General Information

Word count approximately 7000

7 Figures and 3 Tables

Figure 15.1	colour
Figure 15.2	black and white
Figure 15.3	colour
Figure 15.4	black and white
Figure 15.5	colour
Figure 15.6	black and white
Figure 15.7	black and white

Table 15.1	black and white
Table 15.2	black and white
Table 15.3	black and white

Contents

Chapter 15

Mining Gene Expression Data

15

Mining Gene Expression Data

Xiaohui Liu (xiaohui.liu@brunel.ac.uk)
Paul Kellam (p.kellam@ucl.ac.uk)

- Key issues in data mining are discussed in the context of modern developments in computing and statistics
- Data mining methods that have been commonly-used in gene expression analysis are presented with simple examples
- Current and future research issues are outlined

DNA microarray technology has enabled biologists to study all the genes within an entire organism to obtain a global view of gene interaction and regulation. This technology has great potential in obtaining a deep understanding of biological processes. However, to realise such potential requires the use of advanced computational methods. In the last chapter, experimental protocols and technical challenges associated with the collection of gene expression data were presented, the nature of the data was highlighted, and the data standards for storing expression data were discussed. In this chapter, we shall introduce some of the most common data mining methods that are being applied to the analysis of microarray data and discuss the likely future directions.

15.1. Data Mining and Intelligent Data Analysis

Data mining has been defined as the process of discovering knowledge or patterns hidden in (often large) datasets. The data mining process is often semi-automatic and interactive, but there is a great desire for it to be automatic. The knowledge or patterns discovered should be meaningful and interesting from a particular point of view. For example, loyalty patterns of previous customers should allow a company to detect among the existing customers those likely to defect to its competitors. In the context of DNA microarray data, the patterns detected should inform us of the likely functions of genes, how genes may be regulated, and how they interact with each other in health and disease process. A comprehensive treatment of various issues involved in building data mining systems can be found in [Hand2001].

15.1.1 Data Analysis

The job of a data analyst typically involves problem formulation, advice on data collection (though it is not uncommon for the analyst to be asked to analyse data which have already been collected, especially in the context of data mining), effective data analysis, and interpretation and reporting of the findings. Data analysis is about the extraction of useful information from data and is often performed by an iterative process in which “exploratory analysis” and “confirmatory analysis” are the two principal components.

Exploratory data analysis, or data exploration, resembles the job of a detective: understanding the evidence collected, looking for clues, applying relevant background knowledge and pursuing and checking the possibilities that clues suggest. This initial phase of data examination typically involves and number of steps, namely:

Data quality checking, processes for determining the levels and types of noise (random variation) in the system, processes for dealing with extreme outlying values and missing values.

Data modification, processes for transforming data values (such as the $\log_2$ transformation seen in chapter 14 and processes for error correction.

Data summary, defined methods for producing tables and visualistion of summary statistic for the data.

Data dimensionality reduction, methods for reducing the high dimensionality of the data to allow patterns to be identified or other data mining tools to be used.

Feature selection and extraction methods that allow defining features of the data to be identified and significance values applied to such features.

Clustering methods, such as hierarchical clustering, which identify potential structures in the data the share certain properties.

Data exploration is not only useful for data understanding, but also helpful in generating possibly interesting hypotheses for further investigation, a more “confirmatory” procedure for analysing data. Such procedures often assume a potential model structure for the data, and may involve estimating the model parameters and testing hypotheses about the model.

15.1.2 New Challenges and Opportunities

Over the last 15 years, we have witnessed two phenomena which have affected the work of modern data analysts more than any others. Firstly, the size of machine-readable data sets has increased and the problem of "data explosion" has become apparent. Many analysts no longer have the luxury of focusing on problems with a manageable number of variables and cases (say a dozen variables and a few thousand cases); and problems involving thousands of variables and millions of cases are quite common, as is evidenced in biology throughout this book.

Secondly, data analysts, armed with traditional statistical packages for data exploration, model building, and hypothesis testing, now require more advanced analysis capabilities. Computational methods for extracting information from large quantities of data, or "data mining" methods, are maturing, e.g. artificial neural networks, Bayesian networks, decision trees, genetic algorithms, statistical pattern recognition, support vector machines, and visualisation. This combined with improvements in computing such as increased computer processing power and larger data storage device capacity both at cheaper costs, coupled with networks providing more bandwidth and higher reliability, and On-Line Analytic Processing (OLAP) all allow vastly improved analysis capabilities.

These improvements are timely as functional genomics data provides us with a problem where traditional mathematical or computational methods are not really suited for modelling gene expression data: such data are noisy and high-dimensional with up to thousands of variables (genes) but a very limited number of observations (experiments).

In this chapter we will present some of the most commonly used methods for gene expression data exploration, including hierarchical clustering, K-means, and self-organising maps (SOM). Also, support vector machines (SVM) have become popular for classifying expression data, and we will describe the basic concepts of SVM.

15.2. Data Mining Methods for Gene Expression Analysis

To date two main categories of data mining methods have been used to analyse gene expression data: clustering and classification. Clustering is about the organisation of a collection of unlabeled patterns (data vectors) into clusters based on similarity, so that patterns within the same cluster are more similar to each other than they are to a pattern belonging to a different cluster. It is important for exploratory data analysis since it will often reveal interesting structures of the data, thereby allowing formulation of useful hypotheses to test. In the context of gene expression data analysis, clustering methods have been used to find clusters of co-expressed/co-regulated genes which can be used to distinguish between diseases that a standard pathology might find it difficult to tell apart as reported by Alizadeh and colleagues in Nature (2000).

As early DNA microarray experiments have shown that genes of similar function yield similar expression patterns, work on the use of *classification* techniques to assign genes to functions of a known class has increased. Instead of learning functional classifications of genes in an unsupervised fashion like clustering, classification techniques start with a collection of labelled (pre-classified) expression patterns; the goal being to learn a classification model that will be able to classify a new expression pattern. Classification has also been extensively used to distinguish (classify) different samples, for example, breast cancer samples into distinct groups based on their gene expression patterns. We will now explore these methods in more detail.

15.3. Clustering

Given a gene expression matrix, clustering methods are often used as the first stage in data analysis. The goal of clustering here is to group sets of genes together that share the similar gene expression pattern across a dataset.

There are three essential steps in a typical pattern clustering activity:

1) *Pattern representation* refers to things such as the number of available patterns, the number of features available to the clustering algorithm, and for some clustering methods the desired number of clusters. Suppose we have a gene expression data set with 1000 genes, whose expression levels have been measured for 10 time points or 10 different samples. In this case, we would have 1000 available patterns and 10 features available for each pattern. One clustering aim would therefore be to see how we could group these 1000 patterns using these 10 features into a number of *natural* clusters.

