CHAPTER 3

CLASSIFICATION

To be or not to be: That is the question.
—WILLIAM SHAKESPEARE

INTRODUCTION

Humans have always aspired to predict the future based on the past and the nature of the unknown based on the qualities of the known. It can be said that 'knowledge is power' only to the extent that it is useful in making such predictions as accurately as possible. The phenomenal success of fields such as astrology and palmistry to which people flock, despite the fact that their scientific value is under debate, attests to the importance of making accurate predictions.

Machine learning is a branch of artificial intelligence whose goal is to write programs that automatically learn from experience. This field started in the 1950s, but went out of fashion in the 1960s when limitations in the explored methods became evident. It re-emerged in the 1970s and today it is a reasonably mature field.

Classification techniques were developed as an important component of machine learning algorithms in order to extract rules and patterns from data that could be used for prediction. Classification techniques are used to classify data records into one among a set of predefined classes. They work by constructing a model of a *training dataset* consisting of example records with known class labels.

Many classification techniques developed in machine learning have been used in data mining applications. Unlike machine learning, where the extracted patterns are meant to be used by programs to make better predictions, in data mining the emphasis is on the human understandability of the extracted

patterns. Many data mining applications also require that the techniques be scalable to a huge amount of data. Classification techniques have been used in numerous applications ranging from *spam* detection in electronic mails (*e-mails*), credit-card fraud detection, speech recognition, and computer vision.

In this chapter, the various classification algorithms (or classifiers), their applications, and evaluation will be studied. Other issues that will be touched upon include the selection of features for good classification, the methods used to handle missing data, and the ways in which classifiers can be combined to yield better classification.

3.1 Basic Problem Definition

Consider the following example that can be used to grasp the abstract definitions provided later in this section.

Example 3.1 A customer who applies for a loan from a bank may fall into one of three classes—good (repays the loan on time), ugly (repays the loan late) and bad (does not repay the loan). The bank, of course, wants all its customers to be good.

In order to maximize the chance that all its customers are good, the bank normally screens each customer for their potential to repay the loan. Each customer is expected to fill detailed forms about themselves, their family, existing property, and occupation. Each customer is thus defined by a number of properties, which could be numeric (such as age and annual income) or categorical (such as gender and occupation).

Over a period of time, the bank obtains sufficient data about the customers of each class. This data can be analyzed by a classification program to determine the properties of the customers of each class. Subsequently, when new customers apply for loans, their properties can be verified to check the class to which they are likely to belong. If they are likely to be ugly or bad, then more stringent measures can be applied to ensure that they have the potential to repay the loan.

The problem of classification is formally defined below.

Instance An instance is an object that is represented by a vector of numeric or categorical attributes.

The input data model for classification consists of instances (also known as samples, cases, objects, or records) each of which belongs to a specific 'class'. In Example 3.1, the objects are customers and the classes are 'good', 'ugly',

and 'bad'. Each object is defined by a number of attributes (also known as features or variables), which could be numeric (such as age and annual income) or categorical (such as gender and occupation).

Labelled instance An instance is said to be labelled if the class that it belongs to is known.

The classification system is provided with the class labels of some objects, such objects are said to be labelled and are also referred to as examples. Classification is also known as supervised learning because it uses available examples to supervise the learning process.

Classification Given a set of n labelled instances $x_1,\ldots,x_n$ represented by vectors $v_1,\ldots,v_n$, and corresponding classes $C_1,\ldots,C_n$, the classification problem is to determine a function f such that $f(x_i) = C_i$.

In the above definition, f is usually not a closed-form function, but an algorithm that takes as input, an instance x_i, and outputs a class. In practice, different types of algorithms have been devised in the research literature and each type takes the set of labelled examples as input and computes a 'classification model'. This model is then used to predict the classes of unlabelled objects. Thus, classification approaches a two-phase approach as follows:

Phase 1: Model Construction During this phase, the labelled examples are analyzed and a model is built and stored on disk.

Phase 2: Classification During this phase, the model stored on the disk is loaded into the main memory and used to classify new unlabelled instances.

For most applications, however, it is not possible to determine a function that exactly solves the classification problems in the manner described in the above definition. Instead, most approaches settle with an approximate solution that produces some error during classification, i.e. for some of the instances x_i, $f(x_i)$ may not be equal to C_i. It is desirable to have classifiers that are as accurate as possible. The different ways of measuring accuracy will be described in Section 3.3.

An implicit assumption in classification is that the class value, to a large extent, depends only on features that are actually used in the input data model. If the class value depends heavily on the features that are not part of the data model, then the classification model built will not be accurate even when the best classification techniques are used. Hence, the task of selecting good features for a particular application is important.

3.2 Applications

Classification has a wide spectrum of applications including detecting *spam* in *e-mails*, medical diagnosis, detecting credit-card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, game playing, and robot locomotion. In fact, the entire scientific knowledge can be viewed as a means to predict the future in terms of the past. This is precisely the goal of classification.

3.2.1 Text Classification

If we are to identify a single application that has made computers and the Internet popular, it would probably be *e-mail*. Applications, such as *e-mail*, which are responsible for making technology popular, are sometimes known as 'killer applications'. Unfortunately, the existence of numerous *spam* (or unwanted) *e-mails* makes it quite painful for the users to use this technology.

The task of automatically separating out *spam e-mails* is a form of text classification—the objects to be classified consist of text documents or their parts (such as words, sentences, and paragraphs). Text classification is also useful on the web for classifying webpages into a Yahoo!-type directory hierarchy. Several natural language processing (NLP) applications involve classification. These include tagging (classifying words into their part of speech) and word-sense disambiguation (when words have multiple meanings, they have to be classified into one of those meanings).

3.2.2 Fraud Detection

Fraudsters have been exploiting businesses for their own financial gain ever since the beginning of commerce. The problem is worse today due to technological advances, which make fraud relatively easy to commit and difficult to detect. From cheque fraud and credit card fraud to computer intrusion and identity theft, fraud is a crime to which every business is vulnerable. This is particularly true of banks that trade money digitally rather than using paper.

