Chapter 8
Decision Trees

8.1 The Problem

Linear and logistic regressions make predictions about numbers, but we also need algorithms to classify instances of data in a certain class, i.e., to label the instance as belonging to a class. The decision tree is our first approach to solve classification problems. However, decision trees can perform regression too, hence their name classification and regression trees (CART). The random forests that we will encounter in a later chapter are powerful variations of CART.

A CART is represented by a binary tree whose root is on top, and at each level, each node (including the root node) receives a data input that is examined. If the value of the feature is below a certain value, the left branch of the binary tree is followed; otherwise, the right branch is followed [1]. At the bottom level, we find the leaf nodes, or terminal nodes, which represent outcome values.

When we take an instance of data, we use the tree to compare the instance's attribute values to the root and decide whether the instance belongs to one subbranch or the other. The process is continued until we reach a leaf that represents a class.

Figure 8.1 represents a decision tree that mimics the underwriting process of a mortgage application. Each mortgage application contains the number of dependents, loan-to-value ratio, marital status, payment-to-income ratio, interest rate, years at the current address, and years in a current job.

8.2 A Practical Example

The Ionosphere dataset that was introduced in the previous chapter was collected in Goose Bay, Labrador, and represents 34 input features of continuous values and one binary output that classifies the measurement as either "good" (i.e., value *g*) or "bad"

C. El Morr et al., *Machine Learning for Practical Decision Making*, International Series in Operations Research & Management Science 334,
https://doi.org/10.1007/978-3-031-16990-8_8

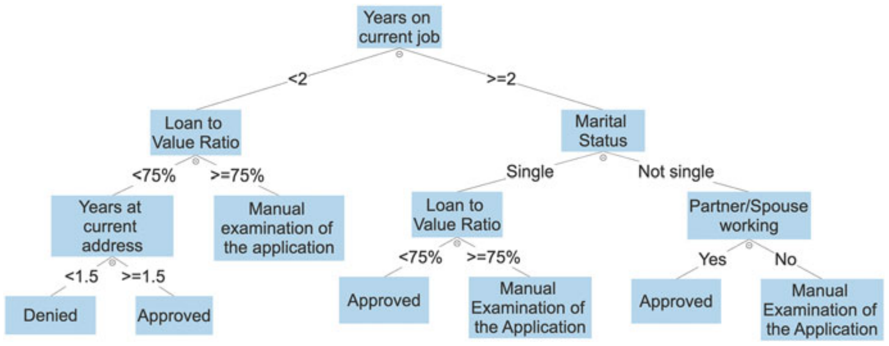


Fig. 8.1 An example of a decision tree to represent decision-making for a mortgage application (adapted from Maimon and Rokach [2])

(i.e., value *b*) [3]. In the previous chapter, we could build a logistic regression model with nearly 89% accuracy.

However, given the continuous nature of the input features, we can apply a decision tree to the same dataset to make a classification decision.

Open the dataset and choose the REPTree algorithm from the Trees classifiers (Fig. 8.2).

REPTree is the name of the CART algorithm in Weka. Keep the defaults for all REPTree parameters and run the algorithm (Fig. 8.3).

In the detailed accuracy table, we can notice the precision of almost 89.5%, which constitutes a slight improvement compared to the logistic regression model. Right-click on the Result List and click on Visualize Tree (Fig. 8.4).

Weka displays the decision tree for the Ionosphere dataset (Fig. 8.5). The feature a05 is in the root node, where a decision is made based on the threshold 0.02. The leaf nodes are radar detection outcomes: *b* (bad) or *g* (good). Only four features contributed to the decision tree: a03, a05, a22, and a27.

Using this decision tree, we can predict for any measurements made in the future if the radar detection will be bad or good based on the values of only four features.

8.3 The Algorithm

8.3.1 Tree Basics

In Fig. 8.6, we can recognize many elements of a decision tree.

The *root node* is the top (first) node of a decision tree, and a *leaf node* is an end node. At the root, all the dataset is present and is divided into homogeneous subsets based on certain decision rules. The leaves are nodes that do not split; they represent the outcome variable.

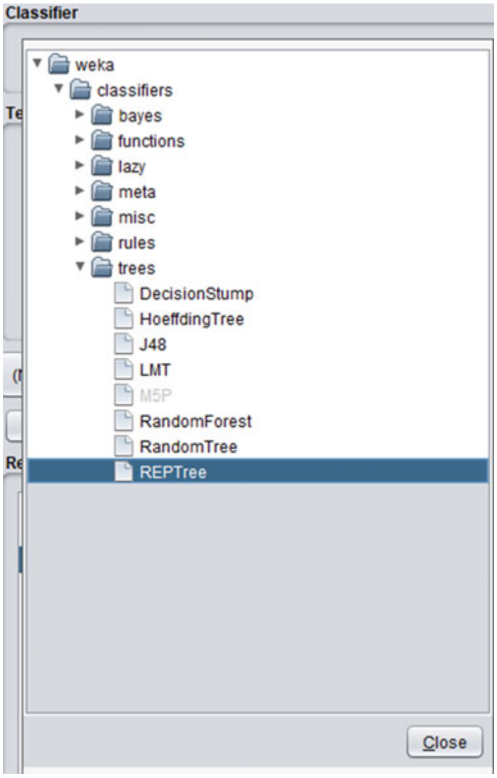
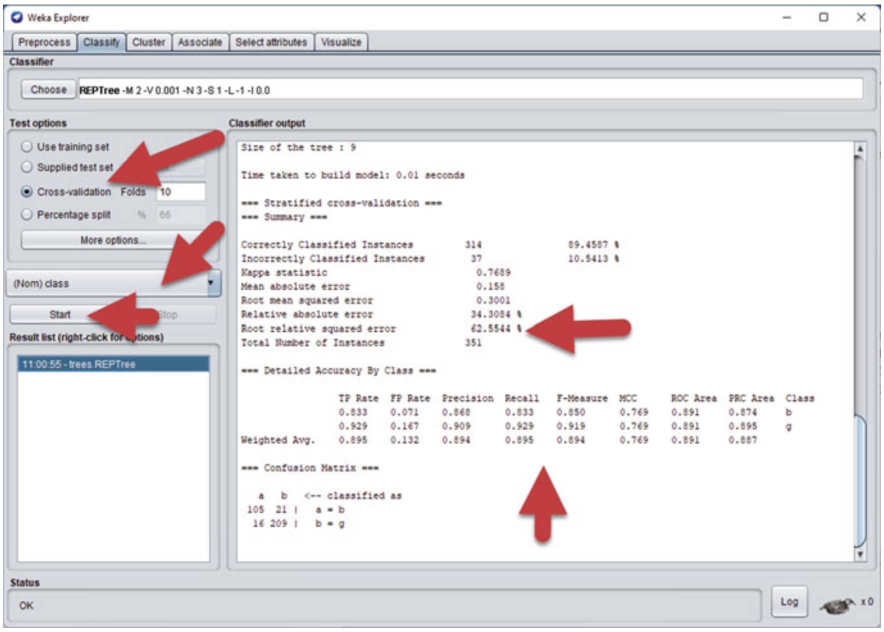


Fig. 8.2 REPTree chosen from the Trees classifiers in Weka

Every internal node between the root and the leaves is a *decision node* that splits the data based on splitting rules. Every node in the tree is a *parent node* for any node directly below it, which is called a *child node* to the parent. A subset of nodes and associated leaves is called a *subtree* or *branch*. While splitting is the process of dividing the data at a certain node into two or more sub-nodes, *pruning* is the process of deleting sub-nodes of a decision node and redistributing data associated with it; we will see more about pruning and its necessity below.

Trees can apply to classification problems when the outcome is a categorical variable, as we have seen in the ionosphere example; other examples include predicting if a patient will be subject to readmission after discharge, or if a person will get vaccinated for COVID-19, or will get her loan approved, or will make it to the Olympic finals. Also, trees can apply to regression problems where the outcome is a numeric (i.e., continuous) variable; for example, based on several features, we can try to predict a future house price, the infection rate in a population, the area of land that will be subject to desertification, or the number of migrants crossing the Mediterranean Sea.

