Chapter 6
Decision Trees

Modeling Phase

- Select and apply one or more modeling techniques
- Calibrate model settings to optimize results
- If necessary, additional data preparation may be required for supporting a particular technique

Dataset: Adult

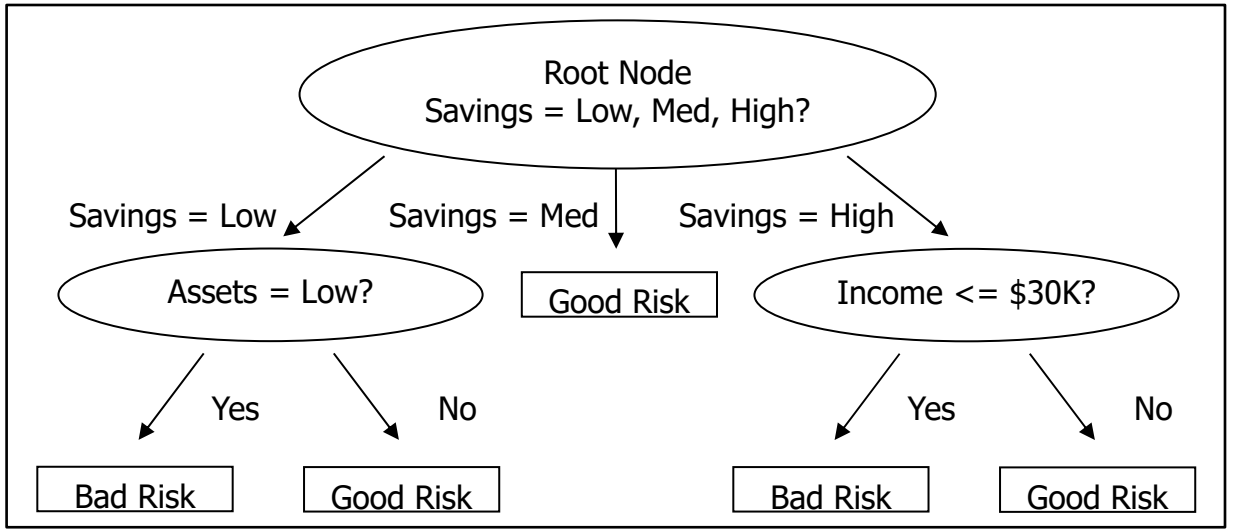
- UCI Machine Learning Repository
  - https://archive.ics.uci.edu/ml/datasets/Adult
- 2 data sets adapted from the Adult Dataset
  - adult_ch6_test
  - adult_ch6_training
- 2 predictor variables
  - Marital Status – categorical predictor with classes married, divorced, never-married, separated, widowed
  - Cap_gains_losses – numerical predictor, equal to capital gains + capital losses
- 1 target variable
  - Income – categorical target variable with 2 classes: >50k (> $50,000 income annually) and <=50k (<= $50,000 income annually)

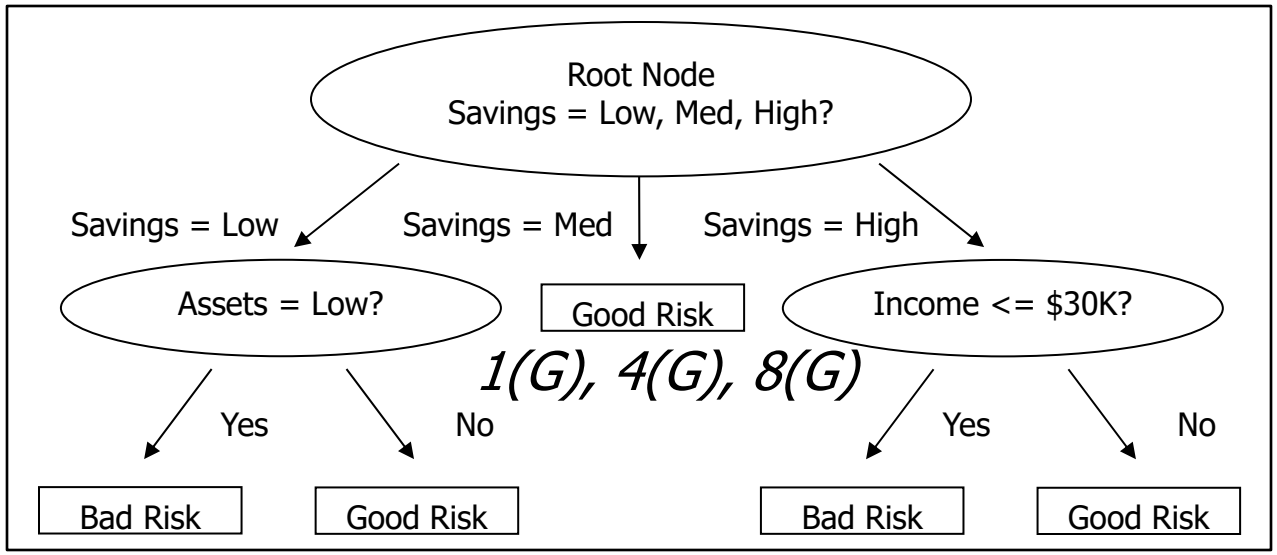
Model: Decision Trees

- Decision Trees
  - A Decision Tree is collection of decision nodes, connected by branches, extending downward from root node (top of the decision tree) to terminating leaf nodes (bottom of the decision tree)
  - Beginning with the root node, variables are tested at the decision nodes with each possible outcome resulting in a branch
  - Each branch leads to decision node or terminating leaf node
  - The decision tree partitions the dataset into subsets; i.e., branches terminate at pure leaf nodes which describe all subsets of records arriving at leaf nodes with the same target class value
  - A diverse leaf node has a mixture of records with different target class values ("Good Risk" and "Bad Risk"). The decision tree algorithm may not be able to split the records into pure subsets
    - For example, subset of records has *Savings* = "High", *Income* <= $30,000, and *Assets* = "Low". Leaf node contains 2 "Good Risk", and 3 "Bad Risk" records
  - All records may contain the same predictor values. No way to split further leading to pure leaf node
    - 3/5 records are classified "Bad Risk" with 60% confidence

Decision Trees (*cont'd*)

- Example
  - *Credit Risk* is the target variable
  - Customers are classified as either "Good Risk" or "Bad Risk"
  - Predictor variables are *Savings* (Low, Med, High), *Assets* (Low, High), and *Income*
  - Highest-level decision node is root node and tests whether record has *Savings* = "Low", "Med", or "High"
  - Records are split according to value of *Savings*. For example, records with *Savings* = "Low" go down leftmost branch to next decision node
  - Records with *Savings* = "Med" proceed down middle branch to leaf node. This terminates branch with all records *Savings* = "Med" classified as "Good Risk"
  - Additional decision nodes not required because records are classified with 100% accuracy





2(B), 7(B) *5(G)* *3(B), 6(G)* *...*

Customer	Savings	Assets	Income ($1000s)	Credit Risk
1	Medium	High	75	Good
2	Low	Low	50	Bad
3	High	Medium	25	Bad
4	Medium	Medium	50	Good
5	Low	Medium	100	Good
6	High	High	25	Good
7	Low	Low	25	Bad
8	Medium	Medium	75	Good
...				

