An Introduction to Multivariate Analysis Techniques

Pietro Biassoni
Università degli Studi and INFN Milano
pietro.biassoni@mi.infn.it

January 13, 2011

Contents

Introduction

Multivariate statistical analysis (MVA) have been developing from the last decades of the XX century and is currently an actual research field. The need of processing large quantities of data and discriminating between class of events which have very similar experimental signatures has brought MVA to be largely used among the physics community. In particular in high energy physics these techniques are used in order to discriminate signal events which are very similar to background. In the LHC and *super$B$-factories* era, MVA will become a routinely used techniques to achieve the maximum discrimination power, in order to investigate very rare phenomena or to reduce huge backgrounds. These notes are far away to be an exhaustive explanation of MVA, their goal is to provide the student an early introduction and some practical example of their possible application. A comprehensive introduction to MVA can be found in [1], if you are interested in getting a more technical and exhaustive explanation about MVA techniques please refer to [2]. In the first chapter of these notes we give an introduction to widely used MVA techniques, in the second chapter we show a possible application of the presented techniques.

Chapter 1

Multivariate Data Analysis Techniques

1.1 Introduction to Signal-Background Discrimination

In general we consider an "event" $\mathbf{x}$ belonging to the space of events Ω of dimension n as a vector of n x_i random variables. $\mathbf{x}$ is distributed according $f(\mathbf{x}, \theta)$, the probability density function (PDF) of $\mathbf{x}$, where some of the θ_i parameters of the function can eventually be unknown *a priori*.

If we have to distinguish between many hypotheses $H_0,\ H_1, ...$, each of such hypotheses will imply a given PDF $f_i(\mathbf{x}|H_i)$. In order to investigate the agreement of the observed data with a given hypothesis we have to construct a test statistic $\mathbf{t}(\mathbf{x})$. Then for each hypothesis a PDF $g_i(\mathbf{t}|H_i)$ is defined. The test statistic $\mathbf{t}$ is allowed to have as many dimension as $\mathbf{x}$ has, but the advantage of constructing a test statistic with dimension $m < n$ is to preserve discrimination between hypotheses and reduce the amount of data to be stored.

Once we have got the test statistic PDF $g_i(\mathbf{t}|H_i)$, we can define a cut that identify a *critical region* for t, where the null hypothesis H_0 is rejected (and its complementary, the *acceptance region*, where the null hypothesis is accepted). So, if the value of the test statistic evaluated for a certain event $\mathbf{x_A}$ belong to the *critical region*, we state that $\mathbf{x_A} \notin H_0$.

If we accept H_0 for $\mathbf{t} > \mathbf{t_{cut}}$[1], the quantities

$$\alpha = \int_{\mathbf{t_{cut}}}^{+\infty} g(\mathbf{t}|H_0)\mathrm{d}t \qquad \text{and} \qquad \beta = \int_{-\infty}^{\mathbf{t_{cut}}} g(\mathbf{t}|H_1)\mathrm{d}t$$

are defined. α is the probability of rejecting H_0, when H_0 is true. This is called *type I error*. β is the probability of accepting H_0, when H_0 is false. This is called *type II*

[1]Here I abuse the use of the notation. As in one dimension $t > t_{cut}$ is easily understood as "t outside signal region", I use $\mathbf{t} > \mathbf{t_{cut}}$ to mean "$\mathbf{t}$ outside the multidimensional signal region"

error. The critical region has to be chosen in order to maximize some figure of merit (FOM), in order to achieve the better discrimination between various hypotheses. We usually restrict to the case with only two hypotheses: the null one (”signal”) and another one (”background”).

1.1.1 Neyman-Person Lemma

Neyman-Person lemma states that given an efficiency ϵ for the null hypothesis, defined as

$$\epsilon = \frac{\text{events in acceptance region}}{\text{total events}}$$

the maximum signal purity is achieved in the region

$$\frac{g(\mathbf{t}|H_0)}{g(\mathbf{t}|H_1)} > c(\epsilon)$$

where c is some number determined according to ϵ, a proof of this can be found in [3]. Note that a test based on the (multidimensional) Neyman-Person acceptance region is equivalent to the one-dimensional statistic given by

$$r = \frac{g(\mathbf{t}|H_0)}{g(\mathbf{t}|H_1)} \tag{1.1}$$

which is called *likelihood ratio* for the hypotheses H_0 and H_1. Anyway the likelihood ratio test statistic is quite unpractical in the common use because it relies on the knowledge of the exact (multidimensional) functional forms for $g(\mathbf{t}|H_i)$ and also implies great computational effort.

Simpler test statistics can be used following simple assumption on the form of the test statistics. In addiction to the possibility of choosing a critical region where apply a cut, PDFs $g(\mathbf{t}|H_i)$ of the test statistics can eventually be used into the a fit. In the following subsections we describe the most commonly used test statistics.

1.1.2 Weak variables and Overtraining

When choosing what kind of classifier you want to use in your analysis, you should keep in mind that everything brings you benefits, but also risks. The advantage of MVA classifier is, generally speaking, that it can achieve a better discrimination power with respect to a simple cut analysis, especially in presence of poorly discriminating variables. These variables are usually called *weak variables* and are characterized by having similar distributions for signal and background samples.

The main risk of using MVA classifier is *overtraining*. Overtraining happens if the classifier *overfits* data, so it looks at very special features of the training sample. Some wrong procedures can led to overtraining, such as the use of a training sample whose events aren't statistically independent (*oversampling*) or a wrong tuning of the classifier's parameters. The capability of estimating if a classifier is overtraining or not depends on the type of classifier. So when you have to choose what is the best classifier for your purposes, you have to pay attention to achieve a good discrimination power, but also to avoid overtraining.

Finally you also have to take into account the *transparency* of the classifier. If you say *"I call signal events those that have a value of the photon energy grater than 1.6* GeV*"*, everyone will understand what you are doing. If you said *"I train a neural network with 16 different variables and I call signal events those that have the output value of the network smaller than 0.3"*, someone can be a bit confused about what you are doing. So when choosing a classifier you should also take into account how easy is to understand what the classifier is doing.

1.1.3 Analysis Software

Many MVA software exist, both commercial and open-source. The two most used MVA software in the HEP community are:

- Tool for Multivariate Analysis (TMVA) [1]
- StatPatternRecognition (SPR) [4]

Both are open-source projects, the first being also included into ROOT [5]. The first one has the advantage of having an user-friendly graphical interface, the second one is, instead, little bit more complicated, but offers many powerful classifiers, not included into the first one.

1.2 Overview of MVA Techniques

1.2.1 Cut Selection

The simplest test statistics one can imagine is

$$\mathbf{t} = \mathbf{x}$$

which is at the origin of the simple cut analysis technique. The optimization relies on choosing the best set of $\mathbf{t_{cut}}$ cut values which maximize a predetermined FOM, usually

the statistical significance

$$SS = \frac{S}{\sqrt{S+B}} \tag{1.2}$$

where S and B are respectively the number of signal and background events after the cuts. In order to maximize the set of $\mathbf{t_{cut}}$ an algorithm called "bump hunting" is usually used[2]. The principle of the operations performed by such an algorithm is illustrated in fig. 1.1, for a 2–dimensional optimization. Anyway, such an algorithm can be applied to any n-dimensional problem. The algorithm starts with a box containing both all signal events (red points) and all background events (blue circles). Then the box is shrunk, moving one side per time, and at each step a fraction α of the events are *peeled* off; α is called *peel parameter* and the new box is chosen between all possible boxes according to the predetermined FOM. The process stops when the current box contain a minimum number of points, chosen by the user. After the shrinking process is terminated, the algorithm try to enlarge the box, if this enanche the FOM; this second process is called *pasting*.

