Chapter 7

Supervised Learning: Classification

7.1 INTRODUCTION

OBJECTIVE OF THE CHAPTER :

In the last chapter on Bayesian Concept Learning, you were introduced to an important supervised learning algorithm – the Naïve Bayes algorithm. As we have seen, it is a very simple but powerful classifier based on Bayes' theorem of conditional probability. However, other than the Naïve Bayes classifier, there are more algorithms for classification. This chapter will focus on other classification algorithms.

The first algorithm we will delve into in this chapter is *k*-Nearest Neighbour (*k*NN), which tries to classify unlabelled data instances based on the similarity with the labelled instances in the training data.

Then, another critical classifier, named as decision tree, will be explained in detail. Decision tree, as the name suggests, is a classifier based on a series of logical decisions, which resembles a tree with branches.

Next, we will explore the random forest classifier, which, in very simplistic terms, can be thought as a collection of many decision trees.

Finally, a very powerful and popular classifier named Support Vector Machine (SVM) will be explored.

So, by the end of this chapter, you will gain enough knowledge to start solving a classification problem by using some standard classifiers.

7.2 EXAMPLE OF SUPERVISED LEARNING

In supervised learning, the labelled training data provide the basis for learning. According to the definition of machine learning, this labelled training data is the experience or prior knowledge or belief. It is called supervised learning because the process of learning from the training data by a machine can be related to a teacher supervising the learning process of a student who is new to the subject. Here, the teacher is the training data.

Training data is the past information with known value of class field or '**label**'. Hence, we say that the '**training data is labelled**' in the case of supervised learning (refer Fig. 7.1). Contrary to this, there is no labelled training data for unsupervised learning. Semi-supervised learning, as depicted in Figure 7.1, uses a small amount of labelled data along with unlabelled data for training.

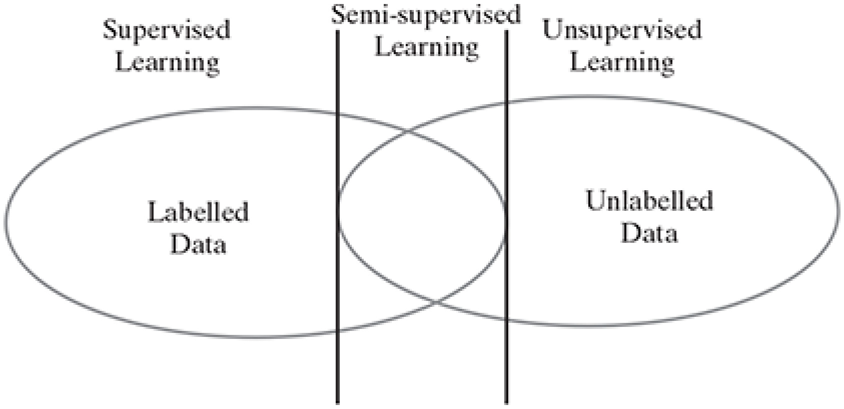


FIG. 7.1 Supervised learning vs. unsupervised learning

In a hospital, many patients are treated in the general wards. In comparison, the number of beds in the Intensive Care Unit (ICU) is much less. So, it is always a cause of worry for the hospital management that if the health condition of a number of patients in the general ward suddenly aggravates and they would have to be moved to ICU. Without previous planning and preparations, such a spike in demand becomes difficult for the hospital to manage. This problem can be addressed in a much better way if it is possible to predict which of the patients in the normal wards have a possibility of their health condition deteriorating and thus need to be moved to ICU.

This kind of prediction problem comes under the purview of supervised learning or, more specifically, under classification. The hospital already has all past patient records. The records of the patients whose health condition aggravated in the past and had to be moved to ICU can form the training data for this prediction problem. Test results of newly admitted patients are used to classify them as high-risk or low-risk patients.

Some more examples of supervised learning are as follows:

- Prediction of results of a game based on the past analysis of results
- Predicting whether a tumour is malignant or benign on the basis of the analysis of data
- Price prediction in domains such as real estate, stocks, etc.

Did you know?

'IBM Watson Health' developed by IBM software provides evidence-backed cancer care to each patient by understanding millions of data points. 'Watson for Oncology' helps physicians quickly identify vital information in a patient's medical record, surface relevant evidence, and explore various treatment options for patients (Source: https://www.ibm.com/watson/health/oncology-and-genomics/oncology).

7.3 CLASSIFICATION MODEL

Let us consider two examples, say 'predicting whether a tumour is malignant or benign' and 'price prediction in the domain of real estate'. Are these two problems same in nature?

The answer is 'no'. It is true that both of them are problems related to prediction. However, for tumour prediction, we are trying to predict which category or class, i.e. ‘malignant’ or ‘benign’, an unknown input data related to tumour belongs to. In the other case, that is, for price prediction, we are trying to predict an absolute value and not a class.

When we are trying to predict a categorical or nominal variable, the problem is known as a classification problem. A classification problem is one where the output variable is a category such as ‘red’ or ‘blue’ or ‘malignant tumour’ or ‘benign tumour’, etc.

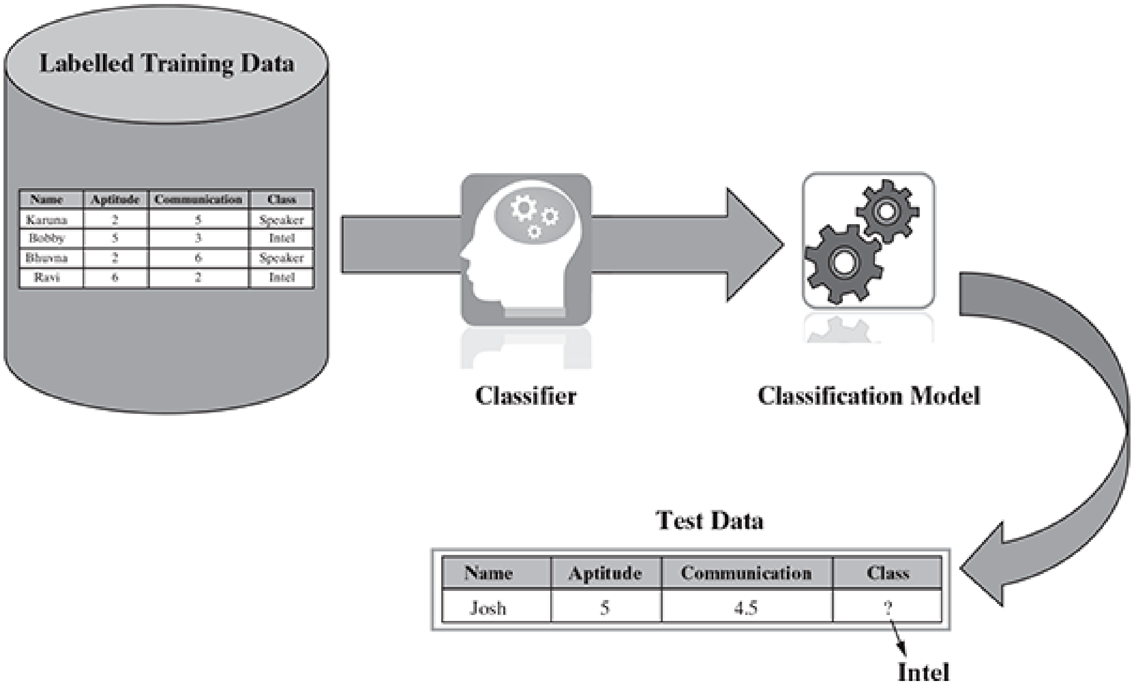
Whereas when we are trying to predict a numerical variable such as ‘price’, ‘weight’, etc. the problem falls under the category of regression.

Note that:

Supervised machine learning is as good as the data used to train it. If the training data is poor in quality, the prediction will also be far from being precise.

We can observe that in classification, the whole problem centres around assigning a label or category or class to a test data on the basis of the label or category or class information that is imparted by the training data. Because the target objective is to assign a class label, we call this type of problem as a classification problem. Figure 7.2 depicts the typical process of classification where a classification model is obtained from the labelled training data by a classifier algorithm. On the basis of the model, a class label (e.g. ‘Intel’ as in the case of the test data referred in Fig. 7.2) is assigned to the test data.

A critical classification problem in the context of the banking domain is identifying potentially fraudulent transactions. Because there are millions of transactions which have to be scrutinized to identify whether a particular transaction might be a fraud transaction, it is not possible for any human being to carry out this task. Machine learning is leveraged efficiently to do this task, and this is a classic case of classification. On the basis of the past transaction data, especially the ones labelled as fraudulent, all new incoming transactions are marked or labelled as usual or suspicious. The suspicious transactions are subsequently segregated for a closer review.



Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bobby	5	3	Intel
Bhuvna	2	6	Speaker
Ravi	6	2	Intel

Name	Aptitude	Communication	Class
Josh	5	4.5	?

FIG. 7.2 Classification model

In summary, classification is a type of supervised learning where a target feature, which is of categorical type, is predicted for test data on the basis of the information imparted by the training data. The target categorical feature is known as **class** .

Some typical classification problems include the following:

- Image classification
- Disease prediction
- Win–loss prediction of games
- Prediction of natural calamity such as earthquake, flood, etc.
- Handwriting recognition **:**

Did you know?

- Machine learning saves lives – it can spot 52% of breast cancer cells at least a year before patients are diagnosed
- US Postal Service uses machine learning for handwriting recognition
- Facebook’s news feed uses machine learning to personalize each member’s feed

7.4 CLASSIFICATION LEARNING STEPS

First, there is a problem which is to be solved, and then, the required data (related to the problem, which is already stored in the system) is evaluated and pre-processed based on the algorithm. Algorithm selection is a critical point in supervised learning. The result after iterative training rounds is a classifier for the problem in hand (refer Fig. 7.3).

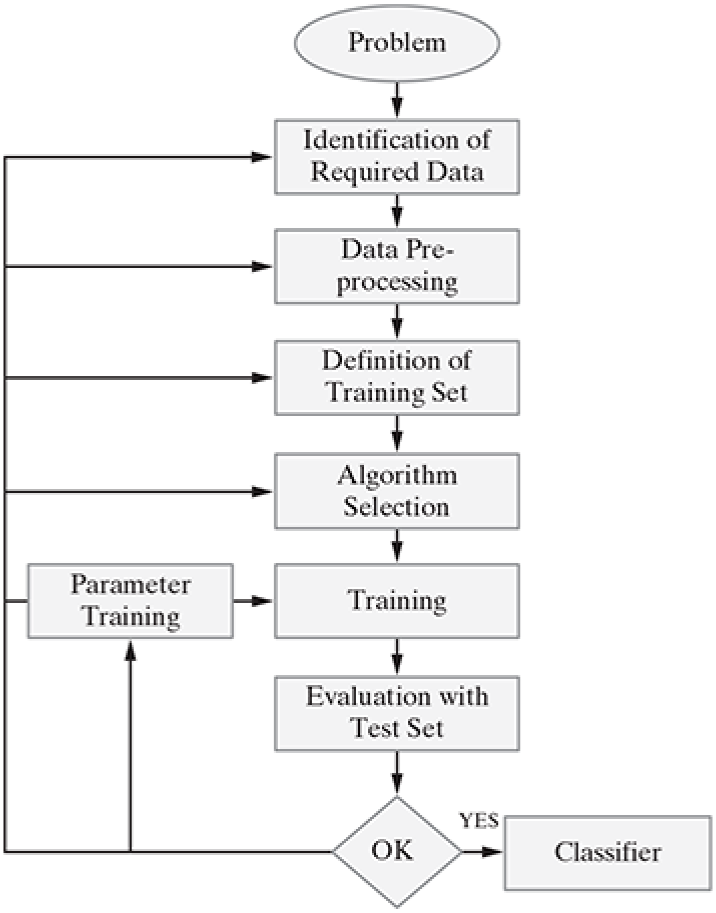


FIG. 7.3 Classification model steps

Problem Identification: Identifying the problem is the first step in the supervised learning model. The problem needs to be a well-formed problem,i.e. a problem with well-defined goals and benefit, which has a long-term impact.

Identification of Required Data: On the basis of the problem identified above, the required data set that precisely represents the identified problem needs to be identified/evaluated. For example: If the problem is to predict whether a tumour is malignant or benign, then the corresponding patient data sets related to malignant tumour and benign tumours are to be identified.

Data Pre-processing: This is related to the cleaning/transforming the data set. This step ensures that all the unnecessary/irrelevant data elements are removed. Data pre-processing refers to the transformations applied to the identified data before feeding the same into the algorithm. Because the data is gathered from different sources, it is usually collected in a raw format and is not ready for immediate analysis. This step ensures that the data is ready to be fed into the machine learning algorithm.

Definition of Training Data Set: Before starting the analysis, the user should decide what kind of data set is to be used as a training set. In the case of signature analysis, for example, the training data set might be a single handwritten alphabet, an entire handwritten word (i.e. a group of the alphabets) or an entire line of handwriting (i.e. sentences or a group of words). Thus, a set of 'input meta-objects' and corresponding 'output meta-objects' are also gathered. The training set needs to be actively representative of the real-world use of the given scenario. Thus, a set of data input (*X*) and corresponding outputs (*Y*) is gathered either from human experts or experiments.

Algorithm Selection: This involves determining the structure of the learning function and the corresponding learning algorithm. This is the most critical step of supervised learning model. On the basis of various parameters, the best algorithm for a given problem is chosen.

Training: The learning algorithm identified in the previous step is run on the gathered training set for further fine tuning. Some supervised learning algorithms require the user to determine specific control parameters (which are given as inputs to the algorithm). These parameters (inputs given to algorithm) may also be adjusted by optimizing performance on a subset (called as validation set) of the training set.

Evaluation with the Test Data Set: Training data is run on the algorithm, and its performance is measured here. If a suitable result is not obtained, further training of parameters may be required.

