CHAPTER 3

DATA MINING TECHNIQUES FOR THE PRACTICAL BIOINFORMATICIAN

Jinyan Li
Institute for Infocomm Research
jinyan@i2r.a-star.edu.sg

Huiqing Liu
Institute for Infocomm Research
huiqing@i2r.a-star.edu.sg

Anthony Tung
National University of Singapore
atung@comp.nus.edu.sg

Limsoon Wong
Institute for Infocomm Research
limsoon@i2r.a-star.edu.sg

The use of computing to organize, manage, and analyse biological and clinical data has become an important element of biology and medical research. In the 1990s, large quantities of DNA sequence data were generated by the Human Genome Project and other genome sequencing efforts. Computing projects in algorithms, software, and databases were crucial in the automated assembly and analysis of the genomic data. In the 2000s, we enter the so-called post-genome era. Many genomes have already been completely sequenced. Genome research migrates from raw data generation into scientific knowledge discovery. Computing shifts correspondingly from managing and integrating sequence databases to discovering knowledge from such biological data.

Data mining tools and techniques are amongst the key computing technologies that are playing an important role in the ongoing post-genome extension of human's understanding of biological life. This chapter is an in-depth review of data mining techniques, including feature selection methods, classification methods, clustering methods, and association rules discovery methods. We also briefly

mention examples of their use in biomedicine.

ORGANIZATION.

Section 1. Feature selection is concerned with the issue of distinguishing signal from noise in data analysis. We begin with an exposition on the curse of dimensionality to motivate the importance of feature selection. Then we present a number of feature selection methods, including signal-to-noise, t-test, Fisher criterion, entropy, χ^2, information gain, information gain ratio, Wilcoxon rank sum test, CFS, and PCA.

Section 2. Classification is the process of finding models, based on selected features from training data, that separate two or more data classes. We review in this section a number of classification methods, including decision tree induction, Bayesian inference, hidden Markov models, artificial neural networks, support vector machines, and emerging patterns.

Section 3. The problem of association rule mining from a dataset is that of how to generate association rules that are interesting. We describe the two best-known methods for mining association rules, *viz.* the Apriori algorithm and the Max-Miner algorithm.

Section 4. Clustering is the process of grouping a set of objects into clusters so that objects within a cluster have high similarity to each other and high dissimilarity with objects in other clusters. We first discuss factors that can affect clustering. Then we describe clustering methods based on the partitioning concept, *viz.* k-means, EM, and k-mediods. Following that, we describe clustering methods based on the hierarchical decomposition concept, *viz.* AGNES and DIANA.

1. Feature Selection Methods

Feature selection is concerned with the issue of distinguishing signal from noise in data analysis. This section begins with subsection on the curse of dimensionality to motivate the importance of feature selection. It is then followed by several subsections, each presenting a different feature selection method.

1.1. *Curse of Dimensionality*

Let us assume that we would like to analyse data points in a d-dimensional unit hypercube. How much of each dimension of the unit hypercube is needed if we would like to sample a proportion p of the total volume of possible data points in this space? To calculate this, let the amount that we would need of each dimension be x. Then $p = x^d$. Thus $x = \sqrt[d]{p}$. Therefore, there is exponential growth in the proportion of each dimension that is needed to sample a fixed proportion of a space, with respect to the number of dimension that the space has. This observation is called the curse of dimensionality.

To appreciate the significance of this curse, let us look at Figure 1. In Part I of this figure, we see that if we utilize 50% of each of the two dimensions, we

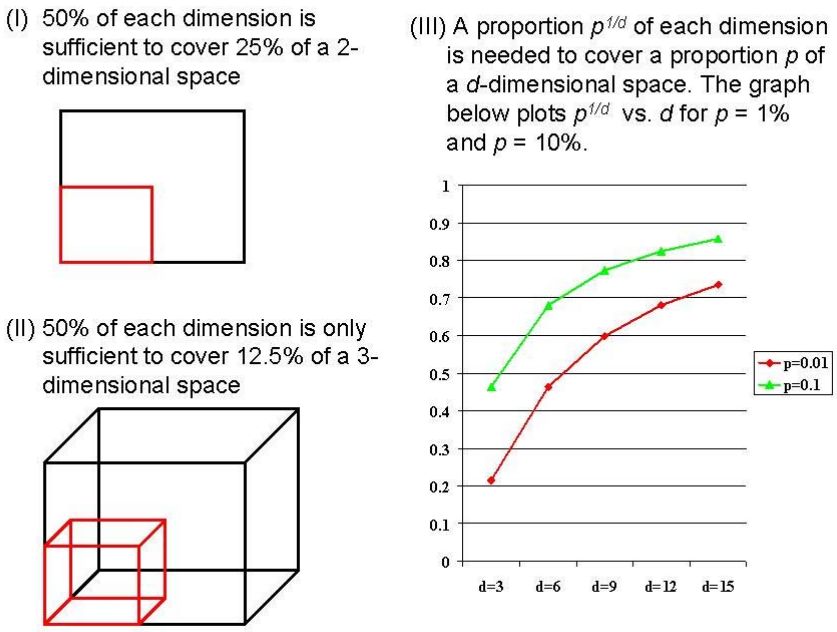

Fig. 1. Illustrations of the curse of dimensionality.

can cover 25% of the 2-dimensional space. In Part II of this figure, we see that if we utilize 50% of each of the three dimensions, we can cover only 12.5% of the 3-dimensional space. Conversely, we can also see from the graph in Part III of this figure that we would need 46% of each dimension to cover 10% of the volume of a 3-dimensional space, and nearly 86% of each dimension to cover 10% of the volume of a 15-dimensional space.

This curse of dimensionality is an important source of difficulty in effective data mining analysis. In order for a data mining tool to properly analyse a space and to reliably make predictions in that space, the tool needs to sample a certain proportion of the data points in that space. As a consequence of the curse, the tool needs an exponential increase in the number of training samples with respect to an increase in the dimensionality of the samples in order to uncover and learn the relationship of the various dimensions to the nature of the samples. [328]

It is therefore an important issue to determine if a dimension is relevant to the

kind of data mining analysis that we would like to perform on a space. If we could determine beforehand that certain dimensions are irrelevant, then we can omit them in our data mining analysis and thus mitigate the curse of dimensionality.

In the data mining tradition, the term "feature" and the term "attribute" are often preferred over the term "dimension." So we use the three terms interchangeably here. Also, a feature f and its value v in a sample are some times referred to as an "item" by the data mining community. A set of items is then called an "itemset," typically written using a notation like $\{f_1 = v_1, ..., f_n = v_n\}$ for an itemset containing features f_1, ..., f_n and associated values v_1, ..., v_n. An itemset can also be represented as a vector $\langle v_1, ..., v_n \rangle$, with the understanding that the value of feature f_i is kept in the i-th position of the vector. Such a vector is usually called a feature vector.

The dimensions or features that are relevant are called signals. The dimensions or features that are irrelevant are called noise. In rest of this section, we present several techniques for distinguishing signals from noise, *viz.* signal-to-noise measure,[297] t-test statistical measure,[133] entropy measure,[242] χ^2 measure,[514] information gain measure,[692] information gain ratio,[693] Fisher criterion score,[251] Wilcoxon rank sum test,[742] as well as a correlation-based feature selection method known as CFS[316] and the "unsupervised" approach of principal component analysis.[399]

1.2. *Signal-to-Noise Measure*

Suppose we have two classes $\mathcal{A}$ and $\mathcal{B}$ of sample data points in the space that we would like to analyse. Then a feature is relevant if it contributes to separating samples in $\mathcal{A}$ from those in $\mathcal{B}$. Conversely, a feature is irrelevant if it does not contribute much to separating samples in $\mathcal{A}$ from those in $\mathcal{B}$.

Let us look at Figure 2 which shows the distributions of the values of 3 features in samples of $\mathcal{A}$ and $\mathcal{B}$ as bars in the three charts. Let us compare Chart I and Chart III. The feature represented by Chart I is not as useful as the feature represented by Chart III for distinguishing $\mathcal{A}$ and $\mathcal{B}$. To see this, we appeal to the following intuitive concept for distinguishing a relevant feature from an irrelevant one: If the values of a feature in samples in $\mathcal{A}$ are significantly different from the values of the same feature in samples in $\mathcal{B}$, then the feature is likely to be more relevant than a feature that has similar values in $\mathcal{A}$ and $\mathcal{B}$. More specifically, in order for a feature f to be relevant, its mean value $\mu_f^{\mathcal{A}}$ in $\mathcal{A}$ should be significantly different from its mean value $\mu_f^{\mathcal{B}}$ in $\mathcal{B}$.

Similarly, the feature represented by Chart II of Figure 2 is not as useful as the feature represented by Chart III for distinguishing $\mathcal{A}$ and $\mathcal{B}$. To see this, we

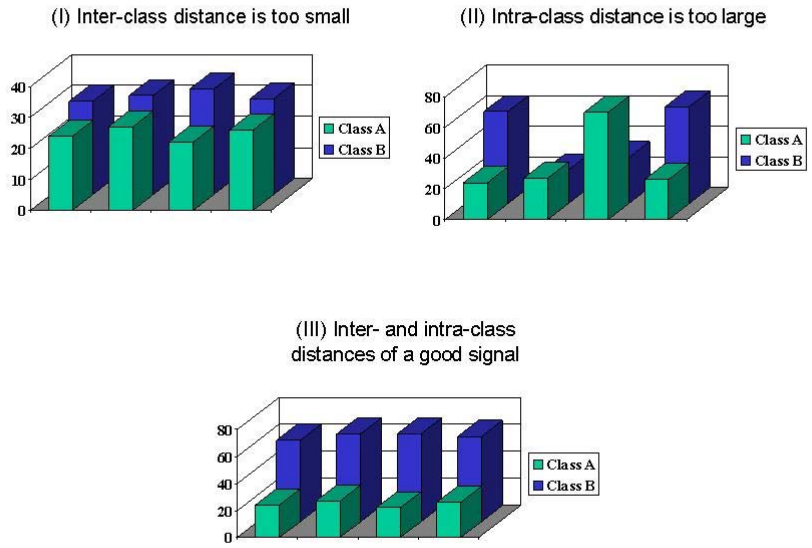

Fig. 2. Characteristics of a feature that is a poor signal (I and II), and of a feature that is a good signal (III).

appeal to a second intuitive concept that if the values of a feature f varies greatly within the same class of samples, then the feature f cannot be a reliable one. More specifically, in order for a feature f to be relevant, the standard deviation $\sigma_f^{\mathcal{A}}$ and variance $(\sigma_f^{\mathcal{A}})^2$ of f in $\mathcal{A}$ and the standard deviation $\sigma_f^{\mathcal{B}}$ and variance $(\sigma_f^{\mathcal{B}})^2$ of f in $\mathcal{B}$ should be small.