1.2.2 Fisher Discriminant

Another simple *ansatz* for the test statistic is a linear combination of the variables

$$t(\mathbf{x}) = \sum_{i=1}^{n} a_i x_i = \mathbf{a^T x} \tag{1.3}$$

where $\mathbf{a^T}$ is the transposed of the coefficients vector. The goal of the minimization is to choose the a_i values that maximize the separation between $g(t|H_0)$ and $g(t|H_1)$. A possible approach, proposed by Fisher [6], is based on the following considerations. The data have the mean values and covariance matrix

$$\begin{aligned} (\mu_k)_i &= \int x_i f(\mathbf{x}|H_k) \mathrm{dx}_i ... \mathrm{dx}_n \\ (V_k)_{ij} &= \int (x-\mu_k)_i (x-\mu_k)_j f(\mathbf{x}|H_k) \mathrm{dx}_i ... \mathrm{dx}_n \end{aligned} \tag{1.4}$$

here i and j refers to the components of $\mathbf{x}$ and $k = 0, 1$ refers to the two different hypotheses H_0 and H_1. By the same way at each hypothesis corresponds different mean and covariance matrix for t distribution

$$\begin{aligned} \tau_k &= \int t\, g(\mathbf{x}|H_k) \Delta t = \mathbf{a^T} \mu_\mathbf{k} \\ \Sigma_k^2 &= \int (t-\tau_k)^2\, g(\mathbf{x}|H_k) \Delta t = \mathbf{a^T} V_k \mathbf{a} \end{aligned} \tag{1.5}$$

[2]Many implementations of such an algorithm exist. In our analysis we use the Bump Hunter algorithm implemented in `StatPatternRecognition` package [4]

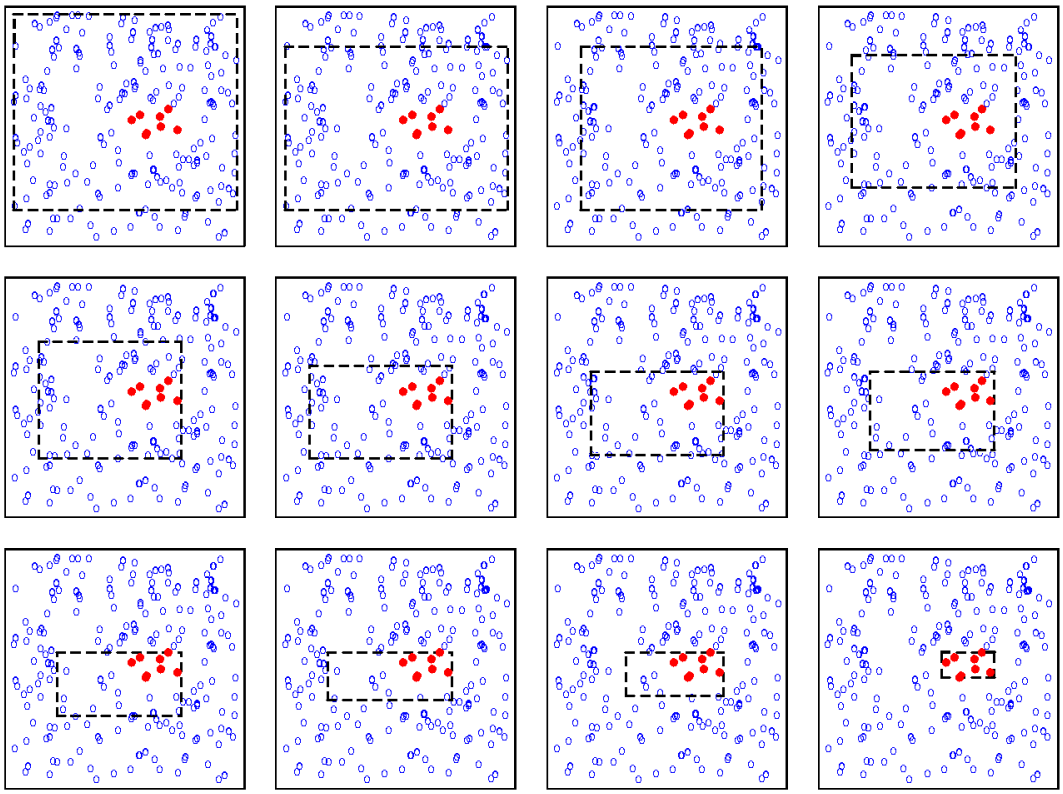

Figure 1.1: Schematic representation of the operation performed by a ”Bump Hunting” algorithm. Red points represents signal events, blue circles background ones. The black box represents acceptance region. The box is shrunk down at each step, removing a fraction α of the events, in order to maximize the chosen FOM.

To increase the separation we should clearly try to maximize $|\tau_0 - \tau_1|$ and at the same time we want to have $g(t|H_0)$ and $g(t|H_1)$ as slight as possible, this being controlled by Σ_0^2 and Σ_1^2. A possible measure of separation that takes into account both these requests is

$$J(\mathbf{a}) = \frac{(\tau_0 - \tau_1)^2}{\Sigma_0^2 + \Sigma_1^2} \tag{1.6}$$

the numerator can be expressed as

$$\begin{aligned} (\tau_0 - \tau_1)^2 &= \sum_{i,j=1}^{n} a_i a_j (\mu_0 - \mu_1)_i (\mu_0 - \mu_1)_j \\ &= \sum_{i,j=1}^{n} a_i a_j B_{ij} = \mathbf{a^T} B \mathbf{a} \end{aligned} \tag{1.7}$$

where matrix B defined as

$$B_{ij} = (\mu_0 - \mu_1)_i(\mu_0 - \mu_1)_j \tag{1.8}$$

represents the separation "between" the two classes corresponding to H_0 and H_1. Similarly the denominator of eq. 1.6 becomes

$$\Sigma_0^2 + \Sigma_1^2 = \sum_{i,j=1}^{n} a_i a_j (V_0 + V_1)_{ij} = \mathbf{a^T} W \mathbf{a} \tag{1.9}$$

here $W_{ij} = (V_0 + V_1)_{ij}$ represents the sum of covariance matrices "within" the two classes. Then eq. 1.6 can be expressed as

$$J(\mathbf{a}) = \frac{\mathbf{a^T} B \mathbf{a}}{\mathbf{a^T} W \mathbf{a}} \tag{1.10}$$

The minimization of this functions gives

$$\mathbf{a} \propto W^{-1}(\mu_0 - \mu_1) \tag{1.11}$$

so coefficients are determined up to an arbitrary scale factor. Usually this scale factor is absorbed into an offset so that the Fisher discriminant $\mathcal{F}$ is defined as

$$\mathcal{F}(\mathbf{x}) = a_0 + \sum_{i=1}^{n} a_i x_i \tag{1.12}$$

Using the Fisher discriminant is equivalent to apply a rotation plus a traslation in the $n-$dimensions variables space, or to choose a cut hyperplane which has a certain orientation and isn't simply defined as $x_i = 0,\ i \in [1, n]$; a graphic representation of this concept in 2-dimensions variable space is shown in fig. 1.2 One can prove that if the PDFs of the variables $f(\mathbf{x}|H_0)$ and $f(\mathbf{x}|H_1)$ are multigaussians with the same covariance matrix $V_0 = V_1 = V$ the Fisher discriminant perform as well as likelihood ratio. In particular we can observe that this is always true in case of uncorrelated variables, while if using correlated variables, one should be aware that the correlations are similar on signal and background events.

1.2.3 Decision Trees

The cut analysis method described in section 1.2.1 finds an acceptance region which is an hyper-rectangle in n dimensions. An obvious generalization is to try to divide the whole space in more regions (hyper-rectangles) that can be assigned to acceptance or rejection

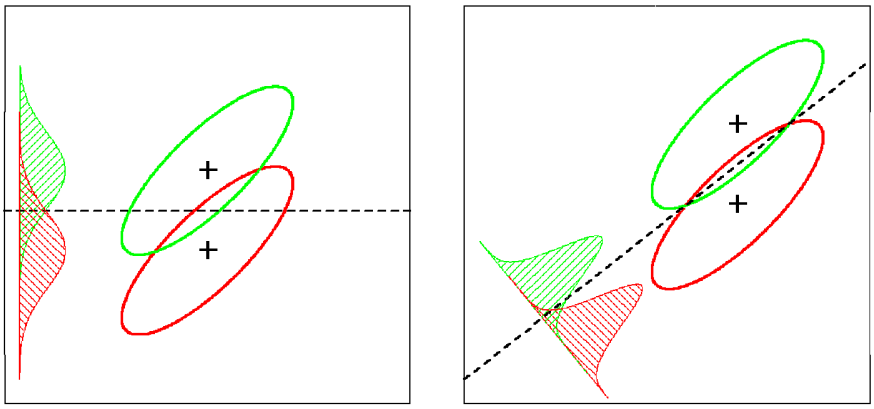

Figure 1.2: Example of the effect of using a Fisher discriminant, in green signal and in red background. In the left figure we can see that distributions doesn't have a good separation, so the power of the cut is limited. In the right figure using a variable obtained by a rotation plus a translation in the variable space, the separation between the variables improves.

region. Such an idea is implemented in the so called tree-based methods; applying a succession of binary cuts on a training sample, a tree-based classifier reach this goal and can then be used to test if an independent event falls in a signal (acceptance) or background (rejection) region. The main issue in using tree-based methods is the high variance of this method, *i.e.* the possibility of having very different results with a small change in the training sample; this problem is solved using the *boosting* and *bagging* algorithms described below. Like all complex classifiers, decision trees also suffer from *overtraining*, *i.e.* training on a specific sample, the algorithm can incur in an adaptation to very particular features of *that* sample and will overperform on it. This gives an artificious high performance on the training sample and possible low performance on other independent samples. There are many implementation both commercial and open-source of decision trees available[3], in the following paragraphs we discuss widely used techniques for training trees and avoiding overtraining.