7.5 COMMON CLASSIFICATION ALGORITHMS

Let us now delve into some common classification algorithms. Following are the most common classification algorithms, out of which we have already learnt about the Naïve Bayes classifier in Chapter 6. We will cover details of the other algorithms in this chapter.

1. *k*-Nearest Neighbour (*k*NN)
2. Decision tree
3. Random forest
4. Support Vector Machine (SVM)
5. Naïve Bayes classifier

7.5.1 *k* -Nearest Neighbour (*k*NN)

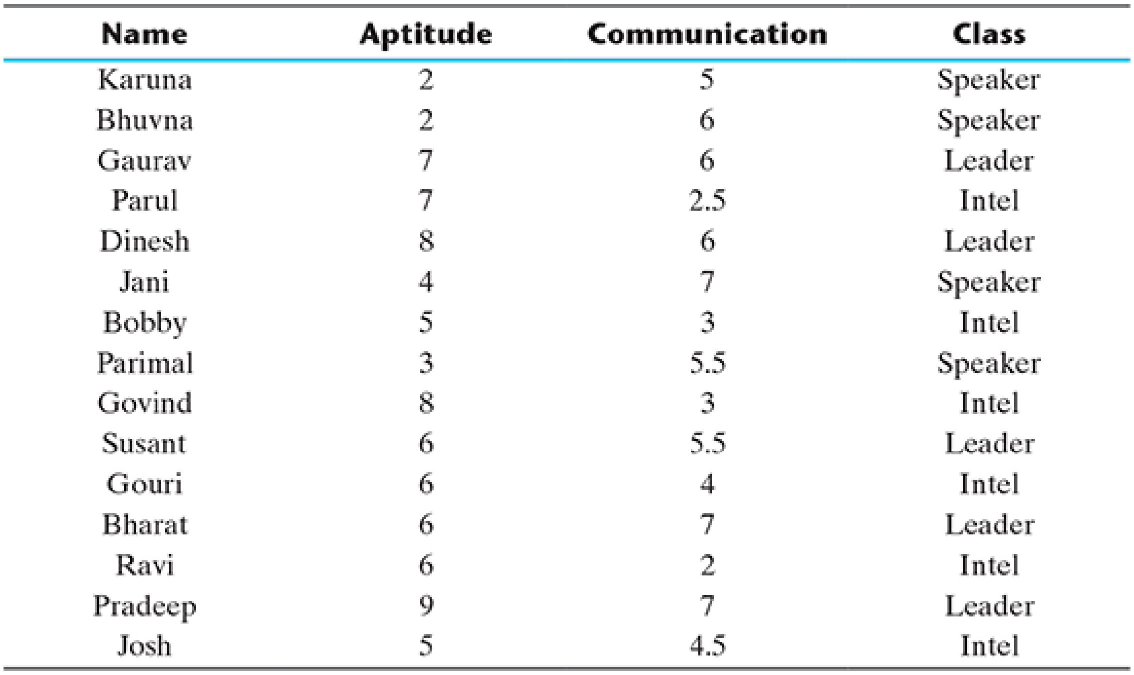
The *k*NN algorithm is a simple but extremely powerful classification algorithm. The name of the algorithm originates from the underlying philosophy of *k*NN – i.e. people having similar background or mindset tend to stay close to each other. In other words, neighbours in a locality have a similar background. In the same way, as a part of the *k*NN algorithm, the unknown and unlabelled data which comes for a prediction problem is judged on the basis of the training data set elements which are similar to the unknown element. So, the class label of the unknown element is assigned on the basis of the class labels of the similar training data set elements (metaphorically can be considered as neighbours of the unknown element). Let us try to understand the algorithm with a simple data set.

7.5.1.1 How kNN works

Let us consider a very simple Student data set as depicted in Figure 7.4. It consists of 15 students studying in a class. Each of the students has been assigned a score on a scale of 10 on two performance parameters – ‘Aptitude’ and ‘Communication’. Also, a class value is assigned to each student based on the following criteria:

1. Students having good communication skills as well as a good level of aptitude have been classified as ‘Leader’
2. Students having good communication skills but not so good level of aptitude have been classified as ‘Speaker’

3. Students having not so good communication skill but a good level of aptitude have been classified as ‘Intel’



Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
Josh	5	4.5	Intel

FIG. 7.4 Student data set

As we have already seen in Chapter 3, while building a classification model, a part of the labelled input data is retained as test data. The remaining portion of the input data is used to train the model – hence known as training data. The motivation to retain a part of the data as test data is to evaluate the performance of the model. As we have seen, the performance of the classification model is measured by the number of correct classifications made by the model when applied to an unknown data set. However, it is not possible during model testing to know the actual label value of an unknown data. Therefore, the test data, which is a part of the labelled input data, is used for this purpose. If the class value predicted for most of the test data elements matches with the actual class value that they have, then we say that the classification model possesses a good accuracy. In context of the Student data set, to keep the things simple, we assume one data element of the input data set as the test data. As depicted in Figure 7.5, the record of the student named Josh is assumed to be the test data. Now that we have the training data and test data identified, we can start with the modelling.

As we have already discussed, in the *k*NN algorithm, the class label of the test data elements is decided by the class label of the training data elements which are neighbouring, i.e. similar in nature. But there are two challenges:

1. What is the basis of this similarity or when can we say that two data elements are similar?
2. How many similar elements should be considered for deciding the class label of each test data element?

To answer the first question, though there are many measures of similarity, the most common approach adopted by *k*NN to measure similarity between two data elements is Euclidean distance. Considering a very simple data set having two features (say f_1 and f_2), Euclidean distance between two data elements d_1 and d_2 can be measured by

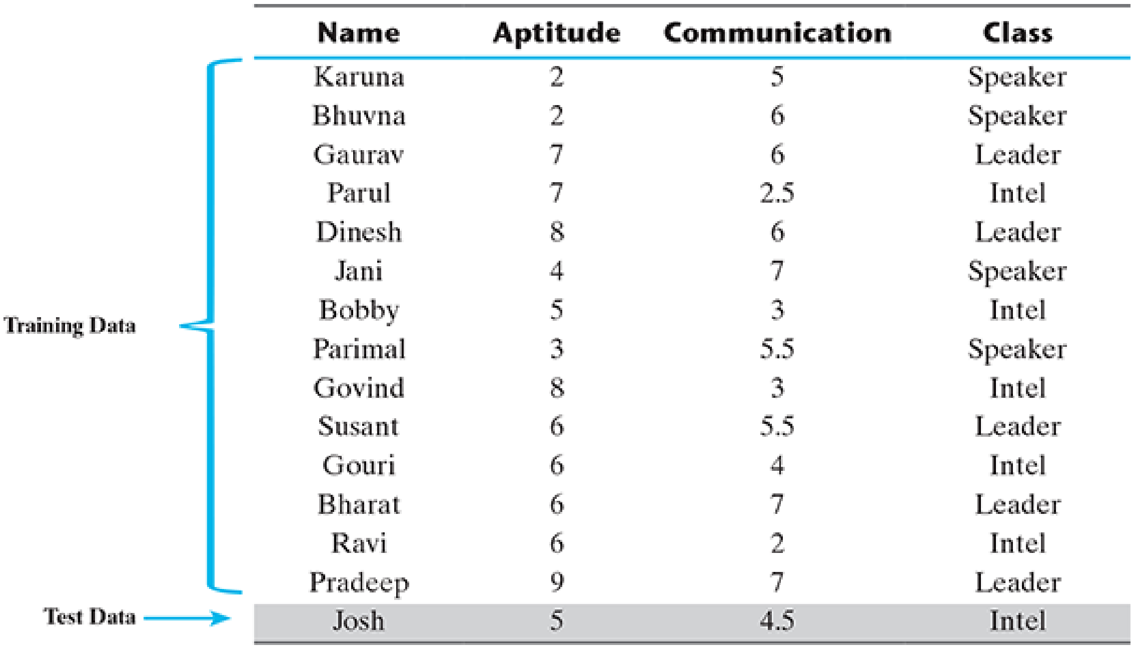
$$\text{Euclidean distance} = \sqrt{(f_{11} - f_{12})^2 + (f_{21} - f_{22})^2}$$

where f_{11} = value of feature f_1 for data element d_1

f_{12} = value of feature f_1 for data element d_2

f_{21} = value of feature f_2 for data element d_1

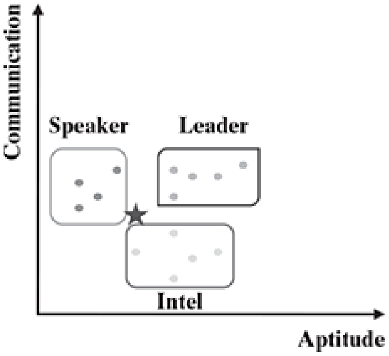
f_{22} = value of feature f_2 for data element d_2

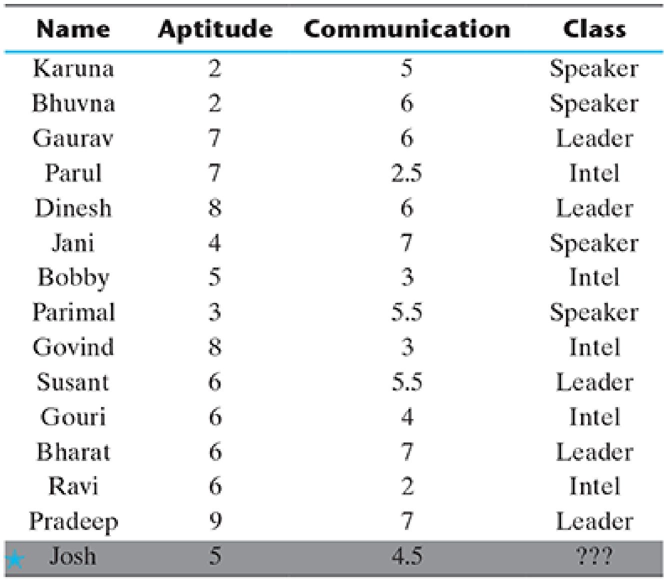

Training Data
Test Data

Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
Josh	5	4.5	Intel

FIG. 7.5 Segregated student data set

So, as depicted in Figure 7.6, the training data points of the Student data set considering only the features ‘Aptitude’ and ‘Communication’ can be represented as dots in a two-dimensional feature space. As shown in the figure, the training data points having the same class value are coming close to each other. The reason for considering two-dimensional data space is that we are considering just the two features of the Student data set, i.e. ‘Aptitude’ and ‘Communication’, for doing the classification. The feature ‘Name’ is ignored because, as we can understand, it has no role to play in deciding the class value. The test data point for student Josh is represented as an asterisk in the same space. To find out the closest or nearest neighbours of the test data point, Euclidean distance of the different dots need to be calculated from the asterisk. Then, the class value of the closest neighbours helps in assigning the class value of the test data element.

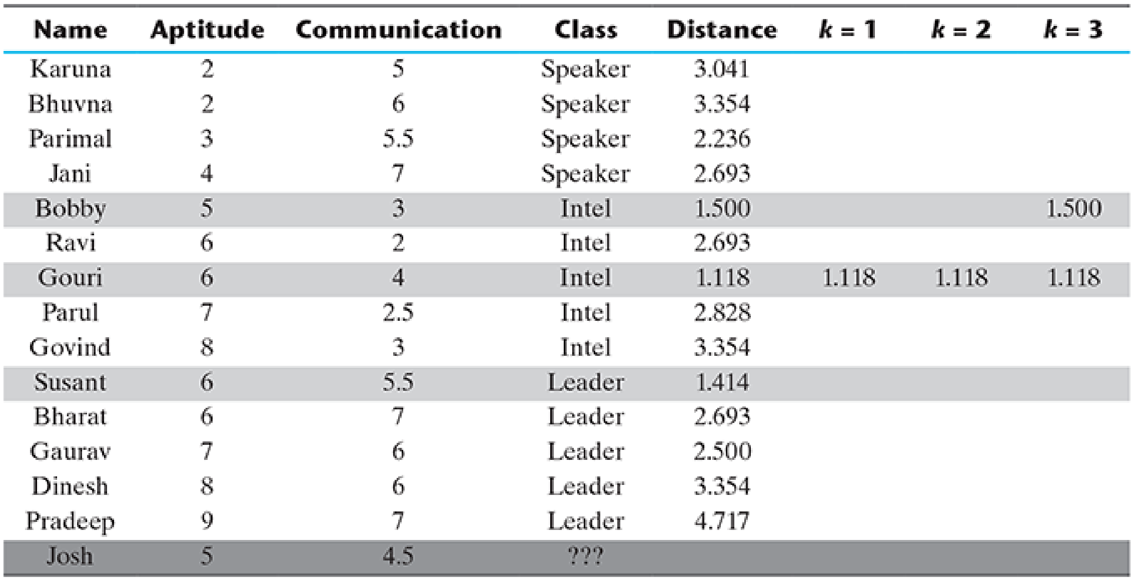




Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
★ Josh	5	4.5	???

FIG. 7.6 2-D representation of the student data set

Now, let us try to find the answer to the second question, i.e. how many similar elements should be considered. The answer lies in the value of '*k*' which is a user-defined parameter given as an input to the algorithm. In the *k*NN algorithm, the value of '*k*' indicates the number of neighbours that need to be considered. For example, if the value of *k* is 3, only three nearest neighbours or three training data elements closest to the test data element are considered. Out of the three data elements, the class which is predominant is considered as the class label to be assigned to the test data. In case the value of *k* is 1, only the closest training data element is considered. The class label of that data element is directly assigned to the test data element. This is depicted in Figure 7.7.