Requirements for using Decision Trees

- The Decision Tree is supervised classification method
- The target variable must be categorical
- A pre-classified target variable must be included in training set
- Decision trees "learn" by example, so the training set should contain records with varied attribute values
- If the training set systematically lacks definable subsets, classification becomes problematic
- There are different measures for leaf node purity
- Classification and Regression Trees (CART) and C5.0 are two leading algorithms used in data mining

Classification and Regression Trees

- Classification and Regression Trees (CART) were developed by Breiman, 1984
- Splits at decision nodes are binary, resulting in two branches
- CART recursively partitions data into subsets with similar values for the target variable
- The CART Algorithm grows the tree as follows
  - Conduct an exhaustive search of all available predictor variables and all possible splitting values
  - Choose the optimal split according to "goodness" of candidate split

CART – Gini Index

- The equation for the Gini Index

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

- Will help us determine the impurity at a decision tree node

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous,	**Homogeneous,**
High degree of impurity	**Low degree of impurity**

CART – Gini Index

- Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Example Data

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Gini – Outlook Predictor

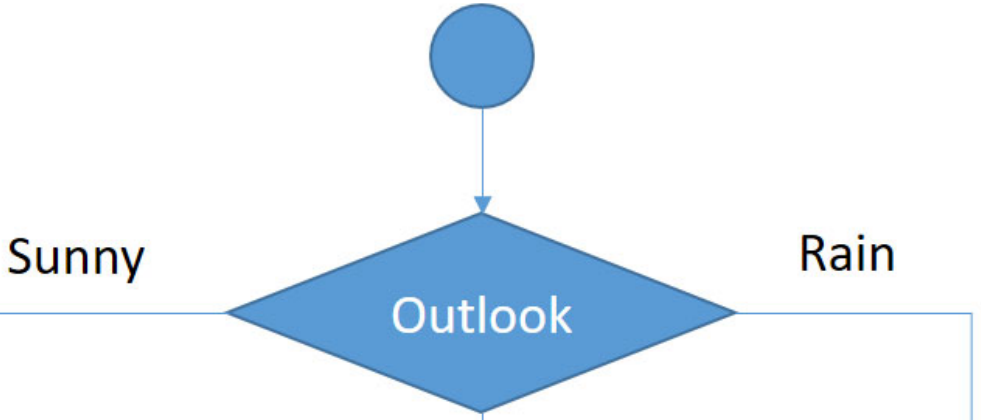
- Gini(Outlook=Sunny) = 1 – $(2/5)^2$ – $(3/5)^2$ = 1 – 0.16 – 0.36 = 0.48
- Gini(Outlook=Overcast) = 1 – $(4/4)^2$ – $(0/4)^2$ = 0
- Gini(Outlook=Rain) = 1 – $(3/5)^2$ – $(2/5)^2$ = 1 – 0.36 – 0.16 = 0.48
- Calculate weighted sum of gini indexes for outlook predictor
  - Gini(Outlook) = (5/14) x 0.48 + (4/14) x 0 + (5/14) x 0.48 = 0.171 + 0 + 0.171 = 0.342

Outlook	Yes	No	Number of instances
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5

Gini – All Predictors

- The Gini Index value for Outlook is the lowest, so, choose it

Feature	Gini index
Outlook	0.342
Temperature	0.439
Humidity	0.367
Wind	0.428

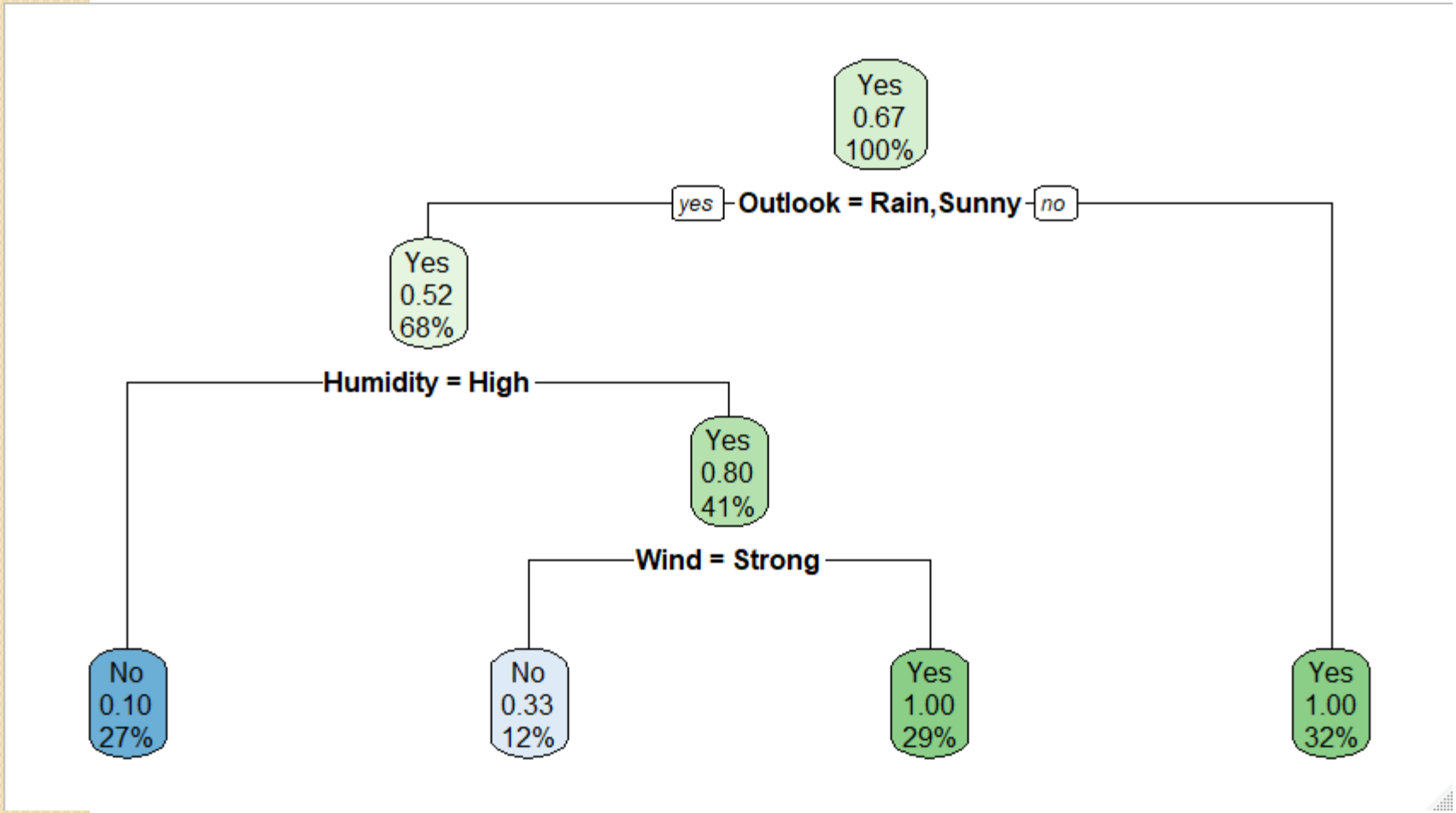


Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Tree built in R using CART (rpart)