Given the properties of a transaction (such as items purchased, amount, location, customer profile, etc.), classification techniques can be used to determine if it is likely to be a fraud or not. Any classification technique cannot be fully accurate in all situations. Therefore, it is imperative to treat the results of a classification as indicative only and to cross-check the same using other stringent methods.

3.2.3 Pattern Recognition

Pattern recognition applications such as optical character recognition (OCR), handwriting recognition, and speech recognition rely on classification techniques. The different characters of handwriting (or phonemes of speech) can be treated as classes and modelled using available examples. These models can then be used to classify other characters (or phonemes) to aid in recognising them. The difficulty in the pattern recognition applications usually lies in selecting the attributes of the objects (characters, phonemes, etc.) that are relevant for classification, a problem known as *feature selection.*

3.3 Evaluation of Classifiers

As described in Section 3.1, it is rarely possible to build fully accurate classifiers. It is desirable to build classifiers that make as few mistakes as possible in deciding the classes of instances. In this section, popular approaches that are used to measure the accuracy of a classifier are described.

3.3.1 Holdout Method

In general, all the available labelled examples are not used for building the classification model. Instead, the labelled examples are randomly divided into two parts, the *training dataset* and the *test dataset.* Only the training dataset is used to construct the classification model.

Training dataset The training dataset is that portion of the available labelled examples that is used to build the classification model.

The reason for keeping aside some of the labelled examples for a test dataset is to test the performance of the constructed classification model. It would be a gross error if we do not keep aside a test dataset and instead use the same examples for building the model and testing it. The accuracy values reported by such a method would make even some very naïve classifiers seem highly accurate.

Test dataset The test dataset is that portion of the available labelled examples that is used to test the performance of the classification model.

In order to use the test dataset to evaluate the classifier, each record in the test dataset is entered as input to the classifier and the class predicted by the classifier, for that record, is noted. During this classification, the available class label of the record being classified is ignored. Once the predicted class

is obtained, it is compared to the actual class of the record. If they are the same, then the classifier has been successful, else it has failed.

In order to obtain a good measure of the performance of a classifier, it is necessary that the test dataset has approximately the same class distribution as the training dataset, as illustrated in the following example.

Example 3.2 Consider that 5% of the records in a training dataset belong to a class *C* and 60% of the records in a test dataset belong to class *C*. The test dataset and the training dataset differ significantly in their class distribution.

From the training dataset, the classifier may incorrectly assume that only a minority of all the records actually belong to class *C*. It may use this incorrect assumption and decide not to classify a majority of the test data records into class *C*. Doing this would lead to many errors in the classification.

In order to rectify this situation, it is necessary that both the training data and the test data have similar distribution of classes.

Stratified test dataset A stratified test dataset is one that has the same distribution of classes as the training dataset.

If we focus on any particular class *C*, there are four possibilities obtained by comparing the actual class labels with those output by the classifier. These possibilities are shown in the matrix, in Fig. 3.1, which shows that the fraction of records those were not actually in class *C*, but classified as *C*, is *b*. Such a matrix is known as a *confusion matrix* because it shows the percentage of records of a class that were confused as belonging to other classes.

		Actual Class	
		C	Not *C*
Predicted Class	*C*	*a*	*b*
	Not *C*	*c*	*d*

Fig. 3.1 Normalized confusion matrix (2 classes)

The most popular metric to evaluate the merit of a classifier is the accuracy with which it classifies new records.

Accuracy The accuracy of a classifier is the probability of it correctly classifying records in the test dataset.

In practice, accuracy is measured as the percentage of records in the test dataset that are correctly classified by a classifier. If there are only two classes (C and not C) then accuracy is computed as

$$accuracy = a + d$$

where a and d are defined in the matrix in Fig. 3.1.

Other popular metrics include *precision*, *recall*, and *F-score*, which are defined below.

Precision The precision of a classifier is the probability of records actually being in a class C if they are classified as being in class C.

For the two class situation of Fig. 3.1, precision is computed as $a/(a+b)$.

Recall The recall of a classifier is the probability that a record is classified as being in a class C if it actually belongs to class C.

Recall may seem similar to *precision*, but it is quite different. For the two class situation of Fig. 3.1, it is computed as $a/(a+c)$.

***F*-score** F-score is the harmonic mean of precision and recall.

It is computed as $2 \times \text{recall} \times \text{precision}/(\text{recall} + \text{precision})$ and is intended to exhibit the merits of both precision and recall.

3.3.2 Cross-validation

The holdout technique described above is the basic technique used most commonly for evaluating the performance of a classifier. Sometimes, to obtain a better measure of the performance, the holdout technique is repeated several times over different training and test datasets and the average accuracy is reported.

One practical problem that is usually faced in the basic holdout method is that the available labelled examples are few in number. Some of these examples are used for the training dataset and some for the test dataset. The test data records are not used for training and thus, are wasted, in a sense. To make the best use of all available examples for training, the following scheme known as *cross-validation* is employed.

Cross-validation technique First the labelled examples are randomly divided into k partitions $S_1, S_2, \ldots, S_k$, where k is user-defined (usually about 10). Then each S_i, in turn, is treated as a test dataset, and the remaining partitions are clubbed together as the training dataset, and the classification performance is measured via the holdout method. After repeating this for each S_i, i.e. k times, the accuracy of the classifier is output as the percentage of the correctly classified records with respect to the total number of the examples. This strategy ensures that every example is used for training as well as testing.

3.3.3 Confusion Matrix

Often, popular measures of accuracy are misleading. For example, if only 1% people in the entire population have cancer, then a classifier that classifies all people as having no cancer would be 99% accurate!

Therefore, it is often best to output a confusion matrix to the user showing the classification statistics for each class, as shown in Fig. 3.2. Here, for example, there are 8% records of class C_3 that were classified as C_1. If the classifier is perfect, then the diagonal entries would all be 100% and the remaining entries would be zero. Notice that the entries in each column add up to 100%–this is because the records of that class is classified into some class or the other.

		Actual Class		
		C_1	C_2	C_3
Predicted Class	C_1	**90%**	**2%**	8%
	C_2	5%	**91%**	3%
	C_3	5%	7%	**89%**

Fig. 3.2 Normalized confusion matrix (3 classes)

3.3.4 Other Performance Metrics

While predictive accuracy is the most important metric, classifier performance can be measured in other ways as follows:

Classification time This is the amount of time taken to classify a new record. In several applications, classification needs to be done online and in real time. Hence, it is imperative that the classification time is as small as possible.