One obvious way to combine these two concepts is the signal-to-noise measure proposed in the first paper[297] that applied gene expression profiling for disease diagnosis:

$$s(f, \mathcal{A}, \mathcal{B}) = \frac{|\mu_f^{\mathcal{A}} - \mu_f^{\mathcal{B}}|}{\sigma_f^{\mathcal{A}} + \sigma_f^{\mathcal{B}}}$$

Then a feature f can be considered better than a feature f' if $s(f, \mathcal{A}, \mathcal{B}) > s(f', \mathcal{A}, \mathcal{B})$. Thus given a collection of candidate features in samples of $\mathcal{A}$ and $\mathcal{B}$, we simply sort them by their signal-to-noise measure and pick those with the

largest signal-to-noise measure.

1.3. *T-Test Statistical Measure*

However, the statistical property of $s(f, \mathcal{A}, \mathcal{B})$ is not fully understood. A second and older way to combine the two concepts of good signals mentioned earlier is the classical t-test statistical measure given below:

$$t(f, \mathcal{A}, \mathcal{B}) = \frac{|\mu_f^{\mathcal{A}} - \mu_f^{\mathcal{B}}|}{\sqrt{\frac{(\sigma_f^{\mathcal{A}})^2}{n^{\mathcal{A}}} + \frac{(\sigma_f^{\mathcal{B}})^2}{n^{\mathcal{B}}}}}$$

where $n^{\mathcal{A}}$ and $n^{\mathcal{B}}$ are the number of samples in $\mathcal{A}$ and $\mathcal{B}$ seen. Then a feature f can be considered better than a feature f' if $t(f, \mathcal{A}, \mathcal{B}) > t(f', \mathcal{A}, \mathcal{B})$. Thus given a collection of candidate features in samples of $\mathcal{A}$ and $\mathcal{B}$, we simply sort them by their t-test statistical measure, and pick those with the largest t-test statistical measure, or those that satisfy a suitably chosen statistical confidence level.

The t-test statistical measure is known[58,133] to require a Student distribution with

$$\frac{\left(\frac{(\sigma_f^{\mathcal{A}})^2}{n^{\mathcal{A}}} + \frac{(\sigma_f^{\mathcal{B}})^2}{n^{\mathcal{B}}}\right)^2}{\frac{\left(\frac{(\sigma_f^{\mathcal{A}})^2}{n^{\mathcal{A}}}\right)^2}{n^{\mathcal{A}} - 1} + \frac{\left(\frac{(\sigma_f^{\mathcal{B}})^2}{n^{\mathcal{B}}}\right)^2}{n^{\mathcal{B}} - 1}}$$

degrees of freedom. Hence if the values of the feature f is known not to obey this distribution, the t-test statistical measure should not be used for this feature.

1.4. *Fisher Criterion Score*

Closely related to the t-test statistical measure is the Fisher criterion score[251] defined as

$$fisher(f, \mathcal{A}, \mathcal{B}) = \frac{(\mu_f^{\mathcal{A}} - \mu_f^{\mathcal{B}})^2}{(\sigma_f^{\mathcal{A}})^2 + (\sigma_f^{\mathcal{B}})^2}$$

A feature f can be considered better than a feature f' if $fisher(f, \mathcal{A}, \mathcal{B}) > fisher(f', \mathcal{A}, \mathcal{B})$. The Fisher criterion score is known to require a nearly normal distribution,[251] and hence it should not be used if the distribution of the values of a feature is not nearly normal.

Each of $s(f, \mathcal{A}, \mathcal{B})$, $t(f, \mathcal{A}, \mathcal{B})$, and $fisher(f, \mathcal{A}, \mathcal{B})$ iis easy to compute and thus straightforward to use. However, there are three considerations that may make them ineffective. The first consideration is that they require the values of f to follow specific statistical distributions.[133, 251] It is often the case that we do not know what sort of statistical distributions the candidate features have. The second consideration is that the population sizes $n^{\mathcal{A}}$ and $n^{\mathcal{B}}$ should be sufficiently large. Small population sizes can lead to significant underestimates of the standard deviations and variances.

The third consideration is more subtle and we explain using the following example. Let f_1 and f_2 be two features. Suppose f_1 has values ranging from 0 to 99 in class $\mathcal{A}$ with $\mu_{f_1}^{\mathcal{A}} = 75$ and has values ranging from 100 to 199 in class $\mathcal{B}$ with $\mu_{f_1}^{\mathcal{B}} = 125$. Suppose f_2 has values ranging from 25 to 125 in class $\mathcal{A}$ with $\mu_{f_2}^{\mathcal{A}} = 50$ and has values ranging from 100 to 175 in class $\mathcal{B}$ with $\mu_{f_2}^{\mathcal{B}} = 150$. We see that $\mu_{f_2}^{\mathcal{B}} - \mu_{f_2}^{\mathcal{A}} = 100 > 50 = \mu_{f_1}^{\mathcal{B}} - \mu_{f_1}^{\mathcal{A}}$. Suppose the variances of f_1 and f_2 in $\mathcal{A}$ and $\mathcal{B}$ are comparable. Then according to the signal-to-noise and t-statistics measures, f_2 is better than f_1. However, we note that the values of f_1 are distributed so that all those in $\mathcal{A}$ are below 100 and all those in $\mathcal{B}$ are at least 100. In contract, the values of f_2 in $\mathcal{A}$ and $\mathcal{B}$ overlap in the range 100 to 125. Then clearly f_1 should be preferred. The effect is caused by the fact that $t(f, \mathcal{A}, \mathcal{B})$, $s(f, \mathcal{A}, \mathcal{B})$, $fisher(f, \mathcal{A}, \mathcal{B})$ are sensitive to all changes in the values of f, including those changes that may not be important.

So, one can consider alternative statistical measures that are less sensitive to certain types of unimportant changes in the value of f. What types of changes in values of f are not important? One obvious type is those that do not shift the values of f from the range of $\mathcal{A}$ into the range of $\mathcal{B}$. An alternative that takes this into consideration is the entropy measure.[242]

1.5. *Entropy Measure*

Let $P(f, \mathcal{C}, S)$ be the proportion of samples whose feature f has value in the range S and are in class $\mathcal{C}$. The class entropy of a range S with respect to feature f and a collection of classes $\mathcal{U}$ is defined as

$$Ent(f, \mathcal{U}, S) = -\sum_{\mathcal{C} \in \mathcal{U}} P(f, \mathcal{C}, S) \times \log_2(P(f, \mathcal{C}, S))$$

It follows from this definition that the purer the range S is, so that samples in that range are more dominated by one particular class, the closer $Ent(f, \mathcal{U}, S)$ is to the ideal entropy value of 0.

Let T partitions the values of f into two ranges S_1 (of values less than T) and S_2 (of values at least T). We sometimes refer to T as the cutting point of the

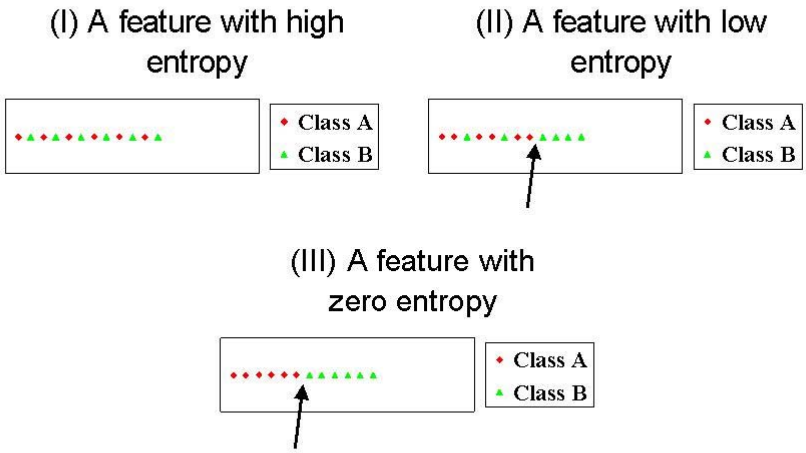

Fig. 3. Each of the three charts represents a feature. We placed the values that a feature takes on in samples of $\mathcal{A}$ and $\mathcal{B}$ on the horizontal, sorted according to magnitude. Chart I is characteristic of a feature that is a poor signal. Chart II is characteristic of a potentially good signal. Chart III is characteristic of the strongest signal.

values of f. The entropy measure[242] $e(f, \mathcal{U})$ of a feature f is then defined as

$$e(f, \mathcal{U}) = \min\{E(f, \mathcal{U}, \{S_1, S_2\}) \mid (S_1, S_2) \text{ is a partitioning of the values of } f \text{ in } \bigcup \mathcal{U} \text{ by some point } T\}$$

Here, $E(f, \mathcal{U}, \{S_1, S_2\})$ is the class information entropy of the partition (S_1, S_2). The definition is given below, where $n(f, \mathcal{U}, s)$ means the number of samples in the classes in $\mathcal{U}$ whose feature f has value in the range s,

$$E(f, \mathcal{U}, S) = \sum_{s \in S} \frac{n(f, \mathcal{U}, s)}{n(f, \mathcal{U}, \bigcup S)} \times Ent(f, \mathcal{U}, s)$$

The entropy measure $e(f, \mathcal{U})$ is illustrated in Figure 3. Chart I of this figure is characteristic of a feature that is a poor signal, because it is not possible to partition the range of the values into an interval containing mostly samples of $\mathcal{A}$ and an interval containing mostly samples of $\mathcal{B}$. In general, the entropy measure for such a feature is a large value. Chart II is characteristic of a potentially good

signal, because by partitioning the range of the values at the point indicated by the arrow, one of the interval contains purely samples in $\mathcal{B}$ while the other interval is dominated by samples in $\mathcal{A}$. In general, the entropy measure for such a feature is a small value. Chart III is characteristic of the strongest signal, because by partitioning the range of the values at the point indicated by the arrow, both intervals become pure. The entropy measure for this feature is 0. So under the entropy measure a feature f is more useful than a feature f' if $e(f,\mathcal{U}) < e(f',\mathcal{U})$.