The decision to split the data at a certain node, including the root, is not straightforward; the final tree and associated decisions (i.e., regression, classification) change drastically if we split based on one feature or another. We need a



REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

Cross-validation Folds 10

Percentage split % 66

```
Size of the tree : 9

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         314               89.4587 %
Incorrectly Classified Instances        37               10.5413 %
Kappa statistic                          0.7689
Mean absolute error                      0.158
Root mean squared error                  0.3001
Relative absolute error                 34.3084 %
Root relative squared error             62.5544 %
Total Number of Instances              351

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.833    0.071    0.868      0.833    0.850      0.769    0.891     0.874     b
                 0.929    0.167    0.909      0.929    0.919      0.769    0.891     0.895     g
Weighted Avg.    0.895    0.132    0.894      0.895    0.894      0.769    0.891     0.887

=== Confusion Matrix ===

   a   b   <-- classified as
 105  21 |   a = b
  16 209 |   b = g
```

Fig. 8.3 REPTree execution in Weka

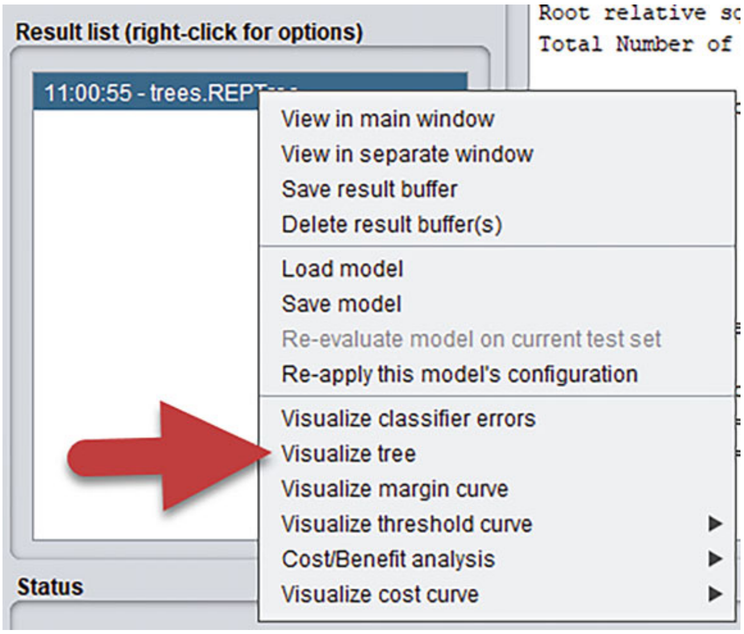


Fig. 8.4 Visualize tree option

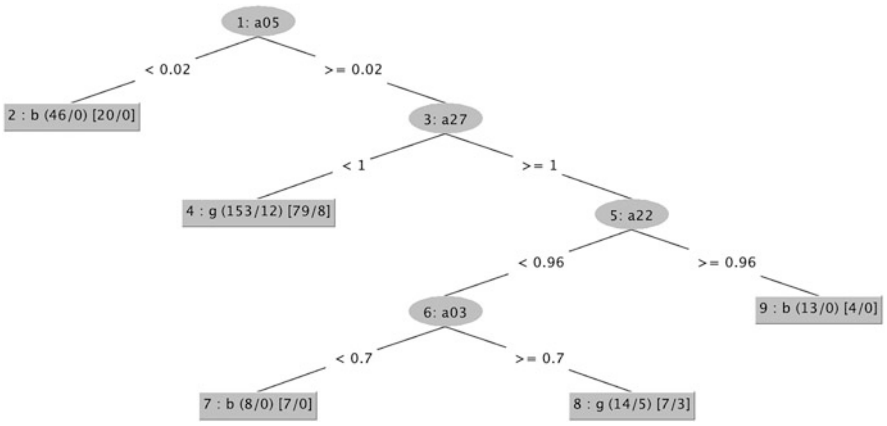


Fig. 8.5 A decision tree for the ionosphere problem

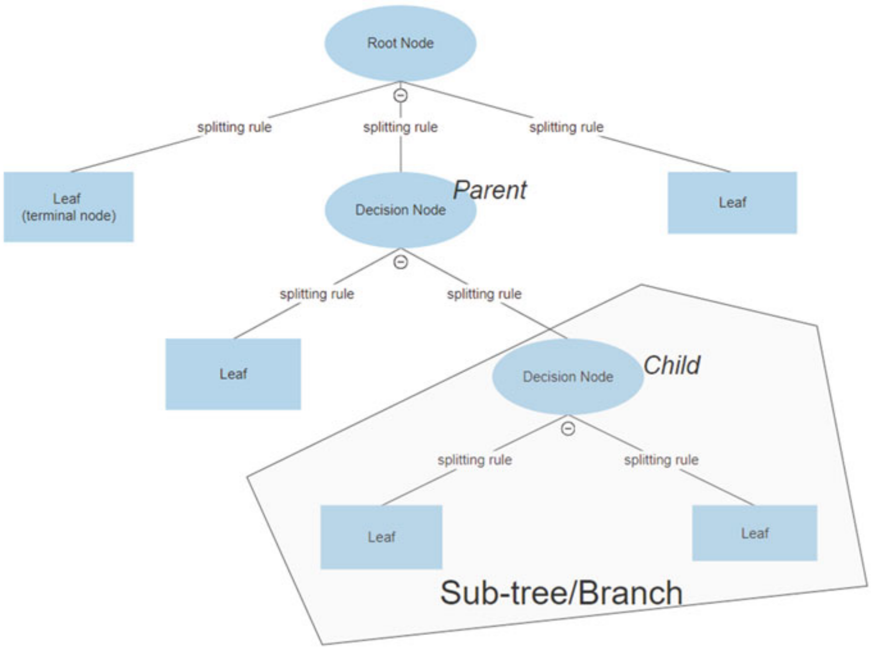


Fig. 8.6 Decision tree elements

decision criterion to decide which feature to choose at a certain node to base our splitting upon. How to choose a decision criterion? Each time we split the data into two or more subsets, we are aiming at homogenizing the subsets; therefore, researchers have invented functions to measure *homogeneity* or *purity* at a node with respect to the outcome variable and based the decision criteria of those purity measurements: at each node, we choose to split the data based on the feature that maximizes data homogeneity.

Researchers have invented many algorithms to create trees and select the feature for splitting, such as:

1. ID3: Extension of a previous version, D3 (ID stands for "iterative dichotomizer")
2. C4.5: Successor to ID3
3. CART: Classification and regression tree
4. CHAID: Chi-square automatic interaction detection. It executes multilevel splits for classification trees.
5. MARS: Multivariate adaptive regression splines

A *greedy* algorithm like ID3 acts as follows: at each node, the decision tree algorithm will use the available dataset at that node, calculate all possible splits using every possible feature, and choose the feature that maximizes data homogeneity to do the split. The split is done, and the dataset is distributed among the sub-nodes. The same process continues at each sub-node using the remaining features until no more splitting can be done.

There are many data homogeneity functions, including the following:

1. Entropy
2. Information gain (IG)
3. Gini index
4. Information gain ratio

Entropy measures randomness (i.e., think of it as the opposite of homogeneity), so at each node, we choose to split the feature that minimizes entropy. Information gain is a measure of homogeneity; hence, at each node, we choose to split the feature that minimizes information gain. Like information gain, the Gini index is another measure of homogeneity. The information gain ratio is a correction included for the information gain; it is a measurement that allows us to avoid splitting based on an attribute with high information gain but a large number of distinct values [4], such as a credit card number. A feature like customers' credit card numbers presents high information gain, but we should not split based on this feature, as it will not be helpful to predict anything about a future customer, who necessarily will have a different credit card number (i.e., the variation of distinct values of credit card numbers is extremely high). The information gain ratio for such features will be low, and the splitting criterion will not make the split based on ratios below the average IG. Information gain is a criterion used for categorical features, while the Gini index is used for continuous attributes. Below, we will see formulas and examples for entropy and information gain.

Entropy: We seek to minimize entropy because it is a measurement of non-homogeneity; the higher the entropy, the worse the solution for splitting the data will be. Entropy is calculated in the following way:

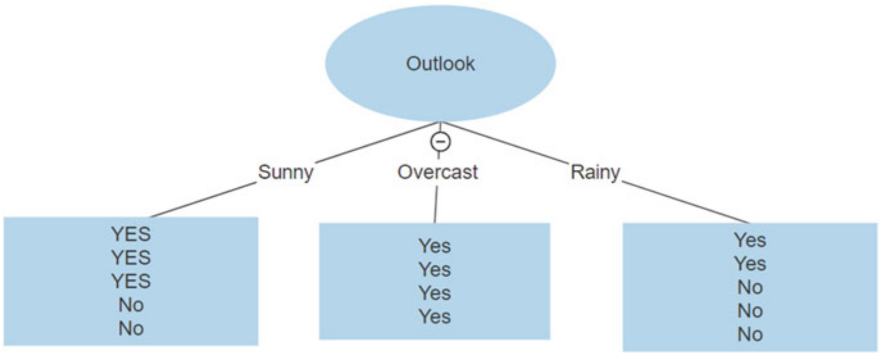


Fig. 8.7 A decision tree example showing the decision to play outdoors in relation to weather outlook

$$E(\mathrm{S}) = \sum\nolimits_{n=1}^{c} (-p_i \log_2(p_i))$$

S is the current state (e.g., current node), and p_i is the percentage of class *i* in a node of state *S*, or the probability of an event *i* of state *S*. Suppose we have a set of 16 instances at a node in relation to a feature called Humidity with two values, "High" and "Normal," where 10 instances have a value of "High" for Humidity and five have a value of "Normal." The entropy at this node in this status is calculated as

$$\text{Entropy (Humidity)} = \text{Entropy } (5, 10) = p_{\text{Normal}} \times \log_2(p_{\text{Normal}}) + p_{\text{High}} \times \log_2(p_{\text{High}})$$

$$\text{Entropy} = -\frac{5}{15} \times \log_2\left(\frac{5}{15}\right) - \frac{10}{15} \times \log_2\left(\frac{10}{15}\right)$$

$$\text{Entropy} = -0.34 \times (-1.585) - 0.67 \times (-0.585)$$

$$\text{Entropy} = 0.931$$

For two features, we will illustrate the use of entropy with more than one feature through an example (Fig. 8.7).

Consider the tree shown in Fig. 8.7.