Name	Aptitude	Communication	Class	Distance	*k* = 1	*k* = 2	*k* = 3
Karuna	2	5	Speaker	3.041			
Bhuvna	2	6	Speaker	3.354			
Parimal	3	5.5	Speaker	2.236			
Jani	4	7	Speaker	2.693			
Bobby	5	3	Intel	1.500			1.500
Ravi	6	2	Intel	2.693			
Gouri	6	4	Intel	1.118	1.118	1.118	1.118
Parul	7	2.5	Intel	2.828			
Govind	8	3	Intel	3.354			
Susant	6	5.5	Leader	1.414			
Bharat	6	7	Leader	2.693			
Gaurav	7	6	Leader	2.500			
Dinesh	8	6	Leader	3.354			
Pradeep	9	7	Leader	4.717			
Josh	5	4.5	???				

FIG. 7.7 Distance calculation between test and training points

Let us now try to find out the outcome of the algorithm for the Student data set we have. In other words, we want to see what class value *k*NN will assign for the test data for student Josh. Again, let us refer back to Figure 7.7. As is evident, when the value of *k* is taken as 1, only one training data point needs to be considered. The training record for student Gouri comes as the closest one to test record of Josh, with a distance value of 1.118. Gouri has class value 'Intel'. So, the test data point is also assigned a class label value 'Intel'. When the value of *k* is assumed as 3, the closest neighbours of Josh in the training data set are Gouri, Susant, and Bobby with distances being 1.118, 1.414, and 1.5, respectively. Gouri and Bobby have class value 'Intel', while Susant has class value 'Leader'. In this case, the class value of Josh is decided by majority voting. Because the class value of 'Intel' is formed by the majority of the neighbours, the class value of Josh is assigned as 'Intel'. This same process can be extended for any value of *k*.

But it is often a tricky decision to decide the value of *k*. The reasons are as follows:

- If the value of *k* is very large (in the extreme case equal to the total number of records in the training data), the class label of the majority

class of the training data set will be assigned to the test data regardless of the class labels of the neighbours nearest to the test data.
- If the value of *k* is very small (in the extreme case equal to 1), the class value of a noisy data or outlier in the training data set which is the nearest neighbour to the test data will be assigned to the test data.

The best *k* value is somewhere between these two extremes.

Few strategies, highlighted below, are adopted by machine learning practitioners to arrive at a value for *k*.

- One common practice is to set *k* equal to the square root of the number of training records.
- An alternative approach is to test several *k* values on a variety of test data sets and choose the one that delivers the best performance.
- Another interesting approach is to choose a larger value of *k*, but apply a weighted voting process in which the vote of close neighbours is considered more influential than the vote of distant neighbours.

7.5.1.2 kNN algorithm

Input: Training data set, test data set (or data points), value of '*k*' (i.e. number of nearest neighbours to be considered)

Steps:

Do for all test data points

Calculate the distance (usually Euclidean distance) of the test data point from the different training data points.

Find the closest '*k*' training data points, i.e. training data points whose distances are least from the test data point.

If *k* = 1

Then assign class label of the training data point to the test data point

Else

Whichever class label is predominantly present in the training data points, assign that class label to the test data point

End do

7.5.1.3 Why the kNN algorithm is called a lazy learner?

We have already discussed in Chapter 3 that eager learners follow the general steps of machine learning, i.e. perform an abstraction of the information obtained from the input data and then follow it through by a generalization step. However, as we have seen in the case of the kNN algorithm, these steps are completely skipped. It stores the training data and directly applies the philosophy of nearest neighbourhood finding to arrive at the classification. So, for *k*NN, there is no learning happening in the real sense. Therefore, *k*NN falls under the category of lazy learner.

7.5.1.4 Strengths of the kNN algorithm

- Extremely simple algorithm – easy to understand
- Very effective in certain situations, e.g. for recommender system design
- Very fast or almost no time required for the training phase

7.5.1.5 Weaknesses of the kNN algorithm

- Does not learn anything in the real sense. Classification is done completely on the basis of the training data. So, it has a heavy reliance on the training data. If the training data does not represent the problem domain comprehensively, the algorithm fails to make an effective classification.
- Because there is no model trained in real sense and the classification is done completely on the basis of the training data, the classification

process is very slow.

- Also, a large amount of computational space is required to load the training data for classification.

7.5.1.6 Application of the kNN algorithm

One of the most popular areas in machine learning where the *k*NN algorithm is widely adopted is recommender systems. As we know, recommender systems recommend users different items which are similar to a particular item that the user seems to like. The liking pattern may be revealed from past purchases or browsing history and the similar items are identified using the *k*NN algorithm.

Another area where there is widespread adoption of *k*NN is searching documents/ contents similar to a given document/content. This is a core area under information retrieval and is known as concept search.

7.5.2 Decision tree

Decision tree learning is one of the most widely adopted algorithms for classification. As the name indicates, it builds a model in the form of a tree structure. Its grouping exactness is focused with different strategies, and it is exceptionally productive.

A decision tree is used for multi-dimensional analysis with multiple classes. It is characterized by fast execution time and ease in the interpretation of the rules. The goal of decision tree learning is to create a model (based on the past data called past vector) that predicts the value of the output variable based on the input variables in the feature vector.

Each node (or decision node) of a decision tree corresponds to one of the feature vector. From every node, there are edges to children, wherein there is an edge for each of the possible values (or range of values) of the feature associated with the node. The tree terminates at different leaf nodes (or terminal nodes) where each leaf node represents a possible value for the output variable. The output variable is determined by following a path that starts at the root and is guided by the values of the input variables.

A decision tree is usually represented in the format depicted in Figure 7.8.

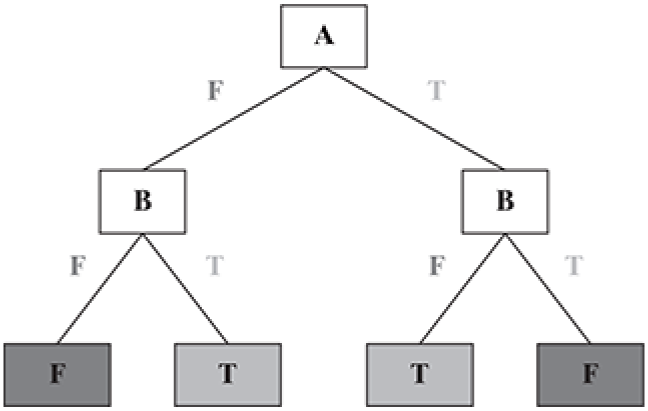


FIG. 7.8 Decision tree structure

Each internal node (represented by boxes) tests an attribute (represented as ‘A’/‘B’ within the boxes). Each branch corresponds to an attribute value (T/F) in the above case. Each leaf node assigns a classification. The first node is called as ‘Root’ Node. Branches from the root node are called as ‘Leaf’ Nodes where ‘A’ is the Root Node (first node). ‘B’ is the Branch Node. ‘T’ & ‘F’ are Leaf Nodes.

Thus, a decision tree consists of three types of nodes:

- Root Node
- Branch Node
- Leaf Node

Figure 7.9 shows an example decision tree for a car driving – the decision to be taken is whether to ‘Keep Going’ or to ‘Stop’, which depends on various situations as depicted in the figure. If the signal is RED in colour, then the car should be stopped. If there is not enough gas (petrol) in the car, the car should be stopped at the next available gas station.

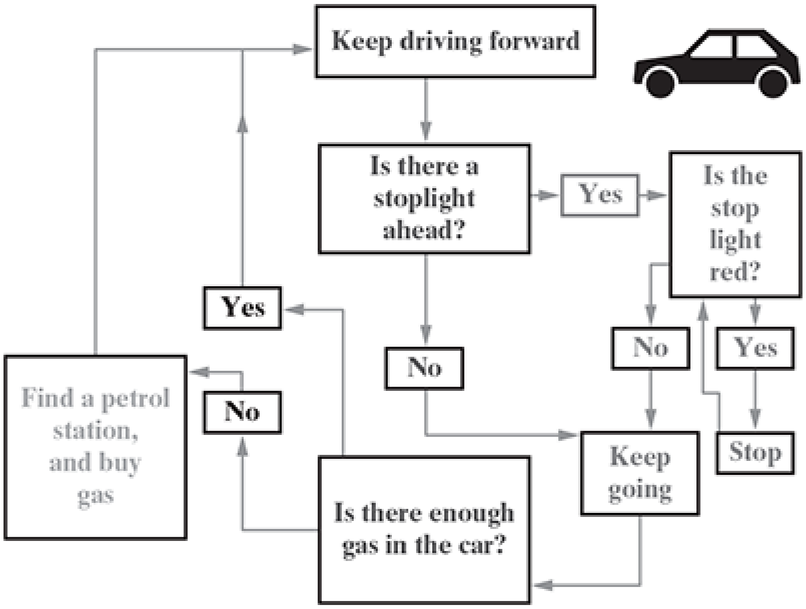

Keep driving forward
Is there a stoplight ahead?
Yes
Is the stop light red?
No
Yes
Yes
No
Find a petrol station, and buy gas
No
Keep going
Stop
Is there enough gas in the car?

FIG. 7.9 Decision tree example

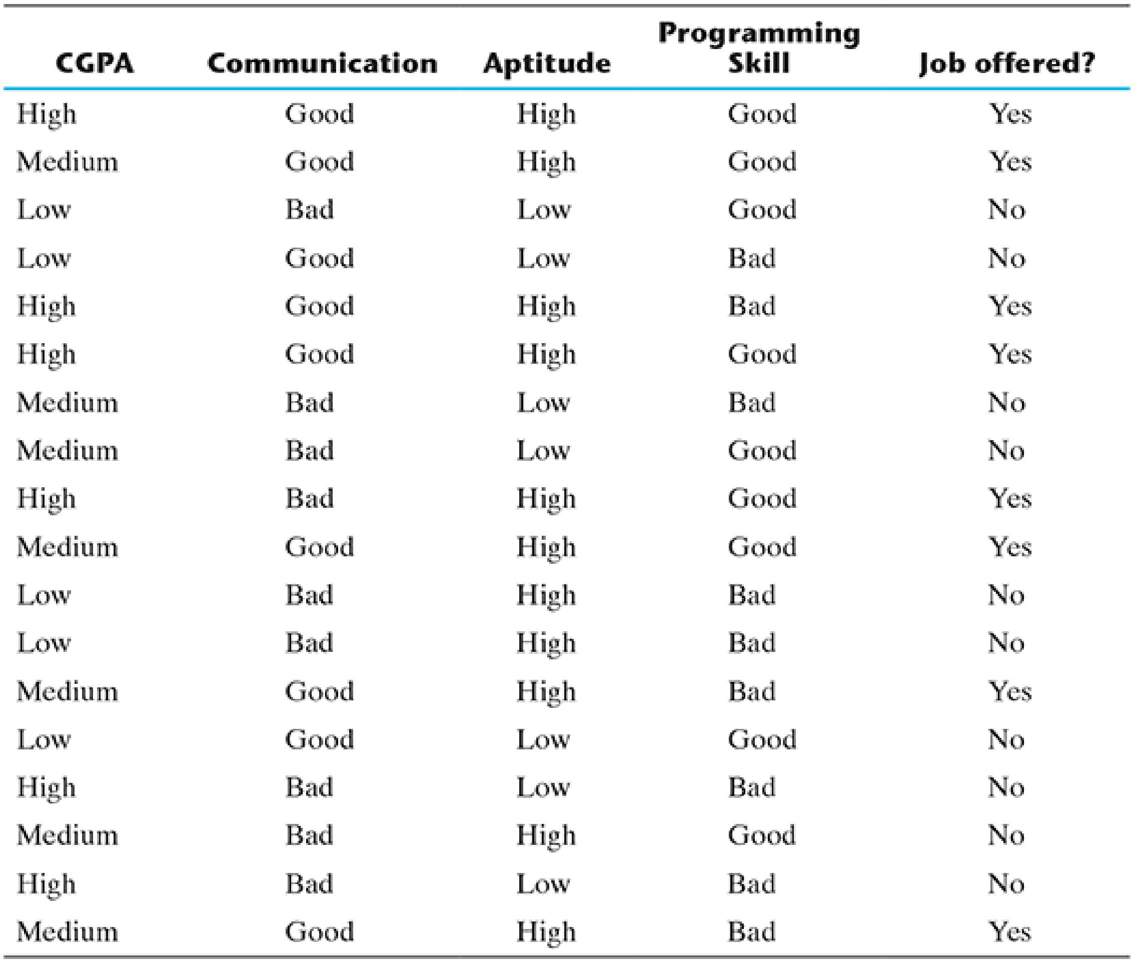
7.5.2.1 **Building a decision tree**

Decision trees are built corresponding to the training data following an approach called recursive partitioning. The approach splits the data into multiple subsets on the basis of the feature values. It starts from the root node, which is nothing but the entire data set. It first selects the feature which predicts the target class in the strongest way. The decision tree splits the data set into multiple partitions, with data in each partition having a distinct value for the feature based on which the partitioning has happened. This is the first set of branches. Likewise, the algorithm continues splitting the nodes on the basis of the feature which helps in the best partition. This continues till a stopping criterion is reached. The usual stopping criteria are –

1. All or most of the examples at a particular node have the same class
2. All features have been used up in the partitioning
3. The tree has grown to a pre-defined threshold limit

Let us try to understand this in the context of an example. Global Technology Solutions (GTS), a leading provider of IT solutions, is coming to College of Engineering and Management (CEM) for hiring B.Tech. students. Last year during campus recruitment, they had shortlisted 18 students for the final interview. Being a company of international repute, they follow a stringent interview process to select only the best of the students. The information related to the interview evaluation results of shortlisted students (hiding the names) on the basis of different evaluation parameters is available for reference in Figure 7.10. Chandra, a student of CEM, wants to find out if he may be offered a job in GTS. His CGPA is quite high. His self-evaluation on the other parameters is as follows:

Communication – Bad; Aptitude – High; Programming skills – Bad



CGPA	Communication	Aptitude	Programming Skill	Job offered?
High	Good	High	Good	Yes
Medium	Good	High	Good	Yes
Low	Bad	Low	Good	No
Low	Good	Low	Bad	No
High	Good	High	Bad	Yes
High	Good	High	Good	Yes
Medium	Bad	Low	Bad	No
Medium	Bad	Low	Good	No
High	Bad	High	Good	Yes
Medium	Good	High	Good	Yes
Low	Bad	High	Bad	No
Low	Bad	High	Bad	No
Medium	Good	High	Bad	Yes
Low	Good	Low	Good	No
High	Bad	Low	Bad	No
Medium	Bad	High	Good	No
High	Bad	Low	Bad	No
Medium	Good	High	Bad	Yes

FIG. 7.10 Training data for GTS recruitment

Let us try to solve this problem, i.e. predicting whether Chandra will get a job offer, by using the decision tree model. First, we need to draw the decision tree corresponding to the training data given in Figure 7.10. According to the table, job offer condition (i.e. the outcome) is FALSE for all the cases where **Aptitude** = **Low**, irrespective of other conditions. So, the feature Aptitude can be taken up as the first node of the decision tree.

