UNIT 2
CLUSTERING AND CLASSIFICATION

Advanced Analytical Theory and Methods: Classification

In classification learning, a classifier is presented with a set of examples that are already classified and, from these examples, the classifier learns to assign unseen examples. In other words, the primary task performed by classifiers is to assign class labels to new observations. Logistic regression is one of the popular classification methods. The set of labels for classifiers is predetermined, unlike in clustering, which discovers the structure without a training set and allows the data scientist optionally to create and assign labels to the clusters.

Most classification methods are supervised, in that they start with a training set of prelabeled observations to learn how likely the attributes of these observations may contribute to the classification of future unlabeled observations. For example, existing marketing, sales, and customer demographic data can be used to develop a classifier to assign a ―purchase‖ or ―no purchase‖ label to potential future customers.

Classification is widely used for prediction purposes. For example, by building a classifier on the transcripts of United States Congressional floor debates, it can be determined whether the speeches represent support or opposition to proposed legislation. Classification can help health care professionals diagnose heart disease patients. Based on an e-mail‘s content, e-mail providers also use classification to decide whether the incoming e-mail messages are spam.

This chapter mainly focuses on two fundamental classification methods: ***decision trees*** and ***naïve Bayes***.

2.1 Decision Trees

A ***decision tree*** (also called ***prediction tree***) uses a tree structure to specify sequences of decisions and consequences. Given input $X = \{x_1, x_2, \ldots x_n\}$, the goal is to predict a response or output variable Y. Each member of the set $\{x_1, x_2, \ldots x_n\}$ is called an ***input variable***. The prediction can be achieved by constructing a decision tree with test points and branches. At each test point, a decision is made to pick a specific branch and traverse down the tree. Eventually, a final point is reached, and a prediction can be made. Each test point in a decision tree involves testing a particular input variable (or attribute), and each branch represents the decision being made. Due to its flexibility and easy visualization, decision trees are commonly deployed in data mining applications for classification purposes.

The input values of a decision tree can be categorical or continuous. A decision tree employs a structure of test points (called ***nodes***) and branches, which represent the decision being made. A node without further branches is called a ***leaf node***. The leaf nodes return class labels and, in some implementations, they return the probability scores. A decision tree can be converted into a set of decision rules. In the following example rule,

income and *mortgage_amount* are input variables, and the response is the output variable *default* with a probability score.

```
IF income < $50,000 AND mortgage_amount > $100K
THEN default = True WITH PROBABILITY 75%
```

Decision trees have two varieties: ***classification trees*** and ***regression trees***. Classification trees usually apply to output variables that are categorical—often binary—in nature, such as yes or no, purchase or not purchase, and so on. Regression trees, on the other hand, can apply to output variables that are numeric or continuous, such as the predicted price of a consumer good or the likelihood a subscription will be purchased.

Decision trees can be applied to a variety of situations. They can be easily represented in a visual way, and the corresponding decision rules are quite straightforward. Additionally, because the result is a series of logical `if-then` statements, there is no underlying assumption of a linear (or nonlinear) relationship between the input variables and the response variable.

2.1.1 Overview of a Decision Tree

Figure 2.1 shows an example of using a decision tree to predict whether customers will buy a product. The term ***branch*** refers to the outcome of a decision and is visualized as a line connecting two nodes. If a decision is numerical, the "greater than" branch is usually placed on the right, and the "less than" branch is placed on the left. Depending on the nature of the variable, one of the branches may need to include an "equal to" component.

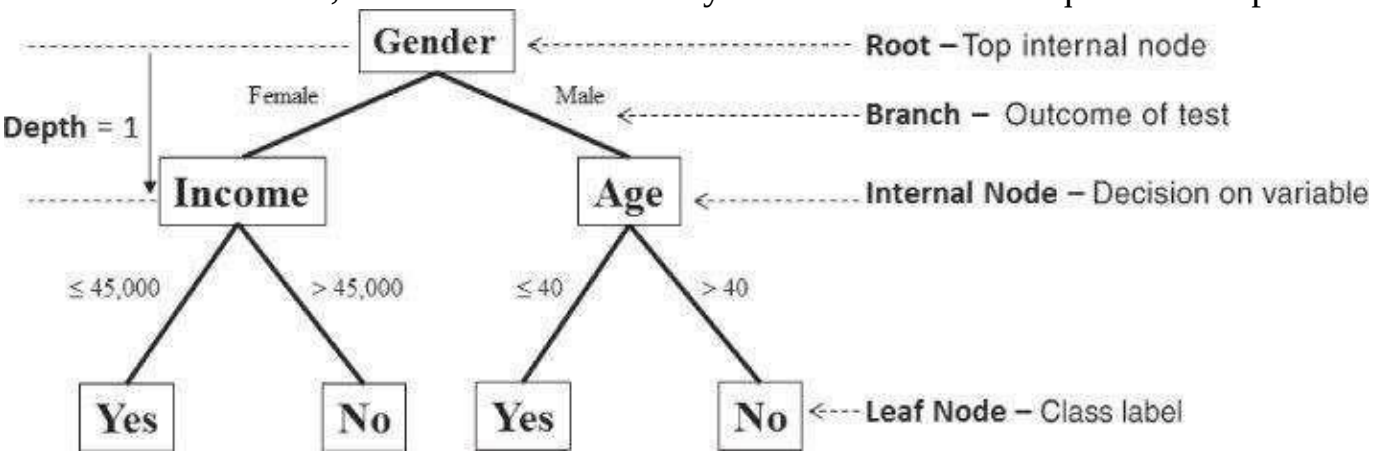


Figure 2.1 Example of a decision tree

Internal nodes are the decision or test points. Each internal node refers to an input variable or an attribute. The top internal node is called the ***root***. The decision tree in Figure 2.1 is a binary tree in that each internal node has no more than two branches. The branching of a node is referred to as a ***split***.

Sometimes decision trees may have more than two branches stemming from a node. For example, if an input variable *Weather* is categorical and has three choices—Sunny, Rainy, and Snowy—the corresponding node *Weather* in the decision tree may have three branches labeled as Sunny, Rainy, and Snowy, respectively.

The ***depth*** of a node is the minimum number of steps required to reach the node from the root. In Figure 2.1 for example, nodes *Income* and *Age* have a depth of one, and the four nodes on the bottom of the tree have a depth of two.

Leaf nodes are at the end of the last branches on the tree. They represent class labels—the outcome of all the prior decisions. The path from the root to a leaf node contains a series of decisions made at various internal nodes.

In Figure 2.1, the root node splits into two branches with a *Gender* test. The right branch contains all those records with the variable *Gender* equal to `Male`, and the left branch contains all those records with the variable *Gender* equal to `Female` to create the depth 1 internal nodes. Each internal node effectively acts as the root of a subtree, and a best test for each node is determined independently of the other internal nodes. The left-hand side (LHS) internal node splits on a question based on the *Income* variable to create leaf nodes at depth 2, whereas the right-hand side (RHS) splits on a question on the *Age* variable.

The decision tree in Figure 2.1 shows that females with income less than or equal to $45,000 and males 40 years old or younger are classified as people who would purchase the product. In traversing this tree, age does not matter for females, and income does not matter for males.

Decision trees are widely used in practice. For example, to classify animals, questions (like cold-blooded or warm-blooded, mammal or not mammal) are answered to arrive at a certain classification. Another example is a checklist of symptoms during a doctor's evaluation of a patient. The artificial intelligence engine of a video game commonly uses decision trees to control the autonomous actions of a character in response to various scenarios. Retailers can use decision trees to segment customers or predict response rates to marketing and promotions. Financial institutions can use decision trees to help decide if a loan application should be approved or denied. In the case of loan approval, computers can use the logical `if-then` statements to predict whether the customer will default on the loan. For customers with a clear (strong) outcome, no human interaction is required; for observations that may not generate a clear response, a human is needed for the decision.

By limiting the number of splits, a short tree can be created. Short trees are often used as ***components*** (also called ***weak learners*** or ***base learners***) in ***ensemble methods***. Ensemble methods use multiple predictive models to vote, and decisions can be made based on the combination of the votes. Some popular ensemble methods include random forest, bagging, and boosting . Section 2.4 discusses these ensemble methods more.

The simplest short tree is called a ***decision stump***, which is a decision tree with the root immediately connected to the leaf nodes. A decision stump makes a prediction based on the value of just a single input variable. Figure 2.2 shows a decision stump to classify two species of an iris flower based on the petal width. The figure shows that, if the petal width is smaller than 1.75 centimeters, it's Iris versicolor; otherwise, it's *Iris virginica*.

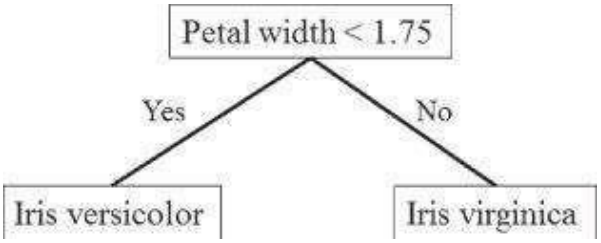


Figure 2.2 Example of a decision stump

To illustrate how a decision tree works, consider the case of a bank that wants to market its term deposit products (such as Certificates of Deposit) to the appropriate customers. Given the demographics of clients and their reactions to previous campaign phone calls, the bank‘s goal is to predict which clients would subscribe to a term deposit. The dataset used here is based on the original dataset collected from a Portuguese bank on directed marketing campaigns as stated in the work by Moro et al. [6]. Figure 2.3 shows a subset of the modified bank marketing dataset. This dataset includes 2,000 instances randomly drawn from the original dataset, and each instance corresponds to a customer. To make the example simple, the subset only keeps the following categorical variables: (1) *job*, (2) *marital* status, (3) *education* level, (4) if the credit is in *default*, (5) if there is a *housing* loan, (6) if the customer currently has a personal *loan*, (7) *contact* type, (8) result of the previous marketing campaign contact (*poutcome*), and finally (9) if the client actually *subscribed* to the term deposit. Attributes (1) through (8) are input variables, and (9) is considered the outcome. The outcome *subscribed* is either yes (meaning the customer will subscribe to the term deposit) or no (meaning the customer won‘t subscribe). All the variables listed earlier are categorical.

	job	marital	education	default	housing	loan	contact	poutcome	subscribed
1	management	single	tertiary	no	yes	no	cellular	unknown	no
2	entrepreneur	married	tertiary	no	yes	yes	cellular	unknown	no
3	services	divorced	secondary	no	no	no	cellular	unknown	yes
4	management	married	tertiary	no	yes	no	cellular	unknown	no
5	management	married	secondary	no	yes	no	unknown	unknown	no
6	management	single	tertiary	no	yes	no	unknown	unknown	no
7	entrepreneur	married	tertiary	no	yes	no	cellular	failure	yes
8	admin.	married	secondary	no	no	no	cellular	unknown	no
9	blue-collar	married	secondary	no	yes	no	cellular	other	no
10	management	married	tertiary	yes	no	no	cellular	unknown	no
11	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
12	management	divorced	secondary	no	no	no	unknown	unknown	no
13	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
14	retired	married	secondary	no	no	no	cellular	unknown	no
15	management	single	tertiary	no	yes	no	cellular	unknown	no
16	retired	married	secondary	yes	yes	no	cellular	unknown	no
17	unemployed	married	secondary	no	yes	no	telephone	unknown	no
18	management	divorced	tertiary	no	yes	no	cellular	unknown	no
19	management	married	tertiary	no	yes	no	cellular	unknown	no
20	blue-collar	married	secondary	no	yes	no	unknown	unknown	no
21	management	divorced	tertiary	no	yes	yes	cellular	failure	yes
22	blue-collar	divorced	secondary	no	yes	no	cellular	failure	no
23	blue-collar	single	secondary	no	yes	no	cellular	failure	no
24	admin.	single	secondary	no	no	no	unknown	unknown	no
25	blue-collar	married	secondary	no	yes	no	cellular	failure	no
26	blue-collar	single	secondary	no	yes	no	unknown	unknown	no
27	housemaid	married	secondary	no	no	no	cellular	unknown	no
28	technician	married	tertiary	no	no	no	cellular	unknown	no

Figure 2.3 A subset of the bank marketing dataset

A summary of the dataset shows the following statistics. For ease of display, the summary

only includes the top six most frequently occurring values for each attribute. The rest are displayed as **(Other)**.

```
   job    marital    education default
blue-collar:435  divorced:  228  primary : 335  no  :1961 management :423 married
:1201 secondary:1010 yes: 39 technician :339  single : 571  tertiary  : 564
admin.  :235       unknown : 91 services :168
retired : 92 (Other) :308
housing  loan    contact    month  poutcome
no  : 916  no :1717 cellular :1287 may :581  failure: 210 yes:1084 yes: 283
telephone: 136  jul :340 other : 79
       unknown : 577  aug :278 success: 58
            jun  :232  unknown:1653
            nov  :183
            apr  :118
            (Other):268 subscribed
no :1789 yes:  211
```

Attribute *job* includes the following values.

```
  admin. blue-collar entrepreneur  housemaid
   235   435   70   63
management  retired self-employed  services
   423   92   69   168
  student technician unemployed  unknown
   36   339   60   10
```

Figure 2.4 shows a decision tree built over the bank marketing dataset. The root of the tree shows that the overall fraction of the clients who have not subscribed to the term deposit is 1,789 out of the total population of 2,000.

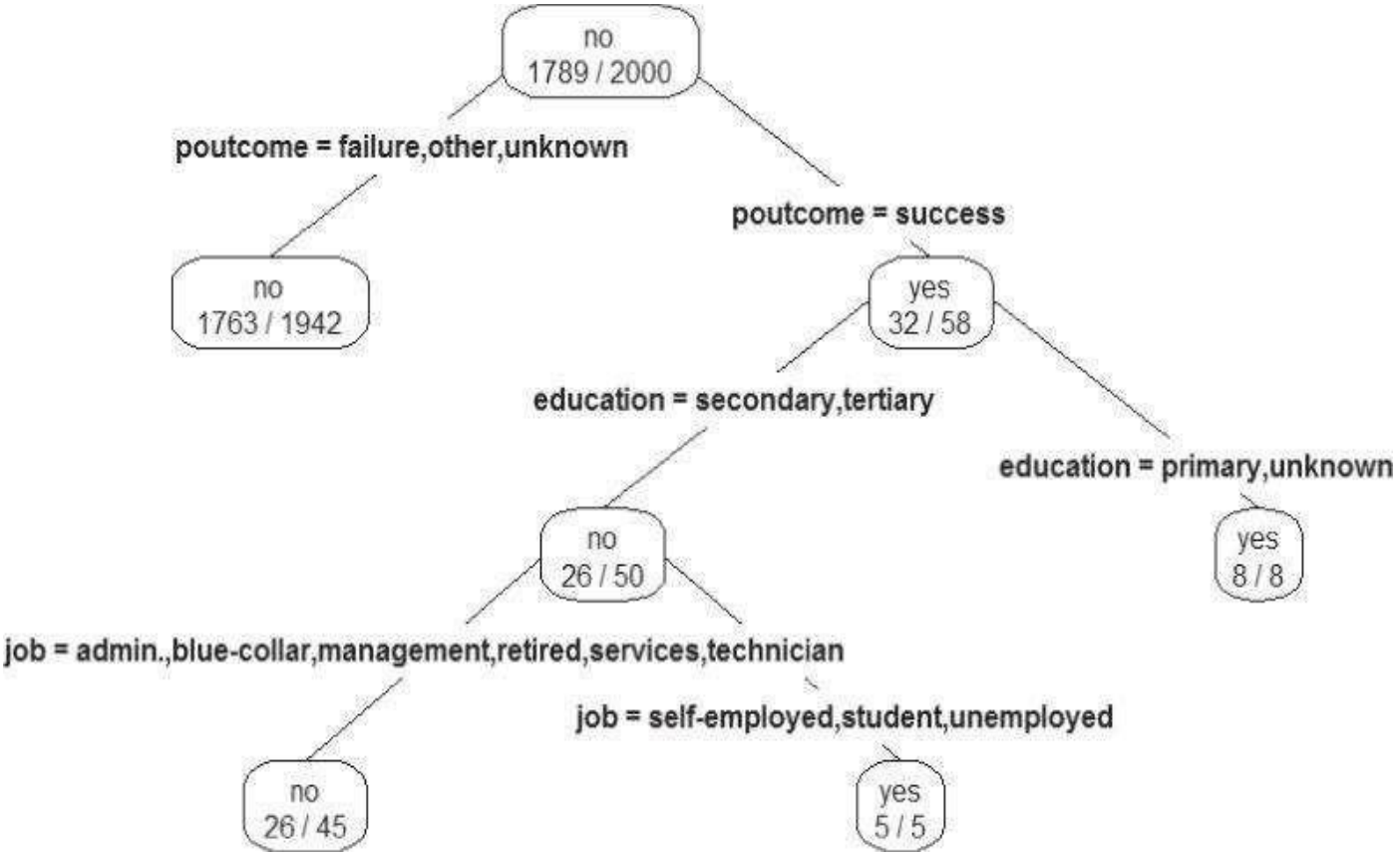


Figure 2.4 Using a decision tree to predict if a client will subscribe to a term deposit

At each split, the decision tree algorithm picks the most informative attribute out of the remaining attributes. The extent to which an attribute is informative is determined by measures such as entropy and information gain, as detailed in Section 2.1.2.

At the first split, the decision tree algorithm chooses the *poutcome* attribute. There are two nodes at depth=1. The left node is a leaf node representing a group for which the outcome of the previous marketing campaign contact is a `failure`, `other`, or unknown. For this group, 1,763 out of 1,942 clients have not subscribed to the term deposit.

The right node represents the *rest* of the population, for which the outcome of the previous marketing campaign contact is a success. For the population of this node, 32 out of 58 clients have subscribed to the term deposit.

This node further splits into two nodes based on the education level. If the *education* level is either secondary or tertiary, then 26 out of 50 of the clients have not subscribed to the term deposit. If the education level is primary or unknown, then 8 out of 8 times the clients have subscribed.

The left node at depth 2 further splits based on attribute *job*. If the occupation is admin, blue collar, management, retired, services, or technician, then 26 out of 45 clients have not subscribed. If the occupation is self-employed, student, or unemployed, then 5 out of 5 times the clients have subscribed.

2.1.2 The General Algorithm

In general, the objective of a decision tree algorithm is to construct a tree T from a training set S. If all the records in S belong to some class C (*subscribed* = yes, for example), or if S is sufficiently pure (greater than a preset threshold), then that node is considered a leaf node and assigned the label C. The ***purity*** of a node is defined as its probability of the corresponding class. For example, in Figure 2.4, the root $P(subscribed = yes) = 1 - \frac{1789}{2000} = 10.55\%$; therefore, the root is only 10.55% pure on the $subscribed = yes$ class. Conversely, it is 89.45% pure on the $subscribed = no$ class.