The entropy for playing outdoors given the different weather outlooks is computed as follows:

$$E(\text{Playing, Outlook}) = P(\text{sunny}) \times E(3\text{ Yes, }2\text{ No}) + P(\text{overcast}) \times E(4\text{ Yes, }0\text{ No}) + P(\text{rainy}) \times E(2\text{ Yes, }3\text{ No}) = \frac{5}{14} \times \left(-\frac{2}{5} \times \log_2\left(\frac{2}{5}\right) - \frac{3}{5} \times \log_2\left(\frac{3}{5}\right)\right) + \frac{4}{14} \times \left(-\frac{4}{4} \times \log_2\left(\frac{4}{4}\right) - \frac{0}{5} \times \log_2\left(\frac{0}{4}\right)\right) + \frac{5}{14} \times \left(-\frac{3}{5} \times \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \times \log_2\left(\frac{2}{5}\right)\right) = \frac{5}{14} \times 0.971 + 0 + \frac{5}{14} \times 0.971 = 0.693$$

Information gain: As a measure of homogeneity, information gain (IG) is opposite to entropy; the higher the information gain, the lower the entropy. Information gain computes the difference between entropy before a split and average entropy after the split. Information gain is used by ID3.

The information gain resulting from splitting the 14 instances of the dataset in Fig. 8.7 into “Sunny” is calculated by

$$E(\text{Playing outdoors}) - E(\text{Playing, Outlook}) = E\left(\frac{9}{14}, \frac{5}{14}\right) - 0.693 = -\frac{9}{14} \times \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \times \log_2\left(\frac{5}{14}\right) - 0.693 = 0.940 - 0.693 = 0.247$$

We will see more about entropy and information gained below.

Gini index: The Gini index provides an indication of purity and is computed as follows:

$$G = \sum_{k=1}^{n} p_k \times (1 - p_k)$$

p_k is the proportion of training instances with class k in the leaf of interest, or the rectangle of interest when we look at scatter graphs (see below).

Sum squared error: When using trees for regression, we choose the split that will minimize the sum squared error across all training samples. The sum squared error is computed as follows:

$$S = \sum_{i=1}^{n} (\text{outcome}_i - \text{prediction}_i)$$

where n is the number of instances in question.

8.3.2 Training Decision Trees

Consider that the training dataset is *S*, the input features are I, and the outcome is *O*. The *split criterion* is the method used to decide if an instance should go to the left or the right of a node. The *stop criterion* is a condition that, if met, will stop the development of the tree.

The algorithm to create the tree can be illustrated using the following example [2]. Suppose we want to develop a smart model to filter spam emails. In real life, we will need many features to create such a model; however, for our illustration, we will suppose that we will build the model based only on two features: the length of the message and the number of new recipients of the email. Below is a scatter graph representing the dataset that we will use to train the decision tree (Fig. 8.8).

We start with one feature and try to divide the dataset in a manner to minimize the cost (the number of classification errors). Figure 8.9 is the result of using the New Recipients feature for classification (Fig. 8.9). Figure 8.10 is the result of using the Email Length feature (Fig. 8.10). Both figures show one single-node decision tree, called a decision stump.

If we use the New Recipients feature, we will end up with nine classification errors, while if we use the Email Length feature, we will end up with nine errors. Obviously, using the Email Length feature will incur less cost (fewer errors in classification); hence, we will use it in the next steps.

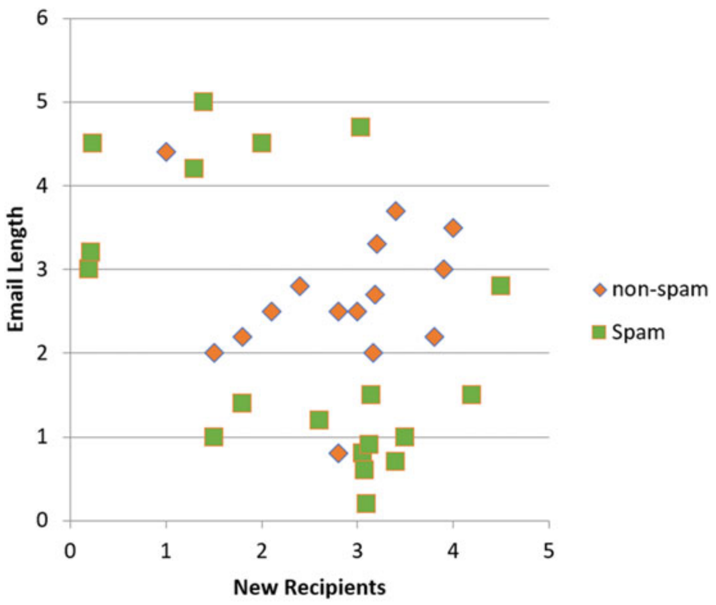
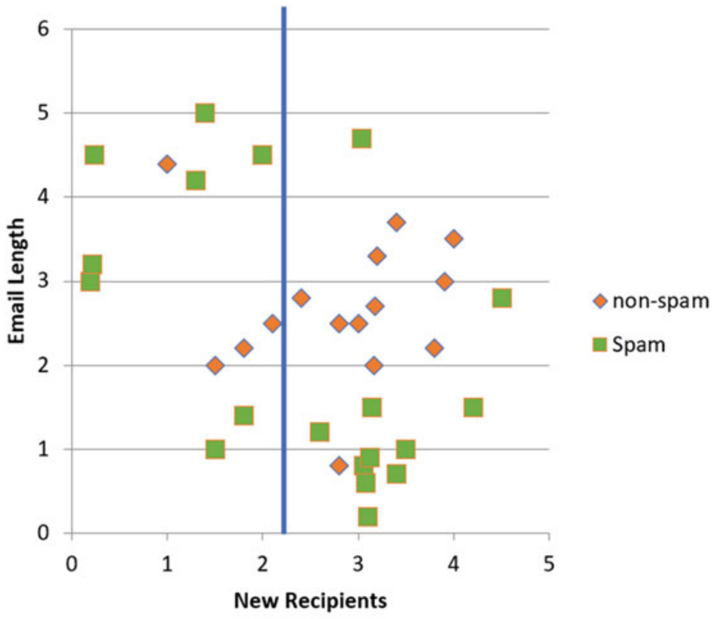


Fig. 8.8 Email spam training dataset scatter graph



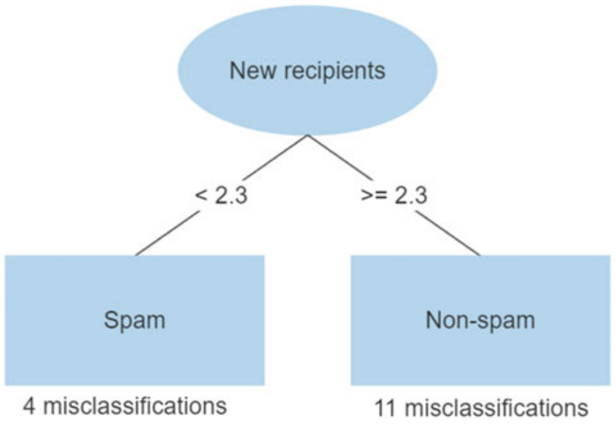


Fig. 8.9 A single-node decision tree using the New Recipients feature

In the next step, we will split the email subset with Email Length ≥ 1.8 into two new subsets: less than 4 and greater than or equal to 4; each area has a few classification errors (Fig. 8.11).

The process will continue until we reach convergence, i.e., until each region contains a sample from one class only (Fig. 8.12). Figure 8.12 shows nine different regions, each consisting of instances of the same class; the corresponding tree is shown in Fig. 8.13. This solution suffers from a lack of generalizability, as it has

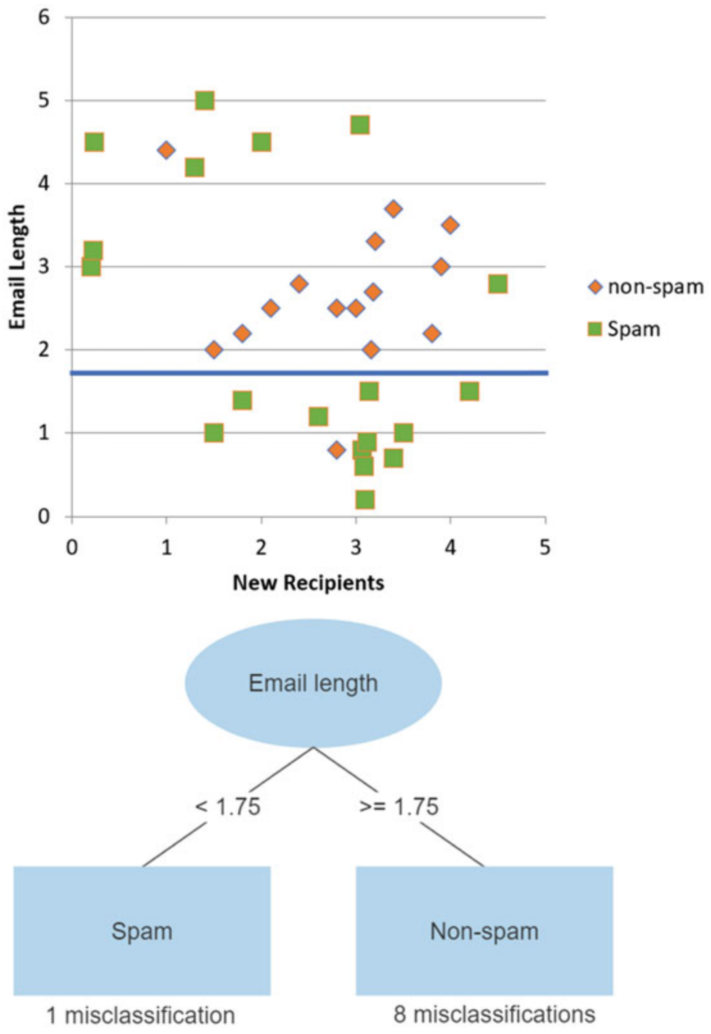


Fig. 8.10 A single-node decision tree using the Email Length feature

learned to classify the training dataset with 100% accuracy, but it risks not faring well with a new dataset.

