Karlsruher Institut für Technologie

IEKP-KA/2014-18

The Full Event Interpretation for Belle II

Thomas Keck

Masterarbeit

an der Fakultät für Physik des
Karlsruher Instituts für Technologie (KIT)

Referent: Prof. Dr. M. Feindt
Korreferent: Dr. T. Kuhr

Institut für Experimentelle Kernphysik

November 2014

Contents

1. Introduction

The Belle II experiment offers the possibility to perform unique measurements with unprecedented precision. The Full Event Interpretation (FEI) described in this thesis is an essential component in a wide range of important analyses, including: the measurement of the CKM element $|V_{ub}|$ through the semileptonic decay $b \to u\ell\bar{\nu}$; the search for a charged-Higgs effect in $B \to D\tau\bar{\nu}$; and the precise measurement of the branching fraction of $B \to \tau\bar{\nu}$, which is sensitive to new physics effects.

Like its predecessor, the Belle II experiment will be located at the KEK High Energy Accelerator Research Organization in Tsukuba, Japan. It is designed to perform measurements in all fields of heavy flavour physics, including: $B^{0/\pm}$ meson decays; $B_s^{(*)}$ meson decays; charm physics; τ lepton physics; spectroscopy; and pure electroweak measurements.

Extracting precise physics results from the obtained data requires large-scale computing capabilities, high-quality software tools and advanced analysis techniques like the FEI. Measurements of decays including neutrinos, in particular rare decays, suffer from missing kinematic information. The FEI recovers this information partially and infers strong constraints on the signal candidates by automatically reconstructing the rest of the event in thousands of exclusive decay channels. This work describes the implementation and application of the FEI in the Belle II Analysis Software Framework (BASF2).

1.1. From Belle to Belle II

In 1999 the Belle experiment was put into operation to probe the flavour structure of the Standard Model in general and the Kobayashi–Masukawa mechanism for CP violation in particular. The Belle detector surrounded the single interaction point of the KEKB e^+e^- collider. The collider operated primarily at the energy of the $\Upsilon(4S)$ resonance, which decays exclusively to an entangled $B\bar{B}$ meson pair. Therefore, this experimental setup is called a B factory [1].

In 2001 the Kobayashi–Masukawa mechanism was verified by Belle [2, 3] and BaBar [4] through the observation of mixing-induced CP violation in $B^0 \to J/\psi K_S$ decays; consequently Makoto Kobayashi and Toshihide Masukawa were awarded the Nobel prize in physics in 2008. By November 2009 the experiment had accumulated a data sample with an integrated luminosity of $1\,\mathrm{ab}^{-1}$.

Other major achievements include:

- the measurements of time-dependent mixing-induced CP violation in further B-decays [5];
- the measurement of direct CP violation in B decays [6];
- the measurement of all unitarity triangle angles;
- the observation of the purely leptonic decay $B^+ \to \tau^+\nu$ [7];
- the discovery of a new resonance $X(3872)$, which exhibits properties not compatible with conventional mesons [8];
- and the observation of mixing in the system of neutral charmed D^0 mesons [9].

In March 2004 an upgrade of the Belle detector was proposed in a Letter of Intent [10]. Due to the success of Belle the project received strong encouragement and consequently the Belle II collaboration was formed in December 2008. At present the upgraded detector is still under construction, its commissioning is planned for 2016. At the same time, the associated accelerator KEKB is being upgraded to SuperKEKB. It will provide 40 times higher luminosity; as a consequence, Belle II is projected to accumulate 50 times more data than Belle [11]. A detailed description of the upgraded experimental setup is given in Chapter 2.

Belle II will have a broad physics program, which will constrain the parameter space of the Standard Model as well as some of its extensions. Measurements of the branching fraction of rare and tauonic decays have the potential to reveal new physics processes. Additionally, precise measurements of many CP asymmetries will allow the investigation of correlations among them and therefore the possibility to discover additional sources of CP violation. Many measurements possible at Belle II are complementary to measurements at energy-frontier experiments like LHCb.

1.2. From Full Reconstruction to Full Event Interpretation

As an analysis technique unique to B factories, the Full Event Interpretation will play an important role in the measurement of rare decays. This technique reconstructs one of the B mesons and infers strong constraints for the remaining B meson in the event using the precisely known initial state of the $\Upsilon(4S)$. The actual analysis is then performed on the second B meson. The two mesons are called tag-side B_{tag} and signal-side B_{sig}, respectively. This situation is depicted in Figure 1.1.