Classification and Regression Trees (*cont'd*)

- Classification Error Rate
  - CART recursively builds tree by following procedure outlined previously
  - After tree "fully grown", not all leaf nodes necessarily homogenous. Some are diverse leaf nodes
  - Leads to level of classification error
  - Consider records in the table below. Cannot be further partitioned, and classified as "Bad"
  - Probability that record in leaf node classified correctly (as "Bad") is 3/5 = 0.60 = 60%
  - Classification error rate for leaf node is 0.40 = 40%. 2/5 "Good" records classified incorrectly ("Bad")
  - CART calculates classification error rate for the tree this way
  - Each leaf node error rate is weighted proportionally to the number of records it contains

Customer	Savings	Assets	Income	Credit Risk
004	High	Low	<=$30K	Good
009	High	Low	<=$30K	Good
027	High	Low	<=$30K	Bad
031	High	Low	<=$30K	Bad
104	High	Low	<=$30K	Bad

Classification and Regression Trees (*cont'd*)

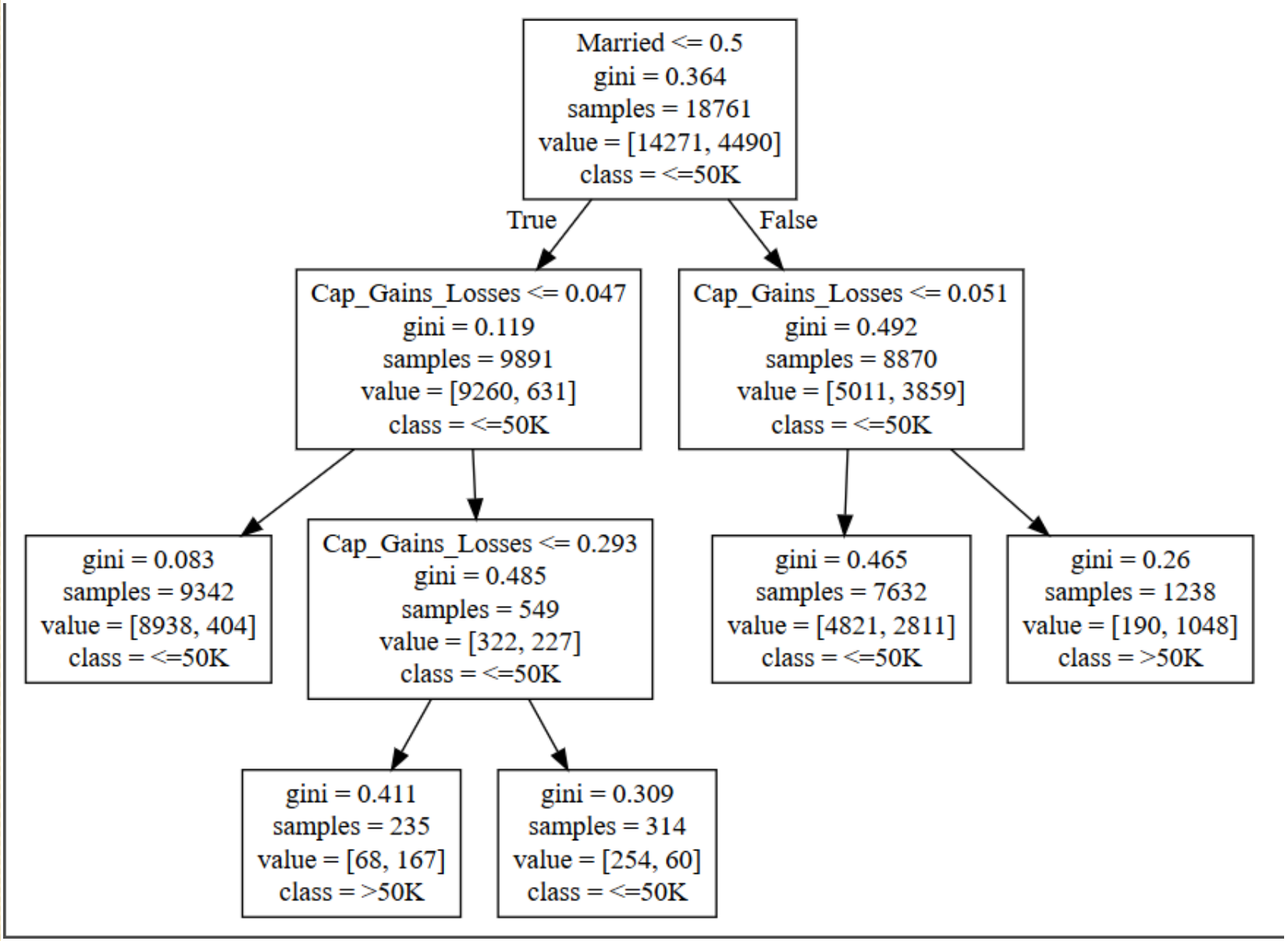
- Pruning
  - As the tree grows, each subset of records to partition becomes smaller and less representative
  - Fully grown tree has lowest classification error rate
  - However, resulting model overfits the training set
  - CART avoids memorizing the training set by pruning nodes and branches that reduce generalizability
  - CART algorithm finds adjusted classification error rate that penalizes the tree for having too many leaf nodes (too much complexity)

CART with Python

- import pandas as pd
- import numpy as np
- import statsmodels.tools.tools as stattools
- from sklearn.tree import DecisionTreeClassifier, export_graphviz
- adult_tr = pd.read_csv('adult_ch6_training')
- y = adult_tr[['Income']]
- #CART using sklearn needs categorical variables converted to dummy variable form
- #make the Marital Status column an array
- mar_np = np.array(adult_tr['Marital status'])
- #create a matrix of dummy variables for each value in Marital Status
- #mar_cat contains 5 columns, one for each category
- #each row in mar_cat represents a record in adult_tr
- #each row will have a value in 1 in the column which matches the categorical value
- #mar_cat_dict shows which column represents a category value
- (mar_cat, mar_cat_dict) = stattools.categorical(mar_np, drop=True, dictnames=True)
- #add dummy variables back by making mar_cat a dataframe
- mar_cat_pd = pd.DataFrame(mar_cat)

CART with Python

- X = pd.concat((adult_tr[['Cap_Gains_Losses']], mar_cat_pd), axis = 1)
- #set a variable to the names of the columns of the predictor variables
- X_names = ["Cap_Gains_Losses", "Divorced", "Married", "Never-married",\
- "Separated", "Widowed"]
- y_names = ["<=50K", ">50K"]
- #CART, gini is for the gini splitting criterion (also called gini index)
- #do not allow the tree to get larger than 5 nodes
- #fit gives the predictor and then target variables
- #the decision tree is stored in cart01
- cart01 = DecisionTreeClassifier(criterion="gini",max_leaf_nodes=5).fit(X,y)
- #to examine the tree structure
- #go to http://www.webgraphviz.com/ to visualize .dot file
- #can install graphviz as well to convert .dot file to .png file
- export_graphviz(cart01, out_file = "cart01.dot", feature_names=X_names,\
- class_names=y_names)

