Karlsruher Institut für Technologie

IEKP-Bachelor-KA/2015-07

sPlot-based Training of Multivariate Classifiers in the Belle II Analysis Software Framework

Benjamin Lipp

Bachelor Thesis

Fakultät für Physik
Karlsruher Institut für Technologie (KIT)

Referent: Prof. Dr. Michael Feindt
Korreferent: Dr. Martin Heck

Institut für Experimentelle Kernphysik

Mai 2015

Karlsruher Institut für Technologie

IEKP-Bachelor-KA/2015-07

sPlot-basiertes Training multivariater Klassifizierer im Belle II Analysis Software Framework

Benjamin Lipp

Bachelorarbeit

Fakultät für Physik
Karlsruher Institut für Technologie (KIT)

Referent: Prof. Dr. Michael Feindt
Korreferent: Dr. Martin Heck

Institut für Experimentelle Kernphysik

Mai 2015

Erklärung zur Selbstständigkeit

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung vom 17.05.2010 beachtet habe.

Karlsruhe, den 12.05.2015, _____

Benjamin Lipp

Contents

List of Figures

List of Tables

1. Introduction

Belle II is a particle physics experiment currently being built at the KEK High Energy Accelerator Research Organization in Tsukuba, Japan as an upgrade of the Belle experiment. Data taking is planned to start in the year 2018. The Belle experiment collected data from 1999 to 2010. Important findings of Belle include the confirmation of the Kobayashi–Masukawa mechanism and other measurements related to CP violation. Two major goals of the Belle II experiment are the further study of CP violation and of rare B decays.

The Belle II experiment consists of the SuperKEKB asymmetric e^-e^+ collider and the Belle II multi-purpose particle detector. The collider operates at the energy of the $\Upsilon(4S)$ resonance, which means that many of the electron positron collisions produce a $\Upsilon(4S)$ meson; this is an exited quarkonium state of a b and $\bar{b}$ quark, called bottomium. A $\Upsilon(4S)$ decays into an entangled pair of a B and a $\bar{B}$ meson in over 96 % of the cases which leads to the name *B factory* for such colliders. The logo of the Belle II collaboration shown in Figure 1.1 depicts these important facts: electron and positron have different energies, and the main purpose is the production of B mesons. The Belle II detector features a vertex detector, a drift chamber, a time-of-propagation counter, an aerogel ring-imaging cherenkov detector, an electromagnetic calorimeter, and systems to detect K_L^0 mesons and muons. A superconducting solenoid generates a homogeneous magnetic field of 1.5 T in the inner parts. Figure 1.2 shows a view of the detector.

Multivariate methods are widely used in experimental particle physics to analyse data and more precisely to classify certain data points as belonging to a certain type. They are suited very well for this task given the fact that they are developed for classifing data containing numerous variables—which is the case for measurements done in collider experiments. Usually the classifiers are trained on data obtained through Monte Carlo simulations, which use a model of the corresponding detector and available knowledge about the physical processes. However, neither the model of the detector nor the knowledge about physical properties

Figure 1.1: The logo of the Belle II collaboration [1]. The B is composed of to e's of different sizes, illustrating the fact that Belle II is a B factory built with an asymmetric collider.

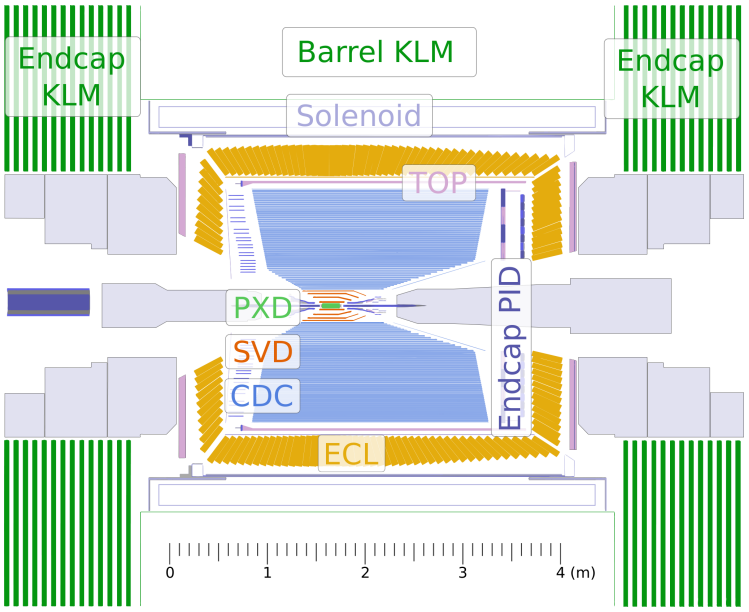


Figure 1.2: Graphical representation of the Belle II detector [2].

like branching ratios is perfect. Thus, the data measured by the experiment can differ from the simulated data on which the classifiers were trained—this means the performance of the classifiers is not optimised for real data. An example for the mis-modelling in Monte Carlo are interferences of particles in multi-body decays. One problem resides on the theory side and is that hadronisation is not yet fully understood but described with a variety of phenomenological models. On the other hand, the toolchain used to produce Monte Carlo events is not yet able to realise all sorts of interferences observed in real events: For this to work, the tools used at different stages of event generation would need to become more coordinated than it is possible at the moment for example with EvtGen and Pythia. For cases where the differences between Monte Carlo and real data are significant, techniques have been developed that account for these and use real data to improve the Monte Carlo data or that train solely on real data [3].

Within the scope of this Thesis, data-driven training based on the sPlot formalism was implemented for the Belle II Analysis Software Framework (BASF2). BASF2 is the software framework of the Belle II experiment. sPlot is a statistical tool to unfold data distributions [4] and the per-event weights it calculates can be used in a training. Firstly, in Chapter 2, various methods of data-driven training of multivariate classifiers are presented, the sPlot formalism is introduced and an algorithm is described that trains a classifier using the sPlot weights. Some properties of the weights that are important for a training are highlighted, as well. In Chapter 3, the implementation of the new module and its usage are described in detail. This Chapter might be the most relevant one for users of BASF2 that already used sPlot in another context. In Chapter 4, some applications of sPlot in past analyses are briefly discussed. Then, the application of the new module is shown on the basis of a simple example analysis. The performance of the classifier trained using sPlot is compared with a classifier trained with Monte Carlo data. In Chapter 5, the results of the work done in the scope of this Thesis are summarised and some suggestions for the further development of the module and data-driven training in general are presented.

2. Data-driven Training Methods for Multivariate Classifiers

Several methods have been developed to improve the training of multivariate classifiers using real data—the issues of simulated data were described in the introduction. In this chapter, a selection of these methods is presented, finally leading to the sPlot formalism which can be used to perform a training solely on real data. As a preparation, the concept of multivariate classification is introduced and all related terms needed for the understanding of the following chapters are explained.

2.1 Multivariate Classification

The task of multivariate classification is to assign a class to a multidimensional vector $\boldsymbol{x}$. In particle physics, this is typically used to classify an event measured in a detector as one of the two classes *signal* and *background*—signal denoting an event that matches a certain hypothesis and background denoting all other events. More formally, the classification is a mathematical function mapping a multidimensional vector $\boldsymbol{x} \in \mathbb{R}^n$ to a one-dimensional value:

$$\begin{aligned} f : \mathbb{R}^n &\to \mathbb{R}\,, \\ \boldsymbol{x} &\mapsto y\,. \end{aligned} \tag{2.1}$$

The classifier output y is also called test-statistic. Usually, it is normalised to 1. Under certain conditions, this makes the interpretation as a probability possible. For the actual classification a critical value T_c has to be chosen that discriminates between the classes—a classifier output below this value is interpreted as decision for background, an output above it as signal. In a well-trained classifier, the mapping f is built in a such way that a loss function

$$\mathrm{err}\left(\{(\boldsymbol{x}_i, w_i, t_i)\}\right) = \sum_i w_i\, g\left(y_i, t_i\right) \qquad w_i, t_i \in \mathbb{R}, g : \mathbb{R}^2 \to \mathbb{R}, y_i = f(\boldsymbol{x_i}) \tag{2.2}$$

calculated on the basis of all training data points $(\boldsymbol{x}_i, w_i, t_i)$ is minimal. Each vector $\boldsymbol{x}_i$ represents one event and is composed of the numerous variables measured by the detector. Each event is given to the training as a sample for class t_i with weight w_i. Normally, $t_i = 1$ is used for signal and $t_i = 0$ or $t_i = -1$ is used for background. The function g rates the difference of the target value t_i and the output y_i of the classifier—this can for example be done with the quadratic difference.

The per-event weights w_i that are set to 1 in a standard training, can be used to change the influence of events in the training: For the minimization of the loss function it is favourable to optimize the classification of events that have greater positive weights than other events, as the function g then evaluates to a small value. When applied to a data sample, the classifier will perform better for events that are similar to these events. Negative weights have a different effect: An event with a negative weight can be used to create a large negative term within the loss function by optimizing the classifier output for this event to be as far from the target value as possible. If the training sample is dominated by negative weights this can lead to the loss function having no lower bound—what this implies for the training depends on the particular multivariate method. However, when used reasonably, negative weights can compensate effects of other events with positive weights in the training. In the case of side-band subtraction and sPlot, the effect of negative weights is well-defined and useful for data-driven training. This is discussed in more detail in the according sections of this chapter and in the case of sPlot further illustrated in Section 4.2.

There are different types of multivariate classifiers, two of them are neural networks and boosted decision trees. Each of them is based on its own training technique, which is used to adjust their internal parameters based on the training data. The details are not described here. An overview can be found in [5] and [6]. During training the classifiers constantly evaluate their quality to prevent *overtraining*. A classifier is termed over-trained if it learns statistical fluctuations and does not generalize the data—this can lead to a very poor performance on an independent data sample. Thus, to obtain this independent data sample the training sample is split into two samples of equal size—one of them is actually used for the training and the other one serves as an independen test sample. Statistical tests like the Kolmogorow-Smirnow test can be used to quantify the difference of the classifier output distribution on the training and the test sample. Ideally, the distributions of the classifier output are not different on the training and the test sample.

The discriminating performance of a classifier can be evaluated through many different plots. Two types of plots are presented hereafter that are produced on the basis of the training and test samples. Firstly, the distributions of the classifier output for signal and background events can be compared. An example is Figure 4.5 in Section 4.2. If the ratio of signal and background events per bin increases monotonously from left to right this is a sign for a well-trained classifier. The second possibility are *receiver operating characteristic curves*, hereafter referred to by *ROC curves*. ROC curves are a whole family of plots, but the most interesting ones for multivariate classification display *signal purity* or *background rejection* over *signal efficiency*. These three quantities are calculated by integrating over certain parts of the classifier output distributions and depend on the cut value T_c. The ROC curves are built pointwise by calculating these quantities for the whole interval of T_c. An example is Figure 4.6 in Section 4.2. Signal efficiency is the fraction of all signal events in the sample that are classified as signal:

$$\text{signal efficiency} = \int_{T_c}^{\infty} f(\boldsymbol{x}|t_1)\,\mathrm{d}y\,. \tag{2.3}$$

As mentioned in the beginning, the classifier output is normalised. Here, $f(\boldsymbol{x}|t_1)$

is the classifier output for signal events—events for which hypothesis t_1 is true, and $f(\boldsymbol{x}|t_0)$ is the classifier output for background events. These two distributions are normalised respectively. Signal purity is the fraction of all events classified as signal that are actually signal events and thus classified correctly:

$$\text{signal purity} = \frac{N_{\text{sig}} \int_{T_c}^{\infty} f(\boldsymbol{x}|t_1)\, \mathrm{d}y}{N_{\text{sig}} \int_{T_c}^{\infty} f(\boldsymbol{x}|t_1)\, \mathrm{d}y + N_{\text{bkg}} \int_{T_c}^{\infty} f(\boldsymbol{x}|t_0)\, \mathrm{d}y}\,. \tag{2.4}$$

As in this Equation, both the signal and the background distribution are used, the actual numbers of signal event N_{sig} and background events N_{bkg} do not cancel. A low signal purity means that the classifier mis-classifies more background events as signal than it classifies signal events as signal. Background rejection is the fraction of all background events that are classified as background:

$$\text{background rejection} = \int_{-\infty}^{T_c} f(\boldsymbol{x}|t_0)\, \mathrm{d}y\,. \tag{2.5}$$

The larger the integral of such a ROC curve is, the better is the discriminating performance of the classifier. Ideally, signal efficiency, signal purity and background rejection are all 1—then the integral is maximal and 1, as well. If the separation of signal and background is not ideal, a good trade-off for the cut value T_c has to be found. This can be done on the basis of the ROC curves: Each point on the curve is a pair of signal purity and signal efficiency or background rejection and signal efficiency, and corresponds to one setting of T_c.

