292

DATA MINING TECHNIQUES AND APPLICATIONS IN THE POWER TRANSMISSION FIELD

Working Group D1.11

Task Force 04
(Data Mining & Material Insight)

April 2006

DATA MINING
TECHNIQUES AND APPLICATIONS IN
THE POWER TRANSMISSION FIELD

Working Group D1.11

Task Force D1.11.04
(Data Mining & Material Insight)

Convener TF D1.11.04: S Bell (Australia)

Members or Contributors: T Downs (Lead author Australia), D Allan (Convener WG D1.11, Australia), E Gulski (Netherlands), T McGrail (United States), T Saha (Australia), J Davis (Australia), T Blackburn (Australia), D Birtwhistle (Australia), S McArthur (United Kingdom).

PREFACE

In recent years there has been a significant increase in the availability and quality of data. This has been true in a number of areas and has been particularly prevalent in the power transmission field. Fortunately advances in techniques to extract relevant information and trends have mirrored this increase in data. In particular utilisation of data mining has proved successful in helping organisations leverage off these large volumes of data.

A survey was carried out within CIGRE SC D1 in 2003 by TF D1.11.04 of WG D1.11 to identify applications where data mining techniques had been successfully applied. From these results and from enquiries in other areas of power transmission it become apparent there was confusion relating to the term "data mining". The term meant different things to different people and clearly demonstrated the need for clarification in this area. There was also a general under utilization of the techniques available. This brochure was developed with these points in mind and introduces the variety of data mining techniques commonly employed. For this aspect of the work, special acknowledgment is given to TF member Prof Tom Downs.

There are a range of data mining techniques discussed, with varying levels of sophistication within the four areas of classification, clustering, association and regression. At the simpler level, graphical clustering and regression analysis can identify groups of data with distinct similarities. At the more advanced level, various forms of artificial neural networks emulate the learning and cognitive process of the human brain.

Examples are provided to demonstrate where these data mining techniques have been applied in the power transmission field. In highlighting where data mining has been utilised so far it is hoped to challenge the reader to identify other related applications. A detailed reference listing is provided for those more interested.

TABLE OF CONTENTS

1. Introduction

Since digital computers first appeared, the amount of data they store has been increasing at a phenomenal rate. The overwhelming nature of the data being generated has made it increasingly difficult to identify patterns and relationships that exist within the data that might be helpful in assisting with a company's operations or in gaining a competitive advantage. However many industrial organizations have recognized the potential gains that can be made from a detailed analysis of their data and have started introducing techniques such as data warehousing [1] that are aimed at bringing the whole of an organization's data together in such a way that analysis is facilitated. It is worth noting that a 1996 study by the International Data Corporation reported that the average 3-year return on investment for data warehousing was as high as 400% [1].

Data warehousing transforms operational data into a form more suited to data mining, which is a process aimed at discovering useful structure in large databases. *Structure* here refers to patterns or relations that exist within the data. A useful *pattern* is one that allows helpful predictions to be made from new data. For instance, in a business application, one might have a database on customer purchases in which behaviour patterns discovered for previous customers can be used to identify current customers who are more likely to switch products and who therefore should be given special attention. A *relation* is a property discovered in data that indicates a dependency between attributes, a simple example being the fact that people who own computers are more likely to buy online than those in the community who do not.

In scientific applications, stored data typically consists of sets of measurements. For instance, in medicine, the data may be stored as blood pressure, heart rate, etc. If there are n such measurements per patient, they are stored in vectors of length n. Depending on context, the entries in a data vector may be referred to as *variables*, *features*, *attributes* or *fields*. Not all attributes are measured values. Continuing with the medical example, the data vector for each patient might contain the patient's age, sex and marital status. Thus, some entries in a data vector represent non-numerical quantities, and variables such as sex and marital status are known as *categorical* variables. Categorical variables have only a finite number of possible values and so can be represented by integer values in the data vector. For instance, marital status might be given the value 1 for single, 2 for married, 3 for widowed.

Taking the medical data example a little further, it is fairly obvious that data stored on a patient would normally contain rather more than a vector of numerical and categorical variables. There is likely to be textual commentary on the patient's condition and possibly image data from X-rays and the like. These kinds of data can be interpreted by human intervention and significant features added to the patient's data vector in the form of additional numerical values. Thus, in this, and in most other practical situations, the important data can be stored in a vector of numerical values and most data mining techniques have been developed on the assumption that data are represented in this way.

2. Data Mining Techniques

Data mining aims to extract useful information from large databases. The techniques involved draw heavily upon the long-established science of statistics and the more recently developed methods of machine learning. Statistics have been used for many years as a means of creating models that can assist with company activities, e.g. load forecasting in the power industry. *Machine learning* provides practical enhancements to statistical techniques by allowing computing machines to construct statistical models in an automated fashion directly from data.

There are four basic machine learning tasks in data mining: classification, clustering, association and regression, as illustrated below in Figure1:

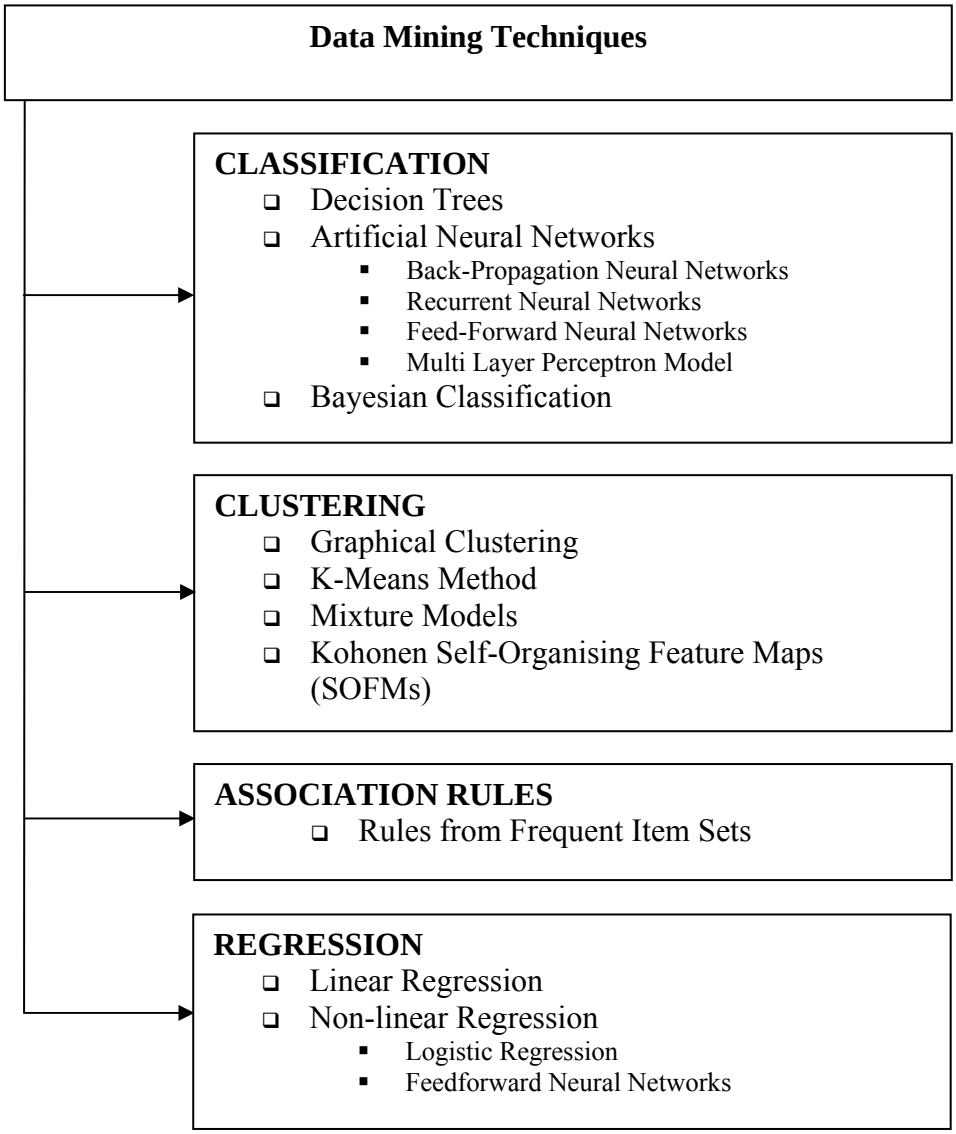


Figure 1

- In *classification,* we seek to distinguish between different classes of objects based upon data descriptions of these objects. A typical example from the banking industry is to classify people seeking loans into those who are good risks and those who are not. This can be done in an automated fashion based upon data regarding income, age, previous history, etc.

- *Clustering*, like classification, seeks to identify groups of data that have similar characteristics. It differs from classification in that the properties of groups that might be of interest are not predefined. Classification, where the groups *are* predefined (e.g. good risk, bad risk in the above example) is known as a *supervised* learning problem. Clustering is known as *unsupervised* learning. As an example, clustering is frequently used in the advertising industry to identify different groups who purchase a company's products. The company may have no prior knowledge of the types of people who buy its goods, and clustering can identify different groups using data on income, age, marital status, number of children, etc. This information can then be employed to target the different groups in advertising campaigns.

- *Association* learning is concerned with identifying rules that tend to apply among items in data. These are not usually hard and fast rules, but rules that apply with reasonably high probability. For instance, a database holding information on traffic patterns in a telecommunication network could be mined to determine sets of circumstances that tend to be associated with congestion in a part of the network. This information could then be used to establish rules for redirecting traffic around that part of the network in order to avoid incipient congestion.

- *Regression* is concerned with determining a description of data in terms of some mathematical function. When such a function has been found, its role is to assign a numerical value to any data item. A regression function is most frequently, but not always, employed to predict future values of some quantity based on available data. A typical example is load forecasting where a function is determined that provides a good fit to previous load data and can be extrapolated to provide predictions of future load.

We will now examine these four basic machine learning tasks in a little more detail. These will be explained in terms of examples from a wide variety of applications. More detailed examples, selected specifically from the power industry, will be described in Section 3.

2.1 Classification

As described in Section 1, we can assume that each data item is stored as a vector of numerical values. If we have n such data items, we will represent them by $\boldsymbol{x}_i$, $i = 1, 2, \ldots, n$.

In general, each of these data vectors will belong to one of several possible classes, and this can be represented by assigning to each vector $\boldsymbol{x}_i$ a class label y_i. The y_i are usually integer values that distinguish between the various classes.

The classification task is then a two step process. We construct a classifier using examples from a database. This is known as the *training phase,* where the classifier "learns" its task using one of the machine learning techniques described below. The second step is to apply it to the classification of additional data independent of that used in the training phase. This is known as the *test phase* and allows us to assess how well the classifier has learnt its task. If it performs well in the test phase, it is probably ready for use in the field, where it will be required to classify previously unseen data all the time.