In contrast, if not all the records in S belong to class C or if S is not sufficiently pure, the algorithm selects the next most informative attribute A (duration, marital, and so on) and partitions S according to A_s values. The algorithm constructs subtrees T_1, T_2… for the subsets of S recursively until one of the following criteria is met:

- All the leaf nodes in the tree satisfy the minimum purity threshold.
- The tree cannot be further split with the preset minimum purity threshold.
- Any other stopping criterion is satisfied (such as the maximum depth of the tree).

The first step in constructing a decision tree is to choose the most informative attribute. A common way to identify the most informative attribute is to use entropy-based methods, which are used by decision tree learning algorithms such as ID3 (or Iterative Dichotomiser 3) [7] and C3.5 [8]. The entropy methods select the most informative attribute based on two basic measures:

- ***Entropy,*** which measures the *impurity* of an attribute
- ***Information gain,*** which measures the *purity* of an attribute

Given a class X and its label $x \in X$, let $P(x)$ be the probability of x. H_x the entropy of X, is defined as shown in Equation 2.1.

$$H_X = -\sum_{\forall x \in X} P(x)\log_2 P(x) \quad \textbf{2.1}$$

Equation 2.1 shows that entropy H_X becomes 0 when all $P(x)$ is 0 or 1. For a binary classification (`true` or `false`), H_X is zero if $P(x)$ the probability of each label x is either zero or one. On the other hand, H_X achieves the maximum entropy when all the class labels are equally probable. For a binary classification, $H_X = 1$ if the probability of all class labels is 50/50. The maximum entropy increases as the number of possible outcomes increases.

As an example of a binary random variable, consider tossing a coin with known, not necessarily fair, probabilities of coming up heads or tails. The corresponding entropy graph is shown in Figure 2.5. Let $x = 1$ represent heads and $x = 0$ represent tails. The entropy of the unknown result of the next toss is maximized when the coin is fair. That is, when heads and tails have equal probability $P(x = 1) = P(x = 0) = 0.5$, entropy $H_x = -(0.5 \times \log_2 0.5 + 0.5 \times \log_2 0.5) = 1$. On the other hand, if the coin is not fair, the probabilities of heads and tails would not be equal and there would be less uncertainty. As an extreme case, when the probability of tossing a head is equal to 0 or 1, the entropy is minimized to 0. Therefore, the entropy for a completely pure variable is 0 and is 1 for a set with equal occurrences for both the classes (head and tail, or yes and no).

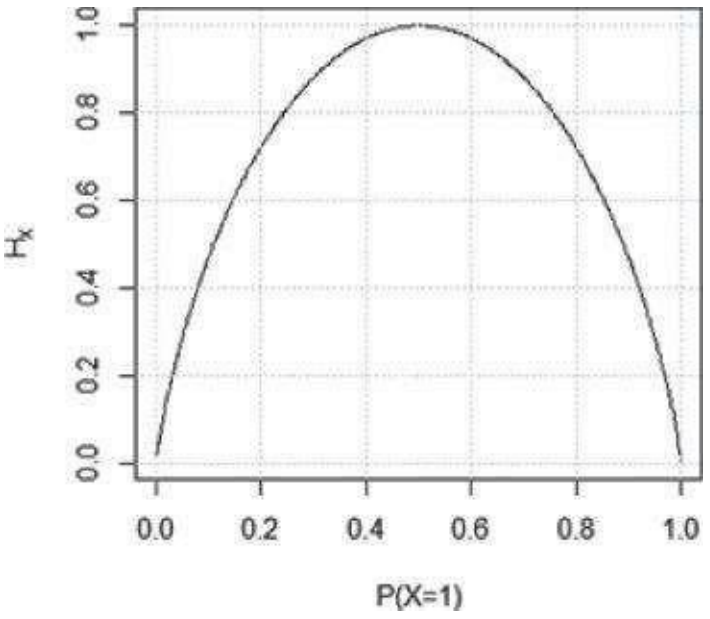


Figure 2.5 Entropy of coin flips, where X=1 represents heads

For the bank marketing scenario previously presented, the output variable is *subscribed*. The base entropy is defined as entropy of the output variable, that is $H_{subscribed}$. As seen previously, $P(subscribed = \text{yes}) = 0.1055$ and $P(subscribed = \text{no}) = 0.8945$. According to Equation 2.1, the base entropy $H_{subscribed} = -0.1055 \cdot \log_2 0.1055 - 0.8945 \cdot \log_2 0.8945 \approx 0.4862$.

The next step is to identify the conditional entropy for each attribute. Given an attribute X, its value x, its outcome Y, and its value y, conditional entropy $H_{Y|X}$ is the remaining entropy of Y given X, formally defined as shown in Equation 2.2.

$$H_{Y|X} = \sum_{x} P(x) H(Y|X=x)$$
$$= -\sum_{\forall x \in X} P(x) \sum_{\forall y \in Y} P(y|x) \log_2 P(y|x) \quad \textbf{2.2}$$

Consider the banking marketing scenario, if the attribute *contact* is chosen, $X = \{\text{cellular, telephone, unknown}\}$. The conditional entropy of *contact* considers all three values.

Table 2.1 lists the probabilities related to the *contact* attribute. The top row of the table displays the probabilities of each value of the attribute. The next two rows contain the probabilities of the class labels conditioned on the *contact*.

Table 2.1 Conditional Entropy Example

Cellular	**Telephone**	**Unknown**	
P(contact)	0.6435	0.0680	0.2885
P(subscribed=yes | contact)	0.1399	0.0809	0.0347
P(subscribed=no | contact)	0.8601	0.9192	0.9653

The conditional entropy of the *contact* attribute is computed as shown here.

$$H_{subscribed|contact} = -[0.6435 \cdot (0.1399 \cdot \log_2 0.1399 + 0.8601 \cdot \log_2 0.8601)$$
$$+ 0.0680 \cdot (0.0809 \cdot \log_2 0.0809 + 0.9192 \cdot \log_2 0.9192)$$
$$+ 0.2885 \cdot (0.0347 \cdot \log_2 0.0347 + 0.9653 \cdot \log_2 0.9653)]$$
$$= 0.4661$$

Computation inside the parentheses is on the entropy of the class labels within a single *contact* value. Note that the conditional entropy is always less than or equal to the base entropy—that is, $H_{subscribed|marital} \leq H_{subscribed}$. The conditional entropy is smaller than the base entropy when the attribute and the outcome are correlated. In the worst case, when the attribute is uncorrelated with the outcome, the conditional entropy equals the base entropy.

The information gain of an attribute *A* is defined as the difference between the base entropy and the conditional entropy of the attribute, as shown in Equation 2.3.

$$InfoGain_A = H_S - H_{S|A} \quad \textbf{2.3}$$

In the bank marketing example, the information gain of the *contact* attribute is shown in Equation 2.3.

$$InfoGain_{contact} = H_{subscribed} - H_{contact|subscribed}$$
$$= 0.4862 - 0.4661 = 0.0201 \quad \textbf{2.4}$$

Information gain compares the degree of purity of the parent node before a split with the degree of purity of the child node after a split. At each split, an attribute with the greatest information gain is considered the most informative attribute. Information gain indicates the purity of an attribute.

The result of information gain for all the input variables is shown in Table 2.2. Attribute *poutcome* has the most information gain and is the most informative variable. Therefore, *poutcome* is chosen for the first split of the decision tree, as shown in Figure 2.3. The values of information gain in Table 2.2 are small in magnitude, but the relative difference matters. The algorithm splits on the attribute with the largest information gain at each round.

Table 2.2 Calculating Information Gain of Input Variables for the First Split

Attribute	Information Gain
poutcome	0.0289
contact	0.0201
housing	0.0133
job	0.0101
education	0.0034
marital	0.0018
loan	0.0010
default	0.0005

Detecting Significant Splits

Quite often it is necessary to measure the significance of a split in a decision tree, especially when the information gain is small, like in Table 2.2.

Let N_A and N_B be the number of class A and class B in the parent node. Let N_{AL} represent the number of class A going to the left child node, N_{BL} represent the number of class B going to the left child node, N_{AR} represent the number of class B going to the right child node, and N_{BR} represent the number of class B going to the right child node.

Let p_L and p_R denote the proportion of data going to the left and right node, respectively.

$$p_L = \frac{N_{AL} + N_{BL}}{N_A + N_B}$$

$$p_R = \frac{N_{AR} + N_{BR}}{N_A + N_B}$$

The following measure computes the significance of a split. In other words, it measures how much the split deviates from what would be expected in the random data.

$$K = \frac{\left(N'_{AL} - N_{AL}\right)^2}{N'_{AL}} + \frac{\left(N'_{BL} - N_{BL}\right)^2}{N'_{BL}} + \frac{\left(N'_{AR} - N_{AR}\right)^2}{N'_{AR}} + \frac{\left(N'_{BR} - N_{BR}\right)^2}{N'_{BR}}$$

where

$$N'_{AL} = N_A \times p_L$$

$N'_{BL} = N_B \times p_L$

$N'_{AR} = N_A \times p_R$

$N'_{BR} = N_B \times p_R$

If K is small, the information gain from the split is not significant. If K is big, it would suggest the information gain from the split is significant.

Take the first split of the decision tree in Figure 2.4 on variable *poutcome* for example. $N_A = 1789$, $N_B = 211$, $N_{AL} = 1763$, $N_{BL} = 179$, $N_{AR} = 26$, $N_{BR} = 32$.

Following are the proportions of data going to the left and right node.

$p_L = 1942/2000 = 0.971$ and $p_R = 58/2000 = 0.029$.

The N'_{AL}, N'_{BL}, N'_{AR}, and N'_{BR} represent the number of each class going to the left or right node if the data is random. Their values follow.

$N'_{AL} = 1737.119$, $N'_{BL} = 204.881$, $N'_{AR} = 51.881$ and $N'_{BR} = 6.119$

Therefore, $K = 126.0324$, which suggests the split on *poutcome* is significant.

After each split, the algorithm looks at all the records at a leaf node, and the information gain of each candidate attribute is calculated again over these records. The next split is on the attribute with the highest information gain. A record can only belong to one leaf node after all the splits, but depending on the implementation, an attribute may appear in more than one split of the tree. This process of partitioning the records and finding the most informative attribute is repeated until the nodes are pure enough, or there is insufficient information gain by splitting on more attributes. Alternatively, one can stop the growth of the tree when all the nodes at a leaf node belong to a certain class (for example, *subscribed* = yes) or all the records have identical attribute values.

In the previous bank marketing example, to keep it simple, the dataset only includes categorical variables. Assume the dataset now includes a continuous variable called *duration* –representing the number of seconds the last phone conversation with the bank lasted as part of the previous marketing campaign. A continuous variable needs to be divided into a disjoint set of regions with the highest information gain. A brute-force method is to consider every value of the continuous variable in the training data as a candidate split position. This brute-force method is computationally inefficient. To reduce the complexity, the training records can be sorted based on the duration, and the candidate splits can be identified by taking the midpoints between two adjacent sorted values. An examples is if the duration consists of sorted values {140, 160, 180, 200} and the candidate splits are 150, 170, and 190.

Figure 2.6 shows what the decision tree may look like when considering the *duration* attribute. The root splits into two partitions: those clients with $\text{duration} < 456$ seconds, and those with $\text{duration} \geq 456$ seconds. Note that for aesthetic purposes, labels for the *job* and *contact* attributes in the figure are abbreviated.

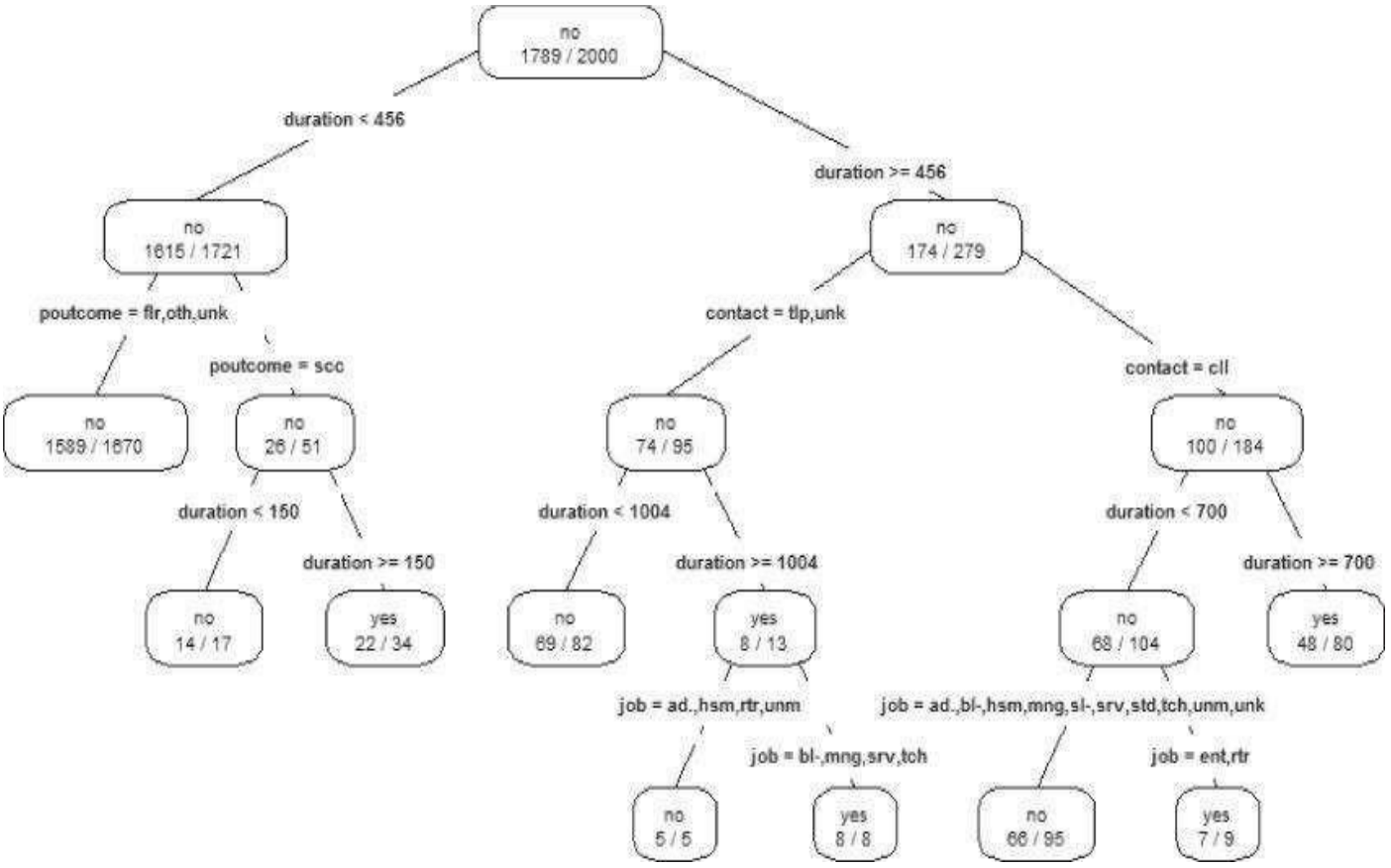


Figure 2.6 Decision tree with attribute duration

With the decision tree in Figure 2.6, it becomes trivial to predict if a new client is going to subscribe to the term deposit. For example, given the record of a new client shown in Table 2.3, the prediction is that this client will subscribe to the term deposit. The traversed paths in the decision tree are as follows.

- *duration* ≥ 456 *contact* = cll (cellular) *duration* < 700
- *job* = ent (entrepreneur), rtr (retired)

Table 2.3 Record of a New Client

Job M	rital E	ucation D	fault H	using L	an C	ntact D	ration P	utcome
retired	married	secondary	no	yes	No	cellular	598	unknown

2.1.3 Decision Tree Algorithms

Multiple algorithms exist to implement decision trees, and the methods of tree construction vary with different algorithms. Some popular algorithms include ID3 [7], C3.5[8], and CART [9].

ID3 Algorithm

ID3 (or Iterative Dichotomiser 3) [7] is one of the first decision tree algorithms, and it was

developed by John Ross Quinlan. Let A be a set of categorical input variables, P be the output variable (or the predicted class), and T be the training set. The ID3 algorithm is shown here.

```
1 ID3 (A, P, T)
2 if T∈ϕ
3  return ϕ
4 if all records in T have the same value for P
5  return a single node with that value
6 if A∈ϕ
7  return a single node with the most frequent value of P in T
8 Compute information gain for each attribute in A relative to T
9 Pick attribute D with the largest gain
10 Let {d1,d2…dm} be the values of attribute D
11 Partition T into {T1,T2…Tm} according to the values of D
12 return a tree with root D and branches labeled d1,d2…dm
   going respectively to trees ID3(A-{D}, P, T1),
   ID3(A-{D}, P, T2), … ID3(A-{D}, P, Tm)
```

C3.5

The C3.5 algorithm [8] introduces a number of improvements over the original ID3 algorithm. The C3.5 algorithm can handle missing data. If the training records contain unknown attribute values, the C3.5 evaluates the gain for an attribute by considering only the records where the attribute is defined.

Both categorical and continuous attributes are supported by C3.5. Values of a continuous variable are sorted and partitioned. For the corresponding records of each partition, the gain is calculated, and the partition that maximizes the gain is chosen for the next split.

The ID3 algorithm may construct a deep and complex tree, which would cause overfitting (Section 2.1.4). The C3.5 algorithm addresses the overfitting problem in ID3 by using a bottom-up technique called ***pruning*** to simplify the tree by removing the least visited nodes and branches.

CART

CART (or Classification And Regression Trees) [9] is often used as a generic acronym for the decision tree, although it is a specific implementation.

Similar to C3.5, CART can handle continuous attributes. Whereas C3.5 uses entropy-based criteria to rank tests, CART uses the Gini diversity index defined in Equation 2.5.

$$Gini_X = 1 - \sum_{\forall x \in X} P(x)^2 \quad 2.5$$

Whereas C3.5 employs stopping rules, CART constructs a sequence of subtrees, uses cross-validation to estimate the misclassification cost of each subtree, and chooses the one with the lowest cost.

2.1.4 Evaluating a Decision Tree

Decision trees use ***greedy algorithms***, in that they always choose the option that seems the best available at that moment. At each step, the algorithm selects which attribute to use for splitting the remaining records. This selection may not be the best overall, but it is guaranteed to be the best at that step. This characteristic reinforces the efficiency of decision trees. However, once a bad split is taken, it is propagated through the rest of the tree. To address this problem, an ensemble technique (such as random forest) may randomize the splitting or even randomize data and come up with a multiple tree structure. These trees then vote for each class, and the class with the most votes is chosen as the predicted class.

There are a few ways to evaluate a decision tree. First, evaluate whether the splits of the tree make sense. Conduct sanity checks by validating the decision rules with domain experts, and determine if the decision rules are sound.

