Chapter 9

Multi variate analysis

Consider a data sample Ω described by the set of variables $\underline{x}$ that is composed of two (or more) populations. Often we are faced with the task of trying to identify or separate one sub-sample from the other (as these are different classes or types of events). In practice it is often not possible to completely separate samples of one class A from another class B as was seen in the case of likelihood fits to data. There are a number of techniques that can be used in order to try and optimally identify or separate a sub-sample of data from the whole, and some of these are described below in order of increasing complexity. Each of the techniques described has its own benefits and dis-advantages, and the final choice of the "optimal" solution of how to separate A and B can require subjective input from the analyst. In general this type of situation requires the use of multi variate analysis (MVA).

The simplest approach is that of cutting on the data to improve the purity of a class of events, as described in section 9.1. More advanced classifiers such as Bayesian classifiers, Fisher discriminants, neural networks, and decision trees are subsequently discussed. The Fisher discriminant described in section 9.3 has the advantage that the coefficients required to optimally separate two populations of events are determined analytically up to an arbitrary scale factor. The Neural Network (section 9.4) and Decision Tree (section 9.5) algorithms described here require a numerical optimisation to be performed. In this context the optimisation process is called training, and having trained an algorithm with a set of data one has to validate the solution. Having discussed several classifier algorithms, the concepts of Bagging and Boosting are described as variants on the training process. In the following it is assumed that the data Ω contain only two populations of events. These populations are either referred to as A and B, or as signal and background depending on the context. It is possible to generalise these approaches to an arbitrary number of populations.

9.1 Cutting on variables

An intuitive way to try and separate two classes of events is to ***cut*** on the data to select interesting events in a cleaner environment than exists in the whole data set. For example if you consider the case where the data set Ω contains two types of events A and B that are partially overlapping. One possible solution to the problem of separating A from B is to select the events that satisfy $A \setminus B$. If $C = A \setminus B \neq \emptyset$, then this will be a pure sample of interesting events[1]. The pertinent questions are (i) what sacrifice has been made in order to obtain C, and (ii) would it have been possible to reject less data and obtain a more optimal separation of A and B so that we can further study a subset of the data?

What do we mean by making a cut on the data? Consider the data sample above Ω which contains two classes of events: A and B, each of which is described by discriminating variables in n dimensions. If we cut on the value of one or more of the dimensions, then we decide to retain an event e_i that passes some

[1] Purity is defined as the fraction of signal (or interesting events) in a sample of data.

threshold

$$P(e_i \in A) > 0 \tag{9.1.1}$$

and would decide to discard an event if

$$P(e_i \notin A) \text{ is significant.} \tag{9.1.2}$$

There is an element of subjectivity in the second condition. We can think of a cut in some variable x as a binary step function $f(x)$ where in the case of a positive step we may write

$$f(x) = 1 \text{ for } x > X_0, \tag{9.1.3}$$
$$f(x) = 0 \text{ elsewhere.} \tag{9.1.4}$$

and for a negative step, we change the inequality from $>$ to $<$. In order to optimise the cut on x we need to determine what it is we aim to achieve. If we assume our signal events are those of class A, then it follows that we would like to retain as many events of type A as possible, while discarding as many events of type B as possible. If $A \cap B = \emptyset$, then it is possible to determine X_0 by inspection of the distributions. If however $A \cap B \neq \emptyset$, we need to choose what we mean by the term *optimally separating A and B*.

The following lists some of the possible ways to determine X_0 for *optimal* separation for a given test statistic.

1. If it is of paramount importance to obtain a pure sample of A with no contamination or dilution from B, then we define X_0 in such a way that satisfies $C = A \setminus B \neq \emptyset$ with as many events passing into C as possible. Practically this usually is achieved at a significant cost in statistics and so will probably not be a sensible criteria for optimisation for many of the situations encountered.

2. We can introduce the notion of the ***significance*** S of the signal content (amount of A) in Ω relative to the background (amount of B). Then we can choose the value of X_0 that results in the greatest significance of signal. A common definition of significance is the test statistic

$$S = \frac{N_S}{\sqrt{N_S + N_B}}, \tag{9.1.5}$$

   where N_S is the number of signal events, and N_B is the number of background events that pass a given cut with cut-value $x = X_0$. The motivation for this definition of significance is that we want to compare any hint of a signal found to the statistical uncertainty on the number of events in the data. The underlying logic is that we want to be able to minimise any incorrect claim of a signal that would arise from statistical fluctuations in the background sample. As a result if we compute a numerical value for S, we normally say that the expected significance for a given cut is $S\sigma$, assuming that the denominator corresponds to a Gaussian uncertainty on the total number of observed events. Other test statistics used include purity and the signal to background ratio N_S/N_B.

3. If we are searching for an effect that is expected to be absent from the data, then we may want to optimise in such a way that we minimise the uncertainty on the background estimation (or number of events of type B that will remain in the sample), as this will dominate the uncertainty we obtain on the possible presence of a signal, and hence on any limit we are able to place that rules out the effect we seek.

The previous discussion with regard to making cuts has been based on a single dimension. In the case that all relevant dimensions $\underline{x}$ in Ω are uncorrelated, it is sufficient (and efficient) to optimise the cut values $\underline{X}_0$ one dimension at a time. The values of $\underline{X}_0$ obtained through such a procedure would be optimal. The more general situation encountered is when two or more dimension are correlated. For such cases one would

ideally like to simultaneously optimise the values of $\underline{X}_0$, however this is often not practical in terms of time or resources[2]. A possible alternative to this is to iteratively optimise the values of $\underline{X}_0$ one dimension at a time. If on subsequent iterations of the optimisation the value of X_0 obtained for a given dimension does not change appreciably, then you will have obtained the cut value for this dimension. In practice it may take several iterations to achieve this when two or more dimensions are correlated.