Training a Decision Tree

We consider a training sample of n elements $\mathbf{x}$, the training starts with the root node, where an initial splitting criterion for the full training sample is determined. The split results in two subsets of training events, each of them goes through the same algorithm in order to determine the next splitting iteration. This procedure is repeated until the whole tree is built. At each node, the split is determined by finding the variable and corresponding cut value that provides the best separation between signal and background.

[3]For our studies we use TMVA [1] package which is now included into ROOT [5] distribution.

The node splitting is stopped once it has reached the minimum number of events. The end (*leaf*) nodes are classified as signal or background according to the class the majority of events belongs to.

A variety of separation criteria can be configured to assess the performance of a variable and a specific cut requirement. Because a cut that selects predominantly background is as valuable as one that selects signal, the criteria are symmetric with respect to the event classes. All separation criteria have a maximum where the samples are fully mixed, *i.e.*, at purity $p = 0.5$, and fall off to zero when the sample consists of one event class only. Tests have revealed no significant performance disparity between the following separation criteria:

- *Gini Index*, defined by $p \cdot (1 - p)$.
- *Cross entropy*, defined by $-p \cdot \ln(p) - (1 - p) \cdot \ln(1 - p)$.
- *Misclassification error*, defined by $1 - \max(p, 1 - p)$.

The splitting criterion being always a cut on a single variable, the training procedure selects the variable and cut value that optimizes the increase in the separation index between the parent node and the sum of the indexes of the two daughter nodes, weighted by their relative fraction of events. The cut values are optimized numerically, in figure 1.3 we show the shape of the above mentioned measures for a two class problem; the fact that *Gini Index* and *Cross entropy* are smooth functions of p, let them to be more amenable for numerical minimization.

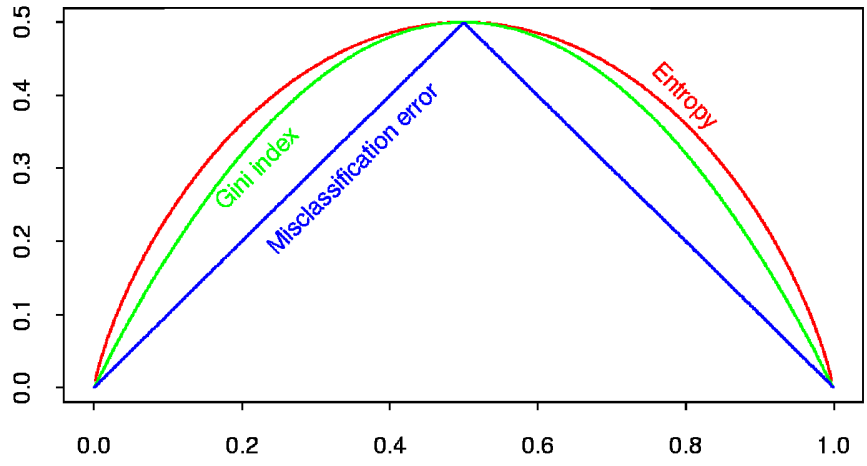

Figure 1.3: Node impurity measures, as function of purity of background events: misclassification error (blue), Gini index (green), entropy (red). Entropy has been scaled in order to pass through $(0.5, 0.5)$.

The choice of the maximum dimension of a tree should take into account two considerations: a very large tree might overfit the data, while a small tree might not capture the important structure. Tree size is a tuning parameter governing the model's complexity and is usually expressed as the minimum number of events in a node, when this minimum number is reached the node is not split anymore and is marked as *leaf* node. The preferred strategy is to grow a large size tree T_0 choosing a small value for this minimum and then prune the tree using *cost-complexity pruning*, that we now describe.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by collapsing any number of internal (not leaf) nodes of T_0. We index with m the leaf nodes, with node m representing a certain region R_m in the variable space. Let $|T|$ denote the number of such regions, so letting

$$\begin{aligned} \hat{c}_m &= \frac{1}{N_m} \sum_{\mathbf{x_i} \in R_m} y_i \\ Q_m(T) &= \frac{1}{N_m} \sum_{\mathbf{x_i} \in R_m} (y_i - \hat{c}_m)^2 \end{aligned} \tag{1.13}$$

where N_m is the total number of events $\mathbf{x_i}$ in the region R_m, we define the *cost-complexity* criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T| \tag{1.14}$$

The idea is to find for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$. The tuning parameter $\alpha > 0$ governs the tradeoff between tree size and its goodness of fit to the data. Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests $\alpha = 0$ correspond to the full tree.

For each α one can show that there is a unique smallest T_α that minimize $C_\alpha(T)$. In figure 1.4 we show the architecture of a pruned decision tree used as *anti-spam* filer.

1.2.4 Boosting and Bagging

Boosting is a general procedure whose application is not limited to decision trees[4][5]. The same classifier is trained several times using a successively boosted (reweighted) training event sample. The final classifier is then derived from the combination of all the individual classifiers. By this way we define a *forest* of decision tree; an event will then be processed by all the trees in the forest and the final output will be a proper average of results of each tree. The bottom line of this procedure is that starting with a weak classifier (*i.e.*

[4] Friedman states that "Boostinh is one of the powerful learning ideas in the last ten years."
[5] Recently the boosting procedure was also applied to neural networks and other classifiers