Training time This is the amount of time taken to build a classification model from the training data. In most applications, this metric is not very critical, however, the training time should be within practical limits, i.e. at most a few hours or days.

Main memory usage This is the amount of main memory required during the classification. Again this is not critical, but the usage should be within practical limits, so as to work on current machines.

Scalability It is important that the classification algorithm be scalable to handle the training datasets that contain a huge number of instances and/or attributes.

3.4 Other Issues

3.4.1 Data Preprocessing

Most pattern discovery technologies require that the input data be passed through a preprocessing stage in order to prepare it for pattern discovery. Preprocessing includes *data cleaning* and *data transformation* as described below. The terms data preprocessing, cleaning, and transformation conjure up mental images of the manual labour involved in these tasks. Therefore, most students and even researchers shy away from this area. This is unfortunate since it is a very important field.

3.4.5 Data Cleaning

Many automated techniques could be built to ease the task of data preprocessing, and the discovery of such techniques is an interesting part of research.

Data cleaning

Data cleaning involves handling noise, inconsistency, and missing values in the input data. Techniques for pattern discovery are normally designed to detect general trends that are persistent throughout the data. Such strong trends are unlikely to be affected by a small amount of noise, inconsistency, or missing values in the data. However, in situations where these undesirable characteristics are present in more than a negligible amount, they must be reduced by using special techniques.

Noise is the presence of inaccuracies in data. This may be caused due to manual or experimental error in measuring some attributes. Several 'smoothing' approaches are available to reduce noise such as the following:

Binning In this method, the values of an attribute are first sorted and then partitioned into ranges or 'bins'. All values within a bin are then replaced by a single value such as the mean or median of that bin.

Regression Smoothing can be achieved by fitting the data into a function such as a linear or non-linear regression. Data values can then be replaced by their smoothed values.

Outliers A special class of data mining algorithms are available to detect outliers. Outliers are the isolated data objects that do not follow the general trends in the data. They are most likely to be noise. These algorithms will be discussed in Chapter 4 along with clustering techniques.

Inconsistencies in data arise due to errors in data entry. An example of an inconsistency is the pin code of a town being wrongly typed while entering the addresses of customers. This kind of error is not noise, it is an inconsistency. The presence of inconsistencies can be prevented to a large extent by following a good database design and explicitly specifying integrity constraints. Nevertheless, even in the most rigorous scenarios, some inconsistency eventually seeps in and may need manual correction.

Both noise and inconsistency are problems in most pattern discovery techniques and can be solved by using generic techniques such as good database design or statistical techniques. For the problem of missing values in data, however, special techniques have been evolved in classification, and these are described in Section 3.4.4.

Data transformation

Data transformation involves handling data format issues, discretization, normalization, generalization, and feature construction.

Data format conversion Data format issues typically arise when data in different formats, from multiple sources, need to be combined into one data repository. Also, different implementations of algorithms may require different input formats and the available data needs to be converted into that format.

Discretization Many algorithms require their input data to consist of only categorical attributes. In such cases, discretization is indispensable. Discretization is the process of converting a numeric attribute into a categorical one. Basic discretization techniques involve sorting the values of an attribute and then dividing these sorted values into ranges. An example of this is equi-

depth partitioning (Section 2.4.1, Chapter 2). More complex methods have been evolved in the research literature. Some of these techniques utilize the clustering algorithms to divide the attribute values into groups. Others use measures such as information gain (described in Section 3.1) to determine the best points at which an attribute can be split to form ranges.

Normalization Numeric attributes are usually scaled to lie within a small range, such as between −1.0 to +1.0. An example of this is the calculation of z-score of an attribute (Section 4.1.2, Chapter 4). Normalization is required to avoid giving undue significance to the attributes having large ranges.

Generalization Some categorical attributes may contain too many values and thereby becoming highly detailed. For example, in a census database, there may be a 'location' attribute whose value could be one amongst hundreds of towns and villages. Generalization involves transforming such attributes into 'higher-level' concepts. In this case, the location attribute could be modified to contain the name of the district in which a town or village is present, instead of the actual town or village.

Feature construction In this process, new features are constructed from the original features and added to the classification problem to help in the mining process. For example, in a spatial dataset that contains, among other things, the widths and lengths of several rectangles, a new attribute could be constructed to store the area of the rectangles as well.

3.4.2 Feature Selection

In many applications, the objects to be classified could be described by a large number of attributes. Some of these attributes may be directly relevant to the classification problem, some attributes may be relevant after some kind of transformation, and some attributes may not be relevant at all.

For example, in classifying *e-mails* as *spam* or *non-spam*, binary attributes indicating the presence or absence of certain words would be relevant. The presence of certain combinations of words might also be very relevant, while there may be some words whose presence would not be helpful in classifying the *e-mail* as *spam* or *non-spam*.

It is desirable that the classifiers automatically decide which attributes are relevant for the problem. However, most classifiers cannot do this and hence, depend on a pre-processing stage where the relevant features are selected. Often, this stage requires input from the users or experts who are well acquainted with the domain under study.

3.4.3 Over-fitting

The general trends in the training dataset are likely to persist in the future data records. It would be unwise to make similar assumptions about details that are too specific to the training dataset and are supported by only a few records. These details may or may not persist in future.

Therefore, classifiers should be built based on the general trends inherent in the training data and not on specific details. When a classifier begins to depend on specific details, it is said to over-fit the training data. Most classifiers have input parameters that decide the level of detail required to depend on the training data. Some classifiers have post-processing stages where the effects of over-fitting are rectified.

3.4.4 Missing Data

Both training data and test data may contain records with missing or incorrect attribute values. Classifiers that can correctly learn and classify even such data are said to be robust. Several approaches exist to deal with missing data and some are listed below. The example shown in Fig. 3.3 is used to demonstrate these techniques. Each record represents the weather condition and the class attribute represents whether people generally play sports in that weather condition, or not. Notice that, in this table, the first row where 'Outlook' is 'overcast' has its 'Windy' attribute as a missing value.