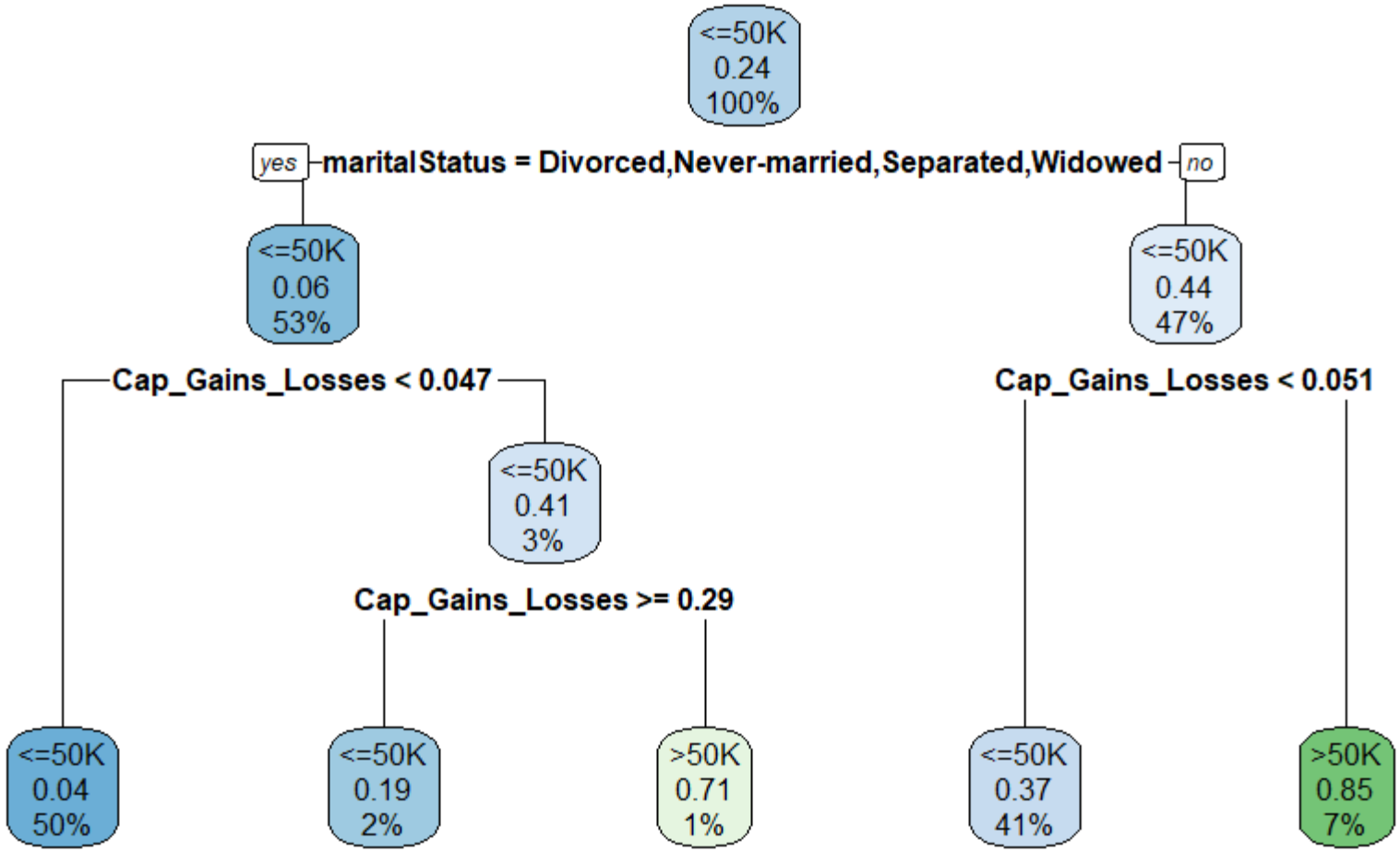

Married <= 0.5
gini = 0.364
samples = 18761
value = [14271, 4490]
class = <=50K
True
False
Cap_Gains_Losses <= 0.047
gini = 0.119
samples = 9891
value = [9260, 631]
class = <=50K
Cap_Gains_Losses <= 0.051
gini = 0.492
samples = 8870
value = [5011, 3859]
class = <=50K
gini = 0.083
samples = 9342
value = [8938, 404]
class = <=50K
Cap_Gains_Losses <= 0.293
gini = 0.485
samples = 549
value = [322, 227]
class = <=50K
gini = 0.465
samples = 7632
value = [4821, 2811]
class = <=50K
gini = 0.26
samples = 1238
value = [190, 1048]
class = >50K
gini = 0.411
samples = 235
value = [68, 167]
class = >50K
gini = 0.309
samples = 314
value = [254, 60]
class = <=50K

CART with Python

- #use the predict command to check the tree's predictions with the training data set
- predIncomeCART = cart01.predict(X)
- #predIncomeCART is a numpy array so make it a dataframe
- predIncomeCART_pd = pd.DataFrame(predIncomeCART)
- pred_CART_y = pd.concat((y,predIncomeCART_pd), axis = 1)
- #create a mask showing where the values differ
- mask = y.where(y.values==predIncomeCART_pd.values).notna()
- mask['Income'].value_counts()