A refinement of the entropy measure is to recursively partition the ranges S_1 and S_2 until some stopping criteria is reached.[242] A stopping criteria is needed because otherwise we can always achieve perfect entropy by partitioning the range into many small intervals, each containing exactly one sample. A commonly used stopping criteria is the so-called minimal description length principle.[242, 888] According to this principle, recursive partitioning within a range S stops iff S is partitioned into ranges S_1 and S_2 such that $n(f,\mathcal{U},S) \times Gain(f,\mathcal{U},S_1,S_2) < \log_2(n(f,\mathcal{U},S)-1)+\log_2(3^k-2)-k\times Ent(f,\mathcal{U},S)+k_1\times Ent(f,\mathcal{U},S_1)+k_2\times Ent(f,\mathcal{U},S_2)$, where $Gain(f,\mathcal{U},S_1,S_2) = Ent(f,\mathcal{U},S) - E(f,\mathcal{U},\{S_1,S_2\})$; and k, k_1, and k_2 are respectively the number of classes that have samples with feature f having values in the range S, S_1, and S_2.

Another refinement is the χ^2 measure.[514] Here, instead of the entropy measure $e(f,\{\mathcal{A},\mathcal{B}\})$ itself, we use the χ^2 correlation of the partitions S_1 and S_2 induced by the entropy measure to the classes $\mathcal{A}$ and $\mathcal{B}$. Some other refinements include the information gain measure and the information gain ratio that are used respectively in ID3[692] and C4.5[693] to induce decision trees. These refinements are described next.

1.6. χ^2 *Measure*

The χ^2 measure is a feature selection technique that evaluates features individually by measuring the chi-squared statistics with respect to the class. For a numeric feature, we should first "discretize" the range of its values into a number of intervals. For this purpose, it is common to simply use those intervals induced by the computation of the entropy measure presented in Subsection 1.5.

Then the χ^2 measure is given by the follow equation:

$$\chi^2(f,\mathcal{U}) = \sum_{i=1}^{m}\sum_{j=1}^{k}\frac{(A_{ij}(f,\mathcal{U}) - E_{ij}(f,\mathcal{U}))^2}{E_{ij}(f,\mathcal{U})}$$

Here, m is the number of intervals induced in computing $e(f,\mathcal{U})$ or its recursive refinement; k is the number of classes in $\mathcal{U}$; $A_{ij}(f,\mathcal{U})$ is the number of samples in the i-th interval that are of the j-th class; and $E_{ij}(f,\mathcal{U})$ is the expected frequency

of $A_{ij}(f,\mathcal{U})$, and hence

$$E_{ij}(f,\mathcal{U}) = R_i(f,\mathcal{U}) \times \frac{C_j(f,\mathcal{U})}{N(\mathcal{U})}$$

where $R_i(f,\mathcal{U})$ is the number of samples in the i-th interval, $C_j(f,\mathcal{U})$ is the number of samples in the j-th class; and using the notation $n^{\mathcal{C}}$ to denote the number of samples seen from class $\mathcal{C}$, we also define $N(\mathcal{U}) = \sum_{\mathcal{C}\in\mathcal{U}} n^{\mathcal{C}}$, as the total number of samples.

We consider a feature f to be more relevant than a feature f' in separating the classes in $\mathcal{U}$ if $\chi^2(f,\mathcal{U}) > \chi^2(f',\mathcal{U})$. Obviously, the $\chi^2(f,\mathcal{U})$ value takes on the worst value of 0 if the feature f has only one interval. The degrees of freedom of the χ^2 statistical measure is $(m-1)\times(k-1)$.[742]

1.7. *Information Gain*

Recall from Subsection 1.5 that $e(f,\mathcal{U}) = \min\{E(f,\mathcal{U},\{S_1,S_2\}) \mid (S_1,S_2)$ is a partitioning of the values of f in $\bigcup\mathcal{U}$ by some point $T\}$. We can interpret this number as the amount of information needed to identify the class of an element of $\mathcal{U}$.

Let $\mathcal{U} = \{\mathcal{A},\mathcal{B}\}$, where $\mathcal{A}$ and $\mathcal{B}$ are the two classes of samples. Let f be a feature and S be the range of values that f can take in the samples. Let S be partitioned into two subranges S_1 and S_2. Then the difference between the information needed to identify the class of a sample in $\mathcal{U}$ before and after the value of the feature f is revealed is $Gain(f,\mathcal{U},S_1,S_2) = Ent(f,\mathcal{U},S_1\cup S_2) - E(f,\mathcal{U},\{S_1,S_2\})$.

Then the information gain is the amount of information that is gained by looking at the value of the feature f, and is defined as

$$g(f,\mathcal{U}) = \max\{Gain(f,\mathcal{U},S_1,S_2) \mid (S_1,S_2) \text{ is a partitioning of the values of } f \text{ in } \bigcup\mathcal{U} \text{ by some point } T\}$$

This information gain measure $g(f,\mathcal{U})$ can also be used for selecting features that are relevant because we can consider a feature f to be more relevant than a feature f' if $g(f,\mathcal{U}) > g(f',\mathcal{U})$. In fact, the ID3 decision tree induction algorithm uses it as the measure for picking discriminatory features for tree nodes.[692]

1.8. *Information Gain Ratio*

However, $g(f,\mathcal{U})$ tends to favour features that have a large number of values. The information gain ratio is a refinement to compensate for this disadvantage. Let

$$GainRatio(f,\mathcal{U},S_1,S_2) = \frac{Gain(f,\mathcal{U},S_1,S_2)}{SplitInfo(f,\mathcal{U},S_1,S_2)}$$

where $SplitInfo(f, \mathcal{U}, S_1, S_2) = Ent(f, \{\mathcal{U}_f^{S_1}, \mathcal{U}_f^{S_2}\}, S_1 \cup S_2)$, and $\mathcal{U}_f^S = \bigcup_{\mathcal{C} \in \mathcal{U}} \{d \in \mathcal{C} \mid$ the feature f in sample d has value in range $S\}$. Then the information gain ratio is defined as

$$gr(f, \mathcal{U}) = \max\{GainRatio(f, \mathcal{U}, S_1, S_2) \mid (S_1, S_2) \text{ is a partitioning of the values of } f \text{ in } \bigcup \mathcal{U} \text{ by some point } T\}$$

The information gain ratio $gr(f, \mathcal{U})$ can of course be used for selecting features that are relevant because we can consider a feature f to be more relevant than a feature f' if $gr(f, \mathcal{U}) > gr(f', \mathcal{U})$. In fact, the C4.5 decision tree induction algorithm uses it as the measure for picking discriminatory features for tree nodes.[693]

1.9. *Wilcoxon Rank Sum Test*

Another approach to deal with the problems of test statistics that require the feature values to conform to specific distributions is to use non-parametric tests. One of the best known non-parametric tests is the Wilcoxon rank sum test, or the equivalent Mann-Whitney test.

The Wilcoxon rank sum test is a non-parametric test for the equality of two populations' mean or median. It does not require the two populations to conform to a specific distribution, other than that the two distributions have the same general shape.[742] It is thus an alternative to the t-test statistical measure and the Fisher criterion score, both of which require the population distributions to be nearly normal. However, if the assumption on near normality is correct, then the Wilcoxon rank sum test may not be as discriminating as the t-test statistical measure or the Fisher criterion score.

The Wilcoxon rank sum test statistical measure $w(f, \mathcal{U})$ of a feature f in a collection $\mathcal{U} = \{\mathcal{A}, \mathcal{B}\}$ of classes $\mathcal{A}$ and $\mathcal{B}$ is obtained using the following procedure:

(1) Sort the values $v_1, v_2, ..., v_{n^{\mathcal{A}}+n^{\mathcal{B}}}$ of f across all the samples in $\mathcal{U}$ in ascending order, so that $v_1 \leq v_2 \leq ... \leq v_{n^{\mathcal{A}}+n^{\mathcal{B}}-1} \leq v_{n^{\mathcal{A}}+n^{\mathcal{B}}}$.
(2) Assign rank $r(v_i)$ to each value v_i above so that ties are resolved by averaging. That is,

$$rank(v_i) = \begin{cases} i & \text{if } v_{i-1} \neq v_i \neq v_{i+1} \\ \dfrac{\sum_{k=0}^{n} j+k}{n+1} & \text{if } v_{j-1} < v_j = ... = v_i = ... = v_{j+n} < v_{j+n+1} \end{cases}$$

(3) Then

$$w(f, \mathcal{U}) = \sum_{v \in \text{ argmin}_{c \in \mathcal{U}} n^c} r(v)$$

That is, the Wilcoxon rank sum test statistical measure is the sum of the ranks for the class that has a smaller number of samples. If the number of samples is same in each class, the choice of which class to use is arbitrary.

To use the Wilcoxon rank sum test to decide if a feature f is relevant, we set up the null hypothesis that: the values of f are not much different in $\mathcal{A}$ and $\mathcal{B}$. Then $w(f, \{\mathcal{A}, \mathcal{B}\})$ is used to to accept or reject the hypothesis. To decide whether to accept or reject the null hypothesis, we have to compare $w(f, \{\mathcal{U}\})$ with the upper and lower critical values derived for a significant level α. For the cardinalities of $n^{\mathcal{A}}$ and $n^{\mathcal{B}}$ that are small, *e.g.* < 10, the critical values have been tabulated, and can be found in *e.g.* Sandy.[742]

If either $n^{\mathcal{A}}$ or $n^{\mathcal{B}}$ is larger than what is supplied in the table, the following normal approximation can be used. The expected value of $w(f, \{\mathcal{A}, \mathcal{B}\})$ is:

$$\mu = \frac{n^{\mathcal{A}} \times (n^{\mathcal{A}} + n^{\mathcal{B}} + 1)}{2}$$

assuming class $\mathcal{A}$ has fewer samples than class $\mathcal{B}$. The standard deviation of $w(f, \{\mathcal{A}, \mathcal{B}\})$ is known to be:

$$\sigma = \sqrt{\frac{n^{\mathcal{A}} \times n^{\mathcal{B}} \times (n^{\mathcal{A}} + n^{\mathcal{B}} + 1)}{12}}$$

The formula for calculating the upper and lower critical values is $\mu \pm z_\alpha \times \sigma$, where z_α is the z score for significant level α. If $w(f, \{\mathcal{A}, \mathcal{B}\})$ falls in the range given by the upper and lower critical values, then we accept the null hypothesis. Otherwise, we reject the hypothesis, as this indicates that the values of feature f are significantly different between samples in $\mathcal{A}$ and $\mathcal{B}$. Thus, those features whose Fisher criterion rejects the hypothesis are selected as important features.