2) *Pattern proximity* is usually measured by a distance (dissimilarity) matric or a similarity function defined on pairs of patterns. The Euclidean distance is probably the most commonly used dissimilarity measure in the context of gene expression profiling. On the other hand, a similarity measure such as such as Pearson's correlation or Spearman's Rank Correlation is also used. For a variety of commonly used proximity measures see Box 1. Other distance metrics such as Mahalanobis Distance, Chebyshev distance and Canberra metric are described in [Webb 99].

3) *Pattern grouping* or clustering methods may be grouped into the following two categories: hierarchical and non-hierarchical clustering [Jain1999]. A hierarchical clustering procedure involves the construction of a hierarchy or tree-like structure, which is basically a nested sequence of partitions, while non-hierarchical or partitional procedures end up with a particular number of clusters at a single step. Commonly used non-hierarchical clustering algorithms include the K-means algorithm, graph-theoretic approaches via the use of minimum spanning trees, evolutionary clustering algorithms, simulated annealing based methods as well as competitive neural networks such as Self-Organising Maps.

Box 1: Measures of dissimilarity (distance) between two individual patterns

A gene expression pattern, **x**, is represented by a vector of measurements [x_1, x_2,…, x_p] (expression levels at *p* different time points). Many of the clustering techniques require some measure of dissimilarity or distance between **x** and **y**, the two *p*-dimensional pattern vectors, so it is useful to summarise several commonly used distances.

Euclidean distance

$$d_e = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

Euclidean distance appears to be the most commonly used measure for a variety of applications, including gene expression profiling. Nevertheless it does have the undesirable property of giving greater emphasis to larger differences on a single variable.

City-Block or Manhattan distance

$$d_{cb} = \sum_{i=1}^{p} | x_i - y_i |$$

This metric uses a distance measure that would be suitable for finding the distances between points in a city consisting of a grid of intersecting thoroughfares (hence the name). It is a little cheaper computationally than Euclidean distance, so it may be a good candidate for applications demanding speedy solutions.

Minkowski distance

$$d_m = \left(\sum_{i=1}^{p} | x_i - y_i |^m \right)^{1/m}$$

The Minkowski distance is a general form of the Euclidean and city block distances: the Minkowski distance of the first order is the city-block distance, while the Minkowski distance of the second order is the same as the Euclidean distance. The choice of *m* depends on the amount of emphasis placed on the larger differences $| x_i - y_i |$

To explain how clustering methods work we will use an artificial dataset of simple numbers represented as a distance matrix (Table 15.1). In addition, we will use a simple eight gene and six sample $\log_2$ ratio gene expression matrix as described in Chapter 14 (Fig 15.1a) to show how clustering methods may work on DNA array data. This gene expression matrix contains three easily identifiable gene expression patterns relative to the reference mRNA source namely: i) non changing, ii) over-expressed to under-expressed and iii) under-expressed to over-expressed (Fig15.1b).

	1	2	3	4	5
1	0	1	5	9	8
2		0	4	8	7
3			0	3	4
4				0	2
5					0

Table 15.1. A dissimilarity matrix for five individuals

			Arrays			
Genes	I	II	III	IV	V	VI
1◆	-3	-3	-1	0	2	3
2■	-2	-2	0	1	2	2
3▲	-3	-2	0	1	2	3
4×	3	2	0	-1	-2	-3
5●	2	2	1	0	-2	-3
6●	3	2	1	0	-2	-3
7+	2	2	2	2	2	2
8–	-2	-2	-2	-2	-2	-2

Figure 15.1a. Gene names are in column 1 (gene 1 to 8) and each gene is colour coded per row, with respect to the figure 15.1b. Arrays (I to VI) are in columns (2-7) with each feature being a gene expression $\log_2$ ratio value (see chapter 14)

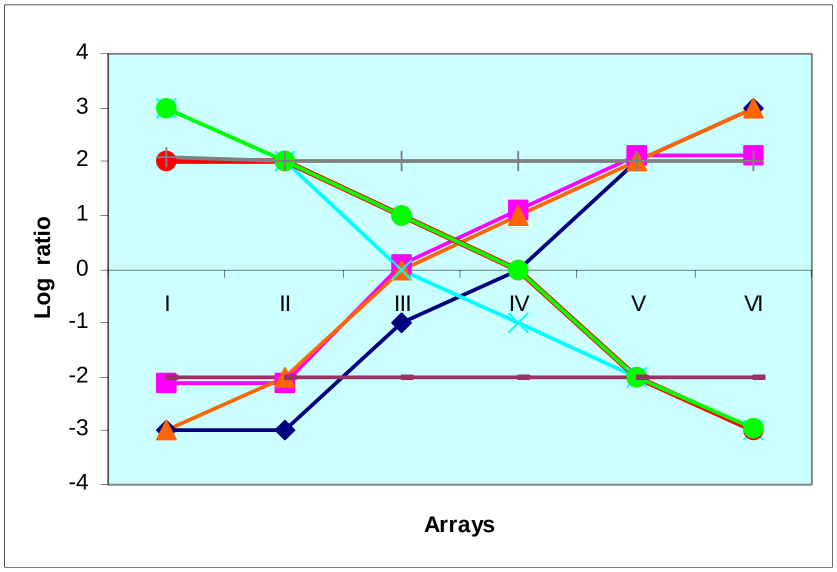

Figure 15.1b. A graphical representation of the data for 8 genes measured across 6 arrays corresponding the gene matrix in Figure 15.1a.

15.3.1. Hierarchical Clustering

The concept of the hierarchical representation of a data set was primarily developed in biology. Therefore it is no surprise that many biologists have found it easier to use hierarchical methods to cluster gene expression data. Basically a hierarchical algorithm will produce a hierarchical tree or *dendrogram* representing a nested set of partitions. Sectioning the tree at a particular level leads to a partition with a number of disjoint clusters. A numerical value, essentially a measure of the distance between two merged clusters, is associated with each position up the tree where branches join.

There are two main classes of algorithm for producing a hierarchical tree. An *agglomerative* algorithm begins with placing each of the *n* available gene expression values across a set of arrays (patterns, also known as the gene expression vector) into an individual cluster. The algorithm proceeds to merge the two most similar groups to form a new cluster, thereby reducing the number of clusters by one. The algorithm continues until all the data fall within a single cluster. A *divisive* algorithm works the other way around: it operates by successively dividing groups beginning with a single group containing all the patterns, and continuing until there are *n* groups, each of a single individual. Generally speaking, divisive algorithms are computationally less efficient.