8.3.3 *A Generic Algorithm*

Consider the dataset shown in Table 8.1, which represents a decision table to play outdoors based on four features related to the weather (Outlook, Temperature, Humidity, and Windy) [5].

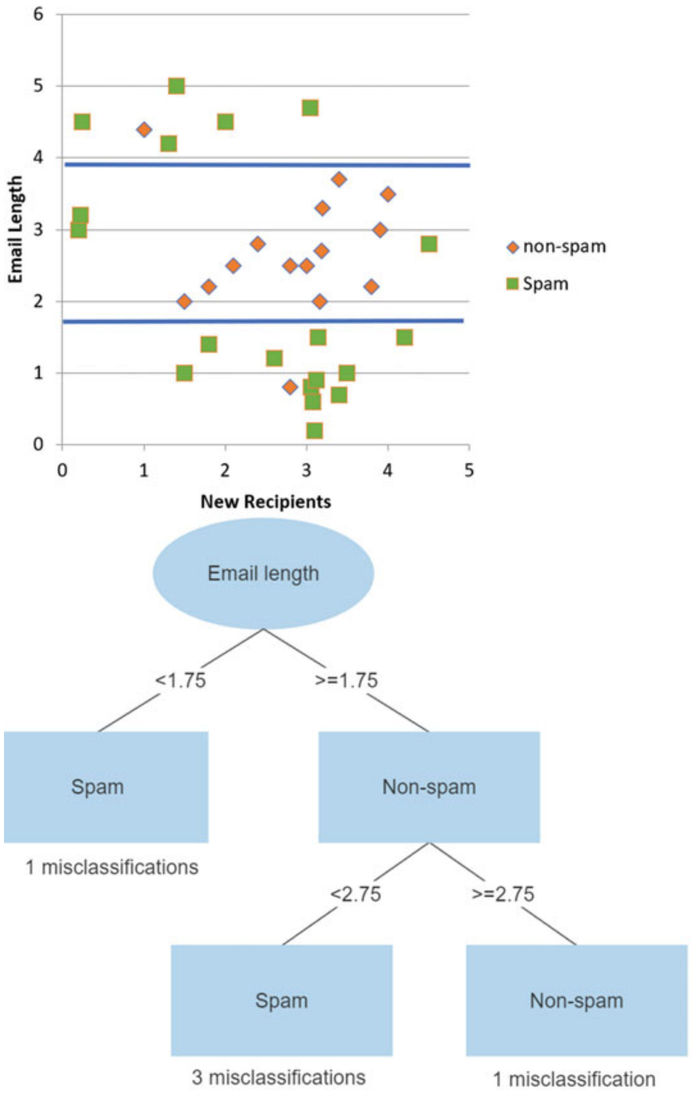


Fig. 8.11 The decision tree after two splits based on the Email Length feature

A decision tree can be created to decide whether to play outdoors or not based on the four weather conditions. The problem of creating the tree can be formulated as follows: choose an attribute to place at the tree's root, split the data to the left and right based on the values of the attribute, repeat the process for the left subset, then repeat the process for the right subset. For any left or right subset, stop when all instances at a node are of the same class. This is a recursive process.

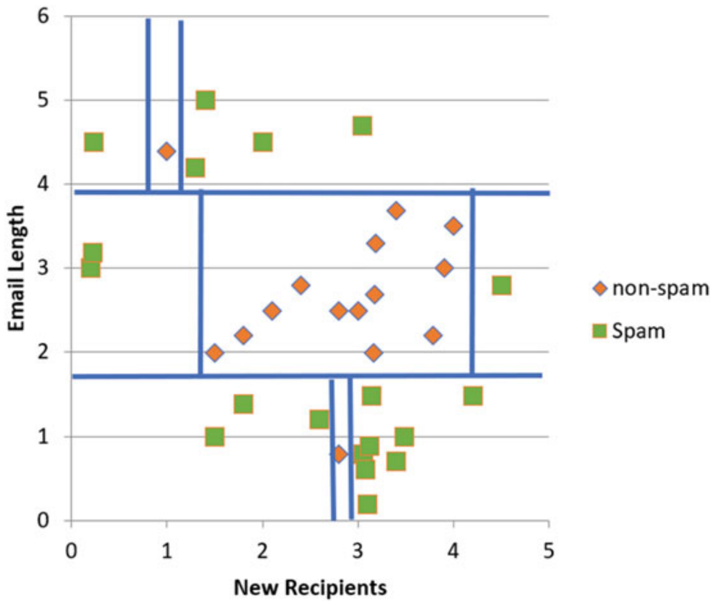


Fig. 8.12 The final graph split into distinct regions

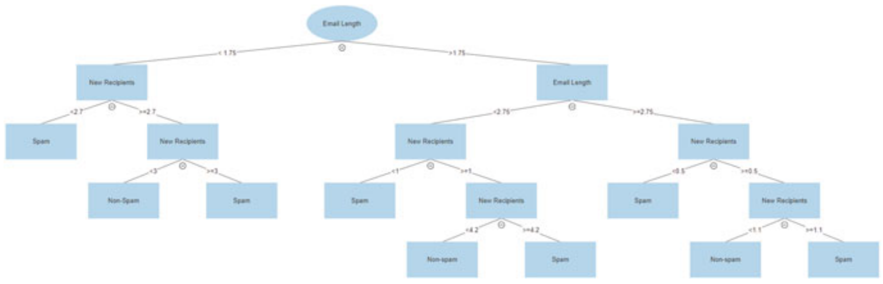


Fig. 8.13 The final tree solution corresponding to the graph in Fig. 8.8

Note that in this example, the data is binary nominal, while in the previous one, the data was numeric. The problem is that of classification and not of regression; however, trees can be used for regression.

Which attribute should we choose for the root? In the previous example about spam, we chose the attribute that generated the fewest classification errors. Here, we will aim at producing the fewest branches; we will do so by using a function known as the information value or entropy:

$$\text{entropy}\,(p_1, p_2 \ldots .p_n) = -p_1 \log(p_1) - p_2 \log(p_2) - \ \ldots \ - p_n \log(p_n)$$

where p_1, p_2, ... p_n are fractions, and $p_1 + p_2 + \ldots + p_n = 1$.

Table 8.1 Weather dataset

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

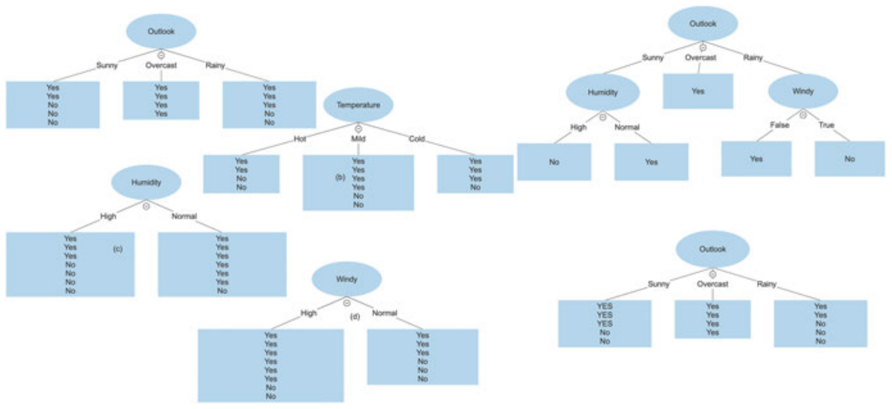


Fig. 8.14 Tree stumps for the weather dataset using each of the four features

The entropy is a measure of each node's purity, and we want to choose the feature that generates the purest daughter node, i.e., has as many instances of the same class as possible. If we take the example above and try to divide the instances based on each feature, we obtain the possibilities shown in Fig. 8.14.

The information value for the tree generated using the Outlook feature (Fig. 11.14) is computed as:

Information value [Sunny] = Information value [2 Yes and 3 No] = Entropy (2/5, 3/5) = −2/5×log(2/5)-3/5×log(3/5) = 0.971.

Information value [Overcast] = Information value [4 Yes and 0 No] = Entropy (4/4, 0/4) = 4/4×log(4/4) + 0/4×log(0/4) = 0.

Information value [Rainy] = Information value [3 Yes and 1 No] = Entropy (3/5, 2/5) = −3/5×log(3/5)−2/5×log(2/5) = 0.971.

