ordinal. This division is accomplished recursively.

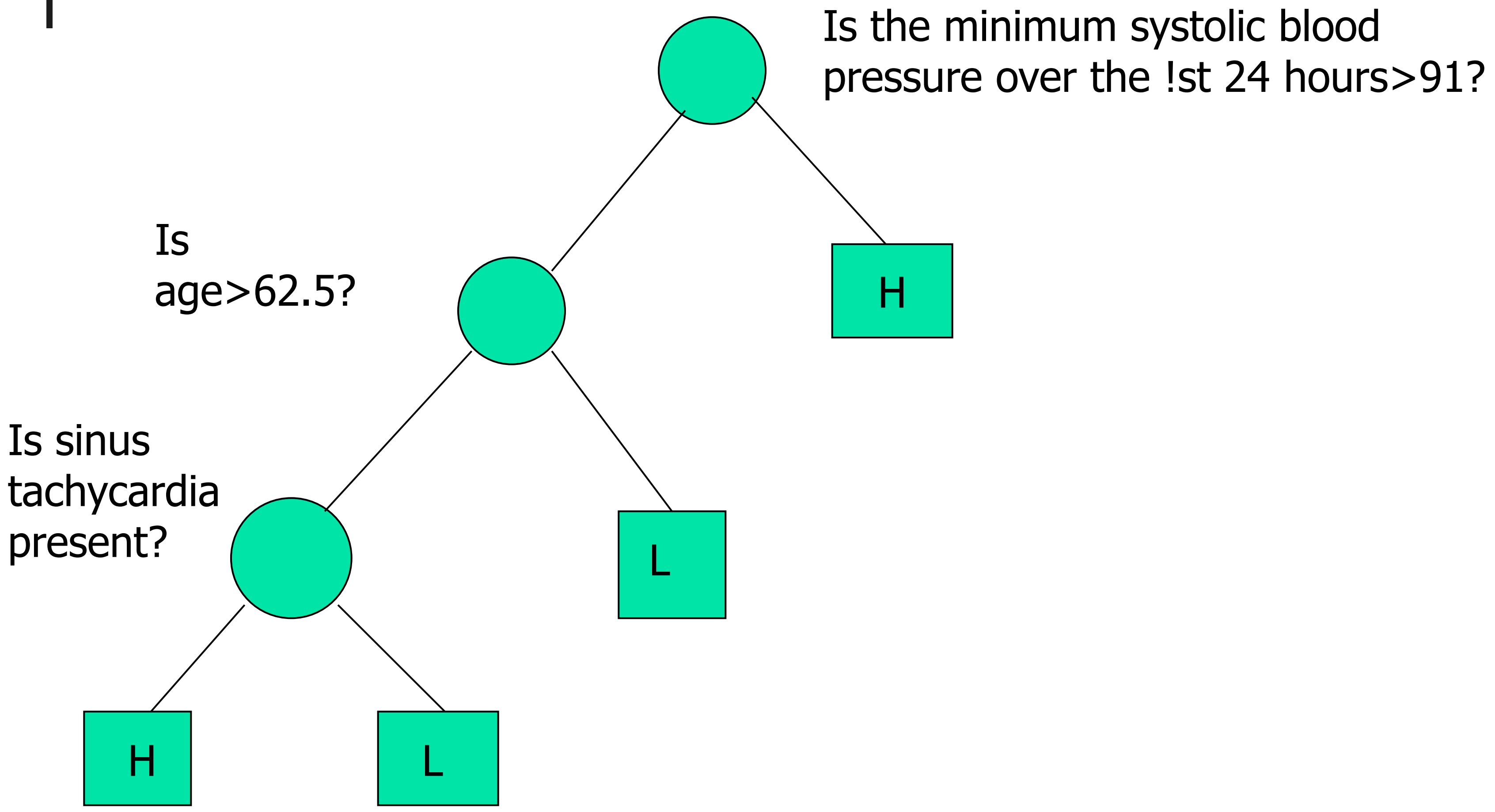
Variety of approaches used

- CART developed by Breiman Friedman Olsen and Stone: "Classification and Regression Trees"
- C4.5 A Machine Learning Approach by Quinlan
- Engineering approach by Sethi and Sarvarayudu

Example 1

- University of California- a study into patients *after* admission for a heart attack
- 19 variables collected during the first 24 hours for 215 patients (for those who survived the 24 hours)
- Question: Can the high risk (will not survive 30 days) patients be identified?

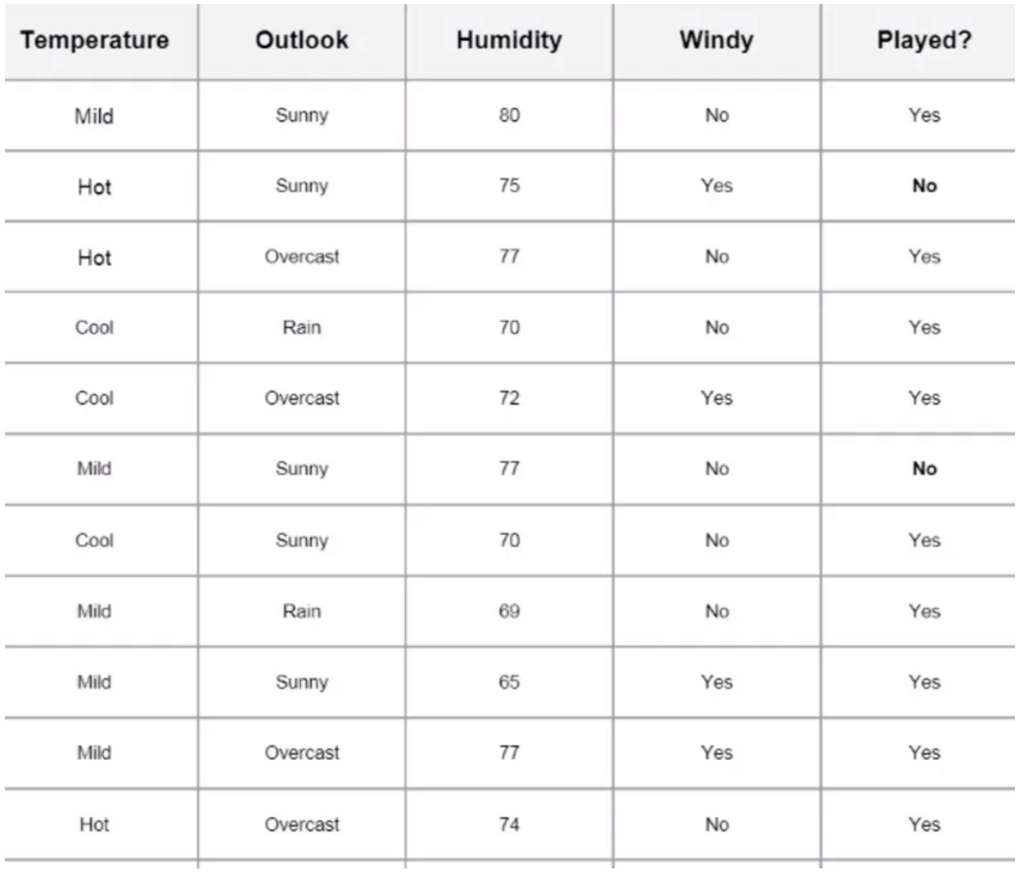
Answer


Is the minimum systolic blood pressure over the !st 24 hours>91?
Is age>62.5?
Is sinus tachycardia present?
H
L
H
L

Example 2

- You play Tennis every weekend and invite one of your friends
- Your friend doesn't always show up to play
- When you ask why he doesn't show up sometimes, he tells you that for him wanting to play depends on the following factors: weather, temperature, humidity, wind, etc.
- To test this, you start keeping track of some of these factors:

Data Table



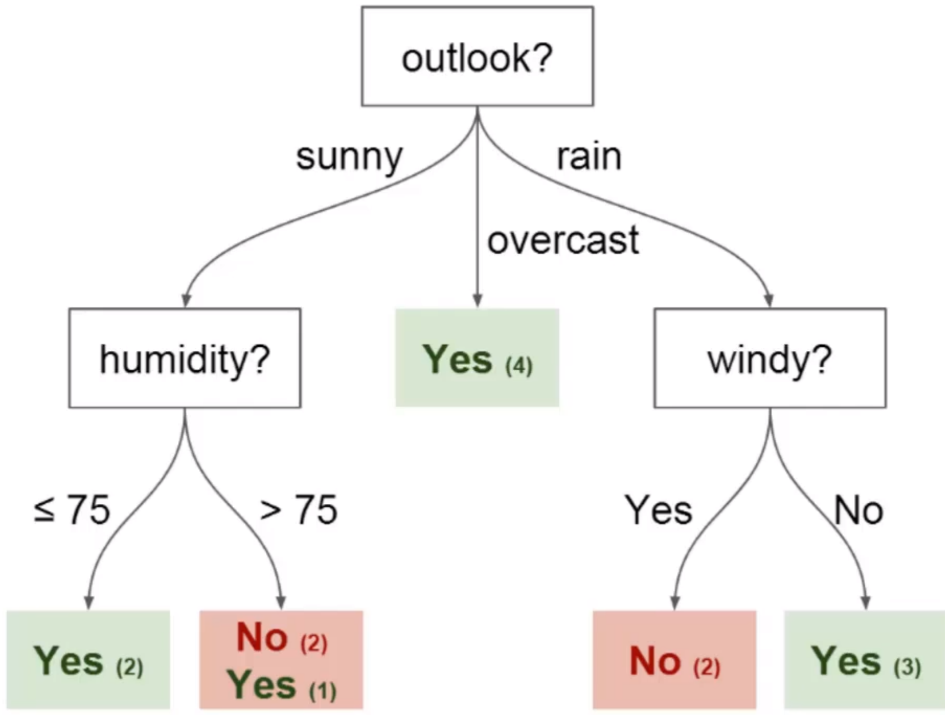
Temperature	Outlook	Humidity	Windy	Played?
Mild	Sunny	80	No	Yes
Hot	Sunny	75	Yes	**No**
Hot	Overcast	77	No	Yes
Cool	Rain	70	No	Yes
Cool	Overcast	72	Yes	Yes
Mild	Sunny	77	No	**No**
Cool	Sunny	70	No	Yes
Mild	Rain	69	No	Yes
Mild	Sunny	65	Yes	Yes
Mild	Overcast	77	Yes	Yes
Hot	Overcast	74	No	Yes

Features of CART

- Binary Splits
- Splits based only on one variable

In this tree we have:

- Nodes
  - Split for the value of a certain attribute
- Edges
  - Outcome of a split to next node



How Is the Tree Produced?

Recursive partitioning: Repeatedly split the records into two parts so as to achieve maximum homogeneity of outcome within each new part

Stopping Tree Growth: A fully grown tree is too complex and will overfit

Impurity of a Node

- Need a measure of impurity of a node to help decide on how to split a node, or which node to split
- The measure should be at a maximum when a node is equally divided amongst all classes
- The impurity should be zero if the node is all one class

Measures of Impurity

- Gini Index
- Information, or Entropy

Gini Index

Gini Index for rectangle *A*

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

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

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

Entropy

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

- *p* = proportion of cases in rectangle *A* that belong to class *k* (out of *m* classes)
- Entropy ranges between 0 (most pure) and $\log_2(m)$ (equal representation of classes)