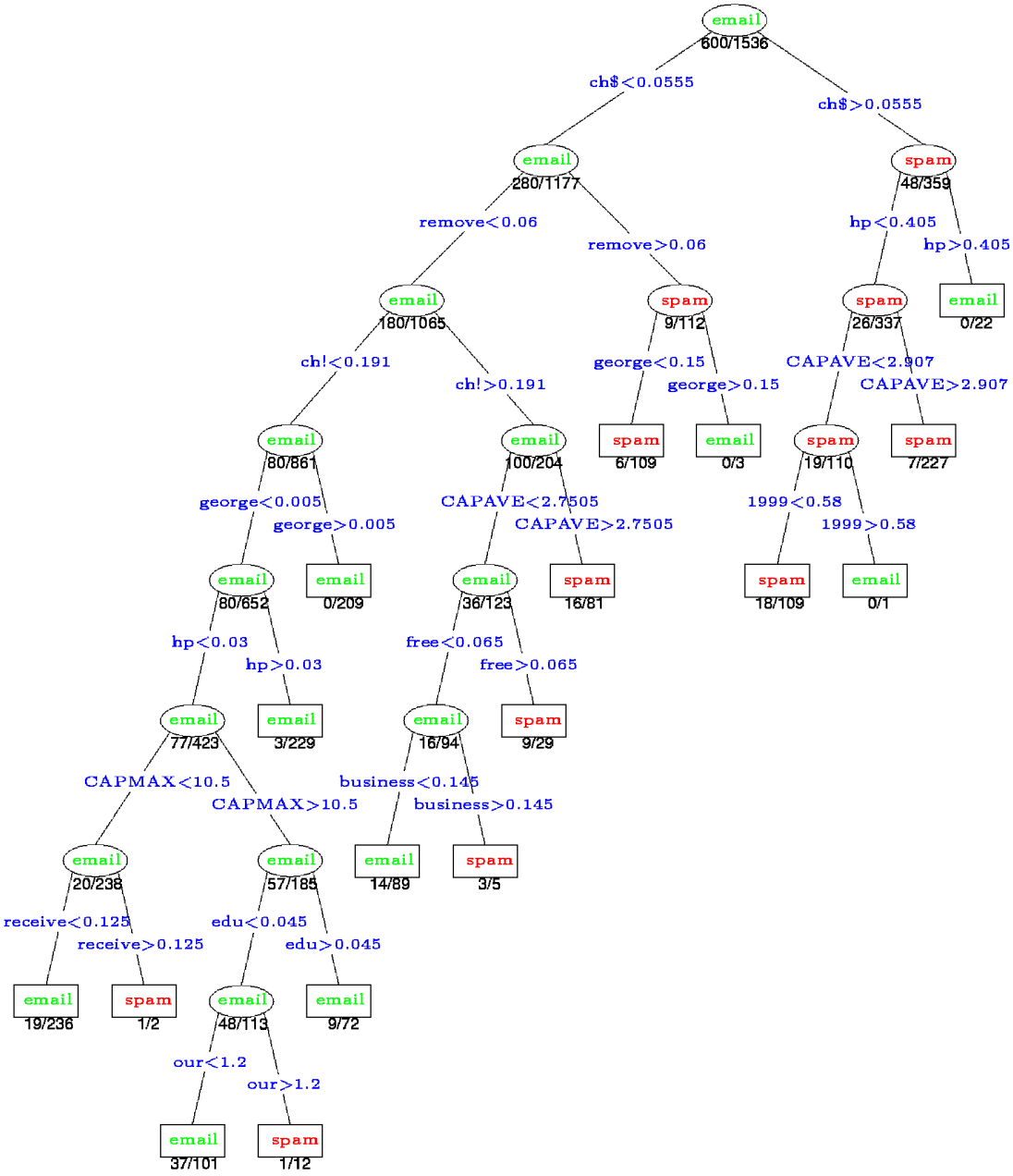

Figure 1.4: Example of a decision tree structure. This tree was trained in order to act as anti-spam filter.

one classifier that has a predictive power just a bit better than random guessing), you train it again with a reweighted sample and you finally give a higher influence to the most accurate classifiers. The most popular boosting algorithm is the so-called AdaBoost [7] (adaptive boost), where events that were misclassified during the training of a tree get a higher event weight in the training of the following trees. Starting with the original event weights when training the first decision tree, the subsequent tree is trained using a modified event sample where the weights of previously misclassified events are multiplied by a common boost weight β. The boost weight is derived from the misclassification rate w of the previous tree,

$$\beta = \frac{1-w}{w}$$

The entire event sample is then renormalized to keep the total number of events (sum of weights) in a tree constant. Letting the result $T(\mathbf{x})$ of an individual tree be $T(\mathbf{x}) = +1$ for signal and $T(\mathbf{x}) = -1$ for background, the resulting event classification $BDT(\mathbf{x})$ for the boosted classifier is then given by

$$BDT(\mathbf{x}) = \sum_{i \in forest} \ln(\beta_i) \cdot T_i(\mathbf{x}) \tag{1.15}$$

where the sum is over all trees in the forest. Small (large) values for $BDT(\mathbf{x})$ indicate a background-like (signal-like) event.

Bagging is a resampling technique, the result of such a technique is usually called *random forest*. The resampling is done with replacement, which means that the same event is allowed to be (randomly) picked several times from the parent sample. This is equivalent to regarding the training sample as being a representation of the PDF of the parent event ensemble. If one draws an event out of this ensemble, it is more likely to draw an event from a region that is high populated. If a selected event is kept in the original sample (that is when the same event can be selected several times), the parent sample remains unchanged so that the randomly extracted samples will have the same parent distribution, albeit statistically fluctuated. Training several decision trees with different resampled training data and combining them into a forest results in an averaged classifier that, just as for boosting, is more stable with respect to statistical fluctuations in the training sample. Technically the resampling is implemented by applying random weights to each event of the parent sample. Therefore the result of the classifier is given by eq. 1.15, yet.

1.2.5 Artificial Neural Networks

A more complex estimator that is able to manage also with highly correlated variables, or variables which have a very poor discrimination power is the *Artificial Neural Network* (ANN).The ANN is composed by some nodes called *neurons*, which are arranged in different layers and connected each other.

Network Architecture

While in principle a neural network with n neurons can have n^2 directional connections, the complexity can be reduced by organizing the neurons in layers and only allowing directional connections from one layer to the immediate next one[6]. This kind of neural network is termed *multilayer perceptron*. The first layer of a multilayer perceptron is the input layer, the last one the output layer, and all others are hidden layers. For a classification problem with n_{var} input variables and 2 output classes the input layer consists of n_{var} neurons that hold the input values, x_1 , ..., x_{nvar} , and one neuron in the output layer that holds the output variable, the neural net estimator y_{ANN}.

Each directional connection between the output of one neuron and the input of another has an associated weight. The output value of each neuron is multiplied with the weight to be used as input value for the next neuron. Each neuron as a *neuron response function* ρ, which maps the neuron input i_1 , ..., i_n onto the neuron output. Often it can be separated into a $\mathcal{R}^n \to \mathcal{R}$ synapsis function κ, and a $\mathcal{R} \to \mathcal{R}$ neuron activation function α, so that $\rho = \alpha \circ \kappa$. The functions κ and α can have the following forms:

$$\kappa \ : \ (y_1^{(l)}, ..., y_n^{(l)} | w_{0j}^{(l)}, ..., w_{nj}^{(l)}) \to \begin{cases} w_{0j}^{(l)} + \sum_{i=1}^{n} y_i^{(l)} w_{ij}^{(l)} & Sum \\ w_{0j}^{(l)} + \sum_{i=1}^{n} \left(y_i^{(l)} w_{ij}^{(l)} \right)^2 & Sum\ of\ Squares \\ w_{0j}^{(l)} + \sum_{i=1}^{n} \left| y_i^{(l)} w_{ij}^{(l)} \right| & Sum\ of\ Absolutes \end{cases} \tag{1.16}$$

$$\alpha \ : \ x \to \begin{cases} x & Linear \\ \frac{1}{1+e^{-kx}} & Sigmoid \\ \frac{e^x - e^{-x}}{e^x + e^{-x}} & Tanh \\ e^{-x^2/2} & Radial \end{cases} \tag{1.17}$$

where $y_i^{(l)}$ is the output of the i^{th} neuron in the l^{th} layer and $w_{ij}^{(l)}$ is the weight of the connection between the i^{th} neuron in the l^{th} layer and the j^{th} neuron in the $(l+1)^{th}$ layer.

When building a network one should keep in mind the theorem by Weierstrass, accertaining that for a multilayer perceptron a single hidden layer is sufficient to approximate a

[6] See for example fig. 2.8.

given continuous correlation function to any precision, given an arbitrary large number of neurons in the hidden layer. If the available computing power and the size of the training data sample are sufficient, one can thus raise the number of neurons in the hidden layer until the optimal performance is reached. It is possible that the same performance can be reached with a network with more than one hidden layer and a potentially much smaller total number of hidden neurons. This would lead to a shorter training time and a more robust network.

Neural Network Training

The most common algorithm for adjusting the weights that optimize the classification performance of a neural network is the so-called back propagation. It belongs to the family of supervised learning methods, where the desired output for every input event is known. The output of a network (here for simplicity assumed to have a single hidden layer with a $\tanh$ activation function, and a linear activation function in the output layer) is given by

$$y_{ANN} = \sum_{j=1}^{n_h} y_j^{(2)} w_{j1}^{(2)} = \sum_{j=1}^{n_h} \tanh\left(\sum_{i=1}^{n_{var}} x_i w_{ij}^{(1)}\right) \cdot w_{j1}^{(2)} \tag{1.18}$$

where n_{var} and n_h are the number of neurons in the input layer and in the hidden layer, respectively, $w_{ij}^{(1)}$ is the weight between input-layer neuron i and hidden-layer neuron j, and $w_{j1}^{(2)}$ is the weight between the hidden-layer neuron j and the output neuron. Simple summation was used in eq. 1.18 as synapsis function κ. During the learning process the network is supplied with N training events $x_a = (x_1, ..., x_{nvar})a \ , \ a = 1, ..., N$. For each training event a the neural network output $y_{ANN,a}$ is computed and compared to the desired output $y_a \in \{1, 0\}$ (1 for signal events and 0 for background events). An *error function* E, measuring the agreement of the network response with the desired one, is defined by

$$E(\mathbf{x_1}, ..., \mathbf{x_N}|\mathbf{w}) = \sum_{a=1}^{N} E_a(\mathbf{x_a}|\mathbf{w}) = \sum_{a=1}^{N} \frac{1}{2}(y_{ANN,a} - \hat{y}_a)^2 \tag{1.19}$$

where $\mathbf{w}$ denotes the ensemble of adjustable weights in the network. The set of weights that minimizes the error function can be found using the method of steepest or gradient descent, provided that the neuron response function is differentiable with respect to the input weights. Starting from a random set of weights $\mathbf{w}(\rho)$ the weights are updated by moving a small distance in w-space into the direction $-\nabla_{\mathbf{w}} E$ where E decreases most

rapidly

$$\mathbf{w}^{(\rho+1)} = \mathbf{w}^{(\rho)} - \eta \nabla_{\mathbf{w}} E \tag{1.20}$$

where η is a positive number called *learning rate*, which is responsible to avoid serious overtraining of the network.