A typical hierarchical agglomerative clustering algorithm is outlined below:

1) Place each pattern (gene expression vector) in a separate cluster
2) Compute the proximity matrix of all the inter-pattern distances for all distinct pairs of patterns
3) Find the most similar pair of clusters using the matrix. Merge these two clusters into one, decrement number of clusters by one and update the proximity matrix to reflect this merge operation.
4) If all patterns are in one cluster, stop. Otherwise, go to the above step 2.

The output of such algorithm is a nested hierarchy of trees that can be cut at a desired dissimilarity level forming a partition. Hierarchical agglomerative clustering algorithms differ primarily in the way they measure the distance or similarity of two clusters where a cluster may consist of only a single object at a time. The most commonly used inter-cluster measures are:

$$d_{AB} = \min_{\substack{i \in A \\ j \in B}} (d_{ij}) \tag{15.1}$$

$$d_{AB} = \max_{\substack{i \in A \\ j \in B}} (d_{ij}) \tag{15.2}$$

$$d_{AB} = \frac{1}{n_A n_B} \sum_{i \in A} \sum_{j \in B} d_{ij} \tag{15.3}$$

Where d_{AB} is the dissimilarity between two clusters A and B, d_{ij} is the dissimilarity between two individual patterns *i* and *j*, n_A and n_B are the number of individuals in clusters A and B respectively. These three inter-cluster dissimilarity measures are the basis of the three of the most popular hierarchical clustering algorithms. The *single-linkage* algorithm uses Equation 15.1: the *minimum* of the distances between all pairs of patterns drawn from the two clusters (one pattern from each cluster). The *complete-linkage* algorithm uses Equation 15.2: the *maximum* of all pairwise distances between patterns in the two clusters. The *group-average* algorithm uses Equation 15.3: the average of the distances between all pairs of individuals that are made up of one individual from each cluster. To see how these algorithms work, we shall apply them to the artificial data set first, and then to the illustrative gene expression data set.

Single Linkage Clustering

From Table 15.1 it is clear that the closest two clusters (which contain a single object each at this stage) are those containing the individuals 1 and 2. These are merged to form a new cluster {1,2}. The distances between this new cluster and the three remaining clusters are calculated according to equation 15.1 as follows:

$$d_{(12)3} = \min(d_{13}, d_{23}) = 4$$
$$d_{(12)4} = \min(d_{14}, d_{24}) = 8$$
$$d_{(12)5} = \min(d_{15}, d_{25}) = 7$$

This gives the new dissimilarity matrix:

	{1,2}	**3**	**4**	**5**
{1,2}	0	4	8	7
3		0	3	4
4			0	2
5				0

The closest two clusters are now those containing individuals 4 and 5, so these are merged to form a new cluster {4,5}. We now have three clusters: {1,2}, 3, {4,5}. The distances between the new cluster and the other two clusters are then calculated:

$$d_{(45)3} = \min(d_{43}, d_{53}) = 3$$
$$d_{(45)(12)} = \min(d_{14}, d_{24}, d_{15}, d_{25}) = 7$$

These lead to the following new dissimilarity matrix:

	{1,2}	**3**	**{4,5}**
{1,2}	0	4	7
3		0	3
{4,5}			0

The closest two clusters are those containing 3 and {4,5}, so these are merged to form a new cluster {3,4,5}. The distance between the new cluster and the other remaining cluster {1,2} is calculated to be 4, which results in the following new dissimilarity matrix:

	{1,2}	{3,4,5}
{1,2}	0	4
{3,4,5}		0

Finally, merge of the two clusters at this stage will form a single group containing all five individuals. The dendrogram illustrating this series of mergers is given in Figure 15.2a

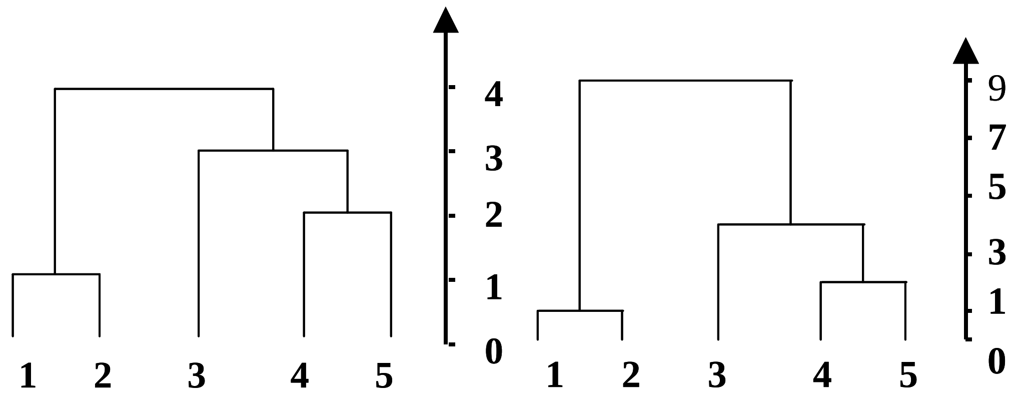

Figure 15.2: Dendrograms from single-linkage (a) and complete-linkage clustering (b)

Complete Linkage Clustering

As with single linkage this clustering approach begins by merging clusters containing individuals 1 and 2 into a new cluster {1,2}. The dissimilarities between this cluster and the three remaining clusters are calculated according to equation 15.2 as follows:

$$d_{(12)3} = \max(d_{13}, d_{23}) = 5$$

$$d_{(12)4} = \max(d_{14}, d_{24}) = 9$$

$$d_{(12)5} = \max(d_{15}, d_{25}) = 8$$

This gives the different dissimilarity matrix:

	{1,2}	3	4	5
{1,2}	0	5	9	8
3		0	3	4
4			0	2
5				0

The similar process takes place as described above for the single linkage algorithm. The final complete-link dendrogram is shown in Figure 15.2b. In this example the tree structure is the same regardless of the linkage method used. The readers are encouraged to apply the group-average algorithm to this data set and work out the corresponding dendrogram.