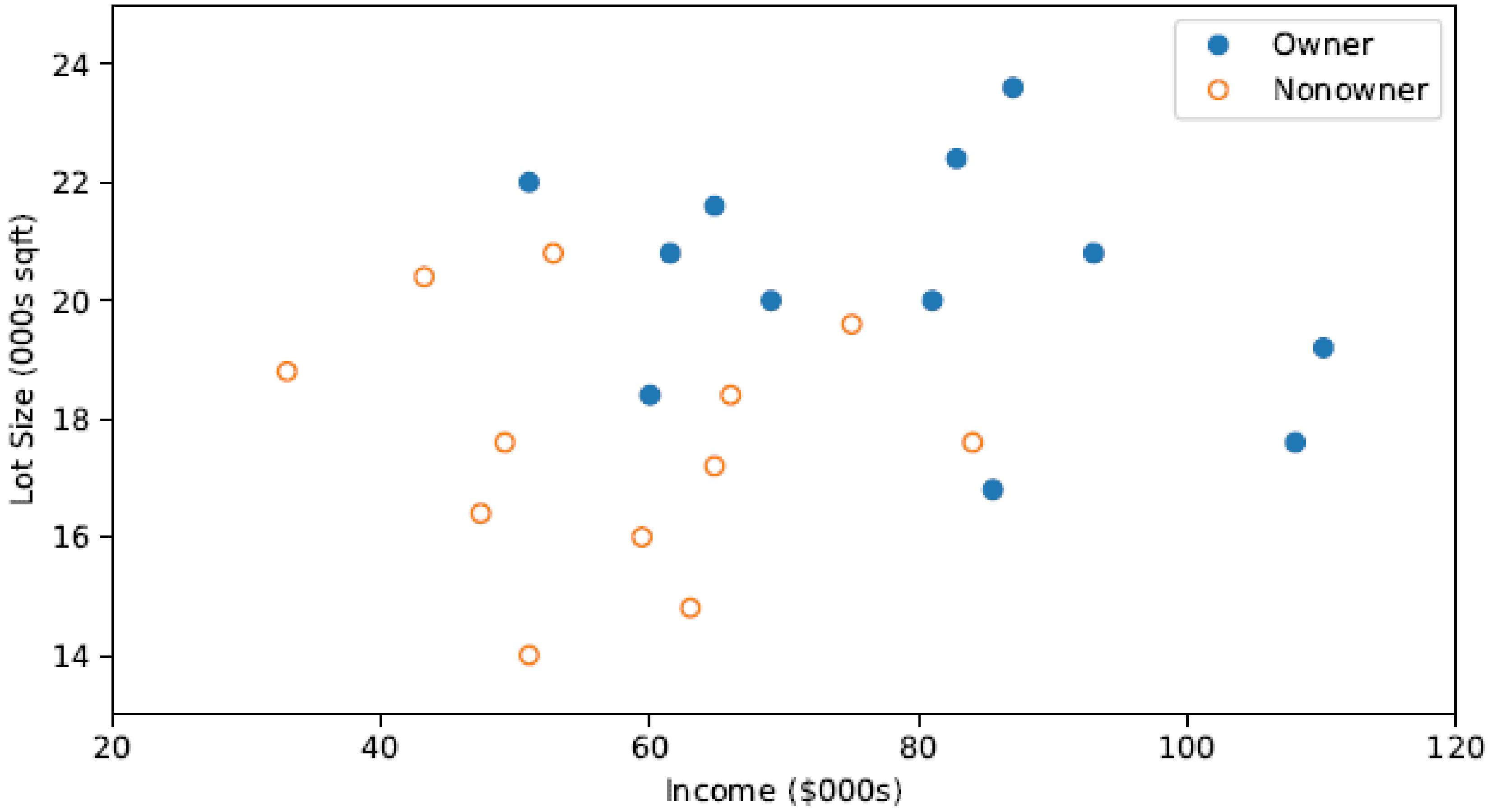
Impurity and Recursive Partitioning

- 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

Example – Classifying Riding Mower Owners

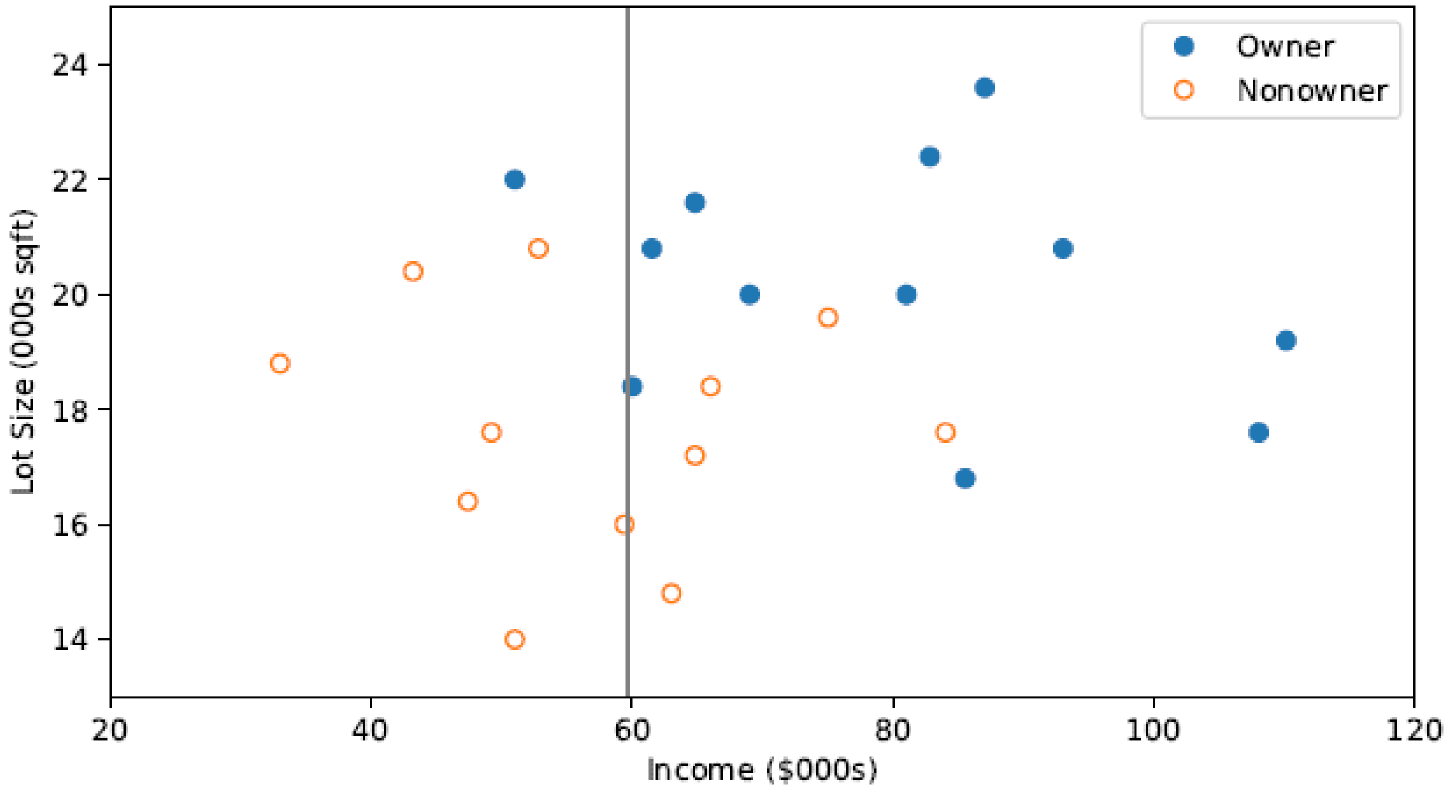
- A riding-mower manufacturer would like to find a way of classifying families in a city into those likely to purchase a riding mower and those not likely to buy one.

Income	Lot_Size	Ownership
60	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87	23.6	owner
110.1	19.2	owner
108	17.6	owner
82.8	22.4	owner
69	20	owner
93	20.8	owner
51	22	owner
81	20	owner
75	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84	17.6	non-owner
49.2	17.6	non-owner
59.4	16	non-owner
66	18.4	non-owner
47.4	16.4	non-owner
33	18.8	non-owner
51	14	non-owner
63	14.8	non-owner

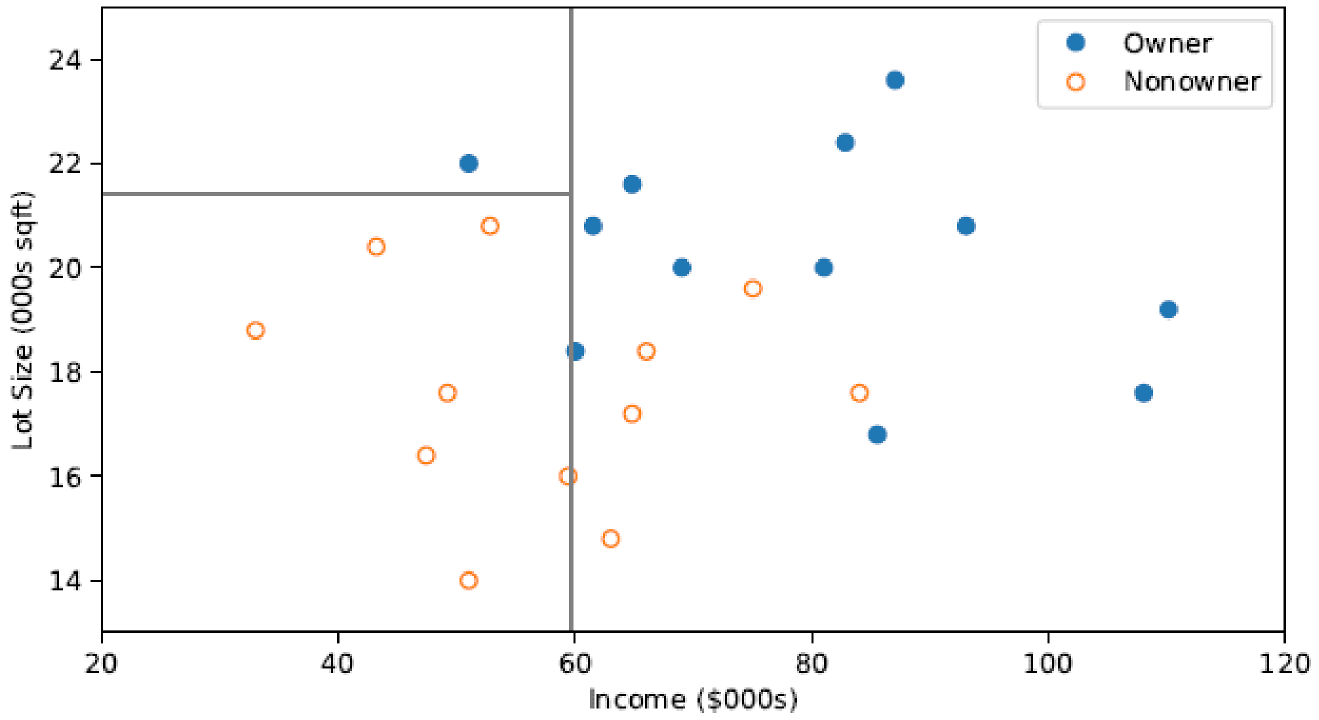


Owner
Nonowner
Lot Size (000s sqft)
Income ($000s)
24
22
20
18
16
14
20
40
60
80
100
120

