A machine learning approach to particle physics data analysis: the process $J/\psi \to \gamma p\bar{p}$

Master's thesis

Tommaso Isolabella

KVI-CART
University of Groningen

Contents

Chapter 1

Introduction

1.1 Big data challenges

Particle physics is the study of the smallest constituents of matter and of their interactions. In order to improve our understanding of phenomena at the very small scale, experiments have become larger and more complex. While a small cloud chamber was enough, in 1933, to discover the positron, current detectors in particle physics are close in size to cathedrals, and are made up of many different layers, each with their own specific function (see for example [1] or [2]).

With the increase in complexity of the experiments comes an increase in the volume of data that needs to be analysed to extract meaningful information. This is desirable, since large amounts of data generally reduce the statistical uncertainty on the results and allow the search of reactions with tiny cross sections. Physicists are often interested in one particular decay (*channel*) that only makes up a tiny fraction of the total collected data, and the *signal* they look for might be hidden under an enormous mole of *background* events. It is therefore interesting to investigate new approaches for the analysis of these data. One of them is *machine learning*, a class of analysis techniques that excel at handling large data samples. The aim of my Master's project, as reported in the following chapters, is to study the feasibility of this approach applied to the $J/\psi \to \gamma p\bar{p}$ channel at the BESIII experiment.

1.2 Charmonium and glueballs

The J/ψ particle was discovered in 1974 [3]. It was the first meson to be discovered composed of a charm-anticharm quark pair. Since then, many other $c\bar{c}$ states have been found, and they have been given the generic name of *charmonium*. The name emphasises the strong similarity with *positronium* (e^+e^- bound states). In addition to their electric charge, quarks and antiquarks also carry a color charge. Studying charmonium, therefore, allows to probe phenomena governed by the strong force such as color confinement.

The electromagnetic force is mediated by the photon which, being neutral, is not subject to self-interaction. The strong force, on the other hand, is carried by gluons which are themselves color-charged. This opens up the possibility for gluons to interact and to form bound states, called *glueballs* (see for example [5]). These states are predicted by theory, in particular by Lattice Quantum Chromodynamics (LQCD, [6]), but their existence has not yet been unambiguously observed. One of the possible channels that lead to the creation of glueballs is $e^+e^- \to J/\psi \to \gamma G \to \gamma p\bar{p}$, where G indicates a glueball state as illustrated in Figure (1.1).

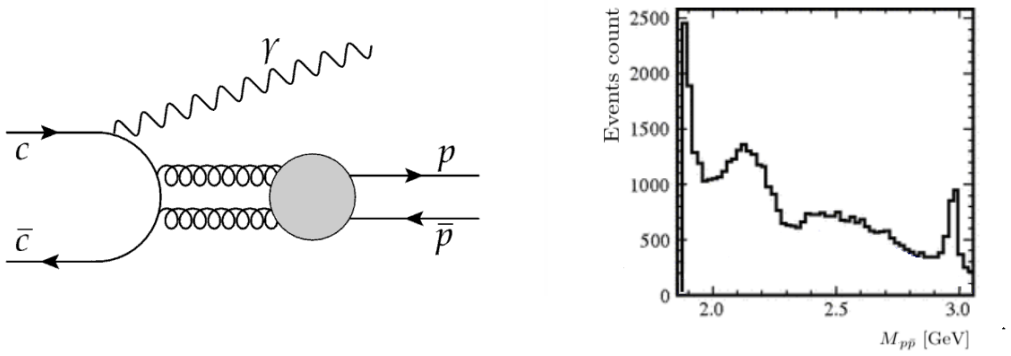


Figure 1.1: On the left-hand side, the Feynman diagram of a possible process where a glueball intermediate resonance is created decaying into a $p\bar{p}$ pair. On the right-hand side, a typical spectrum for the invariant mass of $p\bar{p}$ in $J/\psi \to \gamma p\bar{p}$ is shown. The leftmost peak, near the $p\bar{p}$ creation threshold, is unexpectedly enhanced and is therefore a glueball candidate. The rightmost peak corresponds to the ground-state of charmonium, η_c (*Feynman diagram courtesy of Rosa Kappert, KVI-CART. Spectrum adapted from [4]*).

The BESIII experiment, located at the Beijing Electron Positron Collider, is an excellent J/ψ production factory, with 10^{10} events recorded during its operation. Electron beams collide with positron beams at a center-of-mass energy corresponding to the J/ψ mass, $M_{J/\psi} = 3.097$ GeV. The annihilaion of the e^+e^- pair leads to the production of a $c\bar{c}$ state among other less interesting events such as Bhabha pairs. It is therefore the perfect place to look for glueball resonances produced by $c\bar{c}$ mesons. The $\gamma p\bar{p}$ decay channel was chosen because the reversed process can be observed at proton-antiproton colliders such as $\bar{\text{P}}$ANDA [7].

In order to extract relevant physical information from the data, such as the spin and the parity of a state, the nature of the events must be known; in other words, events need to be classified. Efficient pre-selection algorithms rule out most of the background, restricting the data sample to events that look like the channel of interest. To maximise the statistical significance of the data, an optimal discrimination between signal and background is necessary. Eventually, a detailed partial-wave analysis of the remaining data set allows the extraction of the physical properties of intermediate resonances. The data division into signal and background using machine learning techniques is the topic of my thesis.

Many high-energy physics experiments in the world already employ machine learning in at least some stages of data analysis ([8], [9], [10], [11]). Tasks fulfilled by multivariate algorithms range from real-time triggering and online data selection to high-level offline analysis. It is therefore interesting to see how feasible it will be to apply these methods for the identification of the $J/\psi \to \gamma p\bar{p}$ process in BESIII, by studying the background suppression they can achieve. So far, the BESIII collaboration has not considered machine learning tools for this kind of analysis in their published work.

The standard approach to signal to background optimisation is to make simple cuts that define an acceptance region. These cuts are usually performed on one or two variables, and in order to be applied to experimental data they are optimised with a Monte Carlo data study. Due to its multivariate nature, a machine learning approach can improve the background suppression by looking at several variables at the same time. In particular, it finds the acceptance regions in a high-dimensional feature space that maximise the statistical significance for the signal.

To summarise, in my thesis I will address the following questions:

1. Is machine learning a feasible approach to increase the signal to background ratio for $J/\psi \to \gamma p\bar{p}$?
2. What are the variables that have the highest discriminating power for this task?
3. Is there a significant discrepancy between the performance of two different algorithms, namely boosted decision trees (BDT) and artificial neural networks (ANN)?

1.3 The BESIII experiment

In my analysis I have made use of both Monte Carlo and experimental data. The former was produced through simulations of response of various detectors at BESIII, while the latter was collected in the 2009 and 2012 runs of the experiment. The data that were used as input to the analysis methods in this work consist of reconstructed four-momenta of preselected events, composed of two charged tracks and a photon candidate. BESIII is equipped with a variety of sub-detectors that provide the necessary input to reconstruct these variables [1]. In the following paragraphs I will briefly describe the detector components that were used to select the events that were used in this work.

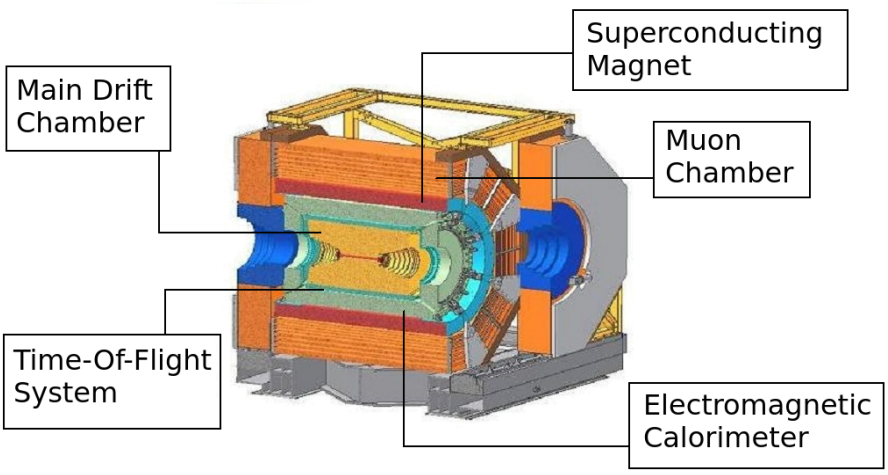


Figure 1.2: A schematic view of the BESIII detector. Shown with a red line at the center of the detector is the pipe where the e^+ and e^- beams travel and collide. In addition to the detector components described in the text, the image also shows the Muon Chamber, used for identification and tracking of muons, and the Superconducting magnet used to create the strong magnetic field required for momentum measurement. *Image from [12].*

The Main Drift Chamber (MDC) is the innermost detector. It is composed of two cylindrical elements, coaxial with the beam pipe, containing a total of 43 cylindrical layers of drift cells. Through a multi-wire system, the MDC can determine the momenta of charged particles that pass through it by reconstructing their trajectory in a magnetic field. Furthermore, measuring how much energy these particles deposit during their flight, the instrument can also determine the identity of the particle (PID). Further information on trajectory fitting can be found in [13].

The Time-Of-Flight (TOF) system is made to identify the nature of charged particles that drift through it. It consists of plastic scintillator bars which are read by photomultiplier tubes.

Together with the MDC energy deposit measurement, the TOF system contributes to the PID.

The Electromagnetic Calorimeter (EMC) is the third cylindrical detector, and it is made for the detection electrons, positrons and photons. In particular, the EMC is built with rings of CsI(Tl) crystals that measure the energy and position of electrons positrons and photons through the identification of electromagnetic showers initiated by the primary particle [14].

1.4 Definitions

In the following chapters I will be using the following quantities as inputs to the machine learning algorithms and for other analysis purposes:

1. The four-momentum of a particle with energy E and momentum $\mathbf{p}$:

$$q = (E, \mathbf{p});$$

2. the invariant mass of the final-state particles in $J/\psi \to \gamma p\bar{p}$:

$$M_{\gamma p\bar{p}} = \sqrt{\left(E_\gamma + E_p + E_{\bar{p}}\right)^2 - \left(\mathbf{p}_\gamma + \mathbf{p}_p + \mathbf{p}_{\bar{p}}\right)^2};$$

3. the invariant mass of a proton-antiproton system:

$$M_{p\bar{p}} = \sqrt{\left(E_p + E_{\bar{p}}\right)^2 - \left(\mathbf{p}_p + \mathbf{p}_{\bar{p}}\right)^2};$$

Throughout the thesis I will use natural units, whereby $c = 1$. Therefore momentum and mass will be expressed, for example, in GeV instead of GeV/c and GeV/c^2, respectively.

Chapter 2

Machine learning algorithms

2.1 General machine learning concepts

Machine learning embeds various techniques whose ultimate goal is to extract meaningful information from a data sample. A machine learning algorithm (in the following also *multivariate method* or *MVA method*) is a mathematical model that contains parameters. These parameters are optimised so that the model can describe the data efficiently. The phase in which the best values for the parameters are found is known as *training*: it is when the algorithm learns from the data. After the phase of learning, the algorithm can make predictions on unknown data. This phase is often referred to as *application*.

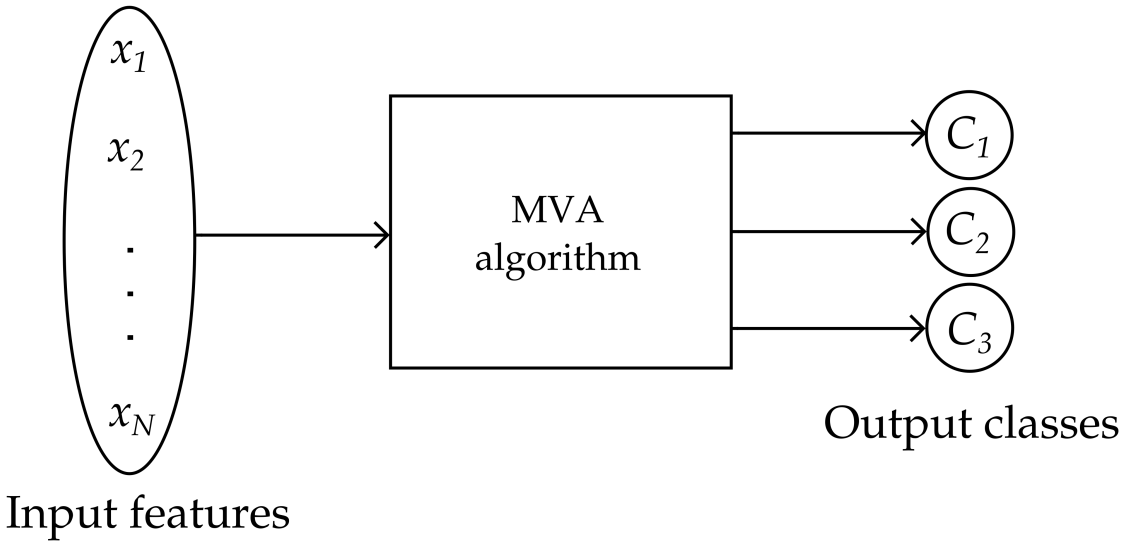


Figure 2.1: A schematic representation of a machine learning algorithm. N input features are provided, and the multivariate method assigns each event in the sample to one of three classes.

Algorithms take as input a set of variables for each data point; these variables are commonly referred to as *features*, and they carry information about the properties of each data point. These features can be, for example, the energy and momenta of particles in a particular reaction, where each data point represents a single event detected in the experiment.

A preliminary subdivision between machine learning algorithms is whether they implement *supervised* or *unsupervised* learning. The former requires each data point to be labeled with the

desired output the algorithm should produce when fed that particular event. The multivariate method learns by adjusting it parameters in order to minimise the error between its output and the label. Unsupervised learning, on the other hand, is used when there is no labeling available for the data. Events are grouped based on similarities in their features. Unsupervised learning methods are often used to preprocess the data before using supervised algorithms. These methods do not require a training phase, and are sometimes referred to as *clustering* methods.

Machine learning can fulfil different tasks based on the required output. The simplest one is when the data belong to two different classes (for example signal and background). This case is called binary classification, and it is the case discussed in this work. Another task is to classify data according to more than two classes. Figure (2.1) shows a three-class algorithm. This kind of multi-class classification is not conceptually different than binary classification, since any k-class problem can be divided into k two-class problems. It is, however, generally simpler to directly implement a multi-class algorithm instead of relying on this theoretical fact. The third task a machine learning algorithm can fulfil is regression, when the output is continuous. Generally, the output of a classifier is also continuous, and can be interpreted as the probability of a data point to belong to a class. The user can then perform a cut on the output that maximises the performance of the classifier according to some criteria of choice.

Two important aspects to consider when building a supervised machine learning algorithm are the *bias* and the *variance*. The bias is the difference between the output of the MVA method for a given data point and the label of the event. It is also known as training error. It models how well the algorithm has grasped the underlying connections and correlations in the data. The variance, or validation error, is the sensitivity of the algorithm to random noise in the data sample. A classifier (or regressor) that has a very high variance will fit the training data perfectly, failing to generalise when provided with a new data set. This phenomenon is known as *overtraining*. The classification error of an algorithm can be decomposed as follows [15]:

$$E = b^2 + V + \sigma_i,$$

where b is the bias, V is the variance and σ_i is the so-called irreducible error, which comes from intrinsic properties of the data and cannot be improved. Generally, the more a machine learning algorithm is trained, the lower its bias will become. After an initial decrease, the variance will then gradually increase, since the model is adapting to the data progressively better. This fact is known as "bias-variance dilemma" or "No Free Lunch theorem", and it implies that it is impossible to model a phenomenon with arbitrary accuracy without overfitting it.

The most common way to evaluate a binary classifier is through its Receiver Operating Characteristic (ROC) curve. There are four possible outcomes for a binary classifier[1]: it can correctly classify a signal event (true positive, TP), classify a background event as signal (false positive, FP), correctly classify a background event (true negative, TN) or classify a signal event as background (false negative, FN). Two quantities can be defined from these values: the signal efficiency, $\varepsilon_S = N_{TP}/(N_{TP} + N_{FN})$, which is the ratio of the correctly identified signal to the total signal present in the dataset, and the background rejection, $r_B = N_{TN}/(N_{TN} + N_{FP})$, which measures the ability of the algorithm to identify and discard background events. These values will vary according to the value of the user-defined cut on the output of the classifier. An ideal algorithm will have $r_B = 1$ for any ε_S. The ROC curve of a classifier is a plot of its background suppression versus signal efficiency for several cut values. The closer this plot is to the line $r_B = 1$, the better the classifier. To quantify the goodnes of a classifier, the area under the curve (AUC) is usually taken, with the assumption that good classifiers will have an AUC close to 1.

In the next sections I will discuss the two machine learning algorithms that I have optimised and used for my research: boosted decision trees (BDT) and artificial neural networks (ANN).

2.2 Decision trees

For each given classification or regression task one can choose from a plethora of different machine learning algorithms, each with its own unique pros and cons. With increasing complexity in the data, more complex models tend to outperform simpler ones because of their higher adaptability. One notable exception is decision trees, whose key characteristic is to iterate simple rules to create complex decision boundaries. Unlike other simple algorithms, decision trees can perform very well in comparison to, say, multi-layer perceptrons or other non-linear models. They can also be represented in an intuitive diagram. This is an important advantage over complicated algorithms, since it makes it straightforward to extract insight knowledge.

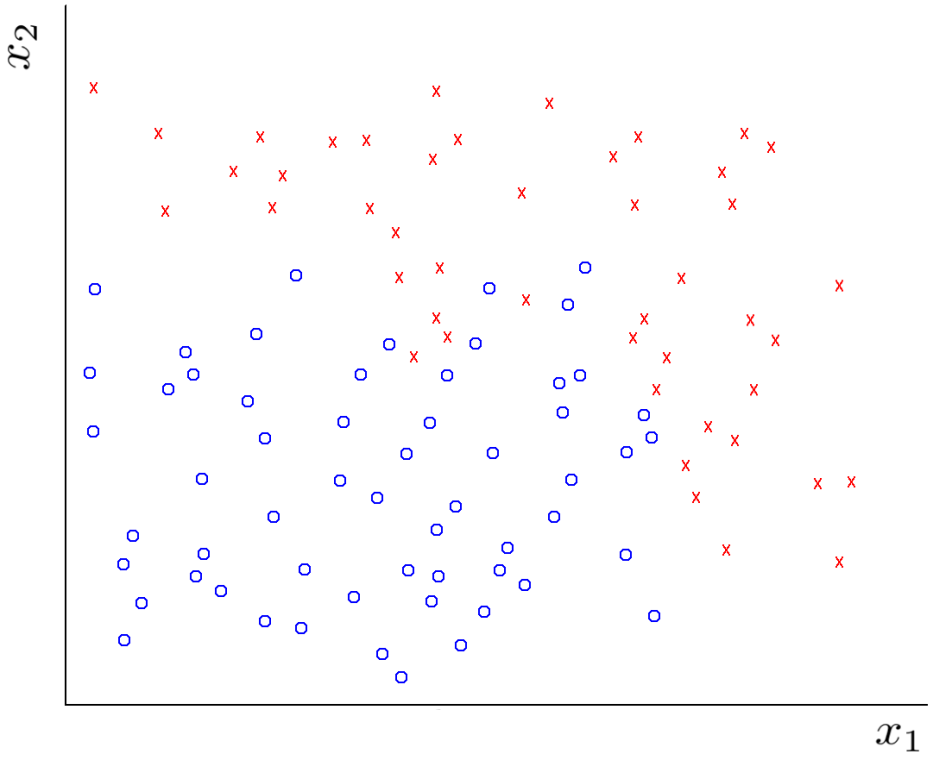


Figure 2.2: Scatter plot of two-class data in a two-dimensional feature space. `x` shaped points belong to class C_x and `o` shaped points belong to class C_o.

Working principle

A decision tree works by repeatedly operating cuts into the dataset, separating it into smaller subset that exhibit a better class separation than the original dataset. It is a supervised learning algorithm. At the end of the training process, the feature space is divided into smaller regions, and previously unseen data are classified according to the region they fall in. Since the problem discussed in this work involves two classes, I will only consider the case of a two-class

[1]We assume that the two classes are signal and background, as is the case in my analysis.

problem (in my case, signal and background); furthermore any k-class problem can be studied in terms of k two-class problems [15].