Clustering gene expression data

Here we would like to show how clustering algorithms such as single-linkage may be applied to the gene expression data set as described in Figures 15.1a and 15.1b. First, we calculate the Euclidean distances between each gene pairs using the gene expression profiles in Figure 15.1a as follows:

Gene Pair	Array I	Array II	Array III	Array IV	Array V	Array VI	Σ of values	√ of Σ = distance (d_e)
(gene 1-2)^2	1	1	1	1	0	1	5	2.36
(gene 2-3)^2	1	0	0	0	0	1	2	1.41
(gene 3-4)^2	36	16	0	4	16	36	108	10.39
•								
•								
(gene 1-3)^2	0	1	1	1	0	0	3	1.73
(gene 2-4)^2	25	16	0	4	16	25	88	9.27
•								
•								
(gene 1-7)^2	25	25	9	4	0	1	64	8
(gene 2-8)^2	0	0	4	9	16	16	45	6.7
(gene 1-8)^2	1	1	1	4	16	48	48	6.93

The corresponding distance matrix is then shown in Table 15.2.

	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
Gene 1	0	2.36	1.73	10.72	10.29	10.81	8.00	6.92
Gene 2		0	1.41	9.27	8.66	9.16	6.08	6.70
Gene 3			0	10.39	9.75	10.30	6.86	7.41
Gene 4				0	1.73	1.41	7.41	6.86
Gene 5					0	1.00	6.86	6.78
Gene 6						0	6.86	7.42
Gene 7							0	2.72
Gene 8								0

Table 15.2. Gene expression distance metrix

Now let us see how the single-linkage method may be applied to this case. The closest two clusters (which contain a single object each at this stage) are those containing genes 5 and 6. These are merged to form a new cluster {5, 6}. The distances between this new cluster and the six remaining clusters are calculated according to Equation 15.1 as follows:

$$d_{(56)1} = \min(d_{15}, d_{16}) = \mathbf{10.29}$$

$$d_{(56)2} = \min(d_{25}, d_{26}) = \mathbf{8.66}$$

. . .

$$d_{(56)8} = \min(d_{58}, d_{68}) = \mathbf{6.78}$$

This gives the new dissimilarity matrix below where genes 5 and 6 are merged and the distances recalculated as per the single linkage metric:

	Gene 1	*Gene 2*	*Gene 3*	*Gene 4*	*Gene {5,6}*	*Gene 7*	*Gene 8*
Gene 1	0	2.36	1.73	10.72	10.29	8.00	6.92
Gene 2		0	1.41	9.27	8.66	6.08	6.70
Gene 3			0	10.39	9.75	6.86	7.41
Gene 4				0	1.41	7.41	6.86
Gene {5,6}					0	6.86	6.78
Gene 7						0	9.80
Gene 8							0

The process of merging the two least dissimilar gene vectors and recalculating the distance matrix is continued until all genes are merged and plotted as a dendrogram.

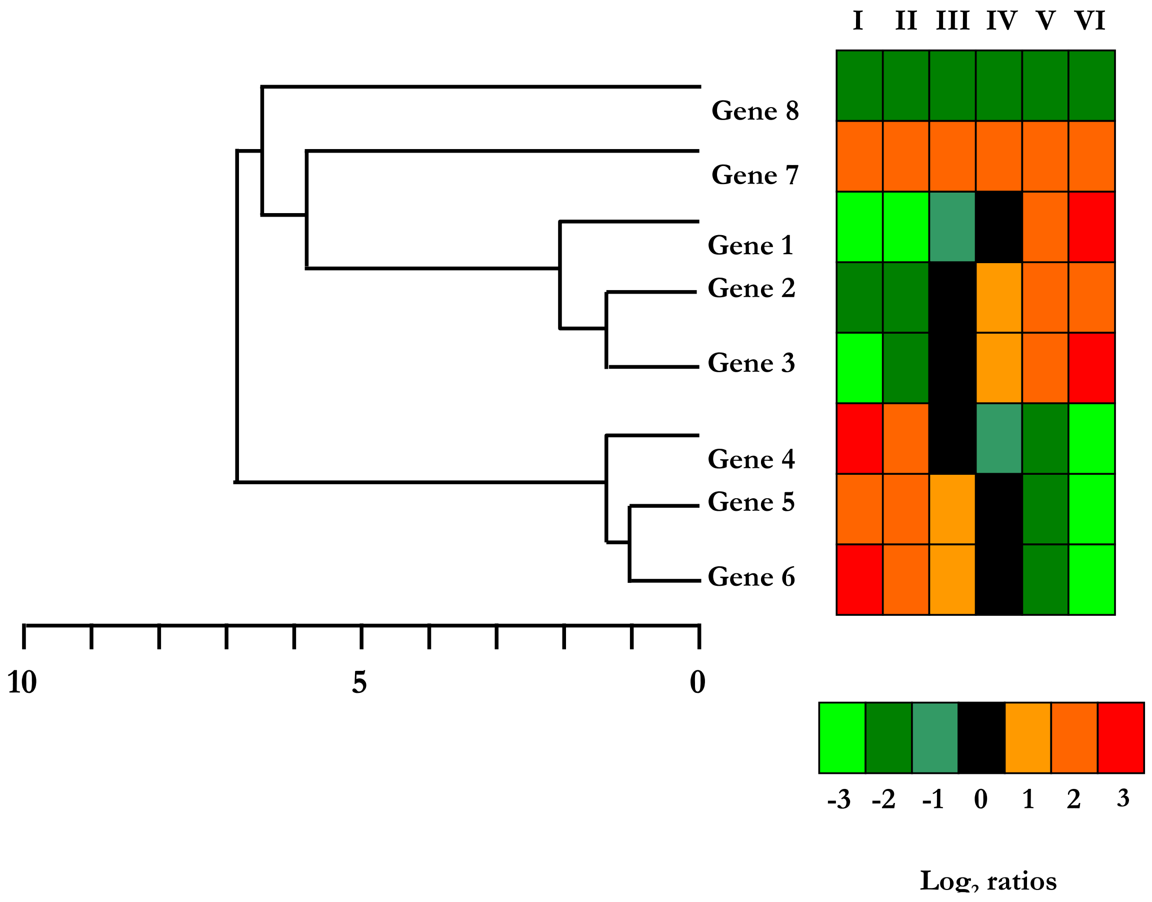

Figure 15.3. The Gene Expression dendrogram.

Interpretation of the dendrogram in Fig 15.3 is greatly aided by displaying the $\log_2$ ratio gene expression values as a coloured block diagram, where each row represents a gene across the columns of samples. The colour of each row/column block (feature) is related to the actual $\log_2$ ratio as shown by the colour range at the bottom of Figure 15.3. The data is taken from figure 15.1a and represents in shades of green and red the same information as plotted in Figure 15.1b. From this it is relatively easy to see the gene expression patterns that lead to the dendrogram structure derived from the single linkage of the gene expression Euclidean distances. Consistent with the known gene expression patterns in Figure 15.1b the 8 genes split into 3 groups in the dendrogram.