Outlook	Temperature (Farenheit)	Humidity (%)	Windy	Class
Sunny	75	70	true	play
Sunny	80	90	true	don't play
Sunny	85	85	false	don't play
Sunny	72	95	false	don't play
Sunny	69	70	false	play
Overcast	72	90		play
Overcast	83	78	false	play
Overcast	64	65	true	play
Overcast	81	75	false	play
Rain	71	80	true	don't play
Rain	65	70	true	don't play
Rain	75	80	false	play
Rain	68	80	false	play
Rain	70	96	false	play

Fig. 3.3 Dataset with a missing value

Approaches

Ignore missing values This method just ignores the records that contain missing values. This is the simplest method of all and yields good results if just a few values are missing. After all, data mining techniques are designed to detect general trends that are persistent throughout the data. These strong trends are unlikely to be affected by a few missing values or errors.

Most common value In this method, the attribute value that occurs most often is selected for all the unknown values of that attribute. In the example table above, the missing value is taken as false because this is the most common value of the Windy attribute.

When the number of missing values is more than 'just a few', it becomes necessary to employ special techniques to handle them. By selecting the most common value, this method is actually making a good guess for each missing value, as to what might have been the actual value in its place. If most of the guesses turn out to be right, the model built from the resulting data is more accurate. By selecting the most frequent value for each attribute, it is likely that most of the guesses are right.

Concept most common value In this method, the attribute value that occurs most often within the records of a class is selected for the unknown values of that attribute in the records of that class. In the given example table, this method would again use the value of false for the missing value. This is because the record with the missing value belongs to the 'play' class. In this class, there are more records whose 'Windy' attribute is 'false'.

Essentially, this method makes a more sophisticated guess than the previous approach by making use of the class information of the records having missing values. This method is therefore likely to make fewer errors in rectifying the missing values. However, it requires more time to compute this guess.

All possible values In this method, a record with a missing attribute value is replaced by several records, in which the missing value is replaced by all possible values of that attribute. In the current example, a new record will be inserted into the table that is identical to the record with the missing value. Then, for the two records with missing values, one will be given a value of 'true' for the 'Windy' attribute, while the other will be given a value of 'false'.

This approach is meant to be used when the number of distinct values of an attribute is few. Although not as sophisticated as the previous two 'guessing' techniques, this technique is likely to be better than the first technique above. The reason being, instead of entirely ignoring a record with missing values, the values of other attributes (without missing values) that occurred together, are used.

Missing values as special values In this technique, missing values are treated as special values labelled 'unknown'. The advantage of this approach is that if there are any patterns relating missing values to class labels, they might be detected. However, in general applications, where there may be missing values at random, this approach is not useful.

Use classification techniques This is a very thorough technique, but it requires much more computation than any of the previous techniques described. In this approach, the classification techniques are themselves used to predict the entries of the missing values.

For an attribute A with missing values, this approach first considers set S of the training records for which the value of attribute A is known. The set S is used as a training dataset of another classification problem where the distinct values of attribute A are considered as classes, and the class labels of the original problem are treated as another attribute. By solving this classification problem, it becomes possible to predict the values of the attribute A in those records where it is missing.

3.4.5 Combining Classifiers

One interesting question that may arise to the reader is whether it is possible to design a classification algorithm that is inherently superior to the others in terms of accuracy. The infamous 'no-free-lunch' theorem states that this is impossible. Given any classifier, it is always feasible to artificially construct an equal number of test datasets in which it performs well and the ones in which it performs poorly. It follows that any classifier that is good for some applications may not be suitable for others. It is therefore desirable to consider using more than one classifier to solve a given problem because at least some of classifiers considered may be good for that particular task. Hence, some techniques are required to combine classifiers. There two broad techniques to combine classifiers in the research literature are as follows:

Bagging

This technique was published in 1996 by Leo Breiman in the *Machine Learning Journal*. In this technique, first each classifier is applied separately for a given test data record, selecting a class label for that record. This is like a democracy, where each classifier 'votes' for some class. Then, the class, which was selected the maximum number of times, is chosen as the class for that record.

Boosting

This technique was published in 1990 by R.E. Schapire in the *Machine Learning Journal*. It is similar to bagging except that the vote of each classifier is weighted according to its performance on the training dataset.

Both bagging and boosting techniques were originally intended to combine the classification models built by a single classification algorithm on different random samples of the training dataset. In fact, bagging is an acronym for 'bootstrap aggregating', which is a technique used to produce multiple random samples of the training dataset. In this technique, m records from the training dataset are selected at random (using uniform distribution) with replacement in order to produce a bootstrap sample, which is used to build a base classifier. Many such classifiers are built and combined using the bagging technique described above.

In boosting, the generation of random samples of the training dataset is achieved in a slightly different way. In 1995, an improvement to the basic boosting algorithm, known as *AdaBoost* (adaptive boosting), was proposed by R.E. Schapire *et al* and published in the proceedings of the European Conference on Computational Learning Theory. In AdaBoost, rather than selecting records from the training dataset at random with uniform distribution, each record is given weightage based on whether it is correctly classified by the classifiers built so far. Records that are difficult to classify are given a higher weightage so that they are more likely to be selected. In a sense, this approach tries harder to tackle difficult records.

An important context, in which combining multiple classifiers is really useful, is in multi-class problems. In some applications, although there may be many classes, it would be easy to create classifiers that discriminate well between just two classes. An example of such an application is OCR where it is required to recognize images of letters of the alphabet of some language. In this problem, there are as many classes as there are the letters of the alphabet.

Usually, it is quite easy to create a classifier that can discriminate well between any two of these classes. Since the classification problem actually contains more than two classes, several of these two-class classifiers can be combined to determine the actual class.

3.5 Classification Techniques

In this section, several classification techniques are described. Most of these techniques have their origins in the field of machine learning. The emphasis

on data mining is that the approaches should result in classification models that are easy to understand by humans. Another requirement is that the techniques be scalable to large datasets.

Some approaches, such as support vector machines (SVM) and most neural networks, are typically not considered as data mining solutions because they do not satisfy the criteria of understandability and scalability. Hence, these approaches will not be discussed here.