We will now describe three of the more widely used methods of constructing classifiers. There are other long-established techniques, such as *fuzzy set and nearest-neighbour methods* [2] as well as more recent approaches such as *support vector machines* [3] but the methods we describe here should be sufficiently illustrative of the kinds of concepts involved.

2.1.1 Decision Trees

Of all the classification methods that have proven useful in practice, the decision tree is the most simple conceptually. It is constructed from data in the form of a tree that is put together in much the same way as a flow chart representation of a computer program. The process is most easily explained in terms of an example and for this we will draw upon a problem currently faced by the mobile phone industry.

Mobile phone companies continuously have to deal with the problem known as *customer churn*, a term that reflects the tendency of customers to switch from one company to another. This movement occurs because customers search for cheaper rates or because they are attracted by special deals for signing up with a different company. We might wish to construct a decision tree using customer data in order to classify customers into churners and non-churners. A very simple form of such a tree is shown in Figure 2, which is adapted from [1].

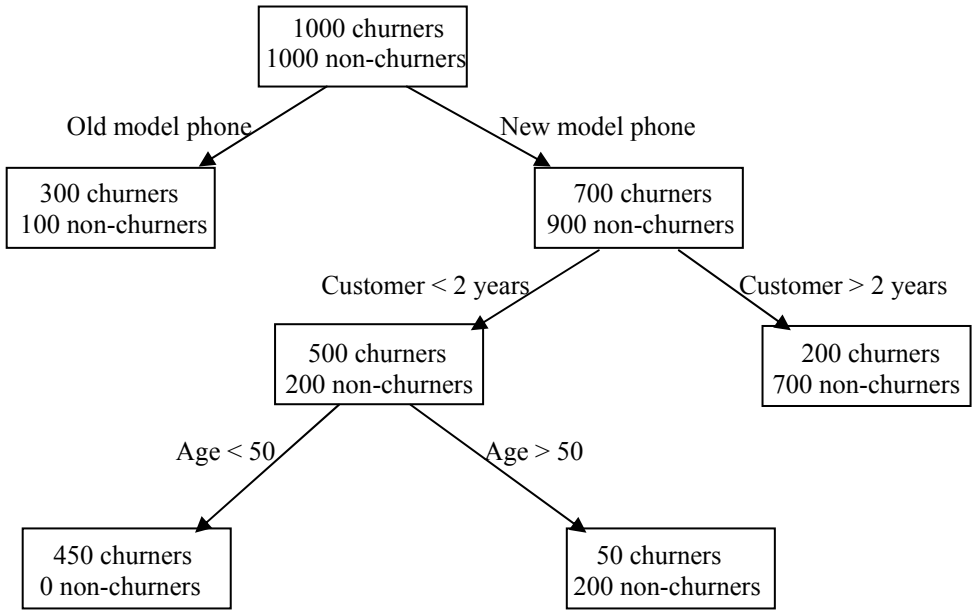


Figure 2

The first box at the top of Figure 2 is known as the *root* of the tree. The tree then branches out in a downward direction. The boxes at which the branching terminates (i.e. the four with no outgoing arrows) are called *leaves* of the tree. The remaining boxes, which have both incoming and outgoing arrows, are called *nodes* of the tree.

The figure illustrates some of the useful properties of decision trees. First of all, it is easy to see how the tree has been constructed - the structure of other types of classifier, as we shall see, is generally much less obviously related to the data on which it is based. Secondly, it clearly shows the information that it was constructed to reveal – that is, the types of customer who are likely to churn. Thirdly, the decision tree is able to handle raw data without any preprocessing which, as we shall see later, is usually necessary with other techniques.

The tree in Figure 2 was developed to try to classify cell phone customers into churners and non-churners. It would do this perfectly on the training data if each of the leaves of the tree were like the bottom-left box and contained just one type of customer. This box does perfect classification on the training data because it tells us that a particular customer group (those with new model phones, who have been customers for less than 2 years and who are aged less than 50) are all churners. But in practical problems, perfect classification is usually impossible and so we have to try and construct the decision tree so that it distinguishes between classes as well as possible.

A major step in the construction of decision trees is the splitting of attributes to provide the branches in the tree. The order in which attributes are split usually has a significant effect on classification performance. In Figure 1, the first attribute chosen for splitting is the phone model type (old or new). But we could equally well have chosen customer age or the length of time a person has been a customer for the first split. A well-known software system for constructing decision trees, called C4.5 [4], uses an entropy measure to determine, at any node in the tree, the attribute that should be selected for splitting. The attribute selected at any node is the one that provides the greatest reduction in the entropy measure, implying that splitting the selected attribute provides the greatest gain in information.

When attributes are categorical, e.g. the phone model type (old or new) in Figure 1, the manner in which the split is carried out is obvious. But if the attribute is real valued, as with the other two attributes in Figure 2, the choice of value at which the attribute is split is important for performance. In Figure 2, the split for customer duration was made at 2 years and the split on customer age at 50 years. C4.5 again uses the entropy measure to assist in determining the best value for a split.

When a decision tree has been constructed for a practical problem, some of its structure will usually be caused by anomalies in the training data such as noise or outliers (e.g. wrongly classified samples in the training data). These anomalies are not representative of the "true" data and the fact that the tree incorporates them into its structure is referred to as *overfitting* the data. As we shall see later, overfitting is a problem with all systems that learn from data and, unless it is properly dealt with, it leads to poor classification performance when the tree is applied to independent test data. Accommodating anomalous data causes the decision tree to be more complex than necessary and it is dealt with by pruning the tree. Statistical measures can be used to identify and remove

the tree's least reliable branches (e.g. those based upon very few data samples) and this generally results in an improvement in the tree's ability to correctly classify independent test data.

Because of their simple and revealing structure, decision trees can provide insights into the logic underlying decision boundaries that exist in data. For this reason they have found wide application in data mining. They do, however, suffer from some limitations. For instance, because of the way that attributes are split in the construction of a tree, they divide up the space of data vectors into rectangular regions and this is not suitable for many problems in science and engineering. They also completely ignore any statistical correlations that might exist among attributes. But the fact that they have been widely applied in practice indicates that there are many areas where these limitations are not especially relevant. More details on the use of decision trees in data mining can be found in the texts on data mining included in the references.

2.1.2 Neural Networks

Neural networks are simple electrical circuits designed to emulate some of the learning processes that take place in the brain. For experimentation and research, they are implemented in software but sometimes, e.g. when a neural network product has been developed for which there is high demand, they are implemented in hardware.

Neural networks for pattern classification make use of the fact that when biological creatures learn something, the strengths of connections between neurons change. Consider a medical example, in which we wish to train a neural network to produce a diagnosis on people suspected of having diabetes. This is a classification problem with 2 classes, where medical data are classified as either indicating diabetes or not. To achieve this, we can use a network like the one in Figure 3, where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ is a data vector that is applied as an input at the left-hand side of the network, as shown. This data vector contains measurements such as blood pressure, plasma glucose, body mass index, etc.

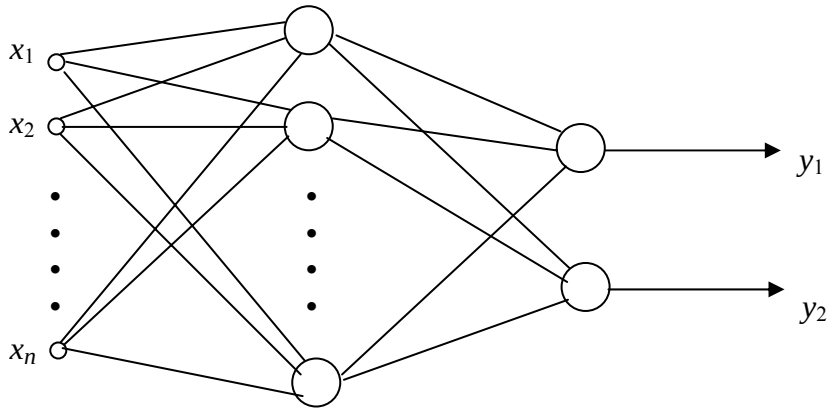


Figure 3

The small circles at the left of the network are simply input terminals and the larger circles are neural elements. These neural elements have output values that vary continuously between 0 and +1 with zero representing the quiescent state and values increasing from zero representing an increasing neural firing rate. The input vector is fed through a set of weighted connections to the neural elements in the middle of the

network and the outputs of these neural elements are fed through additional weighted connections to the two neural elements on the right-hand side, which provide the network outputs. We will now explain how the network can be trained to implement the diagnosis.

For a given input vector, the diagnosis will be indicated by the values of the network outputs. Suppose we decide to train the network so that input data taken from a person with diabetes will cause the output values $y_1 = 1$ and $y_2 = 0$, and input data from a person without diabetes will cause $y_1 = 0$ and $y_2 = 1$. Let us suppose we have 300 data vectors with roughly half being positives (diabetes) and half being negatives (no diabetes). These are the *training data*. We commence with a network where the weighted connections have small random values. In these circumstances, if we feed a data vector to the network input, the output will be unpredictable. But we know the outputs the network should give and so we can measure the error between the network outputs and the desired outputs. On the basis of this, using an algorithm based upon elementary differential calculus [6], we can now make small adjustments to the weighted connections so that the error made by the network on this vector is slightly reduced. We now feed another vector to the network and repeat the process. By the time we have gone through the full set of training data in this way, we can expect that the network has a better chance of classifying the first data vector correctly when it is fed to the network input a second time. But training is quite a slow process and we usually have to go through the full set of training data many times to achieve satisfactory performance. Once trained, however, a network like the one in Figure 3 makes its decisions very quickly – signals simply have to travel from left to right for the decision to be made.

Usually training is carried on until outputs are reasonably close to the desired values. It is possible to use thresholding to force the outputs to the binary values of 0 and 1 but it is usually preferable to have real-valued outputs available because the user then has a measure of confidence in the decisions made by the network. If outputs are very close to the binary values, the decision can be accepted with high confidence, but in a case where they are some way away, the user might choose to decide the example is not classified either way. It is also possible to adjust the manner in which outputs are determined so that a measure of the probability of class membership is obtained [7].

In practical applications, data tend to be noisy. The noise may come from, for instance, inaccurate measurements or erroneous labeling of classes. If training is continued long enough on a set of training data, the neural network, in continuing to reduce the errors it makes on the training set, will eventually start to learn the added noise. This is a manifestation of overfitting. As mentioned above, this has the effect of reducing performance on independent test data (or data that the network will be exposed to when put into practical operation). In order to avoid overfitting, it is normal practice to divide available data into a training set and a test set and, as training proceeds, to occasionally assess performance on the test set. When overfitting begins to occur, performance on the test set will begin to deteriorate and this indicates that training should be terminated.

The network in Figure 3 can be extended in the obvious way to deal with more than two classes. Networks of this sort, known as *feedforward networks*, have been applied in many areas, not just data mining. And as we shall see later, there are other types of neural network that can be applied to other kinds of problem in data mining.