The weights connected with the output layer are updated by

$$\Delta w_{j1}^{(2)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{j1}^{(2)}} = -\eta \sum_{a=1}^{N} (y_{ANN,a} - \hat{y}_a) y_{j,a}^{(2)} \tag{1.21}$$

and the weights connected with the hidden layers are updated by

$$\Delta w_{ij}^{(1)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{ij}^{(1)}} = -\eta \sum_{a=1}^{N} (y_{ANN,a} - \hat{y}_a) y_{j,a}^{(2)} (1 - y_{j,a}^{(2)}) w_{j1}^{(2)} x_{i,a} \tag{1.22}$$

where we have used $\tanh' x = \tanh x (1 - \tanh x)$.

This method of training the network is denoted *bulk learning*, since the sum of errors of all training events is used to update the weights. An alternative choice is the so-called *online learning*, where the update of the weights occurs at each event. The weight updates are obtained from eq. 1.21 and 1.22 by removing the event summations. In this case it is important to use a well randomized training sample.

Chapter 2

Application of MVA to a Real Analysis Case

2.1 Introduction

The goal of this chapter is to show one practical application of the techniques explained in 1. The physical case is taken from my personal analysis experience within the *BABAR* experiment. A bit introduction about *BABAR* Collaboration is then needed[1] in order to understand the following examples.

2.2 *BABAR* experiment

BABAR is a detector which surrounds the interaction point of PEP-II asymmetric-energy e^+e^- storage ring, operating at the Stanford Linear Accelerator Center. At PEP-II, 9.0 GeV electrons collide with 3.1 GeV positrons to yield a center-of-mass energy of $\sqrt{s} = 10.58\ \mathrm{GeV}$, which corresponds to the mass of the $\Upsilon(4S)$ resonance, which decays to a $B^0\overline{B}^0$ or a B^+B^- pair. The asymmetric energies result in a boost from the laboratory to the e^+e^- center-of-mass (CM) frame of $\beta\gamma \approx 0.56$. PEP-II data operations started in 1999 and ceased on 7$^{\mathrm{th}}$ April 2008. Most of the data were taken at the $\Upsilon(4S)$ resonance (on-peak). However approximately 10% were taken at 40 MeV below the resonance peak (off-peak), where there is not $\Upsilon(4S)$ resonance production, to allow studies of non-resonant background in data. In the last period of operation PEP-II also ran at $\Upsilon(3S)$ and $\Upsilon(2S)$ resonance energy, collecting the two largest world datasets ever recorded at those resonances. A scan over $\Upsilon(4S)$ region was also performed. The total integrated luminosity during the duration of the experiment was 432.89 fb^{-1} at $\Upsilon(4S)$ resonance, 30.23 fb^{-1}

[1] But not required for the final exam.

at $\Upsilon(3S)$ resonance, 14.45 fb^{-1} at $\Upsilon(2S)$ resonance and 53.85 fb^{-1} at off-peak energy.

A detailed description of the *BABAR* detector (see fig. 2.1) can be found elsewhere [16]. Surrounding the interaction point is a five-layer double-sided silicon vertex tracker (SVT) that provides precision measurements near the collision point of charged particle tracks in the planes transverse to and along the beam direction. A 40-layer drift chamber (DCH) surrounds the SVT. Both of these tracking devices operate in the 1.5 T magnetic field of a superconducting solenoid to provide measurements of the momenta of charged particles. Charged hadron identification is achieved through measurements of particle energy loss in the tracking system and the Cherenkov angle obtained from a detector of internally reflected Cherenkov light (DIRC). A CsI(Tl) electromagnetic calorimeter (EMC) provides photon detection, electron identification, and π^0, η, and K_L^0 reconstruction. Finally, the instrumented flux return (IFR) of the magnet allows discrimination of muons from pions and detection of K_L^0 mesons. For the first 214fb^{-1} of data, the IFR was composed of a resistive plate chamber system. For the most recent 212fb^{-1} of data, a portion of the resistive plate chamber system has been replaced by limited streamer tubes.

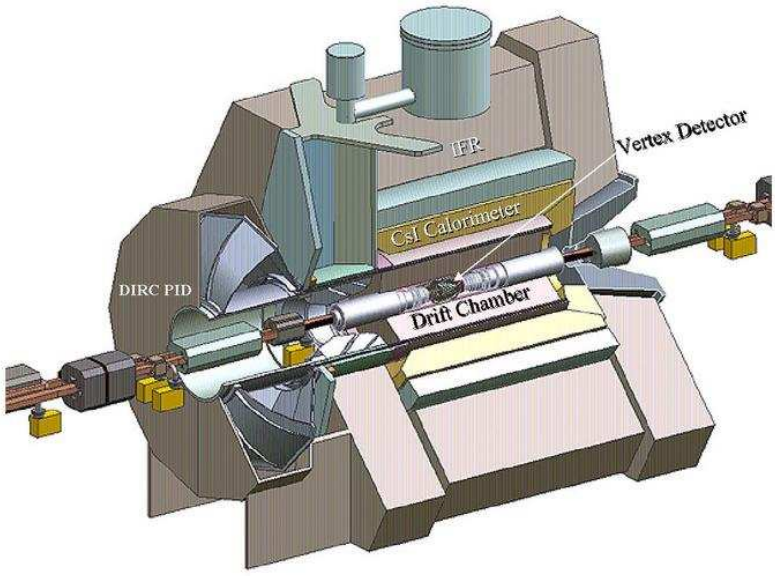

Figure 2.1: *BABAR* detector where each subdetector is indicated.

2.3 Use of an ANN for K_L^0 Background Suppression

2.3.1 Physical Case

The K_L^0 meson is a neutral particle which is a superimposition of K^0 and $\overline{K^0}$ mesons. K_L^0 is mainly studied in its dominant decay $K_L^0 \to \pi\, l \nu_l$. Mean lifetime of K_L^0 meson is $\tau = (5.116 \pm 0.020) \times 10^{-8} s$, so the K_L^0 decay length is something about 15 m. Then K_L^0 mesons escape outside the *BABAR* detector before it decays and we can consider it as a long-time living neutral particle.

Inside the *BABAR* detector K_L^0 can be identified in the EMC calorimeter or in the IFR. If the K_L^0 is identified in the IFR, we are pretty sure that it's a *real* K_L^0, because misidentification with a muon is quite difficult. Unfortunately IFR has a spatial resolution of about 15 cm, so it would be better to identify the K_L^0 in the EMC, which has a spatial resolution of some 5 cm. Unfortunately in the EMC we also detect photons that can be misidentified as K_L^0 (*"fake" K_L^0*). Anyway, there are many variables, that describe the electromagnetic shower shape in the EMC, which can help in γ/K_L^0 separations.

This selection was optimized for the study of $B^0 \to \eta' K_L^0$ decay [17].

2.3.2 Variables Used

EMC variables used for our classifiers are:

- *Number of crystals* in the bump.
- *Total Energy* of the bump.
- *Second moment*, defined as:
$$\mu_2 = \frac{\sum E_i \cdot r_i^2}{\sum E_i},$$
where E_i is the energy of the $i-th$ crystal and r_i its distance from the cluster center.
- *Lateral moment* , defined as:
$$\mu_{LAT} = \frac{\sum_{i=2,n} E_i \cdot r_i^2}{(\sum_{i=2,n} E_i \cdot r_i^2) + 25(E_0 + E_1)},$$
where E_0 refers to the most energetic crystal and E_n to the least energetic one.
- *S1/S9:* The energy of the most energetic crystal (S1) divided by the energy sum of the 3x3 crystal block (S9) with the most energetic crystal in its center.

– *S9/S25:* The energy sum of the 3x3 crystal block (S9) with the most energetic crystal in its center, divided by the energy sum of the 5x5 crystal block (S25) with the most energetic crystal in its center.

– *Zernike moments* $|Z_{20}|$, $|Z_{42}|$ of the spatial energy distribution of the EMC cluster expressed as a series of Zernike polynomials

$$\zeta' : E(x,y) = \sum_{n,m} Z_{n,m} \cdot \zeta'_{n,m}(r,\phi)$$

In fig. 2.2 and 2.3 you can find input variable distributions comparison for signal MC, real sideband data and off-peak data (real data taken where the signal process is kinematically forbidden). In fig. 2.4 you can find correlation matrix coefficients for input variables.