1.10. *Correlation-Based Feature Selection*

All of the preceeding measures provide a rank ordering of the features in terms of their individual relevance to separating $\mathcal{A}$ and $\mathcal{B}$. One would rank the features using one of these measures and select the top n features. However, one must appreciate that there may be a variety of independent "reasons" why a sample is in $\mathcal{A}$ or is in $\mathcal{B}$. For example, there can be a number of different contexts and mechanisms that enable protein translation. If a primary context or mechanism involves n signals, the procedure above may select only these n features and may ignore

signals in other secondary contexts and mechanisms. Consequently, concentrating on such top n features may cause us to lose sight of the secondary contexts and mechanisms underlying protein translation.

This issue calls for another concept in feature selection: Select a group of features that are correlated with separating $\mathcal{A}$ and $\mathcal{B}$ but are not correlated with each other. The cardinality in a such a group may suggest the number of independent factors that cause the separation of $\mathcal{A}$ and $\mathcal{B}$. A well-known technique that implements this feature selection strategy is the Correlation-based Feature Selection (CFS) method.[316]

Rather than scoring and ranking individual features, the CFS method scores and ranks the worth of subsets of features. As the feature subset space is usually huge, CFS uses a best-first-search heuristic. This heuristic algorithm embodies our concept above that takes into account the usefulness of individual features for predicting the class along with the level of intercorrelation among them.

CFS first calculates a matrix of feature-class and feature-feature correlations from the training data. Then a score of a subset of features is assigned using the following heuristics:

$$Merit_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k \times (k-1) \times \overline{r_{ff}}}}$$

where $Merit_S$ is the heuristic merit of a feature subset S containing k features, $\overline{r_{cf}}$ is the average feature-class correlation, and $\overline{r_{ff}}$ is the average feature-feature intercorrelation.

Symmetrical uncertainties are used in CFS to estimate the degree of association between discrete features or between features and classes.[316] The formula below measures the intercorrelation between two features or the correlation between a feature X and a class Y which is in the range $[0, 1]$:

$$r_{xy} = 2 \times \frac{H(X) + H(Y) - H(X,Y)}{H(X) + H(Y)}$$

where $H(X) + H(Y) - H(X,Y)$ is the information gain between features and classes, $H(X)$ is the entropy of the feature. CFS starts from the empty set of features and uses the best-first-search heuristic with a stopping criterion of 5 consecutive fully expanded non-improving subsets. The subset with the highest merit found during the search is selected.

1.11. *Principal Component Analysis*

The feature selection techniques that we have described so far are "supervised" in the sense that the class labels of the samples are used. It is also possible to perform

feature selection in an "unsupervised" way. For example, the principal component analysis (PCA) approach[399] widely used in signal processing can be used in such a manner.

PCA is a linear transformation method that diagonalizes the covariance matrix of the input data via a variance maximization process. PCA selects features by transforming a high-dimensional original feature space into a smaller number of uncorrelated features called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. Feature selection through PCA can be performed by the following steps:

(1) Calculate the covariance matrix C of data X, where X is a matrix with n rows and m columns. Here, n is the number of samples and m is the number of features. Each column data of X may have to be normalized. Each element $C[i, j]$ of the matrix C is the linear correlation coefficient between the elements of columns i and j of X and is calculated as:

$$C[i,j] = \frac{\sum_{k=1}^{n}(X[k,i] - \mu_{X[\_,i]}) \times (X[k,j] - \mu_{X[\_,j]})}{\sqrt{\sum_{k=1}^{n}(X[k,i] - \mu_{X[\_,i]})^2 \times \sum_{k=1}^{n}(X[k,j] - \mu_{X[\_,j]})^2}}$$

where $X[k, i]$ is the kth element in the ith column of X, and $\mu_{X[\_,i]}$ is the mean of ith column of X.

(2) Extract eigenvalues λ_i, for $i = 1$, 2, ..., m, using the equation below, where I is an identity matrix:

$$|C - \lambda_i \times I| = 0$$

(3) Compute eigenvectors e_i, for $i = 1, 2, ..., m$, using the equation

$$(C - \lambda_i \times I) \cdot e_i = 0$$

(4) These are the so-called "principal components" of X. Rank the eigenvectors according to the amount of variation in the original data that they account for, which is given by

$$variance_i = \frac{\lambda_i}{\sum_{k=1}^{m} \lambda_k}$$

(5) Select as "feature generators" those eigenvectors that account for most of the variation in the data. In this step, enough eigenvectors that account for some percentage—*e.g.*, 95%—of the variance in the original data are chosen, while the rest are discarded.

Let g_1, ..., $g_{d'}$ be the selected feature generators, with $d' \leq d$. Given a data point X_i in the original d-dimensional space, it is mapped into a data point $Y_i = \langle X_i \cdot g_1$, ..., $X_i \cdot g_{d'} \rangle$ in the transformed d'-dimensional space. Analysis can then be carried out on the lower dimensional Y_i instead of the high-dimensional X_i.

Indeed, it can be proved that the representation given by PCA is an optimal linear dimension reduction technique in the mean-square sense. [399] It is worth noting that PCA is a unsupervised method since it makes no use of the class attribute. When dealing with k-valued discrete features, one can convert them to k binary features. Each of these new features has a "1" for every occurence of the corresponding kth value of the original discrete feature, and a "0" for all other values. Then the new binary features are treated as numeric features, to which above PCA steps can be applied.

1.12. *Use of Feature Selection in Bioinformatics*

The preceding subsections have provided a fairly comprehensive review of feature selection techniques. These techniques have been used as a key step in the handling of biomedical data of high dimension. For example, their use is prevalent in the analysis of microarray gene expression data. [33, 297, 493, 649] They have also been used in the prediction of molecular bioactivity in drug design. [881] More recently, they have even been used in the analysis of the context of protein translation initiation sites. [494, 515, 928]

Note that in choosing a statistical confidence level for t-test, Wilcoxon rank sum test, and other statistical measures presented in this section, one should be aware of the so-called "multiple comparisons problem," which is especially serious in a microarray setting. [634] In this setting, one typically has several thousand genes or features to choose from and hence performs the t-test, Wilcoxon rank sum test, *etc.* several thousand times. Suppose we perform the tests at the standard $p = 5\%$ confidence level. If there is truly no difference in the values of a feature between the two classes for any feature, one can still expect to observe $n \times 0.05$ significant features, where n is the total number of features under consideration.

A standard conservative solution to this problem is the Bonferroni correction. [711] This correction adjusts the cut-off for significance by dividing the desired confidence level by n; for example, for the 5% confidence level, the adjusted cut-off is $0.05/n$. Of course, this technique is applicable only to feature selection measures with known statistical distributions. For measures with unknown statistical distribution, permutation-based methods are typically used to estimate p values where needed. [297, 634]

2. Classification Methods

As mentioned earlier, classification is the process of finding models that separate two or more data classes. Putting it more prosaicly, we are given some samples of class $\mathcal{A}$ and some samples of class $\mathcal{B}$, can we use them as a basis to decide if a new unknown sample is in $\mathcal{A}$ or is in $\mathcal{B}$? If among all the features of the samples, there is one whose entropy measure is zero, then we can derive the obvious decision rule based on the cutting point T induced on the range of this feature.

Unfortunately, such zero-entropy features are rarely found in the more difficult classification problems. Hence, we often need to use multiple features in an integrated fashion to make prediction in these more difficult classification problems.

In this section, we describe a number of methods that have been used in the biomedical domain for this purpose, including decision tree induction methods, k nearest neighbours (k-NN), Bayesian methods, hidden Markov models (HMM), artificial neural networks (ANN), support vector machines (SVM), and PCL.

2.1. *Decision Trees*

The most popular group of classification techniques is the idea of decision tree induction. These techniques have an important advantage over machine learning methods such as k-NN, ANN, and SVM, in a qualitative dimension: rules produced by decision tree induction are easy to understand and hence can help greatly in appreciating the underlying reasons that separate the classes $\mathcal{A}$ and $\mathcal{B}$.

C4.5 is one of the most widely used decision tree based classifier.[693] C4.5 constructs a decision tree in a recursive process. The process starts by determining the feature that is most discriminatory with regard to the entire training data. Then it uses this feature to split the data into non-overlapping groups. Each group contains multi-class or single class samples, as categorized by this feature. Next, a significant feature of each of the groups is used to recursively partition them until a stopping criteria is satisfied. C4.5 uses the information gain ratio presented in Subsection 1.8 to determine which feature is most discriminatory at each step of its decision tree induction process.

Other algorithms for decision tree induction include CART,[101] ID3,[692] SLIQ,[558] FACT,[520] QUEST,[519] PUBLIC,[699] CHAID,[405] ID5,[852] SPRINT,[769] and BOAT.[281] This group of algorithms are most successful for analysis of clinical data and for diagnosis from clinical data. Some examples are diagnosis of central nervous system involvement in hematooncologic patients,[523] prediction of post-traumatic acute lung injury,[696] identification of acute cardiac ischemia,[763] prediction of neurobehavioral outcome in head-injury survivors,[822] and diagnosis of myoinvasion.[521] More recently, they have even been used to reconstruct

molecular networks from gene expression data.[783]

2.2. *Bayesian Methods*

Another important group of techniques[56, 214, 339, 393, 398, 478, 571, 731] are based on the Bayes theorem. The theorem states that

$$P(h|d) = \frac{P(d|h) \times P(h)}{P(d)}$$

where $P(h)$ is the prior probability that a hypothesis h holds, $P(d|h)$ is the probability of observing data d given some world that h holds, and $P(h|d)$ is the posterior probability that h holds given the observed data d.

Let H be all the possible classes. Then given a test instance with feature vector $\{f_1 = v_1, ..., f_n = v_n\}$, the most probable classification is given by

$$\text{argmax}_{h_j \in H} P(h_j | f_1 = v_1, \ldots, f_n = v_n)$$

Using the Bayes theorem, this is rewritten to

$$\text{argmax}_{h_j \in H} \frac{P(f_1 = v_1, \ldots, f_n = v_n | h_j) \times P(h_j)}{P(f_1 = v_1, \ldots, f_n = v_n)}$$

Since the denominator is independent of h_j, this can be simplified to

$$\text{argmax}_{h_j \in H} P(f_1 = v_1, \ldots, f_n = v_n | h_j) \times P(h_j)$$

However, estimating $P(f_1 = v_1, \ldots, f_n = v_n | h_j)$ accurately may not be feasible unless the training set is sufficiently large.

One of the popular ways to deal with the situation above is that adopted by the Naive Bayes classification method (NB).[214, 393] NB assumes that the effect of a feature value on a given class is independent of the values of other features. This assumption is called class conditional independence. It is made to simplify computation and it is in this sense that NB is considered to be "naive." Under this class conditional independence assumption,

$$\text{argmax}_{h_j \in H} P(f_1 = v_1, \ldots, f_n = v_n | h_j) \times P(h_j)$$

$$= \text{argmax}_{h_j \in H} \prod_i P(f_i = v_i | h_j) \times P(h_j)$$

where $P(h_j)$ and $P(f_i = v_i | h_j)$ can often be estimated reliably from typical training sets.