2.1.3 Bayesian Classification

Suppose we have a data vector $\boldsymbol{x}$ whose class label is unknown. Suppose also that there are m possible classes C_i, $i = 1,2, \ldots m$, that $\boldsymbol{x}$ could belong to. The Bayesian approach is a probability method and it allows us to calculate, on the basis of available data, the probability that $\boldsymbol{x}$ belongs to class C_i. This probability is written $\Pr(C_i \mid \boldsymbol{x})$ which should be read as the probability of class C_i given the vector $\boldsymbol{x}$. A rule formulated in the eighteenth century by the English clergyman Thomas Bayes allows us to estimate this probability as follows:

$$\Pr(C_i \mid \boldsymbol{x}) = \frac{\Pr(\boldsymbol{x} \mid C_i)\Pr(C_i)}{\Pr(\boldsymbol{x})} \qquad (2.1)$$

In (2.1) the term $\Pr(\boldsymbol{x} \mid C_i)$ is the probability that vector $\boldsymbol{x}$ will occur in the data if the class is indeed C_i. Mathematicians refer to this as the *likelihood* of the data item $\boldsymbol{x}$ given the class C_i. Usually the vector $\boldsymbol{x}$ has several attributes which means that $\Pr(\boldsymbol{x})$ and $\Pr(\boldsymbol{x} \mid C_i)$ are multidimensional probabilities. Consequently, large amounts of data are required to estimate these probabilities with reasonable accuracy, and in addition, the evaluation of equation (2.1) is computationally quite demanding. For instance, if $\boldsymbol{x}$ was made up of 10 binary variables, we would have to estimate 2^{10} probabilities, *i.e.* 512 probabilities. Because of this, it is common to make a major simplifying assumption, which leads to a method that is much simpler to implement and which is known as the *naïve Bayes* method [2], [5].

The simplifying assumption is, essentially, that the features used in a classification problem are statistically independent. Once this assumption is made, multidimensional distributions can be dispensed with altogether and the computations become far simpler. The independence assumption rarely, if ever, holds in a practical problem but the Naïve-Bayes classifier has proved to be competitive with other methods, especially for smallish data sets, and is particularly effective for data where some of the feature values are not available for some patterns. This latter case is very important in some application areas, e.g. in medicine, where sometimes a full set of readings of a patient's condition cannot be taken. More details are given in Appendix A1.

2.2 Clustering

Clustering is a process of collecting data into groups according to their similarity. It is rather like classification except that the data objects have no class labels.

As we pointed out earlier, learning to classify objects is a form of supervised learning because it requires the learner to associate data with labels that are provided by a "teacher". The fact that there are no labels to assist with the clustering process means that it has to be carried out in an unsupervised fashion. These two forms of learning are common in human activity. During our lifetimes we have learnt (by being told) the names of a vast number of things. For instance, we know there is a class of creature called animals and within that class there are many subclasses – cats, dogs, cows, etc. We learn to classify them in terms of their labels, i.e. their names. But before language ever developed, human beings learnt to classify animals without the use of labels. And in

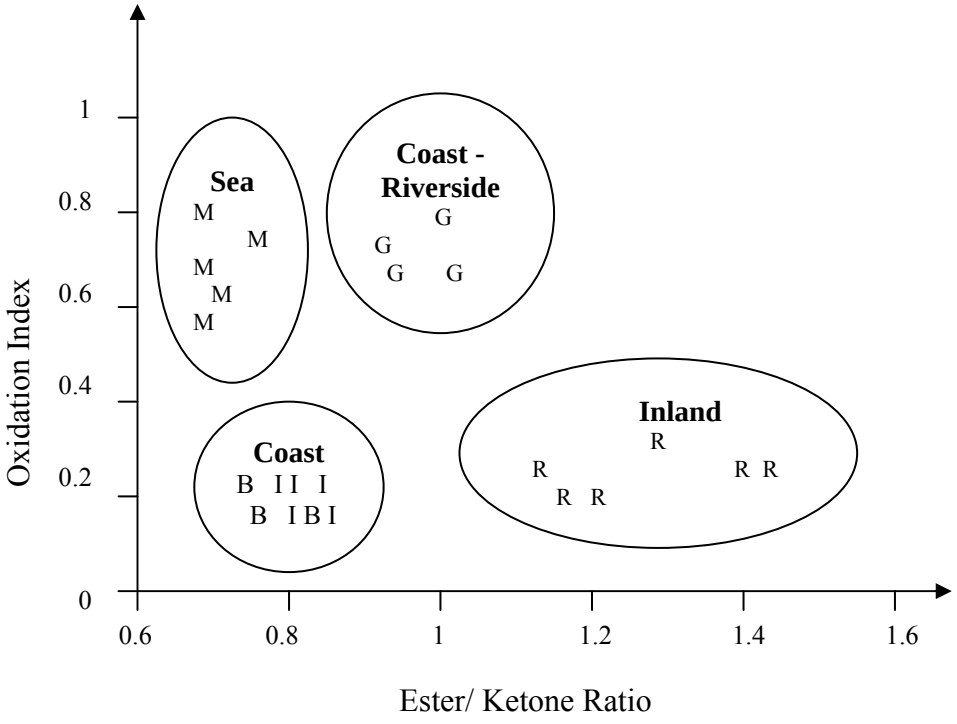
particular some of them learnt, in a totally unsupervised fashion, which animals to run away from. Sometimes it was the last thing they learnt.

To get a feel for the issues in clustering, we can consider a problem that is being widely studied these days, namely the analysis of gene expression data. Gene expression means gene activity and techniques are available for measuring activity numerically. An array of genes is experimented on in various ways (e.g. application of heat shock) and the activity of each gene is measured. One aim of this work is to identify groups of genes that behave similarly and this can be achieved by data clustering. Suppose just three experiments have been carried out on a set of genes and measurements have been taken on the activity of each gene. Suppose the three numerical measurements for a given gene are *x*, *y* and *z*. The *x*, *y*, *z* numbers clearly define a point in 3-dimensional space. And genes that behave similarly will have *x*, *y*, *z* values that are similar. Consequently, they will be located close together in 3-dimensional space. If a fourth experiment is carried out, genes that behave similarly will have measurements that lie close together in 4-dimensional space, and so on. Thus, if we carry out *n* experiments, and we wish to identify genes that behave similarly throughout the *n* experiments, we need to look for clusters that are close together in *n*-dimensional space.

2.2.1 Graphical Clustering

A basic graphical clustering example is given in the next Figure 4 [8]. The adapted plot illustrates two important chemical indicators for insulator samples from one manufacturer: the oxidation index and the ester/ketone ratio. A number of individual groups can be identified in the data. It can be seen the sea-side insulators from location M and the coastal insulators from a foggy area at location G have high values of oxidation index. In addition the coastal insulators have slightly higher ester/ketone ratios. Inland insulators from location R have the highest values of the ester/ketone ratio. On the other side, coastal insulators from both locations, B and I, have the lowest values of the oxidation index and ester/ketone ratio.

Figure 4



Thus, clustering is basically a process of identifying data items that lie close together in the space of attributes and so clustering methods are generally based upon the measurement of distance between data items.

2.2.2 The k-Means Method

The k-means method of clustering can be explained very simply. k is the number of clusters and each cluster is represented by its mean position. At the outset, if we have N data objects, k of these are randomly selected and the position of each one is treated as a cluster mean. Each of the remaining objects is then assigned to the cluster to which it is closest. After this assignment, the mean of each cluster is recalculated and *all* data objects are now assigned to clusters whose mean is the closest. The procedure is repeated until the clusters remain unchanged. A simple, one-dimensional example will demonstrate how it works:

Suppose our data objects are {1, 5, 7, 10, 15, 17, 21, 25, 30} and let $k = 2$. Suppose now that our random selection of 2 objects gives us 1 and 7. These are treated as the initial mean values so we write $m_1^{(1)} = 1$ and $m_2^{(1)} = 7$. The remaining objects 5, 10, 15, 17, 21, 25 and 30 are all closer to 7 than to 1 so this first step yields clusters $C_1^{(1)} = \{1\}$ and $C_2^{(1)} = \{5, 7, 10, 15, 17, 21, 25, 30\}$. The mean values of the two clusters are now computed as $m_1^{(2)} = 1$ and $m_2^{(2)} = 16.25$. We then reassign all the objects to the clusters whose mean is closest. This gives $C_1^{(2)} = \{1, 5, 7\}$ and $C_2^{(2)} = \{10, 15, 17, 21, 25, 30\}$. The new means are $m_1^{(3)} = 4.33$ and $m_2^{(3)} = 19.67$. The new clusters are $C_1^{(3)} = \{1, 5, 7, 10\}$ and $C_2^{(3)} = \{15, 17, 21, 25, 30\}$. The process now terminates because computation of the new mean values leads to no change in the clusters.

Normally, of course, we are dealing with data vectors whose dimension is greater than unity, but the procedure remains the same with the Euclidean distance being employed to compute the measure of closeness. For instance, with 4- dimensional vectors, if the mean is positioned at the point w, x, y, z, then the distance D to a vector at the point w_1, x_1, y_1, z_1 is computed as:

$$D = \sqrt{(w-w_1)^2+(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} \qquad (2.2)$$

Although it is widely used, the k-means method does have a number of disadvantages. The method can only be used where the mean of a cluster is defined, which means that it cannot generally be used when data contain categorical variables. In addition, the user has to specify k at the outset so different values of k often have to be tried before a satisfactory result is obtained. Other drawbacks are that it is unable to discover nonconvex clusters and that it is sensitive to noise and outliers in the data. Some of these disadvantages are overcome by the probability modeling approach we discuss next.

2.2.3 Mixture Models

Mixture models are useful for modeling statistical data where the underlying structure is unclear or complicated. To gain an appreciation of how mixture models operate, consider first the artificial situation where we have data consisting of a set of numbers generated by two different Gaussian distributions. These data will form two clusters that we can label X and Y. If the data from the two clusters are labeled in this way, we could then use simple formulae to estimate the mean and standard deviation of each distribution and thereby obtain a concise description of the data clusters.

But how can we obtain a similar description if there are no labels attached to the data and we have no distributional information to help us identify the clusters? This is where the mixture model comes in. As its name implies, a mixture model is based upon the assumption that the data are generated by a mixture of distributions. An example of a mixture model is given in Appendix A2.

Like the *k*-means method, the mixture model method of clustering requires the user to choose the number of clusters at the outset. But some techniques are available that allow estimation of the best number of clusters to use. For more details on this see reference [9].

2.2.4 Self-Organising Feature Maps

The Self-Organizing Feature Map (SOFM) was introduced by the Finnish researcher Teuvo Kohonen [10] and is based upon a form of learning, called *competitive* learning, that is common in the brain. Thus, the SOFM is a type of neural network.

Kohonen's learning algorithm is usually applied to a planar array of neural elements in the manner indicated in Figure 5. The inputs to the array are real-valued, and we will suppose there are N of them, giving us the input vector $\boldsymbol{x} = [x_1, x_2, \ldots, x_N]$. Each neural element is connected to the full set of inputs, with neuron i connected to input x_j by weight w_{ij}. The full set of weights to neuron i will be denoted by the vector $\boldsymbol{w}_i$ and, prior to learning, all weight values are set to small random values.