A challenging issue with hierarchical clustering is how to decide the *optimal* partition from the hierarchy, i.e. what is the best number of groups? One approach is to select a partition that best fits the data in some sense, and there are many methods that have been suggested in the literature [Everitt93]. It has also been found that the single-link algorithm tends to exhibit the so-called *chaining* effect: it has a tendency to cluster together at a relatively low level objects linked by chains of intermediates. As such, the method is appropriate if one is looking for "optimally" connected clusters rather than for homogeneous spherical clusters. The complete-link algorithm, on the other hand, tends to produce clusters that tightly bound or compact, and has been found to produce more useful hierarchies in many applications than the single-link algorithm [Jain99]. The group-average algorithm is also widely-used. Detailed discussion and practical examples of how these algorithms work can be found in [Jain99,Webb99].

15.3.2. K-Means

A non-hierarchical or partitional clustering algorithm produces a single partition of the data instead of a clustering structure such as the dendrogram. This type of algorithm has advantages in applications involving large data sets where the construction of dendrograms is computationally prohibitive. These algorithms usually obtain clusters by optimising a criterion function, of which the square error is the most commonly used. K-means is the best-known partitional algorithm employing such a criterion where the square error for each cluster j, [j= 1,2, …K], is the sum of the squared Euclidean distances between each pattern $\mathbf{x}_i^{(j)}$ in the cluster *j* and its centre, or mean vector, of the cluster, $\mathbf{m}^{(j)}$

$$E_j = \sum_{i=1}^{n_j} d^2_{\mathbf{x}_i,\mathbf{m}^{(j)}} \qquad \text{where} \quad \mathbf{m}^{(j)} = \frac{\sum_{i=1}^{n_j} \mathbf{x}_i^{(j)}}{n_j} \tag{15.4}$$

Where E_j is refereed to as the *within-cluster variation or sum of squares* for cluster *j*, n_j is the number of patterns within cluster *j*, and $d_{x_i,m^{(j)}}$ is the Euclidean distance from pattern $\mathbf{x}_i$ to the centre of the cluster to which it is assigned. Therefore the total square error for the entire clustering with *K* clusters is the sum of the within-cluster sum of squares:

$$E = \sum_{j=1}^{K} E_j \tag{15.5}$$

Basically the K-means algorithm starts with an initial partition with a fixed number of clusters and cluster centers, and proceeds with assigning each pattern to its closest cluster centre so as to reduce the square error between them. This is repeated until convergence is achieved, e.g. there is no reassignment of any pattern from one cluster to another, or the squared error ceases to decrease significantly after certain number of iterations.

Here is a view of the K-means algorithm:

1) Choose *K* cluster centers to coincide with *K* randomly-chosen patterns. Repeat the following steps until the cluster membership is stable
2) Assign each pattern to its closest cluster center
3) Compute the new cluster centers using the new cluster memberships
4) Repeat the above two steps until a convergence criterion is satisfied
5) If desirable, adjust the number of clusters by merging and splitting existing clusters

It should be noted that the last step is not an essential part of the K-means algorithm, but it has been included in some of the most well-known systems, e.g, in ISODATA [Webb99]. The adjustment of the number of clusters may be desirable if certain conditions are met. For example, if a cluster has too many patterns and an unusually large variance along the feature with the largest spread, it is probably a good idea to split the cluster. On the other hand, if two cluster centres are sufficiently close, they should perhaps be merged.

Let us have a look at how this algorithm works with a simple example. Consider the two-dimensional data as shown in Figure 15.4 where there are six patterns. Suppose we set *K* (the number of clusters) to 2 and choose two patterns from the data set as initial cluster centre vectors. Those selected are points 5 and 6. We now apply the algorithm to the data set and allocate individuals to clusters A and B represented by the initial vectors 5 and 6 respectively. It is clear that individuals 1, 2, 3, 4, 5 are allocated to cluster A and individual 6 to cluster B. New centres (means) are then calculated with (2,2) for cluster A and (4,3) for cluster B, and the sum of within-cluster sum-of-squares is calculated using equation 15.5, which gives 6:

$[(1-2)^2 + (1-2)^2] + [(2-2)^2 + (1-2)^2] + [(2-2)^2 + (2-2)^2] + [(2-2)^2 + (3-2)^2] + [(3-2)^2 + (3-2)^2] + [(4-4)^2 + (3-3)^2] = 2 +1 +0 + 1 + 2 + 0 = 6.$

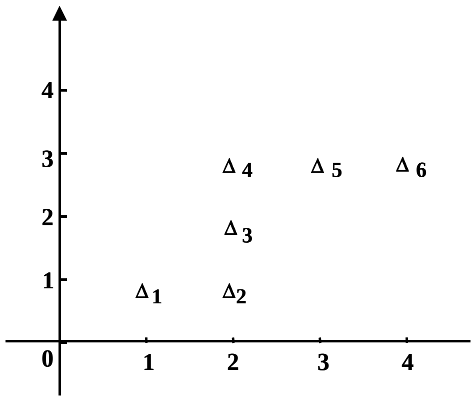

Figure 15.4: Sample data for K-Means

The results of this iteration are summarised in Table 15.3 in Step 1. This process is then repeated, using the new cluster centres for re-grouping individuals. This leads to the assignment of individuals 1, 2, 3, and 4 to cluster A and 5 and 6 to cluster B. The sum of within-cluster sum-of-squares is now reduced to 4. A third iteration produces no change in the cluster membership or the within-cluster sum-of-squares. The algorithm terminates here.

	Cluster A		**Cluster B**		**E**
Step	Membership	Centre	Membership	Centre	
1	1,2,3,4,5	(2,2)	6	(4,3)	6
2	1,2,3,4	(7/4, 7/4)	5,6	(3.5,3)	4
3	1,2,3,4	(7/4, 7/4)	5,6	(3.5,3)	4

Table 15.3. Summary of k-means iterations

The K-means algorithm is popular because it is easy to understand, easy to implement, and has a good time complexity. However, the algorithm can still take a considerable time if the number of patterns involved is very large and the number of clusters is substantial as with some of the large-scale gene expression applications. Another problem is that it is sensitive to the initial partition – the selection of the initial patterns, and may converge to a *local minimum* of the criterion function value if the initial partition is not properly chosen. A possible remedy is to run the algorithm with a number of different initial partitions. If they all lead to the same final partition, this implies that the global minimum of the square error has been achieved. However, this can be time-consuming, and may not always work.