Some example applications of Bayesian classifiers in the biomedical context are mapping of locus controlling a genetic trait,[288] screening for macromolecular crystallization,[348] classification of cNMP-binding proteins,[554] prediction of carboplatin exposure,[375] prediction of prostate cancer recurrence,[198]

prognosis of femoral neck fracture recovery,[461] prediction of protein secondary structure,[34, 420, 808] and reconstruction of molecular networks.[263]

2.3. *Hidden Markov Models*

Related to Bayesian classifiers are hidden Markov models (HMM).[56, 215, 219, 455] A HMM is a stochastic generative model for sequences defined by a finite set S of states, a finite alphabet A of symbols, a transition probability matrix T, and an emission probability matrix E. The system moves from state to state according to T while emitting symbols according to E. In an n-th order HMM, the matrices T and E depend on all n previous states.

HMMs have been applied to a variety of problems in sequence analysis, including protein family classification and prediction,[55, 66, 456] tRNA detection in genomic sequences,[525] methylation guide snoRNA screening,[526] gene finding and gene structure prediction in DNA sequences,[54, 94, 95, 455, 741] protein secondary structure modeling,[256] and promoter recognition.[657, 915]

2.4. *Artificial Neural Networks*

Artificial neural networks (ANN)[56, 141, 729] are another important approach to classification that have high tolerance to noisy data. ANN are networks of highly interconnected "neural computing elements" that have the ability to respond to input stimuli and to learn to adapt to the environment.

Although the architecture of different ANN can differ in several characteristic ways, a typical ANN computing element is a comparator that produces an output when the cumulative effect of the input stimuli exceeds a threshold value. Part I of Figure 4 depicts a single computing element. Each input x_i has an associated weight w_i, which acts to either increase or decrease the input signal to the computing element. The computing element behaves as a monotonic function f producing an output $y = f(net)$, where net is the cummulative input stimuli to the neuron. The number net is usually defined as the weighted sum of the inputs:

$$net = \sum_i x_i \times w_i$$

and the function f is usually defined as a sigmoid:

$$f(net) = \frac{1}{1 + e^{-net}}$$

Such computing elements can be connected in multiple layers into the so-called artificial neural network. Part II of Figure 4 depicts a fully-connected feed-forward artificial neural network with two layers of computing elements. The out-

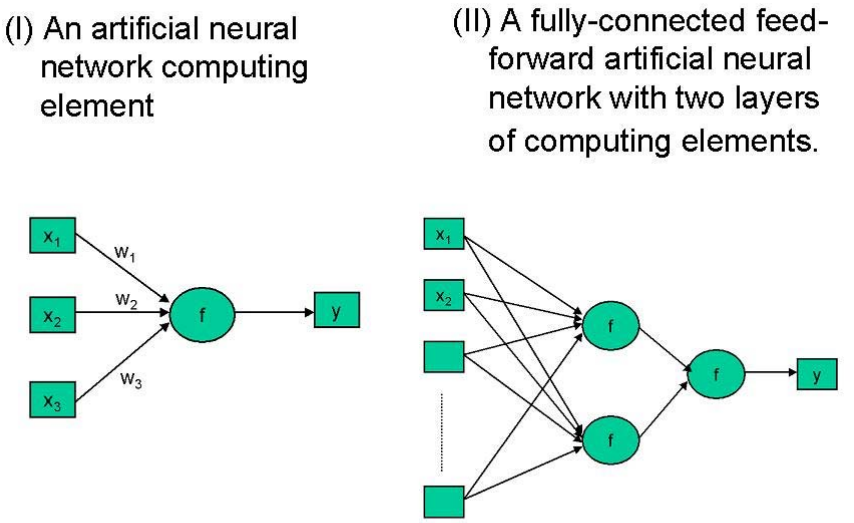

Fig. 4. (I) An artificial neural network computing element. (II) A fully-connected feed-forward artificial neural network with two layers of such computing elements.

put from the two hidden computing elements in the first layer—the so-called hidden layer—are fed as inputs into the computing element at the second layer—the so-called output layer.

The network is used for classification decision as follows. The inputs x_i are fed into the network. Each computing element at the first layer produces its corresponding output, which is fed as input to the computing elements at the next layer. This process continues until an output is produced at the computing element at the final layer.

What makes the ANN work is in how the weights on the links between the inputs and computing elements are chosen. These weights are typically learned from training data using the error back-propagation method. [7,29]

Let us to introduce some notations before we describe the error back-propagation method. Let v_{ij} denote the weight on the link between x_i and the j-th computing element of the first layer—*i.e.*, the hidden layer—in the artificial neural network. Let w_j denote the weight on the link between the j-th computing element of the first layer and the computing element of the last layer—*i.e.*, the output layer. Let z_j denote the output produced by the j-th computing element of

the first layer. Then the output y produced by the artificial neural network for a given training sample is given by

$$y = f\left(\sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)\right)$$

For a given training sample, this y may differ from the targeted output t for that particular training sample by an error amount Δ. The crux of ANN training is in how to propagate this error backwards and to use it to adapt the ANN. This is accomplished by adjusting the weights in proportion to the negative of the error gradient. For mathematical convenient, the squared error E can be defined as

$$E = \frac{(t-y)^2}{2}$$

In finding an expression for the weight adjustment, we must differentiate E with respect to weights v_{ij} and w_j to obtain the error gradients for these weights. Applying the chain rule a couple of times and recalling the definitions of y, z_j, E, and f; we derive

$$\begin{aligned}
\frac{\delta E}{\delta w_j} &= \frac{\delta E}{\delta \sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)} \times \frac{\delta \sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)}{\delta w_j} \\
&= \frac{\delta E}{\delta \sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)} \times f\left(\sum_i x_i \times v_{ij}\right) \\
&= \frac{\delta E}{\delta y} \times \frac{\delta y}{\delta \sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)} \times f\left(\sum_i x_i \times v_{ij}\right) \\
&= (t-y) \times f'\left(\sum_j w_j \times f\left(\sum_i x_i \times v_{ij}\right)\right) \times f\left(\sum_i x_i \times v_{ij}\right) \\
&= (t-y) \times f'\left(\sum_j w_j \times z_j\right) \times z_j \\
&= \Delta \times y \times (1-y) \times z_j
\end{aligned}$$

The last step follows because f is a sigmoid and thus

$$f'(x) = f(x) \times (1 - f(x))$$

Then the adjustment Δ_{w_j} to w_j is defined as below, where η is a fixed learning rate.

$$\Delta_{w_j} = -\eta \times \frac{\delta E}{\delta w_j} = -\eta \times \Delta \times y \times (1-y) \times z_j$$

However, for the weights v_{ij}, we do not have a targeted output to compute errors. So we have to use the errors Δ_{w_j} as surrogates and apply a similar derivation to obtain

$$\Delta_{v_{ij}} = -\eta \times \Delta_{w_j} \times z_j \times (1 - z_j) \times x_i$$

The above derivation provides the adjustments on the weights for one training sample. An "epoch" in the training process is a complete iteration through all training samples. At the end of an epoch, we compute the total error of the epoch as the sum of the squares of the Δ of each sample. If this total error is sufficiently small, the training process terminates. If the number of epoch exceeds some predefined limit, the training process also terminates.

A number of variations are possible. The most obvious are in the configuration of the ANN in terms of the number of layers and in terms of how the computing elements are connected. Another variation is the definition of the function f. For example, the "tansig" function $tanh(net) = (e^{net} - e^{-net})/(e^{net} + e^{-net})$ is another popular choice for f. There are also variations in the method used for propagating errors backwards to adapt the ANN, such as some forms of regularization to prevent overfitting of the ANN to the training data.[82]

Successful applications of artificial neural networks in the biomedical context include protein secondary structure prediction,[687,713,725] signal peptide prediction,[164,231,611] gene finding and gene structure prediction,[782,848] T-cell epitope prediction,[362] RNA secondary structure prediction,[800] toxicity prediction,[120] disease diagnosis and outcome prediction,[761,847,865] gene expression analysis,[430] as well as protein translation initiation site recognition.[335,658]

2.5. *Support Vector Machines*

Let $\phi : D \to D'$ be a function that embeds a training sample in the input space D to a higher dimensional embedding space D'. A kernel $k(x, y) = (\phi(x) \cdot \phi(y))$ is an inner product in D'. It can often be computed efficiently. By formulating a learning algorithm in a way that only requires knowledge of the inner products between points in the embedding space, one can use kernels without explicitly performing the embedding ϕ.

A support vector machine (SVM) is one such learning algorithm. It first embeds its data into a suitable space and then learns a discriminant function to separate the data with a hyperplane that has maximum margin from a small number of critical boundary samples from each class.[124,855] A SVM's discriminant function $G(T)$ for a test sample T is a linear combination of kernels computed at the

training data points and is constructed as

$$G(T) = sign\left(\sum_i \alpha[i] \times Y[i] \times k(T, X[i]) + b\right)$$

where $X[\_]$ are the training data points, $Y[\_]$ are the class labels (which are assumed to have been mapped to 1 or –1) of these data points, $k(\cdot,\cdot)$ is the kernel function, and b and $\alpha[\_]$ are parameters to be learned from the training data.

As it turns out, the training of a SVM is a quadratic programming problem on maximizing the Lagrangian dual objective function

$$\max_\alpha W(\alpha) = \sum_{i=1}^{l} \alpha[i] - \frac{1}{2} \times \sum_{i=1}^{l}\sum_{j=1}^{l} \alpha[i] \times \alpha[j] \times Y[i] \times Y[j] \times k(X[i], X[j])$$

subject to the constraint that $\forall i.0 \leq \alpha[i] \leq C$ and $\sum_{i=1}^{l} \alpha[i] \times Y[i] = 0$. Here, C is a bound on errors, $X[i]$ and $X[j]$ are the ith and jth training samples, $Y[i]$ and $Y[j]$ are the corresponding class labels (which are mapped to 1 and –1), and $k(\cdot,\cdot)$ is the kernel function. The kernel function normally used is a polynomial kernel

$$k(X, X') = (X \cdot X')^d = \sum_{i_1} \ldots \sum_{i_d} X[i_1] \times \ldots \times X[i_d] \times X'[i_1] \times \ldots \times X'[i_d]$$

In the popular WEKA data mining software package,[888] the quadratic programming problem above is solved by sequential minimal optimization.[668] Once the optimal solution $\alpha[\_]$ is obtained from solving the quadratic programming problem above, we can substitute it into the SVM decision function $G(T) = sign(\sum_i \alpha[i] \times Y[i] \times k(T, X[i]) + b)$ given earlier.