In a simple example with only two input features, x_1 and x_2, the data can be represented in a scatter plot as illustrated in Figure (2.2). The decision tree applies univariate cuts to the data. These cuts have the general expression

$$x_i - c_{ij} = 0,$$

where x_i denotes the $i-$th feature and c_{ij} the $j-$th cut on x_i. In our two dimensional example, the applied cuts are straight lines parallel to one of the axes, and at the first iteration the data set is split into two subset. In a simple decision tree with only one node, for example with a single cut on x_2, the situation might look like Figure (2.3).

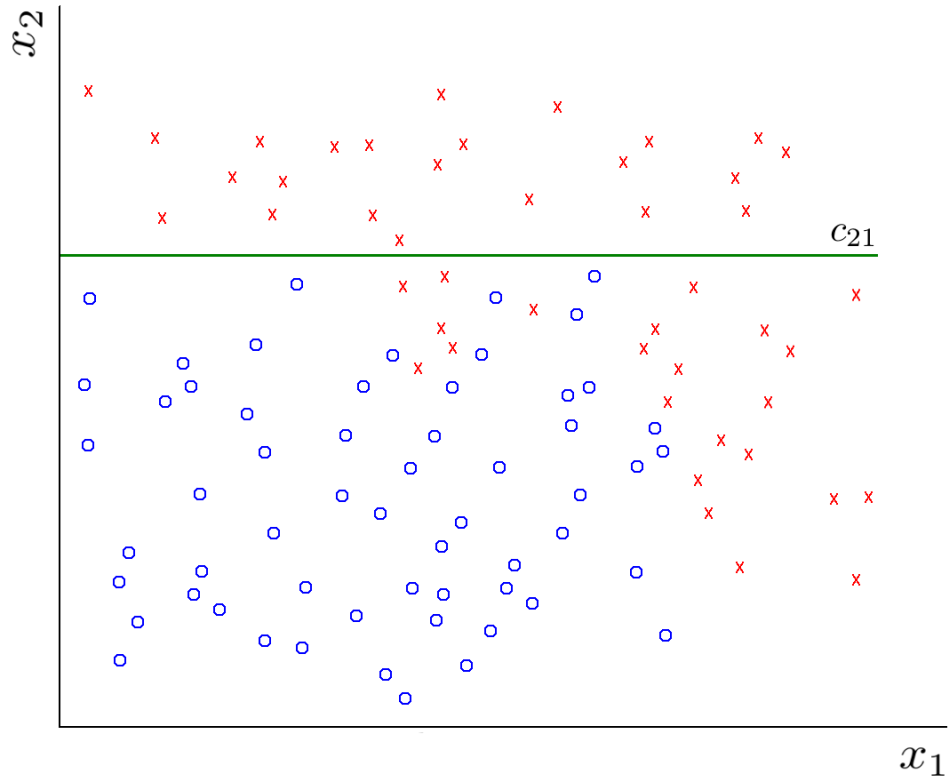


Figure 2.3: The first cut on the dataset. The region above the cut contains only points belonging to the class C_x; that region is therefore *pure*.

As the image shows, data points with x_2 larger than the cut value all belong to the same class. For this subset, therefore, no more cuts are needed. For the other subset, the one that falls below c_{21}, the data are still heavily mixed between the two classes. A further division with an additional cut on x_1, as illustrated in Figure (2.4), can significantly improve the separation. After this cut the resulting subsets are not as mixed as before, but they are not pure: the region with x_1 larger than c_{11} contains a few data points belonging to the *circle* class, C_o, and similarly the other region contains events belonging to the *cross* class, C_x. In order to reach optimal separation of the data one could continue cutting until each region only contains instances belonging to the same class. However, this is not always desirable because it can lead to overtraining the model.

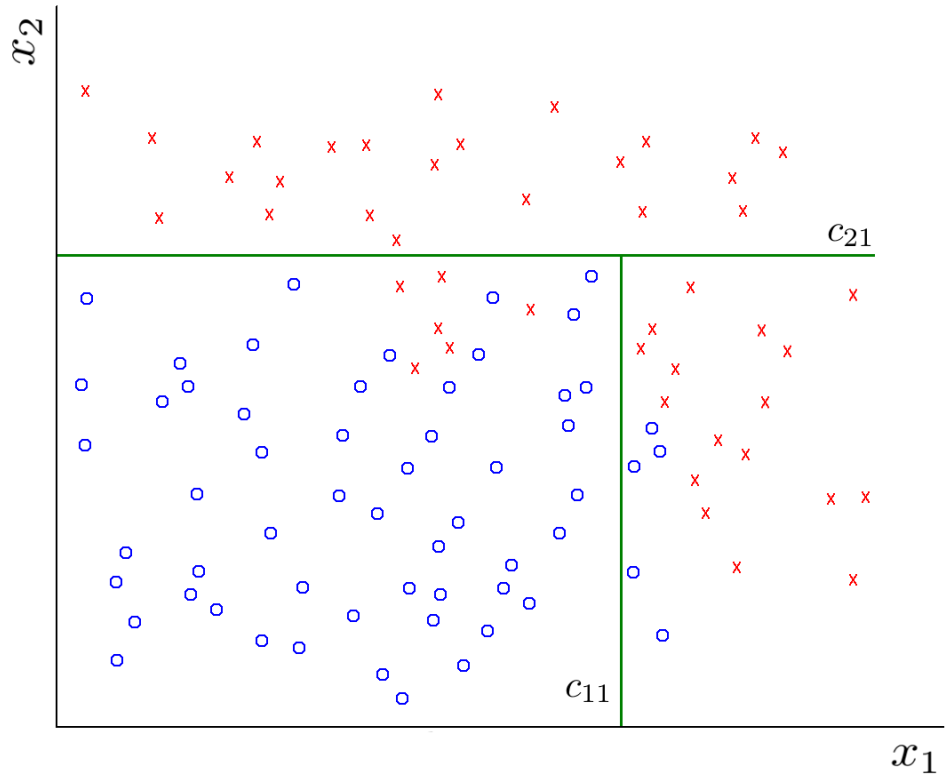


Figure 2.4: The second cut on the subset. Neither of the subsets obtained with this cut are pure, since they contain a mixture of data points belonging to both classes. In order to prevent overtrainig it is advisable to stop cutting at this stage. Each subregion in the feature space, then, is associated a probability to contain events from a particular class.

In essence, there are two fundamental questions that define the decision tree algorithm: 1) which variable to cut on and 2) what is the optimal value of the cut. To answer these questions one needs to quantify how well separated the classes are after each cut. A subset of the data contained in a region is considered pure if it only contains instances belonging to the same class. In the previous example, the region $x_2 > c_{21}$ is pure, whereas the other two regions are slightly impure. There are several indicators to measure the purity of the data in a given region of the feature space; the most common ones are the following.

- Entropy: $E(p) = -p \log_2 p - (1-p) \log_2 (1-p)$;
- Gini index: $G(p) = 2p(1-p)$;
- Misclassification error: $M(p) = 1 - \max(p, 1-p)$,

where p is the fraction of instances belonging to one of the two classes (note that these indicators are symmetric with respect to class interchange). Based on these indicators, the purity of the two regions obtained after splitting the dataset is calculated, and maximised by selecting both the variable and its cut value. The process is then iterated until each region in the feature space has a purity higher than a given threshold. At this point, the decision tree is finished and each new data point is then classified according to the region it falls in.

As stated above, decision trees are simple to train and to interpret. Knowledge extraction is straightforward, at least in principle, because a decision tree is equivalent to a collection

of IF - THEN logical sequences. The tree considered above is the collection of the following conditionals:

- **if** $x_2 > c_{21}$ **then** the instance belongs to C_x;
- **if** $x_2 < c_{21}$ **and** $x_1 > c_{11}$ **then** the instance belongs to C_x with a probability of 78%;
- **if** $x_2 < c_{21}$ **and** $x_1 < c_{11}$ **then** the instance belongs to C_o with a probability of 89%.

This interpretability allows knowledge extraction and better understanding of the data, as opposed to more complicated algorithms. In fact, rarely do neural networks or kernel machines create models that are as easy to understand as the ones created by decision trees.

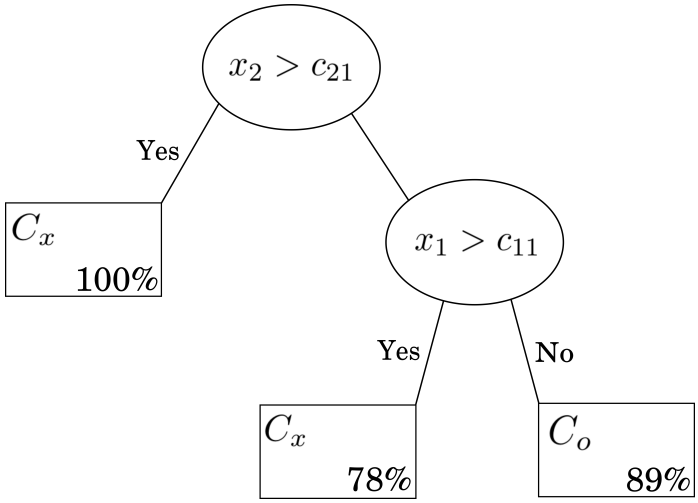


Figure 2.5: The decision tree for the simple example in the text. The probability for the predominant class in each leaf node is shown.

Decision trees can also be schematically represented in a diagram, with bubbles connected by lines. The bubbles are called nodes and represent subsets of the sample, whereas lines connect bubbles and show the outcome of different decisions. The decision tree of the previous example is shown in Figure (2.5). Each node contains a certain subset and the cut that has been chosen for that subset. After each cut, the subsample is split into two smaller sets, connected to the original one by lines. The first node contains the full data set, and the final nodes (so called *leaf* nodes) correspond to the regions the feature space is split into.

The graphical representation of decision trees further simplifies knowledge extraction, at least in the case of moderately simple trees. It resembles a reversed tree structure, and it, therefore, gave rise to the name "decision tree".

Combining multiple classifiers

Decision trees are generally vulnerable to fluctuations in the training sample [16]. For this reason they are often combined together in order to form a so-called *forest*, that is a collection of trees. This makes the whole classifier more robust to statistical fluctuations of the data. Multiple classifiers, called *base-learners*, are trained on different subset of the entire training sample. In application phase, events are classified according to the weighted average of the responses of the individual base-learners. Examples of methods in which multiple classifiers are combined include bagging and boosting, with the latter being a refinement and an evolution

of the former. These methods, boosting in particular, have been shown to reduce both training and validation error, making the algorithms more robust and refractary to overtraining [15].

Bagging

Bagging consists of repeatedly training a set of base-learners on different subsamples of the training sample. The training subset, s_i, is picked at random from the whole dataset S. The i-th base-learner, l_i, is then trained on this sample. If there is a total of N_l base-learners, the classification of a previously unseen data point $\mathbf{x}$ is the average of the responses of the single classifiers:

$$y(\mathbf{x}) = \frac{1}{N_l} \sum_{i=1}^{N_l} y_i(\mathbf{x}),$$

where $y_i(\mathbf{x})$ is the output of the i-th classifier. For decision trees,

$$y_i(\mathbf{x}) = \begin{cases} +1 \text{ for signal} \\ -1 \text{ for background} \end{cases}$$

The overall output will be close to $+1$ for signal events and to -1 for background events. This process of averaging the response of multiple classifiers is known as *voting*. Bagging can be quite useful when it is applied to algorithms that do not change their output drastically with a slight variation of the input sample; these algorithms are commonly referred to as *stable*. Unfortunately, decision trees and artificial neural networks are *unstable*. A slight difference in the dataset causes a large difference in the output. These algorithms thus have a high variance, and bagging is not optimal when used with them [15]. Furthermore, since the training subsamples are picked with replacement, some instances might be used several times for training while others might never be considered. This introduces a bias in the training that is not always negligible. For these reasons, in many classification and regression problems bagging is replaced with boosting, especially when coupled with decision trees. The most common boosting procedure is AdaBoost (short for adaptive boosting).

Boosting

The most common boosting technique is called ADAptive BOOSTing (AdaBoost). In AdaBoost each base-learner l_i is trained on a subset s_i containing the events that were misclassified by the previous algorithm l_{i-1}. At the beginning, all the events on the dataset S are assigned equal probabilities (weights) to be drawn for the subset. After running the sample s_1 through l_1, the misclassification rate, ε_1, and the boosting parameter for l_1, α_1, are calculated as

$$\varepsilon_1 = \frac{N(\text{misclassified})}{N(\text{total})} \qquad \rightarrow \qquad \alpha_1 = \frac{\varepsilon_1}{1-\varepsilon_1} < 1.$$

All the events that were correctly classified by l_1 have their weights multiplied by α_1. The probabilities are then normalised in order to sum to 1. This has the effect of reducing the drawing probability for correctly classified events and increasing it for both the misclassified and the previously undrawn ones. The process is then iterated until the misclassification rate gets smaller than a user-defined threshold, or a predefined number of base-learners is trained. In the application phase, each new data point is classified according to a weighted majority vote (weighted average) of the responses of each single base-learner. The weights assigned to

each base learner are proportional to their accuracy on the training sample. The response of the boosted classifier is therefore

$$y(\mathbf{x}) = \frac{1}{N_l} \sum_{i=1}^{N_l} \log\left(\frac{1}{\alpha_i}\right) y_i(\mathbf{x}).$$

AdaBoost favours simple classifiers that have a large bias. Very precise classifiers would, conversely, have a very low misclassification rate and each new subsample would mostly be made of few significant entries and other instances repeated many times over. In the case of decision trees, for example, the base-learners are so-called *decision stumps*, which are trees grown for one or two levels only.

Learning can be improved further by slowing the learning pace. This is usually done via a *learning rate* parameter, β, at the exponent of the boosting parameter: $\alpha \to \alpha^\beta$. This allows for a slower learning that further decreases the bias and the variance of the overall classification.

2.3 Artificial neural networks

Artificial neural networks play a very important role in machine learning. Despite not being as straightforward in their working principle as decision trees, they are among the most used algorithms since they are very customisable and can adapt to a variety of different tasks.

The models produced by neural networks are often very complex, and unlike decision trees they do not allow easy knowledge extraction. They are therefore usually considered as a black box algorithm, and for many practical purposes it is convenient to treat them as such. Neural networks are, however, based on a very simple processing unit, called linear perceptron. It is the combination of many of them that creates the complex neural networks required for advanced data processing tasks.

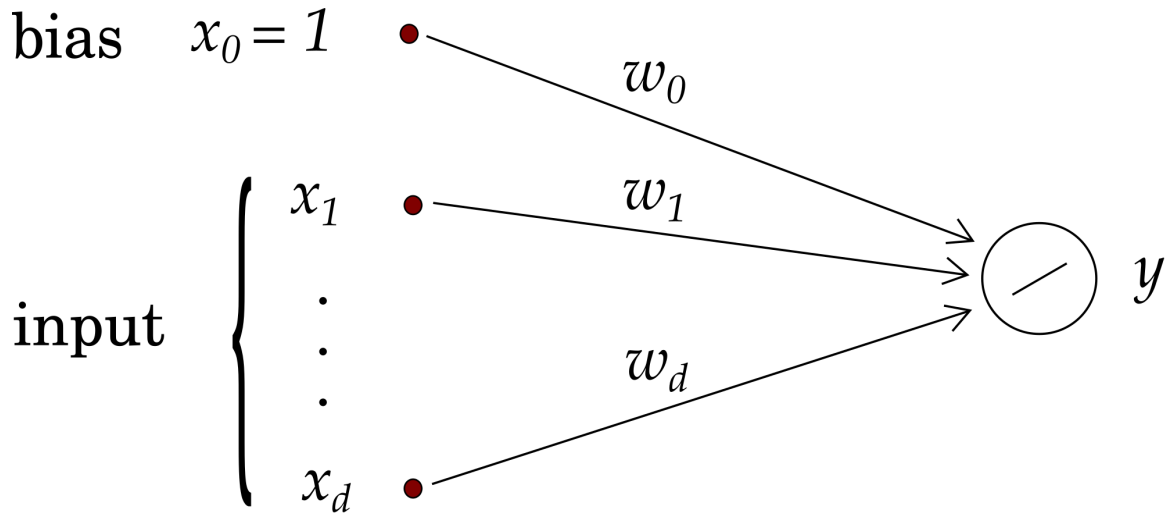


Figure 2.6: Schematic representation of a linear perceptron with d features, a bias node, x_0, and the output neuron, y. The coefficients in the hyperplane equation (2.1) are represented by connections from the input nodes to the output neutron; they are the weights, w_i.

Linear perceptron

Neural networks in their simplest form implement a linear discriminant. When data are linearly separable, a hyperplane is sufficient in order to split the classes: in this case, a *linear perceptron*

like the one shown in Figure (2.6) can be a good implementation for it.

The linear perceptron defines a function from the d-dimensional feature space to $\mathbb{R}$:

$$y = f(\mathbf{x}|\mathbf{w}) = \mathbf{w}^T\mathbf{x} = w_0 + \sum_{i=1}^{d} w_i x_i, \tag{2.1}$$

where $\mathbf{w}^T = (w_0, w_1, -, w_d)$ is the weight vector and $\mathbf{x}^T = (1, x_1, x_2, -, x_d)$ the generalised input vector which includes the bias. The bias node is set to 1 in order to have hyperplanes that do not necessarily intersect the origin.

In a neural network the function characterising a neuron is called *activation function*. The perceptron is called linear because, as it can be seen in equation (2.1), the activation function of the output node is linear. Once the weights and hence the hyperplane are determined, any new data point $\mathbf{x}_i$ is classified according to its position with respect to the hyperplane. In particular, with reference to Figure (2.7), for a two-dimensional example a binary check is performed:

$$\mathbf{x}_i \in \begin{cases} C_x & \text{if} \quad \mathbf{w}^T\mathbf{x}_i > 0 \\ C_o & \text{otherwise} \end{cases}$$

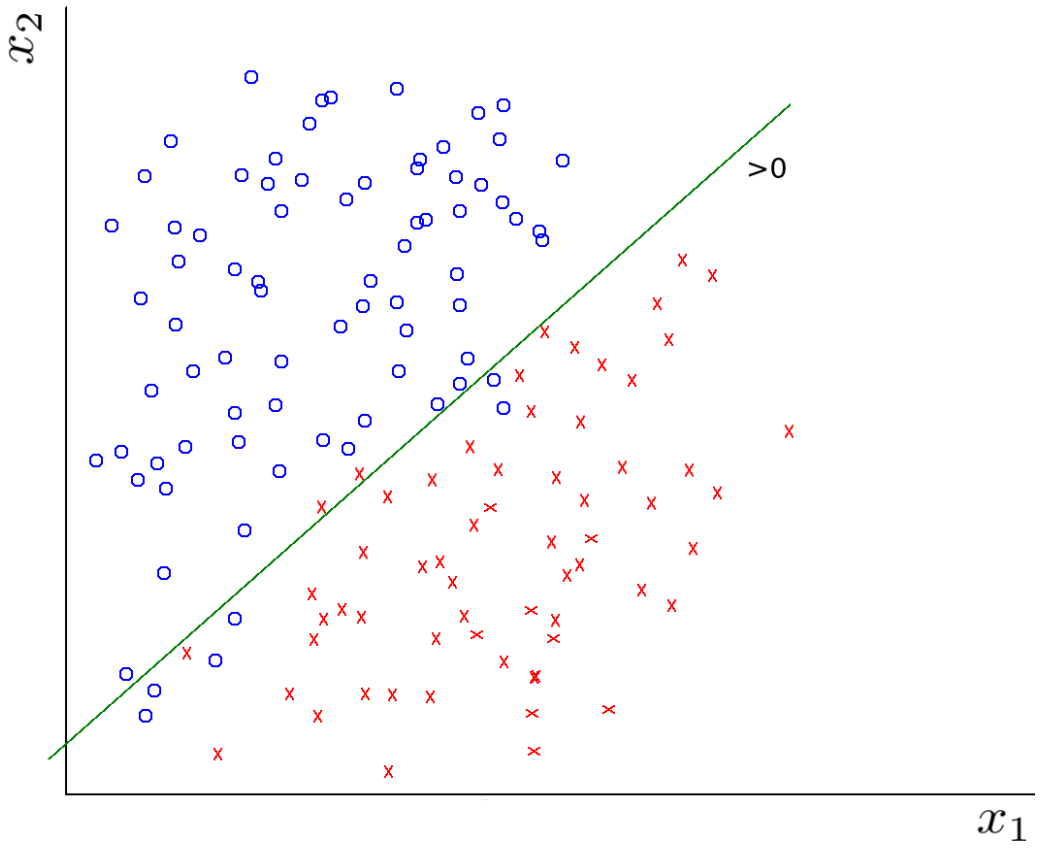


Figure 2.7: A linearly separable dataset with the separating hyperplane. Also shown the half plane where the function is greater than zero.