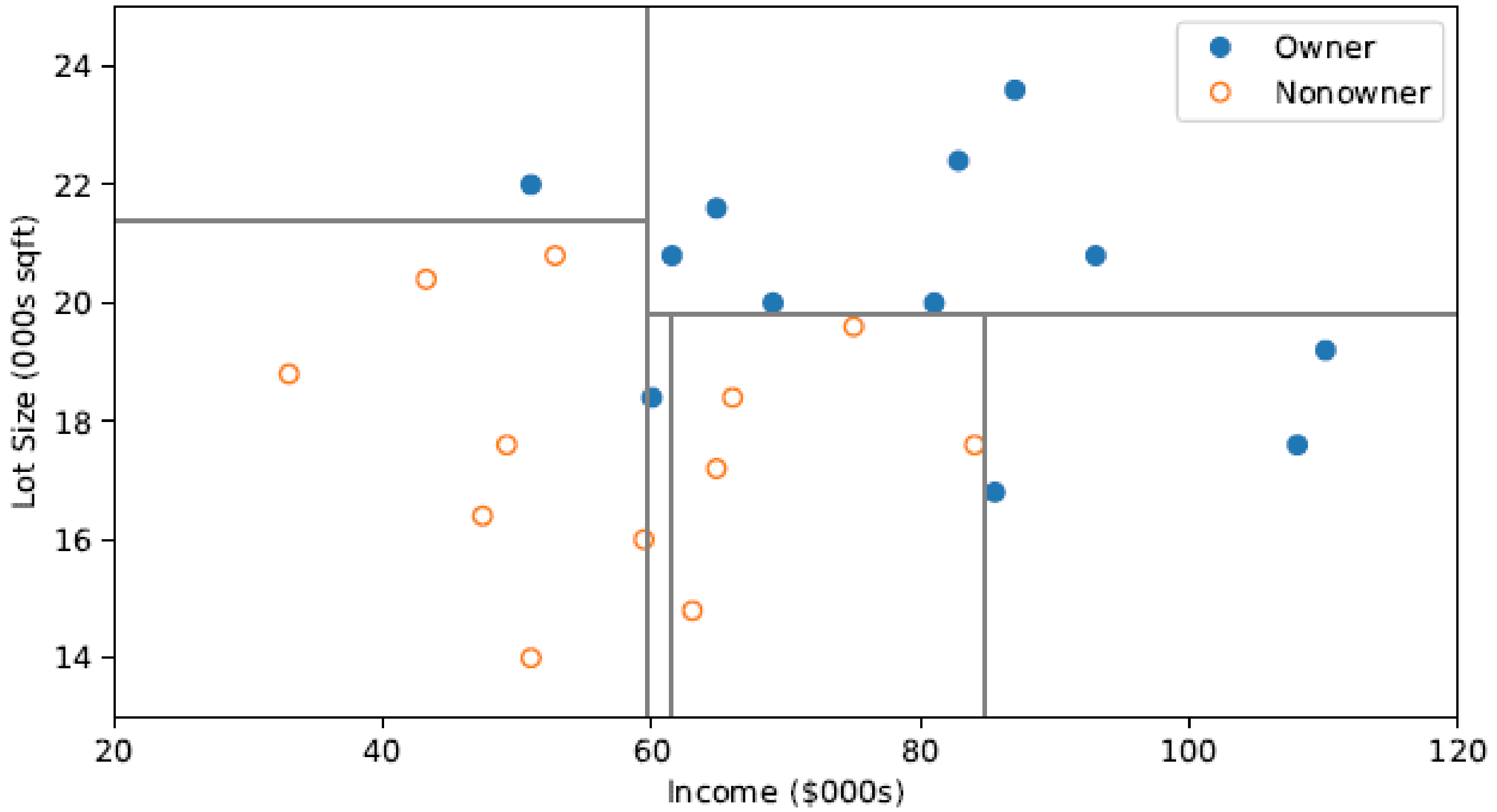
The first split: Income = 59.7



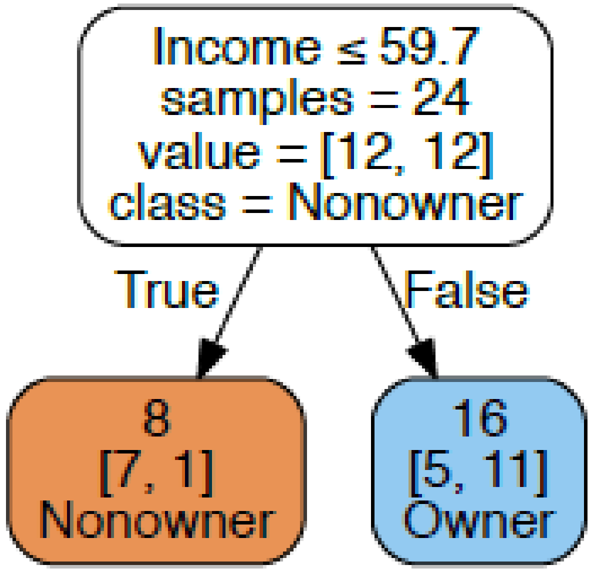
Second Split: Lot size = 21.4



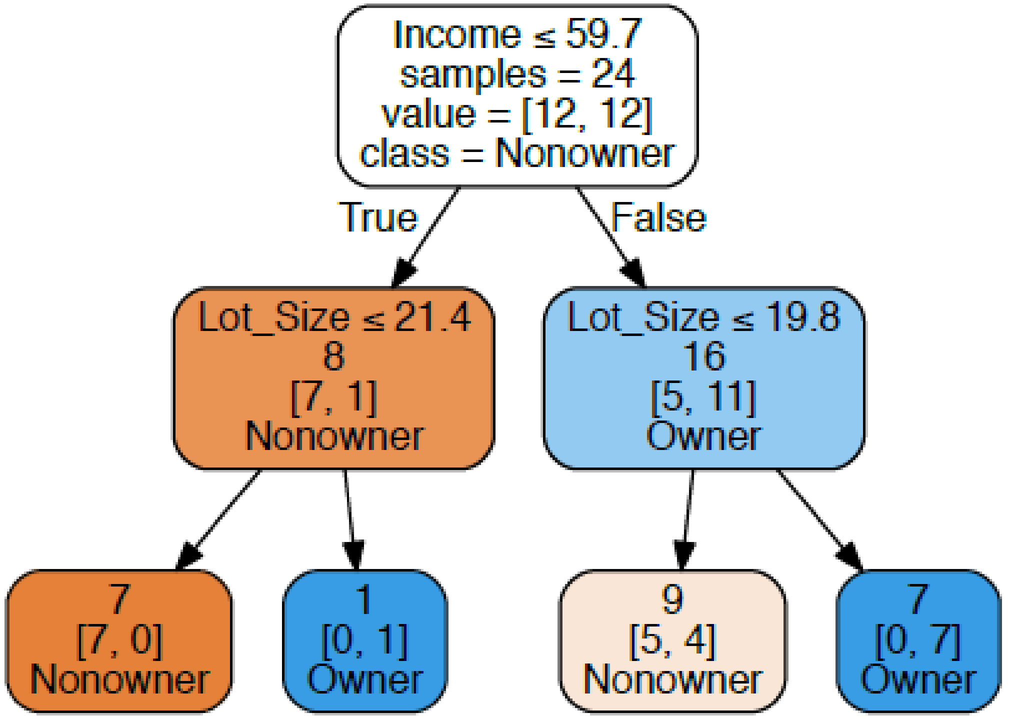
After All Splits



First Split – The Tree

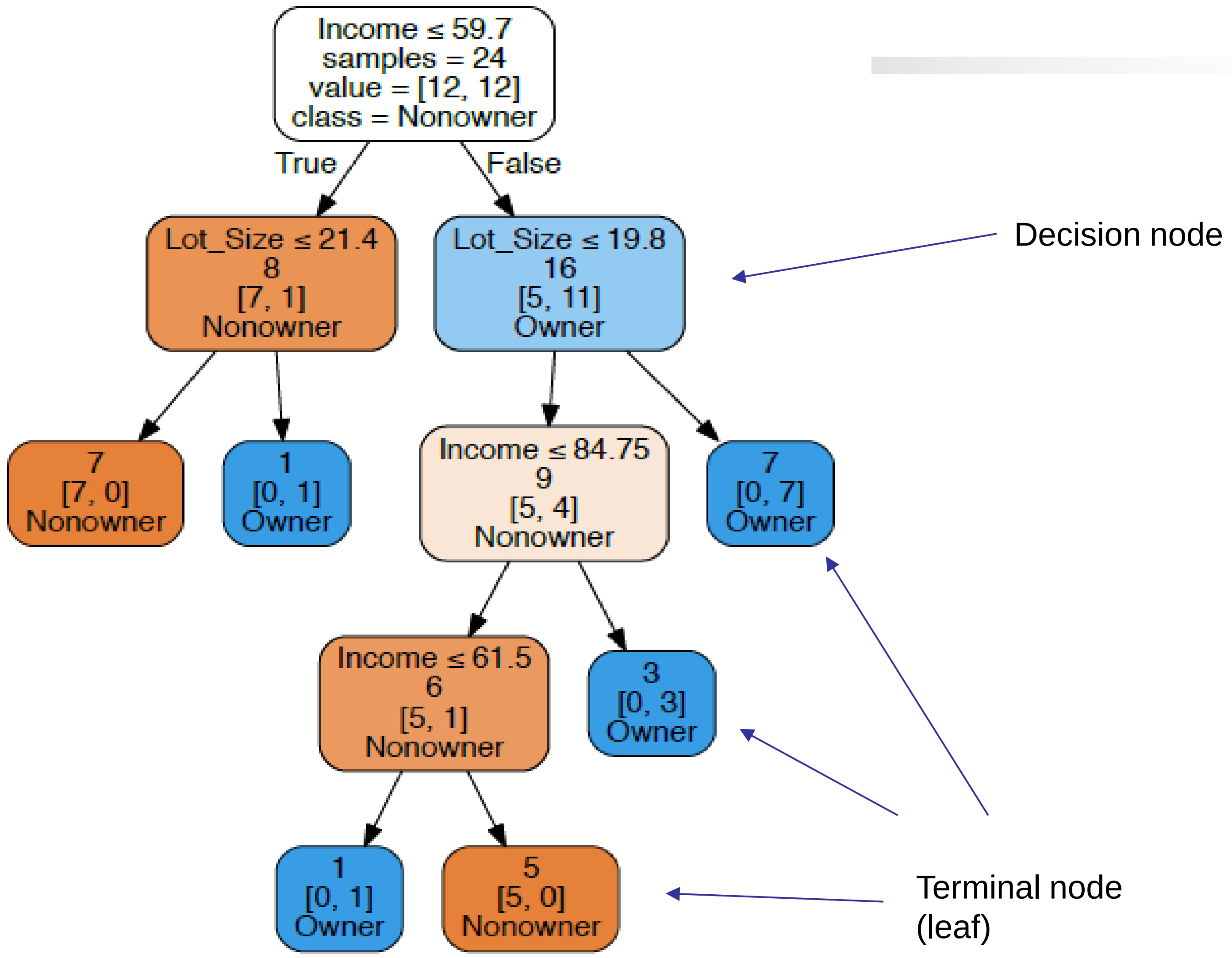


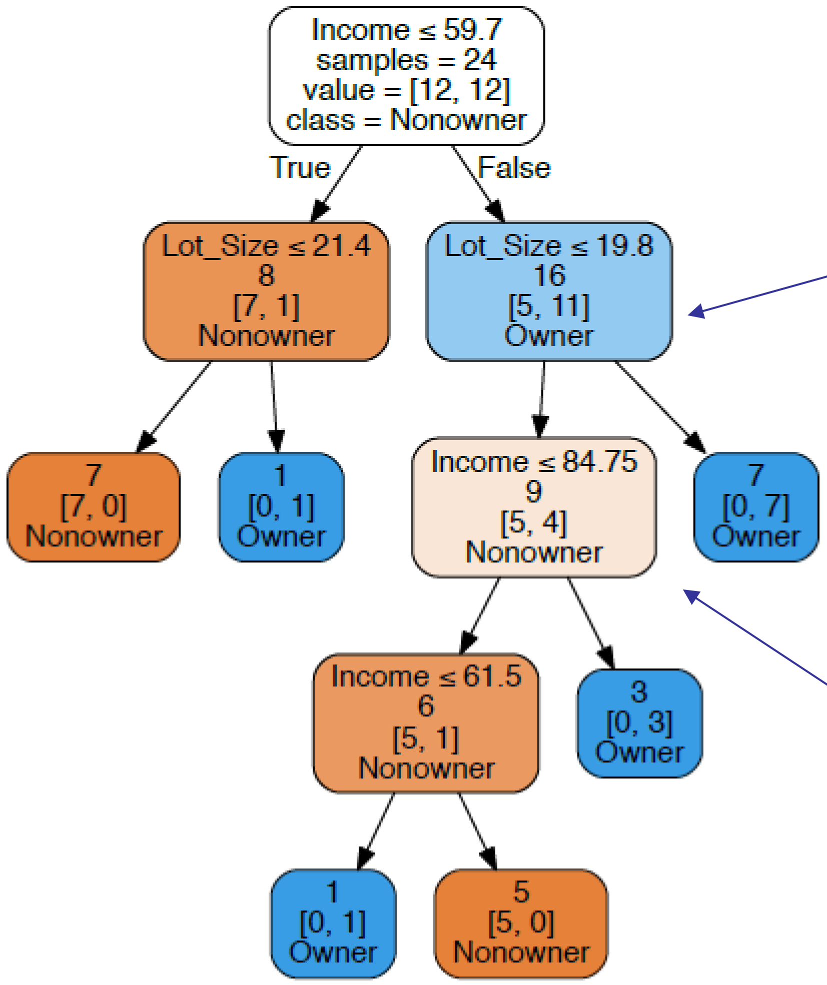
Tree After 3 Splits



The first split is on Income, then the next split is on Lot Size for both the low income group (at lot size 21.4) and the high income split (at lot size 19.8)

Tree After All Splits

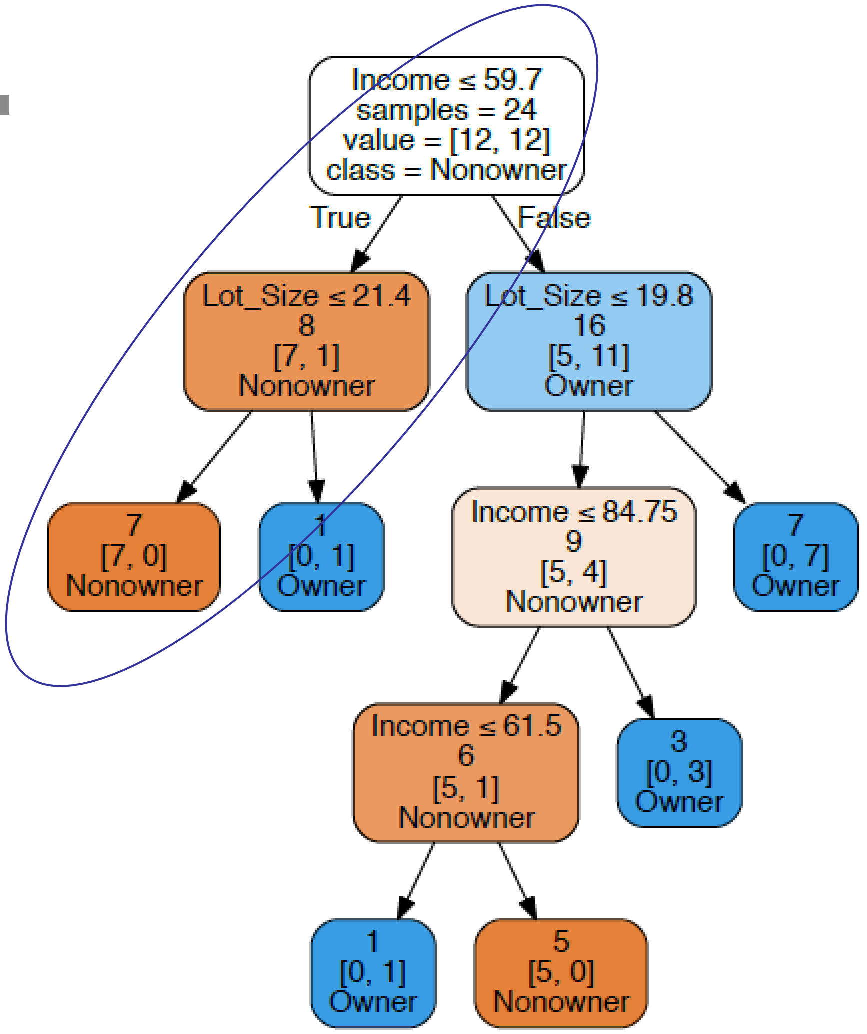