CART with R

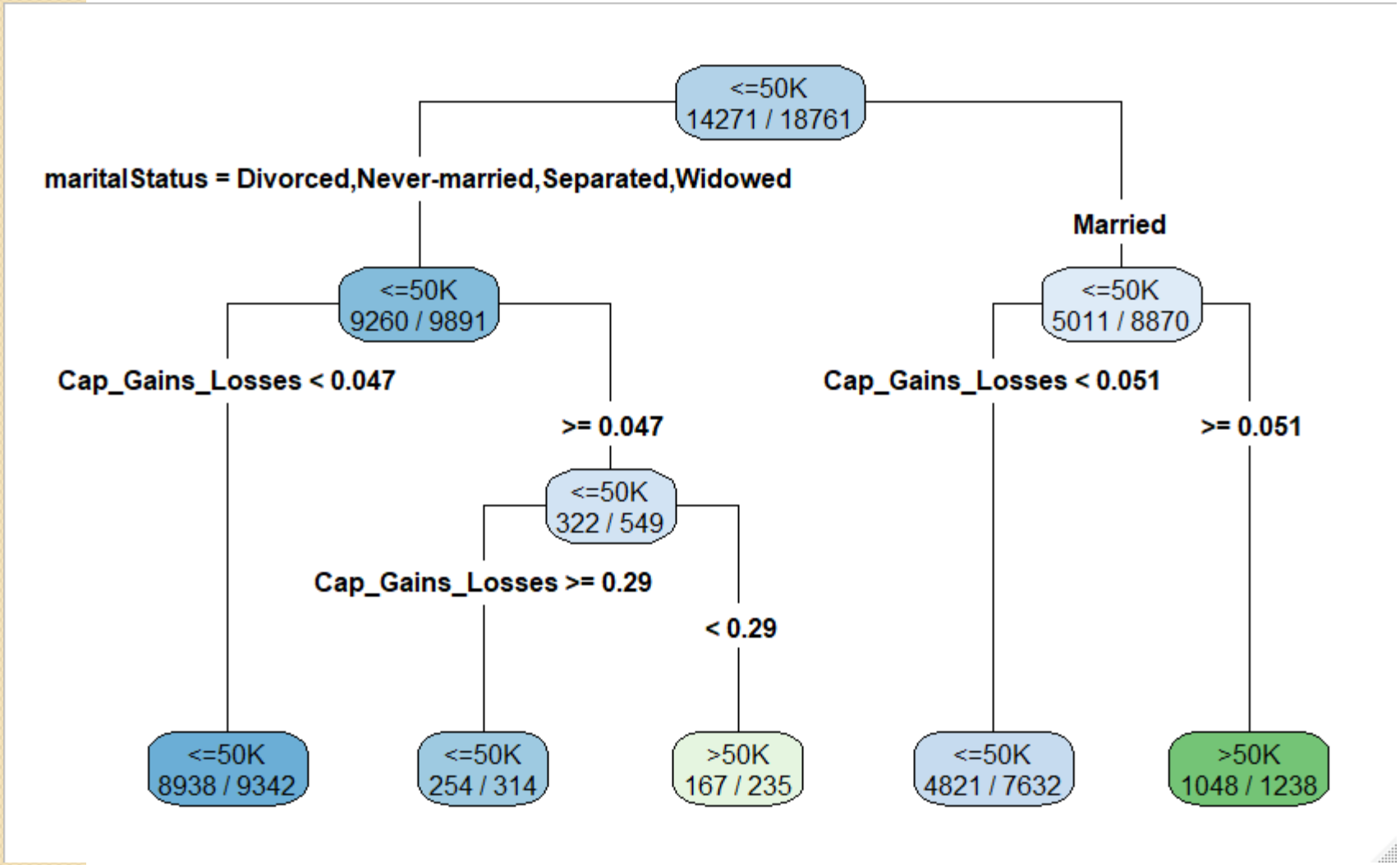
- adult_tr <- read.csv("adult_ch6_training")
- #rename Marital.status column using the colnames function
- colnames(adult_tr)[1] <- "maritalStatus"
- #change the categorical variables to factors
- adult_tr$Income <- factor(adult_tr$Income)
- adult_tr$maritalStatus <- factor(adult_tr$maritalStatus)
- #install and load the rpart and rpart.plot packages (only install once)
- install.packages(c("rpart","rpart.plot"))
- library(rpart); library(rpart.plot)
- #build the CART model using a "class" (classification) model
- cart01 <- rpart(formula = Income ~ maritalStatus + Cap_Gains_Losses, data = adult_tr, method = "class")
- #plot the CART model
- rpart.plot(cart01)


<=50K
0.24
100%
yes
maritalStatus = Divorced,Never-married,Separated,Widowed
no
<=50K
0.06
53%
<=50K
0.44
47%
Cap_Gains_Losses < 0.047
Cap_Gains_Losses < 0.051
<=50K
0.41
3%
Cap_Gains_Losses >= 0.29
<=50K
0.04
50%
<=50K
0.19
2%
>50K
0.71
1%
<=50K
0.37
41%
>50K
0.85
7%

CART with R

- #utilize type=4 to label each branch with its specific value instead of yes/no
- #utilize extra=2 to add the correct classification proportion to each node
- rpart.plot(cart01, type = 4, extra = 2)

- #get classifications using CART model for the training data set
- #build a predictor variable dataframe
- X = data.frame(maritalStatus = adult_tr$maritalStatus, Cap_Gains_Losses = adult_tr$Cap_Gains_Losses)
- #use the predict function
- predIncomeCART = predict(object = cart01, newdata = X, type = "class")
- predIncomeCART_df = data.frame(Income = predIncomeCART)
- missed <- adult_tr[which(adult_tr$Income != predIncomeCART_df$Income),]


<=50K
14271 / 18761
maritalStatus = Divorced,Never-married,Separated,Widowed
Married
<=50K
9260 / 9891
<=50K
5011 / 8870
Cap_Gains_Losses < 0.047
>= 0.047
Cap_Gains_Losses < 0.051
>= 0.051
<=50K
322 / 549
Cap_Gains_Losses >= 0.29
< 0.29
<=50K
8938 / 9342
<=50K
254 / 314
>50K
167 / 235
<=50K
4821 / 7632
>50K
1048 / 1238

C5.0 Algorithm

- C5.0 is an extension of the C4.5 algorithm developed by Quinlan
- C4.5 is an extension of the ID3 algorithm developed by Quinlan
- Similar to CART, C5.0 builds a tree by recursively visiting decision nodes and choosing the optimal spit, until no further splits are possible

- Key Differences Between CART and C5.0
  - Unlike CART, C5.0 is not limited to binary splits and produces a tree with a variable shape
  - C5.0 produces a branch for each categorical value which may result in "bushiness"
  - C5.0 uses a different algorithm to measure homogeneity occurring at leaf nodes

C5.0 Algorithm *(cont'd)*

- C5.0 uses information gain or entropy reduction to select the optimal split at each decision node
- In engineering, information would be analogous to a signal while entropy would be analogous to noise

- What is Entropy?
  - For an event with probability = p, the average amount of information in bits required to transmit the result is $-log_2(p)$
  - For example, toss a fair coin with $p = 0.5$
  - The result of the toss transmitted using $-log_2(0.5) = 1$ bit of information
  - *1* bit of information represents a result toss as 0 or 1
  - Corresponds to either "HEAD" or "TAIL"

C5.0 Algorithm *(cont'd)*

- Variable X can have k values with probabilities $p_1, p_2, \ldots, p_k$
- For variables with several outcomes, use the weighted sum of $-\log_2(p)$'s to transmit the result
- Entropy of X is defined using the formula:

$$H(X) = -\sum_j p_j \log_2(p_j)$$

- Represents the smallest number of bits, on average per symbol, needed to transmit a stream of symbols corresponding to the observed values of X

C5.0 Algorithm *(cont'd)*

- How Does C5.0 use Entropy?
  - Assume a candidate split *S*, partitions the data set *T* into subsets T_1, T_2, ..., T_k
  - Mean information requirement is calculated as the weighted sum of entropies associated with each subset *T*

$$H_S(T) = \sum_{i=1}^{k} P_i H_S(T_i)$$

  - *Pi* represents the proportion of records in subset T_i
- Information Gain
  - C5.0 uses Information Gain

$$\text{gain}(S) = H(T) - H_S(T)$$

  - Represents the increase in information by partitioning the training data *T* according to the candidate split *S*
  - For each candidate split, C5.0 chooses the split that has the maximum information gain, gain(S)

C5.0 Algorithm *(cont'd)*

Customer	Savings	Assets	Income ($1000s)	Credit Risk
1	Medium	High	75	Good
2	Low	Low	50	Bad
3	High	Medium	25	Bad
4	Medium	Medium	50	Good
5	Low	Medium	100	Good
6	High	High	25	Good
7	Low	Low	25	Bad
8	Medium	Medium	75	Good