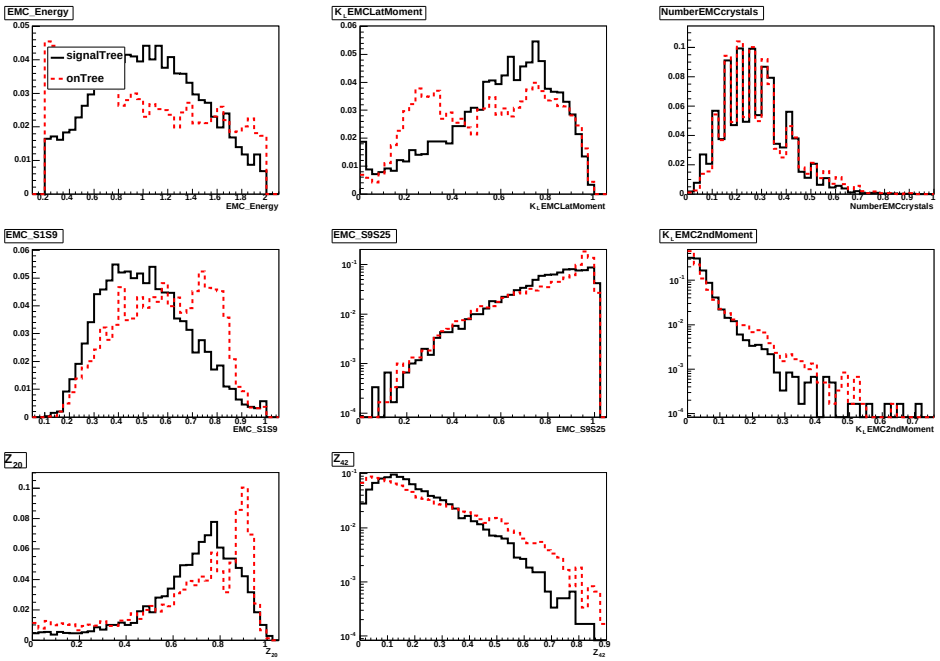

Figure 2.2: ANN input variables comparison between signal MC (continuum black line) and real sideband data (dashed red line).

2.3.3 Train and Test

We use TMVA [1] to train, test and evaluate our classifiers. The training configuration is the following: 3000 events for both signal and background samples as training samples and independent 3000 events for both signal and background for validation. We use real sideband events as background and MC signal events as signal[2].

[2] $B \to \eta'_{\eta(\gamma\gamma)\pi\pi} K^0_L$ decay mode

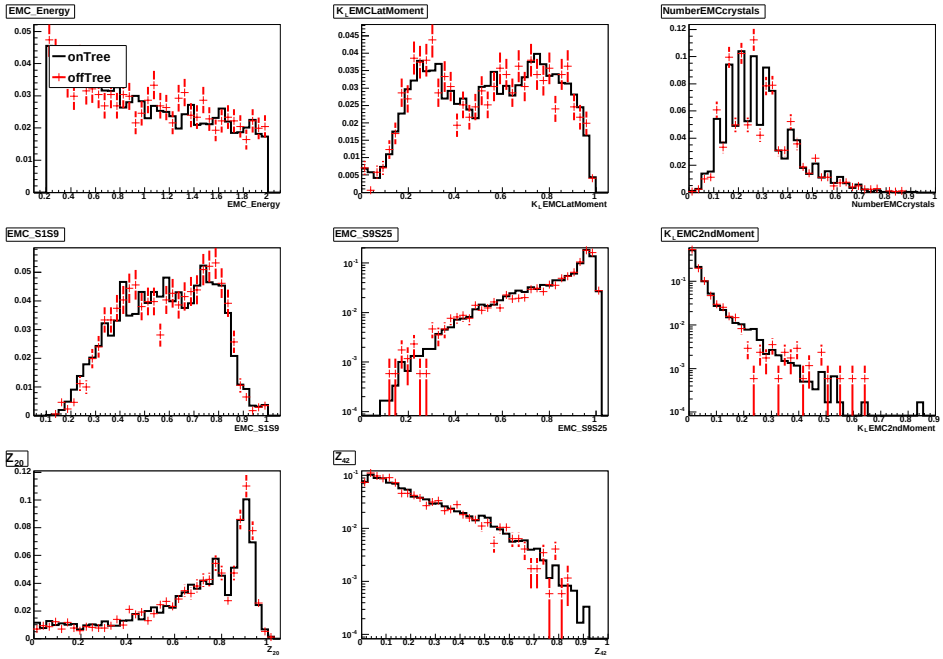

Figure 2.3: ANN input variables comparison between real sideband data (continuum black line) and off-peak data (red points).

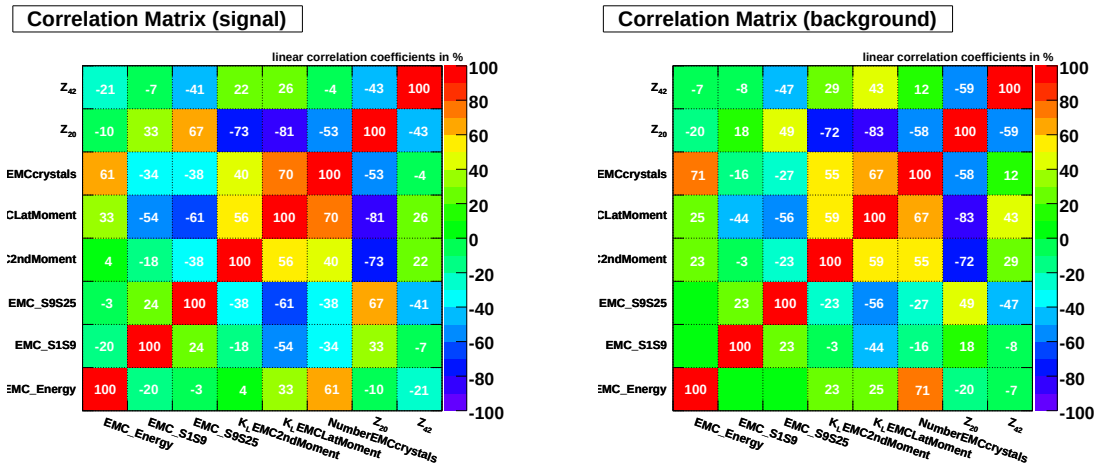

Figure 2.4: Correlations between the EMC variables used in the ANN for signal MC and real sideband data events sets.

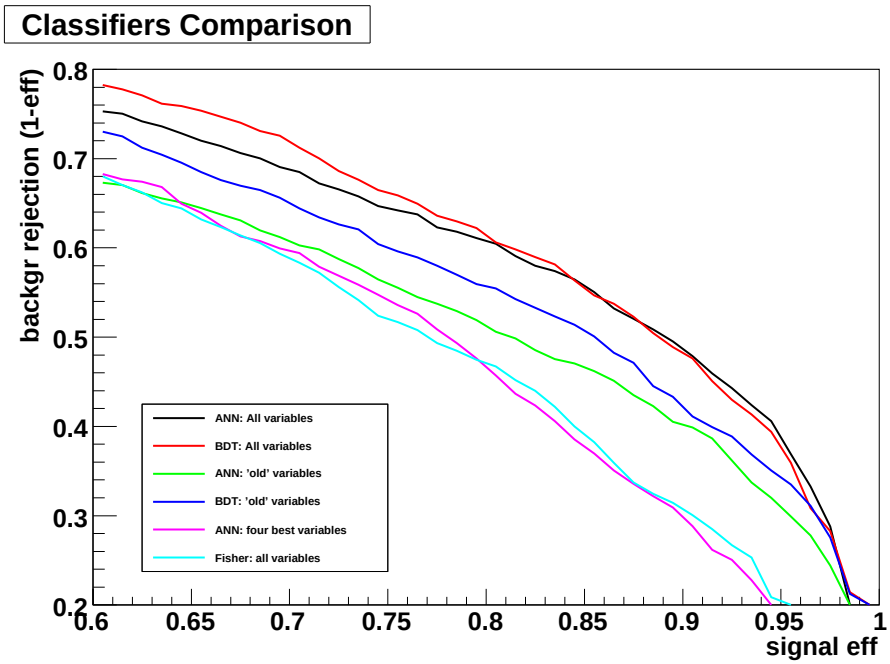

Figure 2.5: ROC curves (signal efficiency vs background rejection) for many choices of ANN and BDT input variables: ANN/BDT All variables (black/red), ANN/BDT without EMC Total Energy (green/blue), ANN four best ranked variables (violet).