Under the competitive learning rule, for a given input $\boldsymbol{x}$, the "winning" neuron i^* is the one whose weight vector is closest to $\boldsymbol{x}$. Once a winner has been identified, the Kohonen learning rule below is applied:

$$\Delta w_{ij} = \eta(x_j - w_{ij}) \qquad \text{for all } i \text{ close to } i^*.$$

where η is a small positive constant. For any neuron i to which this rule is applied, the effect is to move its vector of weights $\boldsymbol{w}_i$ a little closer to the input vector $\boldsymbol{x}$. As more and more data vectors are presented, different neurons and their neighbours have their weight vectors driven close to different types of input vector. The ultimate effect is that different groups of neurons become receptive to different types of input. To see this, we will examine the effect of a weight vector moving closer to a particular input vector.

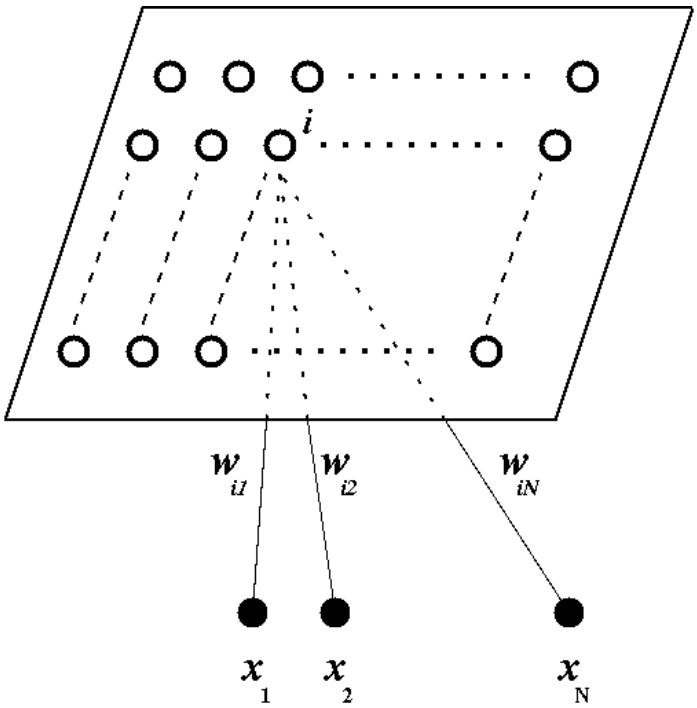


Figure 5

Consider again neuron i whose weight vector is $\boldsymbol{w}_i$. With input vector $\boldsymbol{x}$ the weighted input to neuron i is $w_{i1}x_1 + w_{i2}x_2 + \cdots + w_{iN}x_N$. This is an inner product of the two vectors $\boldsymbol{w}_i$ and $\boldsymbol{x}$, so it can be written $|\boldsymbol{w}_i| \cdot |\boldsymbol{x}| \cos\phi$, where ϕ is the angle between the vectors $\boldsymbol{w}_i$ and $\boldsymbol{x}$. And, as vector $\boldsymbol{w}_i$ is moved toward vector $\boldsymbol{x}$, the angle between them decreases, meaning that $\cos\phi$ increases. Thus, when vector $\boldsymbol{w}_i$ moves toward vector $\boldsymbol{x}$, the weighted input to neuron i increases. And since, under Kohonen's algorithm, this only occurs to neuron i and its neighbours, neuron i and its neighbours become receptive to inputs like $\boldsymbol{x}$.
Kohonen was able to exploit this effect to construct a speech-driven typewriter [11] in which groups of neural elements responded to different sounds.

2.3 Association Rules

As stated earlier, association learning is concerned with identifying rules that tend to apply with reasonably high probability among items in data. A rule consists of a proposition and a consequence, e.g. "If my football team wins, then I am happy".

This is a *deterministic* rule because I am always happy when my football team wins. Thus, for a deterministic rule, if the proposition is true, the consequence is also true. For large data sets, we are usually more interested in rules that tend to be true. These are *probabilistic* rules for which, if the proposition is true, the consequence is true with probability p.

2.3.1 Rules from Frequent Item Sets

The mining of association rules was first developed in relation to *market basket analysis*, which is aimed at identifying items that tend to be purchased together (and hence are added to the same shopping basket). The market basket provides a convenient and easily understood framework in which to introduce some of the terminology and ideas concerning association rule mining.

Each entry in a database of market basket data is a set of items purchased by a customer. A simple version of an association rule in this context is "If a customer purchases items *A*, *B* and *C*, then that customer also purchases item *D*". This can also be stated as "Purchase of items *A*, *B* and *C* implies purchase of item *D*". The mathematical symbol for implication is "$\Rightarrow$", so the rule can be written

$$\{A, B, C\} \Rightarrow D$$

and, more generally, we write $I \Rightarrow X$, where I is a set of items (an *itemset*) and X is some other item.

The rule $I \Rightarrow X$ is said to hold with *support* s, where s is the percentage of transactions in the database that contain both I and X. The accuracy of the rule is measured by the confidence, c, which is the percentage of transactions containing I that also contain X. Note the difference in these two definitions. For the case where there are n entries in the database, with m containing itemset I and l containing both I and X, we have $s = l/n$ and $c = l/m$.

Table 1 shows an example in which 12 transactions have involved selections from 6 items. In this case, the rule $A \Rightarrow B$ has support $s = 3/12 = 0.25$ and confidence $c = 3/6 = 0.5$. Note that confidence here is equivalent to the probability that a rule is true.

Transaction	*A*	*B*	*C*	*D*	*E*	*F*
t_1	0	1	0	1	0	1
t_2	1	0	0	1	1	0
t_3	1	1	0	0	0	0
t_4	0	1	1	0	1	0
t_5	1	1	0	0	1	0
t_6	1	0	1	1	0	1
t_7	0	0	1	0	0	1
t_8	0	0	0	1	0	1
t_9	0	0	0	0	0	0
t_{10}	1	1	1	0	1	0
t_{11}	1	0	0	1	1	0
t_{12}	0	0	0	1	1	0

Table 1

These two measures provide a basis for mining association rules from large databases. Mining for rules is a two-step process:

1. Find all frequent itemsets – these are the itemsets whose support is greater than some prespecified threshold.

2. Generate strong rules from these itemsets – here "strong" is determined by some prespecified threshold for confidence.

Finding frequent itemsets is obviously quite simple but, for large databases, finding them all can be computationally expensive. If the total number of items is k, the number of possible itemsets is $2^k - 1$. This number rapidly becomes very large with increasing k. If there are 1000 items, the number of possible itemsets is greater than the number of elementary particles in the universe. And for many practical applications, the number of items can be as large as 10^6.

Because of the potentially vast numbers of itemsets, methods have had to be found for reducing the number without losing important information. The best known association rule algorithm is called the *Apriori algorithm* and is widely used commercially. It is based upon the observation that an itemset I can only have large support if all its subsets have large support also.

The Apriori algorithm commences by identifying all single items whose support is above a specified threshold. This is done by a straightforward count of the occurrences of each item in the set of transactions in the database. Those items whose support is found to be below threshold cannot contribute to itemsets whose support is above threshold. Thus, on the next step, when the algorithm seeks *pairs* of itemsets whose support is above threshold, it needs only to check pairs of individual items that were found on the first step to be above threshold. On the third step the algorithm seeks all itemsets of size 3 that are above threshold and for this it needs only to check through the items in the pairs found in step 2. The algorithm continues in the obvious way. For more details see [9] and [12].

In its basic form, the Apriori algorithm can be computationally expensive when applied to large databases. For instance, if the number of individual items found to be above threshold on the first step is 5000, the number of pairs it will need to consider on step 2 is greater than 10^7. There is also the fact that if the data set is large enough, it won't fit into main memory, meaning that reading through the full set of data is time-consuming. A variety of methods have been proposed for improving the efficiency of the algorithm.

Probably the simplest conceptually is to apply the Apriori algorithm to a randomly sampled subset of the data. This allows us to trade off accuracy for efficiency. The sample size is chosen to fit into main memory. The Apriori algorithm then computes the frequent itemsets from the sample, using a support threshold that is set to a slightly lower value than required in order to increase the probability of identifying from the sample all of the frequent itemsets in the full data set. When the Apriori algorithm completes its computations, the frequent itemsets it identifies then have their true frequencies determined from the *full* data set so that only one pass through the full data set is required to obtain this information. If required, a second pass through the full data set can be employed which guarantees that no frequent itemsets are missed (see [12]). For other methods of speeding up the Apriori algorithm, see [2].

2.3.2 Generating Rules

Once the frequent itemsets have been identified, association rules can be generated from them. These rules are only of interest if they are reasonably strong, where strength is defined in terms of the support and confidence underlying the rule. Support is specified

prior to the identification of frequent itemsets and strong association rules are identified as those whose confidence measure is above a specified threshold. The confidence of any potential rule can be calculated using the formula

$$Conf(I_1 \Rightarrow I_2) = \frac{Support\{I_1, I_2\}}{Support\{I_1\}} \qquad (2.3)$$

where I_1 and I_2 are two itemsets. The numerator on the right hand side represents the frequency with which I_1 and I_2 occur together in transactions in a data set, and the denominator is the frequency with which I_1 occurs in transactions.

The reader should be able to see that this formula makes sense. The left hand side is the confidence (or probability) that the presence of I_1 implies the presence of I_2. And on the right-hand side, the denominator is the number of times I_1 occurs in transactions and the numerator is the number of times the I_1 and I_2 occur together.

The full set of association rules that meet the strength criterion are then determined by generating all nonempty subsets of the itemsets identified by Apriori. This gives the full set of itemsets that meet the support criterion. These itemsets are then checked for confidence using equation (2.3).

2.3.3 Quality of Rules

Support and confidence provide the main measures for assessing the quality of an association rule. The above discussion of rule generation employs the assumption that infrequent combinations of items are not very helpful and this is generally true in practice. Thus, the Apriori algorithm seeks out only itemsets with large support (frequency). But when it comes to *confidence*, a high value does not necessarily indicate usefulness or interest. In [9] the possibility is pointed out that in a hospital database, data mining might well reveal the rule that pregnancy implies the patient is female, and that this would have confidence close to unity. It is unlikely to be exactly unity because databases always contain errors. But whether the confidence value in this case is either unity or close to it makes no difference – the rule is not at all interesting or helpful. Having knowledge and understanding about variables in a data set can obviously assist in determining whether a rule is interesting and/or useful. But for general data sets, it is difficult to take this kind of knowledge into account in the rule−discovery process.

However, for general data sets, it is possible to apply simple statistical tests to identify at least some useless rules that have misleadingly high values of support and confidence. Consider the following example from a database. Suppose that for two of the items, I_1 and I_2, the rule $I_1 \Rightarrow I_2$ has support 20% and confidence 60%. These values are sufficiently high that we would expect this to be a useful rule. But suppose now that the database indicates that the probability of purchasing item I_2 is 75%. The rule $I_1 \Rightarrow I_2$ has confidence 60%, indicating that the probability of buying I_2 decreases when I_1 is purchased. So clearly, the rule is quite useless.