This overview of multivariate classification is based on [3], [5] and [6].

The terms *signal* and *background* that were already used and that will be used in the following chapters, can occur in two slightly different contexts: on the level of events and on the level of candidates. In both cases they are a label for *satisfying* or *not satisfying* a certain hypothesis. For example in the Belle experiment, different classes of events can be of interest: events in which a certain resonance of the Υ meson was produced, like the $\Upsilon(4S)$ or the $\Upsilon(5S)$, events with two high-energetic photons or continuum events $e^+e^- \to q\bar{q}$ with $q \in \{c, s, u, d\}$. Events that are of interest for an analysis are called signal and the others background. The term *candidate* is used in the context of the reconstruction of decay channels in an event. During reconstruction, the goal is to identify particles in the measured data and to assign them to a mother particle and a decay channel. A candidate that complies with all conditions of a hypothesis, like *is an electron* or like *is a* D^0, is called a signal candidate. In the following chapters, *event* and *candidate* are used synonymously as those two concepts are not different regarding the discussed analysis methods.

2.2 Reweighting of Monte Carlo Data

Existing Monte Carlo (MC) data can be reweighted to account for differences in data [3]. This way, the Monte Carlo data which was possibly expensive to generate can still be used. The events are then used with a certain weight when creating histograms, or when used in a training as discussed in the previous section. When

creating histograms, the corresponding bin is not incremented by one for each event but by the weight the event has.

A basic approach to compensate for differences between Monte Carlo and real data is the usage of *scale factors* [3]. By comparing histograms of a certain variable of Monte Carlo and real data the bins can be found in which they differ. For each bin a different scale factor in $[0, \infty)$ can be introduced to scale the Monte Carlo bin to the height of the real data bin. These scale factors can be used as weights in a training, as well. However, correcting the distribution of one variable can degrade the agreement of Monte Carlo and real data in another variable if there are correlations between these variables. This kind of correlation can be accounted for by using histograms with two or more dimensions when comparing Monte Carlo and real data. Still, this is not satisfying because with an increasing number of dimensions, the needed computation resources increase as well, and the statistics per bin decreases.

A method that accounts for the correlations between all variables at once and calculates weights event by event is the following. A multivariate classifier can be used to discriminate Monte Carlo and real data events and its output be transformed to a per-event weight [3]. It is trained with MC data as background and real data as signal. If the classifier does not learn anything to discriminate these two classes, the agreement of MC and data is perfect. MC events which are classified as MC are unlikely to appear in real data and thus their influence needs to be suppressed by using a small weight. MC events which are classified as real data are likely to appear in real data and thus the influence of MC events that are similar to them needs to be raised with a large weight. After the training, the classifier is applied to all events in the MC data sample. Given the classifier output y_i for MC event i, the corresponding weight $w_i \in [0, \infty)$ can be derived with

$$w_i = \frac{1 + y_i}{1 - y_i} \quad \text{for} \quad y_i \in [-1, 1] \quad \text{or} \tag{2.6}$$

$$w_i = \frac{y_i}{1 - y_i} \quad \text{for} \quad y_i \in [0, 1]\,. \tag{2.7}$$

Like the scale factors, this technique cannot compensate for regions in phase space where no MC events exist but real data events do. Another problem is that the classifier is not aware of the class to which an event belongs. This can lead to a situation where signal events from MC are reweighted to match background events from data and vice versa.

2.3 Side-band Subtraction

Knowing the issues of the presented reweighting techniques for MC, these problems would disappear if multivariate classifiers could be trained solely on real data. This is not directly possible as there are usually no pure samples of signal and background events available. If however in the distribution of a variable a signal peak is visible over some background, a *side-band subtracted* training can be performed [3].

For this training, different regions in the distribution of the *discriminating variable* need to be defined to determine which events are included as samples for signal

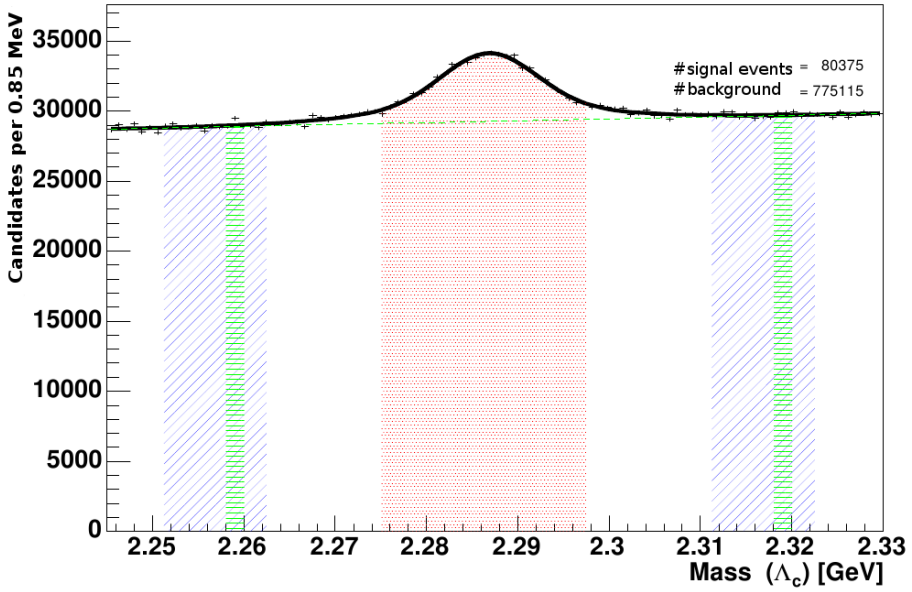


Figure 2.1: Mass distribution of Λ_c candidates. Red is the signal region, blue and green are the side-bands [3, 7].

Table 2.1: Input for a side-band subtracted training based on the distribution shown in Figure 2.1. Taken from [3].

region	number of events	target	weight
red	$N_{\text{sig}} + N_{\text{bkg}}$	signal	+1
blue	N_{bkg}	signal	-1
green	N_{sig}	background	+1

and background. Figure 2.1 shows an example, where the invariant mass M is used as discriminating variable. In principle, multiple discriminating variables can be used to determine the regions—in the case of two, a two-dimensional histogram would be used. The red region is used in the training as signal with weight 1. Based on the side-band it has to be estimated how many signal events N_{sig} and background events N_{bkg} are included within this region. This can for example be done by fitting a model to the data. Then, the size of the blue region has to be chosen such that it consists of N_{bkg} events. It is used in the training as signal with weight -1. This has the effect that the background events from the red region with weight 1 are statistically subtracted which leads to a signal sample that statistically only consists of signal events. The green region is chosen in a size such that it contains N_{sig} events. These are used in the training as background with weight 1. The reason for chosing the background sample in the same size as the signal sample is that this improves the training quality. The training input is summarised in Table 2.1.

The statistical subtraction of background events only works if the background behaves the same across the used range of the discriminating variable. This does not mean that the density of background events has to be the same across the range but that the variables used in the training have to be independent of the discriminating variable. As a consequence, the discriminating variable itself can-

not be used; it is only used to prepare the training sample. Stated differently, the variables used in the training must not contain any information about the discriminating variable, so the classifier cannot predict the value of the discriminating variable from the knowledge of the other variables. As a check of this independence, the training of such a classifier can actually be performed. The quality of this classifier's prediction is a direct measure of the dependency of the variables and the discriminating variable. If variables depending on the discriminating variable are used in the side-band subtracted training, the classifier can learn from these variables in which region the event is located: The discriminating variable provides an explicit assignment to signal and background in this setting of the training, namely the signal peak and the green region, which do not overlap. The classifier will rate any variable depending on it as very significant and classify only based on it. This is not representative for an independent data sample and thus the classifier would perform very poorly. If no variables can be found that are independent of the discriminating variable in the used range, side-band subtracted training cannot be used.

2.4 The sPlot Technique

The sPlot technique can be used as an advanced side-band subtraction. It does not require a clean background region, but only measurable differences between the distributions of the classes [3]. With sPlot, each event is assigned a weight for each class. sPlot does this based on a model that is fitted to the data with an extended maximum likelihood fit and by using the properties of the covariance matrix of this fit. The advantage over side-band subtraction is that there are no regions to be defined. Everything is derived from the model and thus the data on the whole range can be used—given that there are no satellite resonances present within the range.

sPlot[1] was introduced in [4] with the subtitle "A statistical tool to unfold data distributions". This means that based on the model of some discriminating variables, the distributions of other variables can be reconstructed for each class. These other variables are called *control variables* hereafter. The discriminating variable can for example be the invariant mass, as in the example shown in Figure 2.1. A more descriptive example is if the signal and background classes can be unambiguously separated based on some discriminating variables. In this case, the distributions of control variables for signal and background can be determined by histogramming these variables for signal and background events separately. sPlot makes it possible to produce these plots even if it is not clear for all events to which class they belong while accounting for the correlation between the classes. The sPlots are originally thought of as a means of testing the accuracy of the model of the discriminating variables; the agreement of the reconstructed distributions of the control variables with what is expected based on other knowledge is a sign of quality for the model. This can be used as a mutual check of the discriminating variables: One discriminating variable at a time can be excluded from the set

[1] According to the authors, the s in sPlot originates from the usage of the covariance in the calculation of the weights. The covariance is related to the variance which is in turn sometimes denoted with s^2—additionally, the pronounciation of the symbol σ^2 for the covariance starts with an s.

of discriminating variables and its distribution reconstructed with sPlot to see if the data agrees with the model. sPlot calculates errors for the bins of the reconstructed histograms, as well. Thus, it can be seen how exact the reconstrution is possible on the basis of the discriminating variables.

In this section, the sPlot formalism is derived based on the original sPlot paper [4] and mostly follows their notation for better comparability. Some facts relevant for the use of the weights in a training are added to the discussion. Firstly, the method of extended maximum likelihood fit is explained. Then, in Section 2.4.1, a naive but very straightforward method for the calculation of weights is discussed. It is called inPlot and leads to the definition of sPlot weights, which are presented in Section 2.4.2. Section 2.4.3 shows some interesting properties of the weights and compares inPlot and sPlot weights on the basis of an example. In the last section, the complete algorithm for an sPlot-based training is described based on [3].

The method of extended maximum likelihood (EML) is widely used in particle physics to estimate parameters of models for a certain data sample [8]. Usually, a data sample consists of events that belong to different classes, like signal and background; and the model to be fitted describes the distribution of events in a certain variable. This means, in the case of a histogram, that it describes how many events of each class are present in each bin. A model can generally look like this for the discussed problems:

$$\sum_{n=1}^{N_\mathrm{s}} N_n f_n(y)\,, \tag{2.8}$$

where N_s is the number of classes present in the sample, N_n is the *yield* of class n and $f_n(y)$ is the *probability density function (PDF)* of the discriminating variables y for class n. The PDFs are normalised, which means that the yields N_n are the expected number of events for class n. Again, usually N_s is 2, with the classes being signal and background. The parameters of the f_n and the yields N_n are estimated with an EML fit by maximising the extended log-likelihood function

$$\mathcal{L} = \sum_{e=1}^{N} \log\left(\sum_{n=1}^{N_\mathrm{s}} N_n f_n(y_e)\right) - \sum_{n=1}^{N_\mathrm{s}} N_n\,, \tag{2.9}$$

where N is the total number of measured events in the data sample and y_e is the value of the discriminating variables for event e. By maximising $\mathcal{L}$ the set of model parameters is found which is most likely to produce the given data sample. The value of the last sum over all yields is the number of events that the fit actually estimates and can be different from the number of observed events N. This term is also the difference to the standard maximum likelihood fit. In EML, the number of estimated events can differ from the number of actually observed events and thus, it has one extra degree of freedom. In particle physics experiments where the number of events that are measured in each class can fluctuate in line with the Poisson distribution, this degree of freedom is important and leads to better fit results [3].

2.4.1 inPlot as a First Step

As already stated in the introduction of this section, sPlot can be used to reconstruct the true distributions $\boldsymbol{M}_n(x)$ of control variables x for different classes n

from the knowledge of the PDFs $f_n(y)$ of the discriminating variables y of these classes. While the derivation is done for an arbitrary number of classes, this can always be reduced to the two classes signal and background for a better understanding.