The dominant class in this portion of the first split (those with income >= 59.7) is "owner" – 11 owners and 5 non-owners

The next split for this group of 16 will be on the basis of Income, splitting at 84.75

Read down the tree to derive rules



If Income <= 59.7 AND
Lot Size <= 21.4,
classify as “Nonowner”

Model Performance - Confusion matrix

- It is used to measure the performance of a classification algorithm. It calculates the following metrics:

1. **Accuracy:** Proportion of correctly predicted results among the total number of observations

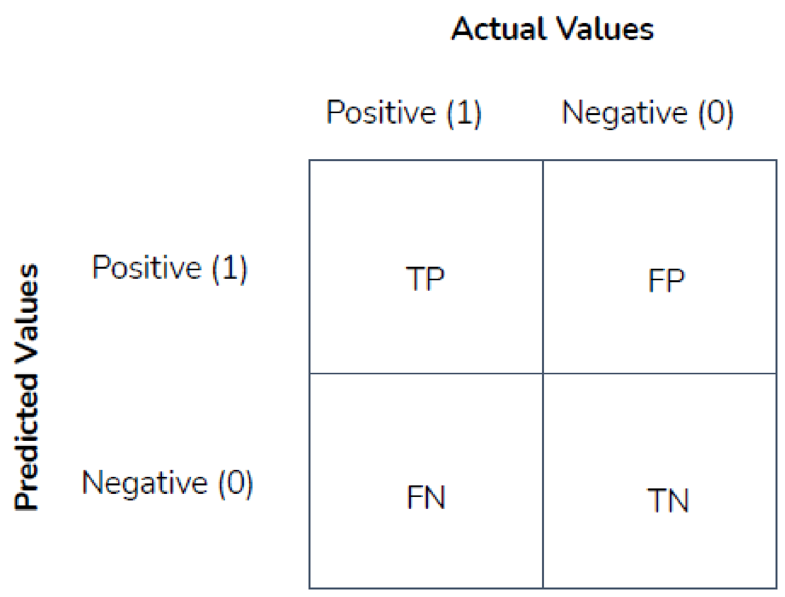
   Accuracy = (TP+TN)/(TP+FP+FN+TN)