To identify the problem here, note that for the rule $I_1 \Rightarrow I_2$,

support = probability of I_1 and I_2 being purchased together = $\Pr(I_1, I_2)$

confidence = probability of purchasing I_2 given that I_1 is purchased = $\Pr(I_2 \mid I_1)$

and neither of these takes proper account of $\Pr(I_2)$.

An additional measure is needed, and one that can be used is the following correlation measure

$$\text{Correlation} = [\Pr(I_1, I_2) / \Pr(I_1)\Pr(I_2)] \qquad (2.4)$$

which has a value less than unity if I_1 and I_2 are negatively correlated.

Consider again the example in the above paragraph. If the probability of purchasing I_1 is 30%, the correlation measure (2.4) has value $[0.2 / (0.3 \times 0.75)] = 0.889$, indicating negative correlation and hence that the rule is of no use.

This correlation measure is not the complete answer because it is symmetric in I_1 and I_2 and hence gives the same measure for the rule $I_1 \Rightarrow I_2$ and for $I_2 \Rightarrow I_1$. Further discussion on this and a variety of other approaches to the assessment of rule quality are given in [2], [9] and [12], including use of the χ^2 test, which can indicate whether a negative correlation is statistically significant.

2.3.4 Mining Temporal Rules

An important extension of association rule learning is in the identification of sequences of events that lead to significant outcomes such as system breakdowns. One such sequence is depicted in Figure 6 where A, B, C, D, etc., represent different events. These might, for instance be different types of signal arriving at a control room, in which case certain sequences of signal could possibly provide warnings of problems about to occur. The extension of association rule mining that we now describe is aimed at identifying such sequences.

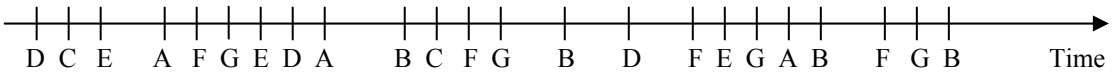


Figure 6

A collection of events within a sequence such as the one in Figure 6 is called an *episode*. Thus, "F is followed by G", which occurs several times in the figure, constitutes an episode. The concept extends to the case where other signals intervene as in the subsequence "FEG" toward the right of the figure. That is, the episode "F is followed by G" occurs in the subsequence FEG.

In order to represent situations that frequently arise in practice, episodes are *partially* ordered subsequences. Thus, the signals D and E in the figure are always very soon followed by the signal A, regardless of the order of occurrence of D and E. The episode "A follows D and E" occurs three times in Figure 6.

More formally, given a set of event types $\mathcal{E}$ an event sequence S is defined as a sequence of pairs (e, t), where $e \in \mathcal{E}$ and t is the time of occurrence of event e. Time is a discrete variable. For example, if the event D in Figure 6 occurred at time 25, the sequence depicted in the figure takes the form

$$S = (\text{D}, 25), (\text{C}, 26), (\text{E}, 27), (\text{A}, 29) \cdots$$

In mining for information in sequences, we search for episodes that occur frequently. And for these episodes to be useful, they need to occur over a short time interval. The width of the time interval has to be specified by a user, and this provides a *time window* in which any episode must occur.

Once the time window has been specified, the frequency of occurrence of an episode in a sequence can be determined. This allows computation of the support of the episode in much the same way as for association rules. We can also calculate the confidence for an episode. To see this, consider again the episode "A follows D and E" from Figure 6. If we set a time window that covers 5 time points in the figure, the first two occurrences of the episode will be identified, but the third one will not. Based upon that evidence, our confidence in the episode would be 2/3. The concepts of support and confidence (plus additional measures such as the correlation measure in (2.4) above) allow identification of useful episodes from within sequences.

The recognition of episodes in this way has a variety of potential applications, particularly in a control room environment where signals from many sources are arriving continually. One such application is described in [13] which is concerned with the management of alarm signals in a telecommunication network.

2.4 Regression

Regression and classification are quite similar mathematically. A classification system receives at its inputs a set of attributes describing a data item and, at its output produces an estimate of the class of that data item. For a two-class problem, the classes are usually represented by +1 and −1. A regression system also receives at its inputs a set of attributes describing a data item, but at its output, it produces a real number rather than an integer. This real number is an estimate of a function which could, for instance, be an estimate of electricity load at some future time (with the inputs being values of variables relevant to load forecasting).

2.4.1 Linear Regression

The simplest and best known form of regression is *linear regression*. It basically involves fitting a straight line to a set of data. An example is shown in Figure 7.

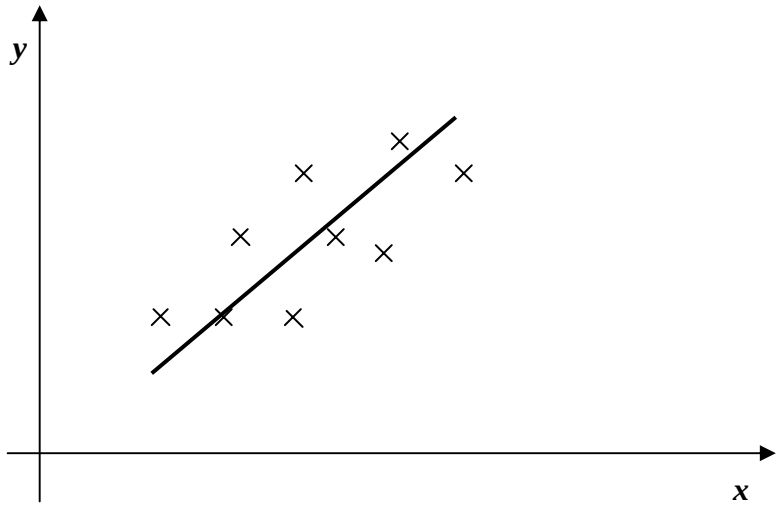


Figure 7

Figure 7 shows a regression line fitted to data defined by a single attribute, ***x***. For each data point x_i, there is a corresponding value y_i. For example, if x_i is the height of person i, and y_i is the weight of that person, a plot of (x_i, y_i) pairs would lead to a diagram something like the one in Figure 7. And we would expect a reasonably linear relationship like the one shown.

We can immediately see how a regression line could be used for prediction. For instance, we could estimate the weight of someone much taller than anyone in the data set by simply extending the regression line. And if ***x*** were a time variable, the same principle could be applied for prediction into the future. For example, the variable ***y*** might be measuring unit sales over time and the line could predict future growth.

The best way to represent a data set of the type in Figure 7 will not always be a straight line. Sometimes data will exhibit some kind of nonlinear trend and a linear regression will not be appropriate. We will discuss methods of nonlinear regression later.

Note that the linear approximation to the data in Figure 7 does not precisely match any individual data point. The differences between data points and the values predicted by

the regression line are called residuals and are usually treated as a form of noise imposing itself the linear relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. For instance, treating the variable $\boldsymbol{x}$ once again as height and the variable $\boldsymbol{y}$ as weight, we would not expect a precise linear relationship between the two because of differences in the physical make-up of the individuals measured. In most regression problems, errors of this nature occur naturally and the regression line is the one that has minimum error.

To determine the line that minimizes error, we can write, for each data pair (x_i, y_i) the equation

$$y_i = a_0 + a_1 x_i + \varepsilon_i \qquad i = 1, 2, \ldots, k \qquad (2.5)$$

where k is the number of data points and ε_i represents the error made on data point i.

To find the line that minimizes error, the least squares method is usually used. Note that the sum of squares of the errors can be written

$$E = \sum_{i=1}^{k} \varepsilon_i^2 = \sum_{i=1}^{k} (y_i - a_0 - a_1 x_i)^2 \qquad (2.6)$$

The coefficients of the regression line are the values of a_0 and a_1 that minimize the squared error in (2.6). Thus, they are obtained by simply differentiating (2.6) with respect to a_0 and a_1 and setting the two resulting equations to zero.

In general, of course, the function $\boldsymbol{y}$ is a function of more than one attribute. For the case of two attributes, Figure 7 would be modified to provide two input axes, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, as illustrated in Figure 8. In this figure, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are at right-angles. The regression line in this case is replaced by a regression *plane*.

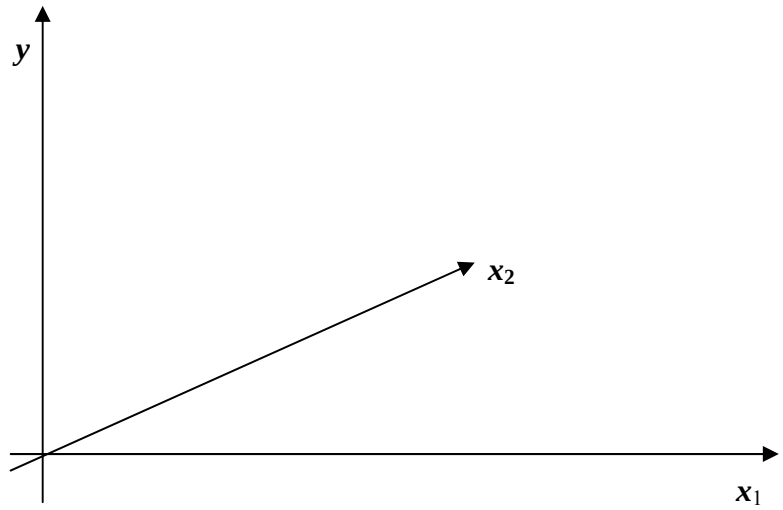


Figure 8

In the case of more than two attributes, the linear regression function is a *hyperplane*, that is, a plane in more than two dimensions which therefore cannot be represented graphically. But regardless of the number of attributes involved, the linear regression function is calculated using the same basic procedure as for the case of a single attribute. If we have m attributes, data point i is represented by a vector $[x_1^i, x_2^i, \ldots, x_m^i]$ and equation (2.5) becomes:

$$y_i = a_0 + \sum_{j=1}^{m} a_j x_j^i + \varepsilon_i \qquad i = 1, 2, \ldots, k \qquad (2.7)$$

and, if we define $x_0^i = 1$, this can be written

$$y_i = \sum_{j=0}^{m} a_j x_j^i + \varepsilon_i \qquad i = 1, 2, \ldots, k \qquad (2.8)$$

If there are n data points, the function values y_i will be given by n expressions of the form (2.8). Now note that the summation in (2.8) is equivalent to the multiplication of row vector $\boldsymbol{x}^i = [1, x_1^i, x_2^i, \ldots, x_m^i]$ by column vector $\boldsymbol{a} = [a_0, a_1, \ldots, a_m]^T$. Thus, the n function values y_i can be written as the matrix equation:

$$\boldsymbol{y} = \mathbf{X}\boldsymbol{a} + \boldsymbol{\varepsilon} \qquad (2.9)$$

where vector $\boldsymbol{a}$ is as defined above and the i^{th} row of $\mathbf{X}$ is equal to $\boldsymbol{x}^i$ defined above. $\mathbf{X}$ is an $n \times (m+1)$ matrix, $\boldsymbol{y}$ is an n-vector and $\boldsymbol{\varepsilon}$ is the vector $[\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_n]^T$.