15.3.3. Self-Organising Maps

The self-organising map (SOM) algorithm was proposed by Teuvo Kohonen in 1981. Apart from being used in a wide variety of fields, the SOM offers an interesting tool for exploratory data analysis, particularly for partitional clustering and visualisation. It is capable of representing high-dimensional data in a low dimensional space (often a two or one dimensional array) that preserves the structure of the original data.

A self-organising network consists of a set of input nodes $V = \{v_1, v_2, \ldots, v_N\}$, a set of output nodes $C = \{c_1, c_2, \ldots, c_M\}$, a set of weight parameters $W = \{w_{11}, w_{12}, \ldots, w_{ij}, \ldots, w_{NM}\}$ $(1 \leq i \leq N,\ 1 \leq j \leq M,\ 0 \leq w_{ij} \leq 1)$, and a map topology that defines the distances between any given two output nodes. The input nodes represent an N-dimensional vector, e.g. a single gene's expression level at N time points. Figure 15.5 is an illustration of how the gene expression example in Figure 15.1b may be analysed using SOM. In this case, each gene is expressed at six time points, therefore there are six input nodes. The output nodes in SOM are usually arranged in a *2*-dimensional array to form a "map". In this illustrative example, there are 20 output nodes on the map although the most appropriate number of output nodes usually needs to be determined via experimentation.

Each input node is fully connected to every output node via a variable connection. A weight parameter is associated with each of these connections, and the weights between the input nodes and output nodes are iteratively changed during the learning phase until a termination criterion is satisfied. For each input vector $\boldsymbol{v}$, there is one associated *winner node* on the output map. A winner node is an output node that has minimum distance to the input vector. This can be seen for the output map for winner node corresponding to gene 1 in Figure 15.5 (addendum).

Here is the sketch of the SOM algorithm:

1) Initialise the topology and size of the output map (*M*)
2) Initialise the connection weights to random values over the interval [0, 1] and normalise both the input vectors and the connected weight vectors. Initialise the gain value η (the learning rate) and the neighbourhood size *r*.
3) Repeat until convergence:
   - Present a new input vector **v**
   - Calculate the Euclidean distance between the input vector and each node on the output map, and designate the node with the minimum distance as the winner node *c*

$$c = \min \sqrt{\sum_{k=1}^{N} \left(v_k - w_{jk}\right)^2}\ ,\ j=1,2,\ldots,N \tag{15.6}$$

   - Update weights *W*, learning rate η and N_c, the neighbourhood surrounding the winner node *c,* in such way that the vectors represented by output nodes are similar if they are located in a small neighbourhood. For each node j $\in N_c$, perform the following:

$$w_j^{(new)} = w_j^{(old)} + \eta\,[v_i - w_j^{(old)}] \tag{15.7}$$

   - Decrease both the neighbourhood size and the learning rate.

The neighbourhood set N_c is a set of nodes that surround the winner node *c*. These nodes in N_c are affected with weight modifications, apart from those changes made to the winner node, as defined in the algorithm. These weight changes are made to increase the matching between the nodes in N_c and the corresponding input vectors. As the update of weight parameters proceeds, the size of the neighbourhood set is slowly decreased to a predefined limit, for instance, a single node. This process leads to one of the most important properties of SOM that similar input vectors are mapped to geometrically close winner nodes on the output map. This is called *neighbourhood preservations*, which has turned out to be very useful for clustering similar data patterns. Going back to Figure 15.5, we can see that the output map status at time *t* and the final map when the computational process has ended. It becomes clear that there are three distinctive clusters on the output map: {1,2,3}, {4,5,6} and {7,8}.

It should be noted that for illustrative purpose, each gene profile in this example has been mapped onto a distinctive node on the output map. This should not always be the case. As long as two gene profile vectors are sufficiently similar, they may be mapped onto the same node on the output map.

It is not always straightforward to visually inspect the projected data on the two-dimensional output map in order to decide the number and size of natural clusters. Therefore, careful analysis or post-processing of output maps is crucial to the partition of the original data set. Like the K-means algorithm, the SOM produces a sub-optimal partition if the initial weights are not chosen properly. Moreover, its convergence is controlled by various parameters such as the learning rate, the size and shape of the neighbourhood in which learning takes place. Consequently, the algorithm may not be very *stable* in that a particular input pattern may produce different winner nodes on the output map at different iterations.

15.3.4. Discussion

Different clustering algorithms may produce different clusters from the same data set. Some may fail to detect obvious clusters, while many will always be able to produce clusters even when there is no natural grouping in the data. This is because each algorithm implicitly forces a structure on the given data. Thus it is extremely important to emphasise that cluster analysis is usually one of the first steps in an analysis. Accepting the results of a cluster algorithm without careful analysis and follow-up study can lead to misleading results. The validation of clustering results is difficult. There is no optimal strategy for cluster interpretation, although a few formal approaches have been proposed such as the use of "measures of distortion" and the use of "internal criteria measures" to determine the number of clusters [Webb99]. The application scope of these methods is rather limited. Relevant domain knowledge such as biological information about the genes and associated proteins in a cluster can help check the consistency between the clustering findings and what has already known and remains one of the most important ways of validating the results.

Hierarchical clustering, K-means and SOM are probably the most commonly used clustering methods for gene expression analysis. By no means should they be considered the only ones worth using. The global search and optimisation methods such as genetic algorithms or simulated annealing can find the optimal solution to the square error criterion, and have already demonstrated certain advantages. Finally, fuzzy clustering may have much to offer as well. Since a gene may have multiple functional roles in various pathways, it can in effect be the member of more than one cluster.

15.4 Classification

The task of *classification* exists in a wide variety of domains. In medical diagnosis, there may exist many cases that correspond to several diseases, together with their associated symptoms. We would like to know whether there are any diagnostic rules underlying these cases that would allow us to distinguish between these diseases. In credit scoring, a company may use information regarding thousands of existing customers to decide whether any new customer should be given a particular credit status. This information can include a variety of financial and personal data. In the gene expression settings, we might want to predict the unknown functions of certain genes, given a group of genes whose functional classes are already known. In all these examples, there are the following common themes. First, we have a set of *cases*, e.g. patients, customers, or genes. Second, we are interested in assigning each new case to one of the pre-defined *classes,* e.g. different disease categories, credit status, or gene function classes. Third, there is a set of *features* on which the classification decision is based, e.g. a set of symptoms, personal or financial information, or gene expression level at different time points. Essentially we are interested in how to construct a classification procedure from a set of cases whose classes are known so that such a procedure can be used to classify new cases.