The weights in the perceptron can be obtained by training it in a variety of different methods. Backpropagation is the most common one and it will be explained in detail in the next sections.

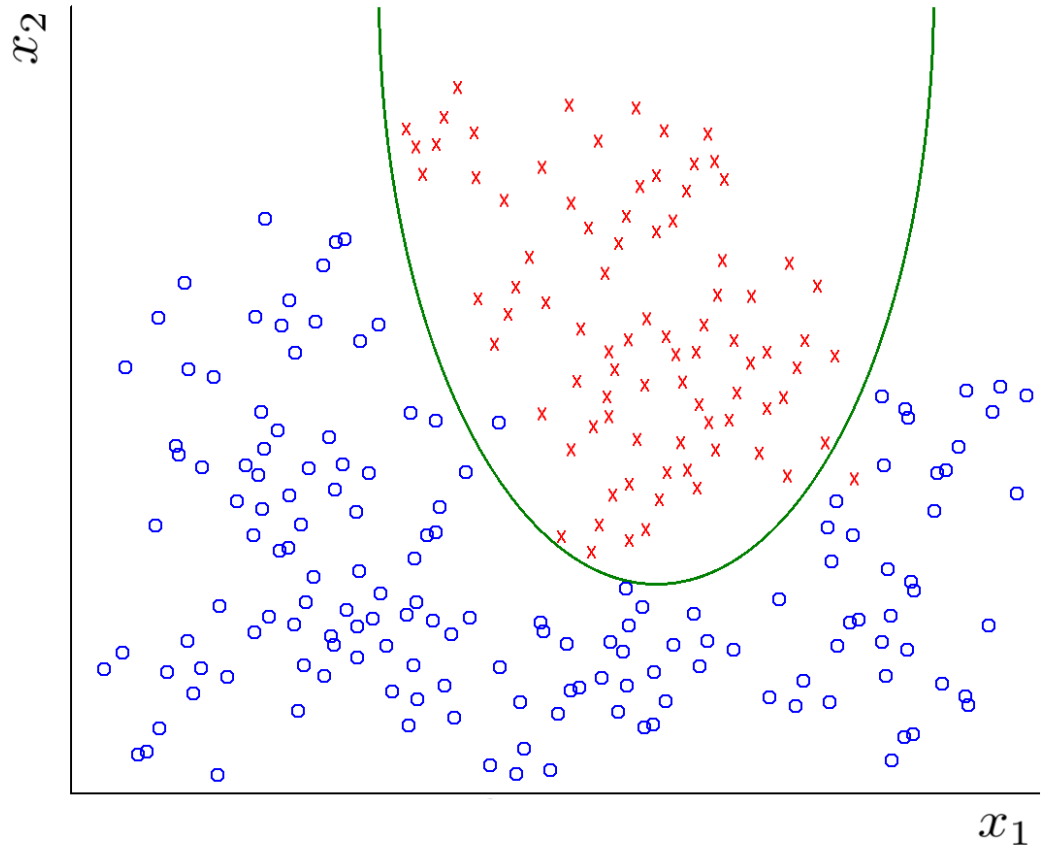


Figure 2.8: A two-dimensional example of non-linearly separable data.

Multilayer perceptron

Most classification problems are complex and an analysis such as the one presented above is not sufficient to discriminate between classes. The data in Figure (2.8), for example, are not linearly separable. A linear perceptron is then not sufficient anymore, and in these cases artificial neural networks evolve to *multilayer perceptrons*.

An extra layer is introduced between the input layer and the output neutron of the perceptron. This layer is called *hidden layer* and it is made up of several nodes, or neurons. Each of the input nodes sends information to each hidden layer node. In turn, each hidden layer node is connected to the output neuron. This way, data are analysed sequentially: the first step processes the input to the hidden layer, and the second step determines the output. Such an architecture is called *feed-forward* multilayer perceptron, and is shown in Figure (2.9). Each neuron in the hidden layer can be regarded as a single perceptron with a nonlinear activation function, α. Common activation functions are:

- the hyperbolic tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$;
- the sigmoid function $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$;
- the radial function $radial(x) = e^{-x^2/2}$.

On each neuron, the activation function acts on a specific combination of the neuron input: this combination is called *synapse function*, κ. The most common synapse functions are:

- linear sum: $\kappa(\mathbf{x}|\mathbf{w}) = w_0 + \sum_{i=1}^{d} w_i x_i$;

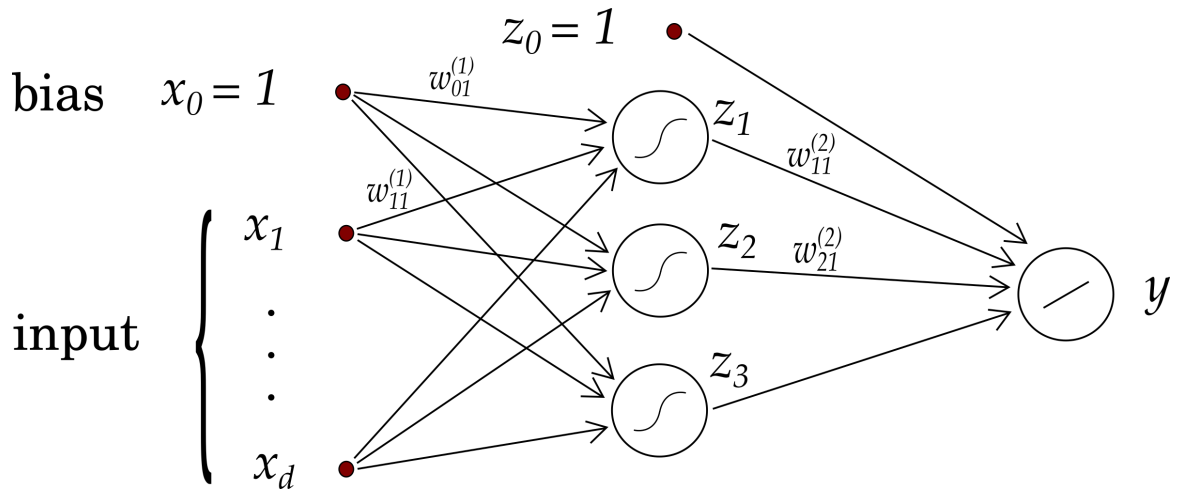


Figure 2.9: A multilayer perceptron with a hidden layer made up by 3 neurons and a bias node.

- sum of squares: $\kappa(\mathbf{x}|\mathbf{w}) = w_0 + \sum_{i=1}^{d}(w_i x_i)^2$;
- sum of absolutes: $\kappa(\mathbf{x}|\mathbf{w}) = w_0 + \sum_{i=1}^{d}|w_i x_i|$.

The output of each single neuron, then, is a composition of the synapse function and the activation function: $y = (\alpha \cdot \kappa)(\mathbf{x}|\mathbf{w}) = \alpha(\kappa(\mathbf{x}|\mathbf{w}))$.
The key characteristic of multilayer perceptrons is the nonlinear activation function in the hidden neurons. This nonlinearity allows the neural network to approximate any discriminating function for data with potentially any degree of complexity [15]. If the connection going from neuron i in layer k to neuron j in layer $k+1$ is characterised by the weight $w_{ij}^{(k)}$, the output y of the multilayer perceptron, as illustrated in Figure (2.9), can be expressed as follows:

$$y = w_{01}^{(2)} + \sum_{j=1}^{3} w_{j1}^{(2)} z_j = w_{01}^{(2)} + \sum_{j=1}^{3} w_{j1}^{(2)} \alpha \left(w_{01}^{(1)} + \sum_{i=1}^{d} w_{ij}^{(1)} x_i \right). \tag{2.2}$$

On assuming, for example, a tanh activation and linear sum synapse functions for the hidden neurons, the output is finally

$$y = w_{01}^{(2)} + \sum_{j=1}^{3} w_{j1}^{(2)} \tanh \left(w_{01}^{(1)} + \sum_{i=1}^{d} w_{ij}^{(1)} x_i \right). \tag{2.3}$$

This makes multilayer perceptrons universal function approximators, greatly exceeding the capability of a linear perceptron. Multilayer perceptrons can be seen as a composition of two separate stages. The first stage corresponds to a data preprocessing step, where the feature space is transformed into a new space which has the dimensions of the hidden layer. In this new space, the second stage is a simple linear perceptron that determines the output of the neural network. In this sense, a multilayer perceptron implements a linear discrimination on a space where each variable is a nonlinear combination of the input variables: the output is a linear combination of nonlinear functions, as can be see in equation (2.2).

Multilayer perceptrons, then, can be interpreted as an algorithm that first transforms the problem to a less complex one, and then implements a simple decision in the new feature space. Unfortunately, most of the time the transformation performed through the hidden layer is too complicated. It is, in fact, highly-nonlinear and can potentially map few input dimensions to hundreds of hidden variables. Understanding artificial neural networks as a black box, model-creating algorithm is therefore often more convenient and straightforward than visualising a

linear discrimination in an alien feature space. Sometimes it is also not possible to express analytically the new feature space in terms of the old one, vanifying the intention to visualise the network's inner machinery.

Whatever the final interpretation of the fine internal workings of the algorithm, multilayer perceptrons excel at classifying data after an extensive round of training performed on labelled data.

Training

Training a neural network, be it a linear or multilayer perceptron, amounts to finding the optimal weights that characterise the connections within the network. There are various techniques to do this; each one of them is performed on one or multiple batches of labelled data. The general aim is to adjust the weights in order to make the network's output as close to the data labels as possible. The most important training routine for neural network is *backpropagation*.

Backpropagation

In backpropagation [16], weights are updated in order to minimise the classification error. The mean square error is defined as

$$E(\mathbf{x}_1, \mathbf{x}_2, -, \mathbf{x}_N|\mathbf{w}) = \sum_{a=1}^{N} E_a(\mathbf{x}_a|\mathbf{w}) = \frac{1}{2}\sum_{a=1}^{N}(y_a - \hat{y}_a)^2, \tag{2.4}$$

where the sum is taken over the whole training batch; y_a and $\hat{y}_a$ are the network output for the event $\mathbf{x}_a$ and its label, respectively. Through y_a, the error depends on the network's weights.

Once the whole batch, which might be the whole dataset or only a sample of it, has passed through the network the weights are updated by minimising the error using *gradient descent* technique. The gradient of the error function in the weight space is calculated. Since it indicates the direction of steepest increase for E, the weights are then changed by a quantity defined as

$$\Delta\mathbf{w} = -\eta\ \nabla_{\mathbf{w}} E \quad \rightarrow \quad \Delta w_i = -\eta\ \frac{\partial E}{\partial w_i}.$$

This modifies each weight in the direction of the maximum decrease of the training error. The parameter $0 < \eta < 1$ is known as *learning rate*. It slows the learning process down, reducing large fluctuations in the weights. It is one of the most important hyperparameters in neural networks: if it is too large, the new value of the weights depends too much on recent training instances; if it is too small convergence might take too long or not happen at all [15]. The gradient of E is calculated through the chain rule and it is as though the error propagated back from the output to the input; hence the name backpropagation.

As an example the weights in the second layer are updated via

$$\Delta w_{j1}^{(2)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{j1}^{(2)}} = -\eta \sum_{a=1}^{N}(y_a - \hat{y}_a) z_{ja}, \tag{2.5}$$

where z_{ja} is the output of the $j-$th hidden neuron for event $\mathbf{x}_a$. One more step back, and the update for the weights in the first layer is

$$\Delta w_{ij}^{(1)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{ij}^{(1)}} = -\eta \sum_{a=1}^{N}(y_a - \hat{y}_a) w_{j1}^{(2)} z_{ja}(1 - z_{ja}) x_{ia}. \tag{2.6}$$

This way, the whole network has received a weight update based on the training result of the whole batch. The network is trained on every batch and the process is then repeated for a number of times, usually referred to as *epochs*, or until convergence. Convergence is reached when the training error over each batch is smaller than a predefined limit, or alternatively when the network does not obtain better results over multiple epochs. This algorithm is called *batch learning*. A special case of it, *online learning*, occurs wen each batch is made up of one event. The weights are updated after every event has passed through the network. This requires a smaller learning rate since there are many more updates. The equations governing online learning are the same as (2.4), (2.5) and (2.6) without the summation over the batch.

Deep neural networks

Neural networks with one hidden layer can approximate any separating function in the feature space, provided that the number of hidden units is large enough ([16], section 8.10). Since a network with d inputs and H hidden layers has a total of $(d+1) \cdot H$ weights, the computation time to train such a network can easily get very large and a trade off might be reached where the precision is sacrificed to reduce processing time.

An alternative way to structure the network is to have multiple hidden layers with fewer neurons. Such networks are known as *deep neural networks* and they tend to outperform single layer networks [17, 18]. It is clear that, with deep neural networks, the very structure of the network is open to tuning: two hidden layers with six neurons each will generally have a different performance than three hidden layers with four neurons each.

For the analysis discussed in this work, a deep neural network was used. A part of the tuning was devoted to the search of the best architecture that on one hand minimised training time while maximising performance.

2.4 TMVA

For my analysis, I used the software package TMVA [16], which stands for "Tool for MultiVariate Analysis". It is a plugin to the popular high-energy physics analysis software, ROOT [19], with which it integrates seamlessly. This makes it easy to work with ROOT's specific data structures like TTrees or .root files. This is handy since data analysis in BESIII is all based on ROOT. The scripts I used throughout my project are available at `https://github.com/amlaeb/HEP-ml`.

In TMVA, different machine learning algorithms (also referred to as multivariate methods) are built into the program and can be customised to some degree through the tuning of hyperparameters. A wide range of multivariate methods is available for the user to choose from. In particular, TMVA implements one boosted decision tree algorithm, **kBDT**, and four different artificial neural networks implementations. They are:

- **kCFMlpANN**, an adaptation from an old FORTRAN code;
- **kTMlpANN**, a ROOT native multilayer perceptron;
- **kMLP**, a new neural network implementation built specifically for TMVA;
- **kDNN**, an implementation designed to work on multi-core and GPU architectures.

Following guidelines on the TMVA user's guide, I decided to stick to the **kMLP** implementation since it seems to better fit my analysis. The first two algorithms are older and perform

worse, while the **kDNN** method is an overshoot for my purposes, since I work with relatively small batches of data.

TMVA, as any other machine learning framework, works in two distinct phases: the training phase and the application phase.

Training phase

For the training of classifiers, TMVA needs labelled data. This data are usually produced by using Monte Carlo (MC) generated data that simulate the physical situation one is interested in studying. The advantage of using MC data instead of, for example, data coming from experimental runs is the 100% accuracy of the label attached to each event.

The data are loaded at the beginning of the script through a TTree containing the signal events and another containing the background events. The variables of interests are chosen among all TBranches of the TTree and form the feature space for the training and the subsequent classification. An important task in any machine learning problem is the determination of the most important variables, namely the features that provide the best discrimination.

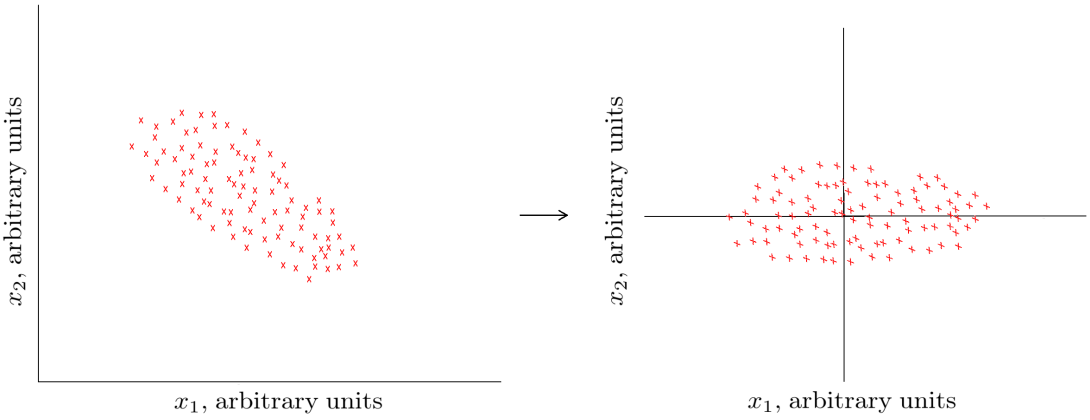


Figure 2.10: An illustration of a principal component analysis (PCA). On the left the unprocessed data sample exhibit linear correlation, which is transformed away by translating and rotating the data. The result is shown on the right. The covariance matrix is diagonalised, and the resulting eigenvectors form the axes of the new feature space, which is thus linearly decorrelated.

TMVA offers the option to perform preprocessing on the input variables. The main purpose of this is to remove correlations between any pair of features, since some classifier cannot take correlations of data into account, and to allow simpler discrimination. The preprocessing routines include a feature normalisation to eliminate differences in the relative range of the variables; a gaussianisation to transform the inputs into normal distributions; a simple variable decorrelation which removes the linear correlations between features and the more advanced principal component analysis (PCA). PCA decorrelates features by rotating them in the feature space in such a way that the direction along which the highest correlation is recorded lies along the first axis, the direction of the second highest variance lies along the second axis and so on (see Figure (2.10)). This is done by diagonalising the covariance matrix, and allows the user to optionally discard the transformed feature that have the lowest variance. In TMVA, however, only the decorrelation step is carried out within PCA: automatically discarding the low-variance features

would lead to a different analysis than the one desired by the user. The user can discard those features in a subsequent analysis.

TMVA allows the user to choose the preprocessing steps when booking the multivariate methods. This way, different methods can use data that has been preprocessed differently, fitting the particular algorithm. Once the multivariate methods have been chosen, their hyperparameters can be set through their booking strings. The dataset is split into a training sample and a testing sample of customisable size. In order to compare the performance of different algorithms, TMVA performs tests on each of them during and after the training. At the end of the training phase an evaluation of the methods is carried out. The features used by the algorithms are ranked according to their discrimination power and various plots are produced. The training results and testing statistics are saved in a .root file and the script terminates.

Application phase

Once the training phase is completed, and the algorithms and their hyperparameters are fine-tuned, one can proceed to the classification of previously unseen data. This is usually done with physical data from experiments that supposedly mirror the Monte Carlo data used for training.

A new script is required for the application phase. In a similar fashion to the training script, a TTree is loaded containing the unclassified data. A loop over all the events in the TTree is performed; each event is passed through every algorithm that had previously been trained, and based on the response of the classifiers and on user-defined cuts, it is classified as either background or signal.

For my analysis, I chose, for the application phase, another batch of Monte Carlo data, produced with the same routine. This data were previously unseen by the classifiers, but at the same time they gave me the possibility to evaluate the performance of the algorithms since each event in the dataset was labelled with certainty.

At the end, I run the multivariate analysis on a batch of experimental data taken with the BESIII detector. With this, the performance of the algorithms, tuned using a Monte Carlo model, was studied with actual beam data.

2.5 Speed of the algorithms

For some physical application, like real-time triggering or data pre-selection, the speed of classification is an important parameter. I my analysis I found that boosted decision trees are considerably slower, in classification phase, than neural networks. This is the opposite of what is found for the training part. In that case, a boosted decision trees algorithm is much faster to learn its optimal parameters than a multilayer perceptron.

The machine I used for my analysis is equipped with an Intel Xeon CPU running at 2.20 GHz. I ran the script on a single core each time. Classification speed was approximately 1.2 ms/event for the decision tree and 6 ns/event for the multilayer perceptron. Training on an entire batch of data (described in Chapter 3) took roughly 45 min to the boosted decision trees, and $7 - 8$ hrs to neural networks.