We find TMVA Artificial Neural Network and Boosted Decision Tree to be most performing methods. In fig. 2.5 we show various ROC curves (signal efficiency vs background rejection) for different input variables for ANN and BDT. It's easy to see that adding EMC total energy as input variables improve a lot the discriminating power of the classifiers. We perform further tests to investigate if a different choice of classifier parameters would improve discriminating power. Results of these tests are shown in fig. 2.6 and 2.7.
We find that BDT with 600 trees in the forest and prune strength of 3 have some 2-3% discriminating power more than ANN with one single layer with N+1 nodes. It's important to notice that our goal is to cut on the classifiers output in order to suppress background, so our interest is focused on high signal efficiency region. In the choose of what method to use, one must consider that increasing the forest dimension and decreasing the prune strength may cause BDT overtraining, and that there is no standard and simple procedure to evaluate if a BDT is overtrained or not. Due to all these facts, we agreed that this little improvement given by BDT is not sufficient to justify a big effort in trying to estimate if BDT is overtrained.

Finally, we decided to use ANN for signal-background rejection. Our ANN has one hidden layer with 10 neurons, learning parameter is 0.05 and we have performed 500 cycles for the training. In fig. 2.8 you can see the network architecture. In fig. 2.9 we show the Neural Network errors with respect to the training epoch. The ANN is trained to return -1 for background-like events and $+1$ for signal-like ones; output of the Neural

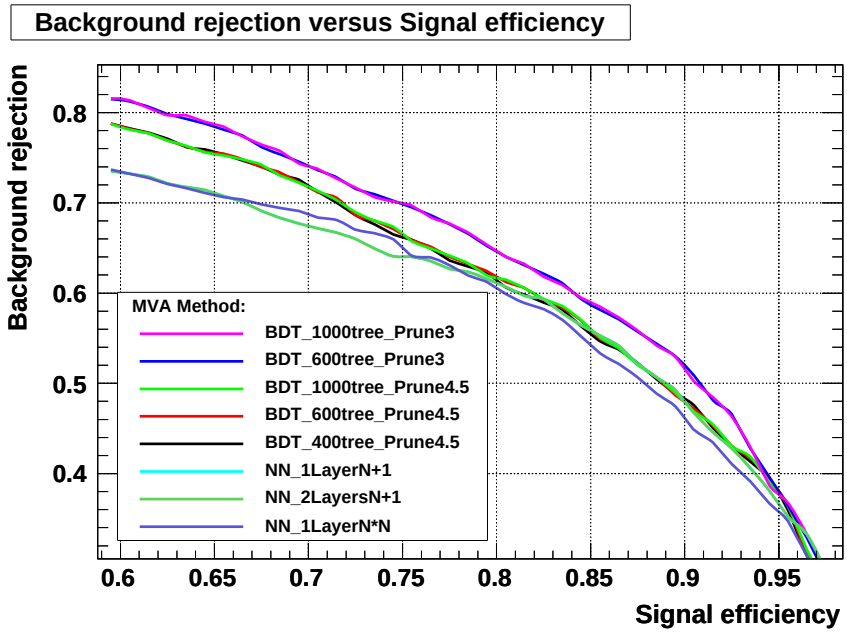

Figure 2.6: ROC curves (signal efficiency vs background rejection) for many choices of ANN and BDT architecture and training parameters using all input variables: BDT 1000 tree, prune strength 3 (violet); BDT 600 tree, prune strength 3 (blue); BDT 1000 tree, prune strength 4.5 (green) BDT 600 tree, prune strength 4.5 (red); BDT 400 tree, prune strength 4.5 (black); ANN 1 layer, N+1 neurons (azure); ANN 2 layers, N+1 neurons per layer (dark green); ANN 1 layer, N^2 neurons (dark blue).

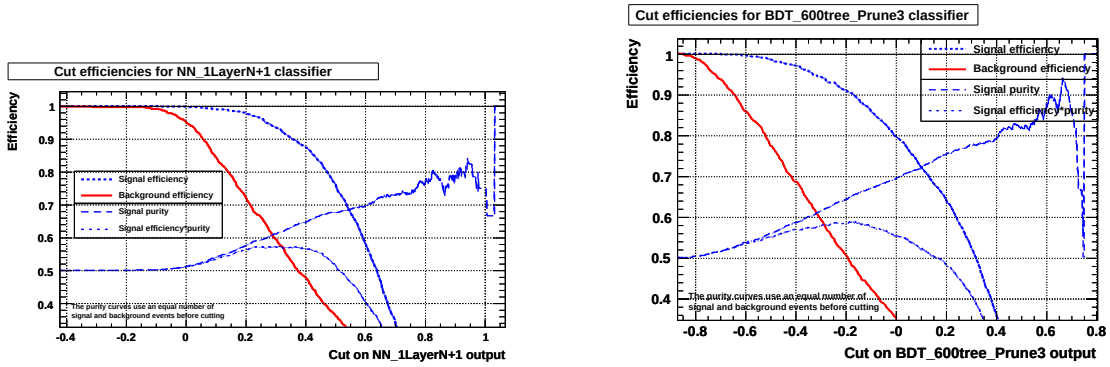

Figure 2.7: Cut efficiency for signal purity (blue dashed), signal efficiency (blue dotted) and background efficiency (red solid) for BDT and ANN with best variables (all) and architecture configuration (ANN:1 layer N+1 neurons;BDT: 600 trees, prune strength 3).

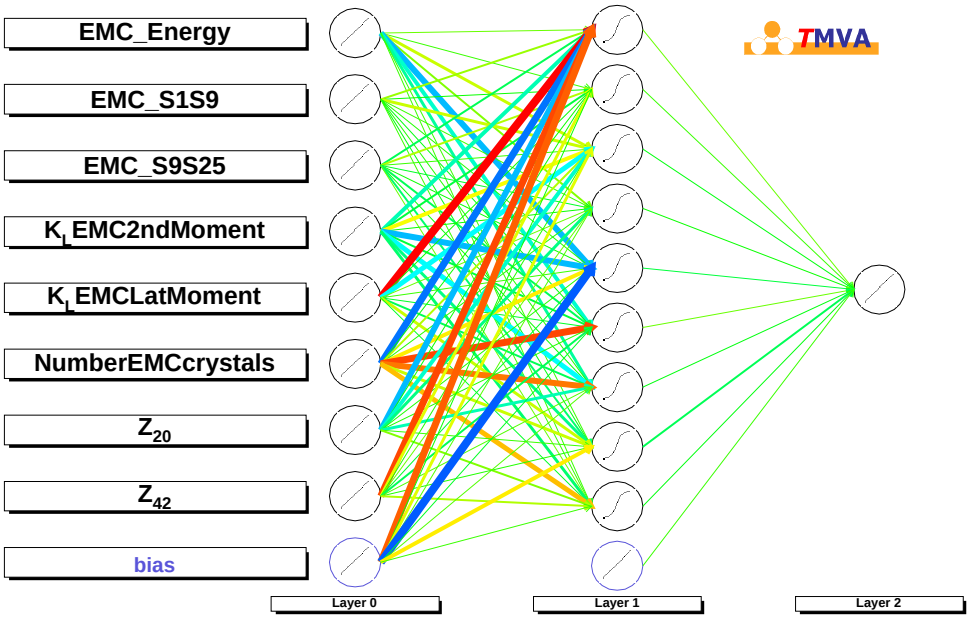

Figure 2.8: ANN Architecture

Network is shown in fig. 2.10.