As in (2.6), the sum of squares of errors is formed and differentiated with respect to each of the coefficients a_j. Setting the resulting equations to zero and solving for the a_j gives the values of the coefficients of the linear regression function that provide minimum error. It can be shown [14] that these regression coefficients can be calculated using the formula:

$$\boldsymbol{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y} \qquad (2.10)$$

The matrix $\mathbf{X}^T\mathbf{X}$ in (2.10) is square, a requirement for invertibility. Another requirement for invertibility is that it be nonsingular, so it is important to use a software package that checks for linear dependencies among the rows of $\mathbf{X}^T\mathbf{X}$. If such dependencies exist, one or more of the y_i in (2.9) have to be dropped from the calculation. It is also possible that some rows of $\mathbf{X}^T\mathbf{X}$ are close to linear dependence and this can lead to computational problems due to numerical instability – see [9] for more discussion on this.

2.4.2 Non-linear Regression

Although a linear regression model is often adequate for a practical problem, it frequently happens that real-world data are not well represented by a linear model. In such circumstances, one can make use of generalized additive models, which are an extension of equation (2.7) as below:

$$y_i = a_0 + \sum_{j=1}^{m} f_j(x_j^i) + \varepsilon_i \qquad i = 1, 2, \ldots, k \qquad (2.11)$$

In this equation, each f_j is an initially unspecified function and remarkably, iterative techniques exist that allow determination of those smooth functions f_j that provide the best fit to a given set of data. We will not pursue this here but refer the interested reader to [14] chapter 9. In the remainder of this section we will briefly describe two specific approaches to nonlinear regression.

2.4.2.1 Logistic Regression

The first method we will explore is widely used for estimating probabilities from data, especially probabilities of outcomes with yes/no answers. For instance, medical data on a patient may or may not indicate that the patient has a particular disease.

Logistic regression can be used to determine a probability of that outcome. We cannot use linear regression to predict probabilities because the linear approximation to the data is unbounded, while probabilities are restricted to the range [0, 1]. Logistic regression uses a transformation of the linear regression formula so that values of the regression function are restricted to the range [0, 1].

For the case of a single attribute, if we are given an attribute value x_i, the linear regression formula will give an output value of the form $y_i = a_0 + a_1 x_i$. In logistic regression, an output value such as this is transformed into the probability $\Pr(y_i = 1)$ using the formula:

$$\Pr[y_i = 1] = \frac{e^{(a_0 + a_1 x_i)}}{1 + e^{(a_0 + a_1 x_i)}} \qquad (2.12)$$

where the formula on the right hand side of (2.12) defines the so-called *logistic curve*, which has a sigmoidal shape, varying between 0 and 1 as x_i varies between $-\infty$ and ∞.

This formula extends naturally to the case of multiple attributes:

$$\Pr[y_i = 1] = \frac{e^{(a_0 + a_1 x_1^i + x_2^i + \cdots)}}{1 + e^{(a_0 + a_1 x_1^i + x_2^i + \cdots)}} \qquad (2.13)$$

To give this a practical meaning, the variable y_i might take on the value 1 when a hospital patient has a particular disease. So, once the coefficients a_j have been found, (2.13) gives the probability of patient i having the disease when his/her attributes (blood pressure, heart rate, and so on) are x_1^i, x_2^i, *etc*. By taking logarithms on both sides of (2.13) the equation can be rearranged to give:

$$\log_e\left(\frac{\Pr[y_i = 1]}{1 - \Pr[y_i = 1]}\right) = a_0 + a_1 x_1^i + a_2 x_2^i + \ldots \qquad (2.14)$$

where the right-hand side is clearly a linear function of the form employed in linear regression. But the presence of the nonlinearity on the left-hand side makes the problem of finding the coefficients a_j that give the best fit to the data rather more difficult than in the linear case.

Finding the best fit in the nonlinear case is generally achieved by some form of iterative technique. For logistic regression, the optimum coefficients a_j can be found using the classical statistical method of *maximum likelihood*. A likelihood function is constructed which involves the available data (vectors of attributes) and the coefficients a_j. The maximum of this function gives the values of the a_j that are most likely to have produced the data, and these are taken as the optimum values. Once the a_j have been determined in this way, they can be used in equation (2.13) to determine the probability

that any future patient has the disease in question. Details of the use of maximum likelihood in logistic regression are given in [14].

2.4.2.2 Feedforward Neural Networks

The feedforward neural network in Figure 9 can be trained to implement regression as well as classification. In the figure, a vector of attributes $\boldsymbol{x}$ is presented to the network inputs on the left. Neural element j in the middle of the network receives a weighted sum of the inputs equal to $\sum_i w_{ij} x_i$. This neural element implements a nonlinear function f_j so its output is equal to $f_j\left(\sum_i w_{ij} x_i\right)$. A weighted sum of the outputs of these neural elements provides the network output y, so we have:

$$y = \sum_j v_j f_j\left(\sum_i w_{ij} x_i\right) \qquad (2.15)$$

and the nonlinear nature of the f_j ensures that the network is capable of implementing nonlinear regression. Comparison with equation (2.11) indicates that the feedforward neural network is more flexible than the generalized additive model. Indeed, it has been shown that this network is capable of implementing any smooth regression function so long as there are sufficient neural elements in the middle layer.

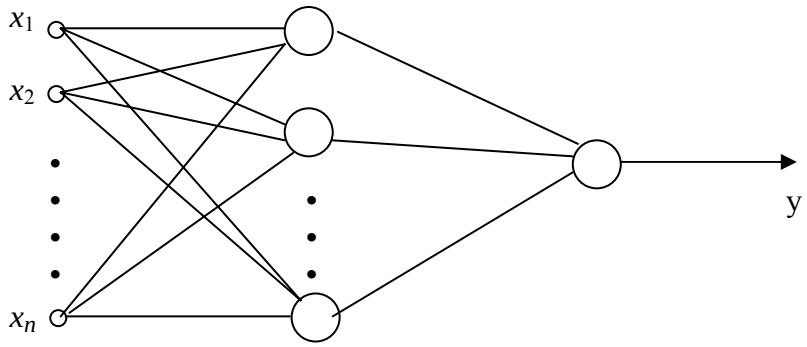


Figure 9

The feedforward neural network is trained to implement a regression function in essentially the same way as it is trained to implement classification. It is presented with a series of input vectors and the values of the weights w_{ij} and v_j are adjusted so as to move the output values toward their desired values. The mean-squared errors made by the network on each data point are summed and used as an overall error measure that allows the weights to be adjusted so that the error gradually reduces. As with the use of neural networks for classification, it is important to have an independent set of data, not used in training, to identify the point where overfitting begins to occur.

When the data used in training a network have been collected sequentially over time, they give a regression function that can be extrapolated into the future to provide predictions of future data values. And if data are available in the form of a time series, a network such as the one in Figure 9 can be trained to provide predictions directly. For one-step-ahead predictions, this is achieved as follows. The first n data points in the series are fed to the n inputs, and the desired output is set to the value of data point n+1. The error made by the network is used to adjust the weights. Next, data points 2 to n+1 are fed to the inputs and the desired output is set to data point n+2. The procedure

continues in the obvious way. Usually several passes of the data have to be made before good predictions are obtained. The network can be trained to give *k*-step-ahead predictions similarly. A network trained in this way is called a *time-delay neural network*.

3. Data Mining Applications in the Power Transmission Field

The following section takes the same format as Section 2 but focuses on applications in the power transmission field where data mining techniques have been applied. The aim is to highlight where data mining has been typically used ad provide further references for the interested reader.

3.1 Classification

3.1.1 Decision Trees

- **Transmission system planning**

  Elia, the Belgian transmission operator has been used decision trees, as data mining techniques in evaluating transmission system planning [15].

  In this case, the data mining technique has evaluated the planning options, including size and location of new power plants; growth of customer demand; growth of embedded generation; evolution of transit flows, and trend of import/export levels [15].

  The method has been implemented in the software product PEPITo [16] to analyse very large power companies databases.

- **Power quality disturbance qualification**

  Research was reported by the University of Waterloo, Ontario, Canada who used inductive learning based on decision trees to classify various power quality disturbances occurring in power systems [17]. The C4.5 machine learning algorithm is used to produce the classification in the form of if/then rules.

  For tree generation, 1400 cases are used and other 1400 examples are used for testing. The cases are generated by a set of mathematical equations and the simulation software, EMTDC/PSCAD. Different categories of disturbances were simulated including sags, swells, and harmonics.

  Results of the work are represented by a decision tree consisting of 32 rules. The classification accuracy of the decision tree reaches 91%, a figure considered far better than in cases where other classification techniques have been used.

3.1.2 Artificial Neural Networks

- Back-Propagation Neural Networks

  - **Load forecasting**

    The prediction of peak electric loads [18] in Japan by 2020 has been attempted using artificial neural networks. In this study, system load forecasting is undertaken for nine Japanese power utilities. Two Artificial Neural Networks (ANN) were designed, a three-layered backpropagation and a second recurrent neural network. Predictions were done for target years: 1999, 2000, 2005, 2010, 2015 and 2020. For testing purposes, two case studies were considered for 2010 and 2020.

    While short-term load forecasting is mainly affected by the weather conditions, long-term load forecasting is more dependent on economic factors. Consequently, the study concerns the economical data that seem to influence long-term electrical load demand. Ten factors were selected as inputs for the proposed ANN, including gross national product, population, number of households, number of air-conditioners, energy consumption and others.

  - **Automatic generation control**

    A Back Propogation Neural Network (BPNN) has been applied to the Automatic Generation Control (AGC) [19] problem, in a research project undertaken by the Istanbul Technical University. An ANN controller is applied to study the AGC problem in a four-area interconnected power system, three areas having steam turbines and the other a hydroturbine.

    The ANN controller controlled the inputs of each area in the power system in conjunction. Results show an increased performance of the ANN controller compared to the conventional types that include an integral controller as secondary controller.

- Recurrent Neural Networks

  - **Power system transient stability**

    A Recurrent Neural Network (RNN) was used to perform power system transient stability [20] in research conducted by the University of the Ryukyus, Okinawa, Japan. The on-line RNN stabilisation controller has been developed to enhance the transient stability of a power system.

    Considering the online tuning type of the model, it is able to stabilise the power system for varied system parameters and operating conditions. A multi-machine power system is considered, with three generators and six

buses. The efficiency of the controller to manage the system oscillations is demonstrated through simulations.

- Feed-Forward Neural Network

  - **Protection equipment**

    A Feed-Forward Neural Network (FFNN) was used to perform power system protection [21] as part of a research project at the Institution of Technology, Banaras Hindu University in India. An ANN based directional over-current relay was designed and developed for power system protection [21].