Example: Given a sample of data with an expected number of 100 signal events over a background of 1000 events, what is the optimal cut value to maximise the significance $S = N_S/(\sqrt{N_S + N_B})$? In order to determine this, we use 10^6 simulated data events for signal and background with known mean and widths that correspond to that expected in the data. These distributions are shown in Figure 9.1, where the signal and background are distributed according to Gaussian PDFs with means of 0.1 and 0.5 and widths of 0.3 and 0.4, respectively. It can be seen that in this case there is a trade off from allowing background to pass the cuts, while retaining a reasonable signal efficiency. The figure also shows the cumulative probability distributions for signal and background, which is equivalent to the efficiency of selecting events for a cut X_0 that rejects higher values of x. The resulting significance distribution for this situation is shown in Figure 9.2 where one can see that $X_0 = 0.34$ would provide optimal separation between signal and background using this method. The significance has a maximum value of 3.8σ for this value of X_0. On further inspection of the figure one can see that for larger values of X_0, there is a drop in significance arising from an increase in background. For smaller values of X_0 there is a drop in significance as signal is removed that would otherwise contribute to a measurement.

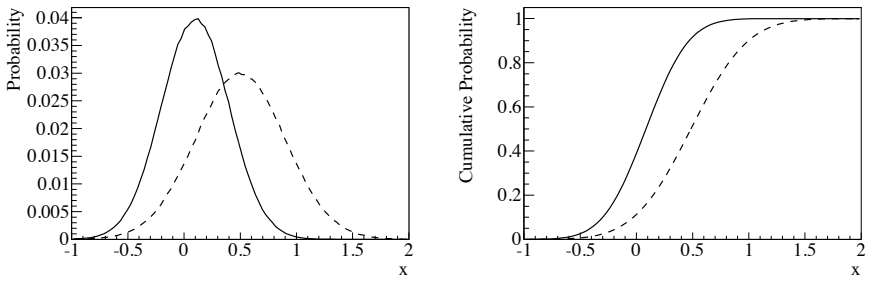

Figure 9.1: (left) The distribution in x of simulated (dashed) signal and (solid) background events, and (right) the cumulative probability distributions summing up from left-to-right for the cut based optimisation example described in the text.

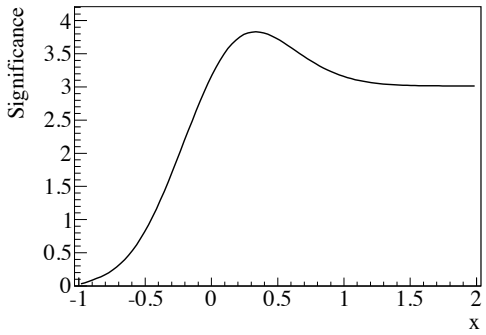

Figure 9.2: The significance computed as $N_S/(\sqrt{N_S + N_B})$ for the cut based optimisation example described in the text.

[2]The number of iterations required to simultaneously optimise m dimensions scales as the number of iterations for one dimension raised to the power of m. This is referred to as the curse of dimensionality as originally noted by Bellman (1961).

9.1.1 Optimisation of cuts as a precursor to further analysis

It should be noted that if the aim of a cut based selection of events is to subsequently use those events with a more complicated algorithm such as a fit based optimisation as discussed in section 8, or with an MVA like those described in the remainder of this chapter, then it doesn't make sense to optimise cut values as described above. The end result of your experiment will be the result of analysis with that more sophisticated technique - and so it is that which should be used to determine what is the *optimal* measurement to make. If one does optimise with a cut based approach, and then performs a more sophisticated data analysis, the end result will generally be less precise than if one applied a loose set of cuts, then performed the subsequent data analysis. It is important to stress that control samples of data or simulated data should be used in the optimisation process so that the result you obtain is not biased.

9.2 Bayesian classifier

The scenario of hypothesis comparison described in section 7.4 can be extended generally to a classification problem, given some data Ω that can be tested against a set of classifications given by H, where the i^{th} classification is given by H_i. For each event ω_j in the data set we can compute the probability $P(\omega_j|H_i)$ that the event is of the i^{th} classification. The most probable hypothesis is given by

$$P_{max}(\omega_j|H_i) = max\left[P(\omega_j|H_i)\right], \tag{9.2.1}$$

i.e. the largest value of $P(\omega_j|H_i)$ for all i is used to identify the classification for an event. Such a classifier is referred to as a Bayesian classifier.

Example: Consider the situation where one is interested in identifying three categories of event: (i) Interesting I and in need of detailed study, (ii) possibly interesting PI at some level, and (iii) not interesting NI. One can compute

$$\begin{aligned} P_I &= P(\omega_i|I), & (9.2.2)\\ P_{PI} &= P(\omega_i|PI), & (9.2.3)\\ P_{NI} &= P(\omega_i|NI). & (9.2.4) \end{aligned}$$

If the largest probability for event ω_i is P_I, one will classify the event as interesting and in need of further study. Similarly if the largest probability is P_{PI} or P_{NI}, the event would be classified as possibly interesting or not interesting, respectively. A more specific example utilising Bayesian classifiers is discussed in section **??**.

9.3 Fisher discriminant

Fisher's linear discriminant (or ***Fisher discriminant***) is a linear combination of the variables $\underline{x}$ to form a single classifier output $\mathcal{O}$ given by (Fisher, 1936)

$$\begin{aligned} \mathcal{O} &= \sum_{i=1}^{n} \alpha_i x_i + \beta, & (9.3.1)\\ &= \underline{\alpha} \cdot \underline{x} + \beta. & (9.3.2) \end{aligned}$$

The sum is over the number of dimensions n in the classification problem. In order to make use of Eq. (9.3.2) in practice we need to determine the weight coefficients α_i, or equivalently the weight vector $\underline{\alpha}$. The value of β does not affect the separation between data types, it adjusts the overall central value of the resulting Fisher distribution, and in the following discussion this parameter will be set to zero.