2. **Precision:** Proportion of true positives to all the predicted positives i.e. how valid the predictions are

   Precision = (TP)/(TP+FP)

3. **Recall:** Proportion of true positives to all the actual positives i.e. how complete the predictions are

   Recall = (TP)/(TP+FN)

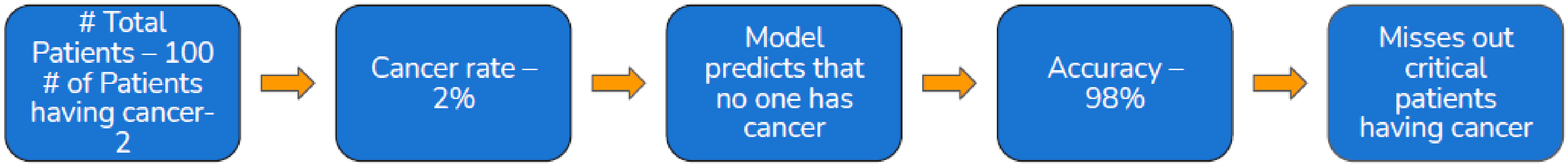


		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

*Recall or Sensitivity: TP/(TP+FN)
**Specificity: TN/(TN+FP)

Why Accuracy is not always good performance measure?

Accuracy is simply the overall % of correct predictions and can be high even for very useless models.

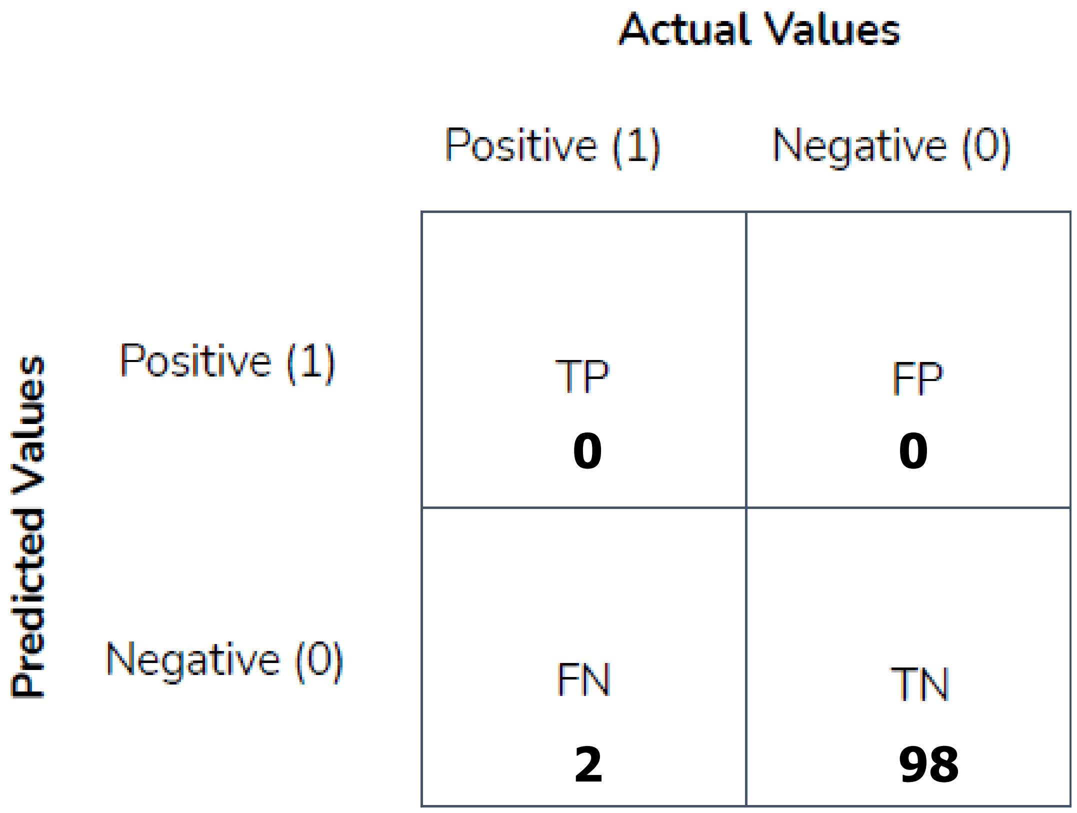

# Total Patients – 100 # of Patients having cancer- 2
Cancer rate – 2%
Model predicts that no one has cancer
Accuracy – 98%
Misses out critical patients having cancer

- Here Accuracy will be 98%, even if we predict all patients do not have cancer.
- In this case, Recall should be used as a measure of the model performance; high recall implies fewer false negatives
- Fewer false negatives implies a lower chance of 'missing' a cancer patient i.e. predicting a cancer patient as one not having cancer.
- This is where we need other metrics to evaluate model performance.

- The other metrics are Recall and Precision
  - Recall - What % of actuals 1s did I capture in my prediction?
  - Precision - What % of my predicted 1s are actual 1s?
- There is a tradeoff - as you try to increase Recall, Precision will reduce and vice versa
- This tradeoff can be used to figure out the right threshold to use for the model

For the cancer problem…

Accuracy = (TP+TN)/(TP+TN+FN+TN)=(0+98)/(0+98+2+0)=98%



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP **0**	FP **0**
	Negative (0)	FN **2**	TN **98**

Precision = TP/(TP+FP) =0/(0+0)=0%

Recall=TP/(TP+FN) =0/(0+2)=0%

Is there a performance measure that can cover both Precision and Recall?

- F1 Score is a measure that takes into account both Precision and Recall.
- F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

- The highest possible value of F1 score is 1, indicating perfect precision and recall, and the lowest possible value is 0.

Import libraries and data

- We need the usual libraries

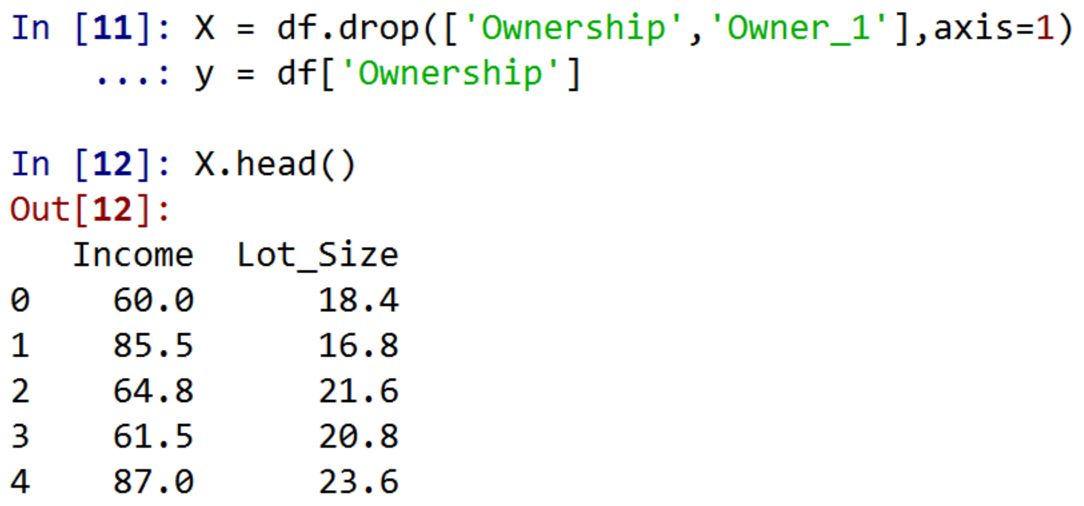
```
In [4]: import pandas as pd
   ...: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: import seaborn as sns
   ...: %matplotlib inline
```