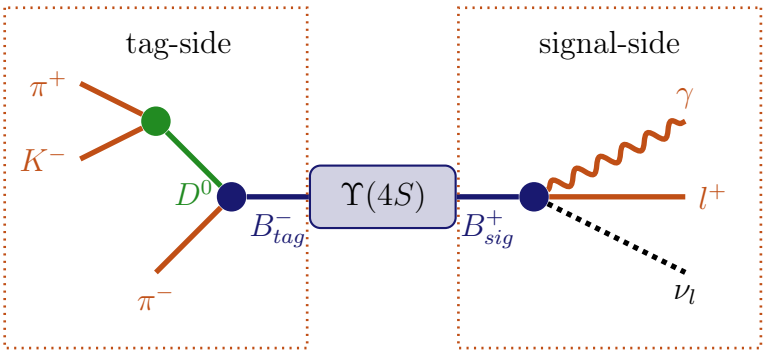


Figure 1.1.: Decay of the $\Upsilon(4S)$ into a charged B meson pair. The signal-side and an important tag-side are shown for the analysis of the $B \to \ell\nu_\ell\gamma$ decay.

In effect the FEI allows one to reconstruct the initial $\Upsilon(4S)$ resonance, and thereby recovering the kinematic and flavour information of B_{sig}. Furthermore, the background can be drastically reduced by discarding $\Upsilon(4S)$ candidates with remaining tracks or energy clusters in the rest of event.

Belle already employed a similar technique called Full Reconstruction (FR) with great success [12]. As a further development the Full Event Interpretation is more inclusive, provides more automation and analysis-specific optimisations.

Both techniques heavily rely on multivariate classifiers (MVC). An introduction into this topic is given in Chapter 3. MVCs have to be trained on a Monte Carlo (MC) simulated data sample. However, the analysis-specific signal-side selection strongly influences the background distributions on the tag-side. Yet this influence had to be neglected by the FR, because the training of the MVCs was done independently from the signal-side analysis [12]. The training was performed only once and took several weeks on the cluster of the KEKCC computing centre.

In contrast, the FEI will be trained for each analysis separately and can thereby take the signal-side selection into account. The analysis-specific training is possible due to the deployment of speed-optimized training algorithms, full automation and

the extensive usage of parallelization on all levels. The total training duration for a typical analysis is in the order of days instead of weeks. In consequence, it is also feasible to retrain the FEI if better MC data or optimized MVCs become available.

The Full Event Interpretation described in this work is built on top of the Belle II analysis software framework. A detailed description of BASF2 is given in Chapter 4. The framework is still under development. In consequence, efficiencies and distributions stated in this thesis serve only illustrating purposes. In particular, at the time of writing there was no beam-background available, the Monte-Carlo matching code did not work properly in some corner cases and the track as well as the vertex fitting was not finished. The FEI itself is described in Chapter 5.

A summary of the first preliminary performance-studies of the FEI is provided in Chapter 6. All details of the studies can be obtained from the separately published Full Event Interpretation Reports (FEIR), which are automatically generated by the FEI during the training. The FEIRs contain key performance indicators and control plots for all reconstructed particles and decay channels.

2. Experimental Setup

This chapter describes the SuperKEKB accelerator and the Belle II detector. At first their respective predecessors KEKB and Belle, which were shut down in 2010, are introduced. Afterwards, SuperKEKB and Belle II are described in more detail, with a special emphasis on improvements and modifications in comparison with their predecessors.

2.1. KEKB

In 1994 the construction of KEKB, a two-ring, asymmetric-energy, electron–positron collider, started in the underground tunnel of the former TRISTAN accelerator [13, sec. 2]. The new collider was dedicated to B physics and its design parameters were chosen with respect to the requirements of a precise observation of B mesons.

The two rings are called HER and LER for the high and low energy ring respectively [13, sec. 2.1]. At the crossing point of the rings their beams collided with a center of mass energy equal to the mass of the $\Upsilon\,(4S)$ resonance

$$E_{CMS} = 2\sqrt{E_{HER}E_{LER}} = 10.58\ \text{GeV} = m_{\Upsilon(4S)}. \tag{2.1}$$

The $\Upsilon\,(4S)$ resonance decays into an entangled $B\bar{B}$ pair with a branching fraction of over 96 %. Neutral $B^0\bar{B}^0$ and charged B^+B^- meson pairs are produced in almost equal parts. Moreover, the B mesons are produced near the energy threshold and without any further particles [15].

As can be seen from Figure 2.1 the resonance is an exited state in the bottonium spectrum[1]. The cross-section for the production $e^+e^- \to \Upsilon(4S)$ is 1.2 nb [16, sec. 11]. In addition, so-called continuum-events $e^+e^- \to q\bar{q}$ with $q = u, d, s, c$ occur[2] with a production cross-section of 2.8 nb [16, sec. 11].

The asymmetric beam energies induced a Lorentz boost

$$\beta\gamma = \frac{E_{HER} - E_{LER}}{E_{CMS}} = 0.42 \tag{2.2}$$

[1] Bottonium is a bound system containing an b quark and $\bar{b}$ anti-quark.

[2] There are further physics processes in e^+e^- collisions like 2γ, $\mu^+\mu^-$, $\tau^+\tau^-$, and Bhabha processes, however these events can be suppressed very efficiently [16, sec. 11].