The objective of learning classification models from sample data is to be able to classify new data successfully, with the corresponding error rates as low as possible. Often one would want to split all the available data randomly into two groups: one for designing the classification model, and the other for testing the model accuracy. Traditionally for a single application of the training-and-testing method, the proportions of the data used for training and testing are approximately a 2/3, 1/3 split. However, if only a small number of cases are available, such a split would have problems since this will leave one with either insufficient training or testing cases. A more suitable way of dealing with such situations would be the use of *resampling* techniques such as *cross-validation* or *bootstrapping*. Take the *k*-fold cross-validation as an example; the cases are randomly partitioned into *k* mutually exclusive test partitions of roughly equal size. For each of the *k* test partitions, all cases not found in this partition are used for training, and the corresponding classifier is tested on this test partition. The average error rates over all *k* partitions are then the cross-validated error rate. In this way, we have made the full use of the limited number of cases, for testing as well as training. The 10-fold cross-validation method appears to be one of the most commonly used.

There have been many classification methods proposed in the statistical, artificial intelligence and pattern recognition communities. Commonly-used methods include Bayesian classifiers, linear discriminant analysis, nearest neighbour classification, classification tree, regression tree, neural networks, genetic algorithms, and more recently, support vector machines. Detailed discussions of most of these topics may be found in [Hand2001, Webb99].

15.4.1. Support Vector Machines (SVM)

Over the last few years, the SVM has been established as one of preferred approaches to many problems in pattern recognition and regression estimation, including handwritten digit recognition, object recognition, speaker identification, face detection, text mining, time series predictions, and gene expression classification. In most cases, SVM generalisation performances have been found to be either equal to or much better than that of the conventional methods. This is largely due to the SVM's ability to achieve the right balance between the accuracy attained on a particular training set, and the *capacity* of the learning machine, that is, the ability of the machine to learn any training set without error. Statistical learning theory [Vapnik 1995] shows that it is crucial to restrict the class of functions that the learning machine can implement to one with a suitable capacity for the amount of training data available.

Suppose we are interested in finding out how to separate a set of training data vectors that belong to two distinct classes. If the data are separable in the input space, there may exist many hyperplanes that can do such a separation, but we are interested in finding the optimal hyperplane classifier – the one with the maximal margin of separation between the two classes. Without going into full details the maximal margin of separation can be uniquely constructed by solving a constrained quadratic optimisation problem involving *support vectors*, a small subset of patterns that that lie on the margin. The support vectors, often just a small percentage of the total number of training patterns, contain all relevant information about the classification problem. For the illustration of these concepts a simple partitioning of a data set into '+' and '-' classes is shown in Figure 15.6. The corresponding linear SVMs can be easily constructed to separate the two classes.

When the two classes are not separable in the input space, the support vector machine can be extended to derive non-linear decision rules using so-called *kernel functions*. Kernel functions in this context are meant to be those that map the input data into some feature space. The key idea is to map the training data nonlinearly into a high-dimensional feature space, and then construct a (linear) separating hyperplane with maximum margin there. This yields a non-linear decision boundary in input space the concept of which is shown in Figure 15.7. The development of SVM began with the consideration of two classes problems, but have recently been extended to multi-class problems.

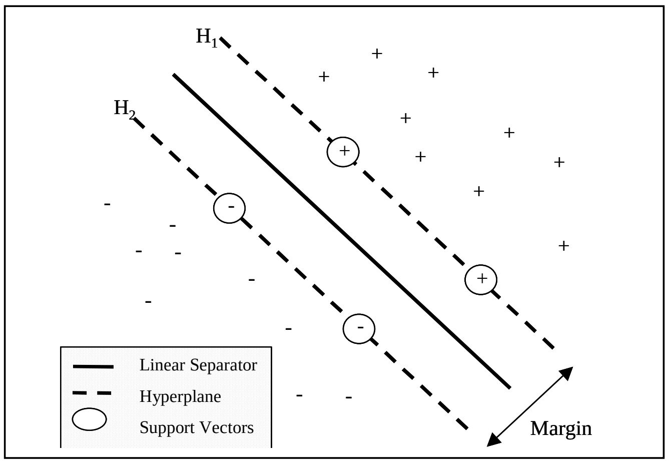

Figure 15.6. Hyperplanes and support vectors

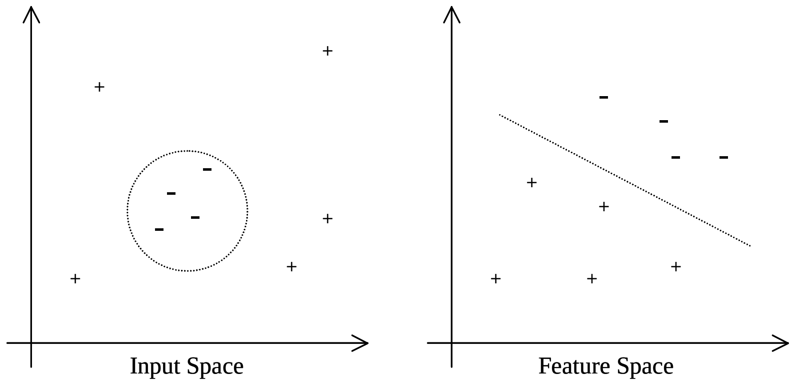

Figure 15.7. Non-linear mapping

Support Vector Machines offer a novel approach to classification. SVM training always finds a global minimum, while its neural network competitors may get stuck with a local minimum. It has simple geometric interpretation and it can be analysed theoretically using concepts from statistical learning theory. It has also found much success with many practical problems. However, it should be noted that currently there is no established theory that can guarantee that a given family of SVMs will have high accuracy on a given problem. SVM is very much characterised by the selection of its kernel, and it is not always easy to choose a kernel function that is most suited to the classification problem in hand. Computational efficiency is also a concern for large-scale applications. A detailed introduction of the SVM can be found in [Cristianini 2000].

15.4.2. Discussion

Classification, or discriminant analysis, is a well-researched subject with a vast literature. However, it is not always easy to decide the most suitable method for a classification task in hand. Given that most of these methods included in many of the commonly-used statistical or data mining packages, performance comparison of these methods on a given problem has been made easier. It is perhaps useful to point out that predictive accuracy on unseen test data is not the only, or even the most important, criterion for assessing the performance of a classification system. Other important factors in deciding that one method is preferable to another include computational efficiency, misclassification cost, and the interpretability of the classification models.