The information value for the feature Outlook is computed as an average considering the number of instances in each subtree =5/14×0.971 + 4/14×0 + 5/14×0.971 = 0.694.

The root before any branching had 9 Yeses and 5 Nos, so the information value at the root was entropy(9/14, 5/14) = 0.940.

The *information gain* made by branching the tree using Outlook = 0.940–0.694 = 0.246.

Gain(Outlook) = 0.246 bits. The unit used for measurement is called "bits" but is not the same as computer bits.

We can do the same computation of the information gain resulting from using the Temperature, Humidity, and Windy features; we compare the results and choose the feature that provided the highest information gain. In our example,

Gain(Outlook) = 0.246 bits
Gain(Temperature) = 0.029 bits
Gain(Humidity) = 0.152 bits
Gain(Windy) = 0.048 bits

Hence, the best choice is to use the Outlook feature to split the tree at the root.

We continue using the same process and logic with each of the subtrees produced by Outlook, using the remaining features (i.e., Humidity and Windy). The result is shown in Fig. 8.15.

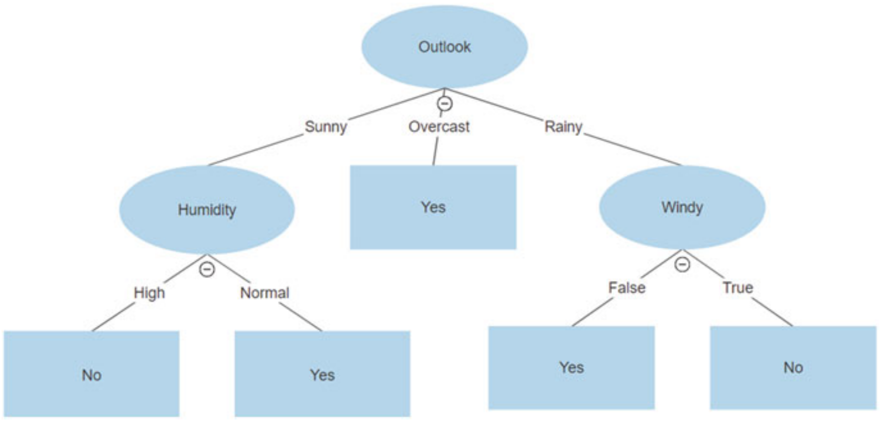


Fig. 8.15 The final decision tree for the weather dataset

8.3.4 Tree Pruning

Fully developed decision trees are complex in structure and risk overfitting as they learn to perfectly classify the training data and become less able to correctly classify new independent datasets. Pruning is a method that simplifies a decision tree; there are two methods for tree pruning: post-pruning or backward pruning and pre-pruning or forward pruning.

Pre-pruning involves a decision to stop developing subtrees while working on the development of a decision tree. Post-pruning seems more onerous, but it has the advantage of taking into account the combined effect of features on the decision instead of looking into the effect of each feature individually and deciding not to use it.

There are two methods for pruning: subtree replacement and subtree raising. In subtree replacement, we investigate the possibility of replacing a subtree with a leaf; it will make the tree less accurate on the training data but more generalizable (for unseen data). Subtree replacement works from the leaves upward in a tree. Subtree raising is more complex and time-consuming but more useful and is used in the well-known C.45 algorithm. In subtree raising, a whole subtree is removed, and its daughters are included in other subtrees. It is common to raise the subtree of the most popular branch.

If we take the example of a fully developed tree before pruning (Fig. 8.16a), subtree replacement of branch B can result in moving 4 and 5 to subtree C and deleting B (Fig. 8.16b); 1, 9, and 10 are the new instances resulting from the addition of instances 4 and 5 to 1, 2, and 3.

Subtree raising can result in the same replacement only if the total instances under 4 and 5 are fewer than those under C; otherwise, we replace B with node 4 or 5, whichever has more instances, and we reclassify all other instances 1, 2, 3 as well as 4 or 5 under the new node. Figure 8.16c shows a subtree raising result when node

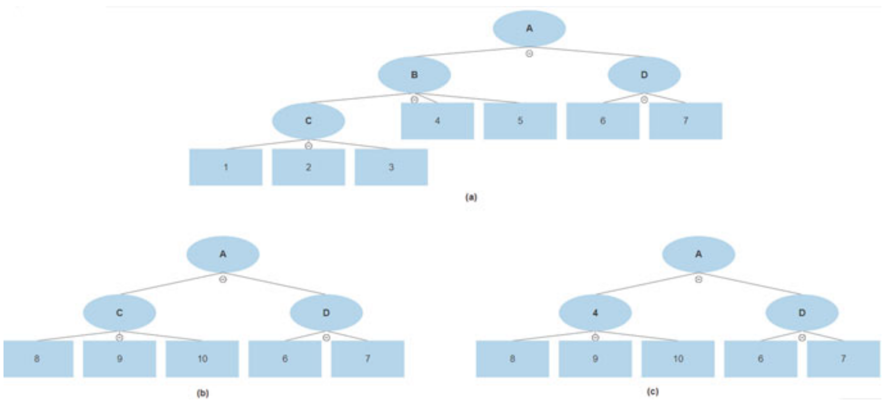


Fig. 8.16 Example of a fully developed tree before (**a**) and after (**b**, **c**) pruning

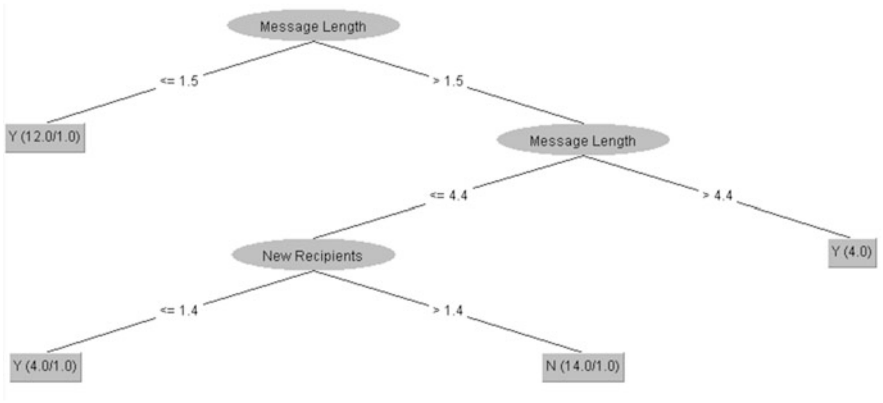


Fig. 8.17 Decision tree for the weather dataset using C.45 in Weka

4 has the most training instances; here, 8, 9, and 10 result from the reclassification of instances 1, 2, 3, 4, and 5.

In practice, if we apply the C.45 decision tree algorithm (called J48 in Weka) using pruning (done by default) to the weather data above, we will obtain the tree illustrated in Fig. 8.17.

In the leaf nodes, the first value in the parentheses is the number of instances from the training set in that leaf, while the second value is the number of instances incorrectly classified in that leaf.

8.4 Final Notes: Advantages, Disadvantages, and Best Practices

Decision trees are nonlinear algorithms, as opposed to the two linear algorithms we have introduced so far: linear regression and logistic regression. Linear discriminant analysis (LDA) is another traditional machine learning linear algorithm that we have not covered. Decision trees do not require specific data preparation steps and can be used for classification as well as for regression. However, in python, the tree-based algorithms in Python require numeric features only; hence, we need to transform categorical features to numeric ones using One-Hot Encoding.

8.5 Key Terms

1. Root
2. Leaf
3. decision node

4. Parent node
5. Child node
6. Subtree
7. Branch
8. Classification and regression trees
9. CART
10. Random forest
11. Binary tree
12. Entropy
13. Information gain
14. Gini index
15. Information gain Ratio
16. REPTree algorithm
17. Split criterion
18. Stop criterion
19. Decision stump
20. Entropy
21. Purity
22. Tree pruning
23. Overfitting
24. Pruning
25. Post-pruning
26. Backward pruning
27. Pre-pruning
28. Forward pruning
29. Subtree replacement
30. Subtree raising
31. C.45 algorithm
32. ID3

8.6 Test Your Understanding

1. Define the information gain ratio.
2. How do you compute the information gain ratio?
3. What does purity measure?
4. Give an example of a purity function.
5. What does CART stand for?
6. Which performs better, the REPTree algorithm or the J45 algorithm?
7. What does entropy measure exactly?
8. Define pre-pruning.
9. Define post-pruning.
10. Which is preferable, subtree raising or subtree replacement?