- As with CART, consider all candidate splits for root node shown below

Candidate Split	Child Nodes		
1	Savings = Low	Savings = Medium	Savings = High
2	Assets = Low	Assets = Medium	Assets = High
3	Income <= $25,000		Income > $25,000
4	Income <= $50,000		Income > $50,000
5	Income <= $75,000		Income > $75,000

C5.0 Algorithm *(cont'd)*

- Calculate Entropy of training set before splitting, where 5/8 records classified "Good" and 3/8 "Bad"

$$H(T) = -\sum_j p_j \log_2(p_j) = -\frac{5}{8}\log_2(\frac{5}{8}) - \frac{3}{8}\log_2(\frac{3}{8}) = 0.9544 \text{ bits}$$

- Compare each candidate split against H(T) = 0.9544, to determine which split maximizes information gain
- Candidate 1
- Split on *Savings*, "High" = 2 records, "Medium" = 3 records, and "Low" = 3 records:

$$P_{High} = \frac{2}{8}, \quad P_{Medium} = \frac{3}{8}, \quad P_{High} = \frac{3}{8}$$

C5.0 Algorithm *(cont'd)*

- Of records *Savings* = "High", 1 is "Good" and 1 is "Bad", so *P* = 0.5 of choosing the "Good" record
- Where *Savings* = "Medium", 3 are "Good", so *P* = 1.0 choosing "Good"
- Of records *Savings* = "Low", 1 is "Good" and 2 are "Bad", so *P* = 0.33 choosing "Good"
- Entropy of 3 branches, "High", "Medium", and "Low", is:

$$H(High) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$$

$$H(Medium) = -\frac{3}{3}\log_2(\frac{3}{3}) - \frac{0}{3}\log_2(\frac{0}{3}) = 0$$

$$H(Low) = -\frac{1}{3}\log_2(\frac{1}{3}) - \frac{2}{3}\log_2(\frac{2}{3}) = 0.9183$$

C5.0 Algorithm *(cont'd)*

- Combining the Entropy for the three branches, along with the corresponding proportion P_i:

$$H_S(T) = \sum_{i=1}^{k} P_i H_S(T_i) = \frac{2}{8}(1) + \frac{3}{8}(0) + \frac{3}{8}(0.9183) = 0.5944 \text{ bits}$$

- Information Gain represented by split on *Savings* attribute is $H(T) - H_{Savings}(T) = 0.9544 - 0.5944 = 0.36$ bits
- How is measure interpreted?
  - Before split, H(T) = 0.9544 bits, on average. Takes 0.9544 bits of information to transmit credit risk associated with 8 records.
  - Splitting on Savings results in HSavings(T) = 0.5944. Now, on average, fewer bits of information required to transmit credit risk associated with 8 records
  - Reduction in Entropy is Information Gain, 0.9544 – 0.5944 = 0.36 bits of information gained by splitting on Savings
  - C5.0 chooses attribute split with highest Information Gain as the optimal split
  - Information Gain for other 4 candidate splits calculated similarly

C5.0 Algorithm *(cont'd)*

- Information Gain for 5 candidate splits occurring at root node summarized below
- Candidate split 2 has highest Information Gain = 0.5487 bits, and chosen for the initial split

Candidate Split	Child Nodes	Information Gain (Entropy Reduction)
1	Savings = Low Savings = Medium Savings = High	0.36 bits
2	Assets = Low Assets = Medium Assets = High	0.5487 bits
3	Income <= $25,000 Income > $25,000	0.1588 bits
4	Income <= $50,000 Income > $50,000	0.3475 bits
5	Income <= $75,000 Income > $75,000	0.0923 bits

C5.0 Algorithm *(cont'd)*

- Initial split results in two terminal leaf nodes and one second-level decision node
- Records with *Assets* = "Low" and *Assets* = "High" have same target class value, "Bad" and "Good", respectively

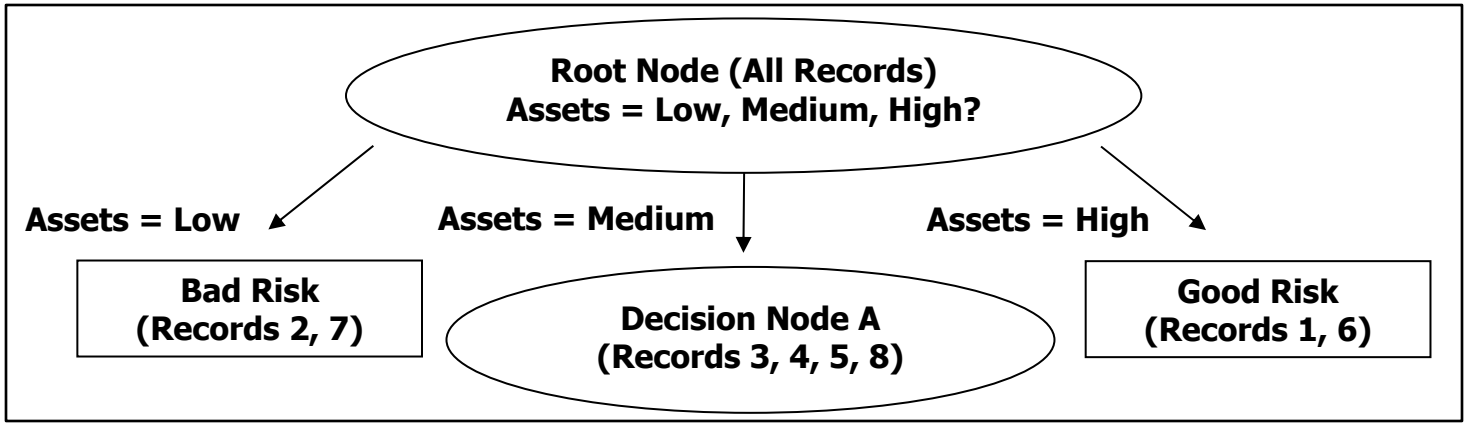


Figure 8.6

- C5.0 iterates at Decision Node A, choosing optimal split from list of four possible candidate splits