As a first step it is assumed that the control variables depend on the discriminating variables in such a way that x is a function of y: $x = x(y)$. This is not a case in which x could actually be used as a control variable for y; Since the knowledge about the distribution of x is implicitly used when constructing the histograms of x, they cannot serve as independent means of evaluating the model for y. A descriptive example is given after the derivation of the inPlot weights.

Under the assumption that the parameters of the PDFs $f_n(y)$ and the yields N_n are determined by a fit as explained above, weights can be defined for each event and each class by

$$\mathcal{P}_n(y_e) = \frac{N_n f_n(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} \,. \tag{2.10}$$

This simply is the ratio of events for the value y_e of the discriminating variables that belong to class n. Stated differently, the weight is the probability to choose an event of class n when choosing from events with the value y_e in the discriminating variables. From Equation 2.10 it can directly be derived that the $\mathcal{P}_n$ are all in $[0, 1]$ and that for each event e, the weights sum up to 1:

$$\sum_{k=1}^{N_s} \mathcal{P}_n(y_e) = 1 \;\forall \text{ events } e \,. \tag{2.11}$$

These weights can now be used to build the reconstructed x distributions $\tilde{M}_n(x)$ for class n bin per bin:

$$N_n \int_{B_{\delta x}(\bar{x})} \tilde{M}_n(x)\, \mathrm{d}x \equiv \sum_{e:\, x_e \in B_{\delta x}(\bar{x})} \mathcal{P}_n(y_e) \,, \tag{2.12}$$

where $B_{\delta x}(\bar{x})$ is the x bin centered on $\bar{x}$ with an edge length of δx. The sum is over all events e in the bin. *All events* of the data sample are used to build these histograms, but with the weight belonging to the class for which the histogram is built. Because of Equation 2.11 each event is only used with total weight 1, as in a standard histogram or training. To prove that $\tilde{M}_n(x)$ reproduces the true distribution $\boldsymbol{M}_n(x)$, a continuous form of Equation 2.12 in the limit of many events and $\delta x \to 0$ is built. For this, the sum over all events that belong to the bin is turned into an integral over the whole range of the discriminating variable y. The delta distribution is used to select just those parts of y that belong to the x value $\bar{x}$ that is currently being evaluated:

$$\sum_{e:\, x_e \in B_{\delta x}(\bar{x})} 1 \;\to\; \int \sum_{j=1}^{N_s} N_j f_j(y)\, \delta(x(y) - \bar{x})\, \mathrm{d}y \,. \tag{2.13}$$

Like the sum over 1, this calculates the number of events with x value $\bar{x}$, including all classes. Multiplying with the weight $\mathcal{P}_n(y)$ leads to the continuous form of

Equation 2.12. Angle brackets are used to denote that this is the expectation value of the reconstructed distribution.

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = \int \sum_{j=1}^{N_s} N_j f_j(y)\, \delta(x(y) - \bar{x})\, \mathcal{P}_n(y)\, \mathrm{d}y\,. \tag{2.14}$$

Inserting the definition of the weights gives

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = \int \sum_{j=1}^{N_s} N_j f_j(y)\, \delta(x(y) - \bar{x})\, \frac{N_n f_n(y)}{\sum_{k=1}^{N_s} N_k f_k(y)}\, \mathrm{d}y\,. \tag{2.15}$$

The two sums over all components of the model cancel:

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = N_n \int \delta(x(y) - \bar{x})\, f_n(y)\, \mathrm{d}y\,. \tag{2.16}$$

$$\equiv N_n \boldsymbol{M}_n(\bar{x})\,. \tag{2.17}$$

The last step can be phrased as follows: As x and y are dependent variables, the probability density of a certain $\bar{x}$ in a class is the sum of the probability densities of the y values in the same class that have $\bar{x}$ as function value $x(y) = \bar{x}$. Or, even more descriptive: A value $\bar{x}$ occurs as often in a class as the corresponding y values do in the same class.

This shows that the weights defined in 2.10 can be used to reconstruct the distribution of control variables if the control variables depend on the discriminating variables. Because the control variables in this case basically belong to the set of discriminating variables, the plots obtained using these weights are called inPlots. As already mentioned, the inPlots cannot be used to evaluate the quality of the model used for the discriminating variables. Because x and y are dependent, by defining a model for y, a model for x is implicitly defined. Thus, the $\mathcal{P}_n$ re-weight the events to be in line with the implicitly defined model for x. The following can be considered as an example: A model for y is used, where in a certain region of y, no events at all are expected to be measured. This implicitly defines, that in certain regions of x, smaller numbers of events or no events are expected, too. If however, there *are* events showing up in this region in a measurement, a small weight is assigned to them according to the model, or even the weight 0. Then, the reconstructed histogram of x will look like the implicitly defined one and the mis-modeling of y cannot be detected. Thus, the inPlots are systematically biased.

2.4.2 The sPlot Formalism

Based on inPlot, weights can be derived for the more interesting case where x is not correlated with y and thus can be used as a control variable. In the case of uncorrelated x and y, there is no way of looking up the x value for a certain y. Because of this, taking Equation 2.14 as a starting point, the integral needs to be

formulated differently:

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = \int \int \sum_{j=1}^{N_s} N_j f_j(y)\, \boldsymbol{M}_j(x)\, \delta(x-\bar{x})\, \mathcal{P}_n(y)\, \mathrm{d}y\, \mathrm{d}x \tag{2.18}$$

$$= \int \sum_{j=1}^{N_s} N_j f_j(y)\, \boldsymbol{M}_j(\bar{x})\, \mathcal{P}_n(y)\, \mathrm{d}y\,. \tag{2.19}$$

Like in Equation 2.14, when omitting the $\mathcal{P}_n(y)$, this would evaluate to the total number of events expected for the x value $\bar{x}$. As there is no direct connection between x and y values, an integration over the whole phase space is needed. In Equation 2.14, there are some discrete y values that contribute with their probability density to the probability density of $\bar{x}$. Here however, as there is no correlation, there is a probability for *all* y values that they occur together with $\bar{x}$. This probability is different in each class j. It is the product of the probability that $\bar{x}$ occurs in this class, $\boldsymbol{M}_j$, and the expected number of events for y in this class: $N_j f_j(y)$. The inPlot weights are used to reduce the total number of events to the number expected for class n:

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = \int \sum_{j=1}^{N_s} N_j f_j(y)\, \boldsymbol{M}_j(\bar{x})\, \frac{N_n f_n(y)}{\sum_{k=1}^{N_s} N_k f_k(y)}\, \mathrm{d}y\,. \tag{2.20}$$

The two sums over all components of the model now do not cancel because of the contribution of $\boldsymbol{M}_j(\bar{x})$ in each class. Thus, in comparision to Equation 2.17, there is a correction term remaining:

$$\langle N_n \tilde{M}_n(\bar{x}) \rangle = N_n \sum_{j=1}^{N_s} \boldsymbol{M}_j(\bar{x}) \left(N_j \int \frac{f_n(y) f_j(y)}{\sum_{k=1}^{N_s} N_k f_k(y)}\, \mathrm{d}y \right) \tag{2.21}$$

$$\neq N_n \boldsymbol{M}_n(\bar{x})\,.$$

This cannot be identified with the true distribution $\boldsymbol{M}_n(\bar{x})$ but is a linear combination of all true distributions. The correction term corresponds to the inverse of the covariance matrix that results from the extended maximum likelihood fit as a second derivative of $-\mathcal{L}$ [8]:

$$\boldsymbol{V}_{nj}^{-1} = \frac{\partial^2(-\mathcal{L})}{\partial N_n \partial N_j} = \sum_{e=1}^{N} \frac{f_n(y_e) f_j(y_e)}{\left(\sum_{k=1}^{N_s} N_k f_k(y_e)\right)^2}\,. \tag{2.22}$$

The sum over all events can be transformed into the continuous form like in Equation 2.19, just that no δ distribution is needed because the covariance matrix depends on all x values:

$$\langle \boldsymbol{V}_{nj}^{-1} \rangle = \int \int \sum_{l=1}^{N_s} N_l f_l(y)\, \boldsymbol{M}_l(x)\, \frac{f_n(y_e) f_j(y_e)}{\left(\sum_{k=1}^{N_s} N_k f_k(y_e)\right)^2}\, \mathrm{d}y\, \mathrm{d}x \tag{2.23}$$

$$= \int \sum_{l=1}^{N_s} N_l f_l(y)\, \frac{f_n(y_e) f_j(y_e)}{\left(\sum_{k=1}^{N_s} N_k f_k(y_e)\right)^2}\, \mathrm{d}y \underbrace{\int \boldsymbol{M}_l(x)\, \mathrm{d}x}_{=1} \tag{2.24}$$

$$= \int \frac{f_n(y_e) f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)}\, \mathrm{d}y\,. \tag{2.25}$$

Using this in Equation 2.21 leads to

$$\langle \tilde{M}_n(\bar{x}) \rangle = \sum_{j=1}^{N_s} \boldsymbol{M}_j(\bar{x}) N_j \langle \boldsymbol{V}_{nj}^{-1} \rangle\,. \tag{2.26}$$

This matrix equation can be inverted and solved for the true distribution

$$N_n \boldsymbol{M}_n(\bar{x}) = \sum_{j=1}^{N_s} \langle \boldsymbol{V}_{nj} \rangle \langle \tilde{M}_j(\bar{x}) \rangle\,. \tag{2.27}$$

This means that the true x distribution in a class n is the linear combination of *all* distributions reconstructed with inPlot weights, and the coefficients are the covariances of the yields of the classes. This can now be used to build the sPlot-reconstructed x distribution ${}_s\tilde{M}_n$ of class n, bin per bin, analogous to Equation 2.12:

$$N_n \int_{B_{\delta x}(\bar{x})} {}_s\tilde{M}_n(\bar{x})\, \mathrm{d}x = \sum_{j=1}^{N_s} \boldsymbol{V}_{nj} \int_{B_{\delta x}(\bar{x})} \tilde{M}_n(\bar{x})\, \mathrm{d}x \tag{2.28}$$

$$= \sum_{j=1}^{N_s} \boldsymbol{V}_{nj} \sum_{e:\, x_e \in B_{\delta x}(\bar{x})} \frac{f_n(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} \tag{2.29}$$

$$= \sum_{e:\, x_e \in B_{\delta x}(\bar{x})} \sum_{j=1}^{N_s} \boldsymbol{V}_{nj} \frac{f_n(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)}\,, \tag{2.30}$$

and finally

$$N_n \int_{B_{\delta x}(\bar{x})} {}_s\tilde{M}_n(\bar{x})\, \mathrm{d}x = \sum_{e:\, x_e \in B_{\delta x}(\bar{x})} {}_s\mathcal{P}_n(y_e)\,, \tag{2.31}$$

with

$${}_s\mathcal{P}_n(y_e) = \frac{\sum_{j=1}^{N_s} \boldsymbol{V}_{nj} f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)}\,. \tag{2.32}$$

To conclude, in the case of uncorrelated variables x and y, the x distributions can be reconstructed using the sWeights ${}_s\mathcal{P}_n$ as defined in Equations 2.31 and 2.32. The weights can be calculated solely based on data. The covariance results directly from the extended maximum likelihood fit. However, according to the authors of [4], using it is numerically less accurate than computing it via the sum of events. In fact, the implementation of sPlot in `RooStats` uses Equation 2.22 to calculate the inverse covariance matrix and then inverts it.

2.4.3 Properties of the Weights

The sWeights have some interesting properties that are also satisfied in non-asymptotic conditions, because they result from the properties of the extended maximum likelihood fit and the properties of the covariance matrix. The properties are derived using three sum rules that are introduced below. If one is interested in the properties only, these definitions can be skipped safely.