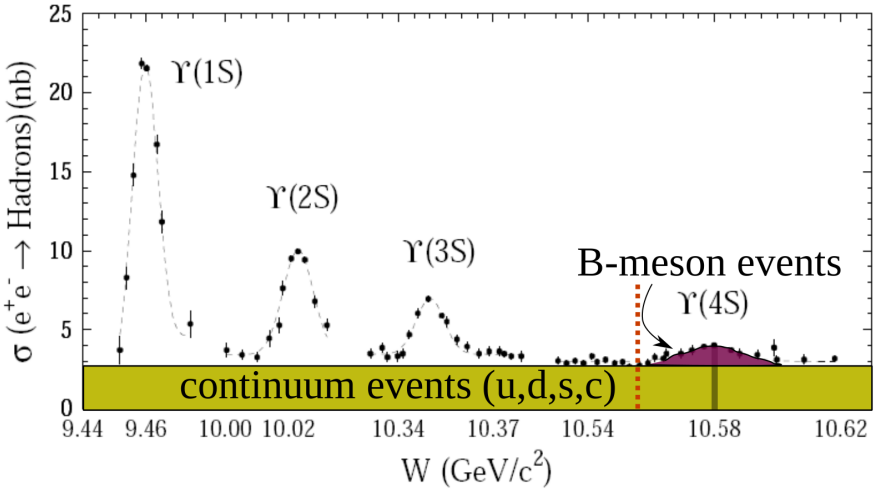


Figure 2.1.: e^+e^- cross section in the $\Upsilon(1S)$ – $\Upsilon(4S)$ region. The red dashed line marks the kinematic threshold for the production of $B\bar{B}$ pairs [14].

of the center of mass frame relative to the laboratory system, which was required to observe the time evolution of B decays. The only interaction point (IP) was surrounded by the Belle detector to detect and identify particles produced by the collisions [13, sec. 2.1].

During its runtime between 1998 and 2010 KEKB reached a peak luminosity of $2.1 \cdot 10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$, which is still the highest luminosity ever obtained by a collider [1, sec. 1.3].

2.2. SuperKEKB

The accelerator was shut down in June 2010 and is currently being upgraded to SuperKEKB to increase the luminosity to $8 \cdot 10^{35}\mathrm{cm}^{-2}\mathrm{s}^{-1}$, which is 40 times as large as the peak luminosity achieved by KEKB [11, sec. 2].

The higher luminosity is being obtained by adopting the Nano-Beam scheme [11, sec. 2], which was first proposed for the Super$B$ factory in Italy. In this scheme the vertical beta function at the interaction point β_y^* is squeezed by almost a factor of 20 with respect to KEKB. For this the final quadrupole magnets have to be positioned closer to the IP, which requires in exchange a larger crossing angle [17, sec. 3.1]. In addition, the beam current in both rings is being doubled to obtain the final design luminosity of $8.0 \cdot 10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$ [17, sec. 2.1.1].

Table 2.1.: Achieved parameters of KEKB and design parameters of SuperKEKB [17].

Machine Parameter	KEKB		SuperKEKB	
	e^-	e^+	e^-	e^+
Beam current (A)	1.64	1.19	3.60	2.61
Energy (GeV) (E_{HER}/E_{LER})	8.0	3.5	7.0	4.0
β_y^* (mm)	5.9	5.9	0.27	0.41
Crossing angle (mrad)	22		83	
Beam emittance (μm)	300	2100	20	10
Beam lifetime (min)	200	150	10	10
Luminosity (10^{34} cm^{-2}s^{-1})	2.11		80	

The increased beam current and the small beta function lead to a shorter beam lifetime due to the Touschek effect[3]. By reducing the asymmetry of the beam energies this problem is mitigated because the Touscheck effect is proportional to E^{-3} [17, sec. 3.2.1]. Therefore, the Lorentz boost is reduced to $\beta\gamma = 0.28$.

The relevant machine parameters of KEKB and SuperKEKB are summarized in Table 2.1.

The layout of the SuperKEKB facility is pictured in Figure 2.2. The two rings are installed side by side and exchange their inner and outer position at the interaction point, where the Belle II detector is placed.

Due to the Nano-Beam scheme more than one order of magnitude lower emittance is required for both electrons and positrons. Low-emittance electrons are produced directly by irradiating a cold cathode with laser photons. In contrast, the emittance of the positrons is much larger because they are produced as secondary particles by shooting electrons on a tungsten target. Therefore, the emittance of the positron beam must be reduced in a damping ring. Afterwards, both beams are injected from a linear accelerator into their respective rings at full energy. Like its predecessor, SuperKEKB will adopt a continuous injection mode. The radio frequency system compensates beam energy losses due to bremsstrahlung. With respect to KEKB it is modified to match the higher beam current.