Much more work is needed to study the relative strengths and weaknesses of the various methods.

15.5. Conclusion and Future Research

While there are many statistical, data mining, or new gene expression data analysis tools that can be relatively easily used for clustering, classification and visualisation, we will witness very intensive research effort on the construction of networks for understanding gene interaction and regulation. Following the clustering of potentially co-expressed genes, it is desirable to develop a computational model that uses the expression data of these genes and relevant biological knowledge to understand interactions and regulation of these genes, for example, how the proteins of the expressed genes in a cluster interact in a functional module. Although the work in this relatively new area of functional genomics, *network modelling*, has begun more recently, a variety of approaches have already been proposed. These range from the Boolean network to Bayesian network, and from time series modelling to the learning of such networks from the literature. The development of most of these approaches is still at an early stage, and it is often very difficult to compare them [D'haeseleer2000].

Major breakthrough in this area will not come by easily. This will require a truly interdisciplinary effort in which experimentalists, bioinformaticians, computer scientists and mathematicians work closely together. A much closer integration of network modelling with other data pre-processing activities as well as with the experimental design of microarrays will be necessary. It will also require much more experience in constructing and comparing gene networks using different methods for the same expression data. Last but not least, the timely and dynamic use of various kinds of relevant information such as genes, proteins, their relationships, diseases, patients, and clinical data, which may exist in various bioinformatics resources, medical databases, scientific literature, and a variety of web sites will greatly aid network modelling.

REFERENCES

[Cristianini 2000] N Cristianini and J Shawe-Taylor (2000) An Introduction to Support Vector Machines, Cambridge University Press.

[D'haeseleer2000] D'haeseleer, S Liang and R Somogyi (2000) Genetic Network Inference: from Co-Expression Clustering to Reverse Engineering, **Bioinformatics**, 16:707-726.

[Everitt1993] BS Everitt (1993) *Cluster Analysis*, Arnold. 3rd edition.

[Hand2001] Hand D.J., Mannila H., and Smyth P. (2001) *Principles of data mining*, MIT Press.

[Jain1999] AK Jain and RC Dubes (1999) Data clustering, **ACM Computing Survey**, 31:264-323.

[Vapnik1995] V N Vapnik (1995) *The Nature of Statistical Learning Theory*, Springer-Verlag.

[Webb1999] A Webb (1999) *Statistical Pattern Recognition*, Arnold.

Addendum

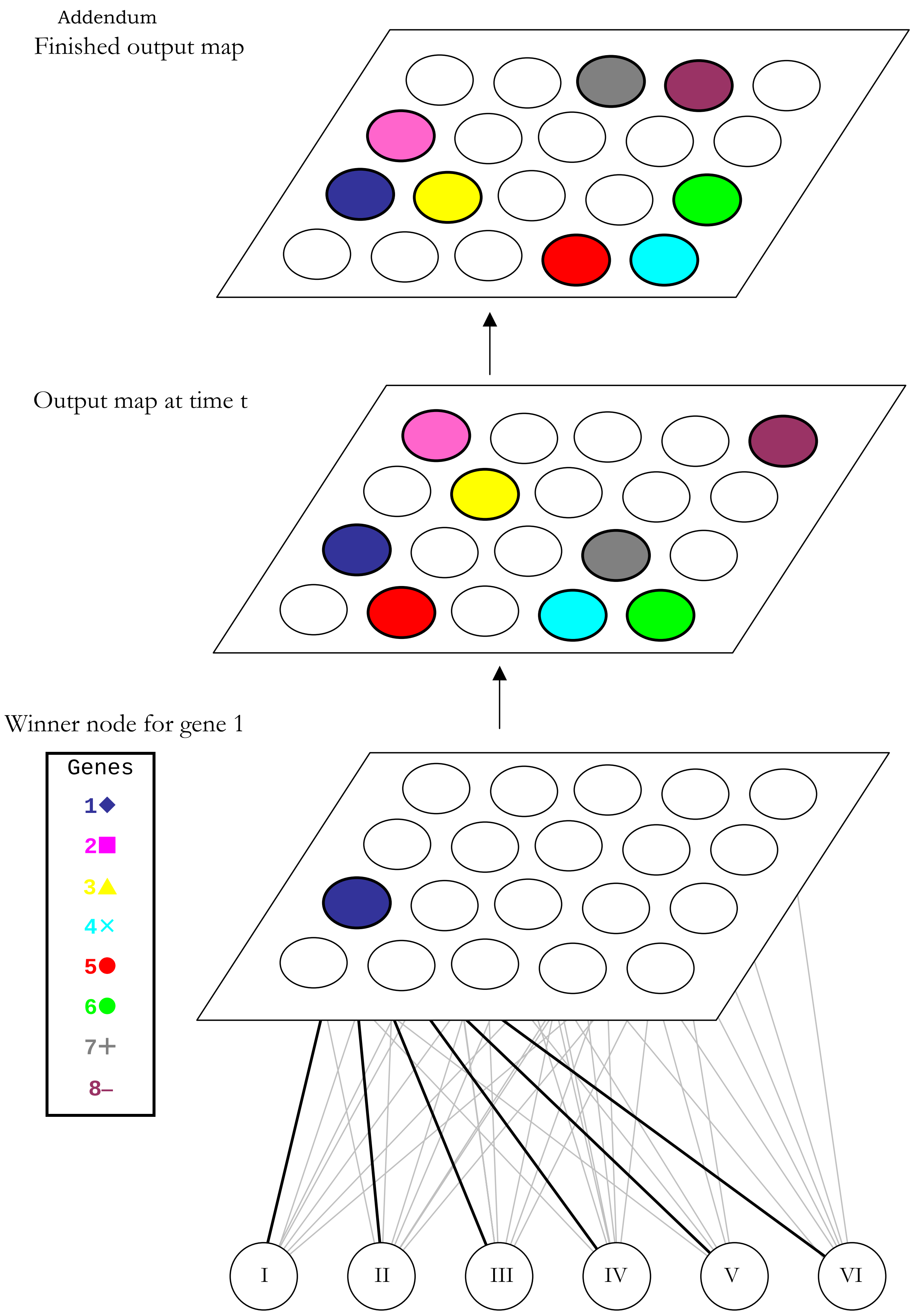

Figure 15.5: Self Organising Map