1. Maximum Likelihood Sum Rule

   Calculating the partial derivative of Equation 2.9 with respect to a yield variable N_j leads to the following result, because $\mathcal{L}$ is extremal in this variable after the fit:

   $$\sum_{e=1}^{N} \frac{f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} = 1 \quad \forall j\,. \tag{2.33}$$

2. Variance Matrix Sum Rule

   From Equations 2.22 and 2.33 it can be derived that

   $$\begin{aligned}\sum_{i=1}^{N_s} N_i \boldsymbol{V}_{ij}^{-1} &= \sum_{i=1}^{N_s} N_i \sum_{e=1}^{N} \frac{f_i(y_e) f_j(y_e)}{\left(\sum_{k=1}^{N_s} N_k f_k(y_e)\right)^2} \\ &= \sum_{e=1}^{N} \frac{f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} \overset{2.33}{=} 1 \quad \forall j\,.\end{aligned} \tag{2.34}$$

3. Covariance Matrix Sum Rule

   The sum over a row or a column l of the covariance matrix equals the yield N_l and thus the number of expected events for this class:

   $$\begin{aligned}\sum_{j=1}^{N_s} \boldsymbol{V}_{jl} &= \sum_{j=1}^{N_s} \boldsymbol{V}_{jl} \underbrace{\sum_{i=1}^{N_s} N_i \boldsymbol{V}_{ij}^{-1}}_{2.34} = \sum_{i=1}^{N_s} \left(\sum_{j=1}^{N_s} \boldsymbol{V}_{ij}^{-1} \boldsymbol{V}_{jl} \right) N_i \\ &= \sum_{i=1}^{N_s} \delta_{ij} N_i = N_l \quad \forall l\,.\end{aligned} \tag{2.35}$$

   From this rule, the equivalence of inPlot and sPlot weights can be deduced for the case that the discriminating variables are totally discriminating.

Then, as all classes are separated in y, the covariance matrix has the yields on the diagonal elements and zeros everywhere else. As a result, in the ${}_s\mathcal{P}_n$ defined in Equation 2.32, only $\boldsymbol{V}_{nn} f_n$ contribute. With $\boldsymbol{V}_{nn} = N_n$ in this case, this reduces to the definition of $\mathcal{P}_n$ in Equation 2.10.

This sum rule is related to the fact that the non-diagonal elements of the covariance are ≤ 0 because the model consists of probability density functions—PDFs are always ≥ 0 and if one component is decreased, another one has to be increased to compensate. For the weights this means generally that they become smaller if there is a correlation with another class—except this class has no contribution at the given y position.

2.4.3.1 Normalization

From these three sum rules it follows that each reconstructed x distribution is normalised:

$$\begin{aligned} N_n \int {}_s\tilde{M}_n(x)\,\mathrm{d}x &= \sum_{e=1}^{N} {}_s\mathcal{P}_n(y_e) = \sum_{e=1}^{N} \frac{\sum_{j=1}^{N_s} \boldsymbol{V}_{nj} f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} \\ &\overset{2.33}{=} \sum_{j=1}^{N_s} \boldsymbol{V}_{nj} \overset{2.35}{=} N_n \quad \forall n\,. \end{aligned} \tag{2.36}$$

Furthermore, the weights assigned to an event sum up to 1:

$$\sum_{l=1}^{N_s} {}_s\mathcal{P}_l(y_e) = \sum_{l=1}^{N_s} \frac{\sum_{j=1}^{N_s} \boldsymbol{V}_{lj} f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} \overset{2.35}{=} \frac{\sum_{j=1}^{N_s} N_j f_j(y_e)}{\sum_{k=1}^{N_s} N_k f_k(y_e)} = 1 \quad \forall e\,. \tag{2.37}$$

This means that the overall sPlot-reconstructed x distribution contains as many events in each bin as were actually observed. Stated differently, the distributions obtained with sPlot represent consistently how x is distributed in the different classes.

2.4.3.2 Statistical Uncertainties

Based on the sWeights, statistical uncertainties for each bin in the reconstructed distributions can be calculated:

$$\sigma\left[N_n \int_{B_{\delta x}(\bar{x})} {}_s\tilde{M}_n(\bar{x})\,\mathrm{d}x\right] = \sqrt{\sum_{e:\, x_e \in B_{\delta x}(\bar{x})} \left({}_s\mathcal{P}_n(y_e)\right)^2}\,. \tag{2.38}$$

In fact, the whole covariance matrix can be reproduced with the sWeights:

$$\sum_{e=1}^{N} ({}_s\mathcal{P}_i(y_e))({}_s\mathcal{P}_j(y_e)) = \boldsymbol{V}_{ij}\,. \tag{2.39}$$

The derivation can be found in [4].

2.4.3.3 Merging of sPlots

One or more classes i and j can be merged into a single class $(i + j)$ without the need to perform the fit or to compute the sWeights again. The weights for the combined class are the sum of the separate weights:

$$ {}_s\mathcal{P}_{(i+j)}(y_e) = {}_s\mathcal{P}_i(y_e) + {}_s\mathcal{P}_j(y_e) \quad \forall e\,. \tag{2.40} $$

All properties including the normalisation and the statistical uncertainties are still valid, this is derived in [4]. This property is useful for example if the model contains multiple types of background events that need to be fitted separately but are used together in a training.

2.4.4 Algorithm for the Training of Multivariate Classifiers

The use of weights calculated by sPlot in a training of multivariate classifiers is discussed in [3]. With sPlot, each event gets assigned a weight for each class and these weights depend on how much this event contributes to the distribution of the class. If each event is used in the training for each class but with the corresponding weight, the classifier will learn that certain regions in phase space are more significant for one class than another—this information is encoded in the weights. The classifier will optimize its separation power without that the truth for any event is known. This implies that such a classifier cannot perform better on Monte Carlo data compared to a classifier that was trained in the standard way. On real data, the classifier trained using sPlot can perform better depending on how significant the mis-modeling in Monte Carlo is. This results in the following algorithm:

1. An extended maximum likelihood fit is performed to find the parameters of the probability density functions f_i in the model and the yields N_i of the different classes. The fit relies on the discriminating variables y but not on the control variables x because x und y are uncorrelated.
2. A second fit is performed where all parameters but the yields N_i are frozen. This is proposed by the authors of [4] to improve the estimation.
3. The sWeights ${}_s\mathcal{P}_i$ are calculated with Equation 2.32. The covariance matrix is calculated by inverting the inverse covariance matrix given in Equation 2.22.
4. Each event is used in the training for every class with the corresponding weight. Usually, this means using the event twice, once for signal and once for background. Care has to be taken when choosing the classifier and its options: Negative weights need to be supported and when splitting the sample into training and test sample, the instances of an event should not be separated. Details are discussed in Section 3.3.1 and 3.3.2. The variables x used in the training need to be uncorrelated with the discriminating variables y.

This algorithm was implemented in the scope of this Thesis for the Belle II Analysis Software Framework. For the calculation of the sWeights, the sPlot implementation in the `RooStats` package is used—it implements points 2 and 3 of the algorithm.

For a better understanding of the influence of sWeights in a training, Equation 2.37 is important; the sum of all weights assigned to an event is 1. This means that a

negative weight of an event in a certain class is always complemented by a larger positive weight when combining the weights of all other classes. Thus, negative sWeights do not remove the lower bound of the loss function. Instead, they always emphasize the optimisation of the classifier output for the other classes.

If variables are used in the training that are correlated with the discriminating variables y, this will affect the performance of the classifier as it was already described for the side-band subtracted training in Section 2.3; a variable correlated with y is very significant during the training. The classifier will learn to give this variable a high priority in the decision and perform badly on an independent data sample. In a side-band subtracted training, certain regions are clearly associated with a certain class because of the way the training sample is built and thus, the training is heavily biased when using a variable correlated with the discriminating variable. This effect is more subtle for a training with sWeights, but essentially the same: The weights still define regions of the discriminating variables, that are associated more with one class than another—this becomes visible in Figure 4.4 of the example analysis. This preference for a certain class is derived from the discriminating variables and thus correlated with any variable that is correlated with y. This information is then picked up by the training and leads to a bias in the decision process. For this reason, as with side-band subtracted training, the discriminating variables can only be used to prepare the training sample.

3. Implementation of sPlot-based Training

In this chapter, the implementation of sPlot-based training within the Belle II Analysis Software Framework (BASF2) is described. BASF2 is the software framework of the Belle II experiment. The sPlot-based training is implemented as a new module within BASF2 which is the main work done in the scope of this Thesis.

BASF2 is described shortly in the next Section 3.1. In Section 3.2, the definition of the physical model using RooFit classes is discussed. sPlot's requirement of a model of the discriminating variables is specified in Sections 2.4.2 and 2.4.4. In Section 3.3, the usage of the new module is documented: All parameters are explained in detail, compatible multivariate classifiers are discussed, and some interesting aspects of the implementation are highlighted.

3.1 The Belle II Analysis Software Framework

BASF2 is used in the Belle II experiment for online and offline data handling [5]. It is written in C++ 11 and Python 2.7 and incorporates many libraries that are widely used in particle physics, one of them is the ROOT Data Analysis Framework [9]. BASF2 is organised in *packages* that unite functionality belonging to a common category, like Monte Carlo event generation, detector simulation, track reconstruction, visualisation and physics analysis. The packages in turn contain libraries, modules and data objects. *Modules* are event processing units that perform self-contained tasks. Most of them are written in C++. Multiple modules can be chained together in a *path*, which can be used for different applications like performing an analysis, generating Monte Carlo data or reconstructing tracks. A common *data store* enables the modules to share data. BASF2 is used via a *steering file*, which is a Python file that defines the path and the parameters of all used modules. Internally, the path is translated to C++ using the Boost Python library, so BASF2 profits from the ease of use of Python and the execution speed of C++. Figure 3.1 illustrates how the BASF2 components act together in the processing of the data event by event. An example steering file is shown in Section 4.2, where an analysis using the new sPlot-based module is discussed. BASF2 supports parallelisation by means of multi-processing and distributed computing.

BASF2 provides an interface to ROOT's TMVA library for multivariate analysis (MVA) [10]. Thus, the user can easily access different multivariate classifiers through one configuration interface and use them to separate correctly reconstructed candidates from background. The TMVA interface provides the two

modules `TMVATeacher` and `TMVAExpert`, that train and apply a specified classifier on the events of a data set [5].

3.2 Specification of the Model using RooFit

There are two implementations of the sPlot algorithm available in the ROOT framework: `TSPlot` [11] in the math libraries; and `RooStats::SPlot` [12] from the RooStats project [13], which is built on top of the `RooFit` package [14]. `TSPlot` was implemented alongside with the publication of the original paper introducing sPlot [4]. In this implementation, the model for the discriminating variables cannot be passed as an analytical function. Instead, the user needs to provide a list of explicit PDF values of each discriminating variable for each candidate. `RooStats::SPlot` takes the model as an analytical function defined with RooFit, which is more comfortable, as fits are often performed with RooFit anyway and thus no conversion of the PDF is necessary. For this reason it was decided to use the `RooStats` implementation of sPlot for the development of the new `TMVASPlotTeacher` module for sPlot-based training in BASF2.

RooFit provides a large selection of C++ classes to define probability density functions: `RooGaussian`, `RooExponential`, `RooChebychev`, and many more. Those basic PDFs can be combined using different operators, for example `RooAddPdf` for a linear combination, or `RooProdPdf` for a product of multiple PDFs. Parameters and observables are defined using the class `RooRealVar`. Chapters 3 and 4 of the RooFit Users Manual [15] describe all concepts needed for the understanding of the following examples. Additional guidance can be found in the tutorial macros distributed with ROOT [16] and the ones available on the RooFit website [17], both showing examples from different applications in physics. There is a tutorial for `RooStats::SPlot`, as well [12]. The following listing shows the definition of the model which is later used for the example analysis in Section 4.2. A plot of the model is shown there, in Figure 4.2. It uses `RooAddPdf` to construct a linear combination of the signal and background PDFs, which, for a majority of cases, is the most straightforward approach.