For **Aptitude** = **High**, job offer condition is TRUE for all the cases where **Communication** = **Good**. For cases where **Communication** = **Bad**, job offer condition is TRUE for all the cases where **CGPA** = **High**.

Figure 7.11 depicts the complete decision tree diagram for the table given in Figure 7.10.

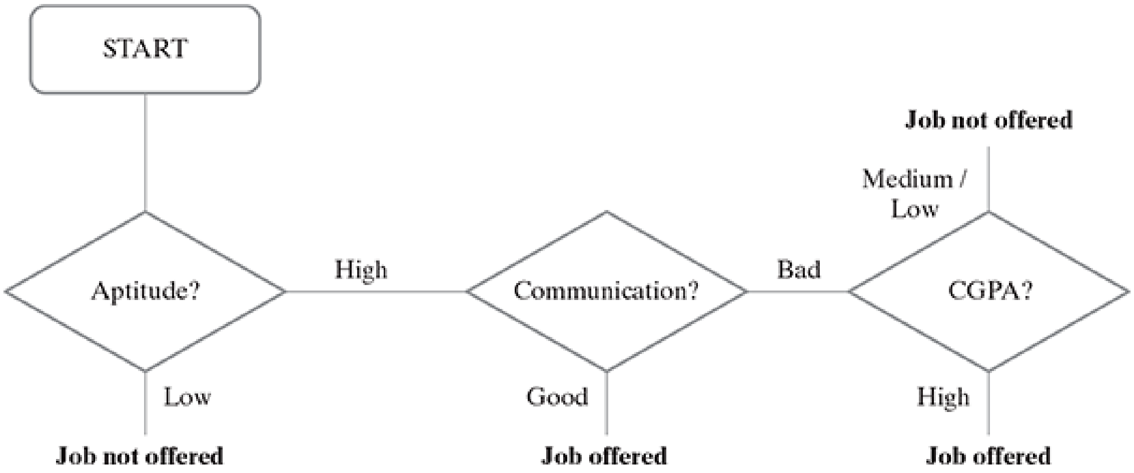


FIG. 7.11 Decision tree based on the training data

7.5.2.2 Searching a decision tree

By using the above decision tree depicted in Figure 7.11, we need to predict whether Chandra might get a job offer for the given parameter values: CGPA = **High**, Communication = **Bad**, Aptitude = **High**, Programming skills = **Bad**. There are multiple ways to search through the trained decision tree for a solution to the given prediction problem.

Exhaustive search

1. Place the item in the first group (class). Recursively examine solutions with the item in the first group (class).
2. Place the item in the second group (class). Recursively examine solutions with the item in the second group (class).

3. Repeat the above steps until the solution is reached.

Exhaustive search travels through the decision tree exhaustively, but it will take much time when the decision tree is big with multiple leaves and multiple attribute values.

Branch and bound search

Branch and bound uses an existing best solution to sidestep searching of the entire decision tree in full. When the algorithm starts, the best solution is well defined to have the worst possible value; thus, any solution it finds out is an improvement. This makes the algorithm initially run down to the left-most branch of the tree, even though that is unlikely to produce a realistic result. In the partitioning problem, that solution corresponds to putting every item in one group, and it is an unacceptable solution. A programme can speed up the process by using a fast heuristic to find an initial solution. This can be used as an input for branch and bound. If the heuristic is right, the savings can be substantial.

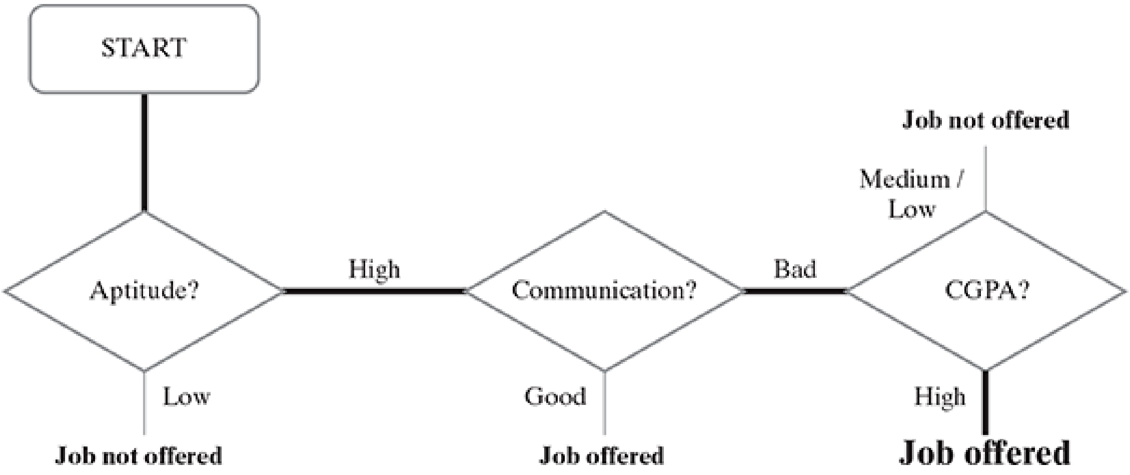


FIG. 7.12 Decision tree based on the training data (depicting a sample path)

Figure 7.12 depicts a sample path (thick line) for the conditions CGPA = **High**, Communication = Bad, Aptitude = High and Programming skills = Bad. According to the above decision tree, the prediction can be made as Chandra **will get the job offer**.

There are many implementations of decision tree, the most prominent ones being C5.0, CART (Classification and Regression Tree), CHAID (Chi-square Automatic Interaction Detector) and ID3 (Iterative Dichotomiser 3) algorithms. The biggest challenge of a decision tree algorithm is to find out which feature to split upon. The main driver for identifying the feature is that the data should be split in such a way that the partitions created by the split should contain examples belonging to a single class. If that happens, the partitions are considered to be **pure**. Entropy is a measure of impurity of an attribute or feature adopted by many algorithms such as ID3 and C5.0. The information gain is calculated on the basis of the decrease in entropy (S) after a data set is split according to a particular attribute (A). Constructing a decision tree is all about finding an attribute that returns the highest information gain (i.e. the most homogeneous branches).

Note:

Like information gain, there are other measures like Gini index or chi-square for individual nodes to decide the feature on the basis of which the split has to be applied. The CART algorithm uses Gini index, while the CHAID algorithm uses chi-square for deciding the feature for applying split.

7.5.2.3 Entropy of a decision tree

Let us say S is the sample set of training examples. Then, Entropy (S) measuring the impurity of S is defined as

$$\textbf{Entropy}(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

where c is the number of different class labels and p refers to the proportion of values falling into the *i*-th class label.

For example, with respect to the training data in Figure 7.10, we have two values for the target class ‘Job Offered?’ – Yes and No. The value of p_i

for class value 'Yes' is 0.44 (i.e. 8/18) and that for class value 'No' is 0.56 (i.e. 10/18). So, we can calculate the entropy as

$$\textbf{Entropy}(S) = -0.44 \log_2(0.44) - 0.56 \log_2(0.56) = 0.99.$$

7.5.2.4 Information gain of a decision tree

The information gain is created on the basis of the decrease in entropy (S) after a data set is split according to a particular attribute (A). Constructing a decision tree is all about finding an attribute that returns the highest information gain (i.e. the most homogeneous branches). If the information gain is 0, it means that there is no reduction in entropy due to split of the data set according to that particular feature. On the other hand, the maximum amount of information gain which may happen is the entropy of the data set before the split.

Information gain for a particular feature A is calculated by the difference in entropy before a split (or $S_{\mathbf{bs}}$) with the entropy after the split ($S_{\mathbf{as}}$).

$$\textbf{Information Gain (S, A) = Entropy (S}_{\textbf{bs}}\textbf{) – Entropy (S}_{\textbf{as}}\textbf{)}$$

For calculating the entropy after split, entropy for all partitions needs to be considered. Then, the weighted summation of the entropy for each partition can be taken as the total entropy after split. For performing weighted summation, the proportion of examples falling into each partition is used as weight.

$$\textbf{Entropy}(\boldsymbol{S}_{\text{as}}) = \sum_{i=1}^{n} w_i \, \text{Entropy}\,(p_i)$$

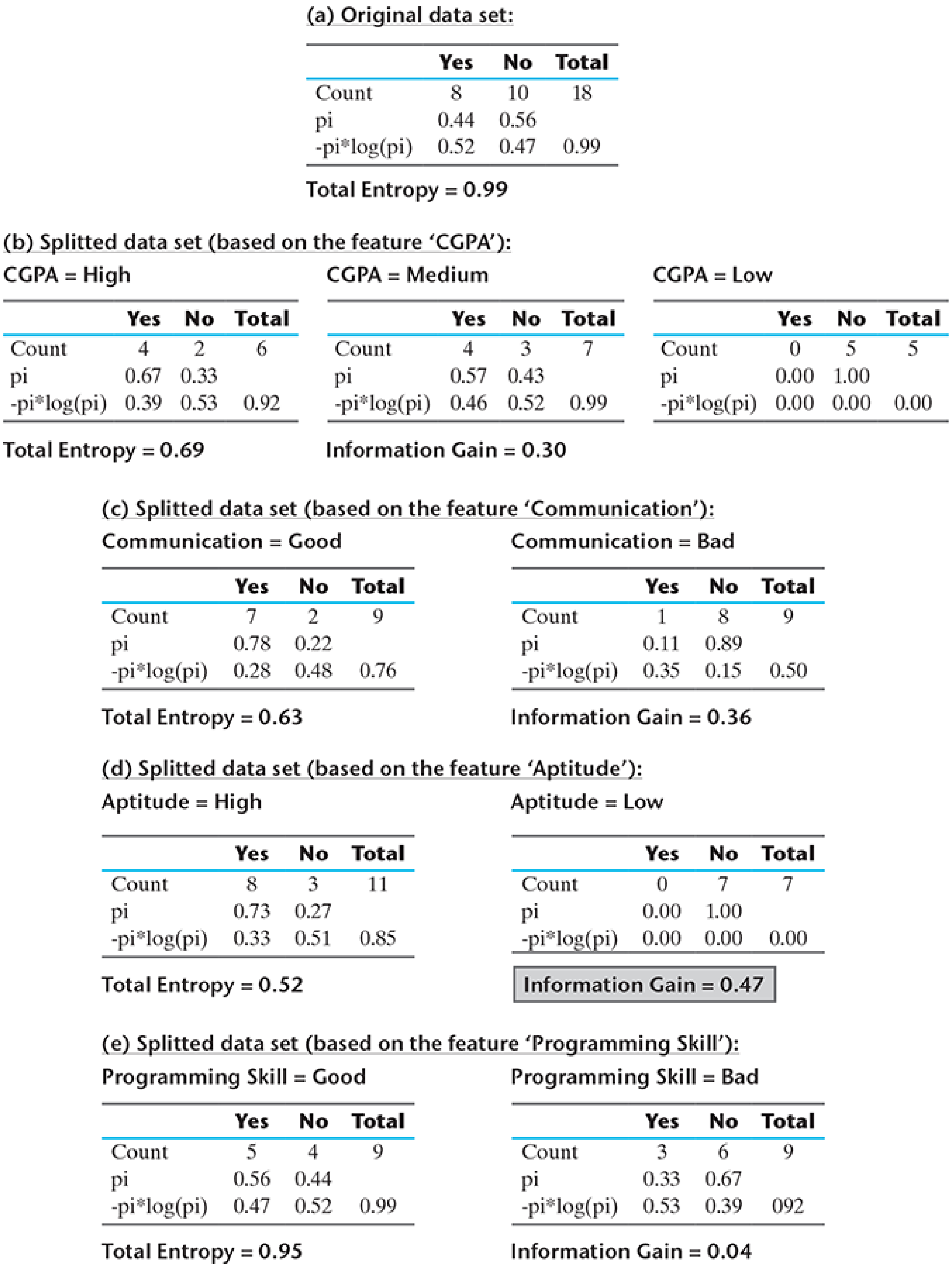
Let us examine the value of information gain for the training data set shown in Figure 7.10. We will find the value of entropy at the beginning before any split happens and then again after the split happens. We will compare the values for all the cases –

1. when the feature ‘CGPA’ is used for the split
2. when the feature ‘Communication’ is used for the split
3. when the feature ‘Aptitude’ is used for the split
4. when the feature ‘Programming Skills’ is used for the split

Figure 7.13a gives the entropy values for the first level split for each of the cases mentioned above.