C5.0 Algorithm *(cont'd)*

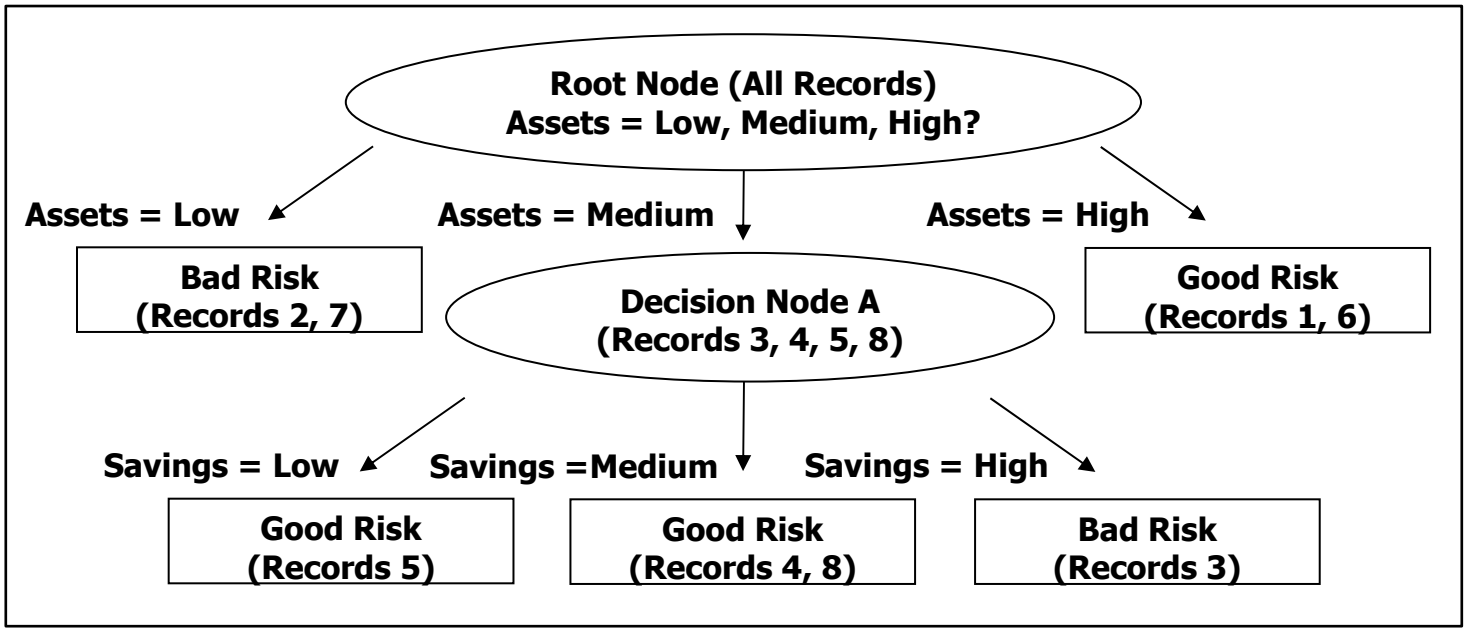


Figure 8.7

Decision Rules

- Decision Trees produce interpretable output in human-readable form
- Decision Rules constructed directly from Decision Tree output, traversing path from root node to a given leaf node
- Decision Rules have form IF antecedent THEN consequent
- Antecedent consists of attributes values from branches of given path
- Consequent is classification of records contained in particular leaf node, corresponding to path

Decision Rules *(cont'd)*

- Recall the full decision tree produced by C5.0. Table below shows Decision Rules associated with tree

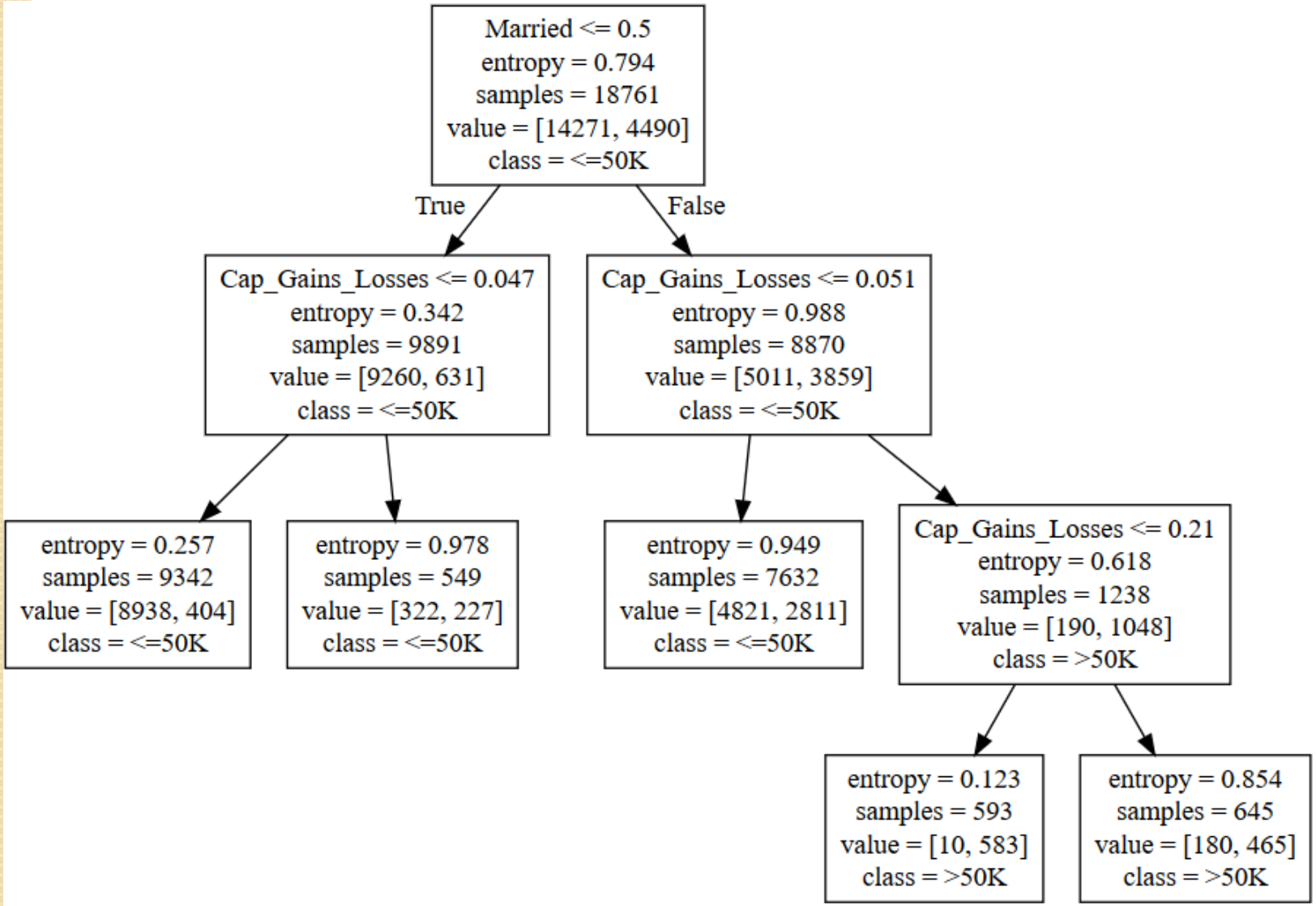
Table 8.10

Antecedent	Consequent	Support	Confidence
If assets = low	then bad credit risk.	2/8	1.00
If assets = high	then good credit risk.	2/8	1.00
If assets = medium and savings = low	then good credit risk.	1/8	1.00
If assets = medium and savings = medium	then good credit risk.	2/8	1.00
If assets = medium and savings = high	then bad credit risk.	1/8	1.00

- Support of decision rule shows proportion of records in training set resting in leaf node
- Confidence is percentage of records in leaf node, for which decision rule is true
- Although confidence levels reported = 100%, results not typical of real-world examples