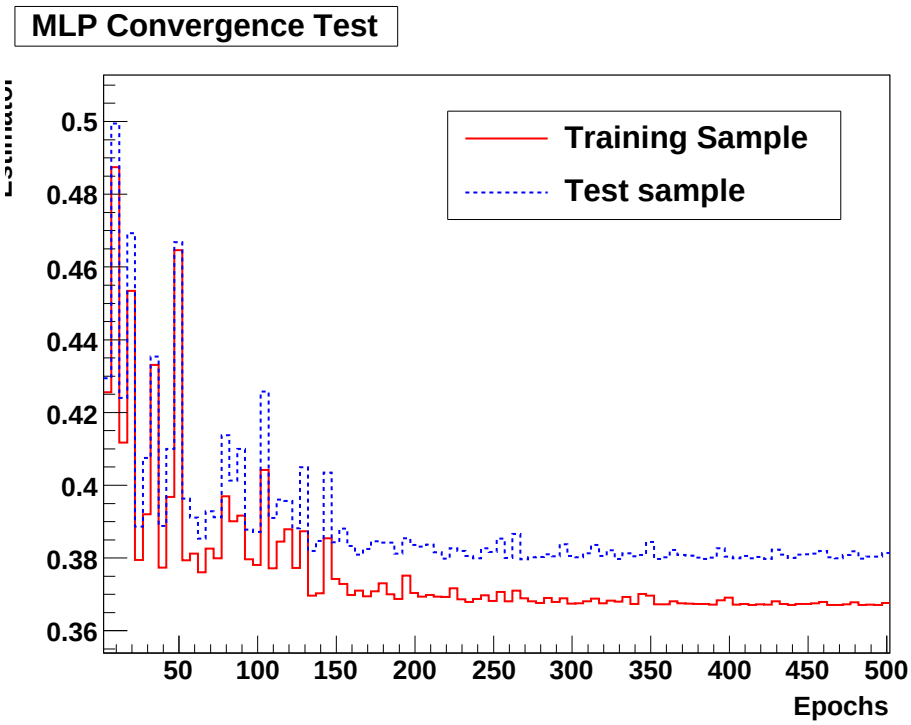

Figure 2.9: ANN Training (solid red) and testing (dashed blue) output respect to training epoch.

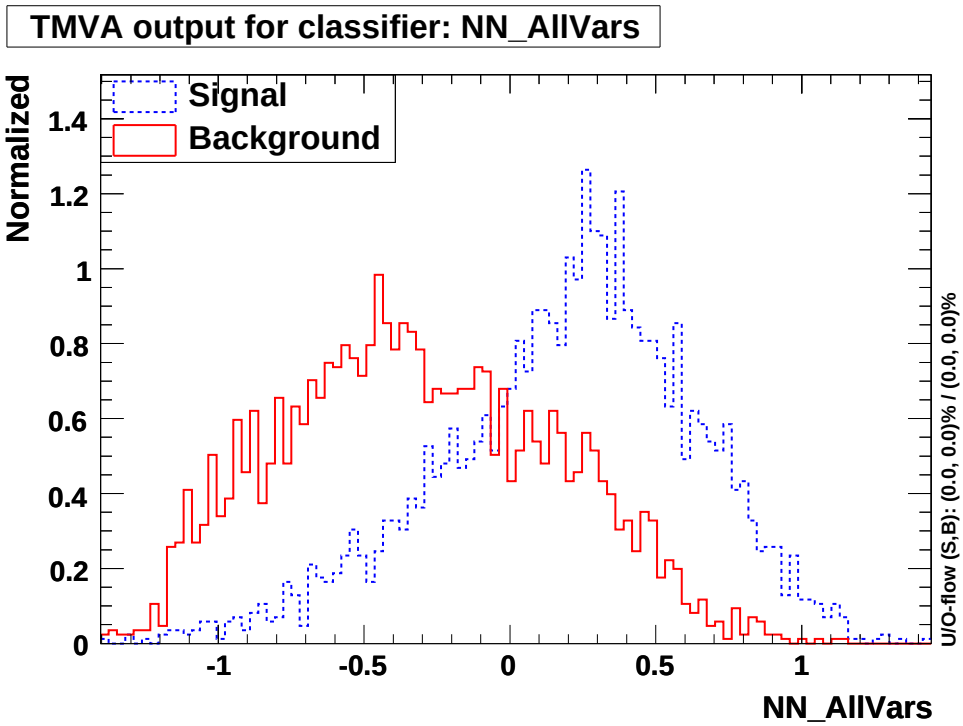

Figure 2.10: ANN output for signal (dashed blue) and background (solid red) events.

2.3.4 Overtraining

To have a clear example of overtraining look at fig. 2.11 The red line refers to the performance of a boosted decision tree trained with a large value of the forest size (1000 trees) and a small value of the prune parameter (2). Its extra-performance is due to the fact that it is overfitting training data, real performance over real data is difficult to extimate.

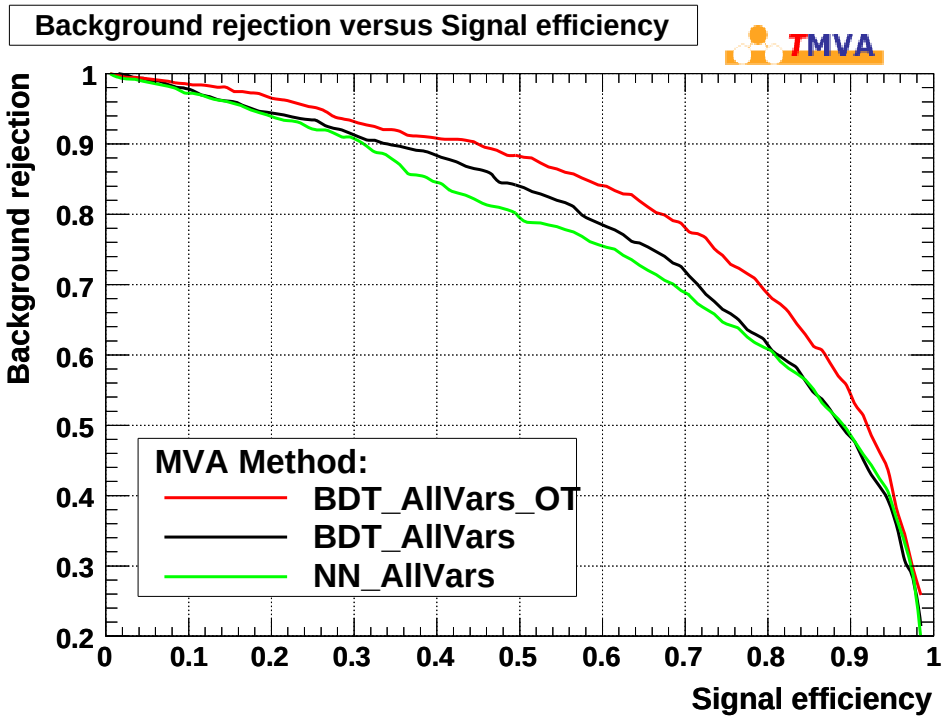

Figure 2.11: ROC curves (signal efficiency vs background rejection) for many choices of ANN and BDT architecture and training parameters using all input variables: BDT 1000 tree, prune strength 2 (red); BDT 400 tree, prune strength 4.5 (black); ANN 1 layer, N+1 neurons (green).

Bibliography

[1] For a description of TMVA package please refer to URL:http://tmva.sourceforge.net/.

[2] J. Friedman *et al.* , *The Elements of Statistical Learning*, Springer, New York (2001).

[3] S. Brandt, *Datenanalyse*, BI-Wissenshaftsverlag, Mannehim (1992); S. Brandt, *Statistical and Computational Methods in Data Analysis*, Springer, New York (1997).

[4] For a description of StatPatternRecognition package please refer to URL:https://sourceforge.net/projects/statpatrec.

[5] ROOT project web site:
URL:http://root.cern.ch.

[6] R. A. Fisher, Ann. Eugen. **7**, 179 (1936)

[7] Y. Freund *et al.* , F. of Computer and System Science **55**, 119 (1997)

[8] L. Demortier, *P-values, What They Are and How to Use Them* http://www-cdf.fnal.gov/ luc/statistics/cdf0000.ps

[9] M. Pivk and F. R. Le Diberder, Nucl. Instrum. Methods Phys. Res., Sect. A **555**, 356 (2005).

[10] See the web page the PAW project:
URL:http://wwwasd.web.cern.ch/wwwasd/paw.

[11] See the web page of the CINT project:
URL:http://root.cern.ch/root/Cint.html.

[12] See the web page of the RooFit project:
URL:http://roofit.sourceforge.net/.

[13] F. James, *MINUIT - function minimization and error analysis*, CERN Program Library Long Writeup D506.

[14] See the web page of the `MiFit` project:
`URL:http://lxmi.mi.infn.it/~lazzaro/MiFit/index.html`.

[15] W.-M. Yao *et al.*, J. Phys. G **33**, 1 (2006).

[16] B. Aubert *et al.*,*BABAR* Collaboration, Nucl. Instrum. Methods Phys. Res., Sect. A **479**, 1 (2002).

[17] B. Aubert *et al.*, *BABAR* Collaboration, hep-ex/0809.1174