As calculated, entropy of the data set before split (i.e. Entropy ($S_{\mathbf{bs}}$)) = 0.99, and entropy of the data set after split (i.e. Entropy ($S_{\mathbf{as}}$)) is

- 0.69 when the feature ‘CGPA’ is used for split
- 0.63 when the feature ‘Communication’ is used for split
- 0.52 when the feature ‘Aptitude’ is used for split
- 0.95 when the feature ‘Programming skill’ is used for split



(a) Original data set:

	Yes	No	Total
Count	8	10	18
pi	0.44	0.56	
-pi*log(pi)	0.52	0.47	0.99

Total Entropy = 0.99

(b) Splitted data set (based on the feature 'CGPA'):

CGPA = High

	Yes	No	Total
Count	4	2	6
pi	0.67	0.33	
-pi*log(pi)	0.39	0.53	0.92

CGPA = Medium

	Yes	No	Total
Count	4	3	7
pi	0.57	0.43	
-pi*log(pi)	0.46	0.52	0.99

CGPA = Low

	Yes	No	Total
Count	0	5	5
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

Total Entropy = 0.69

Information Gain = 0.30

(c) Splitted data set (based on the feature 'Communication'):

Communication = Good

	Yes	No	Total
Count	7	2	9
pi	0.78	0.22	
-pi*log(pi)	0.28	0.48	0.76

Communication = Bad

	Yes	No	Total
Count	1	8	9
pi	0.11	0.89	
-pi*log(pi)	0.35	0.15	0.50

Total Entropy = 0.63

Information Gain = 0.36

(d) Splitted data set (based on the feature 'Aptitude'):

Aptitude = High

	Yes	No	Total
Count	8	3	11
pi	0.73	0.27	
-pi*log(pi)	0.33	0.51	0.85

Aptitude = Low

	Yes	No	Total
Count	0	7	7
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

Total Entropy = 0.52

Information Gain = 0.47

(e) Splitted data set (based on the feature 'Programming Skill'):

Programming Skill = Good

	Yes	No	Total
Count	5	4	9
pi	0.56	0.44	
-pi*log(pi)	0.47	0.52	0.99

Programming Skill = Bad

	Yes	No	Total
Count	3	6	9
pi	0.33	0.67	
-pi*log(pi)	0.53	0.39	092

Total Entropy = 0.95

Information Gain = 0.04

FIG. 7.13A Entropy and information gain calculation (Level 1)

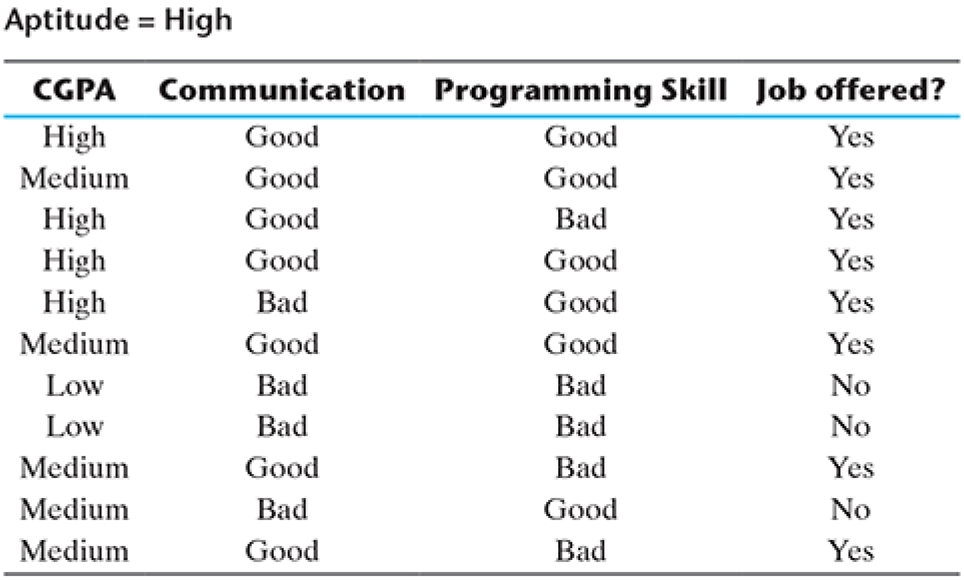
Therefore, the information gain from the feature 'CGPA' = 0.99 − 0.69 = 0.3, whereas the information gain from the feature 'Communication' = 0.99 − 0.63 = 0.36. Likewise, the information gain for 'Aptitude' and 'Programming skills' is 0.47 and 0.04, respectively.

Hence, it is quite evident that among all the features, 'Aptitude' results in the best information gain when adopted for the split. So, at the first level, a split will be applied according to the value of 'Aptitude' or in other words, 'Aptitude' will be the first node of the decision tree formed. One important point to be noted here is that for **Aptitude** = **Low**, entropy is 0, which indicates that always the result will be the same irrespective of the values of the other features. Hence, the branch towards Aptitude = Low will not continue any further.

As a part of level 2, we will thus have only one branch to navigate in this case – the one for **Aptitude** = **High**. Figure 7.13b presents calculations for level 2. As can be seen from the figure, the entropy value is as follows:

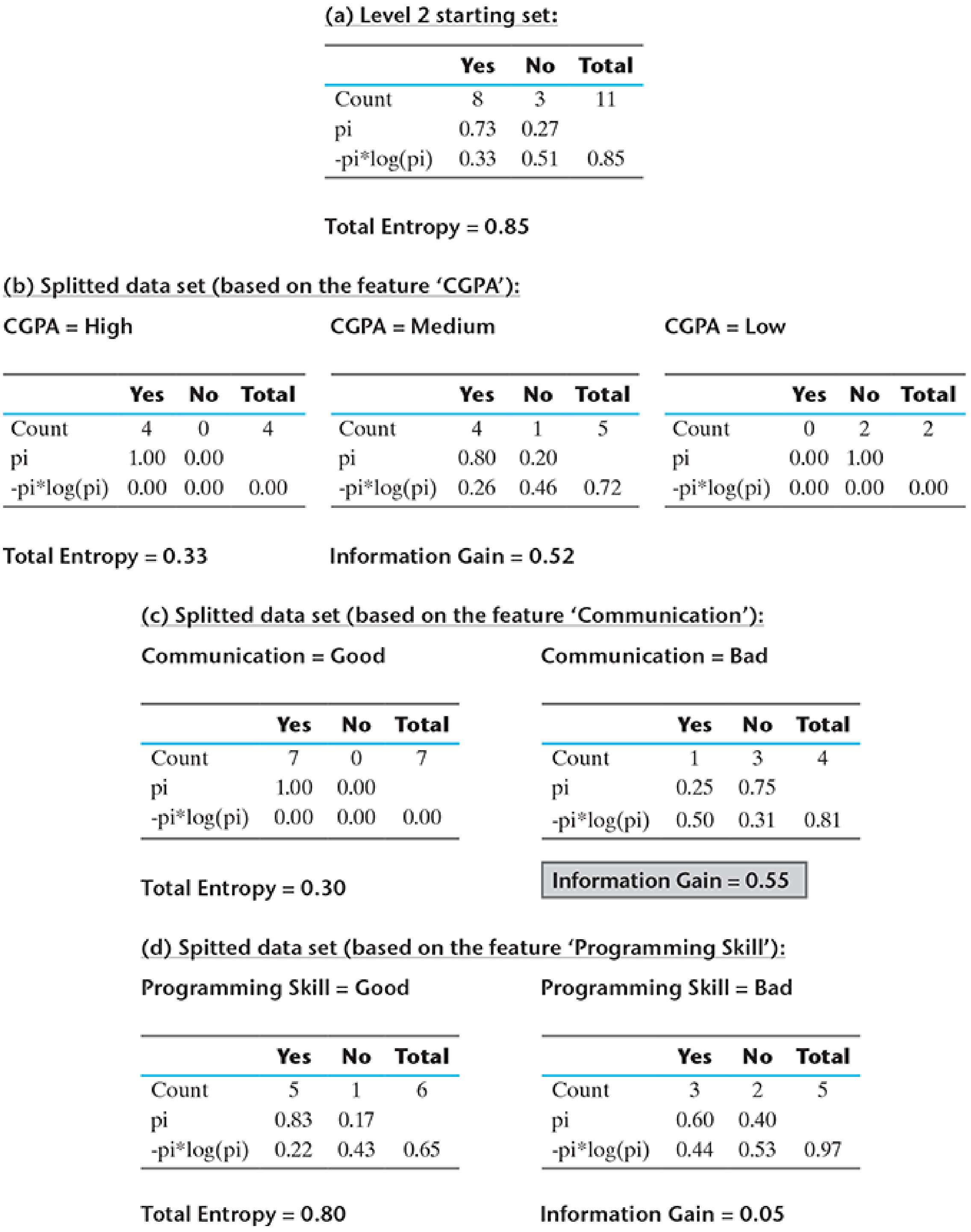
- 0.85 before the split
- 0.33 when the feature 'CGPA' is used for split
- 0.30 when the feature 'Communication' is used for split
- 0.80 when the feature 'Programming skill' is used for split

Hence, the information gain after split with the features CGPA, Communication and Programming Skill is 0.52, 0.55 and 0.05, respectively. Hence, the feature Communication should be used for this split as it results in the highest information gain. So, at the second level, a split will be applied on the basis of the value of 'Communication'. Again, the point to be noted here is that for **Communication** = **Good**, entropy is 0, which indicates that always the result will be the same irrespective of the values of the other features. Hence, the branch towards Communication = Good will not continue any further.



Aptitude = High

CGPA	Communication	Programming Skill	Job offered?
High	Good	Good	Yes
Medium	Good	Good	Yes
High	Good	Bad	Yes
High	Good	Good	Yes
High	Bad	Good	Yes
Medium	Good	Good	Yes
Low	Bad	Bad	No
Low	Bad	Bad	No
Medium	Good	Bad	Yes
Medium	Bad	Good	No
Medium	Good	Bad	Yes



(a) Level 2 starting set:

	Yes	No	Total
Count	8	3	11
pi	0.73	0.27	
-pi*log(pi)	0.33	0.51	0.85

Total Entropy = 0.85

(b) Splitted data set (based on the feature 'CGPA'):

CGPA = High

	Yes	No	Total
Count	4	0	4
pi	1.00	0.00	
-pi*log(pi)	0.00	0.00	0.00

CGPA = Medium

	Yes	No	Total
Count	4	1	5
pi	0.80	0.20	
-pi*log(pi)	0.26	0.46	0.72

CGPA = Low

	Yes	No	Total
Count	0	2	2
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

Total Entropy = 0.33

Information Gain = 0.52

(c) Splitted data set (based on the feature 'Communication'):

Communication = Good

	Yes	No	Total
Count	7	0	7
pi	1.00	0.00	
-pi*log(pi)	0.00	0.00	0.00

Communication = Bad

	Yes	No	Total
Count	1	3	4
pi	0.25	0.75	
-pi*log(pi)	0.50	0.31	0.81

Total Entropy = 0.30

Information Gain = 0.55

(d) Spitted data set (based on the feature 'Programming Skill'):

Programming Skill = Good

	Yes	No	Total
Count	5	1	6
pi	0.83	0.17	
-pi*log(pi)	0.22	0.43	0.65

Programming Skill = Bad

	Yes	No	Total
Count	3	2	5
pi	0.60	0.40	
-pi*log(pi)	0.44	0.53	0.97

Total Entropy = 0.80

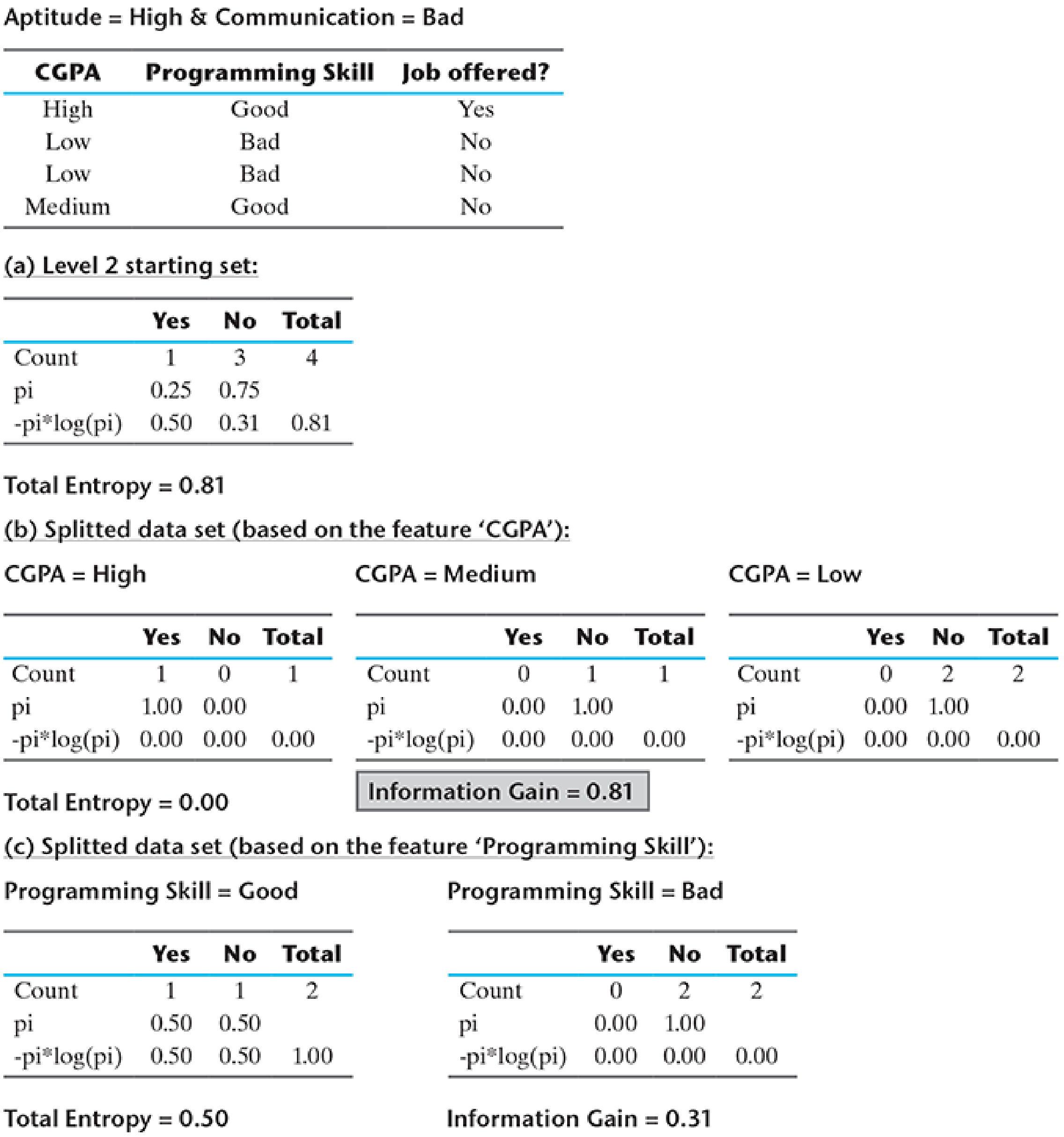
Information Gain = 0.05

FIG. 7.13B Entropy and information gain calculation (Level 2)