Next, look at the depth and nodes of the tree. Having too many layers and obtaining nodes with few members might be signs of overfitting. In overfitting, the model fits the training set well, but it performs poorly on the new samples in the testing set. Figure 2.7 illustrates the performance of an overfit model. The *x*-axis represents the amount of data, and the *y*-axis represents the errors. The blue curve is the training set, and the red curve is the testing set. The left side of the gray vertical line shows that the model predicts well on the testing set. But on the right side of the gray line, the model performs worse and worse on the testing set as more and more unseen data is introduced.

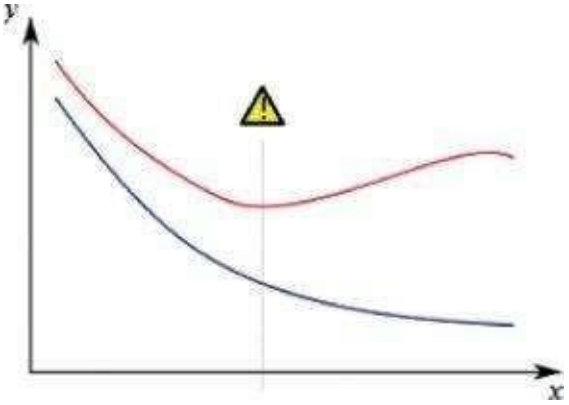


Figure 2.7 An overfit model describes the training data well but predicts poorly on unseen data

For decision tree learning, overfitting can be caused by either the lack of training data or the biased data in the training set. Two approaches [10] can help avoid overfitting in decision tree learning.

- Stop growing the tree early before it reaches the point where all the training data is perfectly classified.
- Grow the full tree, and then post-prune the tree with methods such as reduced-error pruning and rule-based post pruning.

Last, many standard diagnostics tools that apply to classifiers can help evaluate overfitting. These tools are further discussed in Section 2.3.

Decision trees are computationally inexpensive, and it is easy to classify the data. The outputs are easy to interpret as a fixed sequence of simple tests. Many decision tree algorithms are able to show the importance of each input variable. Basic measures, such as information gain, are provided by most statistical software packages.

Decision trees are able to handle both numerical and categorical attributes and are robust with redundant or correlated variables. Decision trees can handle categorical attributes with many distinct values, such as country codes for telephone numbers. Decision trees can also handle variables that have a nonlinear effect on the outcome, so they work better than linear models (for example, linear regression and logistic regression) for highly nonlinear problems. Decision trees naturally handle variable interactions. Every node in the tree depends on the preceding nodes in the tree.

In a decision tree, the decision regions are rectangular surfaces. Figure 2.8 shows an example of five rectangular decision surfaces (A, B, C, D, and E) defined by four values—$\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$—of two attributes ($x_1$ and x_2). The corresponding decision tree is on the right side of the figure. A decision surface corresponds to a leaf node of the tree, and it can be reached by traversing from the root of the tree following by a series of decisions according to the value of an attribute. The decision surface can only be axis-aligned for the decision tree.

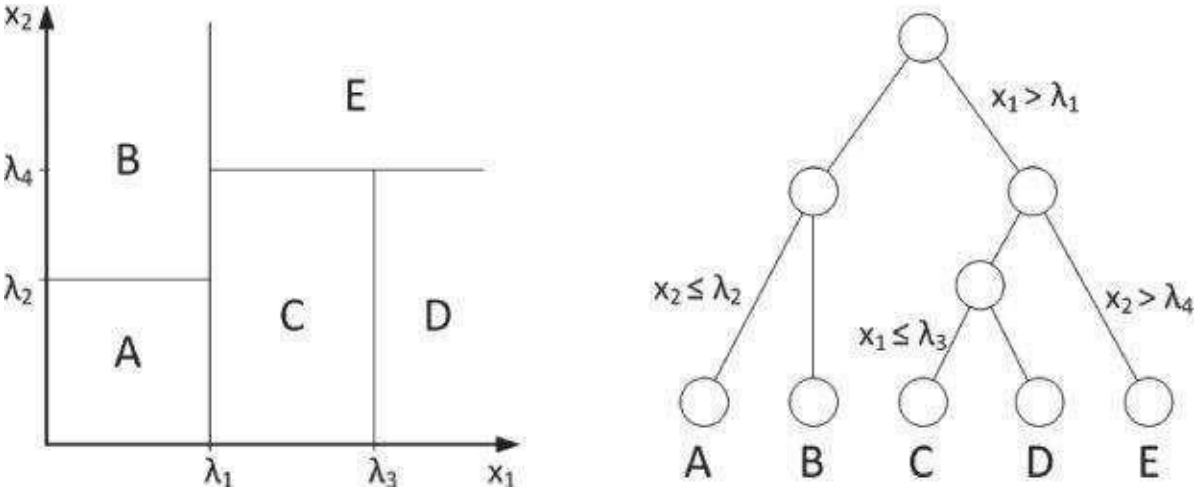


Figure 2.8 Decision surfaces can only be axis-aligned

The structure of a decision tree is sensitive to small variations in the training data. Although the dataset is the same, constructing two decision trees based on two different subsets may result in very different trees. If a tree is too deep, overfitting may occur, because each split reduces the training data for subsequent splits.

Decision trees are not a good choice if the dataset contains many irrelevant variables. This is different from the notion that they are robust with redundant variables and correlated variables. If the dataset contains redundant variables, the resulting decision tree ignores all

but one of these variables because the algorithm cannot detect information gain by including more redundant variables. On the other hand, if the dataset contains irrelevant variables and if these variables are accidentally chosen as splits in the tree, the tree may grow too large and may end up with less data at every split, where overfitting is likely to occur. To address this problem, feature selection can be introduced in the data preprocessing phase to eliminate the irrelevant variables.

Although decision trees are able to handle correlated variables, decision trees are not well suited when most of the variables in the training set are correlated, since overfitting is likely to occur. To overcome the issue of instability and potential overfitting of deep trees, one can combine the decisions of several randomized shallow decision trees—the basic idea of another classifier called random forest [4]—or use ensemble methods to combine several weak learners for better classification. These methods have been shown to improve predictive power compared to a single decision tree.

For binary decisions, a decision tree works better if the training dataset consists of records with an even probability of each result. In other words, the root of the tree has a 50% chance of either classification. This occurs by randomly selecting training records from each possible classification in equal numbers. It counteracts the likelihood that a tree will stump out early by passing purity tests because of bias in the training data.

When using methods such as logistic regression on a dataset with many variables, decision trees can help determine which variables are the most useful to select based on information gain. Then these variables can be selected for the logistic regression. Decision trees can also be used to prune redundant variables.

2.1.5 Decision Trees in R

In R, `rpart` is for modeling decision trees, and an optional package `rpart.plot` enables the plotting of a tree. The rest of this section shows an example of how to use decision trees in R with `rpart.plot` to predict whether to play golf given factors such as weather outlook, temperature, humidity, and wind.

In R, first set the working directory and initialize the packages.

```
setwd("c:/")
install.packages("rpart.plot") # install package rpart.plot library("rpart")
# load libraries
library("rpart.plot")
```

The working directory contains a comma-separated-value (CSV) file named `DTdata.csv`. The file has a header row, followed by 10 rows of training data.

```
Play,Outlook,Temperature,Humidity,Wind yes,rainy,cool,normal,FALSE
no,rainy,cool,normal,TRUE yes,overcast,hot,high,FALSE
no,sunny,mild,high,FALSE yes,rainy,cool,normal,FALSE
yes,sunny,cool,normal,FALSE yes,rainy,cool,normal,FALSE
yes,sunny,hot,normal,FALSE yes,overcast,mild,high,TRUE
no,sunny,mild,high,TRUE
```

The CSV file contains five attributes: *Play*, *Outlook*, *Temperature*, *Humidity*, and *Wind*. *Play* would be the output variable (or the predicted class), and *Outlook*, *Temperature*, *Humidity*, and *Wind* would be the input variables. In R, read the data from the CSV file in the working directory and display the content.

```
play_decision <- read.table("DTdata.csv",header=TRUE,sep=",")
play_decision
Play Outlook Temperature Humidity Wind
1 yes rainy  cool normal FALSE
2 no rainy  cool normal TRUE
3 yes overcast  hot  high FALSE
4 no sunny  mild  high FALSE
5 yes rainy  cool normal FALSE
6 yes sunny  cool normal FALSE
7 yes rainy  cool normal FALSE
8 yes sunny  hot normal FALSE
9 yes overcast  mild  high TRUE
10 no sunny  mild  high TRUE
```

Display a summary of *play_decision*.

```
summary(play_decision)
Play  Outlook Temperature Humidity Wind
no :3 overcast:2 cool:5  high :4 Mode :logical yes:7 rainy :4 hot :2
normal:6 FALSE:7
   sunny :4 mild:3     TRUE :3
            NA's :0
```

The rpart function builds a model of recursive partitioning and regression trees [9]. The following code snippet shows how to use the rpart function to construct a decision tree.

```
fit <- rpart(Play ~ Outlook + Temperature + Humidity + Wind,
    method="class",
    data=play_decision,
    control=rpart.control(minsplit=1),
    parms=list(split='information'))
```

The rpart function has four parameters. The first parameter, Play ~ Outlook + Temperature + Humidity + Wind, is the model indicating that attribute *Play* can be predicted based on attributes *Outlook*, *Temperature*, *Humidity*, and *Wind*. The second parameter, *method*, is set to "class" telling R it is building a classification tree. The third parameter, *data*, specifies the dataframe containing those attributes mentioned in the formula. The fourth parameter, *control*, is optional and controls the tree growth. In the preceding example, control=rpart.control(minsplit=1) requires that each node have at least one observation before attempting a split. The minsplit=1 makes sense for the small dataset, but for larger datasets minsplit could be set to 10% of the dataset size to combat overfitting. Besides minsplit, other parameters are available to control the construction of the decision tree. For example, rpart.control(maxdepth=10,cp=0.001) limits the depth of the tree to no more than 10, and a split must decrease the overall lack of fit by a factor of 0.001 before being attempted. The last parameter (parms) specifies the

purity measure being used for the splits. The value of **split** can be either **information** (for using the information gain) or gini (for using the Gini index).

Enter **summary(fit)** to produce a summary of the model built from **rpart**.

The output includes a summary of every node in the constructed decision tree. If a node is a leaf, the output includes both the predicted class label (*yes* or *no* for *Play*) and the class probabilities—**P(*Play*)**. The leaf nodes include node numbers 4, 5, 6, and 2. If a node is internal, the output in addition displays the number of observations that lead to each child node and the improvement that each attribute may bring for the next split. These internal nodes include numbers 1, 2, and 3.

```
summary(fit) Call:
rpart(formula = Play ~ Outlook + Temperature + Humidity + Wind, data =
play_decision, method = "class",
parms = list(split = "information"),
  control = rpart.control(minsplit = 1))
n= 10
   CP nsplit rel error xerror  xstd
1 0.3333333  0   1 1.000000 0.4830459
2 0.0100000  3   0 1.666667 0.5270463
Variable importance
  Wind  Outlook Temperature
   51   29   20
Node number 1: 10 observations, complexity param=0.3333333 predicted
class=yes expected loss=0.3 P(node) =1
class counts:  3  7 probabilities: 0.300 0.700
left son=2 (3 obs) right son=3 (7 obs) Primary splits:
  Temperature splits as RRL,  improve=1.3282860, (0 missing)
  Wind  < 0.5 to the right, improve=1.3282860, (0 missing)
  Outlook  splits as RLL,  improve=0.8161371, (0 missing)
  Humidity splits as LR,  improve=0.6326870, (0 missing)
Surrogate splits:
  Wind < 0.5 to the right, agree=0.8, adj=0.333, (0 split) Node number 2: 3
observations, complexity param=0.3333333 predicted class=no expected
loss=0.3333333 P(node) =0.3 class counts:  2  1
probabilities: 0.667 0.333
left son=4 (2 obs) right son=5 (1 obs) Primary splits:
  Outlook splits as R-L,  improve=1.9095430, (0 missing)
  Wind < 0.5 to the left, improve=0.5232481, (0 missing) Node number 3: 7
observations, complexity param=0.3333333 predicted class=yes expected
loss=0.1428571 P(node) =0.7 class counts:  1  6
probabilities: 0.143 0.857
left son=6 (1 obs) right son=7 (6 obs) Primary splits:
  Wind  < 0.5 to the right, improve=2.8708140, (0 missing)
  Outlook  splits as RLR,  improve=0.6214736, (0 missing)
  Temperature splits as LR-,  improve=0.3688021, (0 missing)
  Humidity splits as RL,  improve=0.1674470, (0 missing) Node number 4: 2
observations
predicted class=no expected loss=0 P(node) =0.2 class counts:  2  0
probabilities: 1.000 0.000
```

```
Node number 5: 1 observations
predicted class=yes expected loss=0 P(node) =0.1 class counts:  0  1
probabilities: 0.000 1.000
Node number 6: 1 observations
predicted class=no expected loss=0 P(node) =0.1 class counts:  1  0
probabilities: 1.000 0.000
Node number 7: 6 observations
predicted class=yes expected loss=0 P(node) =0.6 class counts:  0  6
probabilities: 0.000 1.000
```

The output produced by the summary is difficult to read and comprehend. The `rpart.plot()` function from the `rpart.plot` package can visually represent the output in a decision tree. Enter the following command to see the help file of `rpart.plot`:

```
?rpart.plot
```

Enter the following R code to plot the tree based on the model being built. The resulting tree is shown in Figure 2.9. Each node of the tree is labeled as either yes or no referring to the *Play* action of whether to play outside. Note that, by default, R has converted the values of *Wind* (True/False) into numbers.

```
rpart.plot(fit, type=4, extra=1)
```

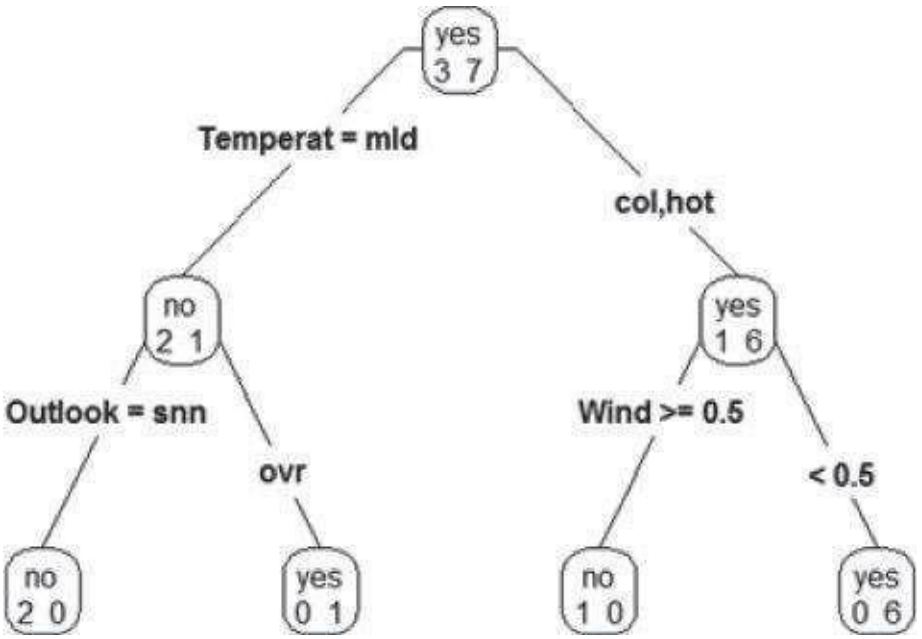


Figure 2.9 A decision tree built from DTdata.csv

The decisions in Figure 2.9 are abbreviated. Use the following command to spell out the full names and display the classification rate at each node.

```
rpart.plot(fit, type=4, extra=2, clip.right.labs=FALSE,
   varlen=0, faclen=0)
```

The decision tree can be used to predict outcomes for new datasets. Consider a testing set that contains the following record.

```
Outlook=‖rainy‖, Temperature=‖mild‖, Humidity=‖high‖, Wind=FALSE
```

The goal is to predict the play decision of this record. The following code loads the data into R as a data frame *newdata*. Note that the training set does not contain this case.

```
newdata <- data.frame(Outlook="rainy", Temperature="mild",
      Humidity="high", Wind=FALSE)
newdata
Outlook Temperature Humidity Wind
1 rainy  mild  high FALSE
```

Next, use the *predict* function to generate predictions from a fitted *rpart* object. The format of the predict function follows.

```
predict(object, newdata = list(),
  type = c("vector", "prob", "class", "matrix"))
```

Parameter *type* is a character string denoting the type of the predicted value. Set it to either prob or class to predict using a decision tree model and receive the result as either the class probabilities or just the class. The output shows that one instance is classified as *Play*=no, and zero instances are classified as *Play*=yes. Therefore, in both cases, the decision tree predicts that the play decision of the testing set is not to play.

```
predict(fit,newdata=newdata,type="prob")
no yes
1  1  0 predict(fit,newdata=newdata,type="class")
1 no
Levels: no yes
```

2.2 Naïve Bayes

Naïve Bayes is a probabilistic classification method based on Bayes' theorem (or Bayes' law) with a few tweaks. Bayes' theorem gives the relationship between the probabilities of two events and their conditional probabilities. Bayes' law is named after the English mathematician Thomas Bayes.

A naïve Bayes classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of other features. For example, an object can be classified based on its attributes such as shape, color, and weight. A reasonable classification for an object that is spherical, yellow, and less than 60 grams in weight may be a tennis ball. Even if these features depend on each other or upon the existence of the other features, a naïve Bayes classifier considers all these properties to contribute *independently* to the probability that the object is a tennis ball.

The input variables are generally categorical, but variations of the algorithm can accept continuous variables. There are also ways to convert continuous variables into categorical ones. This process is often referred to as the ***discretization of continuous variables***. In the tennis ball example, a continuous variable such as weight can be grouped into intervals to be converted into a categorical variable. For an attribute such as *income*, the attribute can be converted into categorical values as shown below.

- **Low Income:** income < $10,000
- **Working Class:** $10,000 ≤ income < $50,000
- **Middle Class:** $50,000 ≤ income < $1,000,000
- **Upper Class:** income ≥ $1,000,000

The output typically includes a class label and its corresponding probability score. The probability score is not the true probability of the class label, but it‘s proportional to the true probability. As shown later in the chapter, in most implementations, the output includes the log probability for the class, and class labels are assigned based on the highest values.

Because naïve Bayes classifiers are easy to implement and can execute efficiently even without prior knowledge of the data, they are among the most popular algorithms for classifying text documents. Spam filtering is a classic use case of naïve Bayes text classification. Bayesian spam filtering has become a popular mechanism to distinguish spam e-mail from legitimate e-mail. Many modern mail clients implement variants of Bayesian spam filtering.

Naïve Bayes classifiers can also be used for fraud detection [11]. In the domain of auto insurance, for example, based on a training set with attributes such as driver‘s rating, vehicle age, vehicle price, historical claims by the policy holder, police report status, and claim genuineness, naïve Bayes can provide probability-based classification of whether a new claim is genuine [12].