Chapter 3

Training phase

3.1 Training dataset

As mentioned in the previous chapter, in order to train the algorithms I have used Monte Carlo (MC) data. In particular I had access to two different data sets organised in .root files. These files were inclusive MC datasets, including both signal and background. One of the inclusive samples has been saved for testing purposes; the other has been used as data set for the training phase.

The MC data sets were generated according to cross sections and branching fractions as found in Particle Data Group's database. These data are therefore expected to be a quite accurate representation of the outcome of the experiment. After the generation of the whole data set containing all the possible processes arising from e^+e^- scattering or annihilation, a preselection has been made to isolated candidates for the $J/\psi \to \gamma p\bar{p}$ process. Events have been preselected with the following conditions:

- two charged tracks, one positive and one negative (identified as p and $\bar{p}$);
- at least one photon candidate;
- $\theta > 10$ degrees, where θ is the angle between photon and charged tracks. This is to avoid bremsstrahlung photons;
- $d < 10$ cm, where d is the distance between the interaction point and the vertex along the beam direction;
- $x < 1$ cm, where x is the distance between the interaction point and the vertex in the plane normal to the beam line.

Signal

Since the inclusive Monte Carlo samples contain both signal and background events, I extracted the signal events and I created a new file containing exclusively those events. The file from which I extracted the signal is the inclusive MC simulation of the 2009 data taking at BESIII.

The aim of my research has been to study the possibility of applying machine learning techniques to the process $J/\psi \to \gamma p\bar{p}$; the signal dataset was therefore made up entirely of these events. For each event, the invariant mass of the final-state particles is calculated. By making a histogram of this observable, one expects a peak around the center-of-mass energy of the e^+e^- beams, which is tuned to the mass of the J/ψ, $M_{J/\psi} = 3.097$ GeV. A histogram of the MC signal

data I used is shown in Figure (3.1). The peak is evident; its spread is due to simulated experimental resolution and the properties of the detector. The total number of events in the signal datasets was $N_s = 55739$. These events included direct $J/\psi \to \gamma p\bar{p}$ decays as well as processes involving intermediate resonances, such as $J/\psi \to \gamma\eta_c$ with $\eta_c \to p\bar{p}$, and $J/\psi \to \gamma X(1835)$ with $X(1835) \to p\bar{p}$ (see Table 3.1).

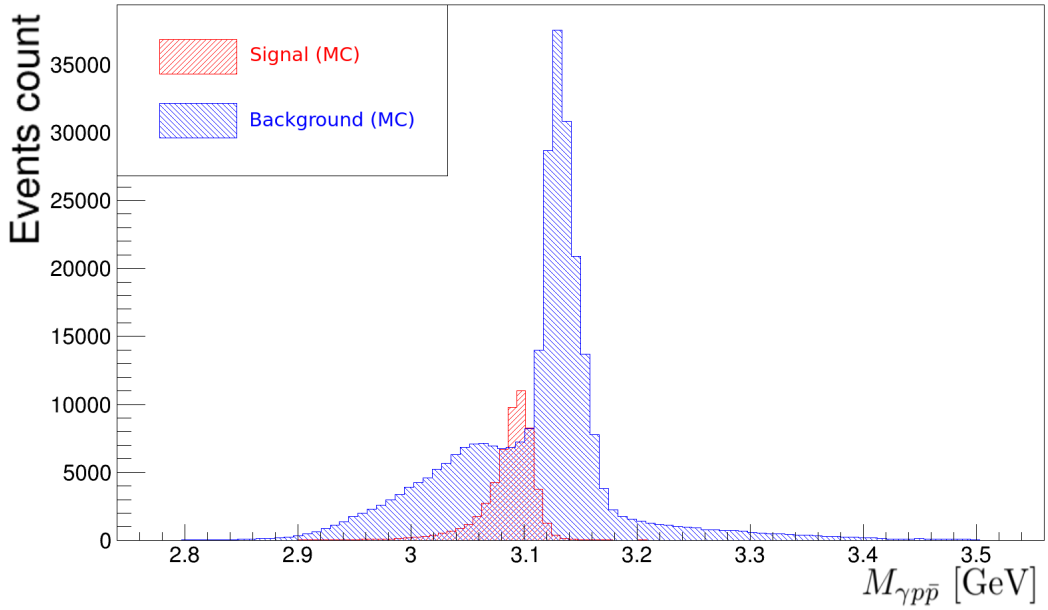


Figure 3.1: The signal and background sample histograms superimposed. Monte Carlo labels are shown. The signal (red) has a clear peak at $M_{J/\psi} = 3.097$ GeV, as expected. The background has a main peak around 3.14 GeV due to $J/\psi \to p\bar{p}$ and a secondary peak at 3.05 GeV due to $J/\psi \to \pi^0 p\bar{p}$.

Background

All the events that were left in the inclusive sample after storing the signal were considered background: $N_b = 294049$ background events were present in the dataset. This background includes all events that are misclassified as $J/\psi \to \gamma p\bar{p}$.

As can be seen in Figure (3.1) the background sample exhibits two distinct peaks. The first peak, around 3.05 GeV, is mainly due to the decay $J/\psi \to \pi^0 p\bar{p}$; the π^0 subsequently decays into two photons; one of them is detected, the other one flies away unnoticed. Therefore the reconstructed invariant mass is lower than the J/ψ mass.

The second peak, around 3.14 GeV, is mainly due to the decay $J/\psi \to p\bar{p}$. The branching fraction for this decay is roughly five times larger than the branching fraction for $J/\psi \to \gamma p\bar{p}$. This justifies the large number of background events in the second peak compared to the signal peak. An extra photon, coming from a different source, is detected: this causes a shift of the peak towards higher invariant mass values.

A summary of the signal and background events, including their branching fractions, is given in Table (3.1).

Event: $J/\psi \to$	Branching fraction	Event type
$\gamma p\bar{p}$	$(3.8 \pm 1.0) \cdot 10^{-4}$	Signal
$\gamma\eta_c \to \gamma p\bar{p}$	$\approx 2.6 \cdot 10^{-5}$	Signal
$\gamma X(1835) \to \gamma p\bar{p}$	$(7.7 \pm^{1.5}_{0.9}) \cdot 10^{-5}$	Signal
$p\bar{p}$	$(2.121 \pm 0.029) \cdot 10^{-3}$	Background
$\pi_0 p\bar{p}$	$(1.19 \pm 0.08) \cdot 10^{-3}$	Background
Any other	N.A.	Background

Table 3.1: Summary table of the events considered in the analysis. Branching fractions are taken from PDG [20].

3.2 Features

One of the most important aspects of machine learning is the role that input variables play in the goodness of a classifier. Some of them have no discrimination power at all, some others can be learned autonomously by the algorithms themselves, other features still play a role only for some MVA methods. It is common practice, for example, to prune off low-discriminating variables when the feature space is used as input for artificial neural networks, whereas boosted decision trees are completely immune to the presence of irrelevant features [16]. This is because boosted decision trees always make cuts on the most discriminating feature for a particular event, ignoring the others, while perceptrons propagate each feature forward multiplied by the relative weight; thus a completely irrelevant feature brings forth unnecessary calculations.

A first general distinction that can be made among physical features is between low-level features and high-level features. Low-level features, like the 4-momentum of a particle, are reconstructed from the raw information coming from one detector. High-level features, on the other hand, are obtained through calculations; an example of this is the invariant mass of a certain system of particles.

Baldi et al. [21] have shown that, generally, boosted decision trees and deep neural networks perform best when they are provided only with low-level features such as the final 4-momenta in a decay chain. Including additional high-level features such as the masses of intermediate resonances reduced the performance, which was further brought down by training the algorithms solely on those high-level features. This is due to the fact that multivariate algorithms can learn high-level variables when they are provided with the low-level ones.

Stimulated by this result, a large part of my analysis was devoted to figuring out the best set of input variables for both boosted decision trees and artificial neural networks. In order to do this, I prepared three different feature spacs: each of them contains all the events described in the previous section, but the algorithms will be trained on different features.

Feature space S_1: four-momenta

The first feature space, S_1, is the most basic one, as it contains only low-level features. It is a 12-dimensional space, since each of the three outgoing particles contributes with the four components of its four-momentum. These features are shown in Figure (3.2).

Even though neural networks might perform better or faster when receiving a low-dimensional feature space, I decided to keep the four-momenta space as the minimum. Therefore, I did not try to further reduce the dimensionality even if it might have made the network faster;

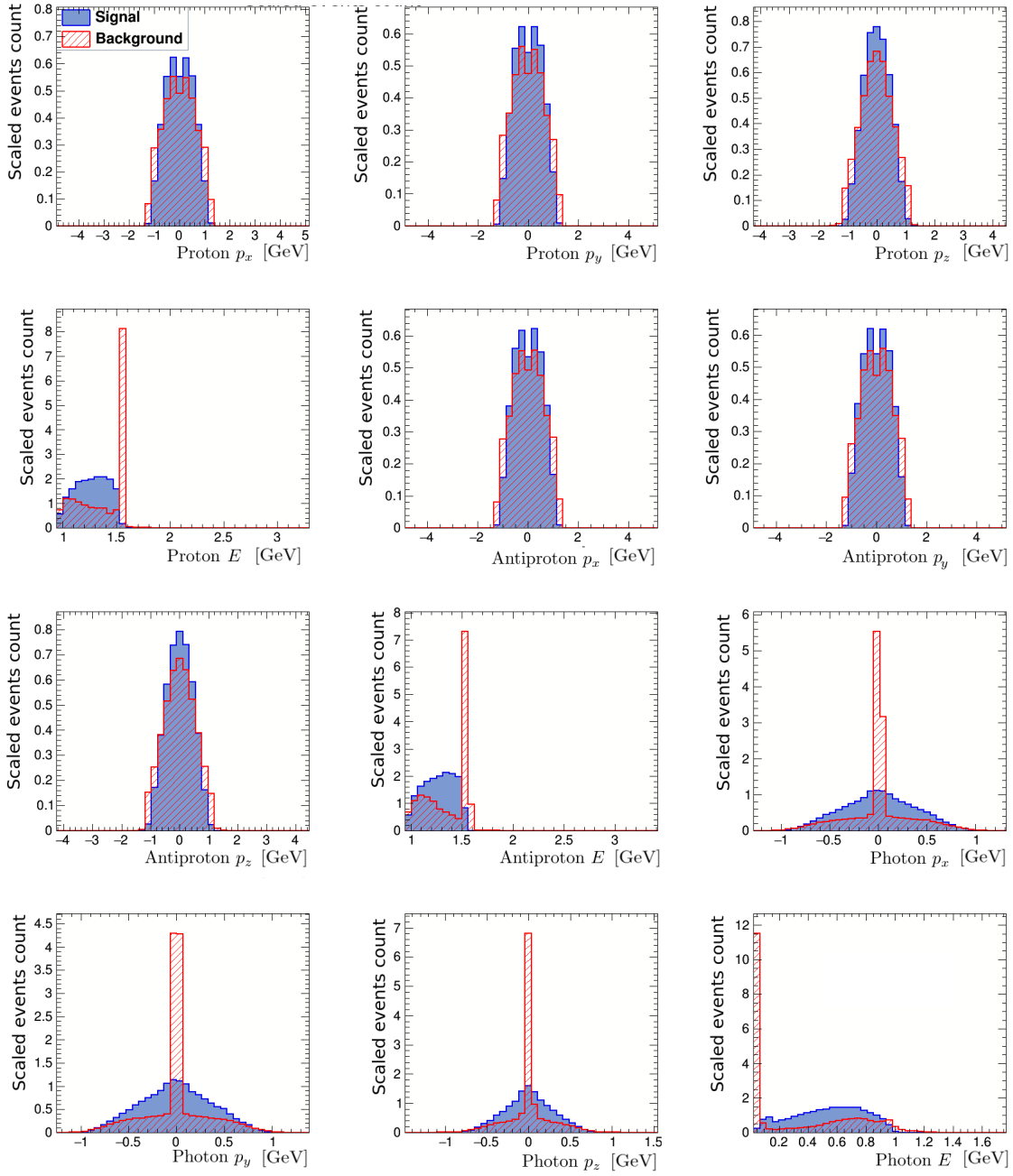


Figure 3.2: The twelve low-level features used in as input variables for features spaces S_1, S_2, S_3. The $y-$axes of the histograms are rescaled in order to show the properties of the features; therefore the height of the bins does not reflect the actual number of events contained in the interval. The aim is to show the different shapes for each variable in the signal and background channels.

the other two feature spaces add to this basis of twelve features. With only some of the components of the outgoing four-momenta, in fact, it would be impossible for the network to learn a discrimination on the invariant mass $M_{\gamma p\bar{p}}$. It might still, of course, discriminate sufficiently well with the remaining features and classify proficiently, but it would be harder to compare the model obtained with explicit $M_{\gamma p\bar{p}}$ cuts.

Feature space S_2: invariant mass

The second feature space, S_2, contains thirteen features: the twelve components of the four-momenta and their square and the invariant mass of the final-state particles, $M_{\gamma p\bar{p}}$, as shown in Figure (3.1). This feature space is meant to be a direct comparison with space one. As mentioned above, in fact, I expect algorithms trained on solely the low-level features to eventually learn their correlations, and in particularly operate implicit cuts on $M_{\gamma p\bar{p}}$. By comparing the results of the algorithms trained on space one and on space two I was able to evaluate this implicit learning.

Feature space S_3: χ^2 of the four-constraints fit

The third feature space, S_3, contains the basic four-momenta components and the χ^2 of the four-constraints (4C) kinematic fit [1]. This fit is a standard procedure to increase the resolution of a measurement. Furthermore, cuts can also be performed on χ^2 to discriminate between signal and background. The 4C fit consists in readjusting the kinematic variables, the four-momentum components, so that they fit the expected topology of the decay. In this case, assuming the J/ψ to be produced at rest, the four-constraints are

$$\begin{cases} (p_p + p_{\bar{p}} + p_\gamma)_x = 0 \\ (p_p + p_{\bar{p}} + p_\gamma)_y = 0 \\ (p_p + p_{\bar{p}} + p_\gamma)_z = 0 \\ E_p + E_{\bar{p}} + E_\gamma = M_{J/\psi}. \end{cases}$$

After the fit, a chi-squared analysis is performed, measuring the discrepancy between the values of a variable before and after the fit. Events that have a very low χ^2 value are, then, likely to be signal events. A histogram of this variable is shown in Figure (3.3).

3.3 Training of boosted decision trees

In this section I will describe the most important hyperparameters for the training of boosted decision trees. These hyperparameters have a crucial influence on the training of the classifier, both for the speed of training and classification and for its accuracy and generality.

In general, decision trees are fairly quick to train, with an average training time of $t_{BDT} = 45$ min. I therefore never considered speed of training as a discriminating factor when deciding which values of the hyperparameters were the best performing. I took the Receiver Operating Characteristic (ROC) curve of each algorithm as an indicator of its sheer performance. I always chose for a hyperparameter the value that maximised the area under the ROC curve (AUC).

For boosted decision trees, in order to find out experimentally the best values for the hyperparameters, I chose the following strategy: I started out with the set of default parameters, and

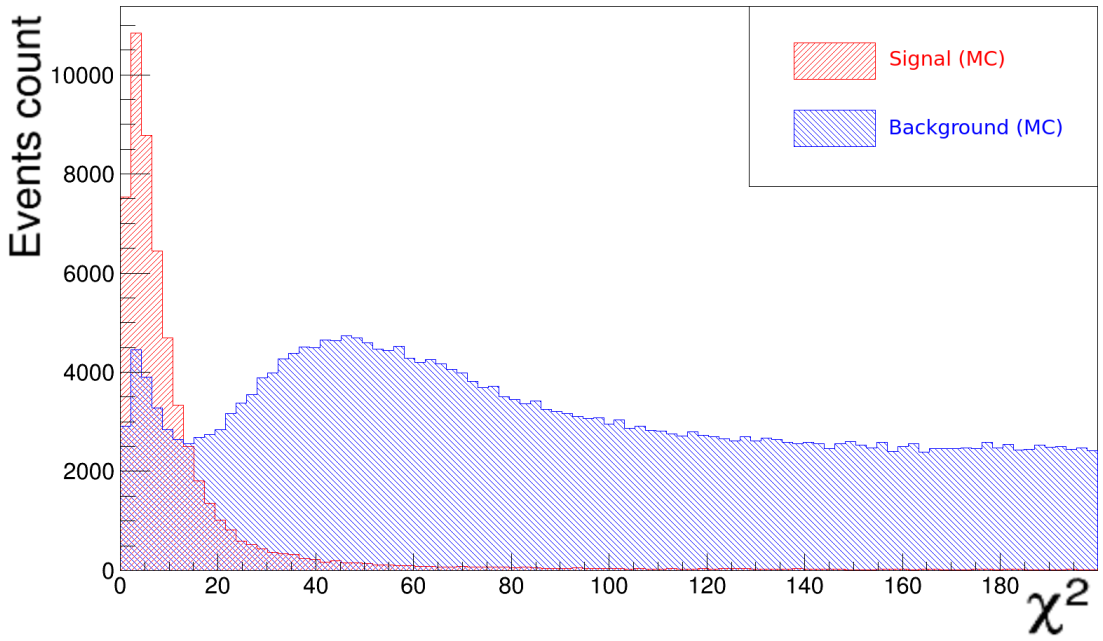


Figure 3.3: Histogram for the χ^2 of the four-constraints fit. Signal and background are determined by the Monte Carlo labels. The peak for low values tells how closely the signal variables resemble the desired topology of a signal event.

changed only the first one. After obtaining the best value for that parameter I moved on to changing the second one, keeping the first one fixed at the value that I found in the previous run. Iterating this process provided me with a good combination of hyperparameter values, although it might have ruled out some particular combinations that could work better. These combinations would be, in any case, very specific and hard to find; and as it will be seen in the following, this strategy turned out to give very good results.

Number of trees

This is one of the most important parameters: it is the number of single decision trees that form the boosted *forest*. As described in the previous sections, each one of these trees is grown only a couple of levels, and in classification phase a weighted average of all the individual responses is taken. Having too few trees in a forest could therefore induce unwanted statistical oscillations in the response; on the other hand, having too many trees can, in extreme cases, bring to a smoothing of the average response, making it harder to discriminate signal and background. It is controlled in TMVA through the parameter `NTrees`. Table (3.2) shows the AUC for different values of the parameter `NTrees`.

Tree growth

This parameter quantifies the maximum number of layers that each single tree is allowed to grow before being stopped. As mentioned in previous sections, boosting is especially effective when it combines rather weak base-learners. Preventing the trees from growing more than 2-3 levels is a good way to make sure that boosting works as expected [16]. It is implemented through the parameter `MaxDepth`.

NTrees	10^3 AUC	NTrees	10^3 AUC
100	783	6000	807
300	787	6500	807
500	790	7000	808
700	792	7500	808
900	794	8000	808
1000	795	8500	809
1100	795	9000	809
1300	796	9500	810
1500	797	10000	810
1700	798	10500	810

Table 3.2: The variation of the AUC with the parameter `NTrees`. This data were taken when the algorithm was still suboptimal, therefore the it is not the absolute value of the AUC which is relevant but its variation for different values of the parameter.

Cuts density

This parameter describes the density of the grid used to scan to find the best cut on the splitting parameter. It is controlled through the TMVA parameter `nCuts`. By setting the value to `-1` it is also possible to scan through all possible number of cuts without having a predefined step size.

Learning rate

In boosting, the learning rate is given by the exponent of the AdaBoost weight (see Chapter 2). It plays the role of a learning rate in that if it is too small there might be no satisfactory learning, while if it is too large recent training events will be much more important in determining the direction of training than previous data points. In TMVA it can be changed through the parameter `AdaBoostBeta`.

3.4 Training of neural networks

Next I discuss the training process of the neural network, describing the procedure and the most important training hyperparameters. Unlike decision trees, where parameters are the only important degree of freedom in the training phase, for neural networks a crucial aspect to consider is its architecture. The performance of a multilayer perceptron can change drastically when a hidden layer is added or when the number of units in some layers is changed [17]. The possible configurations are clearly too many to be tested in a reasonable amount of time, even considering only the most straightforward ones.

Furthermore, in the case of neural networks training time plays an important role. The more the hidden layers the higher the time needed to compute the response for a single event; even adding one single neuron to one single layer brings forth a considerable amount of mathematical operations, in particular matrix multiplications, performed at each iteration. Each neural network took several hours to train. In general, the learning times for artificial neural networks were an order of magnitude higher than the training times for boosted decision trees.