More information on SuperKEKB can be found in the technical design report of Belle II [17].

[3]The Touschek effect is a loss mechanism due to large angle coulomb scattering inside a bunch.

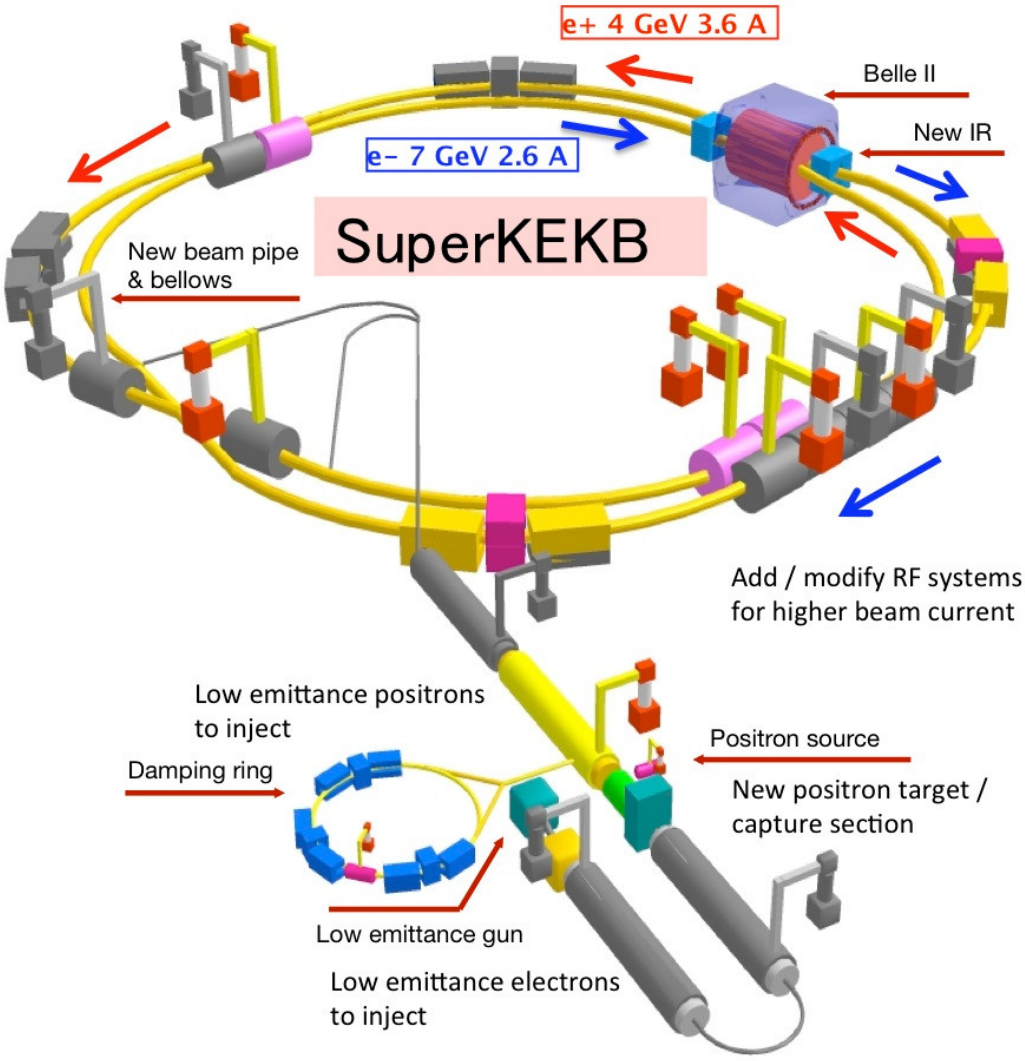


Figure 2.2.: Schematic layout of SuperKEKB [18].

2.3. Original Belle Detector

The single interaction point of KEKB was surrounded by the Belle detector. Like the accelerator, the detector was specifically designed for a precise observation of B meson decays. This includes precise measurement of secondary vertices and good particle identification capabilities. In total, Belle collected a data sample of $(772 \pm 10) \cdot 10^6$ $B\bar{B}$ pairs [17, sec. 3.2.1].

Figure 2.3 shows a schematic side view of the Belle detector.