3.5.1 Decision Trees

Decision trees are one of the most popular classification models in current use in data mining.

Decision tree

A decision tree is a labelled tree where,

- Each interior node is labelled with an attribute.
- Each leaf node is labelled with a class.
- Each edge directed from an internal node *n* is labelled with a value or range of values of the attribute at node *n*.

Example 3.3 Consider the following dataset shown in Fig. 3.4 where each record represents the weather condition and the class attribute shows whether people generally play sports in that weather condition, or not.

Outlook	Temperature (Farenheit)	Humidity (%)	Windy	Class
Sunny	75	70	true	play
Sunny	80	90	true	don't play
Sunny	85	85	false	don't play
Sunny	72	95	false	don't play
Sunny	69	70	false	play
Overcast	72	90	true	play
Overcast	83	78	false	play
Overcast	64	65	true	play
Overcast	81	75	false	play
Rain	71	80	true	don't play
Rain	65	70	true	don't play
Rain	75	80	false	play
Rain	68	80	false	play
Rain	70	96	false	play

Fig. 3.4 Play outside?

An example decision tree constructed from this dataset is shown in Fig. 3.5. The root node represents the entire dataset of 14 records, 9 of which are classified as 'play' and 5 as 'DP' (do not play). Interior nodes are shown with rounded rectangles whereas the leaf nodes are shown with normal rectangles. Node labels are shown in bold.

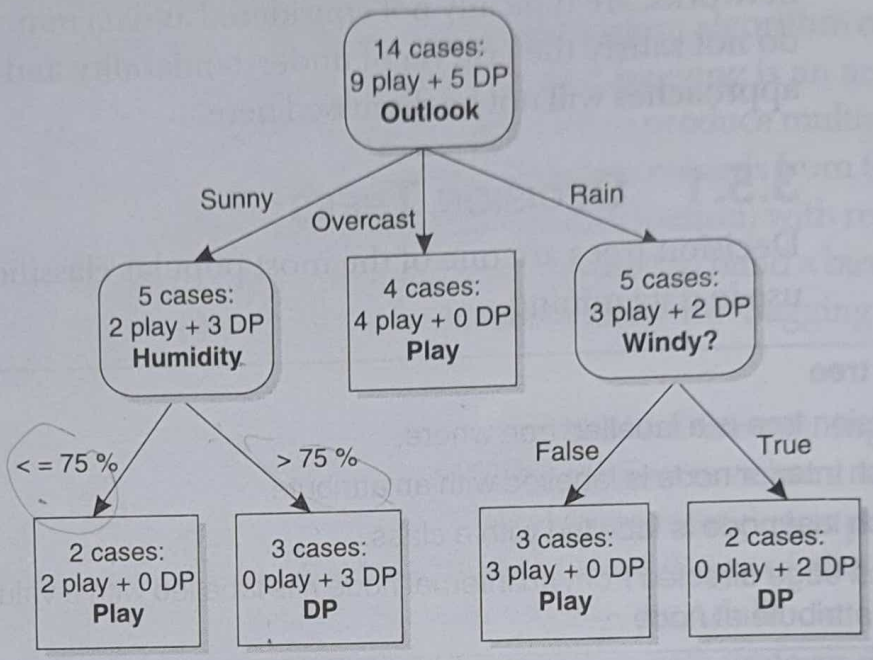

Fig. 3.5 A decision tree

The records at the root node are divided into three nodes at the next level, corresponding to the three possible values of the 'Outlook' attribute—sunny, overcast, and rain. There are five records in the first node representing 'sunny' as indicated in the dataset. Two of these are classified as 'play' and three as 'DP'. These records are further divided into two nodes, corresponding to the 'Humidity' attribute being greater than 75%, or not.

Classification using decision trees

A decision tree is a model of the dataset and can be used to predict the class label for new records. For any new record, it starts at the root node and at each edge a decision is taken whether to follow that edge or not, depending on its state. When a leaf node is reached, the class labelled at the leaf is output.

Example 3.4 Consider the situation, in Example 3.3, where a new unlabelled record has to be classified. For the left-most edge in Fig. 3.4, a question arises whether the 'outlook' attribute of this new record is equal to 'sunny'. If so, that edge is followed and the next node is reached. If not, the question moves on to the next edge. Eventually, a leaf node is reached. Notice that at the leaf nodes, all the records belong to simply one class. For instance, in the left-most leaf node, all the records are classified as 'play'.

In each leaf node of any decision tree, there will always be one such class that will dominate and be used as the label for that node. The remaining classes will either be absent in that leaf node, or their presence will be negligible.

Hence, when we reach the leaf node during classification, we choose the dominant or 'winning' class of that leaf node as the class of the new record being classified. As seen in this example, decision trees are very easy to interpret and use for classification. This has made them very popular in data mining applications.

Decision tree construction

Decision tree construction algorithms initially start with a root node that represents all the records in the training dataset. Next, they recursively partition the records into each node of the tree and for each partition, they create a child to represent it. The split into partitions is determined by the values of some attribute, known as the *splitting attribute*, which is chosen based on various criteria. This recursive splitting terminates at nodes that are pure, i.e. all the records in these nodes belong to only one class. This recursive procedure is shown in the pseudo-code in Fig 3.6.

Partition (Data *D*):
1. *If* all records in D are of same class: return
2. Evaluate splits for each attribute A
3. Use best split found to partition D into $D_1, D_2, ..., D_n$
4. *For each* D_i : Partition (D_i)

Fig. 3.6 Decision tree construction

Several criteria have been devised to determine the best split to partition a dataset. Two popular criteria are described in the following two subsections.

Information gain

The *entropy* H(D) of a dataset D is a measure of the disorder/variation/information in it. If all the records in the dataset belong to the same class, then the entropy would be zero. If the records are uniformly distributed among the different classes, the entropy would be maximized. In general, entropy is computed and defined as follows:

Entropy The entropy H(D) of a dataset D whose records are divided into m classes with probabilities $p_1, p_2, \ldots, p_m$ is defined as

$$H(D) = -\sum_{i=1...m} p_i \log p_i$$

For the purpose of this calculation, 0 log 0 is taken to be zero.