2.2.1 Bayes’ Theorem

The ***conditional probability*** of event C occurring, given that event A has already occurred, is denoted as $P(C|A)$, which can be found using the formula in Equation 2.6.

$$P(C|A) = \frac{P(A \cap C)}{P(A)} \quad \textbf{2.6}$$

Equation 2.7 can be obtained with some minor algebra and substitution of the conditional probability:

$$P(C|A) = \frac{P(A|C) \cdot P(C)}{P(A)} \quad \textbf{2.7}$$

where C is the class label $C \in \{c_1, c_2, \ldots c_n\}$ and A is the observed attributes $A = \{a_1, a_2, \ldots a_m\}$. Equation 2.7 is the most common form of the ***Bayes’ theorem***.

Mathematically, Bayes‘ theorem gives the relationship between the probabilities of *C* and *A*, $P(C)$ and $P(A)$ and the conditional probabilities of *C* given *A* and *A* given *C*, namely $P(C|A)$ and $P(A|C)$.

Bayes‘ theorem is significant because quite often $P(C|A)$ is much more difficult to compute than $P(A|C)$ and $P(C)$ from the training data. By using Bayes‘ theorem, this problem can be circumvented.

An example better illustrates the use of Bayes' theorem. John flies frequently and likes to upgrade his seat to first class. He has determined that if he checks in for his flight at least two hours early, the probability that he will get an upgrade is 0.75; otherwise, the probability that he will get an upgrade is 0.35. With his busy schedule, he checks in at least two hours before his flight only 40% of the time. Suppose John did not receive an upgrade on his most recent attempt. What is the probability that he did not arrive two hours early?

Let C = {John arrived at least two hours early}, and A = {John received an upgrade}, then $\neg C$ = {John did not arrive two hours early}, and $\neg A$ = {John did not receive an upgrade}.

John checked in at least two hours early only 40% of the time, or $P(C)=0.4$. Therefore, $P(\neg C)=1-P(C)=0.6$.

The probability that John received an upgrade given that he checked in early is 0.75, or $P(A|C)=0.75$.

The probability that John received an upgrade given that he did not arrive two hours early is 0.35, or $P(A|\neg C)=0.35$. Therefore, $P(\neg A|\neg C)=0.65$.

The probability that John received an upgrade $P(A)$ can be computed as shown in Equation 2.8.

$$\begin{aligned} P(A) &= P(A \cap C) + P(A \cap \neg C) \\ &= P(C) \cdot P(A|C) + P(\neg C) \cdot P(A|\neg C) \\ &= 0.4 \times 0.75 + 0.6 \times 0.35 \\ &= 0.51 \end{aligned} \qquad \textbf{2.8}$$

Thus, the probability that John did not receive an upgrade $P(\neg A)=0.49$. Using Bayes' theorem, the probability that John did not arrive two hours early given that he did not receive his upgrade is shown in Equation 2.9.

$$\begin{aligned} P(\neg C|\neg A) &= \frac{P(\neg A|\neg C) \cdot P(\neg C)}{P(\neg A)} \\ &= \frac{0.65 \times 0.6}{0.49} \approx 0.796 \end{aligned} \qquad \textbf{2.9}$$

Another example involves computing the probability that a patient carries a disease based on the result of a lab test. Assume that a patient named Mary took a lab test for a certain disease and the result came back positive. The test returns a positive result in 95% of the cases in which the disease is actually present, and it returns a positive result in 6% of the cases in which the disease is not present. Furthermore, 1% of the entire population has this disease. What is the probability that Mary actually has the disease, given that the test is positive?

Let C = {having the disease} and A = {testing positive}. The goal is to solve the probability of having the disease, given that Mary has a positive test result, $P(C|A)$. From the problem description, $P(C)=0.01$, $P(\neg C)=0.99$, $P(A|C)=0.95$ and $P(A|\neg C)=0.06$.

Bayes' theorem defines $P(C|A)=P(A|C)P(C)/P(A)$. The probability of testing positive, that is $P(A)$, needs to be computed first. That computation is shown in Equation 2.10.

$$\begin{aligned} P(A) &= P(A\cap C)+P(A\cap \neg C) \\ &= P(C)\cdot P(A|C)+P(\neg C)\cdot P(A|\neg C) \\ &= 0.01\times 0.95+0.99\times 0.06=0.0689 \end{aligned} \quad \textbf{2.10}$$

According to Bayes' theorem, the probability of having the disease, given that Mary has a positive test result, is shown in Equation 2.11.

$$P(C|A)=\frac{P(A|C)P(C)}{P(A)}=\frac{0.95\times 0.01}{0.0689}\approx 0.1379 \quad \textbf{2.11}$$

That means that the probability of Mary actually having the disease given a positive test result is only 13.79%. This result indicates that the lab test may not be a good one. The likelihood of having the disease was 1% when the patient walked in the door and only 13.79% when the patient walked out, which would suggest further tests.

A more general form of Bayes' theorem assigns a classified label to an object with multiple attributes $A=\{a_1,a_2,\ldots,a_m\}$ such that the label corresponds to the largest value of $P(c_i|A)$. The probability that a set of attribute values A (composed of m variables $a_1,a_2,\ldots,a_m$) should be labeled with a classification label equals the probability that the set of variables $a_1,a_2,\ldots,a_m$ given c_i is true, times the probability of c_i divided by the probability of $a_1,a_2,\ldots,a_m$. Mathematically, this is shown in Equation 2.12.

$$P(c_i|A)=\frac{P(a_1,a_2,\ldots,a_m|c_i)\cdot P(c_i)}{P(a_1,a_2,\ldots,a_m)},\ i=1,2,\ldots n \quad \textbf{2.12}$$

Consider the bank marketing example presented in Section 2.1 on predicting if a customer would subscribe to a term deposit. Let A be a list of attributes {*job*, *marital*, *education*, *default*, *housing*, *loan*, *contact*, *poutcome*}. According to Equation 2.12, the problem is essentially to calculate $P(c_i|A)$, where $c_i\in\{subscribed=yes, subscribed=no\}$.

2.2.2 Naïve Bayes Classifier

With two simplifications, Bayes' theorem can be extended to become a naïve Bayes classifier.

The first simplification is to use the conditional independence assumption. That is, each attribute is conditionally independent of every other attribute given a class label C. See Equation 2.13.

$$P(a_1,a_2,\ldots,a_m|c_i)=P(a_1|c_i)P(a_2|c_i)\cdots P(a_m|c_i)=\prod_{j=1}^{m}P(a_j|c_i) \quad \textbf{2.13}$$

Therefore, this naïve assumption simplifies the computation of $P(a_1,a_2,\ldots,a_m|c_i)$.

The second simplification is to ignore the denominator $P(a_1,a_2,\ldots,a_m)$. Because $P(a_1,a_2,\ldots,a_m)$ appears in the denominator of $P(c_i|A)$ for all values of i, removing the denominator will have no impact on the relative probability scores and will simplify calculations.

Naïve Bayes classification applies the two simplifications mentioned earlier and, as a result, $P(c_i|a_1,a_2,\ldots,a_m)$ is proportional to the product of $P(a_j|c_i)$ times $P(c_i)$. This is shown in Equation 2.13.

$$P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i) \qquad i = 1,2,\ldots n \tag{2.14}$$

The mathematical symbol $\propto$ indicates that the LHS $P(c_i|A)$ is directly proportional to the RHS.

Section 2.1 has introduced a bank marketing dataset (Figure 2.3). This section shows how to use the naïve Bayes classifier on this dataset to predict if the clients would subscribe to a term deposit.

Building a naïve Bayes classifier requires knowing certain statistics, all calculated from the training set. The first requirement is to collect the probabilities of all class labels, $P(c_i)$. In the presented example, these would be the probability that a client will subscribe to the term deposit and the probability the client will not. From the data available in the training set, $P(subscribed = yes) \approx 0.11$ and $P(subscribed = no) \approx 0.89$.

The second thing the naïve Bayes classifier needs to know is the conditional probabilities of each attribute a_j given each class label c_i, namely $P(a_j|c_i)$. The training set contains several attributes: *job*, *marital*, *education*, *default*, *housing*, *loan*, *contact*, and *poutcome*. For each attribute and its possible values, computing the conditional probabilities given $subscribed = yes$ or $subscribed = no$ is required. For example, relative to the *marital* attribute, the following conditional probabilities are calculated.

$P(single|subscribed = yes) \approx 0.35$

$P(married|subscribed = yes) \approx 0.53$

$P(divorced|subscribed = yes) \approx 0.12$

$P(single|subscribed = no) \approx 0.28$

$P(married|subscribed = no) \approx 0.61$

$P(divorced|subscribed = no) \approx 0.11$

After training the classifier and computing all the required statistics, the naïve Bayes classifier can be tested over the testing set. For each record in the testing set, the naïve Bayes classifier assigns the classifier label c_i that maximizes $P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i)$.

Table 2.4 contains a single record for a client who has a career in management, is married, holds a secondary degree, has credit not in default, has a housing loan but no personal loans, prefers to be contacted via cellular, and whose outcome of the previous marketing campaign contact was a success. Is this client likely to subscribe to the term deposit?

Table 2.4 Record of an Additional Client

Job	Marital	Education	Default	Housing	Loan	Contact	Poutcome
management	married	secondary	no	yes	no	cellular	Success

The conditional probabilities shown in Table 2.5 can be calculated after building the classifier with the training set.

Table 2.5 Compute Conditional Probabilities for the New Record

j	a_j	$P(a_j \mid \text{subscribed} = \text{yes})$	$P(a_j \mid \text{subscribed} = \text{no})$
1	job = management	0.22	0.21
2	marital = married	0.53	0.61
3	education = secondary	0.46	0.51
4	default = no	0.99	0.98
5	housing = yes	0.35	0.57
6	loan = no	0.90	0.85
7	contact = cellular	0.85	0.62
8	poutcome = success	0.15	0.01

Because $P(c_i|a_1,a_2,\ldots,a_m)$ is proportional to the product of $P(a_j|c_i)(j\in[1,m])$ times (c_i), the naïve Bayes classifier assigns the class label c_i which results in the greatest value over all j. Thus, $P(c_i|a_1,a_2,\ldots,a_m)$ is

computed for each c_i with $P(c_i|A)\propto P(c_i)\cdot\prod_{j=1}^{m} P(a_j|c_i)$.

For A = {management, married, secondary, no, yes, no, cellular, success},

$$P(yes|A)\propto 0.11\cdot(0.22\cdot 0.53\cdot 0.46\cdot 0.99\cdot 0.35\cdot 0.90\cdot 0.85\cdot 0.15)\approx 0.00023$$

$$P(no|A)\propto 0.89\cdot(0.21\cdot 0.61\cdot 0.51\cdot 0.98\cdot 0.57\cdot 0.85\cdot 0.62\cdot 0.01)\approx 0.00017$$

Because $P(subscribed = yes|A) > P(subscribed = no|A)$, the client shown in Table 2.4 is assigned with the label $subscribed = yes$. That is, the client is classified as likely to subscribe to the term deposit.

Although the scores are small in magnitude, it is the ratio of $P(yes|A)$ and $P(no|A)$ that matters. In fact, the scores of $P(yes|A)$ and $P(no|A)$ are not the true probabilities but are only proportional to the true probabilities, as shown in Equation 2.13. After all, if the scores were indeed the true probabilities, the sum of $P(yes|A)$ and $P(no|A)$ would be equal to one.

When looking at problems with a large number of attributes, or attributes with a high number of levels, these values can become very small in magnitude (close to zero), resulting in even smaller differences of the scores. This is the problem of ***numerical underflow***, caused by multiplying several probability values that are close to zero. A way to alleviate the problem is to compute the logarithm of the products, which is equivalent to the summation of the logarithm of the probabilities. Thus, the naïve Bayes formula can be

rewritten as shown in Equation 2.15.

$$P(c_i|A) \propto \log P(c_i) + \sum_{j=1}^{m} \log P(a_j|c_i) \qquad i = 1,2,\ldots n \tag{2.15}$$

Although the risk of underflow may increase as the number of attributes increases, the use of logarithms is usually applied regardless of the number of attribute dimensions.

2.2.3 Smoothing

If one of the attribute values does not appear with one of the class labels within the training set, the corresponding $P(a_j|c_i)$ will equal zero. When this happens, the resulting $P(c_i|A)$ from multiplying all the $P(a_j|c_i)(j \in [1,m])$ immediately becomes zero regardless of how large some of the conditional probabilities are. Therefore overfitting occurs. Smoothing techniques can be employed to adjust the probabilities of $P(a_j|c_i)$ and to ensure a nonzero value of $P(c_i|A)$. A smoothing technique assigns a small nonzero probability to rare events not included in the training dataset. Also, the smoothing addresses the possibility of taking the logarithm of zero that may occur in Equation 2.15.

There are various smoothing techniques. Among them is the ***Laplace smoothing*** (or add-one) technique that pretends to see every outcome once more than it actually appears. This technique is shown in Equation 2.16.

$$P^*(x) = \frac{count(x)+1}{\Sigma_x[count(x)+1]} \tag{2.16}$$

For example, say that 100 clients subscribe to the term deposit, with 20 of them single, 70 married, and 10 divorced. The ―raw‖ probability is $P(single\,|\,subscribed = yes) = 20/100 = 0.2$. With Laplace smoothing adding one to the counts, the adjusted probability becomes $P(single\,|\,subscribed = yes) = (20+1)/[(20+1)+(70+1)+(10+1)] \approx 0.2039$.

One problem of the Laplace smoothing is that it may assign too much probability to unseen events. To address this problem, Laplace smoothing can be generalized to use ε instead of 1, where typically $\varepsilon \in [0,1]$. See Equation 2.12.

$$P^{**}(x) = \frac{count(x)+\varepsilon}{\Sigma_x[count(x)+\varepsilon]} \tag{2.17}$$

Smoothing techniques are available in most standard software packages for naïve Bayes classifiers. However, if for some reason (like performance concerns) the naïve Bayes classifier needs to be coded directly into an application, the smoothing and logarithm calculations should be incorporated into the implementation.

2.2.4 Diagnostics

Unlike logistic regression, naïve Bayes classifiers can handle missing values. Naïve Bayes is also robust to irrelevant variables—variables that are distributed among all the classes whose effects are not pronounced.

The model is simple to implement even without using libraries. The prediction is based on counting the occurrences of events, making the classifier efficient to run. Naïve Bayes is computationally efficient and is able to handle high-dimensional data efficiently. Related research shows that the naive Bayes classifier in many cases is competitive with other learning algorithms, including decision trees and neural networks. In some cases naïve Bayes even outperforms other methods. Unlike logistic regression, the naïve Bayes classifier can handle categorical variables with many levels. Recall that decision trees can handle categorical variables as well, but too many levels may result in a deep tree. The naïve Bayes classifier overall performs better than decision trees on categorical values with many levels. Compared to decision trees, naïve Bayes is more resistant to overfitting, especially with the presence of a smoothing technique.

Despite the benefits of naïve Bayes, it also comes with a few disadvantages. Naïve Bayes assumes the variables in the data are conditionally independent. Therefore, it is sensitive to correlated variables because the algorithm may double count the effects. As an example, assume that people with low income and low credit tend to default. If the task is to score ―default‖ based on both income and credit as two separate attributes, naïve Bayes would experience the double-counting effect on the default outcome, thus reducing the accuracy of the prediction.

Although probabilities are provided as part of the output for the prediction, naïve Bayes classifiers in general are not very reliable for probability estimation and should be used only for assigning class labels. Naïve Bayes in its simple form is used only with categorical variables. Any continuous variables should be converted into a categorical variable with the process known as discretization, as shown earlier. In common statistical software packages, however, naïve Bayes is implemented in a way that enables it to handle continuous variables as well.

2.2.5 Naïve Bayes in R

This section explores two methods of using the naïve Bayes classifier in R. The first method is to build from scratch by manually computing the probability scores, and the second method is to use the *naiveBayes* function from the *e1071* package. The examples show how to use naïve Bayes to predict whether employees would enroll in an onsite educational program.

In R, first set up the working directory and initialize the packages.

```
setwd("c:/")
install.packages("e1071") # install package e1071 library(e1071) # load the library
```

The working directory contains a CSV file (`sample1.csv`). The file has a header row, followed by 14 rows of training data. The attributes include *Age*, *Income*, *JobSatisfaction*, and *Desire*. The output variable is *Enrolls*, and its value is either `Yes` or `No`. Full content of the CSV file is shown next.

```
Age,Income,JobSatisfaction,Desire,Enrolls
```

```
<=30,High,No,Fair,No
<=30,High,No,Excellent,No
31 to 40,High,No,Fair,Yes
>40,Medium,No,Fair,Yes
>40,Low,Yes,Fair,Yes
>40,Low,Yes,Excellent,No
31 to 40,Low,Yes,Excellent,Yes
<=30,Medium,No,Fair,No
<=30,Low,Yes,Fair,Yes
>40,Medium,Yes,Fair,Yes
<=30,Medium,Yes,Excellent,Yes
31 to 40,Medium,No,Excellent,Yes
31 to 40,High,Yes,Fair,Yes
>40,Medium,No,Excellent,No
<=30,Medium,Yes,Fair,
```

The last record of the CSV is used later for illustrative purposes as a test case. Therefore, it does not include a value for the output variable *Enrolls*, which should be predicted using the naïve Bayes classifier built from the training set.

Execute the following R code to read data from the CSV file.

```
# read the data into a table from the file
sample <- read.table("sample1.csv",header=TRUE,sep=",")
# define the data frames for the NB classifier traindata <-
as.data.frame(sample[1:14,]) testdata <- as.data.frame(sample[15,])
```

Two data frame objects called *traindata* and *testdata* are created for the naïve Bayes classifier. Enter *traindata* and *testdata* to display the data frames.

The two data frames are printed on the screen as follows.

```
traindata
   Age Income JobSatisfaction Desire Enrolls
1   <=30   High           No   Fair      No
2   <=30   High           No Excellent     No
3  31 to 40  High           No   Fair      Yes
4   >40   Medium          No   Fair      Yes
5   >40   Low           Yes   Fair      Yes
6   >40   Low           Yes Excellent     No
7  31 to 40  Low           Yes Excellent     Yes
8   <=30   Medium          No   Fair      No
9   <=30   Low           Yes   Fair      Yes
10   >40   Medium          Yes   Fair      Yes
11   <=30   Medium          Yes Excellent     Yes
12  31 to 40  Medium          No Excellent     Yes
13  31 to 40  High           Yes   Fair      Yes
14   >40   Medium          No Excellent     No testdata
Age Income JobSatisfaction Desire Enrolls
15  <=30   Medium          Yes Fair
```

The first method shown here is to build a naïve Bayes classifier from scratch by manually computing the probability scores. The first step in building a classifier is to compute the

prior probabilities of the attributes, including *Age*, *Income*, *JobSatisfaction*, and *Desire*. According to the naïve Bayes classifier, these attributes are conditionally independent. The dependent variable (output variable) is *Enrolls*.