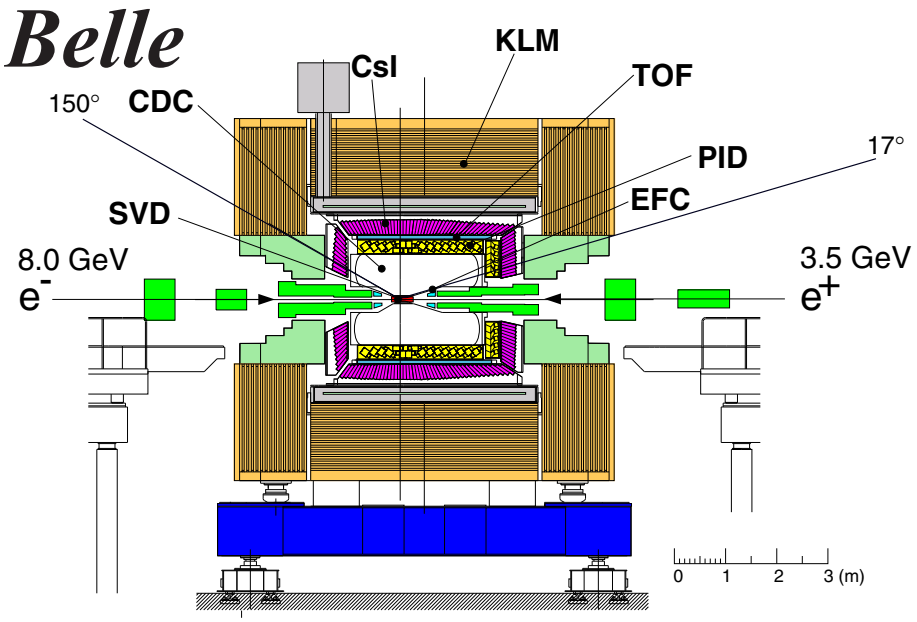


Figure 2.3.: Side view of the original Belle detector [16].

Going outwards from the interaction point the Belle detector consisted of:

- a double-walled Beryllium beam pipe with a radius of 20 mm cooled by He gas;
- radiation-hard Bismuth Germanate crystals used as an extreme forward calorimeter (EFC) as well as a beam and luminosity monitor;
- four layers of double-sided Si strip detectors for precise vertex detection (SVD);
- a drift chamber, which measured momentum and energy loss of charged particles (CDC);
- an Aerogel Cherenkov counter system for particle identification (PID);
- a time-of-flight detector system using plastic scintillation counters (TOF);

- a segmented array of CsI(Tl) crystals with silicon photodiode readout for electromagnetic calorimetry (ECL);
- a superconducting solenoid which provided a homogeneous magnetic field of 1.5 T;
- and an iron support structure, which served as the return path of the magnetic flux and was instrumented with glass-electrode resistive plate counters for K_L and muon detection (KLM).

This summary of the Belle detector is based on the detailed description of the detector in [16].

2.4. Belle II Detector

The original detector is currently upgraded to match the higher luminosity of SuperKEKB. The most important objectives for the upgraded detector are higher physics and background rate tolerance, better physics performance despite smaller Lorentz boost, and improved radiation hardness [11].

Figures 2.4 and 2.9 show the Belle II detector.

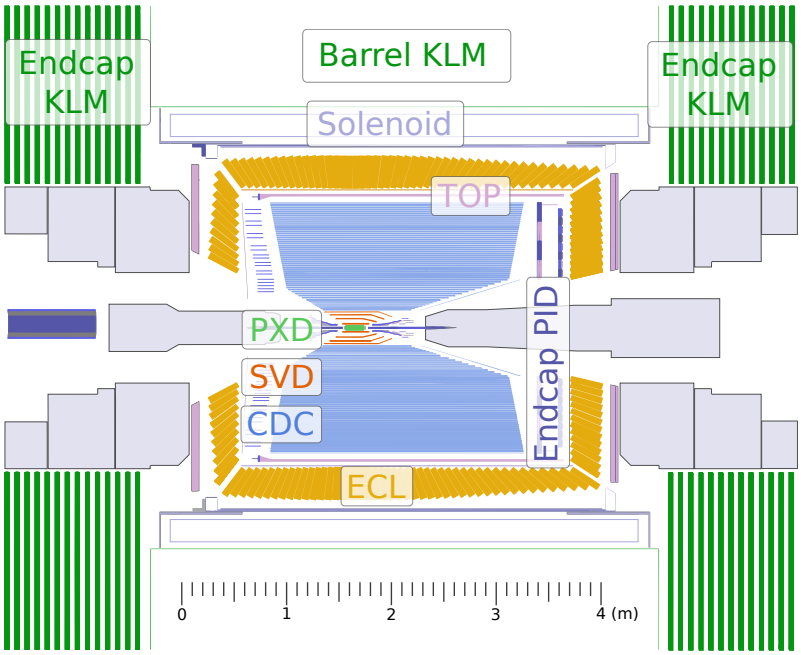


Figure 2.4.: Side view of the upgraded Belle II detector [19].

The following description of the detector is based on the technical design report of Belle II [17] and focuses on the improvements with respect to Belle.

2.4.1. Vertex Detector (PXD and SVD)

The vertex detector is divided into two parts: a two-layered pixel detector (PXD); and a four-layered strip detector (SVD). They are used to precisely reconstruct decay vertices and for finding low-momentum tracks. The design goal is to provide a lifetime resolution as good as or better than Belle, despite the reduced Lorentz boost and the increased background rate.