- Let’s read the data set and check it:

```
In [5]: df = pd.read_csv('MowersTable.csv')

In [6]: df.head()
Out[6]:
   Income  Lot_Size Ownership  Owner_1
0    60.0      18.4     owner        1
1    85.5      16.8     owner        1
2    64.8      21.6     owner        1
3    61.5      20.8     owner        1
4    87.0      23.6     owner        1
```

Train Test Split

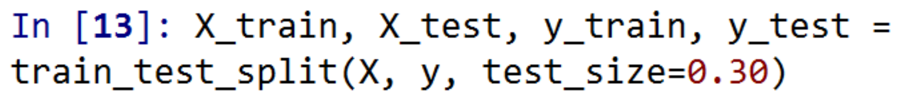
- Let's split up the data into a training set and a test set
  - First let’s create our factor matrix X and our response array y:



```
In [11]: X = df.drop(['Ownership','Owner_1'],axis=1)
    ...: y = df['Ownership']

In [12]: X.head()
Out[12]:
   Income  Lot_Size
0    60.0      18.4
1    85.5      16.8
2    64.8      21.6
3    61.5      20.8
4    87.0      23.6
```

Train Test Split

- Now, let’s split using a 30% for test:



```
In [13]: X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.30)
```

Let’s run the Decision Tree Classifier

- Import the library:

```
In [14]: from sklearn.tree import DecisionTreeClassifier
```

- Create a decision tree object:

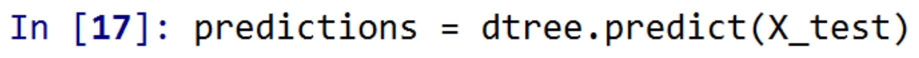
```
In [15]: dtree = DecisionTreeClassifier()
```

- “Fit” the tree:

```
In [16]: dtree.fit(X_train,y_train)
Out[16]: DecisionTreeClassifier()
```

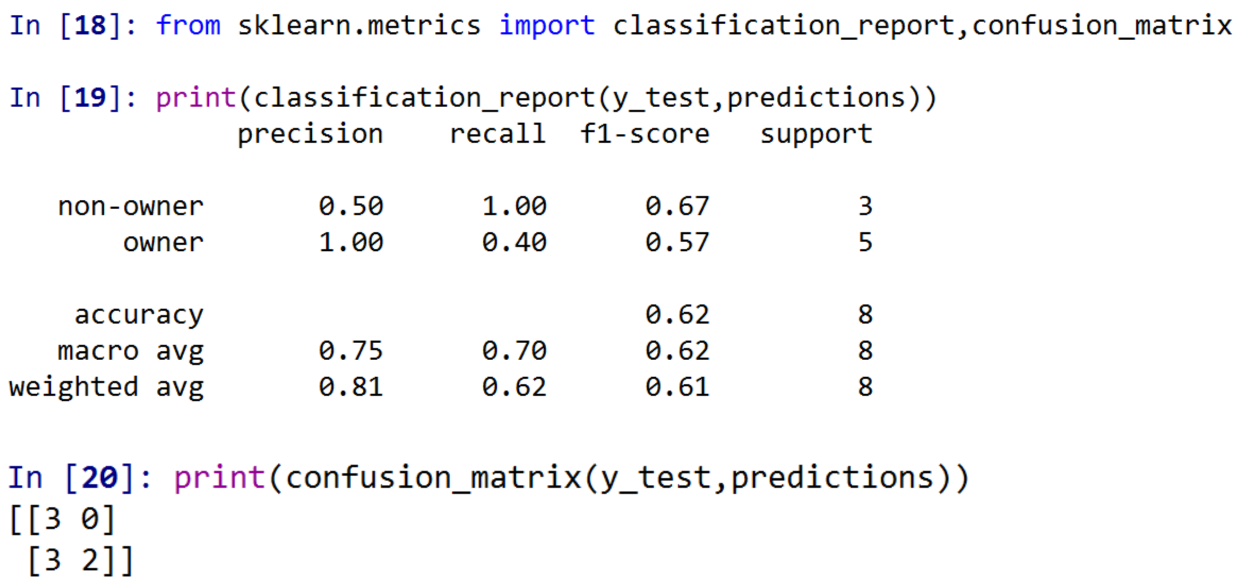
Prediction and Evaluation

- Predict:



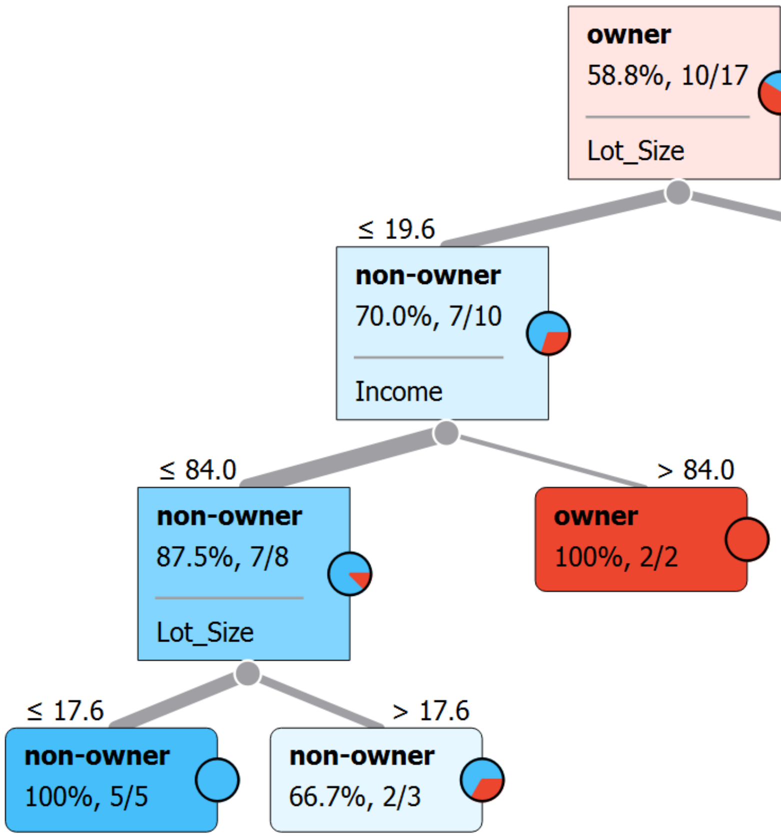
```
In [17]: predictions = dtree.predict(X_test)
```

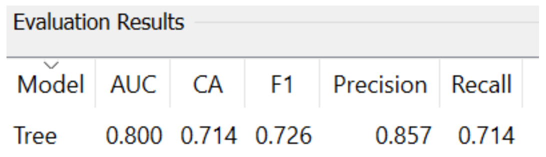
- Evaluate:



```
In [18]: from sklearn.metrics import classification_report,confusion_matrix

In [19]: print(classification_report(y_test,predictions))
              precision    recall  f1-score   support

   non-owner       0.50      1.00      0.67         3
       owner       1.00      0.40      0.57         5

    accuracy                           0.62         8
   macro avg       0.75      0.70      0.62         8
weighted avg       0.81      0.62      0.61         8

In [20]: print(confusion_matrix(y_test,predictions))
[[3 0]
 [3 2]]
```