Training a neural network is therefore a more delicate operation than training a decision tree, and the hyperparameters, along with the architecture, must be chosen to reach a trade-off

between performance and computational time.

Number of epochs

This is the number of training cycles for each batch, controlled in TMVA by setting the parameter `NCycles`. In the case of online learning, described in Chapter 2, the response of the network is calculated for each event and backpropagation is performed `NCycles` times. It is an important parameter because on one hand it determines the convergence of the training, and on the other hand it can lead to overtraining if too large.
An alternative approach to check convergence is through a convergence test, which after each training cycle calculates the error and continues training until the error is lower than a predefined value. In my test I found this approach (implemented in TMVA through the `ConvergenceTest` option) to be considerably slower while not improving performance, so I kept it only as a backup.

Architecture of the network

The most crucial section in the training of multilayer perceptrons. The first important factor when it comes to network architecture is the feature space: when the network is required to learn high-level features many hidden layers are required [8]. When, on the other hand, these features are provided as input variables, a more shallow network might be enough for the classification task.

For this reason I found the best architectures to be different across the feature spaces set that I analysed: space S_1 required a much deeper network than spaces S_2 and S_3.

Furthermore, the problem cannot be simply tackled with a brute force approach. In fact, a neural network with too many hidden layers will easily adapt too closely to the training data, losing generality and thus giving rise to overtraining.

A neural network is designed in TMVA by tuning the parameter `HiddenLayers`. The code employed in TMVA associates a multilayer perceptron with 10 neurons in the first layer, $N+5$ in the second (N being the number of features) and 5 in the third one, to the string `10,N+5,5`. I will also use this handy labelling convention in the following. Table (3.3) shows the variability of the the performance of neural networks when their architecture is changed.

`HiddenLayers`	10^3 AUC	`HiddenLayers`	10^3 AUC
`N-5,N,N-5`	813	`N-6,N+2`	857
`N+10`	819	`N-5,N-5`	858
`N-5,N-5,N-5`	830	`N-3,N-3`	865
`N-3`	837	`N-5,N`	866
`N-3, N+3`	847	`N,N+2`	868
`N`	851	`N,N,N,N,N,N`	834
`N+3`	856	`N,N,N,N,N,N,N`	936

Table 3.3: Various architectures for neural networks in the feature space S_1. It can be seen that shallow networks are always outperformed by the deepest one.

Learning rate

Implemented through the parameter `LearningRate`, it controls the pace of learning. Setting it too large leads recent training instances to be overly important, while a low learning rate could delay convergence.

3.5 Best training parameters

Listed in Tables (3.4) and (3.5) are the values of the hyperparameters that give the best performance during the training runs. All the hyperparameters are the same in all three different feature spaces except for the architecture of the neural networks. This shows how drastically the performance of multilayer perceptrons varies when the feature space changes: when only low-level features were considered, as in space S_1, a deeper network was required to have performance that was comparable to the more shallow networks used in the other feature spaces. As shown below, a Principal Component Analysis improved BDT performance; in addition to that, both for BDT and ANN input variables were normalised to the interval [-1,1] (or [0,1] for intrinsically positive variables like the energy).

BDT parameter	Value
`NTrees`	5000
`MaxDepth`	3
`nCuts`	40
`AdaBoostBeta`	0.3
`VarTransform`[1]	PCA

Table 3.4: The best hyperparameters for boosted decision trees. No hyperparameter changes were necessary across different feature spaces for BDT.

ANN parameter	Value in S_1	Value in S_2	Value in S_3
`HiddenLayers`	N,N,N,N,N,N,N	2N,2N	2N,3N
`NCycles`	400	400	400
`LearningRate`	0.04	0.04	0.04
`NeuronType`[2]	`radial`	`radial`	`radial`
`NeuronInputType`	`sum`	`sum`	`sum`

Table 3.5: The best architectures and hyperparameters for neural networks. While the hyperparameter do not vary across feature spaces, the architecture does vary significantly in order to adapt to the different input variables and their correlation. In particular, a deep network with 7 hidden layers trained on low-level features performs better than a more shallow network trained on the same features plus the invariant mass.

3.6 Comparison of the algorithms

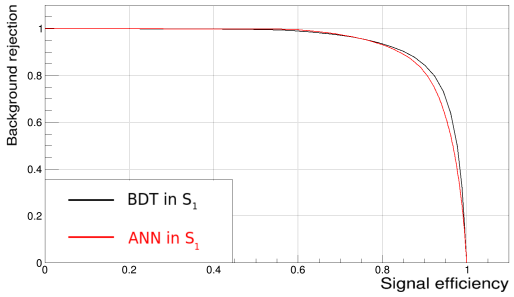
A first comparison of the MVA methods can be made after the extensive training runs. In particular, this first test served as a discrimination procedure to find out how different algorithms

[1] This option implements the preprocessing steps mentioned in the previous chapter.

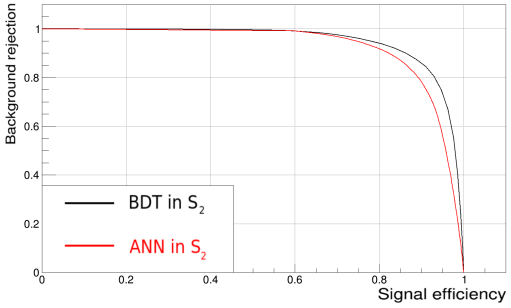
[2] This option and the following implement the neuron activation and synapse functions respectively, which were discussed in Chapter 2.

perform within the same feature space and also across different feature spaces. This first analysis is based on the ROC curves produced during the validation phase of the algorithms, and it is supposed to be only a rough preliminary estimation of the performance of the classifiers. In this phase, the algorithm are tested through the classification of a small validation sample that had been previously set aside and not used for training. The sample I used contained $N_B \approx 44000$ background events and $N_S \approx 5700$ signal events. The outcomes of these tests are shown in Figure (3.4) and summed up in Figure (3.5).

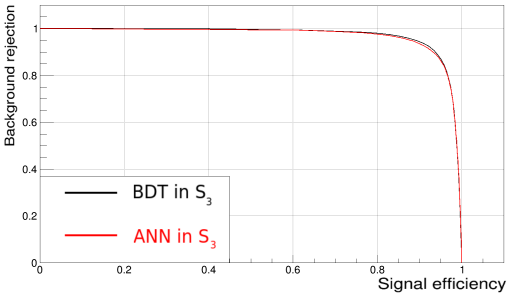
In the feature spaces S_1 and S_3 both BDT and ANN, as shown in Figures (3.4a) and (3.4c), obtained the same performance, while Figure (3.4b) shows a slight advantage of decision trees on the validation set for BDT on feature space S_2. Overall, all the classifiers perform very consistently when optimised for each particular feature space (as in the case of neural network architecture). The differences between BDT and ANN within the same feature space (in particular in S_1 and S_3) are very slight and do not indicate a reason to prefer decision tree over neural nets. The area under the ROC curve (AUC), taken as the best parameter to evaluate training, is shown for all the classifiers in all different feature spaces in Table (3.6)



(a) Feature space S_1



(b) Feature space S_2



(c) Feature space S_3

Figure 3.4: (a), (b), (c): the ROC curves of the classifiers trained in feature spaces, respectively, S_1, S_2, S_3. Classifiers trained on S_1 are shown in red; those trained on S_2 are shown in blue and finally, algorithms trained on S_3 are plotted in black. The latter MVA methods show better validation results.

The preliminary analysis has shown that classifiers trained on feature space S_3 give the highest performance for both BDT and ANN. This seems to indicate that the χ^2 of the $4C$ fit, when included in the feature space, improves discrimination power of classifiers, as opposed to the invariant mass $M_{\gamma p\bar{p}}$. This is probably due to the fact that, while $M_{\gamma p\bar{p}}$ is a simple function of the four-momenta and can be easily learned by the algorithms, χ^2 is more complex and it is difficult for MVA methods to retrieve it when it is not provided as an input. Generally, however, all algorithms showed a robust performance, with AUC values above 90%. This is likely due to the good separation noticeable in the input features (see Figure (3.2)).

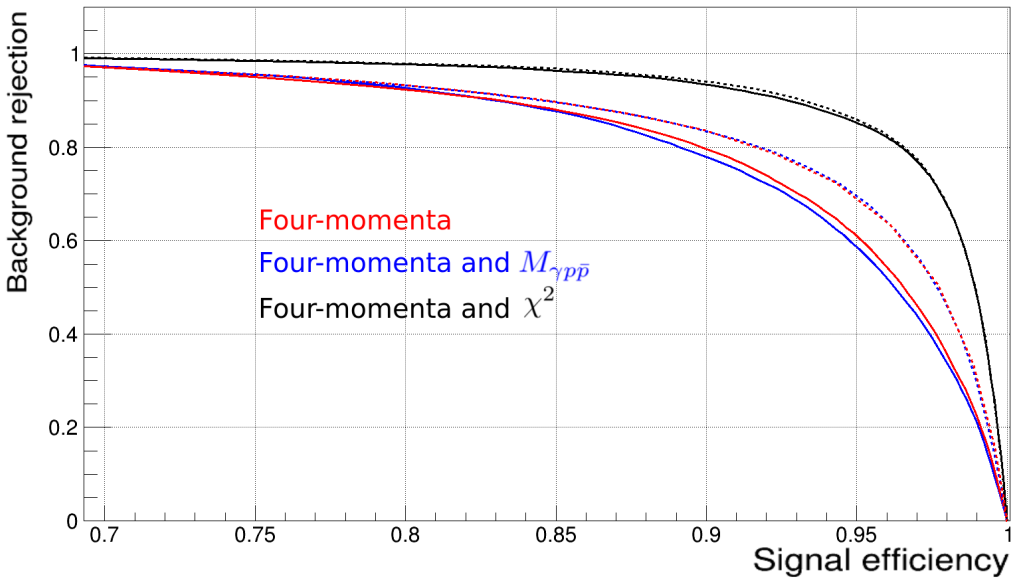


Figure 3.5: Comparison of all the classifiers. Solid lines indicate neural networks, dashed lines represent boosted decision trees. Classifiers trained on S_1 are shown in red; those trained on S_2 are shown in blue and finally, algorithms trained on S_3 are plotted in black. The latter MVA methods show better validation results.

Feature space	BDT	ANN
S_1	0.942	0.936
S_2	0.947	0.926
S_3	0.970	0.968

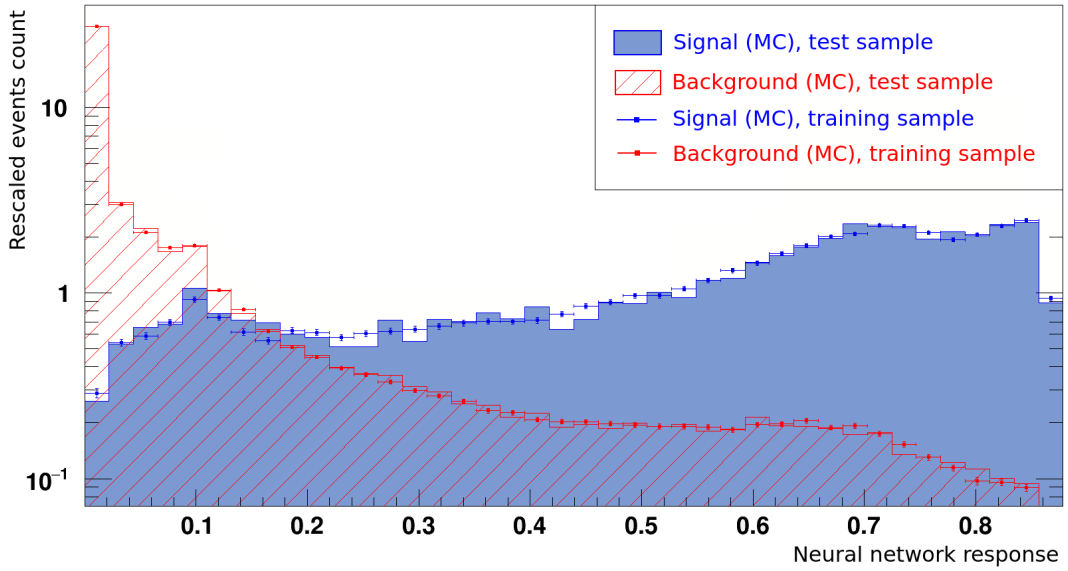
Table 3.6: The value for the area under the curve (AUC) parameter. It varies slightly across feature spaces; this seems to indicate a better performance for classifiers trained with high-level features in addition to the low-level ones.

In the next chapter I will discuss whether this difference in performance also shows on a more extensive data sample, and how multivariate methods compare to a standard analysis approach.

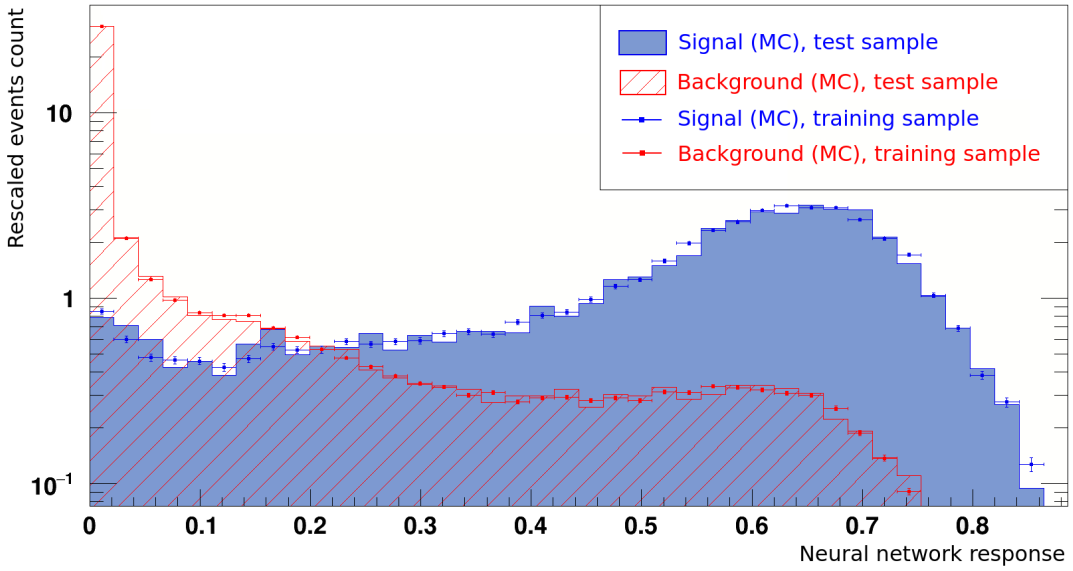
3.7 Overtraining check

In order to avoid overtraining it is good practice to compare the response of classifiers during training to the response during validation. If overtraining occurrs, the algorithms will fit the training data too closely, and the responses will differ a lot. The following figures show, at least qualitatively, that the neural networks I used were not overtrained. Both the histogram obtained from the training sample and the one obtained from the validation (testing) sample are very similar in shape. Boosted decision trees showed similar immunity to overtrainin.

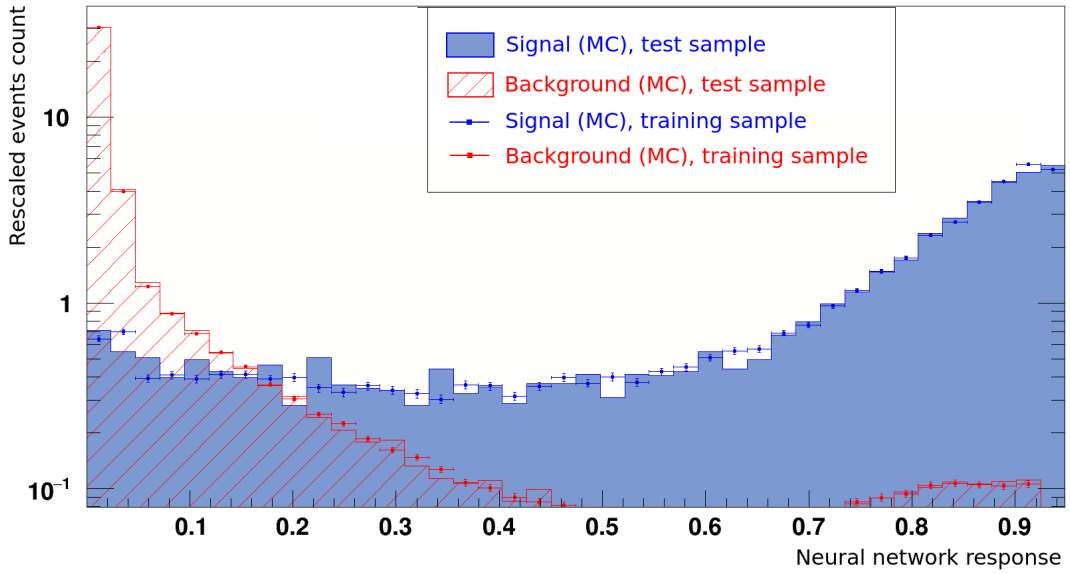
ROC curves are also a good tool to evaluate overtraining, since they are plotted during validation. If the algorithms are overtrained, they will have worse performance during testing and this will lower the curve.



(a) Feature space S_1.



(b) Feature space S_2



(a) Feature space S_3

Figure 3.7: Overtraining check for the neural network algorithms. The bin contents are rescaled so that training and testing histograms are directly comparable. The red lined (blue shaded) histograms correspond to signal (background) for testing data according to their Monte Carlo label. The red (blue) points correspond to signal (background) histograms for training data. No significant difference is noticeable in the histograms, showing no overtraining for neural networks.

Chapter 4

Testing phase

4.1 The testing dataset

The evaluation of training presented in Chapter 3 is a good indication of how the performance of the classifiers. However, the validation sample on which that was carried out was quite small, and a full scale test was necessary. In order to be somewhat independent from the training data, this test was carried out on a different data set. The study of these classifier on a full data set also allowed me to compare their performance to a more traditional approach, using uni- or bi-variate cuts for classification.

The dataset used for testing is made up of Monte Carlo data simulating the 2012 BESIII data taking run, where $(1086.9 \pm 6.0) \times 10^6$ J/ψ events were recorded. The differences between the 2012 dataset that I am using for testing and the 2009 dataset used for training are minimal. Figure (4.1) shows unlabelled and labelled invariant mass histograms of the testing dataset.

The testing sample contained $N_s = 50117$ signal events and $N_b = 287548$ background events, for a total of $N_{test} = 337665$ testing data points.

4.2 Testing procedure

The main goal of my research has been to study the background suppression that is possible to achieve using machine learning and compare it to what can be done using a standard approach. The metric I used to evaluate this comparison is the signal to background ratio in a sample, S/B, defined as the ratio of the number of true signal events to the number of true background events contained in the sample (for example after performing a cut on some variable of interest). The terms *true signal* and *true background* refer to Monte Carlo labels. As a reference, the signal to background ratio of the whole testing dataset was $(S/B)_{tot} = 0.174$, and its statistical significance $\left(\frac{S}{\sqrt{S+B}}\right)_{tot} = 86.2$.

The first step has been creating a benchmark for comparison with the MVA methods. A basic standard approach is finding the cuts on high-level variables that best separate signal from background in the testing set. The separation criterion I considered is the most widely used one, the statistical significance $S/\sqrt{S+B}$. It is calculated by considering the number of true signal (S) and true background (B) events left in the sample after the cut.