Given the data set Ω and the knowledge of which elements in Ω are of class A and which are of class B we can compute the mean and variance of $\underline{x}$ for the two classes. These are $\underline{\mu}_{A,B}$ and $\sigma^2_{A,B}$ where the subscript indicates the event type. Using Eq. (9.3.2) we can also compute the mean M and variance Σ^2 of the Fisher distributions for the two classes of data

$$M_{A,B} = \alpha^T \mu_{A,B} = \sum_i \alpha_i \mu_{A,B}, \tag{9.3.3}$$

$$\Sigma^2_{A,B} = \alpha^T \sigma^2_{A,B} \alpha = \sum_i \sum_j \alpha_i \sigma_{ij\,A,B} \alpha_j, \tag{9.3.4}$$

where we now revert to matrix notation to avoid having to explicitly write out the summations involved. In order to maximise the separation between A and B we want to maximise the difference between M_A and M_B, while at the same time minimise the sum of the variances of the two output distributions. These requirements are expressed in the ratio

$$J(\alpha) = \frac{[M_A - M_B]^2}{\Sigma^2_A + \Sigma^2_B}, \tag{9.3.5}$$

where the squared sum of the mean values of the Fisher distribution for the two classes is

$$[M_A - M_B]^2 = \left[\sum_{i=1}^{n} \alpha_i (\mu_A - \mu_B)_i\right] \left[\sum_{j=1}^{n} \alpha_j (\mu_A - \mu_B)_j\right], \tag{9.3.6}$$

$$= \sum_{i,j=1}^{n} \alpha_i (\mu_A - \mu_B)_i (\mu_A - \mu_B)_j \alpha_j, \tag{9.3.7}$$

$$= \alpha^T B \alpha, \tag{9.3.8}$$

where the matrix B is introduced to represent the separation *between* the classes of events based on mean values. The the sum of the Fisher distribution variances is

$$\Sigma^2_A + \Sigma^2_B = \alpha^T \sigma^2_A \alpha + \alpha^T \sigma^2_B \alpha, \tag{9.3.9}$$

$$= \alpha^T W \alpha, \tag{9.3.10}$$

where the matrix W is the sum of covariance matrices *within* the classes. Thus, we find

$$J(\alpha) = \frac{\alpha^T B \alpha}{\alpha^T W \alpha}. \tag{9.3.11}$$

The optimal separation between classes A and B can be found by minimising J with respect to the weight coefficients α, therefore by satisfying the condition

$$\frac{\partial J(\alpha)}{\partial \alpha} = 0. \tag{9.3.12}$$

One can show (for example see Cowan (1998)) that the maximum separation is found when

$$\alpha \propto W^{-1}(\underline{\mu}_A - \underline{\mu}_B), \tag{9.3.13}$$

so we are able to compute the weights α if we are able to determine the mean values $\mu_{A,B}$, and invert the matrix W. As the coefficients are determined up to some proportionality, we don't have a unique solution

for the set of weights, but have a family of related solutions. This method implicitly assumes that the matrix W can be inverted. If W is singular, then one either has to change the input dimensions to produce a non-singular W matrix, or alternatively use a different classification method.

If we so wish, we can extend the form of Eq. (9.3.2) by scaling or offsetting the input data to lie within a specified range. Furthermore it is possible to scale or offset the computed $\mathcal{O}$ as desired if you want to relocate the mean value or change the range over which the classifier outputs are computed for the data. On doing this the separation between types A and B will remain optimal as defined by the Fisher algorithm.

Example: Consider the situation where we have a data sample comprising two types of events: signal (S) and background (B), each described in two dimensions that are independent: x and y. We want to compute a set of Fisher discriminant coefficients α to separate out S from B so that we can further analyse a clean sample of the signal events. From the data sample, we are able to compute

$$\mu_S = \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix}, \text{ and } \sigma_S = \begin{pmatrix} 0.3 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}, \tag{9.3.14}$$

for the signal, and

$$\mu_B = \begin{pmatrix} 1.0 \\ 1.2 \end{pmatrix}, \text{ and } \sigma_B = \begin{pmatrix} 0.4 & 0.0 \\ 0.0 & 0.5 \end{pmatrix}, \tag{9.3.15}$$

for the background. Here σ_A and σ_B contain the standard deviations of the independent variables x and y, and are not covariance matrices. The distributions of the signal and background data are shown in Figure 9.3. There are regions of the signal that are background free, and similarly there are regions of the background data that are signal free. The objective is to obtain an optimal separation between the two classes of events.

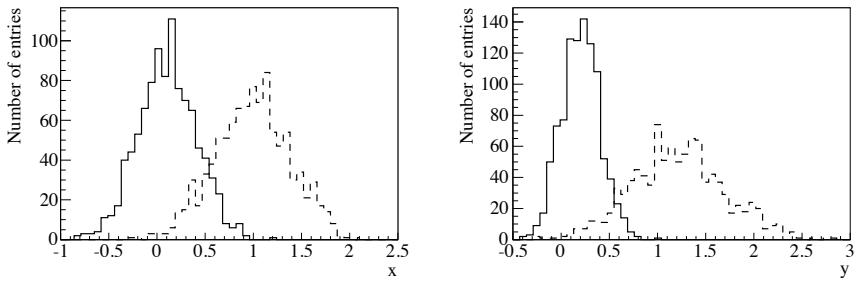

Figure 9.3: Distributions of (left) x and (right) y for (solid) signal and (dashed) background events for the example described in the text.

Given this information we can compute the difference in mean values of S and B as

$$(\mu_S - \mu_B) = \begin{pmatrix} -0.9 \\ -1.0 \end{pmatrix}, \tag{9.3.16}$$

and W is given by[3]

$$W = \begin{pmatrix} 0.25 & 0 \\ 0 & 0.29 \end{pmatrix}, \tag{9.3.17}$$

[3]The original σ matrices provided in this example contain the standard deviations of the data, and the discriminating variables x and y are un-correlated. Hence the individual standard deviations need to be squared to obtain the covariance matrix, and using this one can then construct W. This step is not required if one starts from the two covariance matrices.

where the off-diagonal terms are zero as x and y are uncorrelated for both signal and background. From Eq. (9.3.13) we can determine the weight vector up to some arbitrary scale factor to be

$$\alpha = \begin{pmatrix} -3.6 \\ -3.45 \end{pmatrix}. \tag{9.3.18}$$

Figure 9.4 shows the output fisher distribution $\mathcal{O}$ obtained using the weights computed for this example. The separation between signal and background distributions in terms of $\mathcal{O}$ is better than the separation either with x or with y when one compares with the distributions in Figure 9.3. The signal distribution appears on the right hand side of the figure as a result of the convention adopted in Eq. (9.3.13) where background means are subtracted from signal ones.