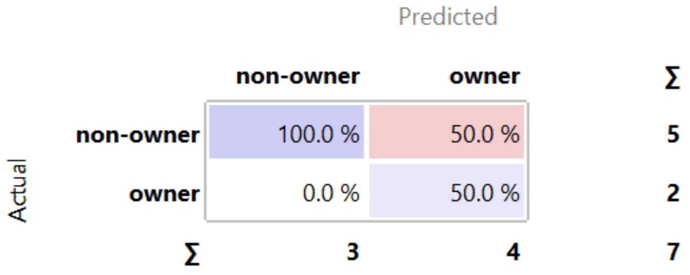
Decision Tree





Evaluation Results

Model	AUC	CA	F1	Precision	Recall
Tree	0.800	0.714	0.726	0.857	0.714



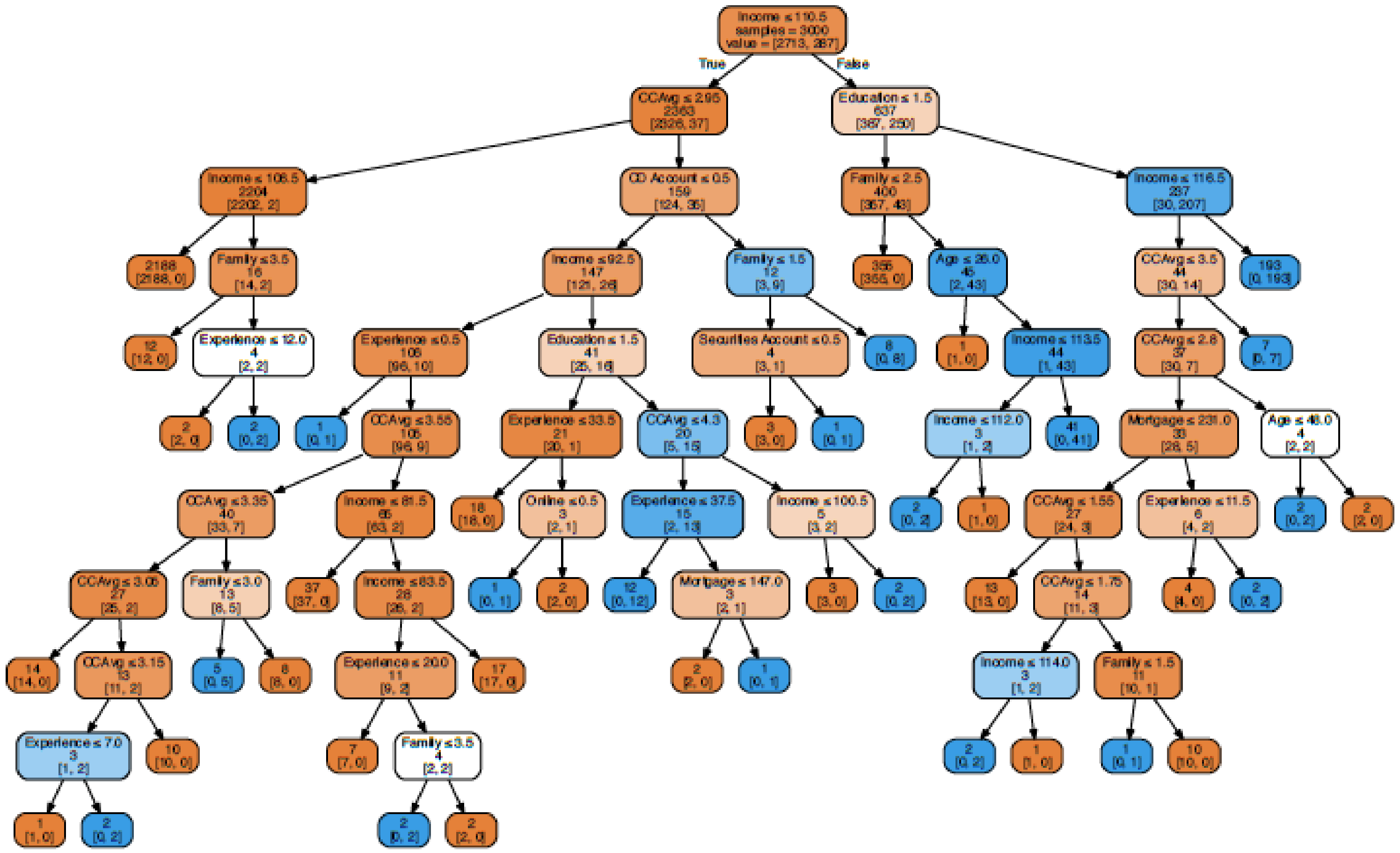
		Predicted		
		non-owner	owner	Σ
Actual	non-owner	100.0 %	50.0 %	5
	owner	0.0 %	50.0 %	2
	Σ	3	4	7

The Overfitting Problem

Full trees are complex and overfit the data

- Natural end of process is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Consider a Loan Acceptance data set with more records and more variables than the Riding Mower data – the full tree is very complex

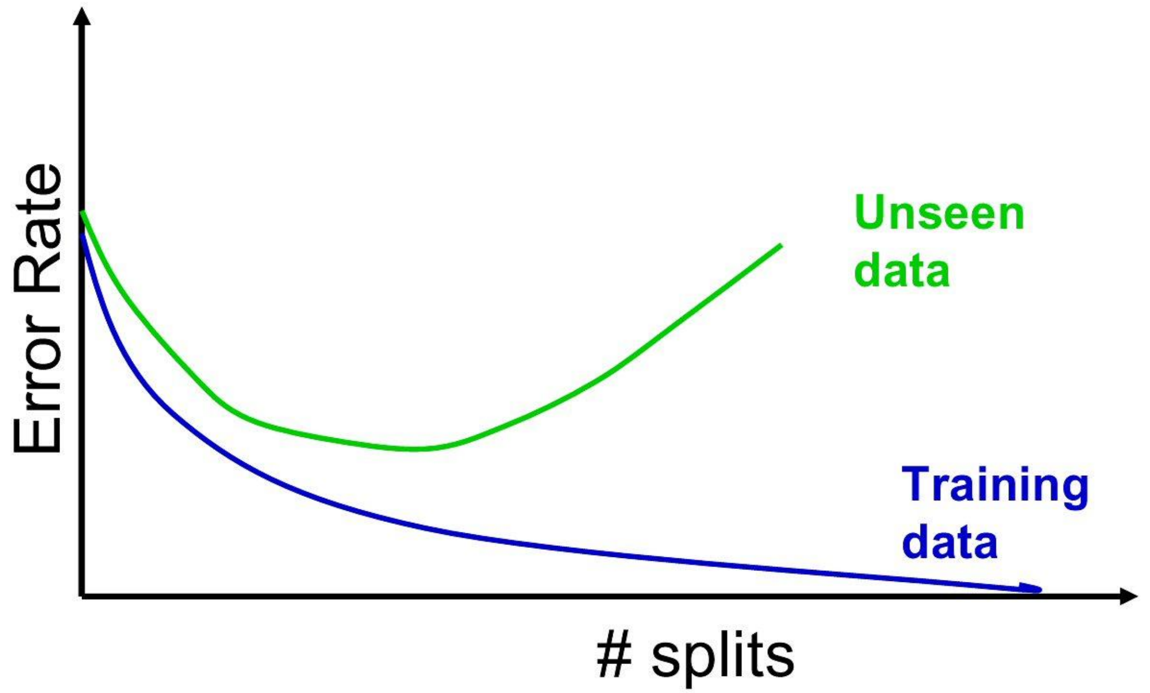
Full trees are too complex – they end up fitting noise, overfitting the data



Trees can also be unstable

- If 2 or more variables are of roughly equal importance, which one CART chooses for the first split can depend on the initial partition into training and validation
- A different partition into training/validation could lead to a different initial split
- This can cascade down and produce a very different tree from the first training/validation partition
- Solution is to try many different training/validation splits – “cross validation”

Overfitting & instability produce poor predictive performance – past a certain point in tree complexity, the error rate on new data starts to increase



Regression Trees

Regression Tree is Similar, Except…

- 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)

Ensemble Variants

- Harness “wisdom of the crowd”
- Random Forest
- Boosted Trees

Random Forest

- Draw multiple random samples from data ("bootstrap resampling")
- Fit tree to each resample using a *random set of predictors*
- Combine the classifications/predictions from all the resampled trees (the "forest") to obtain improved predictions
- Basic idea: Taking an average of multiple estimates (models) is more reliable than just using a single estimate

Boosted Trees

- Fits a succession of single trees
- Each successive fit up-weights the misclassified records from prior stage
- You now have a set of classifications or predictions, one from each tree
- Use weighted voting for classification, weighted average for prediction, higher weights to later trees

Advantages of trees

- Relatively easy to use, understand/visualize
- Single trees 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
- Disadvantage of single trees: instability and poor predictive performance

Summary

- Classification and Regression Trees are an easily understandable and transparent method for predicting or classifying new records
- A single tree is a graphical representation of a set of rules
- Tree growth must be stopped to avoid overfitting of the training data
- Ensembles (random forests, boosting) improve predictive performance, but you lose interpretability and the rules embodied in a single tree

Thank you!!