11. Can we use decision trees for regression analysis? Search for an example in the literature and summarize it.

8.7 Read More

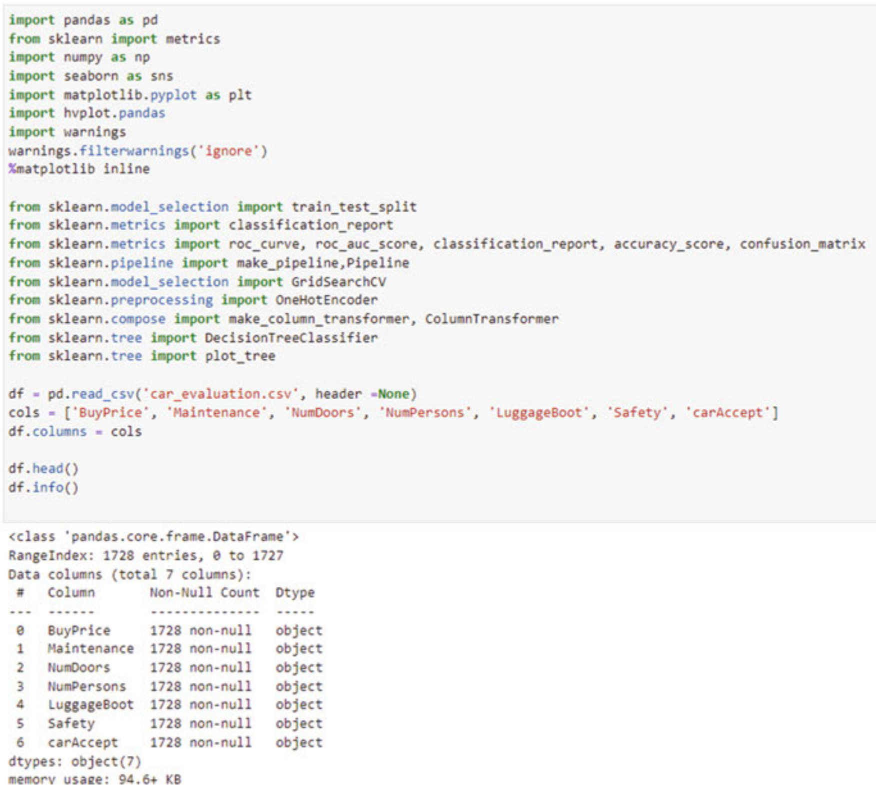
1. Arjoune, Y., & Faruque, S. (2020, 10–13 Dec. 2020). Real-time Machine Learning Based on Hoeffding Decision Trees for Jamming Detection in 5G New Radio. 2020 IEEE International Conference on Big Data (Big Data),
2. Bashar, S. S., Miah, M. S., Karim, A. H. M. Z., & Mahmud, M. A. A. (2019, 20–22 Dec. 2019). Extraction of Heart Rate from PPG Signal: A Machine Learning Approach using Decision Tree Regression Algorithm. 2019 fourth International Conference on Electrical Information and Communication Technology (EICT),
3. Bochkarev, B., & Rakitskiy, A. (2019, 21–27 Oct. 2019). The Study of the Applicability of Machine Learning Methods Based on Decision Trees for Holter Monitoring. 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON),
4. Chandrasegar, T., & Vutukuri, S. B. N. (2019, 22–23 March 2019). Optimized machine learning model using Decision Tree for cancer prediction. 2019 Innovations in Power and Advanced Computing Technologies (i-PACT),
5. Deen, A. J., & Gyanchandani, M. (2020, 8–10 Jan. 2020). Machine Learning Classifiers based on Predicting Membrane Protein using Decision Tree and Random Forest. 2020 Fourth International Conference on Inventive Systems and Control (ICISC),
6. Fontoura, L. C. M. M., Lins, H. W. D. C., Bertuleza, A. S., D'assunção, A. G., & Neto, A. G. (2021). Synthesis of Multiband Frequency Selective Surfaces Using Machine Learning With the Decision Tree Algorithm. IEEE Access, 9, 85,785–85,794. https://doi.org/8.1109/ACCESS.2021.3086777
7. Gupta, S. (2020, 10–13 Dec. 2020). A Hybrid Machine Learning Framework of Gradient Boosting Decision Tree and Sequence Model for Predicting Escalation in Customer Support. 2020 IEEE International Conference on Big Data (Big Data),
8. Hazra, R., Banerjee, M., & Badia, L. (2020, 4–7 Nov. 2020). Machine Learning for Breast Cancer Classification With ANN and Decision Tree. 2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON),
9. Jimoh, I. A., Ismaila, I., & Olalere, M. (2019, 10–12 Dec. 2019). Enhanced Decision Tree-J48 With SMOTE Machine Learning Algorithm for Effective Botnet Detection in Imbalance Dataset. 2019 15th International Conference on Electronics, Computer and Computation (ICECCO),
10. Kuang, W., Chan, Y., Tsang, S., & Siu, W. (2020). Machine Learning-Based Fast Intra Mode Decision for HEVC Screen Content Coding via Decision Trees.

IEEE Transactions on Circuits and Systems for Video Technology, 30(5), 1481–1496. https://doi.org/8.1109/TCSVT.2019.2903547
11. Li, Y., Song, X., Zhao, S., & Gao, F. (2020, 4–7 June 2020). A Line-Fault Cause Analysis Method for Distribution Network Based on Decision-Making Tree and Machine Learning. 2020 fifth Asia Conference on Power and Electrical Engineering (ACPEE),
12. Linty, N., Farasin, A., Favenza, A., & Dovis, F. (2019). Detection of GNSS Ionospheric Scintillations Based on Machine Learning Decision Tree. IEEE Transactions on Aerospace and Electronic Systems, 55(1), 303–317. https://doi.org/8.1109/TAES.2018.2850385
13. Mohagaonkar, S., Rawlani, A., & Saxena, A. (2019, 13–15 March 2019). Efficient Decision Tree using Machine Learning Tools for Acute Ailments. 2019 sixth International Conference on Computing for Sustainable Global Development (INDIACom),
14. Patil, S., & Kulkarni, U. (2019, 23–25 April 2019). Accuracy Prediction for Distributed Decision Tree using Machine Learning approach. 2019 third International Conference on Trends in Electronics and Informatics (ICOEI),
15. Posonia, A. M., Vigneshwari, S., & Rani, D. J. (2020, 3–5 Dec. 2020). Machine Learning based Diabetes Prediction using Decision Tree J48. 2020 third International Conference on Intelligent Sustainable Systems (ICISS),
16. Prapty, A. S., & Shitu, T. T. (2020, 19–21 Dec. 2020). An Efficient Decision Tree Establishment and Performance Analysis with Different Machine Learning Approaches on Polycystic Ovary Syndrome. 2020 23rd International Conference on Computer and Information Technology (ICCIT),
17. Romero, M. P., Chang, Y. M., Brunton, L. A., Parry, J., Prosser, A., Upton, P., Rees, E., Tearne, O., Arnold, M., Stevens, K., & Drewe, J. A. (2020). Decision tree machine learning applied to bovine tuberculosis risk factors to aid disease control decision making. Prev Vet Med, 175, 104,860. https://doi.org/8.1016/j.prevetmed.2019.104860
18. Shemu, N. A., Hossain, M. Z., Saleh, S. M., & Pavel, K. A. A. (2020, 9–10 Jan. 2020). A Machine Learning View for Health Data Mining Emphasizes on the Decision Trees. 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM),
19. Zhang, Y., Liu, J., Zhang, Z., & Huang, J. (2019, 12–14 July 2019). Prediction of Daily Smoking Behavior Based on Decision Tree Machine Learning Algorithm. 2019 IEEE ninth International Conference on Electronics Information and Emergency Communication (ICEIEC),

8.8 Lab

8.8.1 Working Example in Python

We will work with a car evaluation dataset that you can download from the following link:



```
import pandas as pd
from sklearn import metrics
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import hvplot.pandas
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, roc_auc_score, classification_report, accuracy_score, confusion_matrix
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

df = pd.read_csv('car_evaluation.csv', header =None)
cols = ['BuyPrice', 'Maintenance', 'NumDoors', 'NumPersons', 'LuggageBoot', 'Safety', 'carAccept']
df.columns = cols

df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   BuyPrice     1728 non-null   object
 1   Maintenance  1728 non-null   object
 2   NumDoors     1728 non-null   object
 3   NumPersons   1728 non-null   object
 4   LuggageBoot  1728 non-null   object
 5   Safety       1728 non-null   object
 6   carAccept    1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Fig. 8.18 Load car evaluation dataset

https://www.kaggle.com/elikplim/car-evaluation-data-set
This dataset evaluates cars (accept/reject) based on the following structure:

- Buying: buying car price
- Maintenance: maintenance car cost
- NumDoors: number of doors
- NumPersons: number of persons fitting in the car
- LuggageBoot: the size of the trunk ("luggage boot" in the UK)
- Safety: estimated safety of the car
- carAccept: car acceptability

8.8.1.1 Load Car Evaluation Dataset

Import the required libraries and install any library that you have not installed previously, then load the dataset into pandas in Fig. 8.18. Notice that we have read the csv file without the header and then added the column/features names according