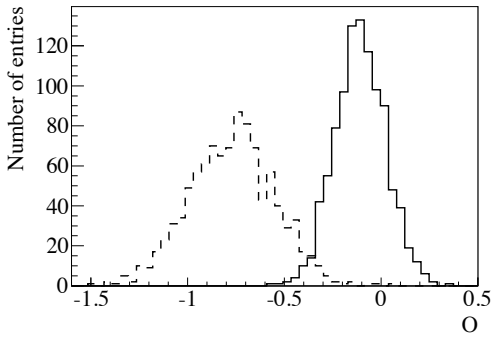

Figure 9.4: The Fisher discriminant output distribution $\mathcal{O}$ for (solid) signal and (dashed) background events for the example described in the text.

9.3.1 Choice of input variables

Often we have a choice of input variables or dimensions that we want to use to separate between classes of events. Some common sense should be used when doing this, as for example if you introduce a dimension where A and B are almost completely overlapping with similar distributions, that dimension will have essentially no weight in the final Fisher discriminant that you compute. In turn you may decide that it is not worth including that variable in your classifier.

A corollary of the method is that if the mean value of the distribution of events in a given dimension is the same for both classes A and B, but the shapes of the two distributions are rather different, by definition the corresponding weight α_i will be zero, [this follows from Eq. (9.3.13)]. In such cases it makes sense to transform the variable somehow in order to make sure that the mean values of the distributions for A and B are different. One possible way to do this if you have with a common mean value for types A and B, where events are distributed differently for the two types, is to fold the data about the mean value and use the resulting distributions as an input to the Fisher discriminant. These distributions will not be symmetric about a common mean for A, and the variable will in turn have a contribution to the separation between the two classes of event.

9.4 Artificial neural networks

There are many variants on the concept of artificial neural networks. These are all built upon complex structures assembled from individual perceptrons, see section 9.4.1. The type of neural network that is most commonly used in physics applications is that of a multi-layer perceptron (MLP) (see section 9.4.2). The

MLP is an ensemble of layers of perceptrons used in order to try and optimally separate classes of events. Typically there are n dimensions input to the network and only a single output, however it is also possible to configure a network with multiple outputs. Only single output MLPs are discussed here. An important, and often overlooked aspect to the use of neural networks is that of validation. After describing the MLP, there is a discussion on training methods in section 9.4.3, and the issue of validation is discussed in section 9.4.4. More information on this topic can be found in a number of books including (Hastie Tibshirani and Friedman, 2009; MacKay, 2011; Rojas, 1996).

9.4.1 Perceptrons

The fundamental building block of a neural network is the perceptron. The ***perceptron*** is the algorithmic analogy of a neuron. There are n inputs to the perceptron, these provide an impulse for the perceptron to react to. The perceptron has a pre-defined action which in turn performs some function and finally gives a response in the form of an output (see Figure 9.5). The simplest type of perceptron is a binary threshold perceptron. This takes an n dimensional input in the form of an event e_i described by the vector $\underline{x_i}$. Given $\underline{x_i}$, the perceptron is used to compute some response $\mathcal{O}$ using a so-called ***activation function*** y. If y_i is above threshold for a given event the response is one, and if it is below threshold the response is zero. The binary threshold perceptron algorithm is

$$\begin{aligned} y_i &= \underline{w} \cdot \underline{x}_i + b, \\ \mathcal{O} &= 1 \text{ if } y_i > 0, \\ &= 0 \text{ otherwise.} \end{aligned} \tag{9.4.1}$$

The vector $\underline{w}$ corresponds to the set of weights used to separate classes of events from each other, and b is a constant offset used to tune the binary perceptron's threshold value. If we think about what the perceptron is actually doing, one can see that we are defining a plane in an n-dimensional space as $\underline{w} \cdot \underline{x}_i + b$, and then accepting all events that occupy space on one side of this plane. The events on the other side of the plane are rejected. In order to optimally select interesting events using a single perceptron we need to determine the parameters $\underline{w}$ and b. So for each perceptron there are $n + 1$ weights (or n weights if you set b to zero) to determine. Training is discussed in more detail in sections 9.4.3 and 9.4.4.

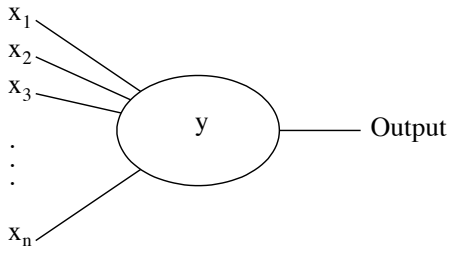

Figure 9.5: A single perceptron with n input values, an activation function y, and a single output.

The n-dimensional binary threshold perceptron is equivalent to the n-dimensional cut based event selection described in section 9.1 applied to a Fisher discriminant. Both algorithms are making a cut in the problem space, and one recognises the similarity between the activation function $\underline{w} \cdot \underline{x_i} + b$, and Eq. (9.3.2).

The function given in Eq. (9.4.1) is called the activation function of the binary threshold perceptron. In practice we are not restricted to a single type of activation function, and we are able to try other options. The following types of activation function are commonly used in perceptrons

- n-dimensional binary threshold function given by Eq. (9.4.1).
- A sigmoid (or logistic) function given by

$$y = \frac{1}{1 + e^{\underline{w}\cdot\underline{x}_i+\beta}}, \tag{9.4.2}$$

  which varies smoothly with output values in the range of 0 to +1.
- The hyperbolic tangent: $y = \tanh(\underline{w} \cdot \underline{x}_i)$ which varies smoothly with output values in the range -1 to $+1$.
- The radial function: $y = e^{-\underline{w}\cdot\underline{x}_i}$ which varies smoothly between 0 and 1.