The ID3 algorithm is a decision tree construction algorithm that follows the overall approach described in Fig. 3.6. The best split for each node is chosen based on a criterion known as *information gain*. Given that a dataset D is split into $D_1, D_2, \ldots, D_n$, the information gain of the split is computed as,

$$\text{gain} = \text{H}(D) - \sum_{i=1}^{n} \text{P}(D_i)\, \text{H}(D_i) \tag{3.1}$$

In this equation, the first part is the entropy of the dataset before the split, whereas the second part is the average or expected entropy of the collection of the datasets after the split. Since entropy is a measure of the information content, the difference computed in Equation (3.1) represents the gain in the information content due to the split.

The ID3 algorithm selects the split with maximum information gain. The principle behind this choice is that such a split would result in a collection of the datasets whose average information content (the second part in Equation (3.1)) is the least. This is desirable because each dataset in this collection would contain records belonging mostly to one class. Hence, classification becomes possible.

In order to select the split with maximum information gain, the ID3 algorithm enumerates all the possible splits for each attribute value. That is, for each attribute A, and each value v of A, the records are split based on whether they contain v or not. To make this approach feasible, the ID3 algorithm only considers categorical attributes because the number of distinct values is small and hence, can be enumerated.

Example 3.5 For the sample data shown in Fig. 3.3, the entropy of the data D is calculated as

$$\text{H}(D) = \text{H}(p_{\text{play}}, p_{\text{don't play}}) = -p_{\text{play}} \log p_{\text{play}} - p_{\text{don't play}} \log p_{\text{don't play}}$$

Since the number of instances of 'play' in the D is 9 and that of 'do not play' is 5,

$$\text{H}(D) = -(9/14) \log (9/14) - (5/14) \log (5/14) = 0.94$$

Note, that the log function here is calculated to the base 2. The ID3 algorithm requires us to calculate the information gain for each possible split.

Let us calculate the gain for the split based on the 'Outlook' attribute. This attribute has three values—'sunny', 'overcast', and 'rain'. The original dataset is therefore split into three parts based on these values. Let us call these three parts as D_{sunny}, D_{overcast}, and D_{rain}. The entropy for these three datasets is as follows:

$$\text{H}(D_{\text{sunny}}) = -(2/5) \log (2/5) - (3/5) \log (3/5) = 0.97$$
$$\text{H}(D_{\text{overcast}}) = -(4/4) \log (4/4) - (0/4) \log (0/4) = 0$$
$$\text{H}(D_{\text{rain}}) = -(3/5) \log (3/5) - (2/5) \log (2/5) = 0.97$$

The probabilities of the outlook attribute being 'sunny', 'overcast', and 'rain' are 5/14, 4/14, and 5/14, respectively, based on their relative frequencies in the data. Thus, the information gain of this split can be computed as follows:

$$\begin{aligned}\text{gain} &= \mathrm{H}(D) - \sum_{i=1}^{n} \mathrm{P}(D_i)\,\mathrm{H}(D_i) \\ &= 0.94 - [(5/14) \times 0.97 + (4/14) \times 0 + (5/14) \times 0.97] \\ &= 0.25\end{aligned}$$

Similarly, other splits can be enumerated and their information gain computed. The ID3 algorithm will then select the split with the maximum information gain to form a level of the decision tree.

Gini value

The gini value of a dataset D is computed as

$$\text{gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$$

In the above expression p_i is the probability that the records of the i^{th} class occur in D. In this context, the goodness of a split of D into $D_1, D_2, \ldots, D_n$ is computed as

$$\text{gini}_{\text{split}}(D) = \sum_{i=1}^{n} \mathrm{P}(D_i)\,\text{gini}(D_i)$$

This measure is used in a classification algorithm known as SPRINT (Scalable PaRallelizable INduction of decision Trees). The principle behind this measure is similar to that of the information gain.

Example 3.6 For the sample data shown in Fig. 3.3, the gini value of the dataset D is calculated as follows:

$$\begin{aligned}\text{gini}(D) &= 1 - \sum_{i=1}^{m} p_i^2 \\ &= 1 - [(9/14)^2 + (5/14)^2] \\ &= 0.46\end{aligned}$$

Let us calculate the goodness of the split based on the 'Outlook' attribute. This attribute has three values—'sunny', 'overcast', and 'rain'. The original dataset is therefore split into three parts based on these values. Let us call these three parts as D_{sunny}, D_{overcast}, and D_{rain}. The gini values for these three datasets are as follows:

$$\begin{aligned}\text{gini}(D_{\text{sunny}}) &= 1 - (2/5)^2 - (3/5)^2 = 0.48 \\ \text{gini}(D_{\text{overcast}}) &= 1 - (4/4)^2 - (0/4)^2 = 0 \\ \text{gini}(D_{\text{rain}}) &= 1 - (3/5)^2 - (2/5)^2 = 0.48\end{aligned}$$

The probabilities of the outlook attribute being 'sunny', 'overcast', and 'rain' are 5/14, 4/14, and 5/14, respectively, based on their relative frequencies in the data. Thus, the goodness of this split can be computed as follows:

$$\text{gini}_{\text{split}} = \sum_{i=1}^{n} P(D_i)\,\text{gini}(D_i)$$

$$= (5/14) \times 0.48 + (4/14) \times 0 + (5/14) \times 0.48$$

$$= 0.343$$

Similarly, other splits can be enumerated and their gini values computed. Then the split with the maximum gini value is to be used to form a level of the decision tree.

Other decision tree approaches

C4.5 is a popular classification algorithm that is based on the ID3 algorithm and provides several enhancements over it.

Handling missing data During the tree construction, the C4.5 algorithm ignores the missing data values. Next, at the time of classification, it predicts the values of missing data based on the attribute values of other records, using the techniques described in Section 3.4.4.

Continuous data Unlike the ID3 algorithm, the C4.5 algorithm also handles attributes having numeric values such as age, temperature, etc. This is done by dividing the domain into ranges based on attribute values that actually exist in the training data.