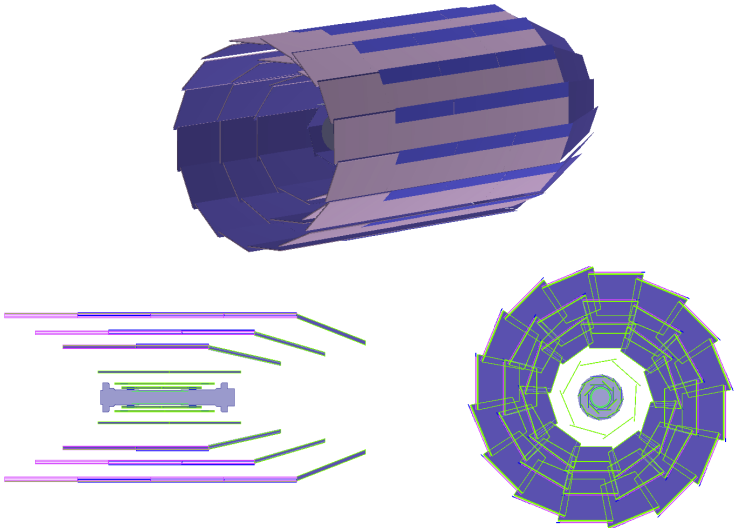


Figure 2.5.: View of the PXD and SVD.

The pixel detector is based on the DEPFET[4] technology, which allows for very thin sensors without the need for additional support or cooling material in the active region of the detector. This reduces the required material budget and multiple scattering, which improves precision vertex reconstruction of B-meson decays. Additionally, the innermost layer of the pixel detector can be located very close to the IP at a radial distance of 14 mm, because the radius of the Beryllium beam pipe will be only 12 mm compared to 20 mm in the past.

In total, the pixel detector uses over eight million pixels leading to a low occupancy[5] of approximately 1% in spite of the long readout time of 20 μs and the increased background rate. To obtain a maintainable event size per readout only pixel clusters are saved which either have enough associated hits in the SVD and CDC or are selected by a multivariate analysis method implemented in FPGAs. In doing so, low-momentum particles which don't reach the CDC and don't have enough hits in the SVD are retained [19].

The SVD provides data to extrapolate the tracks reconstructed in the CDC to the PXD. In addition, it can detect low-momentum tracks without an associated track

[4]DEPleted Field Effect Transistor.
[5]The occupancy is the fraction of channels hit in each triggered event.

in the CDC. It consists of double-sided silicon strip detectors (DSSD) and covers the full $17° - 150°$ acceptance of the Belle II detector. The strips towards the IP are oriented along the z-axis, whereas the strips facing outside are aligned parallel to the r-ϕ plane. As can be seen from the side view in Figure 2.5, the SVD is more strongly instrumented in forward direction.

2.4.2. Central Drift Chamber (CDC)

The Central Drift Chamber (CDC) is used to reconstruct charged tracks and measure their momenta. Moreover, particle-identification information is provided using the specific energy loss of the track inside the volume. On that account, the measured energy losses of each hit of a track are combined in a likelihood for each charged final-state particle hypothesis[6].

The chamber is filled with He(50%):C_2H_6(50%) gas and contains in total 14336 sense wires in a cylindrical arrangement of 56 layers. To obtain information about the z position of a track, the layers are grouped into 9 superlayers which differ in their stereo angles (see Figure 2.6). The inner and outer radius of the CDC is 16 cm and 113 cm respectively.

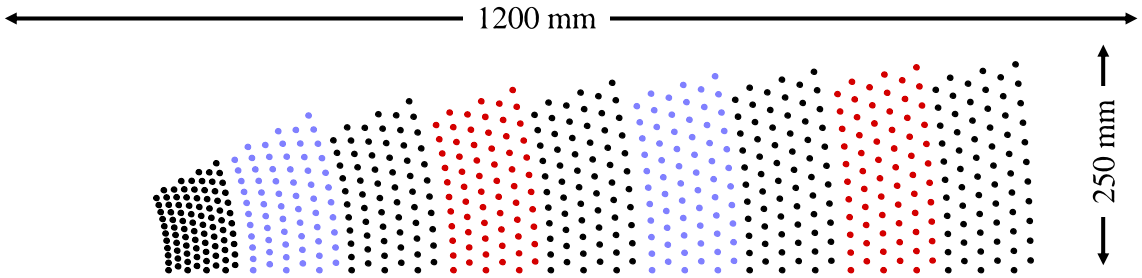


Figure 2.6.: Wire configuration of the CDC [19]. Superlayers with a stereo angle are shown in red and blue.