    The relay senses the fault current and direction of power flow, and operates when current flowing in the circuit exceeds the pre-set value. The approach uses a multi-layer feedforward neural network (FFNN) trained with the error back-propagation algorithm.

    To validate the performance of the model, the ANN-based algorithm is then tested successfully off-line using various simulation equations for overcurrent and under voltage fault conditions.

  - **Polution flashover faults forecasting**

    FFNN has been used to forecast the faults caused by polution flashovers in 15KV overhead distribution networks [22]. This research was undertaken by the University of Patras, Greece.

    The application was trained using data recorded over thirteen years from the 15 KV distribution network of Aegean island of Paros. The monthly numbers of pollution flashovers are forecast using an expert system, consisting of a FFNN and neuro-fuzzy inference system.

    Results reported show the technique was able to forecast with a one month lead time network faults caused by pollution flashovers.

- Multi Layer Perceptron Model

  - **Load forecasting**

    A Multi Layer Perceptron (MLP) model has been used to forecast real and reactive demands of electrical power networks. SNC-Lavalin Inc. of Quebec, Canada have used this technique to forecast demand one day ahead in their EMS package [23].

    A load flow application was designed, consisting of two main algorithms, one based on principles of multi-layer perceptron neural networks:

    - Similar day load forecast algorithm (SDLF)
    - Weather adaptive load forecast algorithm (WALF)

    WALF has been applied by Powerlink Queensland, one of the Australian transmission network utilities.

  - **Power flow**

    A Multi Layer Perceptron (MLP) model has also been applied to performing power flows [24] at the Federal University of Maranhao, Sao Luis in Brazil. The conventional mathematical model of the power flow consists of a set of nonlinear algebraic equations, normally solved with the Newton-Raphson method or its equivalent.

    To benefit from the superior response speed of ANN over conventional methods, MLP neural networks have been used for calculating voltage magnitudes and power factor angles. The proposed ANN methodology has been successfully evaluated using the IEEE-30 bus system.

3.1.3 Bayesian Classification

- **Line thermal overload**

Bayesian time series models have been used to perform on-line risk assessment of conductor thermal overload [25] by modeling weather conditions along transmission lines. This approach has been applied by the Iowa State University of Iowa, USA.

The model predicts the thermal overload risk for the next hour based on the current weather conditions and power system operating conditions.

- **Reliability of power systems**

Bayesian classification was used to assess the power system reliability [26] at the University of Wisconsin of Milwaukee, USA. The model developed was used to determine the function of various components in overall system reliability.

Probabilities of supply availability and load demand across the power system's locations are calculated. The algorithm can provide results addressing a number of factors, including:

- Loss of load probability
- Degree of reliability of a given supply to meet certain demand forecast
- Cause of contingency at a certain location
- What-if contingency study

The Bayesian network model used for determining loss of load probability is build by connecting sub-systems defining main components of the power system. Performance was evaluated on a test system of four generation areas, which are connected by tie lines.

3.2 Clustering

3.2.1 The k-Means Method

- **Transmission system planning**

  Elia, the Belgian transmission network service provider has applied the K-means method to aid in transmission system planning [15].

  The algorithm was successfully tested using PEPITo, an existent data mining software application [16]. This method identified various opportunities for system expansion.

- **Partial discharge in power transformers**

  K-means, back propagation neural network and induced rules from the C5.0 algorithms have been applied to monitor partial discharge in transformers [27], [28]. This research was undertaken by the University of Strathclyde, Glasgow.

  Partial discharge associated with a number of defects in oil filled transformers have been classified by the system, including: bad contacts, floating components, suspended particles, protrusions, rolling particles and surface discharges.

  A combination of data mining techniques was employed as their experience has shown no one technique could perform all the monitoring tasks required, resulting in a hybrid system. Similar techniques are now being employed to help ScottishPower to analyze trip coil currents from distribution switchgear.

3.2.3 Kohonen Self-Organising Feature Maps

- **Chemical ageing indicators of polymeric (EPDM) line insulators.**

  Self Organising Feature Maps (SOFM) were used to perform cluster analysis of the chemical indicators of naturally aged ethylene propylene diene terpolymers (EPDM) insulators [29]. Research was conducted at the Queensland University of Technology (QUT), Australia on samples of insulator material.

  The technique was able to distinguish between insulators from different locations, different manufacturers and insulators with varying degrees of degradation.

  It was concluded that cluster analysis [29] provided a useful method for providing a condition indicator for composite polymer insulators.

- **Monitoring of transformer OLTC**

  SOFM were used to analyse acoustic signatures of transformer on load tap-changers (OLTCs) [30]. The model was developed by the Research Concentration in Electrical Energy of Queensland at QUT, Australia. (Currently the Research Concentration no longer exist).

  Vibration monitoring of OLTCs generate large amounts of operation signatures. It was noticed that each tap change operation generated a signal that contained information on the condition of the tap changer contacts and the drive mechanisms.

  Signals for every tap position were found to be different, and there was significant inconsistence regarding normal condition signals for the same tap position. Kohonen Self Organizing Maps methods have been applied to the data to determine abnormal conditions and to quantify their nature.

- **Dissolved gas data from transformers**

  Kohonen SOFM methods were used to perform gas data analysis (DGA) [31]. The National Grid Company, UK applied this data mining technique to analyse its dissolved gas analysis results for it power transformers. The aim of this technique was to analyse the existing data and determine trends which are not initially obvious to the normal techniques of dissolved gas data analysis.

  Results of this approach display a useful map including each of the monitored gases: H_2, CO, $C0_2$, CH_4, C_2H_4, C_2H_6 and C_2H_2.

- **Power system security**

  SOFM have been used to estimate power system security [32]. Power system network topology frequently changes with increased demand and network

augmentations. When load increases, the power system network is more susceptible in the event of a disturbance.

A Kohonen SOFM was used to estimate the security status of a 6-bus power system. Results demonstrate the feasibility of classification of load patterns for power system static security assessment. The proposed network classifies the unknown loading patterns by use of 15 different line loading patterns for training. Further research is required to consider contingency analysis in the security assessment program.

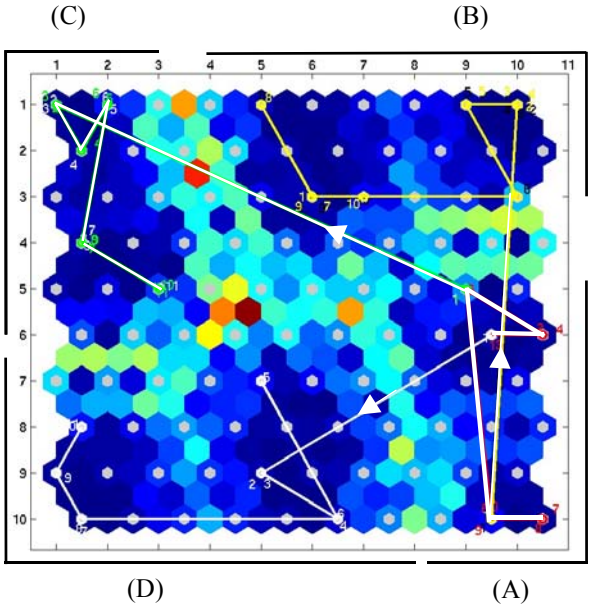


Figure 10

Self Organizing Feature Map (SOFM) for a condition monitoring system. The arrows show transitions from normal (A) to degraded conditions (B,C,D in darker colours).

3.4 Regression

3.4.2 Non-linear Regression

- **Load forecasting**

Local polynomial regression has been used for short-term load forecasting [33] in power systems at the University of South Africa.

The goal was to approximate functional dependence between load and its affecting factors. First the data mining algorithm selects historical load sequences, that satisfy the necessary known factors required by the load model. Then, a series of local regression methods are applied to the selected sequences to estimate the load forecast.

References

[1] A. Berson and S. J. Smith, "*Data warehousing, data mining and OLAP"*, McGraw-Hill, 1997.

[2] J. Han and M. Kamber, "*Data mining, concepts and techniques"*, Morgan Kaufmann, 2001.

[3] V. N. Vapnik, "*Statistical learning theory"*, Wiley, 1998.

[4] J. R. Quinlan, "*C4.5: Programs for machine learning"*, Morgan Kaufmann, 1993.

[5] I. H. Witten and E. Frank, "*Data mining"*, Morgan Kaufmann, 2000.

[6] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "*Learning representations by backpropagating errors"*, Nature, 323, pp. 533-536, 1986.

[7] J. S. Bridle, *"Probabilistic interpretation of feedforward classification outputs, with relationships to statistical pattern recognition"*, Neuro-computing: Algorithms, Architectures and Applications, F Fogleman Soulie and J Herault (Eds), pp. 227-236, 1990.

[8] A. J. McGrail, E. Gulski, E. R. S. Groot, D. Allan, D. Birtwhistle and T. R. Blackburn, *"Data mining techniques to assess the condition oh high voltage electrical plant"*, CIGRE Working Group 15.11, Paper 15.107, Paris 2002.

[9] D. Hand, H. Mannila and P. Smyth, "*Principles of data mining"*, MIT Press, 2001.

[10] T. Kohonen, *"Self-organized formation of topologically correct feature maps*", Biological Cybernetics, 43, pp. 59-69, 1982.

[11] T. Kohonen, *"The neural phonetic typewriter"*, Computer, 21(3), pp. 11-22, 1988.

[12] M. H. Dunham, "*Data mining, introductory and advanced topics"*, Pearson Education, 2003.

[13] K. Hatonen, M. Klemettenen, H. Mannila, P. Ronkainen and H. Toivonen, *"Knowledge discovery from telecommunication network alarm databases*", Proceedings 12th International Conference on Data Engineering, New Orleans, pp. 115-122, 1996.

[14] T. Hastie, R. Tibshirani and J. Friedman, "*The elements of statistical learning"*, Springer, 2001.

[15] S. Vassena et al., *"A probabilistic approach to power systems network planning under uncertainties"*, IEEE Bologna Power Tech Conference Italy, 2003.

[16] Pepite S.A.*, "PEPITo™ software application"*, Belgium, 2003.

[17] T. K. Abdel-Galil, *"Power quality disturbance classification using the inductive inference approach"*, IEEE Transactions on Power Delivery, Issue 99, pp. 1-7, 2004.

[18] B. Kermanshahi and H. Iwamiya, *"Up to year 2020 load forecasting using neural nets"*, International Journal of Electrical Power Energy Systems, Vol. 24, No. 9, pp. 789-797, January 2002.

[19] H. L. Zeynelgil, A. Demiroren and N. S. Sengor, *"The application of ANN technique to automatic generation control for multi-area power system"*, International Journal of Electrical Power Energy Systems, Vol. 24, No. 5, pp. 345-354, June 2002.