```
#Data Visualisation
f, ax = plt.subplots(figsize=(7, 5))
sns.countplot(x='carAccept', data=df)
_ = plt.title('Car Evaluation')
_ = plt.xlabel('carAccept (1==Car Acceptability)')
```

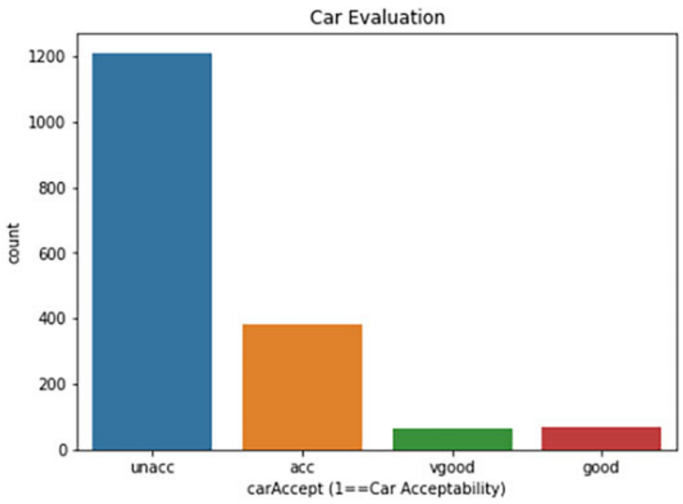


Fig. 8.19 Visualize car evaluation dataset and preprocess data

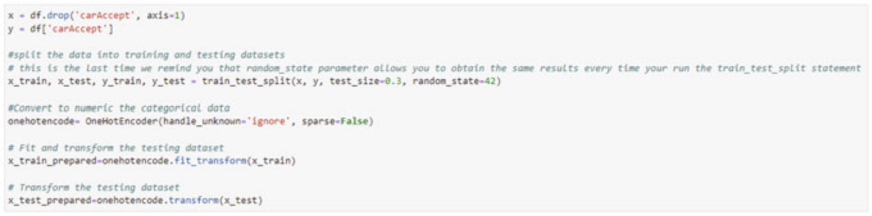
to our taste. None of the features present null values, so we will not process null value at this moment.

8.8.1.2 Visualize Car Evaluation

Visualize the data using the required libraries, we present in Fig. 8.19 a plot of the class (i.e., car evaluation).

8.8.1.3 Split and Scale Data

The next task is to split data into features vector x and class vector y, and then split x and y into training and testing datasets. Since trees in Python cannot process categorical features we need to one-hot encode all categorical features present in our datasets using OnEHotEncode. The result is two variables onehotencoded x_train_prepared and x-test_prepared (Fig. 8.20). You can display those variables to see the result.



```
x = df.drop('carAccept', axis=1)
y = df['carAccept']

#split the data into training and testing datasets
# this is the last time we remind you that random_state parameter allows you to obtain the same results every time your run the train_test_split statement
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

#Convert to numeric the categorical data
onehotencode= OneHotEncoder(handle_unknown='ignore', sparse=False)

# Fit and transform the testing dataset
x_train_prepared=onehotencode.fit_transform(x_train)

# Transform the testing dataset
x_test_prepared=onehotencode.transform(x_test)
```

Fig. 8.20 Converting and scaling data

8.8.1.4 Optimize Decision Tree Model

Grid search is used to tune the decision tree's hyperparamteers and find the optimal decision tree for the dataset. We also print the best parameters found and the best model, as well as its accuracy, and we plot the optimal decision tree (Fig. 8.21). Note that GridSearch will need several minutes to find the optimal tree, be patient.

Finally, we can test the best model by making predictions using the testing dataset. The performance (i.e., accuracy) is also computed and displayed (Fig. 8.22).

8.8.2 Working Example in Weka

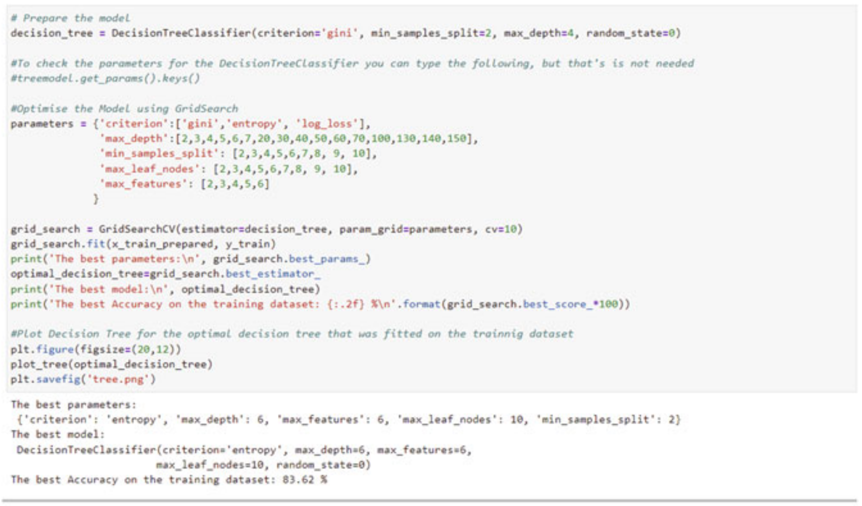
Download and open the iris dataset available from one of the following links:

1. https://archive-beta.ics.uci.edu/ml/datasets/53
2. https://archive.ics.uci.edu/ml/datasets/iris
3. https://www.kaggle.com/uciml/iris

The variables provided are as follows:

1. Id: identification
2. SepalLengthCm: sepal length in cm
3. SepalWidthCm: sepal width in cm
4. PetalLengthCm: petal length in cm
5. PetalWidthCm: petal width in cm
6. Species: the species (Iris-setosa, Iris-versicolor, or Iris-virginica)

1. Open the file in Weka and display the histograms for the four features.
2. Do you have an idea of which features are interesting for our classification problem? Probably not but take a few minutes to study the histograms and write your remarks.
3. Use the dataset to build a decision tree model to classify the irises into one of the three species based on the four features provided. You should be able to display the following output (Fig. 8.23) and tree (Fig. 8.24).



```
# Prepare the model
decision_tree = DecisionTreeClassifier(criterion='gini', min_samples_split=2, max_depth=4, random_state=0)

#To check the parameters for the DecisionTreeClassifier you can type the following, but that's is not needed
#treemodel.get_params().keys()

#Optimise the Model using GridSearch
parameters = {'criterion':['gini','entropy', 'log_loss'],
              'max_depth':[2,3,4,5,6,7,20,30,40,50,60,70,100,130,140,150],
              'min_samples_split': [2,3,4,5,6,7,8, 9, 10],
              'max_leaf_nodes': [2,3,4,5,6,7,8, 9, 10],
              'max_features': [2,3,4,5,6]
             }

grid_search = GridSearchCV(estimator=decision_tree, param_grid=parameters, cv=10)
grid_search.fit(x_train_prepared, y_train)
print('The best parameters:\n', grid_search.best_params_)
optimal_decision_tree=grid_search.best_estimator_
print('The best model:\n', optimal_decision_tree)
print('The best Accuracy on the training dataset: {:.2f} %\n'.format(grid_search.best_score_*100))

#Plot Decision Tree for the optimal decision tree that was fitted on the trainnig dataset
plt.figure(figsize=(20,12))
plot_tree(optimal_decision_tree)
plt.savefig('tree.png')
```

```
The best parameters:
 {'criterion': 'entropy', 'max_depth': 6, 'max_features': 6, 'max_leaf_nodes': 10, 'min_samples_split': 2}
The best model:
 DecisionTreeClassifier(criterion='entropy', max_depth=6, max_features=6,
                        max_leaf_nodes=10, random_state=0)
The best Accuracy on the training dataset: 83.62 %
```

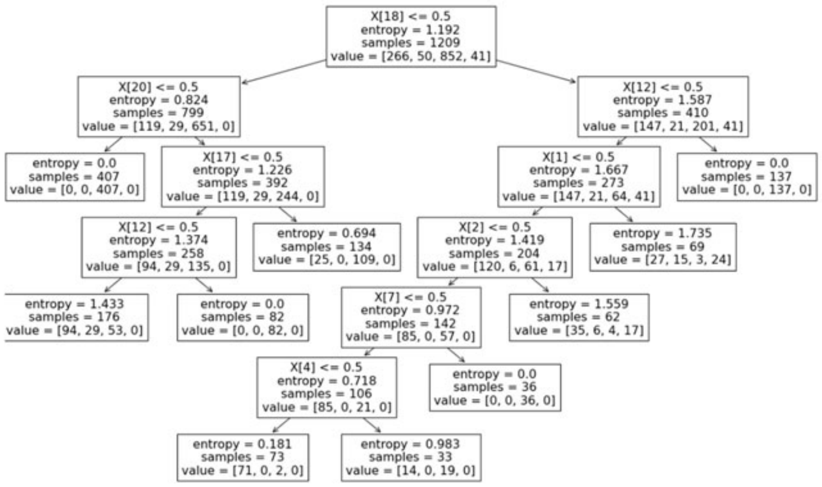
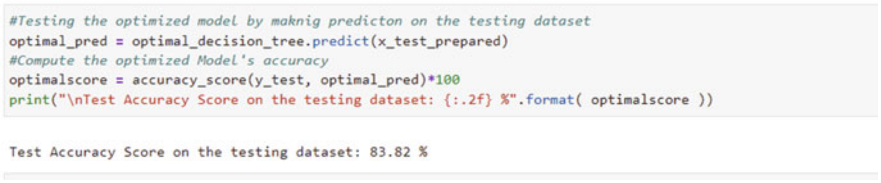


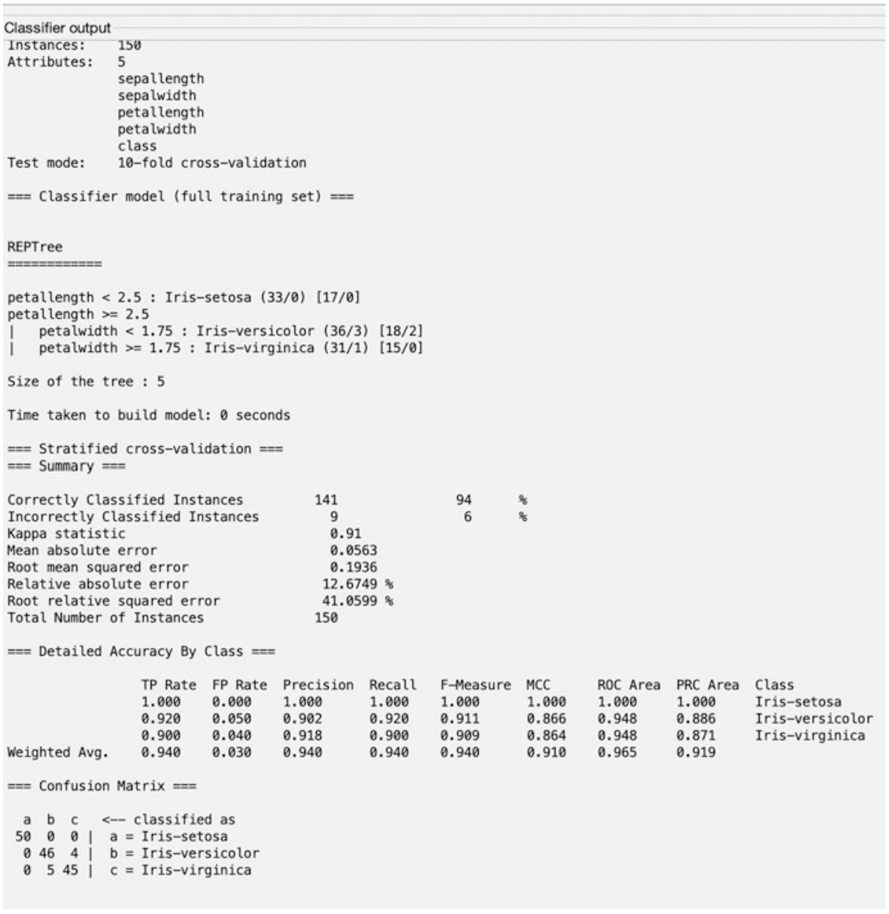
Fig. 8.21 Optimizing decision tree model using grid search cross-validation