Figure 9.6 shows example distributions of the aforementioned activation functions. By using a smoothly varying activation function, as opposed to the binary threshold function described previously, we are able to finely tune the decision as to whether an event is signal like or not in terms of a continuous variable. Another way of thinking about this is that it is possible to consider an event to be a little like signal or background, without having to make a hard judgment as to whether the event is definitely signal or background. This can be useful when sample distributions overlap in data as is often the case. One can think of the use of a continuous activation function as a blurred cut in parameter space for events that lie on the boundary between classification as type A or type B, compared to the hard cut that would be imposed by the binary threshold function. This is equivalent to the probabilistic treatment of events in a likelihood fit.

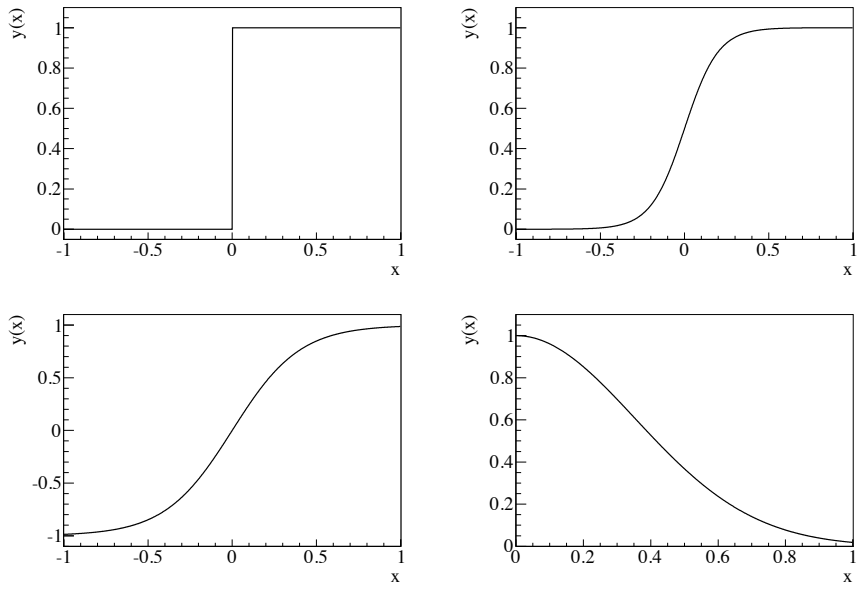

Figure 9.6: Example distributions of the (top left) binary, (top right) sigmoid, (bottom left) hyperbolic tangent, and (bottom right) radial activation functions.

9.4.2 Multi-layer perceptron

A ***neural network*** is a combination of perceptrons, each with n inputs. It is possible to have a single perceptron to govern the output of the network, which would combine the decisions made by each of the input nodes into a single output. Usually the output would be a continuous number between either zero and

one or -1 and $+1$ to indicate if an event e_i was signal like ($\mathcal{O} = +1$) or not ($\mathcal{O} = -1$ or 0 depending on the activation function).

In general a ***multi-layer perceptron*** is more complicated than this picture, and there will be a single input layer connected to the output node via one or more hidden layers. Figure 9.7 shows an MLP with n input nodes, one hidden layer of n nodes, connecting to a single output node. Each of the input nodes has n inputs, and the output of each of these nodes is transmitted to all of the nodes in the next layer. As each perceptron has at least n weight parameters to determine, if there are several hidden layers and n is large, the number of parameters to determine rapidly increases. For m perceptrons, in an input layer, each with an n dimensional input, feeding o perceptrons in a hidden layer, and a single output perceptron, then number of parameters to determine in order to compute the output of the MLP is: $n \times m$ for the input layer, $m \times o$ for the hidden layer, and o for the output node. So there would be a total of $(n + o) \times m + o$ parameters to determine. This assumes that the activation function for each node depends only on factors of $\underline{w} \cdot \underline{x}_i$. So if one has ten inputs ($n = 10$), to ten nodes in the input layer ($m = 10$), with a hidden layer of ten nodes ($o = 10$), and a single output layer, the number of weights to compute is 210. Such a network would be described as having a $10 : 10 : 1$ configuration in shorthand. Even for a $5 : 5 : 1$ MLP with ten inputs, one would have 80 parameters to determine. Such flexibility in the configuration of a network means that a lot of care needs to be taken to ensure that the trained set of *optimal* weights is not fine tuned on fluctuations in training samples. A method of determining the weight parameters is discussed in more detail in section 9.4.3, and section 9.4.4 discusses the importance and main issues of validating the computed weights.

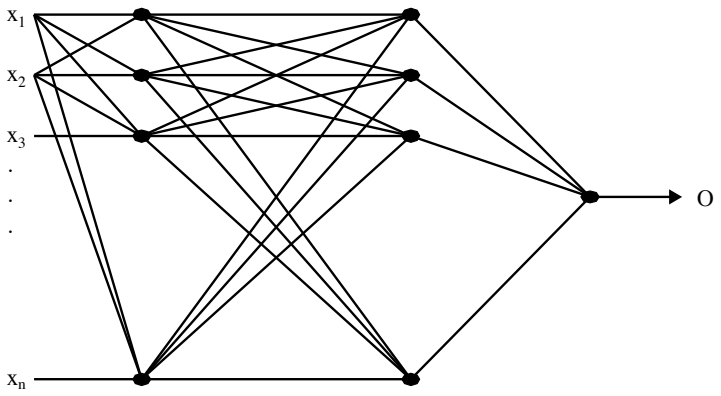

Figure 9.7: A multi-layer perceptron with n input values, one hidden layer of n nodes, and a single output.

9.4.3 Training an MLP

The process of determining the weight parameters for neural network is called ***training***. There are several steps involved in training a MLP which are as follows

1. Define an algorithm to assign an error to a given set of weights.
2. Define the procedure for terminating training, based on the computed error, or other information.
3. Guess an initial set of weights to test the classification process.