Compute the prior probabilities $P(c_i)$ for *Enrolls*, where $c_i \in C$ and $C = \{Yes, No\}$.

```
tprior <- table(traindata$Enrolls)
tprior
  No Yes
0 5 9
tprior <- tprior/sum(tprior)
tprior
     No  Yes
0.0000000 0.3571429 0.6428571
```

The next step is to compute conditional probabilities $P(A|C)$, where $A = \{Age, Income, JobSatisfaction, Desire\}$ and $C = \{Yes, No\}$. Count the number of ―No‖ and ―Yes‖ entries for each *Age* group, and normalize by the total number of ―No‖ and ―Yes‖ entries to get the conditional probabilities.

```
ageCounts <- table(traindata[,c("Enrolls", "Age")])
ageCounts
  Age
Enrolls <=30 >40 31 to 40
   0 0  0
No  3 2  0
Yes  2 3  4
ageCounts <- ageCounts/rowSums(ageCounts)
ageCounts
  Age
Enrolls  <=30  >40 31 to 40
No 0.6000000 0.4000000 0.0000000
Yes 0.2222222 0.3333333 0.4444444
```

Do the same for the other attributes including *Income*, *JobSatisfaction*, and *Desire*.

```
incomeCounts <- table(traindata[,c("Enrolls", "Income")]) incomeCounts <-
incomeCounts/rowSums(incomeCounts) incomeCounts
  Income
Enrolls  High  Low Medium
No 0.4000000 0.2000000 0.4000000
Yes 0.2222222 0.3333333 0.4444444
jsCounts <- table(traindata[,c("Enrolls", "JobSatisfaction")])
jsCounts <- jsCounts/rowSums(jsCounts)
jsCounts
  Jobsatisfaction
Enrolls  No  Yes
No 0.8000000 0.2000000
Yes 0.3333333 0.6666667
desireCounts <- table(traindata[,c("Enrolls", "Desire")]) desireCounts <-
desireCounts/rowSums(desireCounts) desireCounts
  Desire
Enrolls Excellent  Fair
```

```
No 0.6000000 0.4000000
Yes 0.3333333 0.6666667
```

According to Equation 2.7, probability $P(c_i|A)$ is determined by the product of $P(a_j|c_i)$ times the (c_i) where $c_1 = \text{Yes}$ and $c_2 = \text{No}$. The larger value of $P(\text{Yes}|A)$ and $P(\text{No}|A)$ determines the predicted result of the output variable. Given the test data, use the following code to predict the *Enrolls*.

```
prob_yes <-
    ageCounts[-Yes‖,testdata[,c(-Age‖)]]*
    incomeCounts[-Yes‖,testdata[,c(-Income‖)]]*
    jsCounts[-Yes‖,testdata[,c(-JobSatisfaction‖)]]*
    desireCounts[-Yes‖,testdata[,c(-Desire‖)]]*
    tprior[-Yes‖]
prob_no <-
    ageCounts[-No‖,testdata[,c(-Age‖)]]*
    incomeCounts[-No‖,testdata[,c(-Income‖)]]*
    jsCounts[-No‖,testdata[,c(-JobSatisfaction‖)]]*
    desireCounts[-No‖,testdata[,c(-Desire‖)]]*
    tprior[-No‖]
max(prob_yes,prob_no)
```

As shown below, the predicted result of the test set is *Enrolls*=Yes.

```
prob_yes
   Yes
0.02821869 prob_no
    No
0.006857143 max(prob_yes, prob_no) [1] 0.02821869
```

The *e1071* package in R has a built-in *naiveBayes* function that can compute the conditional probabilities of a categorical class variable given independent categorical predictor variables using the Bayes rule. The function takes the form of naiveBayes(formula, data,…), where the arguments are defined as follows.

formula: A formula of the form class ~ x1 + x2 + … assuming x1, x2… are conditionally independent

data: A data frame of factors

Use the following code snippet to execute the model and display the results.

```
model <- naiveBayes(Enrolls ~ Age+Income+JobSatisfaction+Desire,
      traindata)
# display model model
```

The output that follows shows that the probabilities of *model* match the probabilities from the previous method. The default laplace=laplace setting enables the Laplace smoothing.

```
Naive Bayes Classifier for Discrete Predictors
```

```
Call:
naiveBayes.default(x = X, y = Y, laplace = laplace) A-priori probabilities:
Y     No  Yes
0.0000000 0.3571429 0.6428571
Conditional probabilities:
  Age
Y   <=30  >40 31 to 40
No 0.6000000 0.4000000 0.0000000
Yes 0.2222222 0.3333333 0.4444444
  Income
Y   High  Low Medium
No 0.4000000 0.2000000 0.4000000
Yes 0.2222222 0.3333333 0.4444444
  JobSatisfaction
Y   No  Yes
No 0.8000000 0.2000000
Yes 0.3333333 0.6666667
  Desire
Y  Excellent  Fair
No 0.6000000 0.4000000
Yes 0.3333333 0.6666667
```

Next, predicting the outcome of *Enrolls* with the *testdata* shows the result is *Enrolls*=Yes.

```
# predict with testdata
results <- predict (model,testdata)
# display results results
[1] Yes
Levels: No Yes
```

The *naiveBayes* function accepts a Laplace parameter that allows the customization of the ε value of Equation 2.17 for the Laplace smoothing. The code that follows shows how to build a naïve Bayes classifier with Laplace smoothing $\varepsilon = 0.01$ for prediction.

```
# use the NB classifier with Laplace smoothing
model1 = naiveBayes(Enrolls ~., traindata, laplace=.01)
# display model model1
Naive Bayes Classifier for Discrete Predictors
Call:
naiveBayes.default(x = X, y = Y, laplace = laplace) A-priori probabilities:
Y
     No  Yes
0.0000000 0.3571429 0.6428571
Conditional probabilities:
  Age
Y   <=30    >40 31 to 40
  0.333333333 0.333333333 0.333333333
No 0.598409543 0.399602386 0.001988072
Yes 0.222591362 0.333333333 0.444075305
  Income
Y   High  Low Medium
  0.3333333 0.3333333 0.3333333
```

```
No 0.3996024 0.2007952 0.3996024
Yes 0.2225914 0.3333333 0.4440753
  JobSatisfaction
Y   No  Yes
  0.5000000 0.5000000
No 0.7988048 0.2011952
Yes 0.3337029 0.6662971
  Desire
Y  Excellent  Fair
  0.5000000 0.5000000
No 0.5996016 0.4003984
Yes 0.3337029 0.6662971
```

The test case is again classified as *EnrolIs*=Yes.

```
# predict with testdata
results1 <- predict (model1,testdata)
# display results results1
[1] Yes
Levels: No Yes
```

2.3 Diagnostics of Classifiers

So far, this book has talked about three classifiers: logistic regression, decision trees, and naïve Bayes. These three methods can be used to classify instances into distinct groups according to the similar characteristics they share. Each of these classifiers faces the same issue: how to evaluate if they perform well.

A few tools have been designed to evaluate the performance of a classifier. Such tools are not limited to the three classifiers in this book but rather serve the purpose of assessing classifiers in general.

A ***confusion matrix*** is a specific table layout that allows visualization of the performance of a classifier.

Table 2.6 shows the confusion matrix for a two-class classifier. ***True positives*** (TP) are the number of positive instances the classifier correctly identified as positive. ***False positives*** (FP) are the number of instances in which the classifier identified as positive but in reality are negative. ***True negatives*** (TN) are the number of negative instances the classifier correctly identified as negative. ***False negatives*** (FN) are the number of instances classified as negative but in reality are positive. In a two-class classification, a preset threshold may be used to separate positives from negatives. TP and TN are the correct guesses. A good classifier should have large TP and TN and small (ideally zero) numbers for FP and FN.

Table 2.6 Confusion Matrix

Predicted Class	
Positive	**Negative**

Actual Class	Positive	True Positives (TP)	False Negatives (FN)
Negative	False Positives (FP)	True Negatives (TN)	

In the bank marketing example, the training set includes 2,000 instances. An additional 100 instances are included as the testing set. Table 2.7 shows the confusion matrix of a naïve Bayes classifier on 100 clients to predict whether they would subscribe to the term deposit. Of the 11 clients who subscribed to the term deposit, the model predicted 3 subscribed and 8 not subscribed. Similarly, of the 89 clients who did not subscribe to the term, the model predicted 2 subscribed and 87 not subscribed. All correct guesses are located from top left to bottom right of the table. It's easy to visually inspect the table for errors, because they will be represented by any nonzero values outside the diagonal.

Table 2.7 Confusion Matrix of Naïve Bayes from the Bank Marketing Example

Predicted Cla	**Total**			
Subscribe N	**t Subscribed**			
Actual Class	Subscribed	3	8	11
Not Subscribed	2	87	89	
Total		5	95	100

The ***accuracy*** (or the ***overall success rate***) is a metric defining the rate at which a model has classified the records correctly. It is defined as the sum of TP and TN divided by the total number of instances, as shown in Equation 2.18.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad \textbf{2.18}$$

A good model should have a high accuracy score, but having a high accuracy score alone does not guarantee the model is well established. The following measures can be introduced to better evaluate the performance of a classifier.

As seen in Chapter 6, the ***true positive rate*** (TPR) shows what percent of positive instances the classifier correctly identified. It's also illustrated in Equation 2.19.

$$TPR = \frac{TP}{TP + FN} \quad \textbf{2.19}$$

The ***false positive rate*** (FPR) shows what percent of negatives the classifier marked as positive. The FPR is also called the ***false alarm rate*** or the ***type I error rate*** and is shown in Equation 2.20.

$$FPR = \frac{FP}{FP + TN} \quad \textbf{2.20}$$

The ***false negative rate*** (FNR) shows what percent of positives the classifier marked as negatives. It is also known as the ***miss rate*** or ***type II error rate*** and is shown in Equation

2.21. Note that the sum of TPR and FNR is 1.

$$FNR = \frac{FN}{TP + FN} \quad (2.21)$$

A well-performed model should have a high TPR that is ideally 1 and a low FPR and FNR that are ideally 0. In reality, it's rare to have TPR = 1, FPR = 0, and FNR = 0, but these measures are useful to compare the performance of multiple models that are designed for solving the same problem. Note that in general, the model that is more preferable may depend on the business situation. During the discovery phase of the data analytics lifecycle, the team should have learned from the business what kind of errors can be tolerated. Some business situations are more tolerant of type I errors, whereas others may be more tolerant of type II errors. In some cases, a model with a TPR of 0.95 and an FPR of 0.3 is more acceptable than a model with a TPR of 0.9 and an FPR of 0.1 even if the second model is more accurate overall. Consider the case of e-mail spam filtering. Some people (such as busy executives) only want important e-mail in their inbox and are tolerant of having some less important e-mail end up in their spam folder as long as no spam is in their inbox. Other people may not want any important or less important e-mail to be specified as spam and are willing to have some spam in their inboxes as long as no important e-mail makes it into the spam folder.

Precision and recall are accuracy metrics used by the information retrieval community, but they can be used to characterize classifiers in general. ***Precision*** is the percentage of instances marked positive that really are positive, as shown in Equation 2.22.

$$Precision = \frac{TP}{TP + FP} \quad (2.22)$$

Recall is the percentage of positive instances that were correctly identified. Recall is equivalent to the TPR. Chapter 9, ―Advanced Analytical Theory and Methods: Text Analysis,‖ discusses how to use precision and recall for evaluation of classifiers in the context of text analysis.

Given the confusion matrix from Table 2.7, the metrics can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% = \frac{3 + 87}{3 + 87 + 2 + 8} \times 100\% = 90\%$$

$$TPR\ (or\ Recall) = \frac{TP}{TP + FN} = \frac{3}{3 + 8} \approx 0.273$$

$$FPR = \frac{FP}{FP + TN} = \frac{2}{2 + 87} \approx 0.022$$

$$FNR = \frac{FN}{TP + FN} = \frac{8}{3 + 8} \approx 0.727$$

$$Precision = \frac{TP}{TP + FP} = \frac{3}{3 + 2} = 0.6$$

These metrics show that for the bank marketing example, the naïve Bayes classifier performs well with accuracy and FPR measures and relatively well on precision. However, it performs poorly on TPR and FNR. To improve the performance, try to include more

attributes in the datasets to better distinguish the characteristics of the records. There are other ways to evaluate the performance of a classifier in general, such as *N*-fold cross validation (Chapter 6) or bootstrap [14].

Chapter 6 has introduced the ***ROC curve,*** which is a common tool to evaluate classifiers. The abbreviation stands for ***receiver operating characteristic***, a term used in signal detection to characterize the trade-off between hit rate and false-alarm rate over a noisy channel. A ROC curve evaluates the performance of a classifier based on the TP and FP, regardless of other factors such as class distribution and error costs. The vertical axis is the True Positive Rate (TPR), and the horizontal axis is the False Positive Rate (FPR).

As seen in Chapter 6, any classifier can achieve the bottom left of the graph where TPR = FPR = 0 by classifying everything as negative. Similarly, any classifier can achieve the top right of the graph where TPR = FPR = 1 by classifying everything as positive. If a classifier performs ―at chance‖ by random guessing the results, it can achieve any point on the diagonal line TPR=FPR by choosing an appropriate threshold of positive/negative. An ideal classifier should perfectly separate positives from negatives and thus achieve the top-left corner (TPR = 1, FPR = 0). The ROC curve of such classifiers goes straight up from TPR = FPR = 0 to the top-left corner and moves straight right to the top-right corner. In reality, it can be difficult to achieve the top-left corner. But a better classifier should be closer to the top left, separating it from other classifiers that are closer to the diagonal line.

Related to the ROC curve is the ***area under the curve*** (AUC). The AUC is calculated by measuring the area under the ROC curve. Higher AUC scores mean the classifier performs better. The score can range from 0.5 (for the diagonal line TPR=FPR) to 1.0 (with ROC passing through the top-left corner).

In the bank marketing example, the training set includes 2,000 instances. An additional 100 instances are included as the testing set. Figure 2.10 shows a ROC curve of the naïve Bayes classifier built on the training set of 2,000 instances and tested on the testing set of 100 instances. The figure is generated by the following R script. The ROCR package is required for plotting the ROC curve. The 2,000 instances are in a data frame called *banktrain*, and the additional 100 instances are in a data frame called *banktest*.

```
library(ROCR)
# training set
banktrain <- read.table(―bank-sample.csv‖,header=TRUE,sep=‖,‖)
# drop a few columns
drops <- c(―balance‖, ―day‖, ―campaign‖, ―pdays‖, ―previous‖, ―month‖)
banktrain <- banktrain [,!(names(banktrain) %in% drops)]
# testing set
banktest <- read.table(―bank-sample-test.csv‖,header=TRUE,sep=‖,‖)
banktest <- banktest [,!(names(banktest) %in% drops)]
# build the naïve Bayes classifier nb_model <- naiveBayes(subscribed~.,
      data=banktrain)
# perform on the testing set nb_prediction <- predict(nb_model,
       # remove column ―subscribed‖
       banktest[,-ncol(banktest)],
```

```
        type=_raw‘)
score <- nb_prediction[, c(–yes‖)] actual_class <- banktest$subscribed ==
_yes‘ pred <- prediction(score, actual_class)
perf <- performance(pred, –tpr‖, –fpr‖)
plot(perf, lwd=2, xlab=–False Positive Rate (FPR)‖,
  ylab=–True Positive Rate (TPR)‖)
abline(a=0, b=1, col=–gray50‖, lty=3)
```

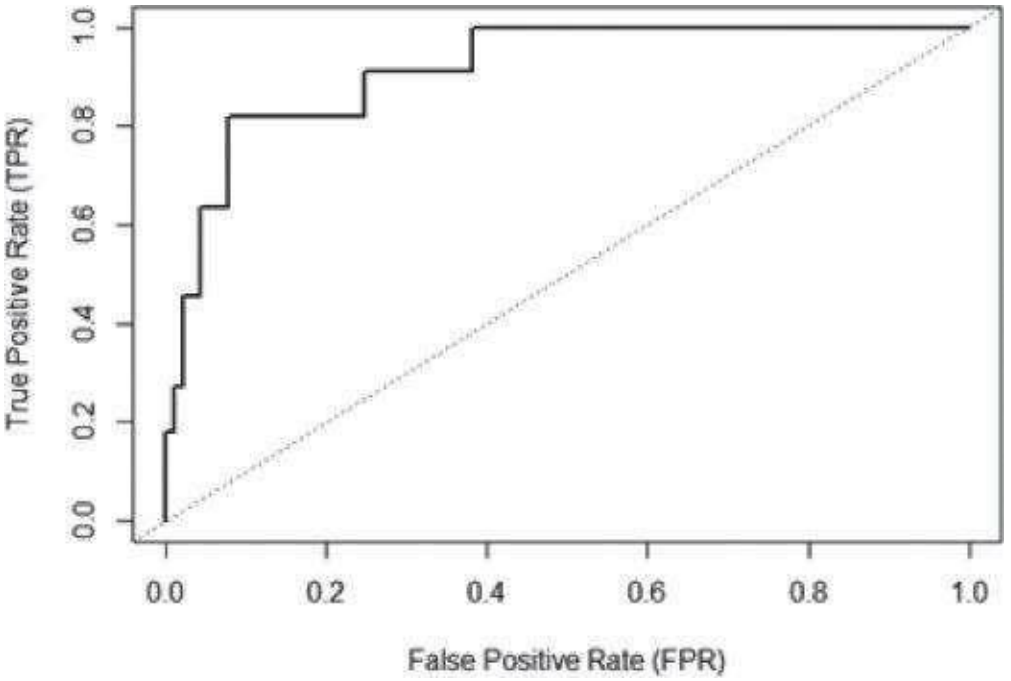


Figure 2.10 ROC curve of the naïve Bayes classifier on the bank marketing dataset

The following R code shows that the corresponding AUC score of the ROC curve is about 0.915.

```
auc <- performance(pred, –auc‖)
auc <- unlist(slot(auc, –y.values‖))
auc
[1] 0.9152196
```

Advanced Analytical Theory and Methods: Clustering

3.1 Overview of Clustering

In general, clustering is the use of ***unsupervised*** techniques for grouping similar objects. In machine learning, unsupervised refers to the problem of finding hidden structure within unlabeled data. Clustering techniques are unsupervised in the sense that the data scientist does not determine, in advance, the labels to apply to the clusters. The structure of the data describes the objects of interest and determines how best to group the objects. For example, based on customers‘ personal income, it is straightforward to divide the customers into three groups depending on arbitrarily selected values. The customers could be divided into three groups as follows:

- Earn less than $10,000
- Earn between $10,000 and $99,999
- Earn $100,000 or more

In this case, the income levels were chosen somewhat subjectively based on easy-to-communicate points of delineation. However, such groupings do not indicate a natural affinity of the customers within each group. In other words, there is no inherent reason to believe that the customer making $90,000 will behave any differently than the customer making $110,000. As additional dimensions are introduced by adding more variables about the customers, the task of finding meaningful groupings becomes more complex. For instance, suppose variables such as age, years of education, household size, and annual purchase expenditures were considered along with the personal income variable. What are the natural occurring groupings of customers? This is the type of question that clustering analysis can help answer.