```
#Testing the optimized model by maknig predicton on the testing dataset
optimal_pred = optimal_decision_tree.predict(x_test_prepared)
#Compute the optimized Model's accuracy
optimalscore = accuracy_score(y_test, optimal_pred)*100
print("\nTest Accuracy Score on the testing dataset: {:.2f} %".format( optimalscore ))
```

```
Test Accuracy Score on the testing dataset: 83.82 %
```

Fig. 8.22 Testing the best model and printing its accuracy on the testing dataset



```
Classifier output
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


REPTree
============

petallength < 2.5 : Iris-setosa (33/0) [17/0]
petallength >= 2.5
|   petalwidth < 1.75 : Iris-versicolor (36/3) [18/2]
|   petalwidth >= 1.75 : Iris-virginica (31/1) [15/0]

Size of the tree : 5

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         141               94      %
Incorrectly Classified Instances         9                6      %
Kappa statistic                          0.91
Mean absolute error                      0.0563
Root mean squared error                  0.1936
Relative absolute error                 12.6749 %
Root relative squared error             41.0599 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 0.920    0.050    0.902      0.920    0.911      0.866    0.948     0.886     Iris-versicolor
                 0.900    0.040    0.918      0.900    0.909      0.864    0.948     0.871     Iris-virginica
Weighted Avg.    0.940    0.030    0.940      0.940    0.940      0.910    0.965     0.919

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 46  4 |  b = Iris-versicolor
  0  5 45 |  c = Iris-virginica
```

Fig. 8.23 Classifier output

8.8.3 *Do It Yourself*

8.8.3.1 Decision Tree: Reflections on the Car Evaluation Dataset

You can notice that the optimal decision tree in Fig. 8.21 does not provide feature names in the leaves, instead we have x[1], x[2], etc. This is because x_train_prepared is an array that has no titles.

1. Is it possible for you to convert it to a data frame (x_train_prepared_df) and provide appropriate feature names for the data frame's columns? Hint: use pandas.DataFrame for conversion and .columns to provide name for the columns like we did in Fig. 8.18.

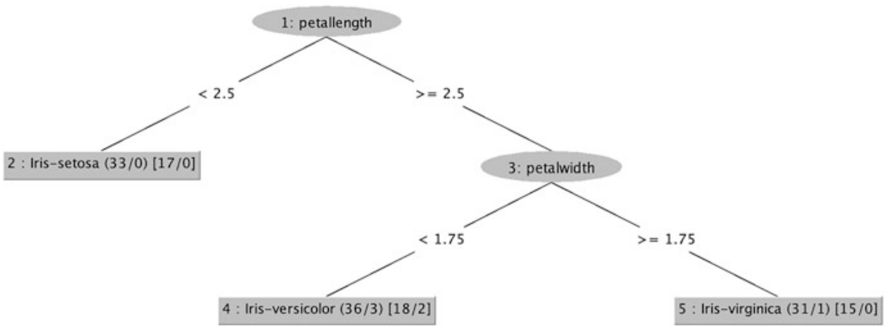

1: petallength
< 2.5
>= 2.5
2 : Iris-setosa (33/0) [17/0]
3: petalwidth
< 1.75
>= 1.75
4 : Iris-versicolor (36/3) [18/2]
5 : Iris-virginica (31/1) [15/0]

Fig. 8.24 Decision tree for the iris classification problem

2. If this is doable then can you use x_ train _prepared_df to fit a new similar model to the one presented in Fig. 8.20 with appropriate column names.
3. Trees are prone to overfitting. The minimum number of samples a node can have before it is a candidate for splitting (min_samples_split), the minimum number of samples a leaf must have (min_samples_leaf), the maximum number of leaf nodes (max_leaf_nodes) and the maximum number of features evaluated for splitting at each node (max_features) can regularize the tree: increasing the min_ hyperparameters and decreasing the max_ hyperparameters values. Try to reduce the change in the parameters provided to gridsearch; for example, remove some of the min and max hyperparameters or all, or add min_samples_leaf and see how the optimal tree and its accuracy change.
4. Can you notice overfitting when you remove parameters related to regularization? Explain.

8.8.3.2 Decision Trees for Regression

We have seen decision trees for classification. In this exercise, we will overview decision trees' use for regression. Download the Boston housing dataset from the following link: https://www.kaggle.com/prasadperera/the-boston-housing-dataset. The dataset is also available from the numeric dataset you downloaded previously.

The dataset is composed of the following features:

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq. ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxide concentration (parts per ten million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centers
9. RAD: index of accessibility to radial highways

10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: $1000(Bk - 0.63)^2$,where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: median value of owner-occupied homes in $1000s

1. Build a decision tree model to predict the median value of a Boston house (MEDV) based on the available features. Below is a sample output from Weka.
2. Do you want to do any preprocessing? Which processing? For which feature?
3. Compare your results before and after preprocessing.
4. Any notes about the data? Do you think that this data relates to questions of bias and racism? Explain your answer.
5. Do you know of any machine learning applications that have previously raised ethical concerns?
6. Do you have any ethical concerns regarding future applications for machine learning?

8.8.3.3 Decision Trees for Classification

Download the train and test datasets of the *Titanic* from https://www.kaggle.com/c/titanic/data?select=train.csv.

The variables provided are as follows:

1. Survival: 0 = No, 1 = Yes
2. Pclass is the ticket class: 1 = first, 2 = second, 3 = 3rd
3. Sex: M or F
4. Age: age in years
5. Sibsp: number of siblings or spouses aboard the ship
6. Parch: number of parents or children aboard the ship
7. Ticket: ticket number
8. Fare: passenger fare
9. Cabin: cabin number
10. Embarked: the port of embarkation, C=Cherbourg, Q = Queenstown, S=Southampton

1. Use decision trees to build a model that predicts the survival of a passenger based on the available features. Note the accuracy of the algorithm and other measurements and display the decision tree.
2. Which other algorithm can you use to tackle this classification problem? Suggest one and execute the necessary instructions to build a new classification model.
3. Compare the two models (the decision tree model and the other suggested model). Which one makes better decisions? On which data have you based your decision?

8.8.4 Do More Yourself

1. Mushroom dataset (classification): https://www.kaggle.com/uciml/mushroom-classification
2. London bike-sharing dataset (regression): https://www.kaggle.com/hmavrodiev/london-bike-sharing-dataset

References

1. A. Burkov, *The Hundred-Page Machine Learning Book* (Andriy Burkov, 2019)
2. O.Z. Maimon, L. Rokach, *Data Mining with Decision Trees: Theory and Applications*, 2nd edn. (World Scientific Publishing Company, 2014)
3. V.G. Sigillito, S.P. Wing, L.V. Hutton, K.B. Baker, Classification of radar returns from the ionosphere using neural networks. J. Hopkins APL Tech. Dig. **10**, 262–266 (1989)
4. J.R. Quinlan, Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1 March 1986). https://doi.org/10.1007/BF00116251
5. I.H. Witten, E. Frank, M.A. Hall, C. Pal, *Data Mining: Practical Machine Learning Tools and Techniques* (Elsevier Science, 2016)