With respect to Belle the readout electronics system is replaced by a faster one based on ASIC chips with less deadtime. Furthermore, the azimuthal cell-size of the innermost layers is reduced to decrease the occupancy due to the large beam background. The CDC will supply an efficient and reliable trigger signal for the Level-1 (L1) trigger system. Fast 2D and 3D track finding algorithms, which are implemented in FPGAs, will provide the necessary information about track counts and opening angles. Additionally, the new 3D track finder provides a z-trigger signal, which is essential to reduce the large beam background without sacrificing physics events.

[6]Only five charged particles are considered as stable final-state particles inside the detector: Electrons; muons; pions; kaons; and protons.

2.4.3. Particle Identification (TOP and ARICH)

The particle identification system of Belle II consists of a Time-Of-Propagation counter (TOP) in the barrel region and a proximity-focusing Aerogel Ring-Imaging Cherenkov detector (ARICH) in the forward end-cap.

The TOP is an array of 16 quartz bars, which will be installed between the calorimeter's inner surface and the outer CDC cover. It uses less material and space in comparison with the former time-of-flight and aerogel Cherenkov counter systems, therefore the calorimeter response to electromagnetic particles is improved.

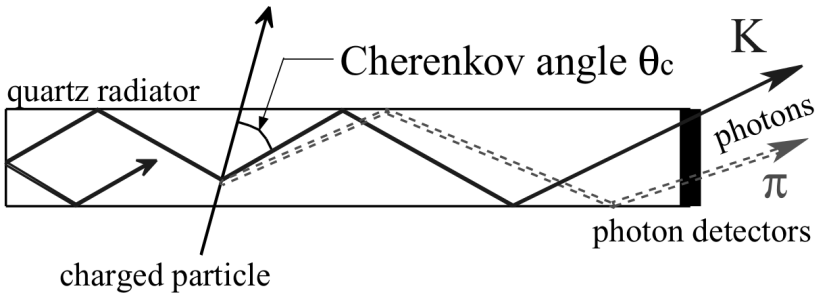


Figure 2.7.: Schematic side-view of TOP counter showing the path of photons due to internal reflection for different final-state particles [17].

Charged particles passing through one of the quartz bars with a velocity higher than the speed of light in quartz create Cherenkov photons. The photons are internally reflected and measured by photon-multipliers at the end surface of the bar in backward direction. The time of propagation of these photons is measured as well as their point of impact. From this three-dimensional information the Cherenkov image can be reconstructed. All detected photons of a track are combined in a likelihood for each final-state particle hypothesis. This likelihood represents the particle identification information of the TOP[7].

Furthermore, the TOP can provide a timing signal with a resolution of a few nanoseconds to the trigger system. In offline reconstruction a time resolution of $< 100\,\mathrm{ps}$ can be achieved.

The ARICH is located outside the CDC in the forward end-cap. Charged particles passing through the ARICH produce Cherenkov photons inside an aerogel radiator, these photons traverse an expansion volume and are finally detected by an array of position sensitive photon detectors, which are capable of detecting single photons in a high magnetic field. The Cherenkov angle of each photon is reconstructed using the photon hit coordinate and the particle trajectory as given by the CDC. Afterwards,

[7] To use this information one has to calculate the likelihood-ratio with respect to another hypothesis.

the velocity of the particle is estimated by combining the measured Cherenkov angles of all detected photons. A non-homogeneous aerogel radiator with an increasing refractive index is used to reduce the spread of the ring image due to emission point uncertainty[8].

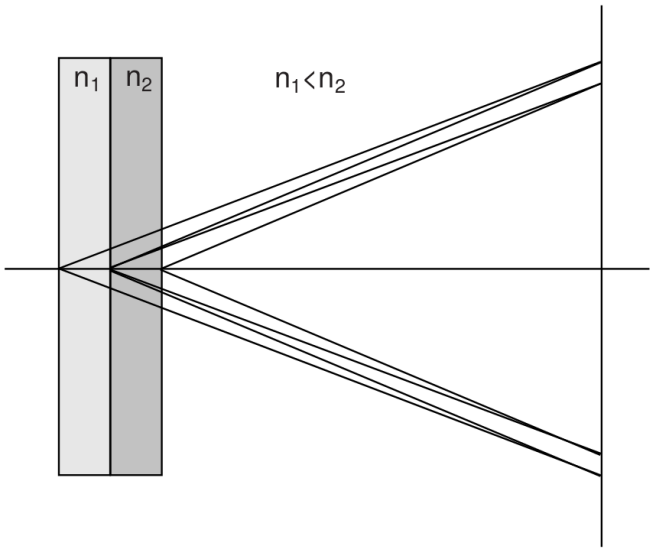


Figure 2.8.: Schematic side-view of a proximity-focusing ring-imaging Cherenkov counter with inhomogeneous radiator [17].