It remains now to find the threshold b. We can estimate b using the fact that the optimal solution $\alpha[\_]$ must satisfy the so-called Karush-Kuhn-Tucker conditions.[328, 668, 757] In particular, for each $\alpha[j] > 0$, $Y[j] \times G(X[j]) = 1$. Hence $Y[j] = sign(\sum_i \alpha[i] \times Y[i] \times k(X[j], X[i]) + b)$. So we estimate b by averaging $Y[i] - \sum_i \alpha[i] \times Y[i] \times k(X[j], X[i])$ for all $\alpha[j] > 0$.

A SVM is largely characterized by the choice of its kernel function. Thus SVMs connect the problem they are designed for to a large body of existing research on kernel-based methods.[124, 700, 855] Besides the polynomial kernel function $k(x, y) = (x \cdot y)^d$, other examples of kernel function include the Gaussian radial basis kernel function,[63, 855] sigmoid kernel function,[757] B_n-spline kernel function,[757] locality-improved kernel function,[940] and so on.

Some recent applications of SVM in the biomedical context include protein homology detection,[388] microarray gene expression data classification,[111] breast

cancer diagnosis,[264, 539] protein translation initiation site recognition,[940, 941] as well as prediction of molecular bioactivity in drug design.[881]

2.6. *Prediction by Collective Likelihood of Emerging Patterns*

Prediction by collective likelihood of emerging patterns (PCL)[492, 497] is a classification method that we have been developing during the last couple of years. This method focuses on (a) fast techniques for identifying patterns whose frequencies in two classes differ by a large ratio,[207] which are the so-called emerging patterns; and on (b) combining these patterns to make decision. Note that a pattern is still emerging if its frequencies are as low as 1% in one class and 0.1% in another class, because the ratio indicates a 10 times difference. However, for the purpose of PCL, we use only emerging patterns that are most general and have infinite ratio. That is, we use only emerging patterns that occur in one class of data but not the other class and that are as short as possible.

Basically, the PCL classifier has two phases. Given two training datasets $D^{\mathcal{A}}$ (instances of class $\mathcal{A}$) and $D^{\mathcal{B}}$ (instances of class $\mathcal{B}$) and a test sample T, PCL first discovers two groups of most general emerging patterns from $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$. Denote the most general emerging patterns of $D^{\mathcal{A}}$ as, $EP_1^{\mathcal{A}}$, $EP_2^{\mathcal{A}}$, $\cdots$, $EP_i^{\mathcal{A}}$, in descending order of frequency. Denote the most general emerging patterns of $D^{\mathcal{B}}$ as $EP_1^{\mathcal{B}}$, $EP_2^{\mathcal{B}}$, $\cdots$, $EP_j^{\mathcal{B}}$, in descending order of frequency.

Suppose the test sample T contains these most general emerging patterns of $D^{\mathcal{A}}$: $EP_{i_1}^{\mathcal{A}}$, $EP_{i_2}^{\mathcal{A}}$, $\cdots$, $EP_{i_x}^{\mathcal{A}}$, $i_1 < i_2 < \cdots < i_x \leq i$, and these most general emerging patterns of $D^{\mathcal{B}}$: $EP_{j_1}^{\mathcal{B}}$, $EP_{j_2}^{\mathcal{B}}$, $\cdots$, $EP_{j_y}^{\mathcal{B}}$, $j_1 < j_2 < \cdots < j_y \leq j$. The next step is to calculate two scores for predicting the class label of T. Suppose we use k ($k \ll i$ and $k \ll j$) top-ranked most general emerging patterns of $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$. Then we define the score of T in the $D^{\mathcal{A}}$ class as

$$score(T, D^{\mathcal{A}}) = \sum_{m=1}^{k} \frac{frequency(EP_{i_m}^{\mathcal{A}})}{frequency(EP_m^{\mathcal{A}})},$$

and the score in the $D^{\mathcal{B}}$ class is similarly defined in terms of $EP_{j_m}^{\mathcal{B}}$ and $EP_m^{\mathcal{B}}$. If $score(T, D^{\mathcal{A}}) > score(T, D^{\mathcal{B}})$, then T is predicted as the class of $D^{\mathcal{A}}$. Otherwise it is predicted as the class of $D^{\mathcal{B}}$. We use the size of $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$ to break tie.

Recall that PCL uses emerging patterns that have infinite ratio and are most general. That is, given two datasets $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$, an emerging pattern—to be used by PCL—is a pattern such that its frequency in $D^{\mathcal{A}}$ (or $D^{\mathcal{B}}$) is non-zero but in the other dataset is zero (*i.e.*, infinite ratio between its support in $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$), and none of its proper subpattern is an emerging pattern (*i.e.*, most general). We can immediately see that if $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$ has n Boolean attributes, then there are 2^n

possible patterns. Hence a naive method to extract emerging patterns requires 2^n scans of the dataset. A more efficient method for extracting emerging patterns is therefore crucial to the operation of PCL.

We briefly discuss here an approach for developing more practical algorithms for finding such emerging patterns. Let us first recall the theoretical observation of Dong and Li[207] that the emerging patterns from $D^{\mathcal{A}}$ to $D^{\mathcal{B}}$—all of them together, not just most general the ones—form a convex space. Now, it is known that a convex space C can be represented by a pair of borders $\langle L, R\rangle$, so that (a) L is anti-chain, (b) R is anti-chain, (c) each $X \in L$ is more general than some $Z \in R$, (d) each $Z \in R$ is more specific than some $X \in L$, and (e) $C = \{Y \mid \exists X \in L, \exists Z \in R, X \subseteq Y \subseteq Z\}$. Actually, L are the most general patterns in C, and R the most specific patterns in C. We write $[L, R]$ for C.

Dong and Li[207] show that: if $D^{\mathcal{A}}$ and $D^{\mathcal{B}}$ have no duplicate and no intersection and are of the same dimension, then the set of emerging patterns to $D^{\mathcal{A}}$ from $D^{\mathcal{B}}$ is given by $[\{\{\}\}, D^{\mathcal{A}}] - [\{\{\}\}, D^{\mathcal{B}}]$. Having thus reduced emerging patterns to this "border formulation," we derive a more efficient approach to discover them.

The main trick is as follows. Let $D^{\mathcal{A}} = \{A_1, \ldots, A_n\}$ and $D^{\mathcal{B}} = \{B_1, \ldots, B_m\}$. Then

$$\begin{aligned} & [\{\{\}\}, D^{\mathcal{A}}] - [\{\{\}\}, D^{\mathcal{B}}] \\ = & [\{\{\}\}, \{A_1, \ldots, A_n\}] - [\{\{\}\}, \{B_1, \ldots, B_m\}] \\ = & [L, \{A_1, \ldots, A_n\}] \end{aligned}$$

where

$$\begin{aligned} L = & \min(\bigcup_i^n \{\{s_1, \ldots, s_m\} | s_j \in A_i - B_j, 1 \le j \le m\}) \\ = & \bigcup_i^n \min\{\{s_1, \ldots, s_m\} | s_j \in A_i - B_j, 1 \le j \le m\} \end{aligned}$$

which we know how to optimize very well[207, 208, 935] using novel border-based algorithms and constraint-based algorithms based on set enumeration tree.[732]

A number of other classification methods based on the idea of emerging patterns exist. They include CAEP,[209] which uses all of the most-general emerging patterns with ratio exceeding a user-specified threshold; DeEPs,[491] which uses emerging patterns in an instance-based manner; as well as techniques that score emerging patterns in *ad hoc* ways.[498]

Any way, although the PCL classifier is a very recent development, it has already proved to be a good tool for analysing biomedical data. Examples include gene expression data, proteomic data, and translation initiation sites.[492–494, 497, 499, 918]

3. Association Rules Mining Algorithms

A major data mining task is to discover dependencies among data. Association rules[10] are proposed to achieve this goal. In the context of supermarket business data, association rules are often used to describe dependencies such as the presence of some items in a transaction implies, with a certain probability, the presence of other items in the same transaction. An example[10] of such an association rule is the statement that 90% of transactions involving the purchase of `bread` and `butter` also involve the purchase of `milk`. In this section, we first provide some preliminary definitions. Then we outline two algorithms for mining association rules, *viz.* Apriori[12] and Max-Miner.[70]

3.1. *Association Rules*

Recall from Subsection 1.1 that a data point in a feature space can be thought of as an itemset $\{f_1 = v_1, ..., f_n = v_n\}$. It is a convention of the data mining community to write an itemset as $\{f'_1, ..., f'_m\}$ if each v_i is either 1 or 0 and $\{f'_1, ..., f'_m\} = \{f_i |\ v_i = 1, 1 \leq i \leq n\}$. Under this convention, the itemset is also called a "transaction." Observe that transactions contain only those items whose feature values are 1 rather than those items whose feature values are 0. In this section, we use the term "transaction" instead of "data point." Incidentally, an itemset consisting of k items is called a k-itemset; the number k is called the "length" or "cardinality" of the itemset.

We say that a transaction T contains an itemset X if $X \subseteq T$. We also say that an itemset X occurs in a transaction T if $X \subseteq T$. Let a database $\mathcal{D}$ of transactions and an itemset X be given. Then the support of X in $\mathcal{D}$, denoted $supp^{\mathcal{D}}(X)$, is the percentage of transactions in $\mathcal{D}$ that contain X. The count of X in $\mathcal{D}$, denoted $count^{\mathcal{D}}(X)$, is the number of transactions in $\mathcal{D}$ that contain X. Observe that

$$supp^{\mathcal{D}}(X) = \frac{count^{\mathcal{D}}(X)}{n^{\mathcal{D}}}$$

where $n^{\mathcal{D}}$ is the number of transactions in $\mathcal{D}$. Then given a threshold δ between 0 and 1, an itemset X is said to be a large itemset or a frequent itemset if $supp^{\mathcal{D}}(X) \geq \delta$.

An association rule is an implication of the form $X \rightarrow^{\mathcal{D}} Y$, where X and Y are two itemsets in a dataset $\mathcal{D}$ and $X \cap Y = \{\}$. We often omit the superscript $\mathcal{D}$ if the dataset is understood. The itemset X is called the antecedent of the rule. The itemset Y is called the consequent of the rule. We define the support of the rule as the percentage of the transactions in $\mathcal{D}$ that contain $X \cup Y$. We also define the confidence of the rule as the percentage of the transactions in $\mathcal{D}$ containing X

that also contain Y. Note that the support of the rule $X \rightarrow^{\mathcal{D}} Y$ is $supp^{\mathcal{D}}(X \cup Y)$ and the confidence is $count^{\mathcal{D}}(X \cup Y)/count^{\mathcal{D}}(X)$.