4. Evaluate the error defined in step (1) for a given set of data containing (preferably) equal numbers of target types for signal and background.
5. Determine a new set of weights based on mis-classified events.
6. Iterate the last two steps until the convergence criteria defined in step (2) has been reached.
7. Validate the weights obtained via this procedure (see section 9.4.4).

The case of a single perceptron

The ***error*** assignment for a single perceptron is based on the ability to correctly classify if an event e_i is of the appropriate type. For example signal events should be classified as signal, and background events should be classified as background.

If the signal classification (class A) is type= 1, and the background classification (class B) is assigned type= 0 for an event e_i, then we can define an error on the output of a perceptron ϵ_i as

$$\epsilon_i = \frac{1}{2}(t_i - y_i)^2, \tag{9.4.3}$$

where t_i is the true target type for the event, and y_i is the output of the perceptron. The value of y_i computed for an event will depend on the set of weight vectors used in the computation, and on the form of the activation function chosen for the perceptron. The misclassification of the event is given by $t_i - y_i$, however we want to be able to sum up error terms, and so it is conventional to square this difference to maintain a positive definite quantity. Similarly the factor of $1/2$ is also conventional.

If there are N events in the data sample Ω, then the total error from a single perceptron will be given by

$$E = \sum_{i=1}^{N} \epsilon_i \tag{9.4.4}$$

$$= \frac{1}{2}\sum_{i=1}^{N}(t_i - y_i)^2. \tag{9.4.5}$$

Having computed the error on the event classification it is desirable to be able to compute a new set of weight vectors that are closer to the optimal set than the initial guess. If the initial weight vector is $\underline{w}_m$, then we want to compute a new weight vector

$$\underline{w}_{m+1} = \underline{w}_m + \Delta\underline{w}, \tag{9.4.6}$$

such that $\underline{w}_{m+1}$ is closer to the minimum of the total error function than $\underline{w}_m$. If we consider the E versus $\underline{w}$ parameter space, then we can estimate the derivative of E with respect to $\underline{w}$ as

$$\frac{\Delta E}{\Delta\underline{w}} \simeq \frac{\partial E}{\partial\underline{w}}, \tag{9.4.7}$$

so

$$\Delta E \simeq \Delta\underline{w}\frac{\partial E}{\partial\underline{w}}. \tag{9.4.8}$$

If we choose $\Delta\underline{w}$ such that it depends on the rate of change of E with respect to the weight vector we can ensure that we take a small step toward the minimum if

$$\Delta\underline{w} = -\alpha\frac{\partial E}{\partial\underline{w}}, \tag{9.4.9}$$

where α is a small positive parameter called the ***learning rate***. Thus the total change in error ΔE given by

$$\Delta E \simeq -\alpha\left(\frac{\partial E}{\partial\underline{w}}\right)^2, \tag{9.4.10}$$

which is always a negative quantity by construction. The functional form of E is given by Eq. (9.4.5), so once the activation function is defined, hence the $\underline{w}$ dependence of E has been chosen, it is possible to compute $\Delta\underline{w}$ and hence ΔE for a data sample. This method of iteratively computing weight vectors is often called the gradient descent method or the Δ Rule, and was previously encountered when discussing optimisation of fit model parameters in section 8.1.1.

Back propagation: training a MLP

When one moves from a single perceptron to a MLP, the error assignment algorithm is more complicated. One has to assign some importance to different contributions to the final network output, and where necessary to work back from the output layer to the input layer to modify the choice of weights. Back propagation is a generalisation of the gradient descent algorithm discussed above. The weight determination for the input layer of perceptrons is based on Eq. (9.4.6), where once again the objective is to minimise the total error rate E. Each perceptron in the network contributes an error rate corresponding to Eq. (9.4.5).

As with the case of training a single perceptron, having determined the error for an ensemble of events, given an initial assumed set of weights, one can iterate and estimate a new set of weights. This process follows an analogous procedure to that outlined above. This method is a generalisation of the Δ Rule, so again it works on the concept of error minimisation through gradient descent. Detailed descriptions of the back propagation method can be found in a number of texts, for example see MacKay (2011) and Rojas (1996).

9.4.4 Training validation for a neural network

Training validation is discussed as a sub-section in its own right to highlight the importance of this topic. It is not sufficient to assume that a computed set of weights for a network is correct. Having obtained what is assumed to be a reasonable set of weights, it is necessary to perform cross checks to ensure that the solution is not tailored to statistical fluctuations in the data used to compute them. There are many local minima that could be found through the minimisation of E with respect to the weight parameters − so how can one determine if the minimum obtained is really the global minimum, or if it is one of the local minima?

The problem arises as the MLP with a given set of weights w has a total classification error E as computed for some training data sample Ω_{train}. This training sample is a reference where target types of each event, either as class A or as class B are known beyond doubt. In practice we will want to apply the MLP to a classification problem using a different data sample comprising real data Ω_{data} where the target type is not certain. How do we know that the MLP will behave reasonably when applied to Ω_{data}? If we have sufficient training data then we can construct a statistically distinct set Ω_{validate} that is equivalent to Ω_{train} in all respects, but satisfies $\Omega_{\text{train}} \cap \Omega_{\text{validate}} = \emptyset$. If the MLP gives the same total error for both Ω_{train} and Ω_{validate}, then it is reasonable to expect the MLP to behave as expected when we apply it to Ω_{data}. Hence to ensure that we have not fine tuned the weights of the MLP, we need to check the total error E obtained from the network using Ω_{train}, and then compute the total error E' obtained when the network is applied to Ω_{validate}. When $\partial E/\partial\underline{w}$ has reached a minimum, and both E and $E - E'$ are sufficiently small, we can

assume that the weights computed for the MLP are not fine tuned and that the training has converged. Hence we can use the network with confidence on a real data set. In order to determine if E is sufficiently small we have to set an ***error threshold*** δ by hand.