As a part of level 3, we will thus have only one branch to navigate in this case – the one for **Communication** = **Bad**. Figure 7.13c presents calculations for level 3. As can be seen from the figure, the entropy value is as follows:

- 0.81 before the split

- 0 when the feature ‘CGPA’ is used for split
- 0.50 when the feature ‘Programming Skill’ is used for split



Aptitude = High & Communication = Bad

CGPA	Programming Skill	Job offered?
High	Good	Yes
Low	Bad	No
Low	Bad	No
Medium	Good	No

(a) Level 2 starting set:

	Yes	No	Total
Count	1	3	4
pi	0.25	0.75	
-pi*log(pi)	0.50	0.31	0.81

Total Entropy = 0.81

(b) Splitted data set (based on the feature ‘CGPA’):

CGPA = High

	Yes	No	Total
Count	1	0	1
pi	1.00	0.00	
-pi*log(pi)	0.00	0.00	0.00

CGPA = Medium

	Yes	No	Total
Count	0	1	1
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

CGPA = Low

	Yes	No	Total
Count	0	2	2
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

Total Entropy = 0.00

Information Gain = 0.81

(c) Splitted data set (based on the feature ‘Programming Skill’):

Programming Skill = Good

	Yes	No	Total
Count	1	1	2
pi	0.50	0.50	
-pi*log(pi)	0.50	0.50	1.00

Programming Skill = Bad

	Yes	No	Total
Count	0	2	2
pi	0.00	1.00	
-pi*log(pi)	0.00	0.00	0.00

Total Entropy = 0.50

Information Gain = 0.31

FIG. 7.13C Entropy and information gain calculation (Level 3)

Hence, the information gain after split with the feature CGPA is 0.81, which is the maximum possible information gain (as the entropy before split was 0.81). Hence, as obvious, a split will be applied on the basis of the value of ‘CGPA’. Because the maximum information gain is already achieved, the tree will not continue any further.

7.5.2.5 Algorithm for decision tree

Input: Training data set, test data set (or data points)

Steps:

```
Do for all attributes
    Calculate the entropy $E_i$ of the attribute $F_i$
    if $E_i < E_{\mathbf{min}}$
        then $E_{\mathbf{min}} = E_{\mathbf{i}}$ and $F_{\mathbf{min}} = F_i$
    end if
End do
    Split the data set into subsets using the attribute $F_{\mathbf{min}}$
```

Draw a decision tree node containing the attribute $F_{\mathbf{min}}$ and split the data set into subsets

Repeat the above steps until the full tree is drawn covering all the attributes of the original table.

7.5.2.6 Avoiding overfitting in decision tree – pruning

The decision tree algorithm, unless a stopping criterion is applied, may keep growing indefinitely – splitting for every feature and dividing into smaller partitions till the point that the data is perfectly classified. This, as is quite evident, results in overfitting problem. To prevent a decision tree getting overfitted to the training data, pruning of the decision tree is essential. Pruning a decision tree reduces the size of the tree such that the model is more generalized and can classify unknown and unlabelled data in a better way.

There are two approaches of pruning:

- Pre-pruning: Stop growing the tree before it reaches perfection.
- Post-pruning: Allow the tree to grow entirely and then post-prune some of the branches from it.

In the case of pre-pruning, the tree is stopped from further growing once it reaches a certain number of decision nodes or decisions. Hence, in this strategy, the algorithm avoids overfitting as well as optimizes computational cost. However, it also stands a chance to ignore important information contributed by a feature which was skipped, thereby resulting in miss out of certain patterns in the data.

On the other hand, in the case of post-pruning, the tree is allowed to grow to the full extent. Then, by using certain pruning criterion, e.g. error rates at the nodes, the size of the tree is reduced. This is a more effective approach in terms of classification accuracy as it considers all minute information available from the training data. However, the computational cost is obviously more than that of pre-pruning.

7.5.2.7 **Strengths of decision tree**

- It produces very simple understandable rules. For smaller trees, not much mathematical and computational knowledge is required to understand this model.
- Works well for most of the problems.
- It can handle both numerical and categorical variables.
- Can work well both with small and large training data sets.
- Decision trees provide a definite clue of which features are more useful for classification.

7.5.2.8 Weaknesses of decision tree

- Decision tree models are often biased towards features having more number of possible values, i.e. levels.
- This model gets overfitted or underfitted quite easily.

- Decision trees are prone to errors in classification problems with many classes and relatively small number of training examples.
- A decision tree can be computationally expensive to train.
- Large trees are complex to understand.

7.5.2.9 Application of decision tree

Decision tree can be applied in a data set in which there is a finite list of attributes and each data instance stores a value for that attribute (e.g. ‘High’ for the attribute CGPA). When each attribute has a small number of distinct values (e.g. ‘High’, ‘Medium’, ‘Low’), it is easier/quicker for the decision tree to suggest (or choose) an effective solution. This algorithm can be extended to handle real-value attributes (e.g. a floating point temperature).

The most straightforward case exists when there are only two possible values for an attribute (Boolean classification). Example: Communication has only two values as ‘Good’ or ‘Bad’. It is also easy to extend the decision tree to create a target function with more than two possible output values. Example: CGPA can take one of the values from ‘High’, ‘Medium’, and ‘Low’. Irrespective of whether it is a binary value/ multiple values, it is discrete in nature. For example, Aptitude can take the value of either ‘High’ or ‘Low’. It is not possible to assign the value of both ‘High’ and ‘Low’ to the attribute Aptitude to draw a decision tree.

There should be no infinite loops on taking a decision. As we move from the root node to the next level node, it should move step-by-step towards the decision node. Otherwise, the algorithm may not give the final result for a given data. If a set of code goes in a loop, it would repeat itself forever, unless the system crashes.

A decision tree can be used even for some instances with missing attributes and instances with errors in the classification of examples or in the attribute values describing those examples; such instances are handled well by decision trees, thereby making them a robust learning method.

Points to Ponder

Balancing overfitting and underfitting in decision tree is a very tricky topic, involving more of an art than science. The way to master this art is experience in working with more number of data sets with a lot of diversity.

7.5.3 Random forest model

Random forest is an ensemble classifier, i.e. a combining classifier that uses and combines many decision tree classifiers. Ensembling is usually done using the concept of bagging with different feature sets. The reason for using large number of trees in random forest is to train the trees enough such that contribution from each feature comes in a number of models. After the random forest is generated by combining the trees, majority vote is applied to combine the output of the different trees. A simplified random forest model is depicted in Figure 7.14. The result from the ensemble model is usually better than that from the individual decision tree models.

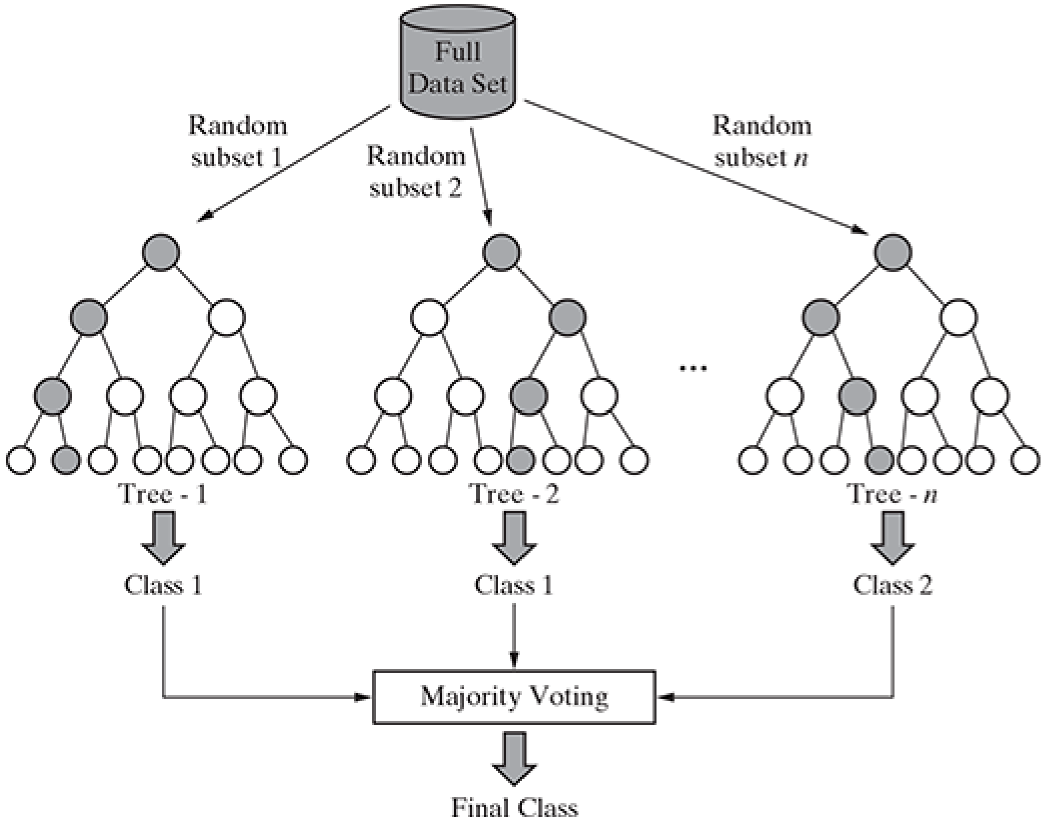


FIG. 7.14 Random forest model

7.5.3.1 How does random forest work?

The random forest algorithm works as follows:

1. If there are *N* variables or features in the input data set, select a subset of '*m*' (*m* < *N*) features at random out of the *N* features. Also, the observations or data instances should be picked randomly.
2. Use the best split principle on these '*m*' features to calculate the number of nodes '*d*'.
3. Keep splitting the nodes to child nodes till the tree is grown to the maximum possible extent.
4. Select a different subset of the training data 'with replacement' to train another decision tree following steps (1) to (3). Repeat this to build and train '*n*' decision trees.
5. Final class assignment is done on the basis of the majority votes from the '*n*' trees.

Points to Ponder

In the random forest classifier, if the number of trees is assumed to be excessively large, the model may get overfitted. In an extreme case of overfitting, the model may mimic the training data, and training error might be almost 0. However, when the model is run on an unseen sample, it may result in a very high validation error.

7.5.3.2 Out-of-bag (OOB) error in random forest

In random forests, we have seen, that each tree is constructed using a different bootstrap sample from the original data. The samples left out of the bootstrap and not used in the construction of the i-th tree can be used to measure the performance of the model. At the end of the run, predictions for each such sample evaluated each time are tallied, and the final prediction for that sample is obtained by taking a vote. The total error rate of predictions for such samples is termed as out-of-bag (OOB) error rate.

The error rate shown in the confusion matrix reflects the OOB error rate. Because of this reason, the error rate displayed is often surprisingly high.

7.5.3.3 Strengths of random forest

- It runs efficiently on large and expansive data sets.
- It has a robust method for estimating missing data and maintains precision when a large proportion of the data is absent.
- It has powerful techniques for balancing errors in a class population of unbalanced data sets.
- It gives estimates (or assessments) about which features are the most important ones in the overall classification.
- It generates an internal unbiased estimate (gauge) of the generalization error as the forest generation progresses.
- Generated forests can be saved for future use on other data.
- Lastly, the random forest algorithm can be used to solve both classification and regression problems.

7.5.3.4 Weaknesses of random forest

- This model, because it combines a number of decision tree models, is not as easy to understand as a decision tree model.
- It is computationally much more expensive than a simple model like decision tree.

7.5.3.5 Application of random forest

Random forest is a very powerful classifier which combines the versatility of many decision tree models into a single model. Because of the superior results, this ensemble model is gaining wide adoption and popularity amongst the machine learning practitioners to solve a wide range of classification problems.

7.5.4 Support vector machines

SVM is a model, which can do linear classification as well as regression. SVM is based on the concept of a surface, called a hyperplane, which draws a boundary between data instances plotted in the multi-dimensional feature space. The output prediction of an SVM is one of two conceivable classes which are already defined in the training data. In summary, the SVM algorithm builds an N-dimensional hyperplane model that assigns future instances into one of the two possible output classes.

7.5.4.1 Classification using hyperplanes

In SVM, a model is built to discriminate the data instances belonging to different classes. Let us assume for the sake of simplicity that the data instances are linearly separable. In this case, when mapped in a two-dimensional space, the data instances belonging to different classes fall in different sides of a straight line drawn in the two-dimensional space as depicted in Figure 7.15a. If the same concept is extended to a multi-dimensional feature space, the straight line dividing data instances belonging to different classes transforms to a hyperplane as depicted in Figure 7.15b.