Clustering is a method often used for exploratory analysis of the data. In clustering, there are no predictions made. Rather, clustering methods find the similarities between objects according to the object attributes and group the similar objects into clusters. Clustering techniques are utilized in marketing, economics, and various branches of science. A popular clustering method is k-means.

3.2 K-means

Given a collection of objects each with n measurable attributes, ***k-means*** [1] is an analytical technique that, for a chosen value of k, identifies k clusters of objects based on the objects‘ proximity to the center of the k groups. The center is determined as the arithmetic average (mean) of each cluster‘s n-dimensional vector of attributes. This section describes the algorithm to determine the k means as well as how best to apply this technique to several use cases. Figure 3.1 illustrates three clusters of objects with two attributes. Each object in the dataset is represented by a small dot color-coded to the closest large dot, the mean of the cluster.

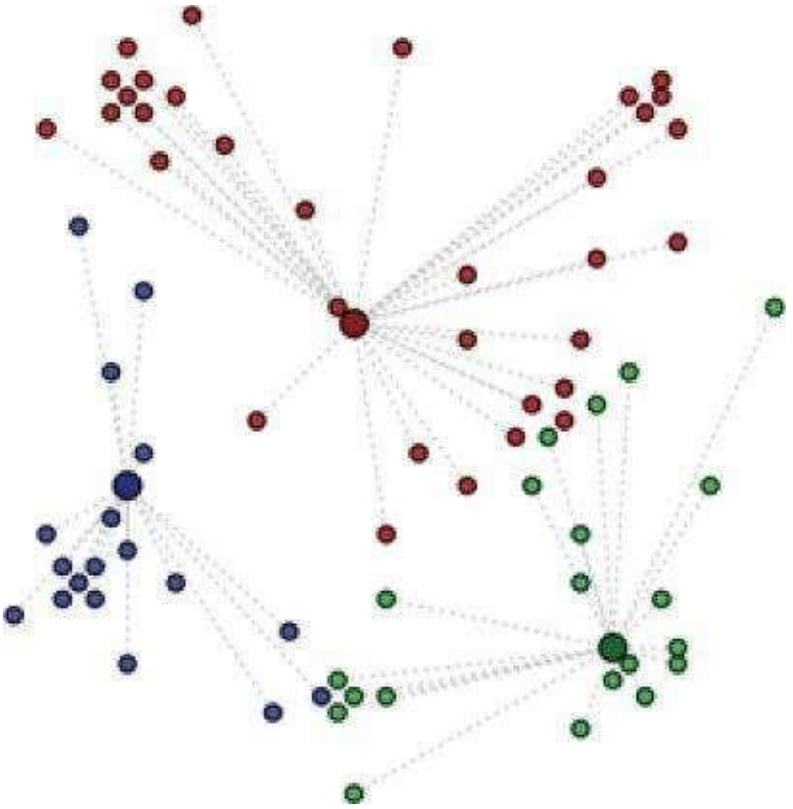


Figure 3.1 Possible k-means clusters for k=3

3.2.1 Use Cases

Clustering is often used as a lead-in to classification. Once the clusters are identified, labels can be applied to each cluster to classify each group based on its characteristics. Classification is covered in more detail in Chapter 7, ―Advanced Analytical Theory and Methods: Classification.‖ Clustering is primarily an exploratory technique to discover hidden structures of the data, possibly as a prelude to more focused analysis or decision processes. Some specific applications of k-means are image processing, medical, and customer segmentation.

Image Processing

Video is one example of the growing volumes of unstructured data being collected. Within each frame of a video, k-means analysis can be used to identify objects in the video. For each frame, the task is to determine which pixels are most similar to each other. The attributes of each pixel can include brightness, color, and location, the x and y coordinates in the frame. With security video images, for example, successive frames are examined to identify any changes to the clusters. These newly identified clusters may indicate unauthorized access to a facility.

Medical

Patient attributes such as age, height, weight, systolic and diastolic blood pressures, cholesterol level, and other attributes can identify naturally occurring clusters. These clusters could be used to target individuals for specific preventive measures or clinical

trial participation. Clustering, in general, is useful in biology for the classification of plants and animals as well as in the field of human genetics.

Customer Segmentation

Marketing and sales groups use k-means to better identify customers who have similar behaviors and spending patterns. For example, a wireless provider may look at the following customer attributes: monthly bill, number of text messages, data volume consumed, minutes used during various daily periods, and years as a customer. The wireless company could then look at the naturally occurring clusters and consider tactics to increase sales or reduce the customer ***churn rate***, the proportion of customers who end their relationship with a particular company.

3.2.2 Overview of the Method

To illustrate the method to find k clusters from a collection of M objects with n attributes, the two-dimensional case (n = 2) is examined. It is much easier to visualize the k-means method in two dimensions. Later in the chapter, the two-dimension scenario is generalized to handle any number of attributes.

Because each object in this example has two attributes, it is useful to consider each object corresponding to the point (x_i, y_i), where x and y denote the two attributes and i = 1, 2 … M. For a given cluster of m points (m ≤ M), the point that corresponds to the cluster's mean is called a ***centroid***. In mathematics, a centroid refers to a point that corresponds to the center of mass for an object.

The k-means algorithm to find k clusters can be described in the following four steps.

1. Choose the value of k and the k initial guesses for the centroids.

In this example, k = 3, and the initial centroids are indicated by the points shaded in red, green, and blue in Figure 3.2.

2. Compute the distance from each data point (x_i, y_i) to each centroid. Assign each point to the closest centroid. This association defines the first k clusters.

In two dimensions, the distance, *d*, between any two points, (x_1, y_1) and (x_2, y_2), in the Cartesian plane is typically expressed by using the Euclidean distance measure provided in Equation 3.1.

3.1 $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

In Figure 3.3, the points closest to a centroid are shaded the corresponding color.

3. Compute the centroid, the center of mass, of each newly defined cluster from Step 2.

In Figure 3.4, the computed centroids in Step 3 are the lightly shaded points of the corresponding color. In two dimensions, the centroid ( ) of the m points in a k-means cluster is calculated as follows in Equation 3.2.

$$(x_c, y_c) = \left(\frac{\sum_{i=1}^{m} x_i}{m}, \frac{\sum_{i=1}^{m} y_i}{m} \right) \quad \textbf{3.2}$$

Thus, (x_c, y_c) is the ordered pair of the arithmetic means of the coordinates of the m points in the cluster. In this step, a centroid is computed for each of the k clusters.

3. Repeat Steps 2 and 3 until the algorithm converges to an answer.

1. Assign each point to the closest centroid computed in Step 3.
2. Compute the centroid of newly defined clusters.
3. Repeat until the algorithm reaches the final answer.

Convergence is reached when the computed centroids do not change or the centroids and the assigned points oscillate back and forth from one iteration to the next. The latter case can occur when there are one or more points that are equal distances from the computed centroid.

To generalize the prior algorithm to n dimensions, suppose there are M objects, where each object is described by n attributes or property values $(p_1, p_2, \ldots p_n)$. Then object *i* is described by $(p_{i1}, p_{i2}, \ldots p_{in})$ for *i* = 1,2,…, M. In other words, there is a matrix with M rows corresponding to the M objects and n columns to store the attribute values. To expand the earlier process to find the k clusters from two dimensions to n dimensions, the following equations provide the formulas for calculating the distances and the locations of the centroids for n ≥ 1.

For a given point, $\boldsymbol{p_i}$, at and a centroid, $\boldsymbol{q}$, located at , the distance, $\boldsymbol{d}$, between $\boldsymbol{p_i}$ and $\boldsymbol{q}$, is expressed as shown in Equation 3.3.

$$d(p_i, q) = \sqrt{\sum_{j=1}^{n} (p_{ij} - q_j)^2} \quad \textbf{3.3}$$

The centroid, $\boldsymbol{q}$, of a cluster of m points, $(p_{i1}, p_{i2}, \ldots p_{in})$, is calculated as shown in Equation 3.3.

$$(q_1, q_2, \ldots, q_n) = \left(\frac{\sum_{i=1}^{m} p_{i1}}{m}, \frac{\sum_{i=1}^{m} p_{i2}}{m}, \ldots, \frac{\sum_{i=1}^{m} p_{in}}{m} \right) \quad \textbf{3.4}$$

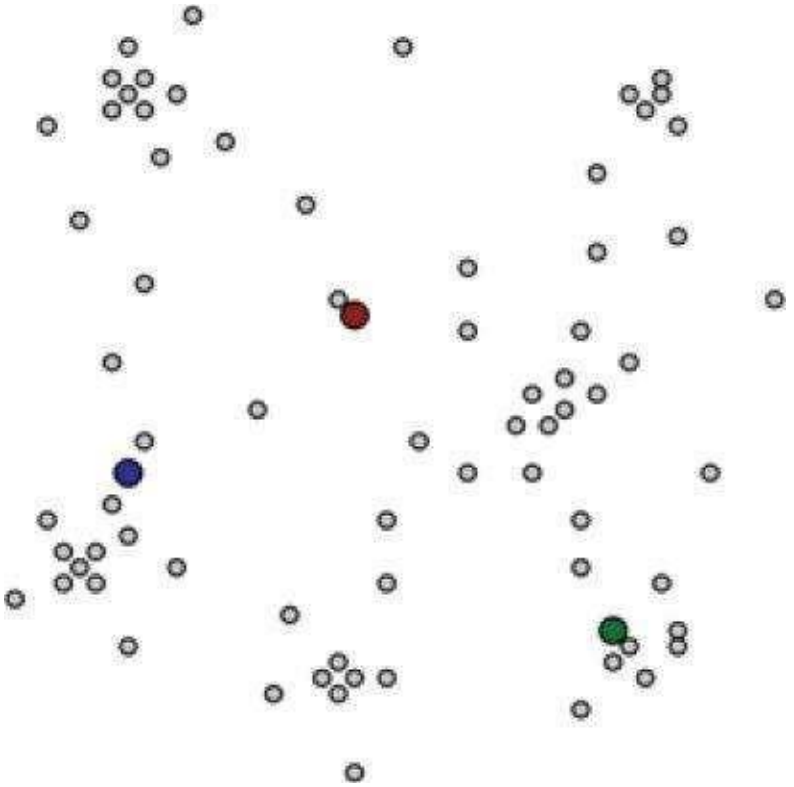


Figure 3.2 Initial starting points for the centroids

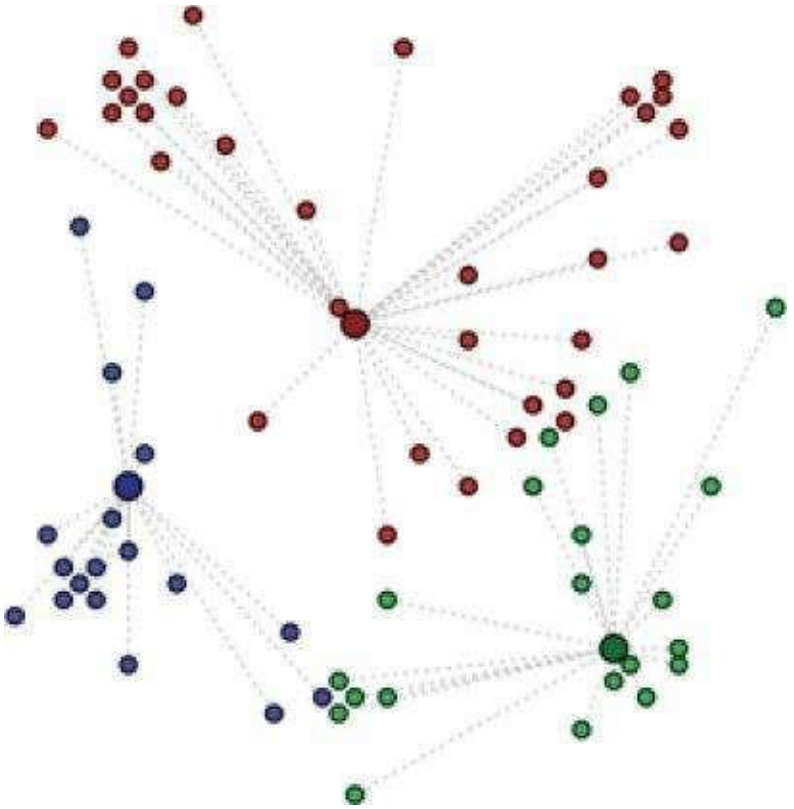


Figure 3.3 Points are assigned to the closest centroid

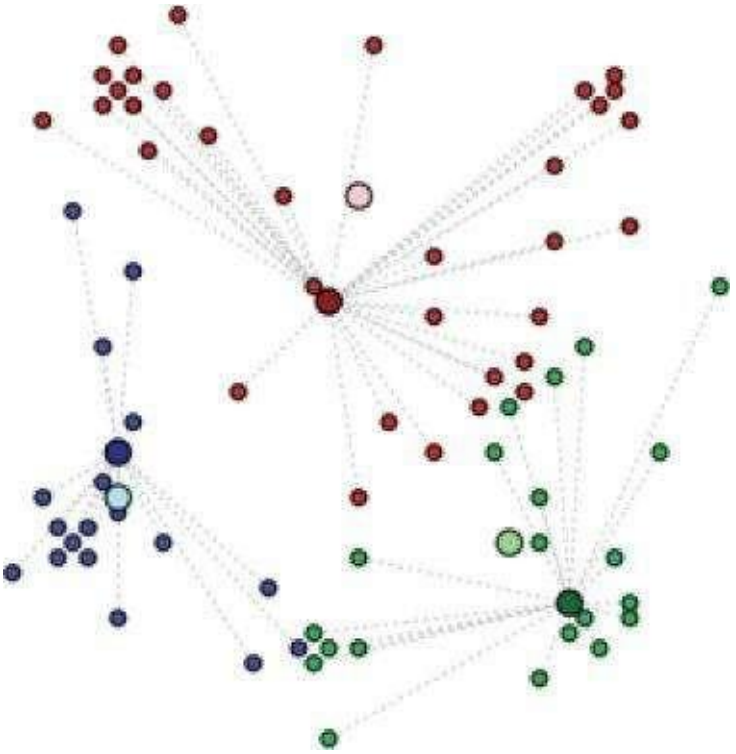


Figure 3.4 Compute the mean of each cluster

3.2.3 Determining the Number of Clusters

With the preceding algorithm, k clusters can be identified in a given dataset, but what value of k should be selected? The value of k can be chosen based on a reasonable guess or some predefined requirement. However, even then, it would be good to know how much better or worse having k clusters versus k – 1 or k + 1 clusters would be in explaining the structure of the data. Next, a heuristic using the Within Sum of Squares (WSS) metric is examined to determine a reasonably optimal value of k. Using the distance function given in Equation 3.3, WSS is defined as shown in Equation 3.5.

$$WSS = \sum_{i=1}^{M} d(p_i, q^{(i)})^2 = \sum_{i=1}^{M} \sum_{j=1}^{n} \left(p_{ij} - q_j^{(i)}\right)^2 \quad \textbf{3.5}$$

In other words, WSS is the sum of the squares of the distances between each data point and the closest centroid. The term $q^{(i)}$ indicates the closest centroid that is associated with the ith point. If the points are relatively close to their respective centroids, the WSS is relatively small. Thus, if k + 1 clusters do not greatly reduce the value of WSS from the case with only k clusters, there may be little benefit to adding another cluster.

Using R to Perform a K-means Analysis

To illustrate how to use the WSS to determine an appropriate number, k, of clusters, the following example uses R to perform a k-means analysis. The task is to group 620 high school seniors based on their grades in three subject areas: English, mathematics, and science. The grades are averaged over their high school career and assume values from 0 to 100. The following R code establishes the necessary R libraries and imports the CSV

file containing the grades.

```
library(plyr) library(ggplot2) library(cluster)
library(lattice) library(graphics) library(grid) library(gridExtra)
#import the student grades
grade_input = as.data.frame(read.csv("c:/data/grades_km_input.csv"))
```

The following R code formats the grades for processing. The data file contains four columns. The first column holds a student identification (ID) number, and the other three columns are for the grades in the three subject areas. Because the student ID is not used in the clustering analysis, it is excluded from the k-means input matrix, *kmdata*.

```
kmdata_orig = as.matrix(grade_input[,c("Student","English",
"Math","Science")])
kmdata <- kmdata_orig[,2:4]
kmdata[1:10,]
```

```
  English Math Science
[1,]  99  96   97 [2,]  99  96   97 [3,]  98  97   97 [4,]  95  100   95 [5,]  95  96
96 [6,]  96  97   96 [7,]  100  96   97 [8,]  95  98   98 [9,]  98  96   96 [10,]  99
99   95
```

To determine an appropriate value for k, the k-means algorithm is used to identify clusters for k = 1, 2, …, 15. For each value of k, the WSS is calculated. If an additional cluster provides a better partitioning of the data points, the WSS should be markedly smaller than without the additional cluster.