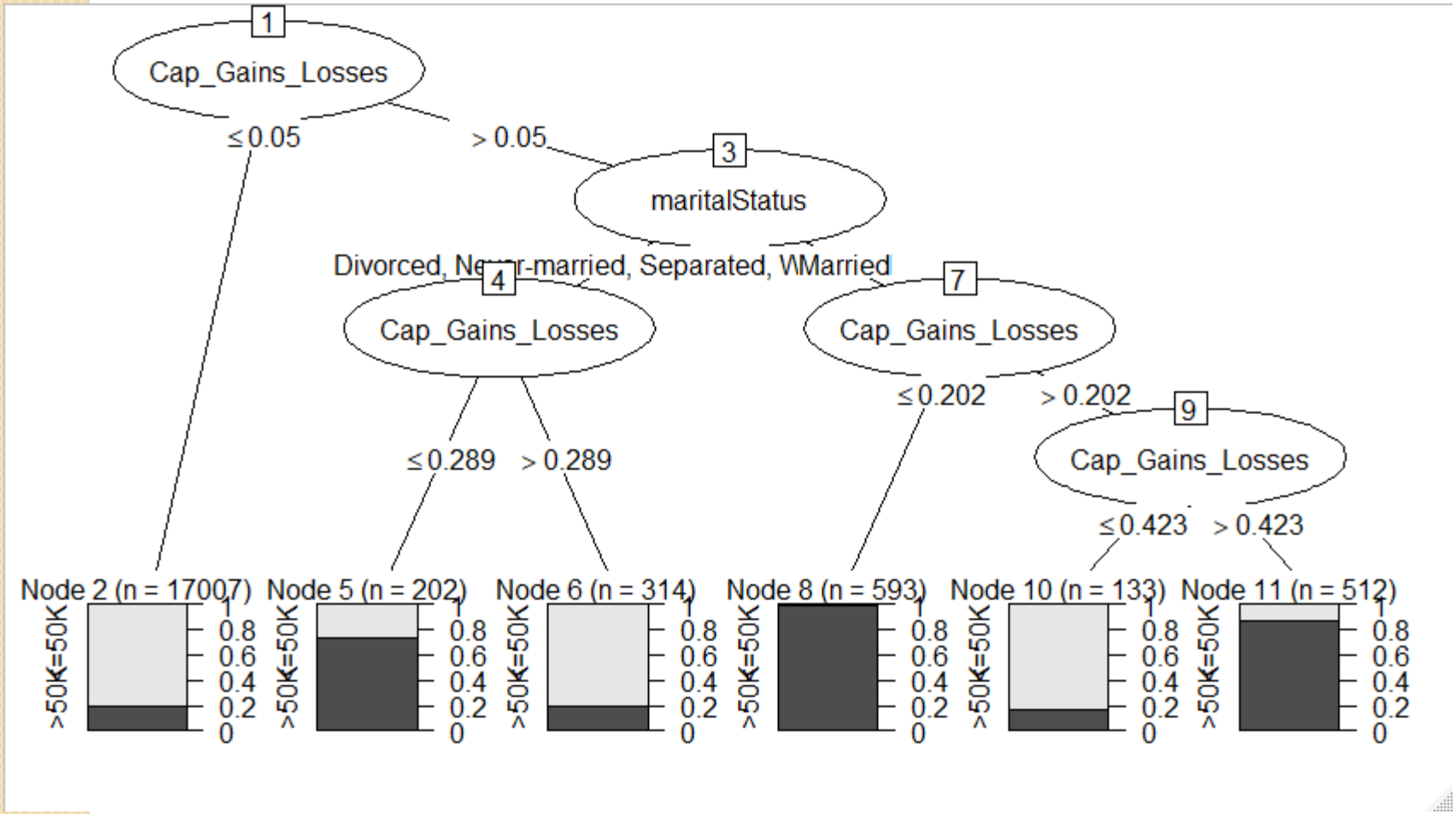
C5.0 with Python

- #C5.0 is not implemented directly in Python
- #We will use the same decision tree function with criterion entropy
- c50_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=5).fit(X,y)
- export_graphviz(c50_01, out_file = "c50_01.dot", feature_names=X_names,\
- class_names=y_names)
- predIncomeC50 = c50_01.predict(X)


Married <= 0.5
entropy = 0.794
samples = 18761
value = [14271, 4490]
class = <=50K
True
False
Cap_Gains_Losses <= 0.047
entropy = 0.342
samples = 9891
value = [9260, 631]
class = <=50K
Cap_Gains_Losses <= 0.051
entropy = 0.988
samples = 8870
value = [5011, 3859]
class = <=50K
entropy = 0.257
samples = 9342
value = [8938, 404]
class = <=50K
entropy = 0.978
samples = 549
value = [322, 227]
class = <=50K
entropy = 0.949
samples = 7632
value = [4821, 2811]
class = <=50K
Cap_Gains_Losses <= 0.21
entropy = 0.618
samples = 1238
value = [190, 1048]
class = >50K
entropy = 0.123
samples = 593
value = [10, 583]
class = >50K
entropy = 0.854
samples = 645
value = [180, 465]
class = >50K

C5.0 with R

- #install and load the C50 package (only install once)
- install.packages("C50")
- library(C50)
- #build the C5.0 model using specifying at least 75 cases in a leaf node
- C5 <- C5.0(formula = Income ~ maritalStatus + Cap_Gains_Losses, data = adult_tr, control = C5.0Control(minCases=75))
- #plot the C5.0 model
- plot(C5)
- predict(object=C5,newdata=X)


1
Cap_Gains_Losses
≤ 0.05
> 0.05
3
maritalStatus
Divorced, Never-married, Separated, Widowed
Married
4
Cap_Gains_Losses
7
Cap_Gains_Losses
≤ 0.289
> 0.289
≤ 0.202
> 0.202
9
Cap_Gains_Losses
≤ 0.423
> 0.423
Node 2 (n = 17007)
Node 5 (n = 202)
Node 6 (n = 314)
Node 8 (n = 593)
Node 10 (n = 133)
Node 11 (n = 512)
>50K
<=50K
1
0.8
0.6
0.4
0.2
0

Random Forests (Ensemble Method)

- CART and C5.0 both produce a single decision tree
- Random Forests build a series of decision trees and combine the trees' classifications of each record into one classification
  - Ensemble methods take multiple models' output into account in order to arrive at a single answer
- Each tree is built using a different randomly built training set (with replacement) and for each node, a subset of predictor variables is considered

Random Forest with Python

- #Random Forests
- from sklearn.ensemble import RandomForestClassifier
- import numpy as np
- #Random Forest requires a target variable formatted as a 1-dimensional array
- #numpy's ravel command creates a 1-dimensional array
- rfy = np.ravel(y)
- #build the random forest with 100 trees
- rf01 = RandomForestClassifier(n_estimators=100,criterion="gini").fit(X,rfy)
- predIncomerandom = rf01.predict(X)

Random Forest with R

- #install and load the C50 package (only install once)
- install.packages("C50")
- library(C50)
- #build the C5.0 model using specifying at least 75 cases in a leaf node
- C5 <- C5.0(formula = Income ~ maritalStatus + Cap_Gains_Losses, data = adult_tr, control = C5.0Control(minCases=75))
- #plot the C5.0 model
- plot(C5)
- predict(object=C5,newdata=X)