Listing 3.1: Example model for the D^0 mass, using PyROOT.

```
1 # observable
2 M = RooRealVar("M", "M", 1.83, 1.9)
3
4 # component PDFs
5 mD0 = RooRealVar("mD0", "D0 Mass", 1.864)
```

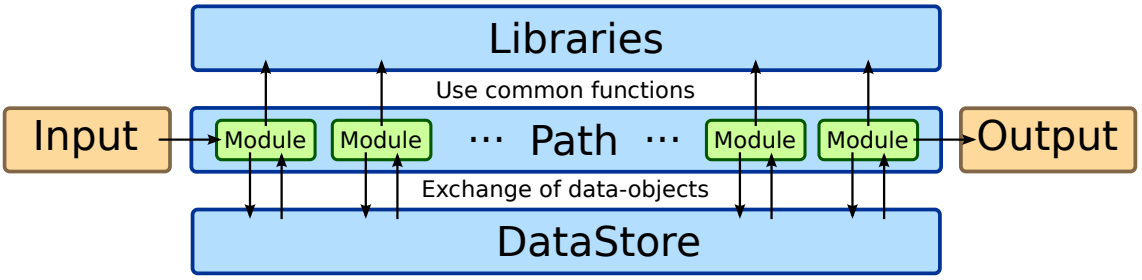


Figure 3.1: Illustration of the event processing chain in BASF2 [5].

```
6  sigmaD0 = RooRealVar("sigmaD0", "Width of Gaussian", 0.00325)
7  sig = RooGaussian("sig", "D0 Model", M, mD0, sigmaD0)
8
9  a0 = RooRealVar("a0", "a0", -0.115)
10 a1 = RooRealVar("a1", "a1", -0.164)
11 bkg = RooChebychev("bkg", "Background", M, RooArgList(a0, a1))
12
13 # coefficients
14 bkgfrac = RooRealVar("bkgfrac", "background frac", 44853)
15 sigfrac = RooRealVar("sigfrac", "signal frac", 18050)
16
17 # linear combination
18 model = RooAddPdf("model", "bkg+sig", RooArgList(bkg, sig), RooArgList(↩
       bkgfrac, sigfrac))
```

The model has to be provided to `TMVASPlotTeacher` via a ROOT file. The filename and the name of the RooFit object inside the file have to be handed over to `TMVASPlotTeacher` via the parameters, which will be described in detail in Section 3.3.1. For the example model, the object name would be `model`. The ROOT file can be produced within the steering file using PyROOT, just before adding a `TMVASPlotTeacher` module to the path. An example is shown in Section 4.2.

The RooFit object defining the model must be derived from `RooAbsPdf`, which is the abstract interface for all probability density functions in RooFit. This way, full flexibility is retained for the user in their definition of the model. However, all PDF operators have to support an extended maximum likelihood fit, as sPlot is built upon this assumption. This was already discussed in Section 2.4.2; the sPlot formalism needs estimates for the yields of all classes. `RooAddPdf` and `RooProdPdf` support extended maximum likelihood fits; if any part of the PDF does not support it, the error message

```
this PDF does not support extended maximum likelihood
```

is shown during the fit. Moreover, `RooStats::SPlot` only calculates weights for a class if the according yield is explicitly declared in the model. In example listing 3.1, the yields are `bkgfrac` for background and `sigfrac` for signal.

The connection between model and data are the discriminating variables. All discriminating variables have to be represented in the model, and there is no limitation on the number of discriminating variables. The ranges of the discriminating variables defined in the model should match the ranges these variables cover in the data, see line 2 in listing 3.1. Otherwise, the fit which is performed inside `TMVASPlotTeacher` could fail. More importantly, for the sWeights to be meaningful, a good choice of the ranges is essential: Discriminating and control variables have to be independent. As a check of this independence, the training of such a classifier can actually be performed, see Section 4.2. The quality of this classifier's prediction is a direct measure of the dependency of control and discriminating variables. These considerations concerning the ranges usually mean that they are chosen closely to the signal peak in a way that the data set contains at least as many background as signal candidates, while not including any satellite resonances.

In a model with multiple independent discriminating variables, the PDF for each class would consist of a PDF for each discriminating variable, multiplied using

`RooProdPdf`:

$$y_\mathrm{s} \cdot \underbrace{f_{\mathrm{s},1}(x_1) f_{\mathrm{s},2}(x_2)}_{\texttt{RooProdPdf}} + y_\mathrm{b} \cdot \underbrace{f_{\mathrm{b},1}(x_1) f_{\mathrm{b},2}(x_2)}_{\texttt{RooProdPdf}} ,$$

where y_s and y_b are the yields for signal and background, and $f_{i,j}(x_j)$ is the PDF for class i in discriminating variable x_j. This is described in more detail in Chapter 6 of the RooFit Users Manual [15]; in the case of correlated discriminating variables a common PDF has to be defined, this is covered in Chapter 7.

The module imposes no limit on the number of classes. Weights are calculated for every class that has a yield defined in the model. If more than two yields are present, one classifier is trained to separate each class from the others respectively; this approach is usually called *one vs. rest*. The sPlot-based training still works in such a case because of the merging property of the sPlots defined in Equation 2.40; the weights of the other classes are virtually summed up to built one new class.

During the run of `TMVASPlotTeacher`, two fits of the model to the data are performed. One of them is invoked directly from the module and the other from the `RooStats::SPlot` class. `SPlot` expects that the PDFs of all classes which are part of the model are already fitted and normalized, and only determines the yields by this fit. To do so, it sets all parameters except the yields explicitly to constant. The first fit inside `TMVASPlotTeacher` fits all parameters to prepare for the fit by `SPlot`. Depending on the structure of the model, it might be necessary to set the parameters explicitly to be *not constant*:

Listing 3.2: Example model for the D^0 mass, additions for the fit.

```
1 mD0.setConstant(kFALSE)
2 sigmaD0.setConstant(kFALSE)
3 a0.setConstant(kFALSE)
4 a1.setConstant(kFALSE)
5 dummy = 42
6 bkgfrac = RooRealVar("bkgfrac", "fraction of background", dummy, 0., dummy↩
      )
7 sigfrac = RooRealVar("sigfrac", "fraction of background", dummy, 0., dummy↩
      )
8 bkgfrac.setConstant(kFALSE);
9 sigfrac.setConstant(kFALSE);
```

Lines 5 and 6 concern the ranges of the yields. If no value for the ranges are provided, `RooAddPdf` assumes the yields to be fractions within the interval of $[0, 1]$ by default. As `SPlot` uses an extended maximum likelihood fit, the yields need a larger range, ideally from 0 to the number of candidates, to cover all possibilities. The number of candidates is not necessarily known by the user of BASF2 when writing the steering file, except they know the number by other means—a user could also want to use one steering file for different data samples. For this reason, `TMVASPlotTeacher` provides a way to set the range and the initial value later depending on the number of candidates, see the explanation of the parameters `setYieldRanges` and `modelYieldsInitialFractions` in Section 3.3.1. In the steering file, the user might then just insert a placeholder value, like it is done with the `dummy` variable in listing 3.2.

3.3 sPlot-based Training in the Belle II Analysis Software Framework

The new module `TMVASPlotTeacher` can be added to a path like other modules:

Listing 3.3: Adding `TMVASPlotTeacher` to a path inside a steering file.

```
splotteacher = register_module('TMVASPlotTeacher')
splotteacher.param('prefix', ...)
splotteacher.param('method', ...)
splotteacher.param('listNames', 'D0')
...
main.add_module(splotteacher)
```

It has the same output as the `TMVATeacher` module, more precisely it writes the configuration of a trained classifier to disk. Thus, applying a classifier trained with `TMVASPlotTeacher` on a data sample is no different from applying any another classifier which was trained through BASF2's TMVAInterface:

Listing 3.4: Apply a trained classifier inside a steering file.

```
expert = register_module('TMVAExpert')
expert.param('prefix', ...)
expert.param('method', ...)
expert.param('listNames', 'D0')
expert.param('expertOutputName', 'Test_Probability')
main.add_module(expert)

output = register_module('VariablesToNtuple')
output.param('particleList', 'D0')
output.param('variables', ['extraInfo(Test_Probability)', 'isSignal', 'M'↩
    ])
output.param('fileName', prefix + '_expert_per_candidate.root')
output.param('treeName', 'variables')
main.add_module(output)
```

The module `TMVAExpert` reads the configuration of a trained classifier, calculates the classifier output for every candidate and attaches it to in the sample. If the user wants to analyse the classifier output externally, the `VariablesToNtuple` module can be used to write it to disk.

3.3.1 Parameters

`TMVASPlotTeacher` has to be configured through parameters. In the steering file, they are set like it was already shown in the previous listings:

Listing 3.5: Set a parameter inside a steering file.

```
splotteacher.param(<parameter name>, <parameter value>)
```

Some of `TMVASPlotTeacher`'s parameters are the same as for the standard `TMVATeacher` module. The reason for this is that internally, they both use the same interface to access TMVA to train the specified classifier. Thus, to keep this section short, only

parameters introduced for `TMVASPlotTeacher` will be described hereafter. Parameters that appear both in `TMVATeacher` and `TMVASPlotTeacher` are only mentioned if they have to be handled specially for an sPlot training. The full list of parameters of `TMVATeacher` with a short description can be obtained by running `basf2 -m TMVATeacher`. In the following, the parameters are listed with their name, type, default value and value used in the example analysis discussed in Section 4.2.

`prefix` (**`string`**)
default: `'TMVA'` example: `'TMVASPlot'`

This parameter is the same as in `TMVATeacher`. This prefix is used by the `TMVAInterface` to store its configuration file `$prefix.config` and by TMVA itself to write the files `weights/$prefix_$method.class.C` and `weights/$prefix_$method.weights.xml` with additional information. `TMVASPlotTeacher` writes the plot of the fit to `$prefix_pre_splot_fit.png`.

`prepareOption` (**`string`**)
default: `!V:SplitMode=Alternate:MixMode=Block:NormMode=None` example: `(default)`

This parameter is the same as in `TMVATeacher`. This configuration string is passed to TMVA and defines how the data sample is split into training and test sample. The default option guarantees that the instances of an event are not separated. If this is not respected, it could happen that only events with small weights or only events with large weights are chosen for a class. A classifier trained on such a sample would perform poorly. With `NormMode=None`, a change of the weights by TMVA is inhibited. Therefore, this parameter should not be changed.

`methods` (**`list of tuples(name, type, config)`**)
default: `no default` example: `('NeuroBayes', 'Plugin', 'H:V:...')`

This parameter is the same as in `TMVATeacher`. It specifies the classifier and its configuration. For an sPlot training, a classifier has to be used that supports negative weights. This and other requirements for the classifier are discussed in more detail in Section 3.3.2.

`variables` (**`list of strings`**)
default: `no default` example: `'daughter(0, Kid)', 'daughter(1, piid)', ...`

This parameter is the same as in `TMVATeacher`. These are the input variables used by the TMVA method to train the classifier. In an sPlot training, these variables must not depend on the discriminating variables or have any correlation with them. A trivial conclusion is that the discriminating variables themselves are *not* part of this list.

`spectators` (**`list of strings`**)
default: `empty` example: `(default)`

This parameter is the same as in `TMVATeacher`. A list of variables that are saved in the output file, but not used as training input. `TMVASPlotTeacher` automatically adds the discriminating variables to this list. This way, it is more convenient for the user to analyse the distribution of the sWeights with regard to the discriminating variables.

`doNotTrain` (**`boolean`**)
default: `false` example: `(default)`

This parameter is the same as in `TMVATeacher`. If set to `true`, the classifier is not trained but only a data file with the prepared samples is written to disk. The sWeights are included in this file. It is then possible to use BASF2's `externTeacher` command to perform the training without the need of a steering file.

`maxEventsPerClass` (**`unsigned int`**)
default: `0` example: `(default)`

This parameter is the same as in `TMVATeacher`. Maximum number of events per class passed to TMVA. 0 means no limit. This is useful if it is not clear how large the data sample is. In an sPlot training, all events are used for all classes. Thus, this parameter defines how many events of the data sample are used.

`discriminatingVariables` (**`list of strings`**)
default: `empty` example: `'M'`

The list of discriminating variables used by sPlot to determine the sWeights. The names provided here must match the names that were chosen in the declaration of the `RooRealVar`s in the model—that means the names defined in the constructor of the objects, not the names of the C++ or Python variables.

`modelFileName` (**`string`**)
default: `empty` example: `'TMVASPlot_model.root'`

Path to the ROOT file containing the model which describes the distribution of the candidate classes in the discriminating variable. This file will only be opened read-only.

`modelObjectName` (**`string`**)
default: `'model'` example: `'model'`

Name of the `RooAbsPdf` object in the ROOT file.

`modelPlotComponentNames` (**`list of strings`**)
default: `empty` example: `'sig', 'bkg'`

`TMVASPlotTeacher` plots the fitted model to a `PNG` file as a simple check for the user. If names of `RooAbsPdf` objects are provided with this parameter, they are plotted additionally on the same plot. This can be used to assess the fit for each class separately.