The problem of mining association rules is that of how to generate all association rules that have support and confidence greater than or equal to a user-specified minimum support (*minsup*) and a minimum confidence (*minconf*). [10] This problem is solved by decomposing into two sub-problems:

(1) Generate all large itemsets that satisfy the support threshold *minsup*.
(2) For a given large itemset $X = \{f_1, f_2, ..., f_k\}$, generate all association rules and output only those rules that satisfy the confidence threshold *minconf*.

The second sub-problem is simple and straightforward to solve. The first sub-problem is the key efficiency issue of mining association rules, *viz.* to discover all large itemsets whose supports exceeds a given threshold.

A naive approach to solving this problem is to generate all possible itemsets in the dataset $\mathcal{D}$ and then check whether their supports meet the threshold. Obviously, this task rapidly becomes impossible as the number of features increases. This happens because the number of the candidate itemsets increases exponentially when the number of features increases. Fortunately, this problem has been extensively studied in both the database and data mining communities.[10–12, 70, 105, 106, 155, 319, 320, 436, 481, 540, 646, 748, 794, 832, 933] There are two now well-known efficient algorithms, the Apriori algorithm [12] and the Max-Miner algorithm,[70] that partially overcome the difficulty in the naive algorithm. We describe these two algorithms in the next two subsections.

3.2. *The Apriori Algorithm*

Given a database $\mathcal{D}$ and a support threshold δ, the collection of the large itemsets in $\mathcal{D}$ is the union of all large 1-itemsets, all large 2-itemsets, ..., and all large m-itemsets, where m is the longest length of the large itemsets in $\mathcal{D}$. The Apriori algorithm sequentially outputs the large k-itemsets with k from 1 to m. So, the Apriori algorithm is also called a level-wise algorithm.

The basic idea used in the Apriori algorithm is that any subset of a large itemset is large. This property is called the *a priori* property. It is used in the algorithm to narrow search spaces, generating a strict number of large itemset candidates which is much smaller than the number of the candidates generated by the naive method. In other words, if all the large k-itemsets are known, the Apriori algorithm generates only those $(k+1)$-itemsets as candidates whose immediate proper subsets (having k items) must all be large. So, the Apriori algorithm does not generate any of those $(k+1)$-itemsets as a candidate for which there exists its an immediate

proper subset which is not large.

Briefly, the algorithm comprises the three main steps below, with the iteration parameter k initialized to 1. These three steps are repeated until no new large itemsets are found:

(1) Generate a collection of large itemset candidates, each having k items,
(2) Scan the database $\mathcal{D}$ and compute the supports of the candidates.
(3) $k := k + 1$.

The efficiency Step 1 is key to the efficiency of the whole algorithm, as generating more unnecessary candidates can increase its complexity. Agrawal and Srikant[12] propose the APRIORI-GEN function that makes use of the *a priori* property for candidate generation. We use an example[12] to illustrate the main ideas in the APRIORI-GEN function. Let the collection $\mathcal{D}_3$ of all the large 3-itemsets in some database $\mathcal{D}$ be $\{\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{1,3,5\}, \{2,3,4\}\}$. After a joint sub-step in the APRIORI-GEN function, the candidate collection $\mathcal{D}_4$ is $\{\{1,2,3,4\}, \{1,3,4,5\}\}$. After a pruning sub-step, the itemset $\{1,3,4,5\}$ is removed from $\mathcal{D}_4$ because the itemset $\{1,4,5\}$ is not in $\mathcal{D}_3$. As a result, only the itemset $\{1,2,3,4\}$ is a candidate and needs support calculation in Step 2.

The Apriori algorithm achieves good performance by reducing the size of candidate itemsets. However, in some situations where there exist long frequent itemsets (*e.g.*, an itemset with the cardinality of 30), or where a low support threshold is required, the Apriori algorithm still suffers from heavy computational costs. Additional details of the Apriori algorithm and discussions on its effficiency and scalability can be found in Agrawal and Srikant,[12] Bayardo,[70] and Han *et al.*[320]

3.3. *The Max-Miner Algorithm*

The Apriori algorithm explicitly outputs all large itemsets in a level-wise manner. It sequentially finds the large k-itemsets, $1 \leq k \leq m$, where m is the longest length of the large itemsets. In contrast to the Apriori algorithm, the Max-Miner algorithm[70] only outputs a subset of the collection of all large itemsets with respect to a given support threshold in a database. Specifically, Max-Miner outputs only those large itemsets whose proper supersets are not large. These large itemsets are called *maximal large itemsets*. They can be imagined as a frontier boundary that separates the large itemsets from the non-large itemsets. Since any large itemset is a subset of a maximal large itemset, Max-Miner's output implicitly and concisely represents all large itemsets.

Max-Miner has been shown to perform two or more orders of magnitude better than Apriori on some data sets.[70] This is particularly true when the support

threshold is set low, and the data sets are large and high-dimensional. There are two basic ideas behind Max-Miner:

(1) Superset-frequency pruning, which works as follows. Suppose an itemset is known to be large or frequent. Then its subsets must be large or frequent also. Hence, there is no need to generate its subsets for support calculation.
(2) Subset-infrequency pruning, which works as follows. Suppose an itemset is known as non-large or infrequent. Then its supersets must be non-large. Hence, there is no need to generate its supersets for support calculation.

Only the latter pruning technique is used in the Apriori algorithm or its variants.

Max-Miner contains some provision for looking ahead in order to quickly identify long frequent itemsets and short infrequent itemsets to conduct superset-frequency pruning and subset-infrequency pruning. As a result, Max-Miner is highly efficient in comparison to the Apriori algorithm.

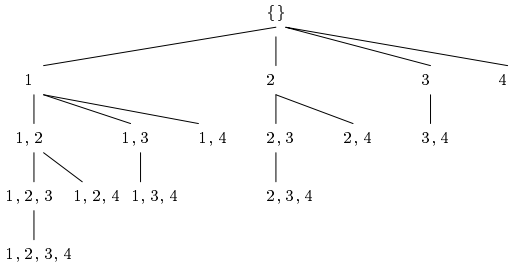

Fig. 5. A complete set-enumeration tree over four items.

The two pruning strategies of Max-Miner are implemented using the framework of set-enumeration trees[732] by incorporating some heuristic. A simple set-enumeration is shown in Figure 5. See Bayardo[70] for more details of Max-Miner, and Bayardo *et al.*[69] for its further refinement. Some methods that are closely related to Max-Miner are Pincer-Search by Lin and Kedem,[508] and MaxEclat and MaxClique by Zaki *et al.*[925]

3.4. *Use of Association Rules in Bioinformatics*

A number of studies[176, 643, 747] have been conducted to make use of the concept of association rules for mining bio-medical data to find interesting rules. We discuss two of them. One discovers association rules from protein-protein interaction

data.[643] The other finds expression associations from gene expression profiling data.[176] Both of the studies have revealed interesting patterns or rules. Some of these rules have been confirmed by work reported in biomedical literature, and some are considered to be new hypothesis worthy of wet-experimental validation.

Protein-protein interactions are fundamental biochemical reactions in organisms. It is important to understand general rules about proteins interacting with each other, such as "the protein having a feature A interacts with the protein having the feature B", "this domain interacts with that domain," and so on. Association rules can capture such knowledge patterns. Oyama *et al.* [643] propose a method that is aimed to reveal what types of proteins tend to interact with each other. For example, a popular interaction rule that "a SH3 domain binds to a proline-rich region" is detected by this method.

Oyama *et al.*[643] use 4307 pairs of yeast interaction proteins from four source databases. Functional, primary structural, and other characteristics are used as features to describe the proteins. The total of 5241 features are categorized into seven types, *viz.* YPD categories, EC numbers, SWISS-PROT/PIR keywords, PROSITE motifs, Bias of the amino acids, Segment clusters, and Amino acid patterns. Under $minsup = 9\%$ and $minconf = 75\%$, Oyama *et al.*[643] have discovered a total of 6367 rules. Instead of using only support and confidence, they have also proposed a new scoring measure to indicate the validity of the rules. The number of positive interactions and the number of proteins concerned with the positive interactions are placed at high priority considerations in this scoring measure.

Gene expression profiling data have been widely used to understand functions of genes, biological networks, and cellular states. Creighton and Hanash [176] proposed a method to analyse gene expression data using association rules, aimed at revealing biologically relevant associations between different genes or between environmental effects and gene expression. In this analysis, there are 6316 transcripts which are used as features. These transcripts correspond to 300 diverse mutations and chemical treatments in yeast. The values of these features are their expression levels under different experiments. The expression range of each feature (transcript) is discretized into three categorical values: up, down, and neither-up-nor-down. An expression value greater than 0.2 for the log base 10 of the fold change is categorized as up; a value less than -0.2 as down; all other values as neither-up-nor-down.

Creighton and Hanash[176] use the Apriori algorithm in the mining of the association rules. The minimum support for mining those frequent itemsets is set as 10%. The minimum confidence for those association rules is set as 80%. To avoid the vast majority of redundant rules and also to alleviate the complexity of the mining problem, they suggest an additional criteria besides a minimum support

to narrow the seach space of candidate frequent itemsets. The additional criteria is that the method generates only those association rules with the left-hand-side itemset containing a single item, ignoring frequent itemsets that cannot form such a rule. Mining expression data using association rules is claimed to be more useful than clustering methods to uncover gene networks. Association rules discovered from gene expression data can also be used to help relate the expression of genes to their cellular environment.

4. Clustering Methods

Clustering is the process of grouping a set of objects into classes or clusters so that objects within a cluster have high similarity in comparison to one another, but are dissimilar to objects in another clusters. Unlike classifier induction discussed in Section 2, clustering does not rely on predefined classes and labeled training examples to guide the machine learning process. Hence, it is also called unsupervised learning. Clustering algorithms can be separated into four general categories, *viz.* partitioning methods, hierarchical methods, density-based methods and grid-based methods. Of the four, partitioning and hierarchical methods are applicable to both spatial and categorical data while density- and grid-based methods are applicable only to spatial data. As such, this section focuses on the first categories of two methods.

4.1. *Factors Affecting Clustering*

To apply clustering in any domain successfully, the following factors must typically be considered:

(1) type of data,
(2) similarity function, and
(3) goodness function.