Thus, an SVM model is a representation of the input instances as points in the feature space, which are mapped so that an apparent gap between them divides the instances of the separate classes. In other words, the goal of the SVM analysis is to find a plane, or rather a hyperplane, which separates the instances on the basis of their classes. New examples (i.e. new instances) are then mapped into that same space and predicted to belong to a class on the basis of which side of the gap the new instance will fall on. In summary, in the overall training process, the SVM algorithm analyses input data and identifies a surface in the multi-dimensional feature space called the hyperplane. There may be many possible hyperplanes, and one of the challenges with the SVM model is to find the optimal hyperplane.

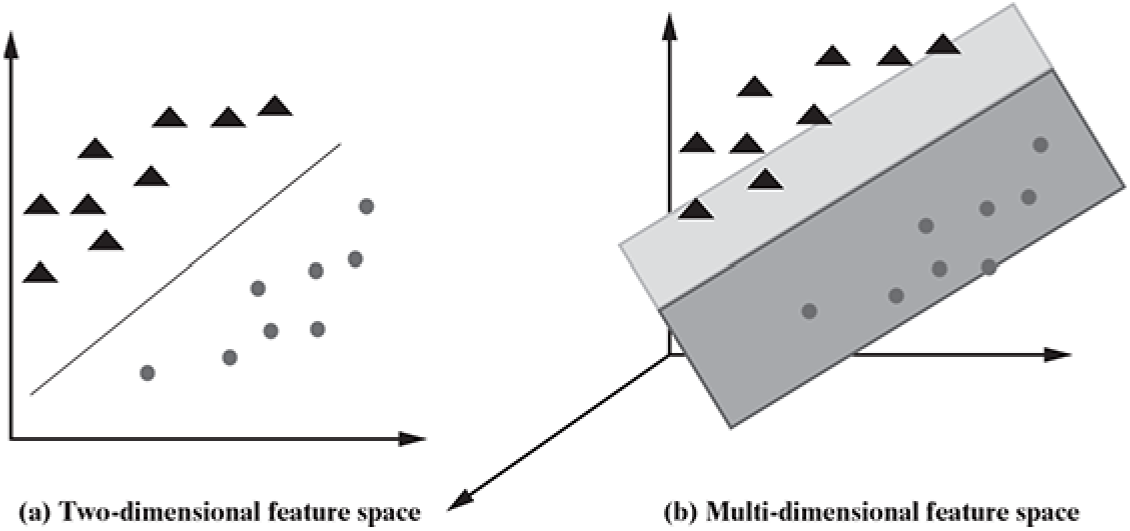

(a) Two-dimensional feature space
(b) Multi-dimensional feature space

FIG. 7.15 Linearly separable data instances

Training data sets which have a substantial grouping periphery will function well with SVM. Generalization error in terms of SVM is the measure of how accurately and precisely this SVM model can predict values for previously unseen data (new data). A hard margin in terms of SVM means that an SVM model is inflexible in classification and tries to work exceptionally fit in the training set, thereby causing overfitting.

Support Vectors: Support vectors are the data points (representing classes), the critical component in a data set, which are near the identified set of lines (hyperplane). If support vectors are removed, they will alter the position of the dividing hyperplane.

Hyperplane and Margin: For an N-dimensional feature space, hyperplane is a flat subspace of dimension (N−1) that separates and classifies a set of data. For example, if we consider a two-dimensional feature space (which is nothing but a data set having two features and a class variable), a hyperplane will be a one-dimensional subspace or a straight line. In the same way, for a three-dimensional feature space (data set having three features and a class variable), hyperplane is a two-dimensional subspace or a simple plane. However, quite understandably, it is difficult to visualize a feature space greater than three dimensions, much like for a subspace or hyperplane having more than three dimensions.

Mathematically, in a two-dimensional space, hyperplane can be defined by the equation:

$c_0 + c_1X_1 + c_2X_2 = 0$, which is nothing but an equation of a straight line.

Extending this concept to an N-dimensional space, hyperplane can be defined by the equation:

$c_0 + c_1X_1 + c_2X_2 + \ldots + c_NX_N = 0$ which, in short, can be represented as follows:

$$\vec{c}.\vec{X} + c_0 = 0$$

Spontaneously, the further (or more distance) from the hyperplane the data points lie, the more confident we can be about correct categorization. So, when a new testing data point/data set is added, the side of the hyperplane it lands on will decide the class that we assign to it. The distance between hyperplane and data points is known as **margin**.

7.5.4.2 Identifying the correct hyperplane in SVM

As we have already discussed, there may be multiple options for hyperplanes dividing the data instances belonging to the different classes. We need to identify which one will result in the best classification. Let us examine a few scenarios before arriving to that conclusion. For the sake of simplicity of visualization, the hyperplanes have been shown as straight lines in most of the diagrams.

Scenario 1

As depicted in Figure 7.16, in this scenario, we have three hyperplanes: A, B, and C. Now, we need to identify the correct hyperplane which better segregates the two classes represented by the triangles and circles. As we can see, hyperplane 'A' has performed this task quite well.

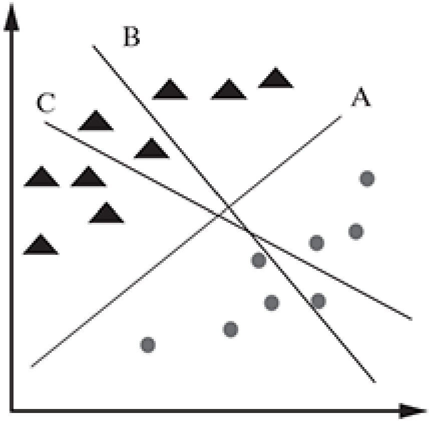


FIG. 7.16 Support vector machine: Scenario 1

Scenario 2

As depicted in Figure 7.17, we have three hyperplanes: A, B, and C. We have to identify the correct hyperplane which classifies the triangles and circles in the best possible way. Here, maximizing the distances between the nearest data points of both the classes and hyperplane will help us decide the correct hyperplane. This distance is called as **margin**.

In Figure 7.17b, you can see that the margin for hyperplane A is high as compared to those for both B and C. Hence, hyperplane A is the correct hyperplane. Another quick reason for selecting the hyperplane with higher margin (distance) is robustness. If we select a hyperplane having a lower margin (distance), then there is a high probability of misclassification.

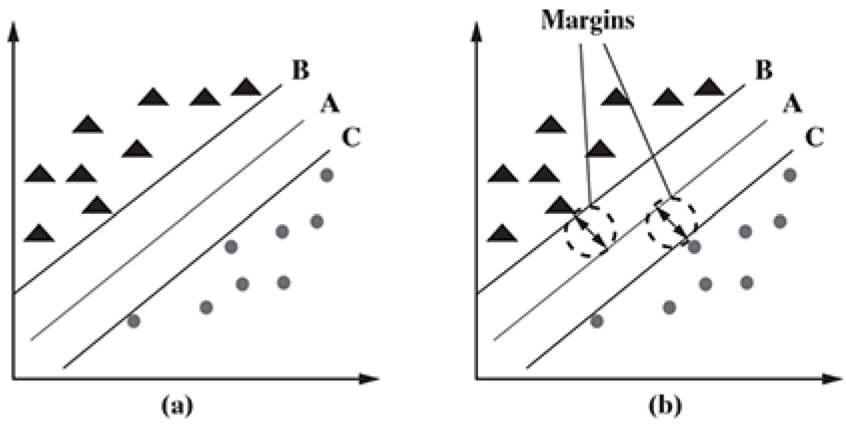


FIG. 7.17 Support vector machine: Scenario 2

Scenario 3

Use the rules as discussed in the previous section to identify the correct hyperplane in the scenario shown in Figure 7.18. Some of you might have selected hyperplane B as it has a higher margin (distance from the class) than A. But, here is the catch; SVM selects the hyperplane which classifies the classes accurately before maximizing the margin. Here, hyperplane B has a classification error, and A has classified all data instances correctly. Therefore, A is the correct hyperplane.

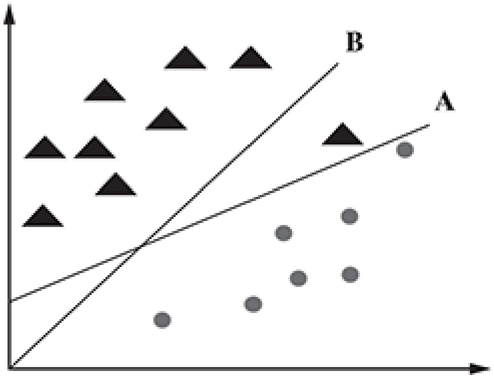


FIG. 7.18 Support vector machine: Scenario 3

Scenario 4

In this scenario, as shown in Figure 7.19a, it is not possible to distinctly segregate the two classes by using a straight line, as one data instance belonging to one of the classes (triangle) lies in the territory of the other class (circle) as an outlier.

One triangle at the other end is like an outlier for the triangle class. SVM has a feature to ignore outliers and find the hyperplane that has the maximum margin (hyperplane A, as shown in Fig. 7.19b). Hence, we can say that SVM is robust to outliers.

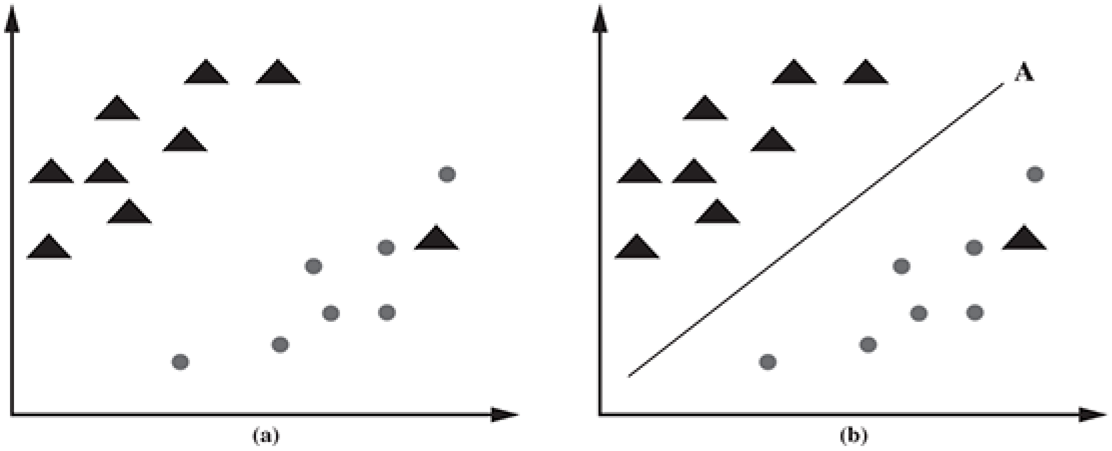


FIG. 7.19 Support vector machine: Scenario 4

So, by summarizing the observations from the different scenarios, we can say that

1. The hyperplane should segregate the data instances belonging to the two classes in the best possible way.
2. It should maximize the distances between the nearest data points of both the classes, i.e. maximize the margin.
3. If there is a need to prioritize between higher margin and lesser misclassification, the hyperplane should try to reduce misclassifications.

Our next focus is to find out a way to identify a hyperplane which maximizes the margin.

7.5.4.3 Maximum margin hyperplane

Finding the Maximum Margin Hyperplane (MMH) is nothing but identifying the hyperplane which has the largest separation with the data instances of the two classes. Though any set of three hyperplanes can do the correct classification, why do we need to search for the set of hyperplanes causing the largest separation? The answer is that doing so helps us in achieving more generalization and hence less number of issues in the classification of unknown data.

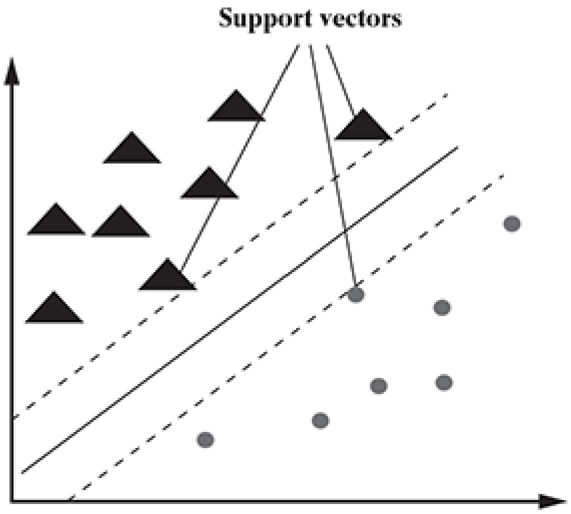


FIG. 7.20 Support vectors

Support vectors, as can be observed in Figure 7.20, are data instances from the two classes which are closest to the MMH. Quite understandably, there should be at least one support vector from each class. The identification of support vectors requires intense mathematical formulation, which is out of scope of this book. However, it is fairly intuitive to understand that modelling a problem using SVM is nothing but identifying the support vectors and MMH corresponding to the problem space.

Identifying the MMH for linearly separable data

Finding out the MMH is relatively straightforward for the data that is linearly separable. In this case, an outer boundary needs to be drawn for the data instances belonging to the different classes. These outer boundaries are known as convex hull, as depicted in Figure 7.21. The MMH can be drawn as the perpendicular bisector of the shortest line (i.e. the connecting line having the shortest length) between the convex hulls.