Typically we use either pure reference data samples that resemble the classes we are trying to separate, or Monte Carlo simulated data for Ω_{train} and Ω_{validate}. While the number of events of class A or B used in training can be different, it is generally better to use equal numbers of both types of events in training.

If we reflect upon the large number of weight parameters that have to be determined when we train a neural network, the next logical question is *"How much data do we need to use when training a given network"*. There has been some discussion on this in the literature, for example it has been noted that for a MLP with a single hidden layer, with W weight parameters that need determining and an error threshold of δ, then you should use more than W/δ events in the training sample (Baum and Haussler, 1989). For more complicated networks this number is multiplied by a factor of $\ln(N/\delta)$, where N is the number of nodes in the network.

Now we return to the issue of local versus global minima. One can try re-training a neural network, starting with different weight sets to check if the same set of weights is converged upon via the training-validation process. If the same weights are obtained from several different trials, then one has some confidence that the solution obtained may be a global minimum. Another possible test would be to try a different minimisation algorithm, and see if the same solution is obtained.

9.5 Decision trees

The concept of a ***decision tree*** (DT) is derived from that of an optimal cut based selection of events. As with the previously described methods, the aim of the DT is to separate classes of events with as small a misclassification error as possible. If one has n dimensions describing classes A and B, then the root node of a DT uses the optimal set of dimensions required to separate A and B with some cut on $\underline{x}_i$. In general, the resulting sub-samples of events will contain both classes, so it is possible to consider further subdivision using an optimal combination of dimensions. This iterative process can be continued until such time as one is able to classify A and B with a satisfactory error rate. Figure 9.8 shows a schematic of a DT. Each of the nodes in the tree will split the data set into an A-like and a B-like part. As a result, the lowest level of the tree will contain a number of A-like and B-like parts. In other words, this layer contains sub-sets of Ω that are either mostly A or mostly B, each with a small misclassification error. As the optimal dimensions are used at each step to separate A and B it is quite possible that some dimensions will be used more than once while others are never used to classify events. The decision process at each node is equivalent to that of a cut based algorithm. The additional flexibility of a DT of many nodes compared to a cut based optimisation means that the algorithm has more flexibility (hence power) to separate A from B. The output of a single DT is a binary separation between signal (1) and background (0).

The algorithm used used to separate data into A and B-like parts of the data is discussed in section 9.1, where the number of dimensions used for a given node is that required to provide optimal separation (i.e. not all dimensions have to be used to make the decisions at all of the branching points in a tree). As a result there are between 1 and n weight parameters to determine per node in the tree. A corollary of this is that a decision tree with m nodes will have between m and $n \times m$ weight parameters to determine. While the weight parameter scaling issues of DTs are not as severe as those for a neural network, it follows that the issues discussed above with regard to training validation of weights for neural networks are also serious issues for DTs. Two techniques that can be used to improve the stability of the trained DTs, with respect to statistical fluctuations in the training sample, are boosting (section 9.5.1) and bagging (section 9.5.2). In general to avoid over-training a DT one can compute an ensemble of trees with minor variations between them (sometimes referred to as a forest), and use the aggregate classification of an event. In this way one can construct a distribution of outputs with value in the range $[0, 1]$.

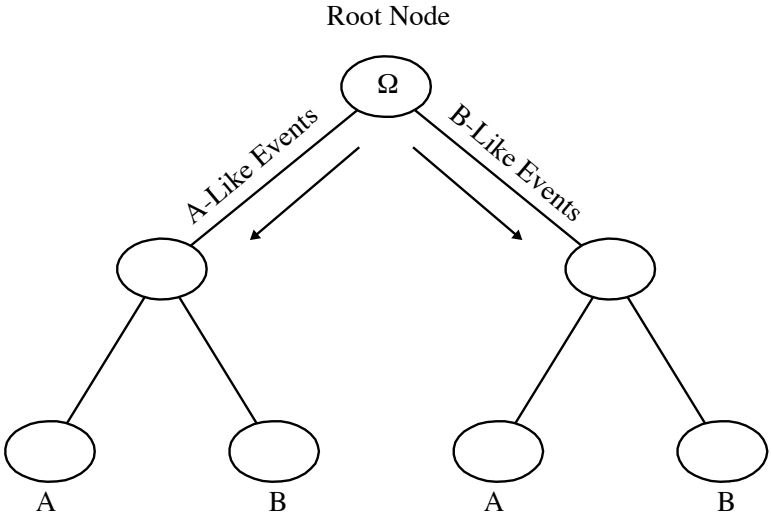

Figure 9.8: A DT with a root node above two layers of nodes that further sub-divide the data sample into pockets of A and B-like events.

9.5.1 Boosting

The aim of training a DT with a ***boosting*** algorithm (referred to as a boosted DT or BDT) is to successively re-weight events in favour of those that are mis-classified from one iteration of the training process to the next. The logic is that the subsequent training iterations will be focused on correctly classifying those events that were previously mis-classified. When boosting a DT one typically re-weights events with a factor α, defined in terms of the error rate ϵ. Having re-weighted events, the total weight of the data sample is renormalised so that the sum of event weights used is constant for all iterations. A number of possible re-weighting factors exist, one of these variants is

$$\alpha = \log\left(\frac{1-\epsilon}{\epsilon}\right), \tag{9.5.1}$$

which is referred to as the AdaBoost.M1 algorithm, and is discussed at length in Hastie Tibshirani and Friedman (2009).

9.5.2 Bagging

Bagging is an alternate (or additional) method for improving the stability of a DT to that of boosting. This method involves sampling sub-sets of data from Ω, and then performing many different training cycles for the DT one for each sub sample of data. The ultimate set of weights used will be the mean value obtained for the ensemble of DTs. If the data sample Ω is not sufficient to provide statistically distinct sub-sets of data one can oversample Ω, and use each event many times for different training cycles. This re-sampling method reduces the susceptibility of a DT to statistical fluctuations.