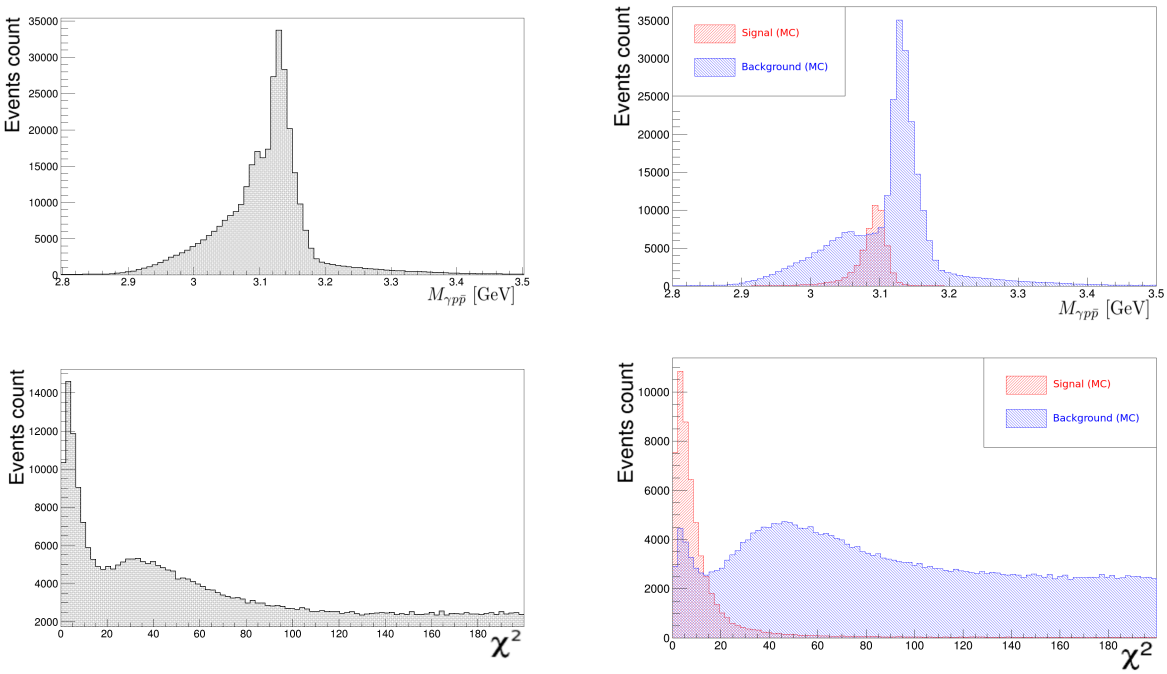


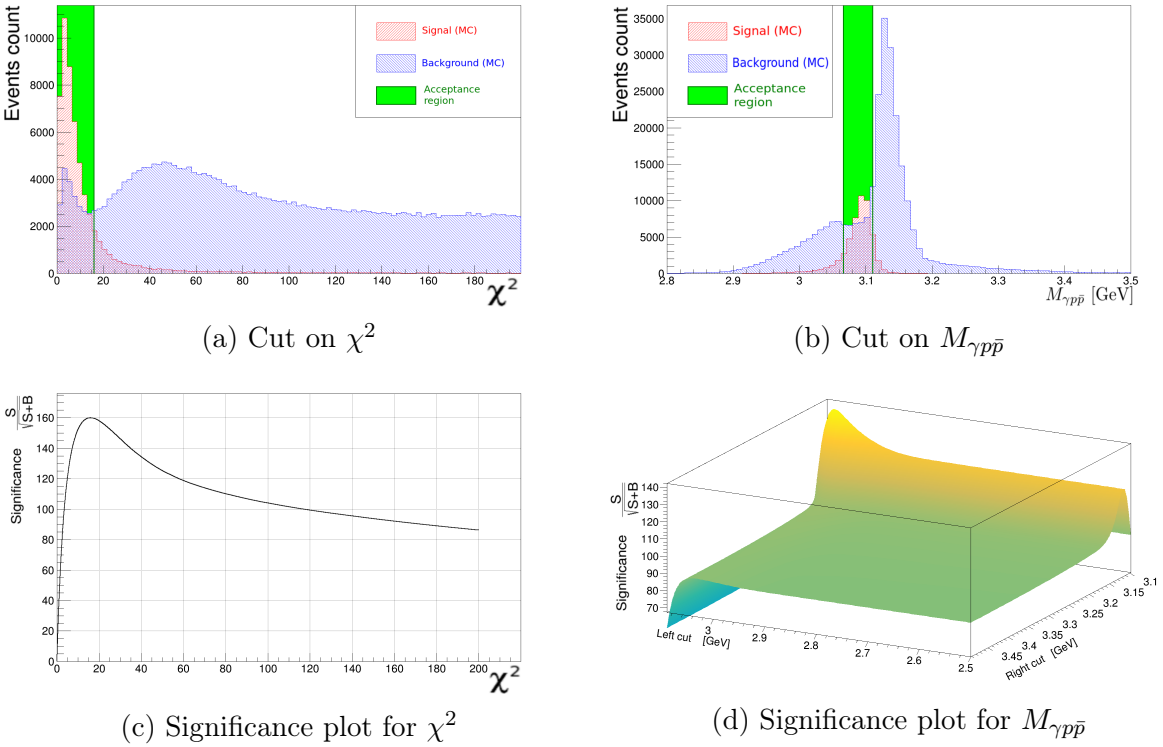
Figure 4.1: The distribution of the final-state invariant mass and chi squared for the MC data set described in the text. The histogram on the top-left is the total $M_{\gamma p\bar{p}}$ distribution, while the top right one shows the signal and background components in red and blue, respectively. The two histrograms on the bottom show the distribution of the χ^2, with the same colouring convention. The signal and background histograms, for both χ^2 and $M_{\gamma p\bar{p}}$, are not renormalised or rescaled; their area provides the actual number of events contained in each sample.

4.3 Standard approach: univariate cuts

I considered two high-level features in my analysis: the χ^2 of the $4C$ fit and the invariant mass of the particles in the final state, $M_{\gamma p\bar{p}}$.

To find the best cut on χ^2, I scanned its histogram from 0 to 200 with a grid size of 0.01, retaining events in the region $[0, C_{\chi^2}]$. The best cut I found is $C_{\chi^2} = 15.94 \pm 0.01$, with the uncertainty on it being equal to the grid size. The statistical significance was $\left(\frac{S}{\sqrt{S+B}}\right)_{\chi^2} = 160.0$, and the signal to background ratio $(S/B)_{\chi^2} = 1.65$.

For the invariant mass analysis I had to determine an acceptance interval for signal events. For the left cut I scanned the histogram from 2.5 GeV to $M_{J/\psi} = 3.097$ GeV with a grid size of 5 MeV; for the right cut I scanned from $M_{J/\psi}$ to 3.5 GeV, again with a step size of 5 MeV. The reason for these choices is the assumption that the best separating interval will include the signal peak at $M_{J/\psi}$. I found the best interval to be $I_{M_{\gamma p\bar{p}}} = [3.065 \pm 0.005, 3.112 \pm 0.005]$ GeV, with a statistical significance of $\left(\frac{S}{\sqrt{S+B}}\right)_{M_{\gamma p\bar{p}}} = 139.6$ and a signal to background ratio of $(S/B)_{M_{\gamma p\bar{p}}} = 1.00$. Again, the uncertainty on the acceptance interval boundaries are given by the step size I used in the scanning procedure.



(a) Cut on χ^2

(b) Cut on $M_{\gamma p\bar{p}}$

(c) Significance plot for χ^2

(d) Significance plot for $M_{\gamma p\bar{p}}$

Figure 4.2: Univariate cuts on the data. The red (blue) shaded histograms correspond to signal (background) contributions according to their Monte Carlo label, while the green area denotes the region where events are retained as classified signal. Also shown are the plots for the statistical significance of the cuts.

4.4 Naive machine learning: bivariate cut

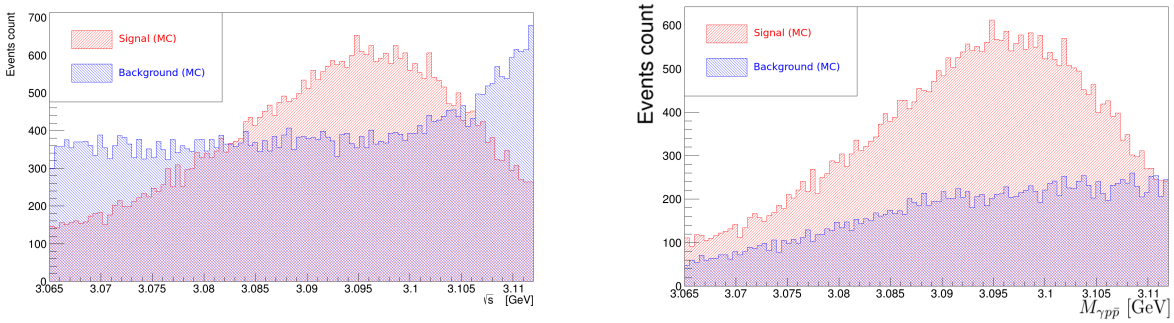
The procedure can be taken one step further by combining both univariate cuts obtained on χ^2 and $M_{\gamma p\bar{p}}$. This gives rise to a rectangular acceptance region in the (χ^2,$M_{\gamma p\bar{p}}$) space, found by maximising the statistical significance. It is bordered by the lines $\chi^2 = 0$, $\chi^2 = 15.94$, $M_{\gamma p\bar{p}} = 3.065$ GeV and $M_{\gamma p\bar{p}} = 3.112$ GeV. This approach is completely equivalent to a simple machine learning algorithm, known as *rectangular cuts*, showing how fine the line between machine learning and conventional analysis tools is.

The combination of the bivariate cuts improves the signal to background ratio of the sample: $(S/B)_{2D} = 2.11$, with a statistical significance of $\left(\frac{S}{\sqrt{S+B}}\right)_{2D} = 155.5$.

Results of the benchmark analysis of Sections (3) and (4) are shown in Figure (4.3), and summarised in Table (4.1).

Cut variable	$S/\sqrt{S+B}$	S/B
$M_{\gamma p\bar{p}}$	139.6	1.00
χ^2	160.0	1.65
$(M_{\gamma p\bar{p}}, \chi^2)$	155.5	2.11

Table 4.1: Comparison of different cuts for the standard analysis. Cutting on both χ^2 and $M_{\gamma p\bar{p}}$ gives the best background suppression while also retaining a good statistical significance.



(a) $M_{\gamma p\bar{p}}$ cut

(b) Combined χ^2 and $M_{\gamma p\bar{p}}$ cut

Figure 4.3: Invariant mass distributions after performing the $M_{\gamma p\bar{p}}$ cut (on the left side) and the combined cut on χ^2 and $M_{\gamma p\bar{p}}$. The amount of leftover background (blue histogram) is much lower for the combined cut.

4.5 Machine learning approach

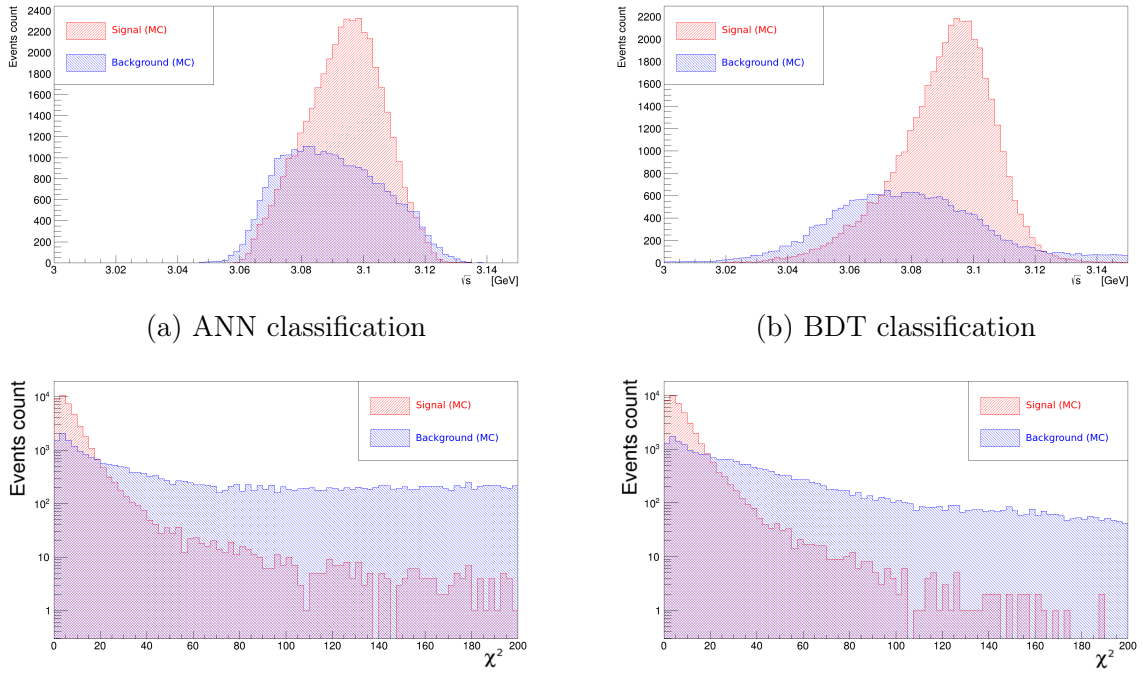
Next I discuss the results obtained with the classifiers whose training I reported in the previous chapter. For each algorithm I followed an equivalent procedure to what I did for the standard approach: I scanned the output of the classifiers (from 0 to 1 for neural networks, from -0.4 to 0.2 for boosted decision trees[1]) with a step size of 0.01. The cut that I chose was, in each case, the one that maximised the statistical significance. After doing so, I calculated the signal to background fraction in the sample remaining after the cut.

Feature space S_1

The best cut for the neural network in this feature space is at $C_{ANN} = 0.33 \pm 0.01$, while for boosted decision trees it is $C_{BDT} = 0.01 \pm 0.01$. The statistical significance of the neural network cut is $\left(\frac{S}{\sqrt{S+B}}\right)_{ANN} = 154.7$; for the boosted decision trees $\left(\frac{S}{\sqrt{S+B}}\right)_{BDT} = 160.5$. The background suppression achieved by these classifiers is $\left(\frac{S}{B}\right)_{ANN} = 1.51$ for neural networks and $\left(\frac{S}{B}\right)_{BDT} = 1.82$

The classification results for both boosted decision trees and neural networks trained on only low-level features are shown in Figure (4.4). In addition to the standard final-state invariant mass histograms, also a χ^2 histogram is shown. It is visually clear how both classifiers successfully learned a discrimination on the invariant mass: both MVA methods, in fact, recover the shape of the signal peak as seen in Figure (4.1). As can be seen by comparing Figure (4.4a) with (4.4b), the boosted decision trees algorithm was more effective at discovering it than the neural network; Figure (4.4b), in fact, shows better resolution in the signal tails than (4.4a). On the other hand, as the χ^2 histograms show, a non negligible number of events was retained even far away from the critical value $\chi^2 = 15.94$. Even though the discrimination is still quite good, this shows that the algorithms could learn this high-level feature less efficiently than $M_{\gamma p\bar{p}}$; this will also become clear when analysing the results of classifiers trained on the feature space S_3.

[1]Even though the output of a single decision tree ranges from -1 to 1, I verified that the range of the forest does not go beyond the limits of $[-0.4, 0.2]$. This is because the voting that takes place during boosting reduces the variance in the output of the whole classifier, effectively reducing the range of the response.



(a) ANN classification

(b) BDT classification

(c) ANN classification

(d) BDT classification

Figure 4.4: Histograms showing the amount of true signal and background remaining after cutting on the MVA outputs. Red (blue) shaded histograms identify signal (background) events, as identified by their Monte Carlo labels.

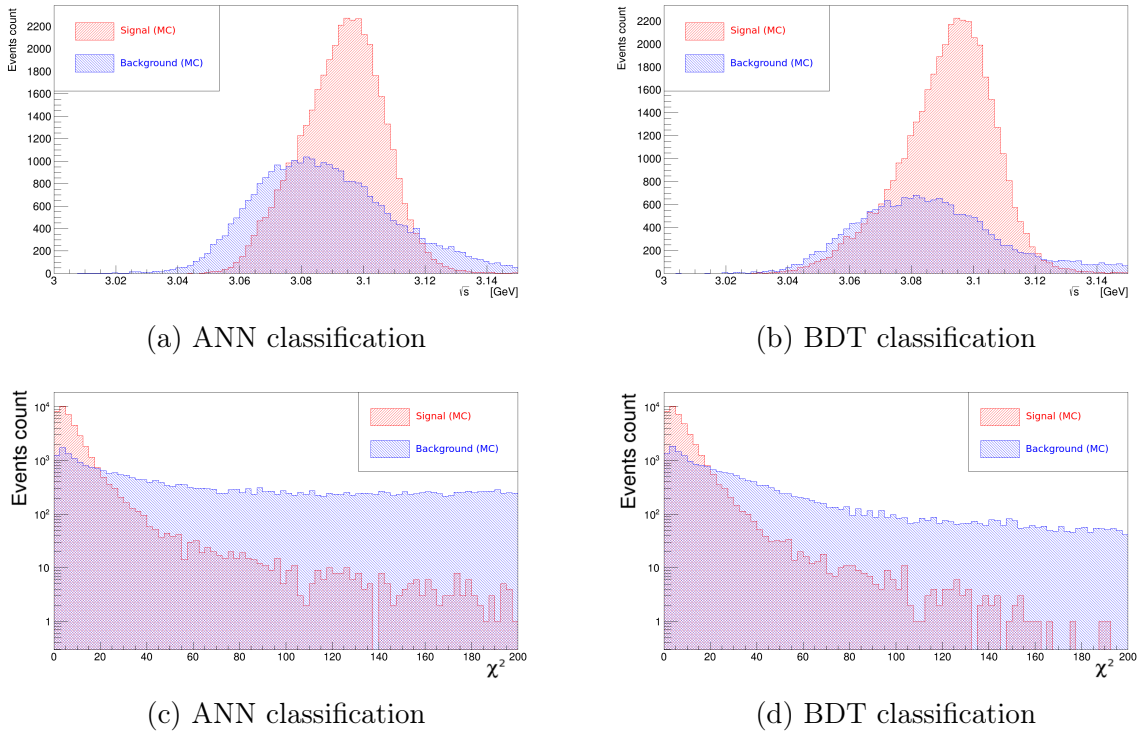
Feature space S_2

In this feature space the invariant mass is provided as an input variable, making the feature space a mix of low- and high-level features. The best cut on the neural network output is at $C_{ANN} = 0.37 \pm 0.01$; on the output of boosted decision trees is $C_{BDT} = 0.01 \pm 0.01$. The statistical significance of the ANN cut ranks very similar to their counterpart in the feature space S_1: $\left(\frac{S}{\sqrt{S+B}}\right)_{ANN} = 150.4$; for BDT $\left(\frac{S}{\sqrt{S+B}}\right)_{BDT} = 163.9$. The background suppression achieved in this feature space is $\left(\frac{S}{B}\right)_{ANN} = 1.33$ for the neural network and $\left(\frac{S}{B}\right)_{BDT} = 1.99$ for boosted decision trees. These results are shown graphically in Figure (4.5).

Feature space S_3

The best cut for the neural network in this feature space is at $C_{ANN} = 0.41 \pm 0.01$; for the boosted forest it is $C_{BDT} = 0.01 \pm 0.01$. The statistical significance of the neural network cut is $\left(\frac{S}{\sqrt{S+B}}\right)_{ANN} = 180.8$; for the boosted decision trees $\left(\frac{S}{\sqrt{S+B}}\right)_{BDT} = 181.6$. The background suppression achieved by these classifiers is $\left(\frac{S}{B}\right)_{ANN} = 3.40$ for neural networks and $\left(\frac{S}{B}\right)_{BDT} = 3.55$

It is interesting to note that when trained and optimised for this feature space, both classifiers have almost exactly the same performance in terms of background suppression. This even performance could be predicted already in the training stage (Chapter 3), since the ROC curves for both classifiers are exactly on top of each other. Invariant mass histograms of the retained events are shown in Figure (4.6); they confirm also visually the very good discrimination reached when machine leaning algorithms are trained with a combination of low-level features and high-



(a) ANN classification

(b) BDT classification

(c) ANN classification

(d) BDT classification

Figure 4.5: Histograms showing the amount of true signal and background remaining after cutting on the MVA outputs in feature space S_2.