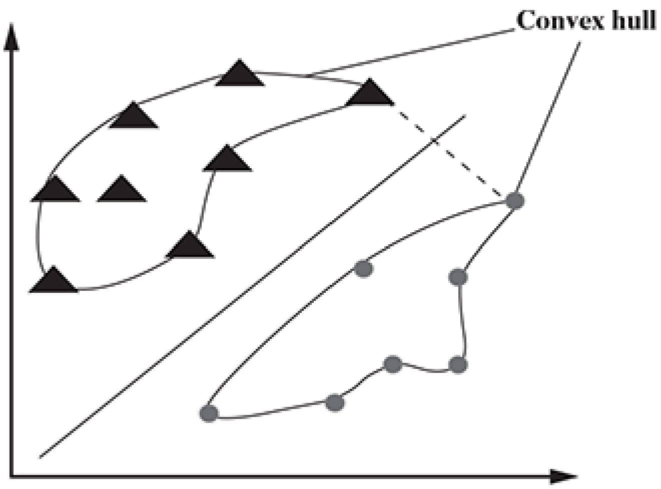


FIG. 7.21 Drawing the MMH for linearly separable data

We have already seen earlier that a hyperplane in the N-dimensional feature space can be represented by the equation: $\vec{c}.\vec{X} + c_0 = 0$

Using this equation, the objective is to find a set of values for the vector $\vec{c}$ such that two hyperplanes, represented by the equations below, can be specified.

$$\vec{c}.\vec{X} + c_0 \geq +1$$
$$\vec{c}.\vec{X} + c_0 \leq -1$$

This is to ensure that all the data instances that belong to one class falls above one hyperplane and all the data instances belonging to the other class falls below another hyperplane. According to vector geometry, the distance of these planes should be $\frac{2}{\vec{c}}$. It is quite obvious that in order to maximize the distance between hyperplanes, the value of $\vec{c}$ should be minimized. So, in summary, the task of SVM is to solve the optimization problem:

Identifying the MMH for non-linearly separable data

Now that we have a clear understanding of how to identify the MMH for a linearly separable data set, let us do some study about how non-linearly separable data needs to be handled by SVM. For this, we have to use a slack variable ξ, which provides some soft margin for data instances in one class that fall on the wrong side of the hyperplane. As depicted in Figure 7.22, a data instance belonging to the circle class falls on the side of the hyperplane designated for the data instances belonging to the triangle class. The same issue also happens for a data instance belonging to the triangle class.

FIG. 7.22 Drawing the MMH for non-linearly separable data

A cost value '*C*' is imposed on all such data instances that fall on the wrong side of the hyperplane. The task of SVM is now to minimize the total cost due to such data instances in order to solve the revised optimization problem:

7.5.4.4 Kernel trick

As we have seen in the last section, one way to deal with non-linearly separable data is by using a slack variable and an optimization function to minimize the cost value. However, this is not the only way to use SVM to solve machine learning problems involving non-linearly separable data sets. SVM has a technique called the **kernel trick** to deal with non-linearly separable data. As shown in Figure 7.23, these are functions which can transform lower dimensional input space to a higher dimensional space. In the process, it converts linearly non-separable data to a linearly separable data. These functions are called **kernels**.

FIG. 7.23 Kernel trick in SVM

Some of the common kernel functions for transforming from a lower dimension ‘*i*’ to a higher dimension ‘*j*’ used by different SVM implementations are as follows:

- Linear kernel: It is in the form
- Polynomial kernel: It is in the form
- Sigmoid kernel: It is in the form
- Gaussian RBF kernel: It is in the form

When data instances of the classes are closer to each other, this method can be used. The effectiveness of SVM depends both on the

- selection of the kernel function
- adoption of values for the kernel parameters

7.5.4.5 Strengths of SVM

- SVM can be used for both classification and regression.

- It is robust, i.e. not much impacted by data with noise or outliers.
- The prediction results using this model are very promising.

7.5.4.6 Weaknesses of SVM

- SVM is applicable only for binary classification, i.e. when there are only two classes in the problem domain.
- The SVM model is very complex – almost like a black box when it deals with a high-dimensional data set. Hence, it is very difficult and close to impossible to understand the model in such cases.
- It is slow for a large dataset, i.e. a data set with either a large number of features or a large number of instances.
- It is quite memory-intensive.

7.5.4.7 Application of SVM

SVM is most effective when it is used for binary classification, i.e. for solving a machine learning problem with two classes. One common problem on which SVM can be applied is in the field of bioinformatics – more specifically, in detecting cancer and other genetic disorders. It can also be used in detecting the image of a face by binary classification of images into face and non-face components. More such applications can be described.

7.6 SUMMARY

1. The objective of classification is to predict the class of unknown objects on the basis of prior class-related information of similar objects. This learning could be used when how to classify a given data is known or, in other words, class values of data instances are available.
2. A critical classification problem in the context of the banking domain is identifying potentially fraudulent transactions. Because there are millions of transactions which have to be scrutinized to identify whether a particular transaction might be a fraud transaction, it is not possible for any human being to carry out this task.

3. Common classification algorithms are *k*NN, decision tree, random forest, SVM, and Naïve Bayes.
4. The *k*NN algorithm is among the best and the simplest of machine learning algorithms. A new instance is classified by a majority vote of its neighbours, with the instance being allocated the class that is predominant among its *k*NN.
5. Decision tree learning is the most broadly utilized classifier. It is characterized by fast execution time and ease in the interpretation of the rules.
6. The decision tree algorithm needs to find out the attribute splitting by which results in highest information gain. For a sample of training examples S, entropy measures the impurity of S.

   where *c* is the number of different class labels and *p* refers to the proportion of values falling into the *i*-th class label.
7. Information gain is created on the basis of the decrease in entropy (*S*) after a dataset is split based on a particular attribute (*A*). Information gain for a particular feature A is calculated by the difference in entropy before a split ($S_{\mathbf{bs}}$) with the entropy after the split ($S_{\mathbf{as}}$).
   Information Gain (S, A) = Entropy ($\mathbf{S_{bs}}$) - Entropy ($\mathbf{S_{as}}$)
8. Random forest is an ensemble classifier (combining classifier) which uses and combines many decision tree models. The result from an ensemble model is usually better than that from one of the individual models.
9. An SVM is a binary linear classifier. The output prediction of an SVM is one of the two conceivable classes which are already defined in the training data. SVM is also an example of a linear classifier and a maximum margin classifier.
10. SVM has a new technique called the kernel trick. These are functions, which take a lower dimensional input space and transform it to a higher dimensional space and in the process converts a non-linearly separable problem to a linearly separable problem. These functions

are called kernels. When all the classes are closer to each other, this method can be used.

MULTIPLE CHOICE QUESTIONS

1. Predicting whether a tumour is malignant or benign is an example of?
   1. Unsupervised Learning
   2. Supervised Regression Problem
   3. Supervised Classification Problem
   4. Categorical Attribute
2. Price prediction in the domain of real estate is an example of?
   1. Unsupervised Learning
   2. Supervised Regression Problem
   3. Supervised Classification Problem
   4. Categorical Attribute
3. Let us consider two examples, say ‘predicting whether a tumour is malignant or benign’ and ‘price prediction in the domain of real estate’. These two problems are same in nature.
   1. TRUE
   2. FALSE
4. Supervised machine learning is as good as the data used to train it.
   1. TRUE
   2. FALSE
5. Which is a type of machine learning where a target feature, which is of categorical type, is predicted for the test data on the basis of the information imparted by the training data?
   1. Unsupervised Learning
   2. Supervised Regression
   3. Supervised Classification
   4. Categorical Attribute
6. Classification is a type of supervised learning where a target feature, which is of categorical type, is predicted for the test data on the basis of the information imparted by the training data. The target categorical feature is known as?
   1. Object

   2. Variable
   3. Method
   4. Class
7. This is the first step in the supervised learning model.
   1. Problem Identification
   2. Identification of Required Data
   3. Data Pre-processing
   4. Definition of Training Data Set
8. This is the cleaning/transforming the data set in the supervised learning model.
   1. Problem Identification
   2. Identification of Required Data
   3. Data Pre-processing
   4. Definition of Training Data Set
9. This refers to the transformations applied to the identified data before feeding the same into the algorithm.
   1. Problem Identification
   2. Identification of Required Data
   3. Data Pre-processing
   4. Definition of Training Data Set
10. This step of supervised learning determines 'the type of training data set'.
    1. Problem Identification
    2. Identification of Required Data
    3. Data Pre-processing
    4. Definition of Training Data Set
11. Entire design of the programme is done over here in supervised learning.
    1. Problem Identification
    2. Training
    3. Data Pre-processing
    4. Definition of Training Data Set
12. Training data run on the algorithm is called as?
    1. Program
    2. Training

   3. Training Information
   4. Learned Function
13. SVM is an example of?
    1. Linear Classifier and Maximum Margin Classifier
    2. Non-linear Classifier and Maximum Margin Classifier
    3. Linear Classifier and Minimum Margin Classifier
    4. Non-linear Classifier and Minimum Margin Classifier
14. ---------- in terms of SVM means that an SVM is inflexible in classification
    1. Hard Margin
    2. Soft Margin
    3. Linear Margin
    4. Non-linear Classifier
15. ---------- are the data points (representing classes), the important component in a data set, which are near the identified set of lines (hyperplane).
    1. Hard Margin
    2. Soft Margin
    3. Linear Margin
    4. Support Vectors
16. ---------- is a line that linearly separates and classifies a set of data.
    1. Hyperplane
    2. Soft Margin
    3. Linear Margin
    4. Support Vectors
17. The distance between hyperplane and data points is called as:
    1. Hyper Plan
    2. Margins
    3. Error
    4. Support Vectors
18. Which of the following is true about SVM?
    1. It is useful only in high-dimensional spaces
    2. It always gives an approximate value
    3. It is accurate
    4. Understanding SVM is difficult

19. Which of the following is true about SVM?
    1. It is useful only in high-dimensional spaces
    2. It requires less memory
    3. SVM does not perform well when we have a large data set
    4. SVM performs well when we have a large data set
20. What is the meaning of hard margin in SVM?
    1. SVM allows very low error in classification
    2. SVM allows high amount of error in classification
    3. Underfitting
    4. SVM is highly flexible
21. What sizes of training data sets are not best suited for SVM?
    1. Large data sets
    2. Very small training data sets
    3. Medium size training data sets
    4. Training data set size does not matter
22. Support Vectors are near the hyperplane.
    1. True
    2. False
23. In SVM, these functions take a lower dimensional input space and transform it to a higher dimensional space.
    1. Kernels
    2. Vector
    3. Support Vector
    4. Hyperplane
24. Which of the following options is true about the *k*NN algorithm?
    1. It can be used only for classification
    2. It can be used only for regression
    3. It can be used for both classification and regression
    4. It is not possible to use for both classification and regression
25. Which of the following will be Euclidean distance between the two data points A(4,3) and B(2,3)?
    1. 1
    2. 2
    3. 4
    4. 8

26. Which of the following will be Manhattan distance between the two data points A(8,3) and B(4,3)?
    1. 1
    2. 2
    3. 4
    4. 8
27. When you find many noises in data, which of the following options would you consider in *k*NN?
    1. Increase the value of *k*
    2. Decrease the value of *k*
    3. Noise does not depend on *k*
    4. *K* = 0
28. What would be the relationship between the training time taken by 1-NN, 2-NN, and 3-NN?
    1. 1-NN > 2-NN > 3-NN
    2. 1-NN < 2-NN < 3-NN
    3. 1-NN ~ 2-NN ~ 3-NN
    4. None of these
29. Which of the following algorithms is an example of the ensemble learning algorithm?
    1. Random Forest
    2. Decision Tree
    3. *k*NN
    4. SVM
30. Which of the following is not an inductive bias in a decision tree?
    1. It prefers longer tree over shorter tree
    2. Trees that place nodes near the root with high information gain are preferred
    3. Overfitting is a natural phenomenon in a decision tree
    4. Prefer the shortest hypothesis that fits the data

SHORT ANSWER-TYPE QUESTIONS (5 MARKS EACH)

1. What is supervised learning? Why it is called so?
2. Give an example of supervised learning in a hospital industry.

3. Give any three examples of supervised learning.
4. What is classification and regression in a supervised learning?
5. Give some examples of common classification algorithms.
6. Explain, in brief, the SVM model.
7. What is cost of misclassification in SVM?
8. Define Support Vectors in the SVM model.
9. Define kernel in the SVM model.
10. What are the factors determining the effectiveness of SVM?
11. What are the advantages of the SVM model?
12. What are the disadvantages of the SVM model?
13. Write notes on
    1. validation error in the *k*NN algorithm
    2. choosing *k* value in the *k*NN algorithm
    3. inductive bias in a decision tree
14. What are the advantages of the *k*NN algorithm?
15. What are the disadvantages of the *k*NN algorithm?
16. Explain, in brief, the decision tree algorithm.
17. What is node and leaf in decision tree?
18. What is entropy of a decision tree?
19. Define information gain in a decision tree.
20. Write any three strengths of the decision tree method.
21. Write any three weaknesses of the decision tree method.
22. Explain, in brief, the random forest model?

LONG ANSWER-TYPE QUESTIONS (10 MARKS EACH)

1. Distinguish between supervised learning, semi-supervised learning, and unsupervised learning.
2. Explain any five examples of classification problems in detail.
3. Explain classification steps in detail.
4. Discuss the SVM model in detail with different scenarios.
5. What are the advantages and disadvantages associated with SVM?
6. Discuss the *k*NN model in detail.
7. Discuss the error rate and validation error in the *k*NN algorithm.

8. Discuss how to calculate the distance between the test data and the training data for *k*NN.
9. Write the algorithm for *k*NN.
10. What is decision tree? What are the different types of nodes? Explain in detail
11. Explain various options of searching a decision tree.
12. Discuss the decision tree algorithm in detail.
13. What is inductive bias in a decision tree? How to avoid overfitting?
14. What are the strengths and weaknesses of the decision tree method?
15. Discuss appropriate problems for decision tree learning in detail.
16. Discuss the random forest model in detail. What are the features of random forest?
17. Discuss OOB error and variable importance in random forest.