The following R code loops through several k-means analyses for the number of centroids, *k*, varying from 1 to 15. For each k, the option *nstart*=25 specifies that the k-means algorithm will be repeated 25 times, each starting with *k* random initial centroids. The corresponding value of WSS for each k-mean analysis is stored in the *wss* vector.

```
wss <- numeric(15)
for (k in 1:15) wss[k] <- sum(kmeans(kmdata, centers=k, nstart=25)$withinss)
```

Using the basic R plot function, each WSS is plotted against the respective number of centroids, 1 through 15. This plot is provided in Figure 3.5.

```
plot(1:15, wss, type="b", xlab="Number of Clusters", ylab="Within Sum of
Squares")
```

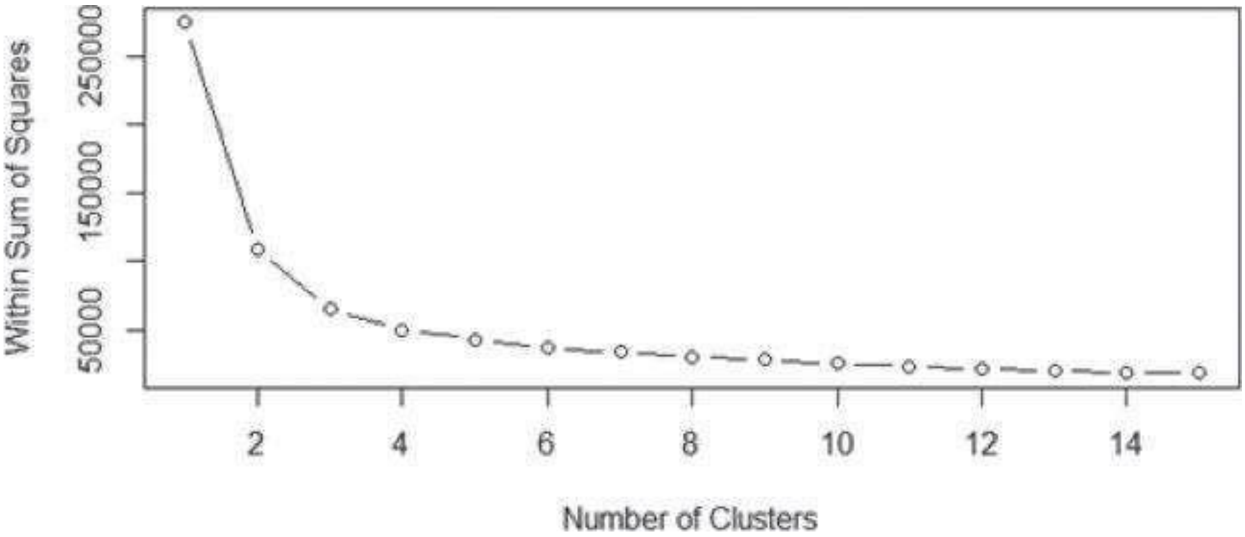


Figure 3.5 WSS of the student grade data

As can be seen, the WSS is greatly reduced when k increases from one to two. Another substantial reduction in WSS occurs at k = 3. However, the improvement in WSS is fairly linear for k > 3. Therefore, the k-means analysis will be conducted for k = 3. The process of identifying the appropriate value of k is referred to as finding the ―elbow‖ of the WSS curve.

```
km = kmeans(kmdata,3, nstart=25)
km

K-means clustering with 3 clusters of sizes 158, 218, 244
Cluster means:
English  Math Science
1 92.21519 93.37342 93.86076
2 73.22018 63.62844 65.84862
3 85.84426 79.68033 81.50820
Clustering vector:
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1  1 1 1 1 1 1 1 1
[41] 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1  1 1 1 1 1 1 1 1
[81] 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1  1 1 1 1 1 1 1 1
[121] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   3 3 3 3 3 3 3 3 3 3
[161] 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 1 1 3 3 1 3 3 3 1 3 3 3 3
   3 3 1 3 3 3 3 3 3 3
[201] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 3 3 3 3 3 3
[241] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 3 3 3 3 3 3
[281] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 3 3 3 3 3 3
[321] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
   3 3 3 3 3 3 3 3 3 3
[361] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 3 2 3 2 3 3 3
```

```
  2 2 2 2 3 3 2 2 2 2
[401] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2
[441] 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 3
  2 2 2 2 2 2 2 2 3 2
[481] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2
[521] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2
[561] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  2 2 2 2 2 2 2 2 2 2
[601] 3 3 2 2 3 3 3 3 1 1 3 3 3 2 2 3 2 3 3 3
Within cluster sum of squares by cluster: [1] 6692.589 34806.339 22983.131
(between_SS / total_SS = 76.5 %)
Available components:
[1] "cluster" "centers" "totss" "withinss" "tot.withinss" [6] "betweenss"
"size"  "iter"  "ifault"
```

The displayed contents of the variable *km* include the following:

- The location of the cluster means
- A clustering vector that defines the membership of each student to a corresponding cluster 1, 2, or 3
- The WSS of each cluster
- A list of all the available k-means components

The reader can find details on these components and using k-means in R by employing the help facility.

The reader may have wondered whether the k-means results stored in *km* are equivalent to the WSS results obtained earlier in generating the plot in Figure 3.5. The following check verifies that the results are indeed equivalent.

```
c( wss[3] , sum(km$withinss) ) [1] 64483.06 64483.06
```

In determining the value of k, the data scientist should visualize the data and assigned clusters. In the following code, the ggplot2 package is used to visualize the identified student clusters and centroids.

```
#prepare the student data and clustering results for plotting df =
as.data.frame(kmdata_orig[,2:4])
df$cluster = factor(km$cluster)
centers=as.data.frame(km$centers)
g1= ggplot(data=df, aes(x=English, y=Math, color=cluster )) + geom_point() +
theme(legend.position="right") + geom_point(data=centers,
    aes(x=English,y=Math, color=as.factor(c(1,2,3))),
    size=10, alpha=.3, show_guide=FALSE)
g2 =ggplot(data=df, aes(x=English, y=Science, color=cluster )) +
geom_point() +
```

```
geom_point(data=centers,
    aes(x=English,y=Science, color=as.factor(c(1,2,3))),
    size=10, alpha=.3, show_guide=FALSE)
g3 = ggplot(data=df, aes(x=Math, y=Science, color=cluster )) +
geom_point() +
geom_point(data=centers,
    aes(x=Math,y=Science, color=as.factor(c(1,2,3))),
    size=10, alpha=.3, show_guide=FALSE) tmp =
ggplot_gtable(ggplot_build(g1)) grid.arrange(arrangeGrob(g1 +
theme(legend.position="none"),
        g2 + theme(legend.position="none"),
        g3 + theme(legend.position="none"),
        main ="High School Student Cluster Analysis",
        ncol=1))
```

The resulting plots are provided in Figure 3.6. The large circles represent the location of the cluster means provided earlier in the display of the *km* contents. The small dots represent the students corresponding to the appropriate cluster by assigned color: red, blue, or green. In general, the plots indicate the three clusters of students: the top academic students (red), the academically challenged students (green), and the other students (blue) who fall somewhere between those two groups. The plots also highlight which students may excel in one or two subject areas but struggle in other areas.

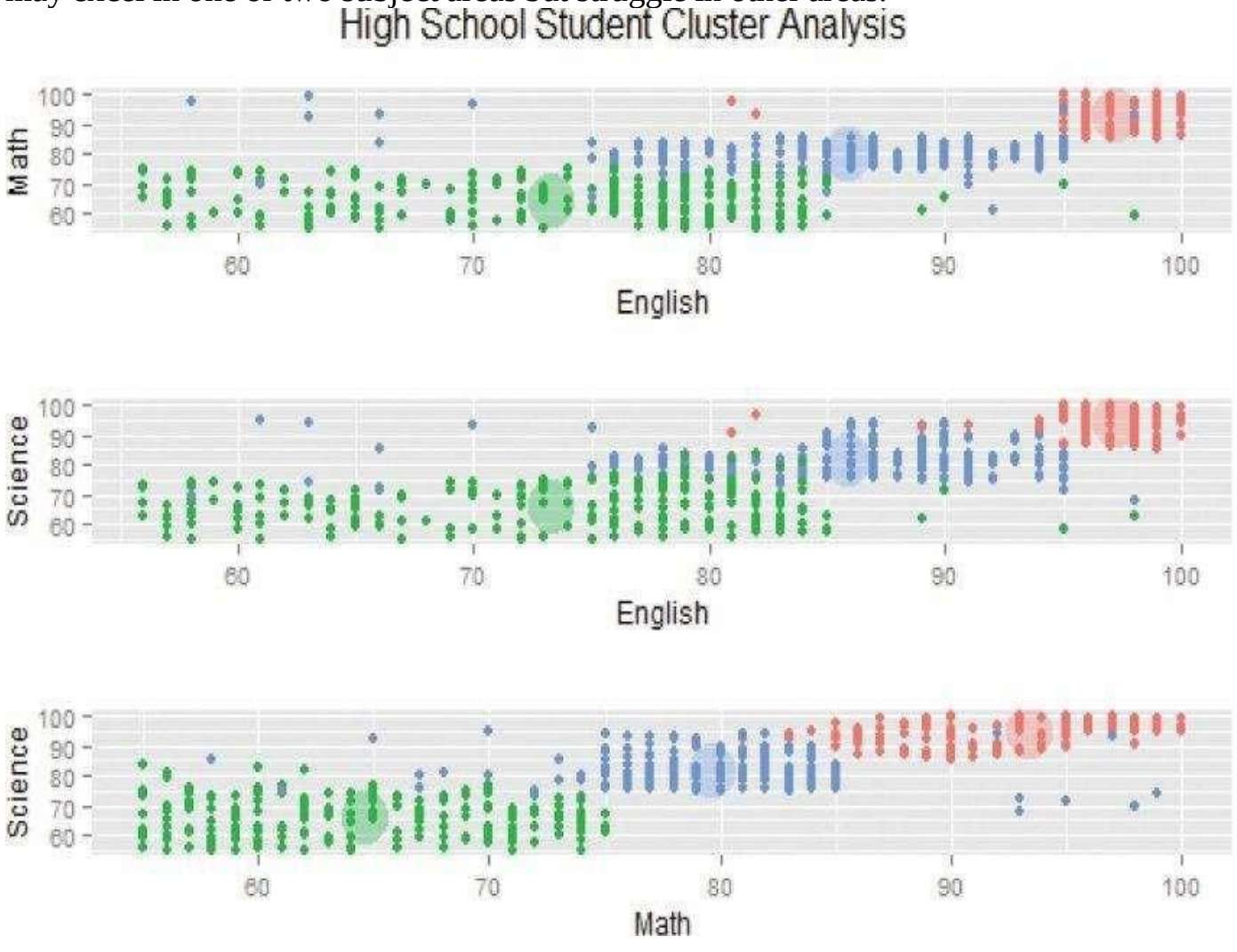


Figure 3.6 Plots of the identified student clusters

Assigning labels to the identified clusters is useful to communicate the results of an analysis. In a marketing context, it is common to label a group of customers as frequent shoppers or big spenders. Such designations are especially useful when communicating the clustering results to business users or executives. It is better to describe the marketing plan for big spenders rather than Cluster #1.

3.2.4 Diagnostics

The heuristic using WSS can provide at least several possible k values to consider. When the number of attributes is relatively small, a common approach to further refine the choice of k is to plot the data to determine how distinct the identified clusters are from each other. In general, the following questions should be considered.

- Are the clusters well separated from each other? Do any of the clusters have only a few points?
- Do any of the centroids appear to be too close to each other?

In the first case, ideally the plot would look like the one shown in Figure 3.7, when n = 2. The clusters are well defined, with considerable space between the four identified clusters. However, in other cases, such as Figure 3.8, the clusters may be close to each other, and the distinction may not be so obvious.

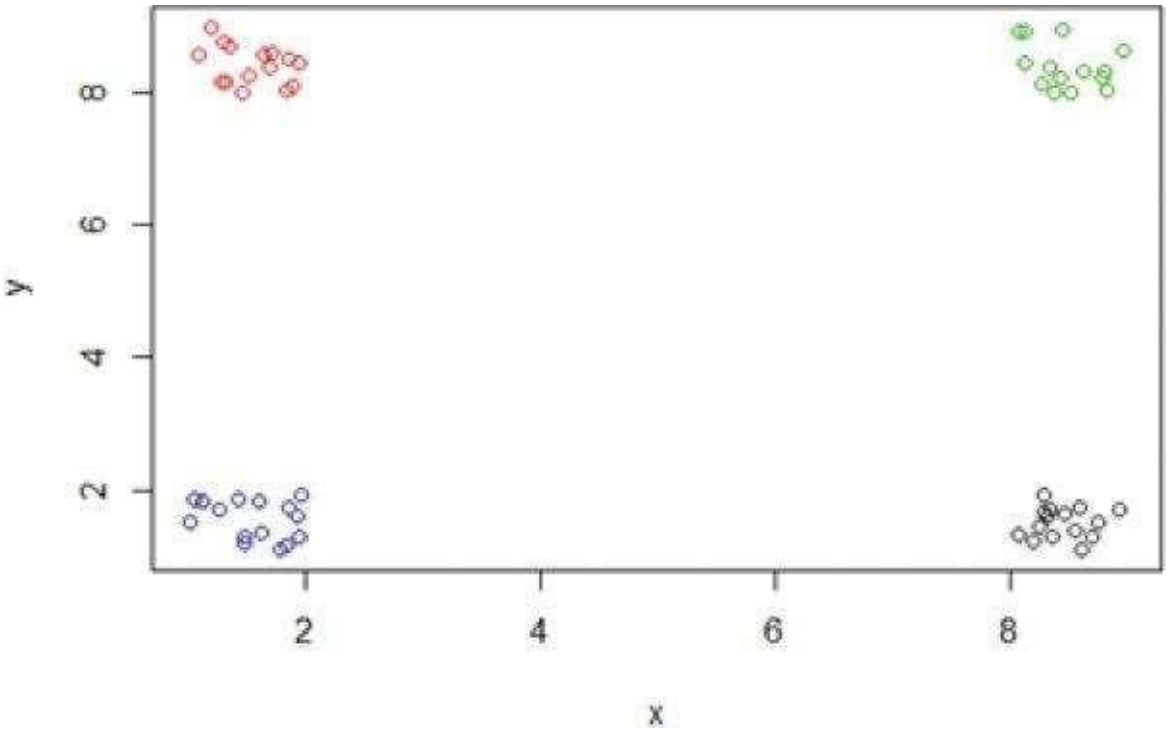


Figure 3.7 Example of distinct clusters

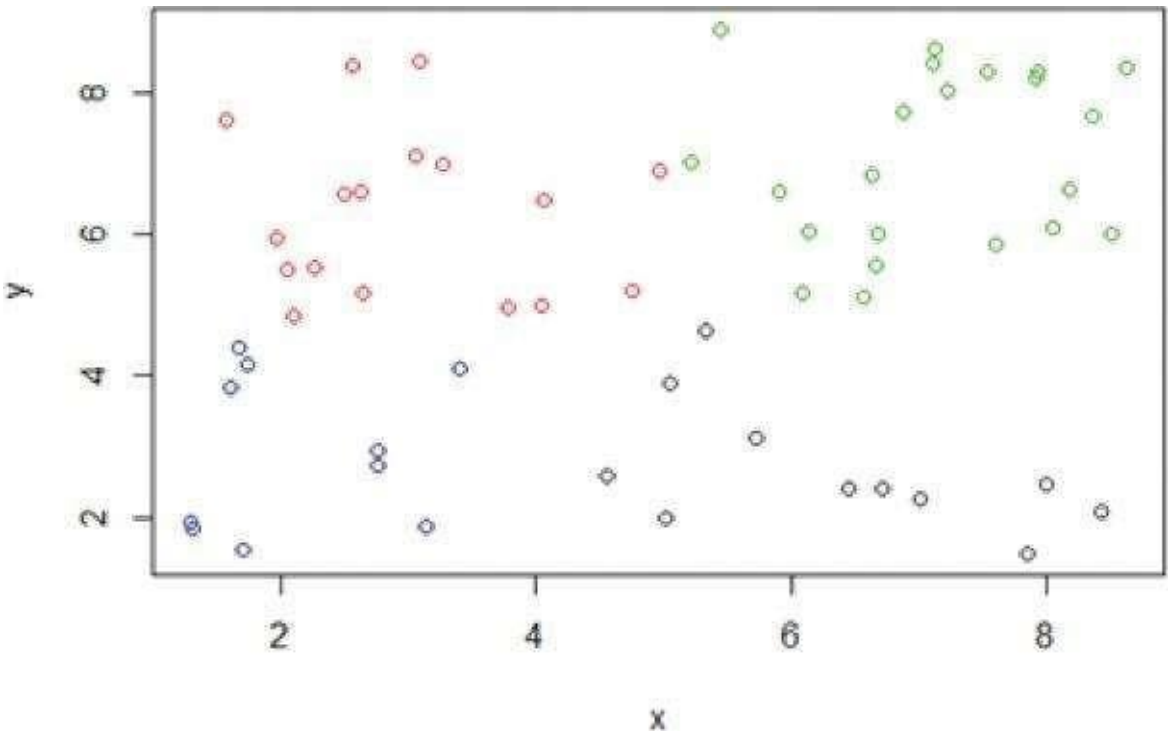


Figure 3.8 Example of less obvious clusters

In such cases, it is important to apply some judgment on whether anything different will result by using more clusters. For example, Figure 3.9 uses six clusters to describe the same dataset as used in Figure 3.8. If using more clusters does not better distinguish the groups, it is almost certainly better to go with fewer clusters.

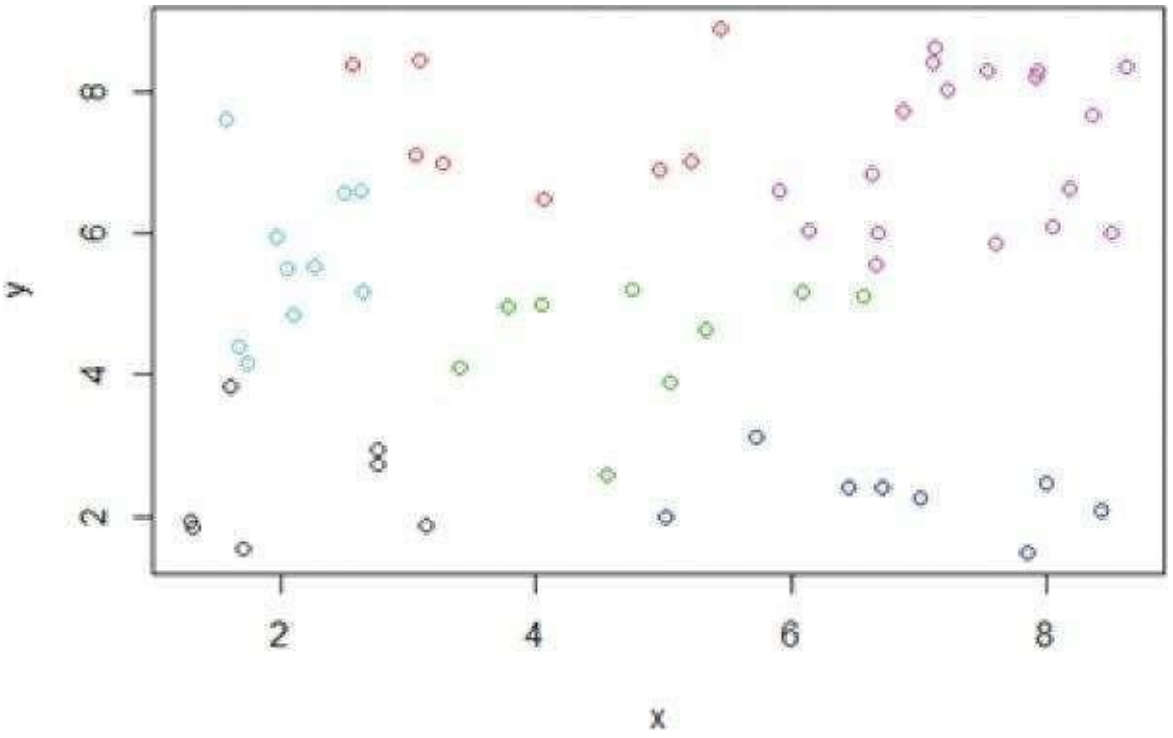


Figure 3.9 Six clusters applied to the points from Figure 3.8

3.2.5 Reasons to Choose and Cautions

K-means is a simple and straightforward method for defining clusters. Once clusters and their associated centroids are identified, it is easy to assign new objects (for example, new customers) to a cluster based on the object‘s distance from the closest centroid. Because the method is unsupervised, using k-means helps to eliminate subjectivity from the analysis.