[20] T. Senjyu, M. Yoshiteru and K. Uezato, "*Online learning recurrent neural network stabilisation controller for multi-machine power system"*, PowerCon, International Conference on Power System Technology. Proceedings (Cat . No. 00EX409), Perth, Australia, pp. 223-228, 4-7 December 2000.

[21] D. N. Vishwakarma and Z. Moravej, *"ANN-based directional overcurrent relay"*, IEEE/PES Transmission+ and Distribution Conference and Exhibition (Cast No. 01CH37294), Vol. 1, pp. 59-64, 28 October – 2 November 2001.

[22] A. D. Tsanakas, G. I. Papaefthimiou, and D. P. Agoris, *"Polution flashover fault analysis and forecasting using neural networks"*, CIGRE Working Group 15.105, Session 2002.

[23] *"Load forecasting application software design document"*, Energy management system (EMS) software, SNC-Lavalin Inc Canada, 1997.
[24] V. L. Paucar and M. J. Rider, *"Artificial neural networks for solving the power flow problem in the electric power systems"*, Electric Power Systems Research, Vol. 62, No. 2, pp. 139-144, June 2002.
[25] J. Zhang, J. Pu, J. D. McCalley, H. S. Stern, and W. A. Gallus, *"A Bayesian approach for short-term transmission line thermal overload risk assessment"*, IEEE Transactions on Power Delivery, Vol. 17, No. 3, pp. 770-778, July 2002.
[26] D. C. Yu, T. C. Nguyen and P. Haddawy, *Bayesian network model for reliability assessment of power systems",* IEEE Transactions on Power Systems, Vol. 14, No. 2, pp. 426-432, May 1999.
[27] S. M. Strachan, G. Jahn, S. McArthur and J. R. McDonald, *"Intelligent diagnosis of defects responsible for partial discharge activity detected in power transformers",* Intelligent System Applications in Power Systems Conference 2003 (ISAP), Paper ISAP 03/109, 2003.
[28] S. D. J. McArthur, S. M. Strachan and G. Jahn*, "Design of a multi-agent transformer condition monitoring system"*, IEEE Transactions on Power Systems, Paper accepted, 2004.
[29] A. Krivda, G. Cash, D. Birtwhistle and G. George, *"Diagnostics of EPDM polymer insulators"*, Conference on Electrical Insulation and Dielectric Phenomena, Austin, Texas USA, October 1999.
[30] P. Kang and D. Birtwhistle, *"Condition monitoring of on-load tap-changers. Part 1&2"*, IEE Proceedings on Generation, Transmission and Distribution, Vol. 148, (4), pp. 301-306, and pp 307-311, July 2001.
[31] D. G. Esp and A. J. McGrail, *"Advances in data mining for dissolved gas analysis"*, ISEI Conference, Anaheim, California USA, 2000.
[32] K. S. Swarup and P. B. Corthis, *"ANN approach assesses system security"*, IEEE Computer Applications in Power, Vol. 15, No. 3, pp. 32-38, July 2002.
[33] R. Zivanovic, *"Local regression-based short-term load forecasting"*, Journal of Intelligent and Robotic Systems, Kluwer Academic Publishers, Vol. 31, No. 1-3, pp. 115-127, May 2001.

Appendix A1

Classification - Bayesian

Let us assume that we have a pattern classification problem in which each example is a vector of attributes A_i, $i = 1,2, \ldots ,k$ so our vector $\boldsymbol{x}$ has attribute values x_1 through to x_k. If we can calculate all relevant probabilities, the optimal estimate (OE) of class membership for $\boldsymbol{x}$ is then the class C_i for which the probability

$$\text{OE} = \Pr(C_i \mid A_1 = x_1 \textbf{ AND } A_2 = x_2 \textbf{ AND } \cdots \textbf{ AND } A_k = x_k) \tag{A1.1}$$

has the largest value. This is the probability that the class is C_i given the attribute values x_1 through to x_k. It is just a more explicit form of the left-hand side of equation (2.1) which tells us that the optimal estimate must be equal to the right-hand side. That is:

$$\text{OE} = \Pr(C_i)\frac{\Pr(A_1 = x_1 \textbf{ AND } A_2 = x_2 \textbf{ AND } \cdots \textbf{ AND } A_k = x_k \mid C_i)}{\Pr(A_1 = x_1 \textbf{ AND } A_2 = x_2 \textbf{ AND } \cdots \textbf{ AND } A_k = x_k)} \tag{A1.2}$$

and, for classification purposes, the denominator can be ignored because it is the same for each class. And so, we have a function, based upon Bayes' rule, that we can use to discriminate between the different classes. This kind of function is called a *discriminant function* by statisticians, and in this case takes the form:

$$f_i(\boldsymbol{x}) = \Pr(C_i)\Pr(A_1 = x_1 \textbf{ AND } A_2 = x_2 \textbf{ AND } \cdots \textbf{ AND } A_k = x_k \mid C_i) \tag{A1.3}$$

and the pattern $\boldsymbol{x}$ is assigned to the class for which this discriminant function takes on the largest value.

The major simplifying assumption that is made is to assume that for each class C_i, the attributes A_j are statistically independent, so that (A1.3) becomes:

$$f_i(\boldsymbol{x}) = \Pr(C_i)\prod_{j=1}^{k}\Pr(A_j = x_j \mid C_i) \tag{A1.4}$$

Now, the terrific thing about this equation is that its probabilities can be estimated from training data in an extremely simple way. All we have to do is count the number of times each class occurs and the number of times each feature occurs for a particular class. The probability $\Pr(C_i)$ is simply the proportion of times class C_i occurs in the training data and the factors under the product sign are estimated using the formula:

$$\hat{\Pr}(A_j = x_j \mid \text{Class} = C_i) = \frac{\#(A_j = x_j \textbf{ AND } \text{Class} = C_i)}{\#(\text{Class} = C_i)} \tag{A1.5}$$

where $\#(\cdot)$ is a count of the number of occurrences of the term in brackets.

Most importantly, if there are missing values, this has very little effect on the estimates. You simply do your counts on the basis of the data that you have – a vector with the value of one feature missing makes no contribution to the estimate of that particular feature but contributes to the estimates of all the others.

Appendix A2

Clustering – Mixture Models

As a simple example, we might assume that the situation is similar to the one in Section 2.2.3 with the data being generated by a mixture of two distributions. Let us refer to these distributions as distribution 1 and 2 and write their probability density functions in the form $f_1(x; \theta_1)$ and $f_2(x; \theta_2)$. Here θ_i is the set of parameters of density function f_i – for instance, if the distributions were Gaussian, the set of parameters would be the mean and standard deviation. The Gaussian distribution is also widely known as the *normal* distribution.

Having assumed that the data are generated in this way, the probability density function for the data item x takes the form:

$$f(x) = p_1 f_1(x; \theta_1) + p_2 f_2(x; \theta_2) \qquad \text{(A2.1)}$$

where p_i is the probability that a data item is generated by distribution i, ($i = 1, 2$).

Since we are assuming two clusters here, each data item must be generated by one of the two distributions. So, with p_1 equal to the probability that a data item is generated by distribution 1, the probability that it is generated by distribution 2 must be equal to $1 - p_1$. Thus, we must have $p_2 = 1 - p_1$. So, the parameters we need to estimate in (A2.1) are θ_1, θ_2 and p_1. But there are no labels, so we don't know which distribution any data object belongs to. So how can we go about the estimation?

This is where a rather magical method, called the *Expectation-Maximization* (*EM*) *algorithm*, comes in. Usually it involves making an assumption about the form of the distributions making up the clusters and for simplicity here, let us assume that the mixture in (A2.1) consists of two Gaussians. This means that we have 5 parameters to estimate, namely the mean and standard deviation of distributions 1 and 2 plus the probability p_1. Now, we have no knowledge about these parameter values, so we simply make an initial guess. The parameter values we have guessed can then be used to estimate, for each data object, the probability that it was generated by distribution 1 or distribution 2. This is done by using Bayes' rule in the form

$$\Pr(C_i \mid x) = \frac{f(x \mid C_i)\Pr(C_i)}{f(x)} \qquad \text{(A2.2)}$$

which is a slight variant of (2.1) to account for the fact that the data objects x are continuously distributed.

In (A2.2) the C_i are the clusters defined by the two Gaussians with $\Pr(C_1) = p_1$ and $\Pr(C_2) = p_2$ in (A2.1). Also, $f(x)$ is the density function defined in (A2.1) and $f(x \mid C_i) = f_i(\mathbf{x}; \theta_i)$. Thus, (A2.2) can be written

$$\Pr(C_i \mid x) = \frac{f_i(x; \theta_i) p_i}{f(x)} \qquad i = 1, 2 \qquad \text{(A2.3)}$$

So, having guessed the values of the parameters θ_i and p_i, equation (A2.3) allows us to estimate (using these guesses) the probability that a data object $\boldsymbol{x}$ comes from either cluster C_1 or cluster C_2. We then estimate this probability for each and every data object.

Once we have these probabilities, we can use them to estimate the parameters of the two distributions. It can be proved that this will improve our original guesses.

Our new estimate for p_i is given by

$$\hat{p}_i = \frac{1}{n}\sum_{r=1}^{n}\hat{Pr}(C_i \mid x_r) \qquad \text{(A2.4)}$$

where n is the number of data objects and x_r is the r^{th} one.

We must now obtain new estimates for the mean and standard deviation of the two Gaussians. If we knew precisely which cluster each data object came from, the mean would be given by the simple average:

$$\hat{\mu}_i = \frac{1}{n_i}(x_1^i + x_2^i + \cdots + x_{n_i}^i) \qquad \text{(A2.5)}$$

where n_i is the number of data objects in cluster i and x_r^i is one of them. But since there are no labels, we don't know the cluster each object comes from, so we have to calculate the average in terms of the probability that a data object belongs to a particular cluster. Thus, in the case where we have no labels, (A2.5) becomes:

$$\hat{\mu}_i = \frac{1}{n\hat{p}_i}\sum_{r=1}^{n}\hat{\Pr}(C_i \mid x_r)x_r \qquad \text{(A2.6)}$$

It is important to observe that if we knew the labels, equation (A2.6) would reduce to (A2.5). This is because p_i would be a known quantity - it would equal the proportion of data objects in cluster i, so that $np_i = n_i$. And if x_r belongs to cluster 1 we would have $\Pr(C_i \mid x_r) = 1$ and otherwise $\Pr(C_i \mid x_r) = 0$. Substituting these values into (A2.6) gives (A2.5). Thus, (A2.6) gives us our best estimate of μ_i, $i = 1, 2$.

Our estimate of the standard deviation is computed similarly:

$$\hat{\sigma}_i^2 = \frac{1}{n\hat{p}_i}\sum_{r=1}^{n}\hat{\Pr}(C_i \mid x_r)(x_r - \hat{\mu}_i)^2 \qquad \text{(A2.7)}$$

Once these new estimates have been obtained, they are substituted back into (A2.3) to update the estimates of which cluster each object belongs to. And these new estimates are then substituted into (A2.4), (A2.6) and (A2.7) to improve the parameter estimates. This process is repeated until convergence.