Pruning Conventionally, decision tree construction is stopped at nodes that are 'pure', i.e. all the examples present in these nodes belong to a single class. However, it could also be stopped at nodes that are almost pure, i.e. the examples mostly belong to a single class. This approach is known as *pre-pruning*. Alternatively, the tree could be grown fully and then *post-pruning* can be applied–in this method, a sub-tree is replaced by a leaf if the misclassification on the test data does not change much.

Gain ratio ID 3 uses gain to measure the goodness of a split. One problem with gain is that it favours the attributes with many values. This leads to over-fitting. To avoid this, in C4.5, the goodness of a split is measured in terms of *gain ratio* which is computed as gain/$H(P(D_1), P(D_2), \ldots, P(D_n))$.

Decision rules Instead of a outputting a decision tree, C4.5 has the option to output decision rules of the form such as 'if these attributes have these values, then the record belongs to this class'.

C5.0 is a commercial version of C4.5 that includes other enhancements including boosting. In all these algorithms, computing the goodness of a split on a dataset D involves a scan of all the records in D. This could be costly if there are many attributes having numerous distinct values. SLIQ and SPRINT are scalable algorithms that overcome this problem by computing the goodness of all the possible splits of a node in a single scan of the dataset. These algorithms are discussed in Chapter 8.

3.5.2 Naïve Bayes

Naïve Bayes is one of the simplest classifiers and works surprisingly well for many applications especially those involving text classification. Given a record X to classify, the general approach is to output that class C_i whose probability of occurrence $P(C_i|X)$ is maximum. To estimate the value of $P(C_i|X)$, this classifier naively assumes that the attributes of X are independent of each other, therefore it is known as Naïve Bayes. Once independence has been assumed, the derivation is used to compute $P(C_i|X)$ as follows:

$$\begin{aligned} P(C_i|X) &= P(X \wedge C_i)/P(X) \\ &= P(X|C_i)\ P(C_i)/P(X) \\ &\propto P(X|C_i)\ P(C_i) \\ &\propto P(A_1{=}x_1|C_i)\ P(A_2{=}x_2|C_i)\ldots P(A_k{=}x_k|C_i)\ P(C_i) \end{aligned}$$

Here, the record X contains attributes A_j with values x_j. The denominator $P(X)$ is ignored because it is common for all the classes. The last line of the derivation is obtained by assuming independence between the attributes.

For classification, the values of $P(A_j{=}x_j|C_i)$ are *pre-computed* and stored for all possible attribute values and classes. At the time of classification, these probability values are used to estimate $P(C_i/X)$ as per the above derivation and the class with the maximum probability of occurrence is output.

Example 3.7 Consider an *e-mail* dataset as a training dataset, where some *e-mails* are classified as *spam* and some as *non-spam*. Consider three keywords 'sell', 'buy', and 'soap'. These three keywords have different probabilities of occurrence in the *spam e-mails* and *non-spam e-mails*. Their presence and absence can be treated as three binary attributes of *e-mails*. Assuming that the presence of any one of these keywords in an *e-mail* does not affect the probability of any of the other two keywords, the attributes are mutually independent.

For the sake of efficiency, the probabilities of these binary attributes in the *spam* and *non-spam* classes are *pre-computed*. In other words, from the entire training

data, P('sell' = present | *spam*), P('sell' = absent | *spam*), etc. for each keyword and class, is computed. These are simply the relative frequencies of the presence and absence of the keywords in each class.

Now, given a new *e-mail X* to be classified as *spam* or *non-spam*, it needs to be checked whether each of the three keywords is present or absent in this *e-mail*. Assume that the new *e-mail* contains 'sell' and 'soap', but not 'buy'. Then, by the Naïve Bayes formula,

$$P(spam \mid X) \propto P(\text{'sell'= present} \mid spam) \times P(\text{'buy'= absent} \mid spam) \times$$

$$P(\text{'soap'= present} \mid spam) \times P(spam)$$

The values of the probabilities on the RHS have already been *pre-computed*, therefore, this computation is a fast operation. Similarly, the indication for P(*non-spam* | *X*) is computed. Finally, for *X*, the class is output as *spam* or *non-spam*, depending on whether P(spam | *X*) or P(non-spam | *X*) is higher.

3.5.3 Bayesian Belief Networks

Naïve Bayes assumes independence between the attributes and hence, is computationally efficient and conceptually simple. Unfortunately, the independence assumption does not hold in many applications. If we remove the independence assumption completely, the problem of estimating $P(C_i/X)$ becomes exponentially complex because every attribute can be dependent on every other attribute.

Luckily, in practice, most attributes do not depend on the other attributes directly. An attribute A_1 may directly affect an attribute A_2 which in turn may directly affect another attribute A_3. But it could be that A_1 does not directly affect A_3. This means that given the value of A_2, the value of A_1 does not have to be known in order to predict the value of A_3. Put another way, A_3 is *conditionally independent* of A_1. Mathematically, this is defined as

$$P(A_3 = x \mid A_2 = y, A_1 = z) = P(A_3 = x \mid A_2 = y)$$

Example 3.8 Both heredity and environment may cause people to have an allergy to peanuts and thereby cause them to test positive during a skin-prick test. The result of a skin-prick test does depend on heredity and environment, but not directly.

If we know that a person has peanut allergy, then the probability that he will test positive no longer depends on heredity or environment. In short, heredity and environment affect whether a person gets peanut allergy, which in turn, causes a positive result during a skin-prick test.

A Bayesian Belief network (or simply bayesian network) is a data structure that represents the direct dependencies between the attributes in a dataset.

Bayesian belief network A Bayesian Belief network is a directed acyclic graph whose nodes represent attributes and an edge exists from a node A_1 to A_2 if A_2 is directly dependent on A_1.

Thus, if there is no path between two nodes in a Bayesian Belief network, then the corresponding attributes are conditionally independent. Since Naïve Bayes can be applied whenever the attributes are independent, here computations can be used which are similar to those in Naïve Bayes. In general, the probability $P(X|\ C_i)$ can be computed as

$$P(X|\ C_i) = P_{j=1\ldots k}\, P(A_j = x_j\,|\,\text{Parents}(A_j),\ C_i)$$

Here, the record X contains attributes A_j with values x_j. Parents(A_j) represent the nodes that are connected by edges pointing towards A_j, since these are the nodes that directly affect the value of A_j.