`modelYieldsObjectNames` (**`list of strings`**)
default: `empty` example: `'bkgfrac', 'sigfrac'`

Name of the `RooRealVar` objects that represent the yields of the classes in the model. There have to be as many yields as classes—the number of yields is interpreted as the number of classes. In the case of two classes, the yield for the signal class has to be the second. The reason is that the classes are internally numbered according to their order and BASF2's TMVA interface interprets the largest class number as signal.

`setYieldRanges` (**`boolean`**)
default: `true` example: `(default)`

If set to `true`, the module sets the upper limit of the yield variables that are defined with `modelYieldsObjectNames` to the number of candidates. The lower limit is not

changed. This feature is useful if the user does not know the number of candidates when writing the steering file, for example if they want to use one steering file for different data samples.

`modelYieldsInitialFractions` (`vector of floats`)
default: `empty` example: `0.99, 0.01`

If set, the initial value of yield i will be set to `initialFraction`$[i]\times$`numberOfCandidates`. If there are fewer initial fractions given than yields, the initial values of the remaining yields are not changed. This feature is useful if the user does not know the number of candidates when writing the steering file, and the fit depends on well-set initial values.

3.3.2 Underlying Multivariate Classifiers

Multivariate classifiers used for an sPlot training must support negative weights. The TMVA user guide includes a tabular overview of the provided classifiers and their abilities [10, page 135]. Weights are supported by all but one classifier. Negative weights are only supported by 6 out of 15 methods: `Likelihood`, `PDE-RS`, `PDE-Foam`, `k-NN`, `SVM` and `BDT`. It is important to note that a classifier marked as supporting negative weights does not necessarily do so in all configurations or even in its default configuration. Thus, the choice of a multivariate method and its configuration needs to be done by carefully consulting the documentation. The following two TMVA methods were shortly tested and seem to work:

```
1 [('BDTStochastic',
2   'BDT',
3   '!H:!V:NTrees=100:BoostType=Grad:Shrinkage=0.10:GradBaggingFraction=0.5:↩
       UseBaggedBoost:nCuts=256:MaxDepth=3'),
4  ('BDT',
5   'BDT',
6   '!H:!V:NTrees=100:BoostType=Grad:Shrinkage=0.10:GradBaggingFraction=1.0:↩
       nCuts=256:MaxDepth=3')]
```

FastBDT, the classifier distributed with BASF2, supports negative weights using the following configuration:

```
1 [('FastBDT',
2   'Plugin',
3   '!H:!V:NTrees=100:Shrinkage=0.10:RandRatio=0.5:NCutLevel=8:NTreeLayers=3↩
       ')]
```

For the example analyses presented in this Thesis, the proprietary classifier NeuroBayes® is used. It supports negative weights out of the box and is able to automatically pre-process input variables, for example it can perform a decorrelation. The configuration tuple is given in Section 4.2.1.

3.3.3 Implementation

`TMVASPlotTeacher` is a BASF2 module written in C++. Although it is very similar to the standard `TMVATeacher`, it is not implemented as a class derived from `TMVATeacher`. The differences in data processing and parameters are so significant

that the implementation would not become cleaner by using inheritance. However, the modules still share a large amount of functionality as they both use the TMVA interface to perform the training. The implementation of `TMVASPlotTeacher` was the main work done in the scope of this Thesis; including one change in the TMVA Interface to make it possible to insert weights into the data sample.

As every module in the event processing chain, `TMVASPlotTeacher` implements three phases: initialisation, event processing, and termination. In the initialisation phase the model is read from the provided ROOT file. Some basic checks are performed to validate the parameters provided by the user. For example it is verified that all discriminating variables and yields are present in the model. At this stage, the discriminating variables are added to the list of spectator variables, as well. In the event processing phase the values of all discriminating variables are collected for each event. `TMVASPlotTeacher` is integrated into BASF2 in such a way that this phase can automatically be parallelized, like it is the case for many other BASF2 modules. In the termination phase the fit of the model to the data, the calculation of sWeights and the training are performed. Directly before the fit, the ranges and initial values of the yield variables are updated using the now known number of candidates depending on how the parameters `setYieldRanges` and `modelYieldsInitialFractions` are set by the user.

As mentioned before, the model needs to be passed to `TMVASPlotTeacher` through a ROOT file. In general, it could be made possible to pass the RooFit object directly from the steering file to the module. However, this would need a tremendous amount of conversion code to be written for the translation of RooFit objects defined with PyROOT to C++ objects. Thus, the detour over a ROOT file was preferred.

4. Applications of sPlot

4.1 Analyses in Belle and other Experiments

We are not aware of analyses in the course of the Belle experiment that use an sPlot-based training. However, sPlot has been used in the way for which it was designed originally, namely as a means to plot the distributions of control variables. For example, in the ongoing analysis by Andreas Heller [18] of the decay $B^+ \to l^+ \nu \gamma$, sPlots are used to determine the systematic uncertainties of the classifiers which were trained on MC. Besides that, sPlots are widely used in B physics [19].

sPlot-based training was used successfully for example in analyses performed by the CDF collaboration. In [20] and [21], the background could be drastically reduced by using the invariant mass of the particles as discriminating variable.

4.2 Example Study in Belle II on $D^0 \to K^-\pi^+$

In the following, an example analysis is performed to show the usage of the new BASF2 module for sPlot-based training, and to compare the performance of the sPlot-based training with a standard training. As no data is available for the Belle II experiment yet, both trainings are done on the same data sample from generic MC simulations. The sPlot-based training does not use the MC truth but only the sWeights. A third training is performed to check if the variables used in the training are correlated to the discriminating variable. In this analysis, one is interested in D^0 mesons that decay into K^- and π^+. The classifiers are trained to separate correctly reconstructed D^0s from background candidates, where the K^- and π^+ did not originate from a D^0.

4.2.1 Steering File for the Trainings

In the following, the steering file used for the analysis is presented step by step.

```
1 from basf2 import *
2 from modularAnalysis import *
3 import ROOT
4 from ROOT import (RooRealVar, RooGaussian, RooChebychev, RooAddPdf, ↩
      RooArgList, RooFit, RooAbsReal, TFile, kFALSE)
5
6 # configuration
7 splotPrefix = "TMVASPlot"
8 modelFileName = splotPrefix + "_model.root"
9 modelObjectName = "model"
```

Firstly, the needed Python libraries are imported. `basf2` and `modularAnalysis` provide access to the BASF2 modules and to the functions needed to build the analysis path. The ROOT libraries are used to define the model needed by sPlot. Then, some variables are defined for later use as file and ROOT object names.

```
10 main = create_path()
11 main.add_module(register_module('RootInput'))
12 main.add_module(register_module('Gearbox'))
```

The path is created, and three modules added to it. `RootInput` is needed to load data from `ROOT` files, the `Gearbox` module loads the detector parameters and the

```
13 fillParticleList('K-', 'Kid > 0.1', path=main)
14 fillParticleList('pi+', 'piid > 0.1', path=main)
```

K^- and π^+ candidates are selected with a soft cut on the corresponding identification likelihood ratio.

```
15 reconstructDecay('D0:raw -> K- pi+', '1.8275 < M < 1.9025', path=main)
16 vertexKFit('D0:raw', 0.001, path=main)
17 cutAndCopyList('D0', 'D0:raw', '1.83 < M < 1.9', path=main, writeOut=True)
18 matchMCTruth('D0', path=main)
```

Now, D^0 candidates are reconstructed from K^- and π^+. A range around the already known invariant mass of the reconstructed D^0 is chosen. Then, a vertex fit is performed to find the secondary vertex of the D^0. A soft cut drops candidates for which the K^- and π^+ tracks have no common vertex. The list of D^0 candidates is then copied while cutting a small piece from both sides of the mass range. This is necessary because the vertex fit changes the invariant mass a bit. The invariant mass of some particles shifts outside of the range, which reduces the statistics in the bins at the borders of the range. Thus, these regions are cut. The last line adds the MC information to the candidates.

```
19 variables = ['daughter(0, Kid)', 'daughter(1, piid)',
20              'decayAngle(0)',
21              'daughter(0, dr)', 'daughter(0, dz)',
22              'daughter(1, dr)', 'daughter(1, dz)',
23              'daughter(0, chiProb)', 'daughter(1, chiProb)',
24              'p_CMS', 'pt_CMS', 'pz_CMS',
25              'dr', 'dz', 'chiProb', 'significanceOfDistance',
26              'VertexZDist']
27 discriminatingVariables = ['M']
```

Here, the variables used in the training and the discriminating variables are defined. Hereafter, the variables used in the training are refered to as control variables to be consistent with the terminology of sPlot. The invariant mass M is used as discriminating variable as it is easy to fit to a model.

```
28 methods = [
29     ('NeuroBayes', 'Plugin', 'H:V:Preprocessing=122:ShapeTreat=OFF:↩
          NtrainingIter=20:NBIndiPreproFlagByVarname=daughter0Kid=94,↩
          daughter1piid=94,' + ",".join( ROOT.Belle2.Variable.↩
          makeROOTCompatible(variable) + "=34" for variable in variables[2:]↩
           ))
```

```
30 ]
```

This is the classifier configuration that is used for all trainings. The proprietary classifier NeuroBayes® can be configured to perform a different preprocessing for each variable. An explanation of the processing flags can be found in [6].

In the following, the model of the discriminating variable is defined using RooFit classes. This has already been discussed in detail in Section 3.2.

```
31 # observable
32 M = RooRealVar("M", "M", 1.83, 1.9)
33 M.setBins(250)
34
35 # Setup component PDFs
36 mD0 = RooRealVar("mD0", "D0 Mass", 1.865)
37 sigmaD0 = RooRealVar("sigmaD0", "Width of Gaussian", 0.025)
38 sig = RooGaussian("sig", "D0 Model", M, mD0, sigmaD0)
39 mD0.setConstant(kFALSE)
40 sigmaD0.setConstant(kFALSE)
41
42 a0 = RooRealVar("a0", "a0", -0.69)
43 a1 = RooRealVar("a1", "a1", 0.1)
44 a0.setConstant(kFALSE)
45 a1.setConstant(kFALSE)
46 bkg = RooChebychev("bkg", "Background", M, RooArgList(a0, a1))
47
48 # Add signal and background
49 dummy = 42
50 bkgfrac = RooRealVar("bkgfrac", "fraction of background", dummy, 0., dummy↩
       )
51 sigfrac = RooRealVar("sigfrac", "fraction of background", dummy, 0., dummy↩
       )
52
53 # Parameters for TMVASPlotTeacher
54 modelYieldsObjectNames = ['bkgfrac', 'sigfrac']
55 modelYieldsInitialFractions = [0.99, 0.01]
56 modelPlotComponentNames = ['sig', 'bkg']
57
58 bkgfrac.setConstant(kFALSE)
59 sigfrac.setConstant(kFALSE)
60
61 model = RooAddPdf(modelObjectName, "bkg+sig", RooArgList(bkg, sig), ↩
       RooArgList(bkgfrac, sigfrac))
62
63 # Write model to file and close the file, so TMVASPlotTeacher can open it.
64 modelFile = TFile(modelFileName, "RECREATE")
65 model.Write(modelObjectName)
66 modelFile.ls()
67 modelFile.Close()
```

The range of the discriminating variable is chosen tight around the D^0 peak, so that no satellite resonances are included. It is important that the range defined

in the model is the same as the range in which candidates were reconstructed. Otherwise, the fit performed before the calculation of the sWeights could fail.

Now, a `TMVASPlotTeacher` is added to the path. The parameters defined in the previous parts of the steering file are passed to it.

```
splotteacher = register_module('TMVASPlotTeacher')
splotteacher.param('prefix', splotPrefix)
splotteacher.param('methods', methods)
splotteacher.param('variables', variables)
splotteacher.param('discriminatingVariables', discriminatingVariables)
splotteacher.param('modelFileName', modelFileName)
splotteacher.param('modelObjectName', modelObjectName)
splotteacher.param('modelYieldsObjectNames', modelYieldsObjectNames)
splotteacher.param('modelYieldsInitialFractions', ↩
    modelYieldsInitialFractions)
splotteacher.param('modelPlotComponentNames', modelPlotComponentNames)
splotteacher.param('listNames', 'D0')
main.add_module(splotteacher)
```

The parameter `listNames` is used to specify that the training should be performed on the D^0 candidates. The discriminating variables are automatically added to the spectator variables, so this parameter is omitted here.