9.6 Choosing an MVA technique

There are a number of factors that should be considered before choosing a particular MVA to separate between classes of events. Some of these factors are logical and based on taking the best classifier to do the job, other factors are subjective and are based on the understanding of the analyst, or indeed the use case of the MVA. If a classifier will be used to provide an end decision on how probable it is that a given element is of class A or B, then your decision to use that classifier might differ from that made if you intended to use the classifier in a fit based minimisation problem, or indeed as an input to another MVA algorithm.

When assessing the logical input required to understand what is the best classifier we want to understand how well class A is separated from class B in our data. There are number of ways to do this, however it can often be instructive to compute curve of the efficiency of class A vs class B. In this case we would consider the best classifier to be the one that has the maximum efficiency of one class while minimising the efficiency of the other.

Consider the example from an experiment where we have signal and background classes for A and B. There are many input variables that distinguish between the classes. These variables are the n dimensions that will be used to classify the data. The single output variable from a classifier is then the quantitative information that we have to decide if one algorithm is better than another at separating signal from background. Figure 9.9 shows the distribution of signal versus background efficiency for these hypothetical test data. The better the event classification, the closer it will pass to the bottom right hand corner. An extreme example of this is the case of being able to identify a sample of pure signal, where the curve will pass through the point $(1, 0)$. Often the rejection rate versus error rate, is plotted when choosing which MVA to use for a problem; such a distribution is known as a receiver operating characteristic.

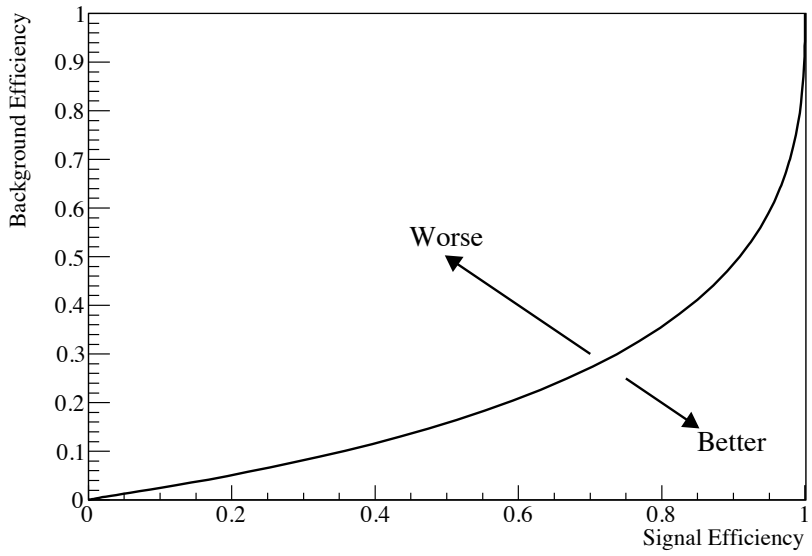

Figure 9.9: The distribution of signal versus background efficiency for a classifier output from an experiment. The better the event classification, the closer the curve will pass to the bottom right hand corner.

One can compute a measure of the separation between types A and B by adopting the definition of separation used for the Fisher discriminant in Eq. (9.3.5) and computing the ratio of the difference of the means (M_i) of the two output distributions and the variances or RMS values (Σ_i),

$$\mathcal{S} = \frac{(M_A - M_B)^2}{\Sigma_A^2 + \Sigma_B^2}. \tag{9.6.1}$$

The greater the value of $\mathcal{S}$ for a given classification algorithm, the more discriminating power that has. If the distributions of one or both of the event types as classified by an algorithm are rather different to that of a Gaussian, then this definition of separation may not be very appropriate.

The fact that an algorithm gives optimal separation for the specified set of discriminating variables used, does not necessarily mean that this is the most optimal solution to the problem. In order to ascertain this we would have to compare classifiers for all possible combinations of input discriminating variables and all possible classifier algorithms. Where it is impractical to perform tests with all of the combinations, one should endeavour to test as many as is reasonable before converging upon a candidate classifier to use in the analysis of data. Having identified such a candidate, the next step in the process is to consider any subjective factors as discussed in the following that may influence the decision of an algorithm or number of dimensions to use to classify the data.

When it comes to addressing the subjective input required to understand which classifier is the best, we should remember that there is no magic recipe to help us. However there are a number of factors that should always be adhered to:

1. Simplicity can be a key factor in determining which classifier is used. The clarity in understanding what is happening to your data in order for a given event to be classified as one type or another, or indeed to be able to easily explain what you are doing to a colleague should not be underestimated.
2. Only use a method that you understand. If you use algorithms that you do not fully understand, you may find that you have a better separation between classes, however you run a risk of having over-trained the algorithm without realising it, or falling foul of some pathological behaviour. The only way to limit such a risk to acceptable levels is to adhere to an abstinence policy of only using algorithms that you understand.
3. Where appropriate, always ensure that sufficient data are used to train and validate an algorithm. There is a necessary trade off between the desire to use as much data as possible as an input to an algorithm, and ensuring that you can validate that your resulting classifier does not suffer from over-training or some other pathology when checked against a statistically independent sample. If you fail to validate a classifier that requires training, then you should not be tempted to use that classifier for anything beyond an educational exercise.
4. Think carefully about the shape of the output classifier in the context of how you wish to use it. For example if you are using this as an input to a fit based optimisation:
    - Are you able to easily parameterise the target shapes of the classifier?
    - If the classifier is a highly irregular, or peaked shape, is there a $1:1$ mapping that you can apply in order to retain the separation power of the classifier, but obtain a distribution that can be parameterised or used as an input variable in a fit?

The quantitative and subjective inputs discussed above all play a role when we want to understand which classifier is the best for solving our problem. The discussion in this section is relevant for any MVA technique, not just the algorithms that have been described in detail here. Each problem that you are faced with will have its own unique set of quantitative and subjective factors that must be considered in order to choose which classifier is the best for a given problem. If in doubt, there is little lost in opting for the simplest algorithm. The cost in doing so is usually some loss of precision in a measurement, however sometimes this can be considered acceptable if the gain in clarity is a subjective factor that carries significant weight for your particular problem.