Example 3.9 Consider a training dataset of text documents, some of which are related to astronomy and the others are not. We thus have two classes—astronomy and general. Consider four binary attributes corresponding to the four keywords 'science', 'technology', 'physics', and 'chemistry'. Unlike in Example 3.7, these attributes are not independent of each other, but are related by the Bayesian network shown in Fig. 3.7.

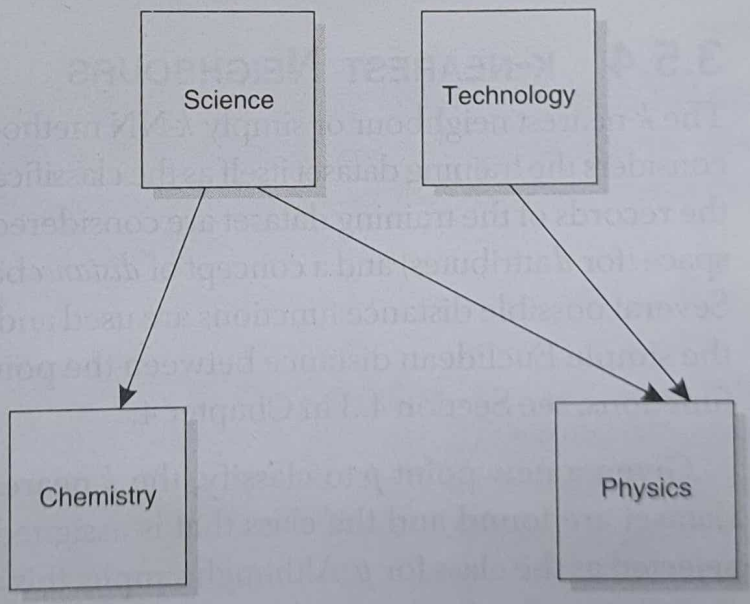

Fig. 3.7 A Bayesian Belief network

From this *Bayesian network*, we can understand relationships between the presences of these keywords. For example, the presence of the keyword 'Science' directly affects the presence of the keywords 'Chemistry' and 'Physics'. But the presence of 'Chemistry' does not directly affect the presence of 'Physics' and vice versa.

Suppose a document X is found, which contains the keywords 'Chemistry', 'Science' and 'Technology', but no 'Physics'. Based on this information, to evaluate whether the document is related to astronomy or not, the following is computed,

$$P(\text{astronomy} \mid X) \propto P(\text{'Chemistry'=present} \mid \text{'Science', astronomy}) \times$$
$$P(\text{'Physics'= absent} \mid \text{'Science', astronomy}) \times$$
$$P(\text{'Physics'= absent} \mid \text{'Technology', astronomy}) \times$$
$$P(\text{'Science'= present} \mid \text{astronomy}) \times$$
$$P(\text{'Technology'= present} \mid \text{astronomy}) \times$$
$$P(\text{astronomy})$$

The values of the probabilities on the RHS could be computed in the pre-processing phase making this computation a fast one. The precomputation of these probabilities is simply a matter of determining the relative frequencies of the presence and absence of various keywords in the astronomy documents. For example, in order to compute P('Physics' = absent | 'Science', astronomy), the relative frequency of the documents, which do not contain the keyword 'Physics' among the astronomy documents that contain the keyword 'Science' in the training dataset, is measured.

The same process is repeated, to compute P(general | X). Finally, the class for X is output as astronomy or general depending on whether P(astronomy | X) or P(general | X) is higher.

3.5.4 K-NEAREST NEIGHBOURS

The k-nearest neighbour or simply k-NN method is a simple technique which considers the training dataset itself as the classification model. In this approach, the records of the training dataset are considered as points in a d-dimensional space (for d attributes) and a concept of *distance* between the records is defined. Several possible distance functions are used and the most popular function is the simple Euclidean distance between the points. For more about distance functions, see Section 4.3 in Chapter 4.

Given a new point p to classify, the k nearest points of p in the training dataset are found and the class that is assigned to most of these k points is selected as the class for p. Although simple, this approach could be very slow during classification because it needs to search for the k nearest points among the entire training dataset, which could be huge. This search can be speeded up considerably by precomputing spatial indices or voronoi diagrams. However, the resulting speed is still likely to be slower than other approaches.

Example 3.10 Consider the 2-class data of heights and weights of students studying in class 7 and class 8 given in Fig. 3.8.

In order to determine the class of the student with roll no. 6 in the table, the *k*-NN method finds the nearest neighbours of this student. When $k = 1$, only one nearest neighbour of this student is found. Evidently the student with roll no. 3 has exactly the same height and weight as the student with roll no. 6. Hence, the roll no. 3 student is the nearest neighbour and belongs to class 7. Therefore, the student with roll no. 6 is also assigned to class 7 by the *k*-NN algorithm when $k = 1$.

Roll No	Height	Weight	Class
1	4'7"	50 kg	7
2	4'9"	52 kg	7
3	5'2"	55 kg	7
4	5'3"	54 kg	8
5	5'2"	56 kg	8
6	5'2"	55 kg	?

Fig. 3.8 Heights and weights of students

However, when $k = 3$, three nearest neighbours need to be determined. In this case, it turns out to be the students with the roll nos 3, 4, and 5. These three students belong to classes 7, 8, and 8, respectively. Since the majority of these neighbouring students belong to class 8, the student with roll no. 6 is assigned to class 8.

Example 3.10 demonstrates that the value of *k* in the *k*-NN method is critical. If it is too small, then the method may occasionally be affected by some exceptional records. If it is too large, then unrelated records will be used to decide the class of a given instance.

3.5.5 Neural Networks

Neural networks use a technology that attempts to produce intelligent behaviour by trying to mimic the structure and function of our nervous system. This nervous system is usually abstracted as a weighted directed graph where the nodes are neurons (nerve cells) and the edges between them are the connections between the neurons. The weight on each edge indicates the type (inhibiting or stimulating) and strength of interaction between the adjacent neurons.

Perceptrons

The simplest kind of neural network is a *perceptron* that consists of a single neuron having several real-valued or binary inputs and a binary output. The