For the standard training, `TMVATeacher` is added to the path.

```
teacher = register_module('TMVATeacher')
teacher.param('prefix', 'TMVA')
teacher.param('methods', methods)
teacher.param('variables', variables)
teacher.param('spectators', discriminatingVariables)
teacher.param('target', 'isSignal')
teacher.param('prepareOption', '!V:SplitMode=random:NormMode=None')
teacher.param('listNames', 'D0')
main.add_module(teacher)
```

The important difference is that for `TMVATeacher`, the target of the training is defined with the parameter `target`, which is the MC truth in this case. The target of the training for a `TMVASPlotTeacher` is defined via the names of the yields in the model, provided via the `modelYieldsObjectNames` parameter.

To test if the control variables are uncorrelated to the discriminating variable M, the following classifier is used:

```
cutAndCopyList('D0:background', 'D0', 'isSignal == 0', path=main)
teacher = register_module('TMVATeacher')
teacher.param('prefix', 'TMVAMCorrelation')
teacher.param('methods', methods)
teacher.param('variables', variables)
teacher.param('spectators', discriminatingVariables)
teacher.param('target', 'isInRegion(M, 1.855, 1.875)')
teacher.param('prepareOption', '!V:SplitMode=random:NormMode=None')
teacher.param('listNames', 'D0:background')
main.add_module(teacher)
```

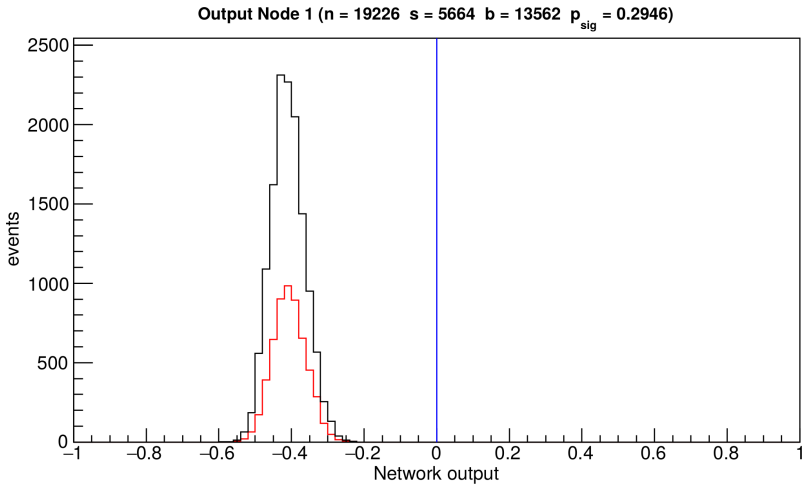


Figure 4.1: Classifier output distributions of the correlation training, taken from the NeuroBayes® report. The red curve is signal, the black one for background. It can be seen that no separation of the two samples is possible. NeuroBayes® uses the target value −1 for background and 1 for signal.

The classifier is trained to predict, for background candidates, if the invariant mass is located in the signal region. Thus, *signal* in this case are background candidates with an invariant mass in the signal region, and *background* are background candidates with an invariant mass outside the signal region. If the control variables are correlated to M, this separation is possible. For this training, a particle list has to be prepared that only contains background candidates. Finally, the path is executed with the following line:

```
99  process(main)
```

4.2.2 Check for Correlation of the Control Variables

The result of the correlation test of control and discriminating variables is shown in Figure 4.1. It shows the classifier output distribution for signal and background candidates. What signal and background means in this classification, was defined above. The distributions are located on top of each other and thus, the classifier cannot distinguish them. This means that the control variables are uncorrelated to the discriminating variables and the sPlot-based training can be used.

4.2.3 Distribution of the sWeights in the Training Sample

As a next step, the calculation of the sWeights is examined. Figure 4.2 shows that the fit of the model to the data was successful, and the model is able to describe the data well.

Figure 4.3 shows the distributions of the sWeights for signal and background. Most candidates are assigned extreme weights. This is consistent with the model, as for many candidates it is clear to which class they belong. More candidates are assigned a large background weight. The reason is that there are more background than signal candidates in the sample. The numbers are 38 452 for background and 24 451 for signal, which adds up to 62 903. Interestingly, the yields are determined by the fit are 44 853 for background and 18 050 for signal. This is an

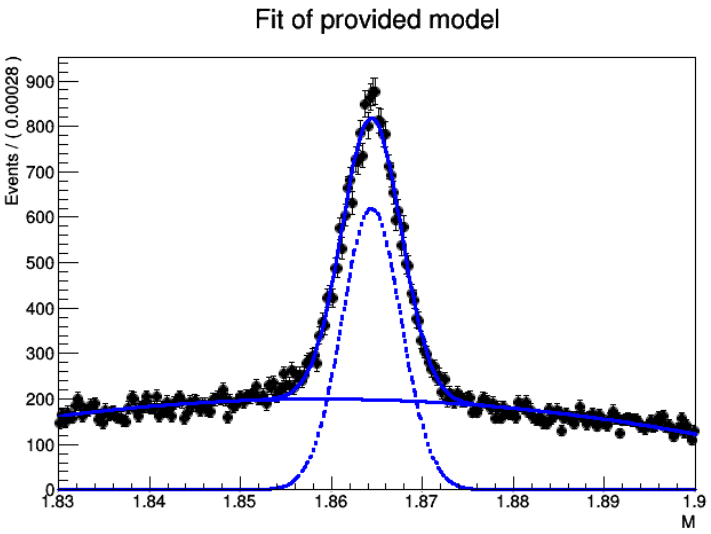


Figure 4.2: Control plot produced by `TMVASPlotTeacher`. The fit of the model to the data succeeded and describes the data well.

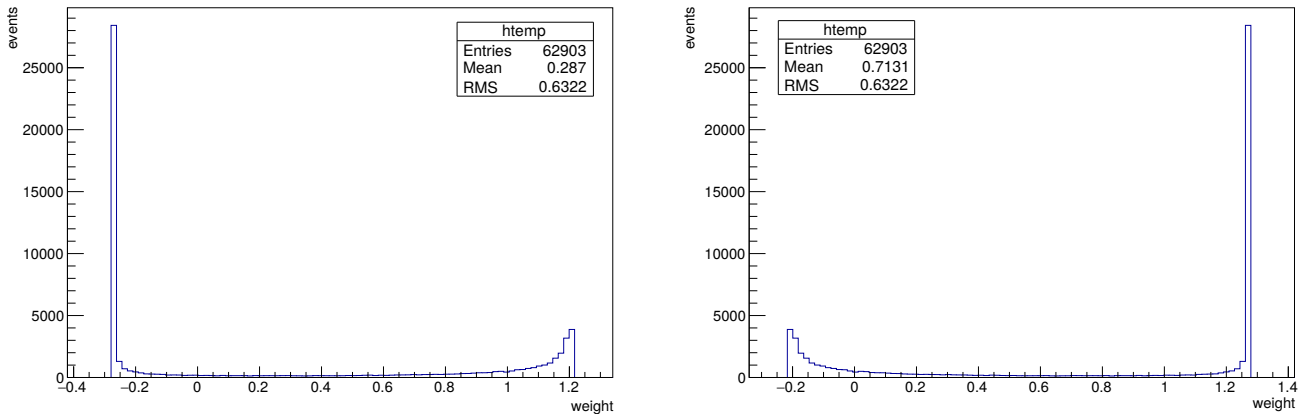
indication that the model does not describe the MC as accurate as thought initially. The fact that the ranges of the weights differ between signal and background is also resulting from the different yields for signal and background; the covariance between the two classes is the same for both the calculation of the signal and background weights with Equation 2.32, but the variance of background is larger than the variance of signal. Two facts that agree with Equation 2.37 are that the mean values of the two distributions sum up to 1, and that the maximum weight of one class sums up to 1 with the minimum of the other class respectively.

Figure 4.4 shows the distribution of the sWeights for signal and background depending on the value of the discriminating variable M. It can be seen that every M is assigned one weight for signal and background respectively, disregarding numerical inaccuracies of the histogram. The signal and background weights sum up to 1 for every M. Consulting Figure 4.2, weights close to 0.5 are assigned to values of M where signal and background distribution are close, too.

4.2.4 Performance Comparision of the Trainings

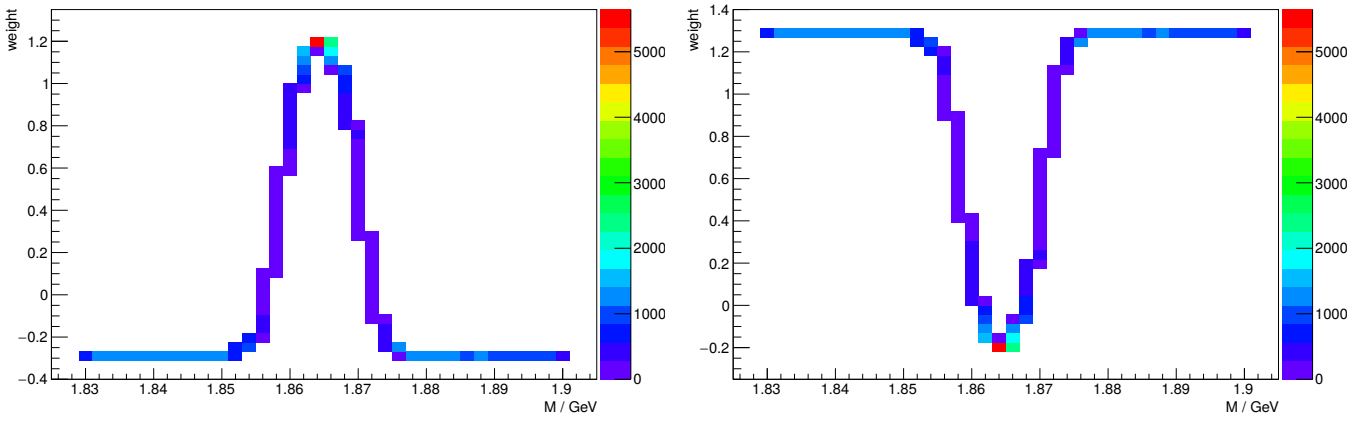
As a last step, the two trained classifiers are now applied on one independent data sample. This sample has the same size than the one that was used for the training. For that purpose, the module `TMVAExpert` is used in a steering file respectively. The file is explained on the basis of the one for the sPlot-based training. To begin with, the steering files start with the same commands as the one used for the trainings to reconstruct the candidates:

```
1 from basf2 import *
2 from modularAnalysis import *
3 main = create_path()
4 main.add_module(register_module('RootInput'))
5 main.add_module(register_module('Gearbox'))
6 main.add_module(register_module('ParticleLoader'))
7 fillParticleList('K-', 'Kid > 0.1', path=main)
8 fillParticleList('pi+', 'piid > 0.1', path=main)
```



(**a**) Signal (**b**) Background

Figure 4.3: Distribution of the sWeights for signal and background. Most candidates are assigned extreme weights. The extremal values differ a bit between signal and background. The mean values of signal and background distribution sum up to 1 corresponding to Equation 2.37.



(**a**) Signal (**b**) Background

Figure 4.4: Distribution of the sWeights for signal and background depending on M. Every M is assigned one weight for signal and background respectively, disregarding numerical inaccuracies of this histogram. Signal and background weights sum up to 1 for every M.

```
reconstructDecay('D0:raw -> K- pi+', '1.8275 < M < 1.9025', path=main)
vertexKFit('D0:raw', 0.001, path=main)
cutAndCopyList('D0', 'D0:raw', '1.83 < M < 1.9', path=main, writeOut=True)
matchMCTruth('D0', path=main)
```

Then, `TMVAExpert` is added to the path:

```
expert = register_module('TMVAExpert')
expert.param('prefix', 'TMVASPlot')
expert.param('method', 'NeuroBayes')
expert.param('listNames', 'D0')
expert.param('expertOutputName', 'Test_Probability')
main.add_module(expert)
```

Here, the same prefix and the same method name are used as with the `TMVASPlotTeacher` module. This way, the expert finds the classifier definition on disk. The output of the classifier is attached to the data set with the name `'Test_Probability'`.