Although k-means is considered an unsupervised method, there are still several decisions that the practitioner must make:

- What object attributes should be included in the analysis?
- What unit of measure (for example, miles or kilometers) should be used for each attribute?
- Do the attributes need to be rescaled so that one attribute does not have a disproportionate effect on the results?
- What other considerations might apply?

Object Attributes

Regarding which object attributes (for example, age and income) to use in the analysis, it is important to understand what attributes will be known at the time a new object will be assigned to a cluster. For example, information on existing customers‘ satisfaction or purchase frequency may be available, but such information may not be available for potential customers.

The Data Scientist may have a choice of a dozen or more attributes to use in the clustering analysis. Whenever possible and based on the data, it is best to reduce the number of attributes to the extent possible. Too many attributes can minimize the impact of the most important variables. Also, the use of several similar attributes can place too much importance on one type of attribute. For example, if five attributes related to personal wealth are included in a clustering analysis, the wealth attributes dominate the analysis and possibly mask the importance of other attributes, such as age.

When dealing with the problem of too many attributes, one useful approach is to identify any highly correlated attributes and use only one or two of the correlated attributes in the clustering analysis. As illustrated in Figure 3.10, a scatterplot matrix, as introduced in Chapter 3, is a useful tool to visualize the pair-wise relationships between the attributes.

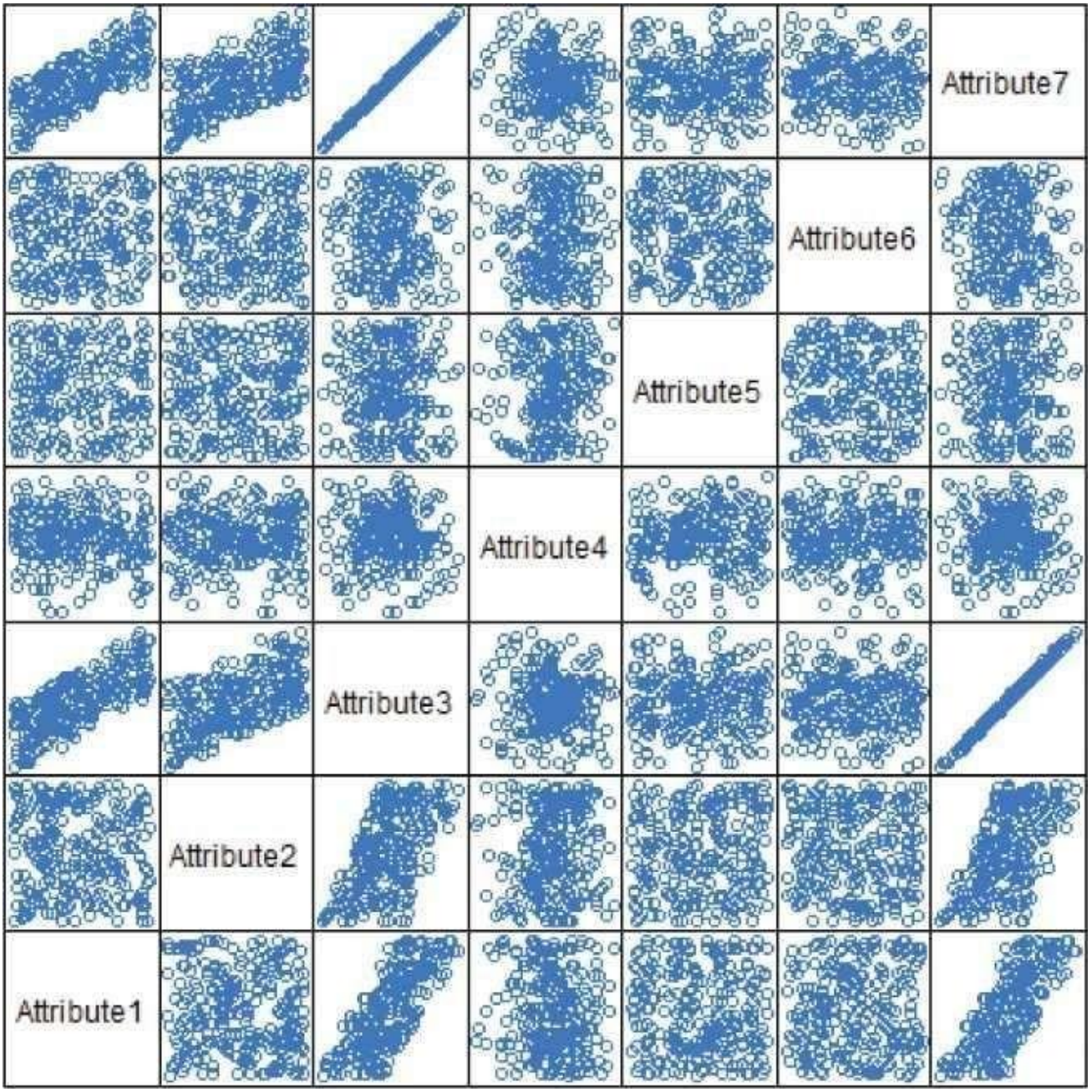


Figure 3.10 Scatterplot matrix for seven attributes

The strongest relationship is observed to be between *Attribute3* and *Attribute2*. If the value of one of these two attributes is known, it appears that the value of the other attribute is known with near certainty. Other linear relationships are also identified in the plot. For example, consider the plot of *Attribute2* against *Attribute3*. If the value of *Attribute2* is known, there is still a wide range of possible values for *Attribute3*. Thus, greater consideration must be given prior to dropping one of these attributes from the clustering analysis.

Another option to reduce the number of attributes is to combine several attributes into one measure. For example, instead of using two attribute variables, one for Debt and one for Assets, a Debt to Asset ratio could be used. This option also addresses the problem when the magnitude of an attribute is not of real interest, but the relative magnitude is a more

important measure.

Units of Measure

From a computational perspective, the k-means algorithm is somewhat indifferent to the units of measure for a given attribute (for example, meters or centimeters for a patient's height). However, the algorithm will identify different clusters depending on the choice of the units of measure. For example, suppose that k-means is used to cluster patients based on age in years and height in centimeters. For k=2, Figure 3.11 illustrates the two clusters that would be determined for a given dataset.

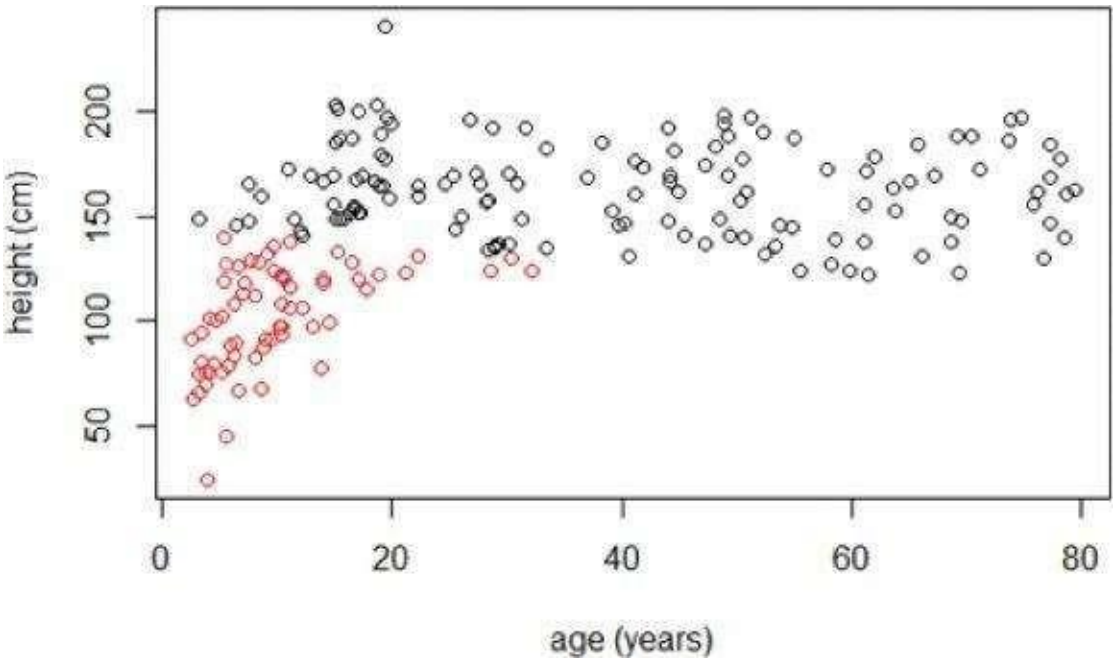


Figure 3.11 Clusters with height expressed in centimeters

But if the height was rescaled from centimeters to meters by dividing by 100, the resulting clusters would be slightly different, as illustrated in Figure 3.12.

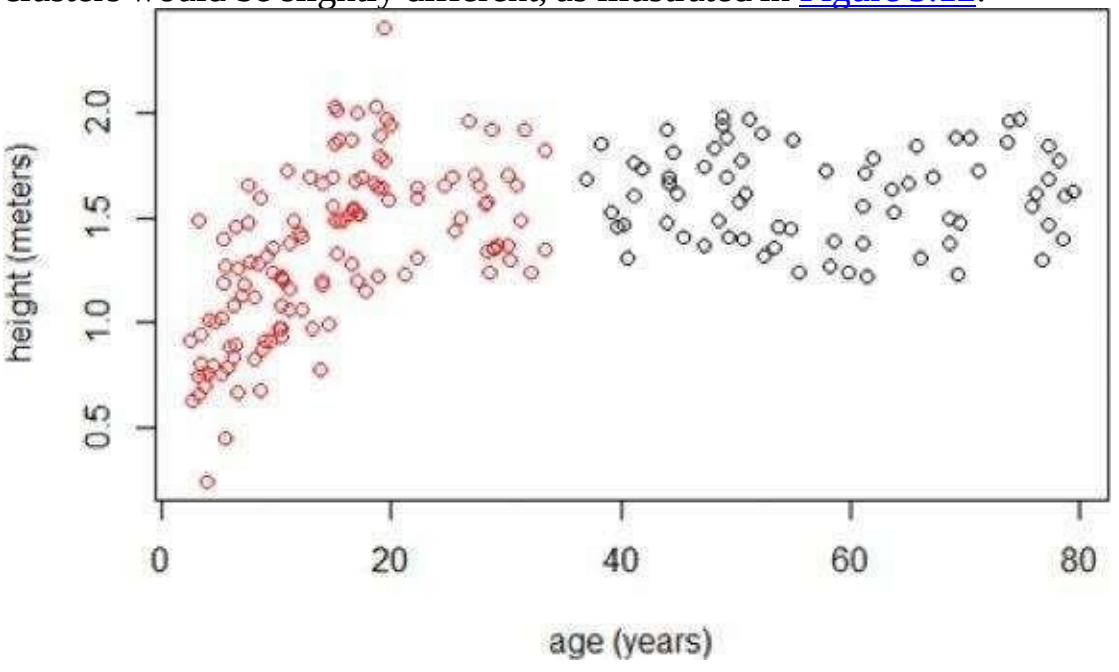


Figure 3.12 Clusters with height expressed in meters

When the height is expressed in meters, the magnitude of the ages dominates the distance calculation between two points. The height attribute provides only as much as the square between the difference of the maximum height and the minimum height or $(2.0-0)^2=4$ to the radicand, the number under the square root symbol in the distance formula given in Equation 3.3. Age can contribute as much as $(80-0)^2=6,400$ to the radicand when measuring the distance.

Rescaling

Attributes that are expressed in dollars are common in clustering analyses and can differ in magnitude from the other attributes. For example, if personal income is expressed in dollars and age is expressed in years, the income attribute, often exceeding $10,000, can easily dominate the distance calculation with ages typically less than 100 years.

Although some adjustments could be made by expressing the income in thousands of dollars (for example, 10 for $10,000), a more straightforward method is to divide each attribute by the attribute‘s standard deviation. The resulting attributes will each have a standard deviation equal to 1 and will be without units. Returning to the age and height example, the standard deviations are 23.1 years and 36.4 cm, respectively. Dividing each attribute value by the appropriate standard deviation and performing the k-means analysis yields the result shown in Figure 3.13.

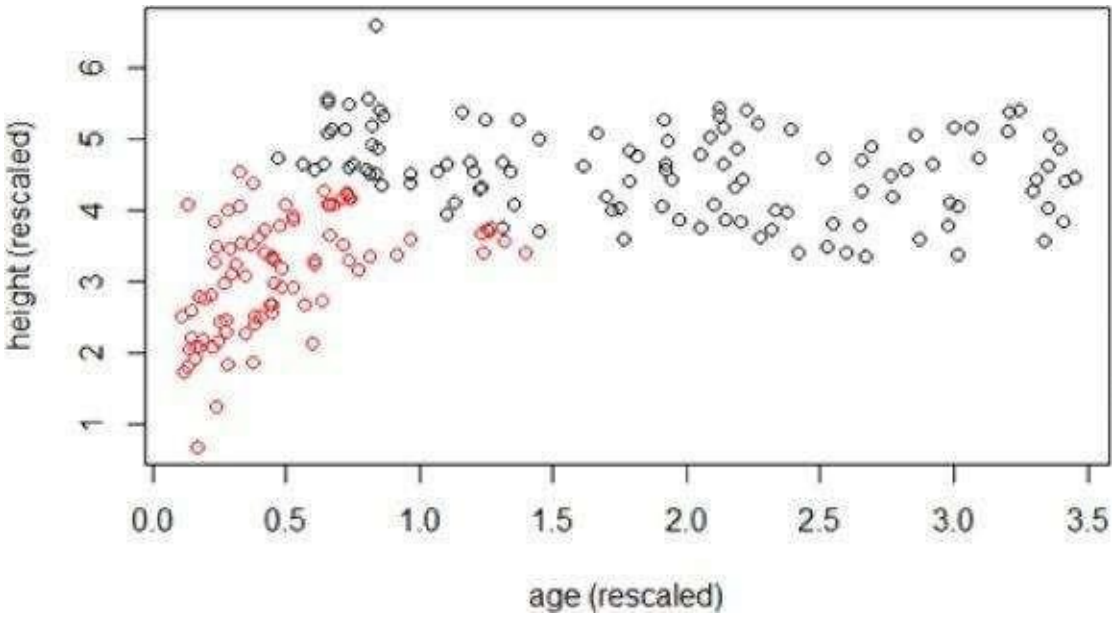


Figure 3.13 Clusters with rescaled attributes

With the rescaled attributes for age and height, the borders of the resulting clusters now fall somewhere between the two earlier clustering analyses. Such an occurrence is not surprising based on the magnitudes of the attributes of the previous clustering attempts. Some practitioners also subtract the means of the attributes to center the attributes around zero. However, this step is unnecessary because the distance formula is only sensitive to

the scale of the attribute, not its location.

In many statistical analyses, it is common to transform typically skewed data, such as income, with long tails by taking the logarithm of the data. Such transformation can also be appied in k-means, but the Data Scientist needs to be aware of what effect this transformation will have. For example, if log_{10} of income expressed in dollars is used, the practitioner is essentially stating that, from a clustering perspective, $1,000 is as close to $10,000 as $10,000 is to $100,000 $(because\ log_{10}1{,}000 = 3, log_{10}10{,}000 = 4, and\ log_{10}100{,}000 = 5)$. In many cases, the skewness of the data may be the reason to perform the clustering analysis in the first place.

Additional Considerations

The k-means algorithm is sensitive to the starting positions of the initial centroid. Thus, it is important to rerun the k-means analysis several times for a particular value of k to ensure the cluster results provide the overall minimum WSS. As seen earlier, this task is accomplished in R by using the `nstart` option in the `kmeans()` function call.

This chapter presented the use of the Euclidean distance function to assign the points to the closest centroids. Other possible function choices include the cosine similarity and the Manhattan distance functions. The cosine similarity function is often chosen to compare two documents based on the frequency of each word that appears in each of the documents [2]. For two points, *p* and *q*, at $(p_1, p_2, \ldots p_n)$ and $(q_1, q_2, \ldots q_n)$, respectively, the Manhattan distance, d_1, between *p* and *q* is expressed as shown in Equation 3.6.

$$d_1(p,q) = \sum_{j=1}^{n} |p_j - q_j| \qquad 3.6$$

The Manhattan distance function is analogous to the distance traveled by a car in a city, where the streets are laid out in a rectangular grid (such as city blocks). In Euclidean distance, the measurement is made in a straight line. Using Equation 3.6, the distance from (1, 1) to (4, 5) would be |1 – 4| + |1 – 5| = 2. From an optimization perspective, if there is a need to use the Manhattan distance for a clustering analysis, the median is a better choice for the centroid than use of the mean [2].

K-means clustering is applicable to objects that can be described by attributes that are numerical with a meaningful distance measure. From Chapter 3, interval and ratio attribute types can certainly be used. However, k-means does not handle categorical variables well. For example, suppose a clustering analysis is to be conducted on new car sales. Among other attributes, such as the sale price, the color of the car is considered important. Although one could assign numerical values to the color, such as red = 1, yellow = 2, and green = 3, it is not useful to consider that yellow is as close to red as yellow is to green from a clustering perspective. In such cases, it may be necessary to use an alternative clustering methodology. Such methods are described in the next section.

3.3 Additional Algorithms

The k-means clustering method is easily applied to numeric data where the concept of distance can naturally be applied. However, it may be necessary or desirable to use an alternative clustering algorithm. As discussed at the end of the previous section, k-means does not handle categorical data. In such cases, k-modes [3] is a commonly used method for clustering categorical data based on the number of differences in the respective components of the attributes. For example, if each object has four attributes, the distance from (a, b, e, d) to (d, d, d, d) is 3. In R, the function `kmode()` is implemented in the `klaR` package.

Because k-means and k-modes divide the entire dataset into distinct groups, both approaches are considered partitioning methods. A third partitioning method is known as Partitioning around Medoids (PAM). In general, a medoid is a representative object in a set of objects. In clustering, the ***medoids*** are the objects in each cluster that minimize the sum of the distances from the medoid to the other objects in the cluster. The advantage of using PAM is that the ―center‖ of each cluster is an actual object in the dataset. PAM is implemented in R by the `pam()` function included in the `cluster` R package. The `fpc` R package includes a function `pamk()`, which uses the `pam()` function to find the optimal value for k.

Other clustering methods include hierarchical agglomerative clustering and density clustering methods. In hierarchical agglomerative clustering, each object is initially placed in its own cluster. The clusters are then combined with the most similar cluster. This process is repeated until one cluster, which includes all the objects, exists. The R `stats` package includes the `hclust()` function for performing hierarchical agglomerative clustering. In density-based clustering methods, the clusters are identified by the concentration of points. The `fpc` R package includes a function, `dbscan()`, to perform density-based clustering analysis. Density-based clustering can be useful to identify irregularly shaped clusters.