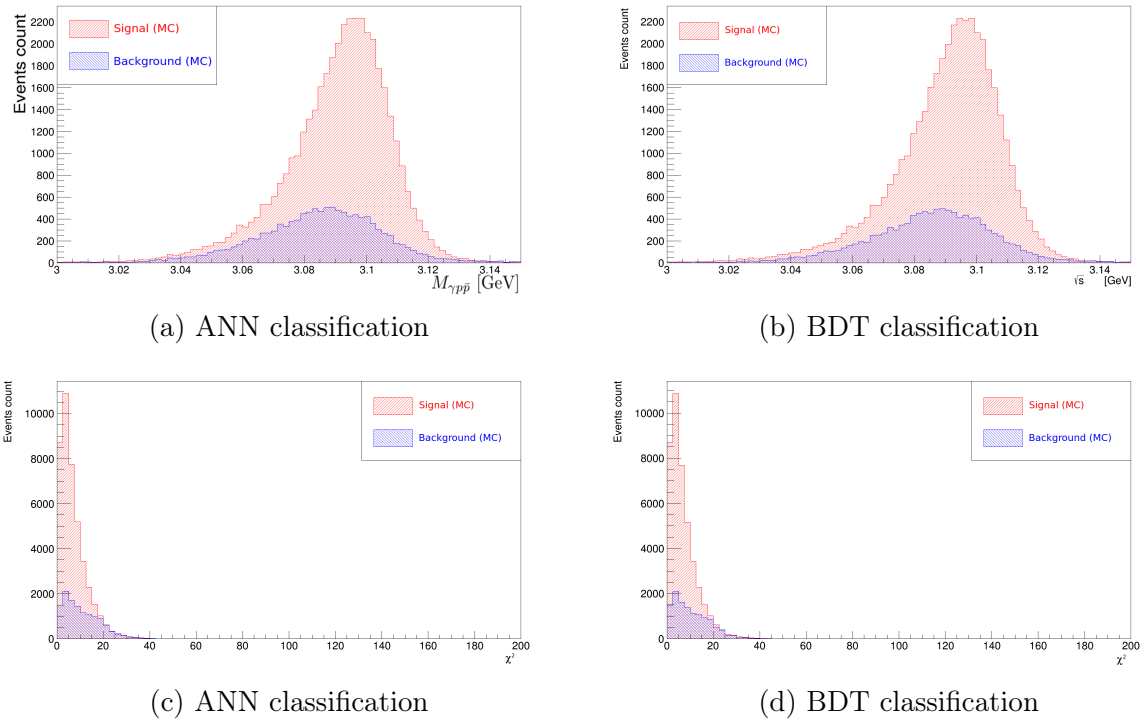
level features.

4.6 Discussion

A summary for the testing results of MVA methods is given in Table (4.2). It is evident how the inclusion of the high-level feature χ^2 drastically improves the performance of the classifiers.

Comparing the results of algorithms on feature spaces S_1 and S_2 shows that, indeed, the MVA methods trained on the low-level features learned to discriminate at a higher level, in particular the invariant mass $M_{\gamma p\bar{p}}$. It is possible to give an intuitive explanation of the difference between the neural network classification across different feature spaces. When provided with only low level features, the network learns to identify and cut out background more efficiently in the tails of the histogram, compared to the network that included $M_{\gamma p\bar{p}}$. Comparing the histograms of Figures (4.6c) and (4.6d) with their counterparts for feature spaces S_1 and S_2 it can be seen how an explicit cut on chi squared was not found by any classifier which did not include χ^2 in its feature space. The reason for this, as anticipated previously, is that χ^2 is a very high-level feature: many calculation steps are required to get to it explicitly, and it is therefore harder to retrieve for a classifier. Keeping in mind that neural networks tend to overtrain when the number of weights grows very large [16], a multilayer perceptron with many more hidden layers than the one I used in space S_1 might be able to learn a discrimination on χ^2. The same can not be said about the boosted decision trees algorithm: while a neural network has no theoretical upper limit on architecture complexity, the BDT method can only be optimised until a certain threshold. The BDT machine that I trained on space S_1 is the best *forest* algorithm with those input features, and thus BDT can not be improved to learn explicit cuts on χ^2.

When compared to the standard approach of uni- or bi-variate rectangular cuts, machine learn-



(a) ANN classification

(b) BDT classification

(c) ANN classification

(d) BDT classification

Figure 4.6: Histograms showing the amount of true signal and background remaining after cutting on the MVA outputs in feature space S_3.

ing algorithms have generally shown a better performance. This improvement in classification can be established not only by the values of S/B across different classification methods, but also graphically by looking at the signal tails in Figures (4.4), (4.5) and (4.6). More events in these tails are retrieved by the multivariate classifiers than by the standard analysis; this shows how a multivariate (and, as is the case of multilayer perceptron, nonlinear) approach can correctly retain events far away from the signal peak.

Overall, classifiers trained on the feature space S_3 proved to work far better than standard analysis and better than other classifiers for the particular problem at hand.

Feature space	MVA method	Best cut ± 0.01	$S/\sqrt{S+B}$	S/B	SIG efficiency	BKG rejection
S_1	ANN	0.33	154.7	1.51	0.79	0.91
S_1	BDT	0.01	160.5	1.82	0.80	0.92
S_2	ANN	0.37	150.4	1.33	0.79	0.90
S_2	BDT	0.01	163.9	1.99	0.81	0.93
S_3	ANN	0.41	180.8	3.40	0.84	0.96
S_3	BDT	0.01	181.6	3.55	0.84	0.96
Standard cut		Best cut	$S/\sqrt{S+B}$	S/B	SIG efficiency	BKG rejection
$M_{\gamma p\bar{p}}$		$[3.065, 3.112]$ GeV	139.6	1.00	0.78	0.86
χ^2_{4C}		15.94	160.0	1.65	0.91	0.82
$(M_{\gamma p\bar{p}}, \chi^2)$		$\uparrow\uparrow$	155.5	2.11	0.71	0.94

Table 4.2: A summary table for the classification with machine learning algorithms, compared to the standard approach results. Also included are the background rejection and the signal efficiency of each particular cut.

Chapter 5

Application phase

5.1 Experimental data classification

As a final step in my research, I analysed a batch of data collected at the BESIII experiment in 2009 and 2012 [22]. This data has been preprocessed as described in Chapter 3 for Monte Carlo (MC) data. In contrast to MC data, these data do not contain any signal and background labels. It is, therefore, harder to study quantitatively the performance of the classifiers. Furthermore, note that the classifiers are trained on MC data. Discrepancies between signal and background distributions of the MC and experimental data might also influence the performance. The data sample was made up of 2165282 ($\approx 2 \cdot 10^6$) events. A histogram of the final-state invariant mass is shown in Figure (5.1).

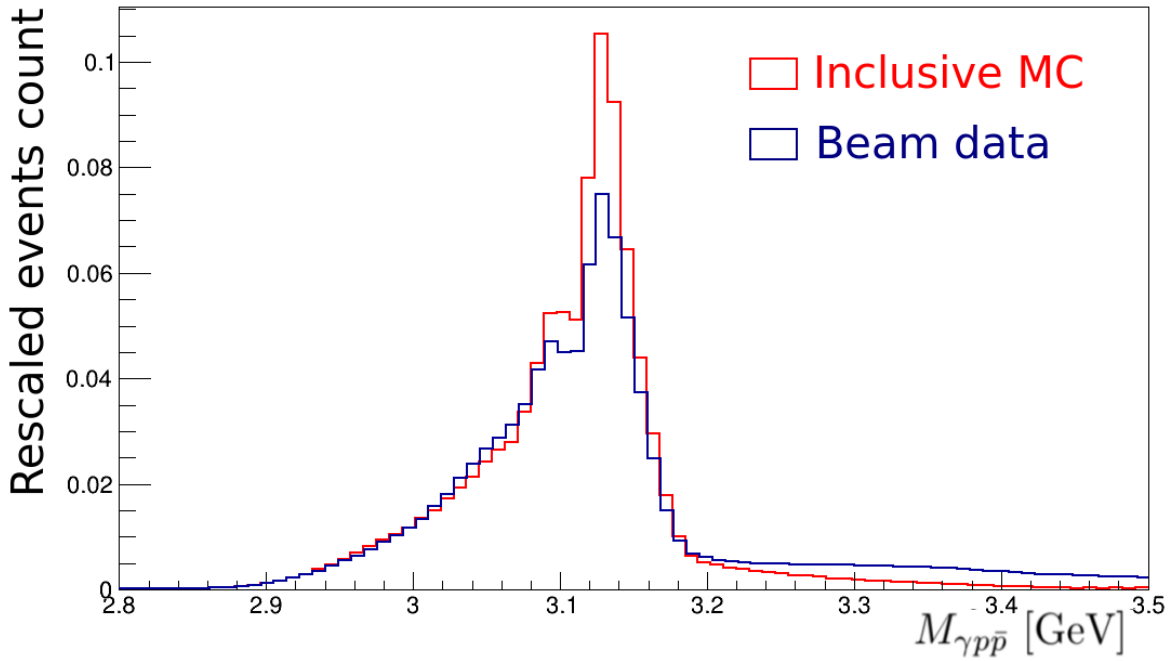
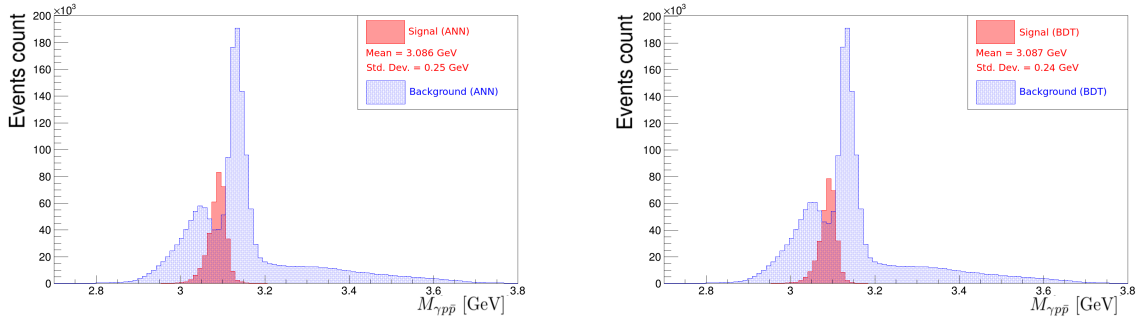


Figure 5.1: This histograms shows the final-state invariant mass of the beam data I have used for the classification. The blue solid line is the experimental data, and the Monte Carlo data that was used for testing in Chapter 4 is shown in red as a comparison. The histograms are rescaled to have unit area.

For the classification I used the algorithms that performed best during testing, namely those trained on the feature space S_3. I used both boosted decision trees and neural networks for

this classification task. This is because there are differences between the MC and experimental data, and therefore there might be some discrepancies in performance with the tests reported in Chapter 4. As it is possible to see in Figure (5.2), the multivariate methods produced very similar results. This confirms once more the results explained in the previous chapters.



(a) ANN classification (b) BDT classification

Figure 5.2: Histograms showing the invariant mass of the final-state as classified by neural networks (ANN, on the left) and boosted decision trees (BDT, on the right). In these figures, unlike in previous chapters, the colors refer to the outcome of the machine learning algorithms. The classifiers have an almost identical performance, as can be seen by both the shapes of the histograms and their mean and standard deviation.

5.2 Preliminary discussion

As mentioned above, both classifiers retrieved the same information from the data. Therefore I will not distinguish further between the algorithms when discussing the qualitative physical results of the classification.

A clear signal peak has been observed, centered around[1] $m_{peak} = 3.0867$ GeV, with a standard deviation of 0.025 GeV. This peak corresponds to a particle with mass equal to m_{peak}. This mass is compatible with the accepted mass of the J/ψ, $M_{J/\psi} = 3096.900 \pm 0.006$ MeV [20].

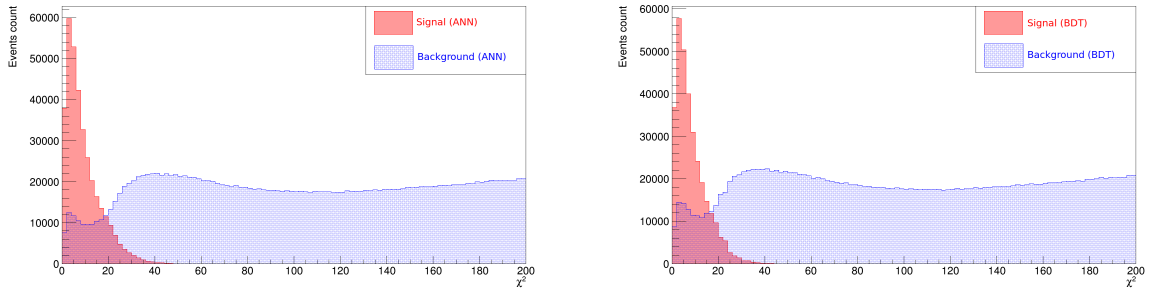
In addition to this, it is possible to retrieve useful information about the background as well. In particular the study of the Monte Carlo background can lead to the identification of the two main sources of background in this data sample. The leftmost blue peak in Figure (5.2) corresponds to the $J/\psi \to \pi^0 p\bar{p}$ decay, while the rightmost one is made up of $J/\psi \to p\bar{p}$ events.

The signal peak shows good separation from the two background peaks; comparing Figure (5.2) to any invariant mass histogram of inclusive Monte Carlo signal and background data from the previous chapters it is possible to see a very close similarity, giving a qualitative account of the precision of this classification.

The only difference between the neural network and the decision tree forest is the number of signal events they retrieve: the ANN method identified $N_{ANN} = 347271$ signal events, while the BDT algorithm classified $N_{BDT} = 315243$ events as signal. Comparing these results to what presented in Chapter 4, it can be noted that this is probably due to the lower background

[1] Since the variances of both results are almost identical, I took an arithmetic average for the histogram peak; taking into account the number of significant digits in the uncertainty, this average is $3.0865 \to 3.087$.

rejection of the ANN method. This means that some of those 30000 extra events, present in ANN classification but not in BDT, are misclassified background events.



(a) ANN classification

(b) BDT classification

Figure 5.3: Chi-squared histograms of the results of the multivariate analysis on the unlabelled data. Red (blue) histograms show signal (background) as classified by the machine learning algorithms.

5.3 Evaluation of performance: the $p\bar{p}$ mass spectrum

As mentioned in the first section, it is hard to quantify the performance of MVA methods since beam data lacks labels. One can, however, make use of prior knowledge of the topology of the events to gain some insights in the process. In particular, the invariant mass spectrum of the $p\bar{p}$ pair for the events in the data sample exhibits two distinct, evident peaks. This spectrum is shown in Figure (5.5). The evident peaks correspond to the $J/\psi \to \gamma\eta_c \to \gamma p\bar{p}$ decay, centered at $M_{\eta_c} = 2980$ GeV, and to $J/\psi \to p\bar{p}$, at $M_{J/\psi} = 3.097$ GeV. These peaks are, respectively, pure signal and pure background peaks, as discussed in Chapter 3. A shape and size comparison of the η_c peak before and after the classification is a good way to measure the signal efficiency of the classifiers.

Procedure

Ideally, the η_c peak after classification should contain the same number of events that the same peak contains before the classification. This would correspond to a classifier with a 100% signal efficiency. In a real situation, some of the signal events in the peak are misclassified as signal. By fitting the peak after classification and rescaling it to fit the peak before classification it is possible to calculate the relativa size of the peaks, and thus the fraction of correctly identified events.

The method I followed consisted in two different. In particular, I first fit the η_c peak that is leftover *after* the MVA classification. The function I used to fit the spectrum is a double Gaussian plus a second-order polynomial:

$$F_{after}(x|\alpha_0, \alpha_1, \alpha_2, \mathbf{p}) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + A_1 e^{\left(\frac{x-\mu_1}{\sigma_1}\right)^2} + A_2 e^{\left(\frac{x-\mu_2}{\sigma_2}\right)^2}, \tag{5.1}$$

where x, in GeV, is the invariant mass of the $p\bar{p}$ system, the α_i are uninteresting coefficients that will be discarded after the fit, and $\mathbf{p} = (A_1, \mu_1, \sigma_1, A_2, \mu_2, \sigma_2)$ is a vector containing the important fit parameters that fix the shape of the η_c peak. The second-order polynomial models the signal that is not due to the decay through the intermediate resonance η_c, while the double

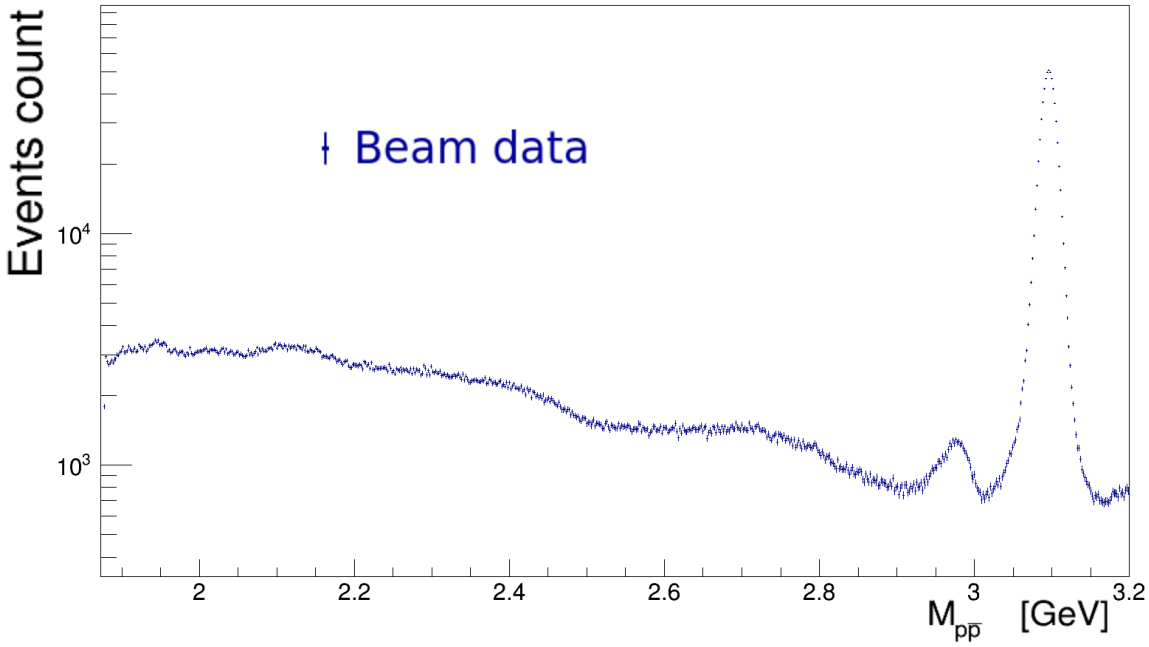


Figure 5.4: The $p\bar{p}$ invariant mass spectrum for the events in the beam data set. The two distinct peaks towards the right end of the spectrum correspond to $J/\psi \to \gamma\eta_c \to \gamma p\bar{p}$ (left), and $J/\psi \to p\bar{p}$ (right). For more details, see text.

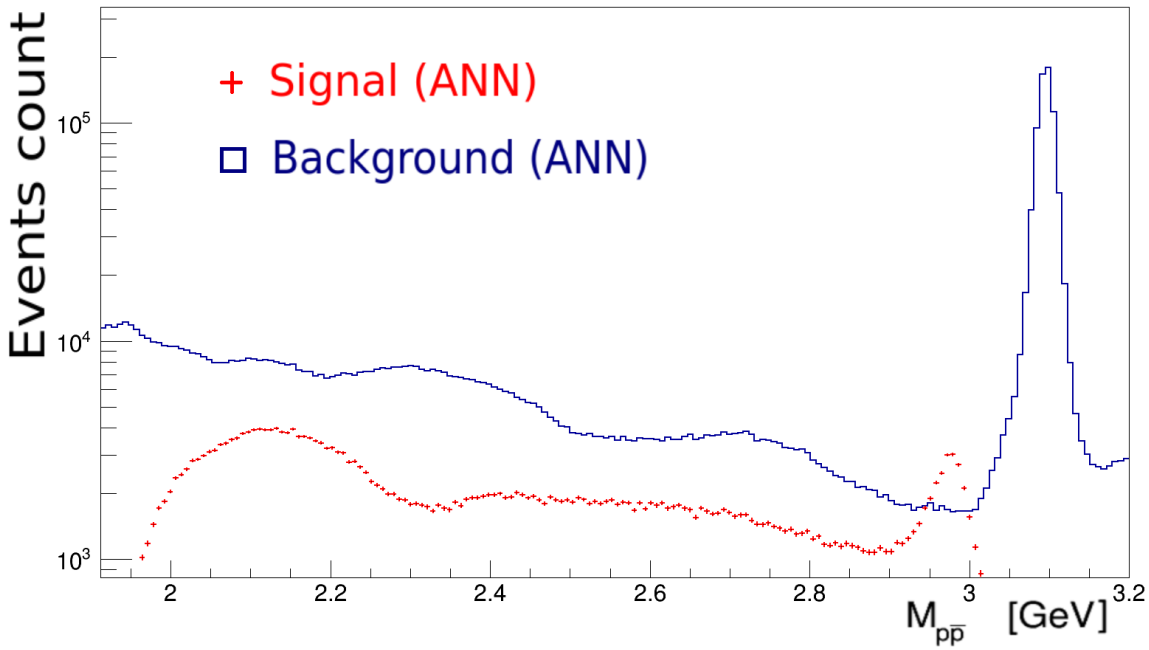


Figure 5.5: The $p\bar{p}$ invariant mass spectrum for the events in the beam data set, after the classification with the artificial neural network. The most notable features of this classification are the absence of a peak in the background (blue) histogram in the region around $M_{\eta_c} = 2.980$ GeV, and the absence of events in the signal (red) histogram in the region $M_{p\bar{p}} > M_{J/\psi} = 3.097$ GeV.