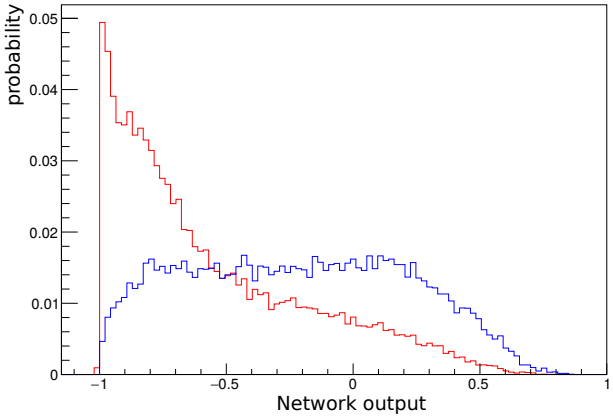
With the `VariablesToNtuple` module, a ROOT file is produced for the further studies of the classifier performance:

```
output = register_module('VariablesToNtuple')
output.param('particleList', 'D0')
output.param('variables', ['extraInfo(Test_Probability)', 'isSignal', 'M'↩
    ])
output.param('fileName', 'TMVASPlot' + '_expert_per_candidate.root')
output.param('treeName', 'variables')
main.add_module(output)
process(main)
```

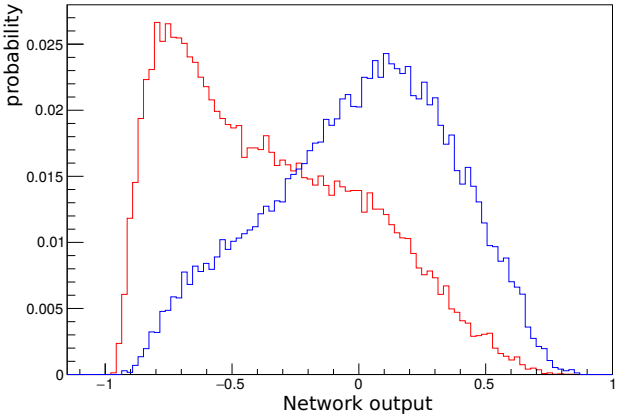
For each candidate, the invariant mass, the MC truth and the classifier output are written to disk. The steering file for the application of the standard training can be derived from this one by replacing the two occurences of `'TMVASPlot'` by `'TMVA'`.

The performance of the classifiers is now examined and compared by means of the classifier output distributions and two ROC plots. The classifier output distributions are shown in Figure 4.5. It can be seen that both classifiers can separate signal and background. The classifier trained using sPlot has a better identification of background than the standard classifier, because the background distribution is shifted more to the left side.

The ROC plots in Figure 4.6 show that the sPlot-based training performs not much worse than the standard training. Consulting Figure 4.5, the main reason for the loss in performance is that the signal distribution of the sPlot-based classifier has large contributions far into the background range.

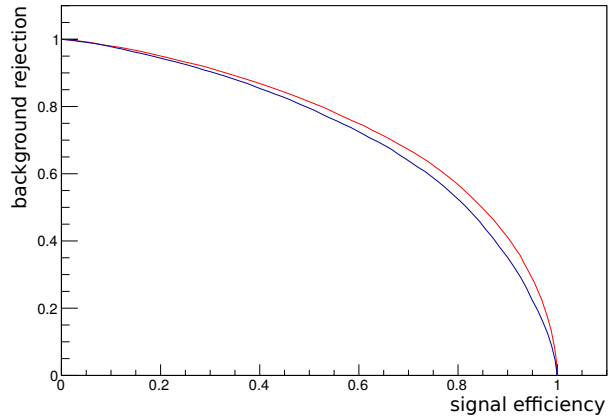


(**a**) sPlot-based training

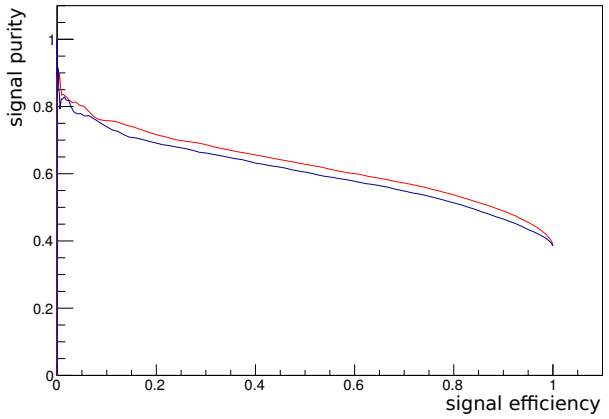


(**b**) Standard training

Figure 4.5: Classifier output distributions for sPlot-based and standard training. The red curve is signal, the blue curve is background. Both classifiers are able to separate signal and background. NeuroBayes® uses the target value −1 for background and 1 for signal.



(**a**) Background rejection over signal efficiency.



(**b**) Signal purity over signal efficiency.

Figure 4.6: ROC plots comparing sPlot-based and standard training. Blue is the sPlot-based training and red the standard training. The sPlot-based training performs not much worse than the standard training.

5. Summary and Outlook

In the scope of this Thesis, an sPlot-based data-driven training of multivariate classifiers was implemented into the Belle II Analysis Software Framework (BASF2). It can now easily be used within the event processing pipeline by the users of BASF2.

Data-driven training is useful in cases where the available simulated Monte Carlo data differs significantly from the real data. The sPlot formalism can, based on a model of some discriminating variables, reconstruct the distributions of control variables that are uncorrelated with the discriminating variables. It does this by introducing weights per event and per class. These weights can be used in a training where the classifier will learn only based on real data to separate the different classes.

On the basis of an example analysis, it was shown how the new module is used within BASF2. The performance of the classifier training using sPlot was compared to a classifier trained on the same data sample the standard way using the Monte Carlo truth. Because no real data was available, the sPlot training was performed on Monte Carlo data, but without using the Monte Carlo truth. Thus, it was expected that the performance is worse than with a standard training. However, it could be seen that the drop in performance was not large. As a result, the classifier could perform better than the classifier trained in the standard way on real data. This is particularly interesting during the initial stage of an experiment, when no MC corresponding to the current detector configuration is available yet.

It is planned to use the sPlot-based training within the Full Event Interpretation (FEI) [5]. The FEI is an advanced analysis method in BASF2. It is useful for analyses that reconstruct the decays of both B mesons originating from an $\Upsilon(4S)$, which is a technique unique to B factories: The decay channel of interest, called the signal side, contains a B meson called B_{sig}. Because the B mesons are entangled, some properties of the other B meson, called B_{tag}, like its flavour and 4-momentum, are implicitly defined. By trying to reconstruct the tag side in addition to the signal side, the accuracy of the analysis can be improved, because the background can be reduced drastically. With sPlot, many classifiers employed in the FEI could be trained on real data automatically. On the tag side, decay channels with high statistics are used anyway, which is a good precondition for data-driven training.

For the future, it is also planned to extend the sPlot method so it can correctly handle variables correlated with the discriminating variables. Then, performing a training on real data becomes even more easy as there are fewer constraints on the choice of variables.

Danksagung

Ich bedanke mich bei Prof. Dr. Michael Feindt für die Möglichkeit, meine Bachelorarbeit in der *B*-Arbeitsgruppe am Institut für Experimentelle Kernphysik anfertigen zu können. Für die Übernahme des Korreferats und viele hilfreiche Diskussionen bedanke ich mich bei Dr. Martin Heck. Besonders danken möchte ich Thomas Keck für die exzellente Betreuung meiner Bachelorarbeit. Ich habe sehr viel gelernt und hatte großen Spaß, mit dir zusammen zu arbeiten. Ich danke der gesamten Arbeitsgruppe für die gemeinsame Zeit und die sehr gute Arbeitsatmosphäre. Danke für die hohe Hilfsbereitschaft, sowie die hilfreichen Diskussionen über meine Arbeit. Ein großer Dank geht an das Admin-Team für das Betreiben der IT-Infrastruktur und besonders an die Paket-Admins für das Erfüllen meiner Wünsche an nun fast jedem einzelnen Arbeitsplatzrechner in Raum 9-2. Ich danke Markus Prim, Igor Babuschkin, Christian Pulvermacher, Pia Lipp und Nefta Kanilmaz für das zusätzliche Korrekturlesen meiner Arbeit. Ich danke meinen Freunden und Kommilitonen, die mich während des Studiums begleitet haben und es zu einer großartigen Zeit gemacht haben. Ich danke meiner Familie und besonders meinen Eltern für die uneingeschränkte Unterstützung, die ich während meines Studiums erfahren habe.

Bibliography

[1] *The Belle II logo*. URL: http://belle2.kek.jp/logos.html (visited on 05/08/2015).

[2] Christian Pulvermacher. "dE/dx particle identification and pixel detector data reduction for the Belle II experiment". In: *not published* (2012). Karlsruher Institut für Technologie (KIT), Diploma Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48263.

[3] D. Martschei, M. Feindt, S. Honc, and J. Wagner-Kuhr. "Advanced event reweighting using multivariate analysis". In: *Journal of Physics: Conference Series* 368.1 (2012), p. 012028. URL: http://stacks.iop.org/1742-6596/368/i=1/a=012028.

[4] M. Pivk and F. R. Le Diberder. "sPlot: A statistical tool to unfold data distributions". In: *Nuclear Instruments and Methods in Physics Research A* 555 (Dec. 2005), pp. 356–369. DOI: 10.1016/j.nima.2005.08.106. eprint: physics/0402083.

[5] T. Keck. "The Full Event Interpretation for Belle II". In: *not published* (2014). Karlsruher Institut für Technologie (KIT), Master Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48602.

[6] M. Büchner. "A Study of Multivariate Techniques for Continuum Suppression". In: *not published* (2014). Karlsruher Institut für Technologie (KIT), Bachelor Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48626.

[7] F. Wick. "Untersuchung von Charm-Baryonen in den Zerfallskanälen $\Lambda_c\pi$ und $\Lambda_c\pi\pi$ mit dem CDF-Detektor". In: *not published* (2009). Karlsruher Institut für Technologie (KIT), Diploma Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/44869.

[8] R. Barlow. "Extended maximum likelihood". In: *Nuclear Instruments and Methods in Physics Research A* 297 (Dec. 1990), pp. 496–506. DOI: 10.1016/0168-9002(90)91334-8.

[9] I. Antcheva et al. "ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization". In: *Computer Physics Communications* 180.12 (2009), pp. 2499–2512. DOI: http://dx.doi.org/10.1016/j.cpc.2009.08.005.

[10] A. Hocker et al. "TMVA - Toolkit for Multivariate Data Analysis". In: *PoS* ACAT (2007), p. 040. arXiv: physics/0703039 [PHYSICS].

[11] *The TSPlot Class in ROOT*. URL: https://root.cern.ch/root/html/TSPlot.html (visited on 05/12/2015).

[12] *The RooStats::SPlot Class in ROOT*. URL: https://root.cern.ch/root/html/RooStats__SPlot.html (visited on 05/12/2015).

[13] L. Moneta, K. Cranmer, G. Schott, and W. Verkerke. "The RooStats project". In: *Proceedings of the 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research. February 22-27, 2010, Jaipur, India*. 2010, p. 57. arXiv: 1009.1003 [physics.data-an].

[14] W. Verkerke and D. Kirkby. "The RooFit toolkit for data modeling". In: *ArXiv Physics e-prints* (June 2003). eprint: physics/0306116.

[15] W. Verkerke and D. Kirkby. *RooFit Users Manual v2.07*. Jan. 2006. URL: http://roofit.sourceforge.net/docs/RooFit_Users_Manual_2.07-29.pdf (visited on 05/01/2015).

[16] *RooFit tutorials distributed with ROOT*. URL: https://root.cern.ch/root/html/tutorials/roofit/index.html (visited on 04/28/2015).

[17] *RooFit Online Tutorials*. URL: http://roofit.sourceforge.net/docs/tutorial/index.html (visited on 04/28/2015).

[18] A. Heller et al. "Search for B+ -> l+ nu gamma decays with hadronic tagging using the full Belle data sample". In: *ArXiv e-prints* (Apr. 2015). arXiv: 1504.05831 [hep-ex].

[19] A. J. Bevan et al. "The Physics of the B Factories". In: *European Physical Journal C* 74 (Nov. 2014), p. 3026. DOI: 10.1140/epjc/s10052-014-3026-9. arXiv: 1406.6311 [hep-ex].

[20] C. Marino. "X_b Search and Measurement of the Y(1S), Y(2S) and Y(3S) Polarization". In: *not published* (2009). Karlsruher Institut für Technologie (KIT), PhD Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/45098.

[21] F. Wick. "Charmed Baryon Spectroscopy and Search for CP Violation in $D^0 \to K^0_S \pi^+ \pi^-$". In: *not published* (2011). Karlsruher Institut für Technologie (KIT), PhD Thesis. URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48216.