It is easy to see that the type of data to be clustered affects the clustering algorithms being applied. For example, clustering a set of DNA sequences is different from clustering a set of gene expression profiles. A useful way for categorizing the data types is to separate them into spatial vs. categorical data. Spatial data are data objects which are mapped into a coordinated space, *e.g.*, the 3-dimensional location of protein molecules. Categorical data are data objects which cannot be represented as objects in a spatial coordinate system, *e.g.*, a set of medical documents. Generally, it is easier to develop efficient clustering algorithms for spatial data through the use of spatial inferences. The clustering of categorical data is

more difficult as explicit computation must usually be made for any pair of data objects in order to judge their similarity based on some similarity function.

Since the aim of clustering is to group similar objects together, it is a natural question to ask when are two objects deemed to be similar. The answer comes in the form of a similarity function that is defined by the user based on the application on hand. For spatial data, the similarity function is usually computed from the coordinates of the data objects, such as the Euclidean distance. It is therefore generally easy to estimate the similarity between two groups of objects by picking a representative for each group—such as their mean or their bounding boxes which can be easily computed—and making spatial inference based on the distance between the two representatives. The similarity of two data objects can thus be approximated without knowing their exact coordinate.

Unlike spatial data, the similarity between categorical data is usually more difficult to approximate. The inherent difficulty come from the fact that a good representative for a group of categorical data is usually harder to determine. For example, if we use edit distance as a similarity measurement, finding a good representive for a group of sequences involves the use of multiple alignment, an operation which requires expensive dynamic programming. The other alternative is to pick one of the sequences from the group that is the most similar to all the other sequences, resulting in the need for an all-pairs comparison.

Without any good way to represent a set of data objects, efficient clustering of categorical data is difficult and approximating the similarity between two groups of object becomes computationally intensive too. One usual technique to solve this problem is to derive some transformation that maps categorical data into spatial coordinates such that objects that are similar are mapped into a spatially close region. The objects can then be clustered using the spatial distance as a similarity function instead.

Finally, a goodness function is usually derived to measure the quality of the clustering. This is again application dependent. In general, a goodness function is an aggregate function that is used to gauge the overall similarity between data objects in the clusters. For example, in k-means clustering,[537] the squared distance to the cluster center for each point is summed up as a measure of goodness. In other cases, a test of good fit is done according to some statistical model—*e.g.*, Gaussian—in order to judge the quality of the clusters.

As mentioned earlier, partitioning and hierarchical methods of clustering are applicable to both spatial and categorical data. We discuss them in the next two subsections.

4.2. *Partitioning Methods*

Partitioning algorithms had long been popular clustering algorithms before the emergence of data mining. Given a set D of n objects in a d-dimensional space and an input parameter k, a partitioning algorithm organizes the objects into k clusters such that the total deviation of each object from its cluster center or from a cluster distribution is minimized. The deviation of a point can be computed differently in different algorithms and is what we earlier called the goodness function.

Three well-known partitioning algorithms are: k-means,[537] Expectation Maximization (EM),[98, 197, 921] and k-medoids.[421] The three algorithms have different ways of representing their clusters. The k-means algorithm uses the centroid (or the mean) of the objects in the cluster as the cluster center. The k-medoids algorithm uses the most centrally located object in the cluster. Unlike k-means and k-medoids, EM clustering uses a distribution consisting of a mean and a $d \times d$ covariance matrix to represent each cluster. Instead of assigning each object to a dedicated cluster, EM clustering assigns each object to a cluster according to a probability of membership that is computed from the distribution of each cluster. In this way, each object has a certain probability of belonging to each cluster, making EM clustering a fuzzy partitioning technique.

Despite their differences in the representation of the clusters, the three partitioning algorithms share the same general approach when computing their solutions. To see this similarity, we first observe that the three algorithms are effectively trying to find the k centers or distributions that will optimize the goodness function. Once the optimal k centers or distributions are found, the membership of the n objects within the k clusters are automatically determined. However, the problem of finding the globally optimal k centers or k distributions is known to be NP-hard.[273] Hence, the three algorithms adopt an iterative relocation technique that finds k locally optimal centers. This technique is shown in Figure 6. The three algorithms differ essentially in their goodness function and in the way they handle Steps 2 and 3 of the iterative relocation technique given in the figure. The general weaknesses of partitioning-based algorithms include a requirement to specify the parameter k and their inability to find arbitrarily shaped clusters.

4.3. *Hierarchical Methods*

A hierarchical method creates a hierarchical decomposition of the given set of data objects forming a dendrogram—a tree that splits the database recursively into smaller subsets. The dendrogram can be formed in two ways: "bottom-up" or "top-down." The "bottom-up" approach, also called the "agglomerative" approach, starts with each object forming a separate group. It successively merges

Input:	The number of clusters, k, and a database containing n objects.
Output:	A set of k clusters that minimizes a criterion function E.
Method:	
	(1) Arbitrarily choose k centers/distributions as the initial solution;
	(2) (re)compute membership the objects according to present solution.
	(3) Update some/all cluster centers/distributions according to new memberships of the objects;
	(4) Goto Step 2 if the value of E changes; otherwise, end.

Fig. 6. An outline of the iterative relocation technique used generally by partitioning methods of clustering.

the objects or groups according to some measures like the distance between the two centers of two groups and this is done until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition holds. The top-down, also called the "divisive" approach, starts with all the objects in the same cluster. In each successive iteration, a cluster is split into smaller clusters according to some measures until eventually each object is in one cluster, or until a termination condition holds.

AGNES and DIANA are two hierarchical clustering algorithms.[421] AGNES (AGglomerative NESting) is a bottom-up algorithm which starts by placing each object in its own cluster and then merging these atomic clusters into larger and larger clusters, until all of the objects are in a single cluster or until a certain termination condition is satisfied. DIANA (DIvisia ANAlysis), on the other hand, adopts a top-down strategy that does the reverse of AGNES by starting with all objects in one cluster. It subdivides the cluster into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions, such as a desired number of clusters is obtained or the distance between the two closest clusters is above a certain threshold distance.

In either AGNES or DIANA, one can specify the desired number of clusters as a termination condition. The widely used measures for distance between clusters

are as follows, where m_i is the mean for cluster C_i, n_i is the number of objects in C_i, and $|p - p'|$ is the distance between two objects or points p and p'.

$$d_{min}(C_i, C_j) = min_{p \in C_i, p' \in C_j} |p - p'|$$

$$d_{mean}(C_i, C_j) = |m_i - m_j|$$

$$d_{avg}(C_i, C_j) = \frac{\Sigma_{p \in C_i} \Sigma_{p' \in C_j} |p - p'|}{n_i \times n_j}$$

$$d_{max}(C_i, C_j) = max_{p \in Ci, p' \in C_j} |p - p'|$$

The two algorithms, AGNES and DIANA, often encounter difficulties regarding the selection of merge or split points. Such a decision is critical because once a group of objects are merged or split, the process at the next step will operate on the newly generated clusters. They can neither undo what was done previously, nor perform object swapping between clusters. Thus merge or split decisions, if not well chosen at some step, may lead to low quality clusters. Moreover, the method does not scale well since the decision of merge or split needs to examine and evaluate a good many number of objects or clusters.

To enhance the effectiveness of hierarchical clustering, recent methods have adopted one of the following two approaches. The first approach represented by algorithms like CURE[303] and CHAMELEON[418] utilizes a more complex principle when splitting or merging the clusters. Although the splitting and merging of clusters are still irreversible in this approach, less errors are made because a better method is used for merging or splitting. The second approach represented by algorithms like BIRCH[934] is to obtain an initial result by using a hierarchical agglomerative algorithm and then refining the result using iterative relocation.

4.4. *Use of Clustering in Bioinformatics*

Recent uses of clustering techniques in bioinformatics are mostly in aspects of gene expression analysis. They are amongst the earliest techniques used for disease subtype diagnosis with microarrays, though classification-based approaches have now largely replaced clustering-based approaches to this problem. Neverthess, they are still very useful for disease subtype discovery with microarrays.[20, 21, 73, 83, 816] They are also amongst the earliest techniques used for time series cell cycle studies with microarrays.[225, 277]

In a recent study,[187] six clustering methods—hierarchical clustering using the umweighted pair group method with arithmetic mean,[225] k-means,[324] DIANA,[421] a fuzzy logic-based methods called Fanny,[421] A model-based clustering methods,[59] hierarchical clustering with partial least squares[186]—are com-

pared in the context of transcriptional analysis of budding yeast sporulation as well as two sets of simulated data. The study concludes that DIANA is the most consistent performer with respect to the known profiles of the three test models.

5. Remarks

In this chapter, we have surveyed a large number of techniques for the data mining tasks of feature selection, classification, association rules extraction, and clustering. We have also briefly mentioned some examples of their applications in the analysis of biological and clinical data. There are other varieties of data mining tasks that we have not discussed. We close this chapter by briefly mentioning some of them below.

Outlier analysis deals with objects that do not comply with the general behavior of a model of the data. Most datamining applications discard outliers. However, in some applications the rare events can be more interesting than the more regularly occurring ones. For example, the detection of new particles in nuclear accelerator experiments.

Trend analysis describes and models regularities or trends for objects whose behavior changes over time. The distinctive features of such an analysis include time-series data analysis, periodicity pattern matching, and clustering of time-related data. The inference of gene relationships from large-scale temporal gene expression patterns[205] is an example of this topic.

Feature generation is another important data mining task, especially for the analysis of biological data such as DNA and protein sequences, and clinical data such as images. The techniques described on the main text of this chapter all assume that our data possess explicit features. However, for data such as DNA sequences, protein sequences, and images, the explicit features present in the data are usually inadequate for analysis. For an image file, the features that are explicitly present are the colour and intensity values at each pixel. For a DNA or protein sequence, the features that are explicitly present are the nucleotide or amino acid symbols at each position of the sequence. Such features are at too low a level for good understanding. Feature generation is the task of generating higher-level features that are more suitable for analysis and for understanding the data. However, some knowledge of the problem domain is typically needed to identify and generate high-level features in a robust manner. A recent bioinformatics example where feature generation plays a key role is the recognition of translation initiation sites.[515, 928]

Finally, inference of missing data is a data mining task that is occasionally needed in the analysis of biological and clinical data. There are many reasons

for such a need. For example, gene expression microarray experiments often generate datasets with many missing expression values due to limitations in current microarray technologies.[837] As another example, patient medical records usually have many missing values due to missed visits and other reasons. How the missing values can be estimated depends on the situation. Sometimes, a missing value in a sample can be estimated based on a corresponding value in other samples that are most similar to this sample. Sometimes, a missing value in a sample can be inferred from secondary data associated with the sample.