Gaussian models the peak.

With the fit, the shape of the double Gaussian was fixed in the best fitting parameters $\hat{\mathbf{p}}$. The second step of the procedure was then to fit, with the parameters just found, the η_c peak in the beam data spectrum *before* the classification. The fitting function was still a double Gaussian plus a second-order polynomial:

$$F_{before}(x|\beta_0,\beta_1,\beta_2,S,\hat{\mathbf{p}}) = \beta_0+\beta_1 x+\beta_2 x^2 + S\left(\hat{A}_1 e^{\left(\frac{x-\hat{\mu}_1}{\hat{\sigma}_1}\right)^2} + \hat{A}_2 e^{\left(\frac{x-\hat{\mu}_2}{\hat{\sigma}_2}\right)^2}\right), \tag{5.2}$$

where the β_i model all the events that are not η_c events (including both signal and background), and S is a scale factor for the η_c peak. The size of the peak is proportional to the number of events in it, therefore the inverse of the scale factor S is a good measure for the signal efficiency of the classifier (since the scale of the double Gaussian in F_{after} is 1).

The fits were carried out using MINUIT, a chi-square minimisation algorithm in ROOT.

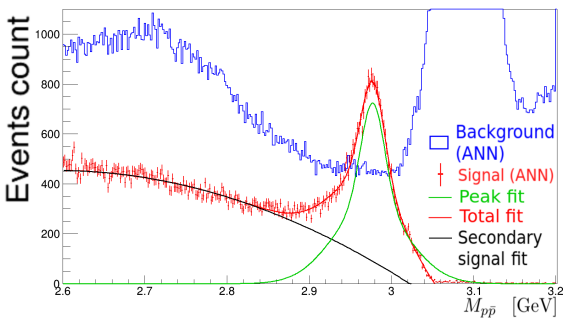
Ft results

The results of the two fits are shown in Figure (5.6). The important parameters for the fit are shown in Table (5.1). The double Gaussian is made up of a sharp curve peaked at the η_c mass and a broader, slightly offset Gaussian that fits the asymmetric left tail.

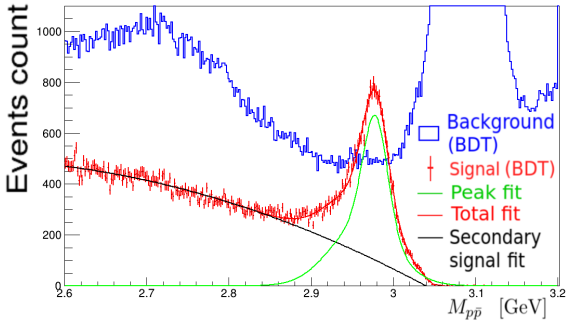
Parameter	Value (ANN)	Value (BDT)
$\hat{A}_1$	$(4.12 \pm 0.18) \times 10^2$	$(4.47 \pm 0.18) \times 10^2$
$\hat{\mu}_1$	(2.9773 ± 0.0004) GeV	(2.9781 ± 0.0004) GeV
$\hat{\sigma}_1$	$(1.44 \pm 0.07) \times 10^{-2}$ GeV	$(1.48 \pm 0.05) \times 10^{-2}$ GeV
$\hat{A}_2$	$(3.12 \pm 0.16) \times 10^2$	$(2.41 \pm 0.16) \times 10^2$
$\hat{\mu}_2$	(2.9744 ± 0.0003) GeV	(2.96220 ± 0.00010) GeV
$\hat{\sigma}_2$	$(4.45 \pm 0.23) \times 10^{-2}$ GeV	$(3.90 \pm 0.16) \times 10^{-2}$ GeV
$\hat{S}$	1.066 ± 0.020	1.036 ± 0.020
$\hat{\beta}_0$	$(5.768 \pm 0.016) \times 10^4$	$(5.82 \pm 0.02) \times 10^4$
$\hat{\beta}_1$	$(-3.645 \pm 0.011) \times 10^4$ GeV^{-1}	$(-3.295 \pm 0.014) \times 10^4$ GeV^{-1}
$\hat{\beta}_2$	$(5.802 \pm 0.021) \times 10^3$ GeV^{-2}	$(5.72 \pm 0.03) \times 10^3$ GeV^{-2}

Table 5.1: Best fit parameters for the η_c fit. The parameter above the double line are fitted to the signal peak after classification, and they are kept fixed in the fit of the data before classification. The parameters below the double line are determined during the fit of the spectrum before classification. For details refer to the equations in the Procedure section.

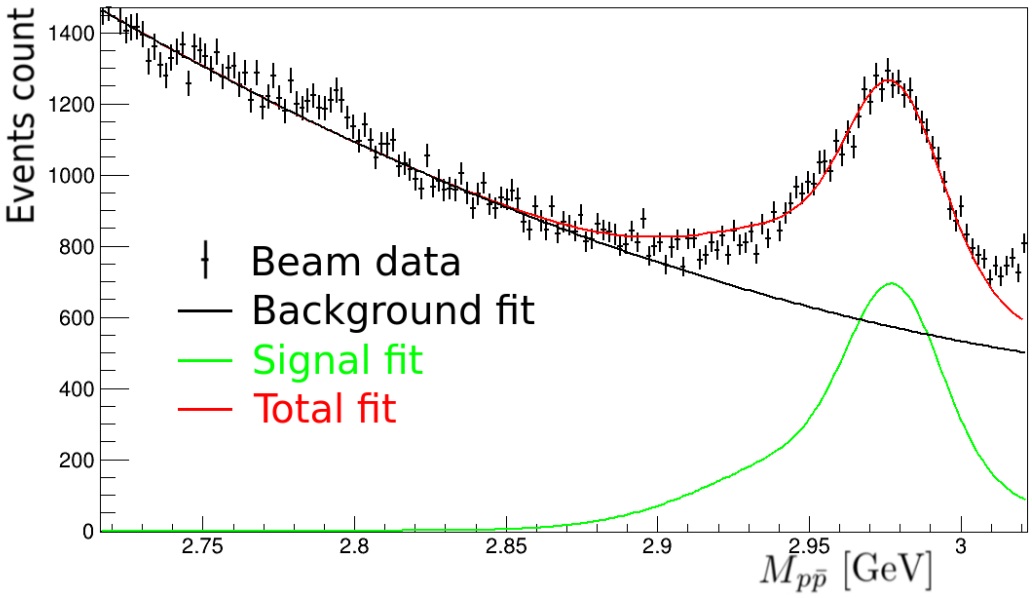
From the best estimate of the parameter S it is now possible to calculate the signal efficiency of the neural network classifier. As stated above, the scale of the double Gaussian after classification (in Eq. (5.1)) is 1, therefore $\varepsilon_S = 1/\hat{S}$. There are sources of uncertainty in the calculation of this number. Most notably, this fit is less accurate in the region of the spectrum close to $M_{J/\psi}$, since a new peak appears there in the total spectrum which was not considered in the fit. There is a large uncertainty in the description of the background. This leads to a large error in the measurement. Furthermore, the uncertainty on the counts, given for each bin by the square root of the number of entries in the bin itself, is quite high. A measure to estimate the uncertainty on the signal efficiency is, therefore, to calculate it for the counted events around the peak. In particular, the relative counting error for a region with S signal events and B



(a) ANN classification and fit



(b) BDT classification and fit



(c) Fit to the total spectrum

Figure 5.6: The fit results for the $M_{p\bar{p}}$ spectrum after classification with neural networks (left), with boosted decision trees (right) and for the total spectrum (bottom). The green line on the top plots fits the η_c peak and its rescaled shape is used to fit the corresponding peak in the total data sample (red line on the bottom peak). The red (blue) histograms in the top figures are the classified signal (background). No peak is distinguishable in the background histogram in the region of the η_c peak.

background events is $\frac{\sqrt{S+B}}{S}$. I calculated this number in the histogram before classification, making use of the relation

$$\frac{\sqrt{S+B}}{S} = \frac{\sqrt{I_{tot}}}{I_s},$$

where I_{tot} is the integral of the total fitting function in the region of interest, and I_s is the integral of the double Gaussian in the same region. The integrals were calculated in the interval $[\mu_x - 2\sigma_x, \mu_x + 2\sigma_x]$, where μ_x and σ_x are, respectively, the statistical averages of the means and standard deviations of the double Gaussian. The resulting relative uncertainties were $\delta_{ANN} = 0.21$, $\delta_{BDT} = 0.27$. The final measurement for the signal efficiencs of the two classifiers are

$$\varepsilon_s^{ANN} = 0.94^{+0.06}_{-0.19},$$
$$\varepsilon_s^{BDT} = 0.97^{+0.03}_{-0.26}.$$

Chapter 6

Conclusion

Going back to the beginning, I can now answer the research question that triggered my work: Can machine learning improve the background suppression for the $J/\psi \to \gamma p\bar{p}$ process in BESIII? I now outline the conclusions of my research, showing that machine learning algorithms indeed improved the signal to backrground ratio compared to standard analysis techniques.

6.1 Features

Not all multivariate classifiers had the same performance. A slight variation was observed between artificial neural networks and boosted decision trees; however, the largest impact on the classification accuracy of the algorithms was not their nature, but the feature space they were provided. Optimised BDT and ANN had very similar performance when provided the same input features, but some input variables give better resolution than others.

Therefore very good results can be reached using machine learning algorithms, but only with a certain combination of input features. My analysis has shown that, out of the three benchmark feature spaces I tested, the one made up of low-level features and the chi-squared of the four-constraint fit (S_3) gave considerably better results than the other two. The feature space S_1, containing only the low-level features, achieved roughly the same performance as the space, S_2, which added to S_1 the invariant mass of the final state. This shows that classifiers trained on only low-level features can learn to discriminate also on higher level variables: in particular, given the four-momenta of the final-state particles the classifiers have reconstructed their invariant mass by finding the appropriate combination of the twelve input variables. The same cannot be said for the χ^2 of the four-constraints fit; the information codified by this variable is too high-level for the classifiers to learn.

6.2 Comparison to the standard approach

For Monte Carlo data, all machine learning algorithms outperformed simple univariate cuts on either the χ^2 or the invariant mass $M_{\gamma p\bar{p}}$. In case of neural networks trained on feature spaces S_1 and S_2, the background suppression was slightly inferior to the one achieved with the χ^2 cut, but the algorithms retrieved signal events even farther away from the peak; this compensate the lower signal to background ratio. All other classifiers achieved better background suppression in addition to retaining signal on a broad range.

On the other hand, only the algorithms trained on feature space S_3 suppressed background more efficiently than the combined χ^2 and $M_{\gamma p\bar{p}}$ cut on the data; as mentioned above, though, also the classifiers that did not perform as well managed to retain more events in the tails, which were discarded by the bivariate cut.

Overall, neural networks and boosted decision trees trained with the chi-squared of the four-constraint fit in addition to the components of the four-momenta of the final-state particles achieved a drastic improvement in background suppression, compared to other classifiers and to standard analysis techniques. Using machine learning algorithms to identify the process $J/\psi \to \gamma p\bar{p}$ can therefore provide very solid results.

6.3 Application to beam data

Although harder to determine with accuracy, multivariate analysis techniques proved to work well on experimental data as well. The study of a known signal peak, the η_c peak in the proton-antiproton mass spectrum, has allowed the calculation of signal efficiencies for both neural networks and boosted decision trees. That, in addition to the correct identification of the main signal peak in the final-state invariant mass spectrum, provides indication that machine learning techniques can be used succesfully to classify experimental data.

6.4 Drawbacks

I identified three drawbacks associated to using machine learning for this type of analysis; they are mostly generic drawbacks associated to machine learning rather than to this particular situation, but they are still worth mentioning since they might affect the analysis.

The first problem is the poor adaptability of classifiers to different problems to the ones they were trained for. The process of training, especially for neural networks, is very specific and requires a long and careful tuning of the hyperparameters and of the classifier's structure. The outcome is an algorithm that is tailored around the problem at hand, and a different problem will most likely require training a new classifier from scratch. The algorithms I trained work very well for the identification of $J/\psi \to \gamma p\bar{p}$, but they are close to useless for the detection of different channels with different topology.

Closely related to this is the second drawback, namely the need of large and reliable datasets for the training. In order to learn properly, the algorithms require a large amount of data, and each event has to be labelled correctly in order for the classifier to learn to discriminate correctly. If the goal is to study a particular channel about which very little is known, too little to produce reliable simulations, supervised machine learning methods like the ones I have tested can be of little help. In these situations it is advisable to resort to unsupervised machine learning algorithms, such as clustering, to find patterns in the unlabelled data.

The third drawback is by far the least serious, and it relates to classification speed: as mentioned in Chapter 2, the most precise algorithm (boosted decision trees on the feature space S_3) is very slow and would not be very reliable to use, even when embedded on graphical processing units, for real-time classification tasks. A compromise might have to be accepted between performance and speed for certain applications.

6.5 Conclusion and outlook

In conclusion, machine learning is a promising approach for the search of $J/\psi \to \gamma p\bar{p}$ in BESIII: when the right feature space is used, multivariate methods can lead to a much higher background suppression than standard methods. For online applications, artificial neural networks are suggested; otherwise, for offline classification tasks, boosted decision trees are to be preferred since they trade speed with accuracy.

An interesting opportunity for further research can be found in the article by Baldi et al., [23], where the authors present a new method that allows neural networks to discover the value of a parameter, unknown during training. The classifier is trained with just some benchmark values for the unknown parameter, and in classification stage it can interpolate efficiently to discover the correct value. The interesting application to particle physics is the discovery of intermediate resonances; in the context of BESII, this method can be applied to the discovery of charmed mesons above the open-charm threshold.

Additionally, the work presented here can be expanded into multi-class classification. Rather than dividing data into signal or background, one could separate it further by identifying, cor each event, the process it corresponds to. This would lead to, for example, separating the final-state invariant mass histograms seen in previous chapters into three main peaks. These peaks would represent $J/\psi \to \gamma p\bar{p}$, which now falls into signal, $J/\psi \to \pi^0 p\bar{p}$ and $J/\psi \to p\bar{p}$, which are now considered background.

Bibliography

[1] D. M. Asner et al. "Physics at BES-III". In: *Int. J. Mod. Phys.* A24 (2009), S1–794. arXiv: 0809.1869 [hep-ex].

[2] *Atlas experiment*. https://en.wikipedia.org/wiki/ATLAS_experiment. Accessed: 11-06-2019.

[3] Avinash Khare. "The November J / psi revolution: Twenty five years later". In: *Curr. Sci.* 77 (1999), p. 1210. arXiv: hep-ph/9910468 [hep-ph].

[4] M. Ablikim et al. "Spin-Parity Analysis of $p\overline{p}$ Mass Threshold Structure in J/ψ and $\psi(3686)$ Radiative Decays". In: *Phys. Rev. Lett.* 108 (11 Mar. 2012), p. 112003. DOI: 10.1103/PhysRevLett.108.112003. URL: https://link.aps.org/doi/10.1103/PhysRevLett.108.112003.

[5] Vincent Mathieu, Nikolai Kochelev, and Vicente Vento. "The Physics of Glueballs". In: *Int. J. Mod. Phys.* E18 (2009), pp. 1–49. DOI: 10.1142/S0218301309012124. arXiv: 0810.4453 [hep-ph].

[6] Biagio Lucini. "Glueballs from the Lattice". In: *PoS* QCD-TNT-III (2013), p. 023. DOI: 10.22323/1.193.0023. arXiv: 1401.1494 [hep-lat].

[7] Ulrich Wiedner. "Future Prospects for Hadron Physics at PANDA". In: *Prog. Part. Nucl. Phys.* 66 (2011), pp. 477–518. DOI: 10.1016/j.ppnp.2011.04.001. arXiv: 1104.3961 [hep-ex].

[8] Dan Guest, Kyle Cranmer, and Daniel Whiteson. "Deep Learning and its Application to LHC Physics". In: *Ann. Rev. Nucl. Part. Sci.* 68 (2018), pp. 161–181. DOI: 10.1146/annurev-nucl-101917-021019. arXiv: 1806.11484 [hep-ex].

[9] Marian Stahl. "Machine learning and parallelism in the reconstruction of LHCb and its upgrade". In: *J. Phys. Conf. Ser.* 898.4 (2017), p. 042042. DOI: 10.1088/1742-6596/898/4/042042. arXiv: 1710.08947 [physics.ins-det].

[10] Michela Paganini. "Machine Learning Algorithms for b-Jet Tagging at the ATLAS Experiment". In: *J. Phys. Conf. Ser.* 1085.4 (2018), p. 042031. DOI: 10.1088/1742-6596/1085/4/042031. arXiv: 1711.08811 [hep-ex].

[11] A. Edelen et al. "Opportunities in Machine Learning for Particle Accelerators". In: (2018). arXiv: 1811.03172 [physics.acc-ph].

[12] *BESIII website*. http://bes3.ihep.ac.cn/. Accessed: 15-06-2019.

[13] Wang Ji-Ke et al. "BESIII track fitting algorithm". In: *Chinese Physics C* 33.10 (Oct. 2009), pp. 870–879. DOI: 10.1088/1674-1137/33/10/010. URL: https://doi.org/10.1088%2F1674-1137%2F33%2F10%2F010.

[14] Miao He. "Simulation and reconstruction of the BESIII EMC". In: *Journal of Physics: Conference Series* 293 (Apr. 2011), p. 012025. DOI: 10.1088/1742-6596/293/1/012025. URL: https://doi.org/10.1088%2F1742-6596%2F293%2F1%2F012025.

[15] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014. ISBN: 0262028182, 9780262028189.

[16] Andreas Hocker et al. "TMVA - Toolkit for Multivariate Data Analysis". In: (2007). arXiv: physics/0703039 [physics.data-an].

[17] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[18] Hrushikesh Mhaskar, Qianli Liao, and Tomaso A. Poggio. "Learning Real and Boolean Functions: When Is Deep Better Than Shallow". In: *CoRR* abs/1603.00988 (2016). arXiv: 1603.00988. URL: http://arxiv.org/abs/1603.00988.

[19] *ROOT data analysis framework*. https://root.cern.ch/. Accessed: 12-06-2019.

[20] M. Tanabashi et al. "Review of Particle Physics". In: *Phys. Rev. D* 98 (3 Aug. 2018), p. 030001. DOI: 10.1103/PhysRevD.98.030001. URL: https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[21] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for Exotic Particles in High-Energy Physics with Deep Learning". In: *Nature Commun.* 5 (2014), p. 4308. DOI: 10.1038/ncomms5308. arXiv: 1402.4735 [hep-ph].

[22] Medina Ablikim et al. "Determination of the number of J/ψ events with inclusive J/ψ decays". In: *Chin. Phys.* C41.1 (2017), p. 013001. DOI: 10.1088/1674-1137/41/1/013001. arXiv: 1607.00738 [hep-ex].

[23] Pierre Baldi et al. "Parameterized neural networks for high-energy physics". In: *Eur. Phys. J.* C76.5 (2016), p. 235. DOI: 10.1140/epjc/s10052-016-4099-4. arXiv: 1601.07913 [hep-ex].