2.4.4. Calorimeter (ECL)

The electromagnetic calorimeter of Belle II reuses the scintillating CsI(Tl) crystals from Belle in the barrel region. The advantages of CsI(Tl) crystals are high light output, short radiation length and good mechanical properties. In the end-caps the CsI(Tl) crystals are replaced with pure CsI crystals which have a shorter scintillation decay time. The whole calorimeter covers a polar angle region of $12.4° - 155.1°$, contains 8736 crystals with a total mass of about 43 tons, and provides 16.1 radiation lengths of material in radial direction.

The scintillation light is read out by photo-diodes, which are glued to the rear surface of the crystals. The analog signal is first pre-amplified and than processed by a digital signal processor which shapes the signal and performs a waveform analysis in real time. With respect to Belle the electronics are upgraded to shorten the shaping time and match the shorter scintillation decay time in the end-caps. This reduces the pile-up noise drastically.

[8]In fact the radiator consists of only two layers with different refractive indices.

The main purpose of the calorimeter is the detection of photons with high efficiency and determination of energy and angular coordinates of these photons with high resolution. This is important for the reconstruction of B-decay products like π^0 and D^* which provide photons in a wide energy range. It is also used to identify electrons based on the shape of the electromagnetic shower and detect K_L^0 together with the KLM.

Furthermore, the calorimeter provides fast trigger signals for the $L1$ trigger system. The total energy is sensitive to physics events with high electromagnetic energy deposit, whereas the amount of isolated clusters is sensitive to multi-hadronic physics events with low energy clusters. In addition, the calorimeter can identify Bhabha and $\gamma\gamma$ events, which are used for online and offline luminosity measurement.

2.4.5. Superconducting Solenoid

The previously described detector components are located inside a superconducting solenoid with a diameter of 3.4 m and a length of 4.4 m, which generates a homogeneous magnetic field of 1.5 T. The solenoid is surrounded by an iron yoke which serves as the return path for the magnetic flux and as overall support structure for all of the detector components.

2.4.6. K_L^0 and μ Detection (KLM)

Like in the original Belle experiment the iron yoke is instrumented with glass-electrode resistive plate chambers (RPC) in the barrel region, whereas the RPCs in the endcaps are replaced with scintillators because of the higher beam background. The iron yoke consists of 14 layers of iron plates and 14 layers of active detector elements. In addition, to the 0.8 interaction lengths of the ECL, the iron plates provide 3.9 interaction lengths or more of material in which K_L^0 mesons can shower hadronically. This outer part of the detector is called KLM and is mostly taken over from the original Belle experiment.

The KLM identifies muons using hits near the crossing of an extrapolated track with a detector layer. Furthermore, it can identify K_L^0 mesons by clustering the hits in its active detector elements and requiring no associated tracks. The clusters only provide information about the direction of the K_L^0 candidates and not about the incident energy, because the number of hits within a cluster fluctuates strongly in the shower development of a K_L^0-nuclear collision. The resolution of the direction can be further improved if the KLM cluster can be associated with a nearby cluster in the ECL.

The end-caps of Belle II are instrumented with scintillator strips. The scintillation light is delivered through wavelength-shifting (WLS) fibers to multi-pixel photon counters (MPPC), which are compact and able to operate in a strong magnetic field.

The KLM can provide trigger signals for muon pair events, which are useful for detector calibration, as well as measuring the efficiencies of other independent trigger signals like from the CDC.

2.4.7. Data Acquisition

Belle II uses a two-level trigger system, with an FPGA-based Level-1 (L1) trigger decision and a high level trigger (HLT) farm.

At the design luminosity the nominal average L1 trigger rate is expected to be up to 30 kHz. For each trigger signal the data is read out by the data acquisition system from the detector front-end electronics. The front-end boards are placed near or inside the detector structure and are based on FPGAs. They digitize the data and perform simple data reduction like zero suppression.

Afterwards, the data from the front-end boards is transferred through the COmmon Pipelined Platform for Electronics Readout (COPPER) to the event builder PCs, which merge the data from the different sub-detectors. Further data compression, formatting and reduction is applied. Finally, the high level trigger farm reconstructs the event and performs a physics-level event selection using the event data from all sub-detectors. The final rate of raw events which are written to the storage is expected to be between 6 kHz and 10 kHz. The HLT consists of multiple Linux-based PC clusters and runs the Belle II Analysis Software Framework (BASF2) [20].

2.4.8. Offline Reconstruction and Monte Carlo Production

The same software framework is used in offline reconstruction, Monte Carlo production, and physics analysis. After machine-dependent calibration parameters are determined, the raw data is reconstructed and stored at the KEK computing centre. Monte Carlo production and reconstruction will be distributed to computing facilities all over the world. Belle II is going to use a grid-based approach like the LHC experiments to share computing resources at the different computing facilities (so-called grid-sites).

The reconstructed information is saved in *mDST* files[9], which are stored at the grid-sites. The final physics analysis is performed on analysis-specific ntuple-files, which can be created from the *mDST* files.

[9] A reduced and compressed form of the data summary tables, which store the raw-data.

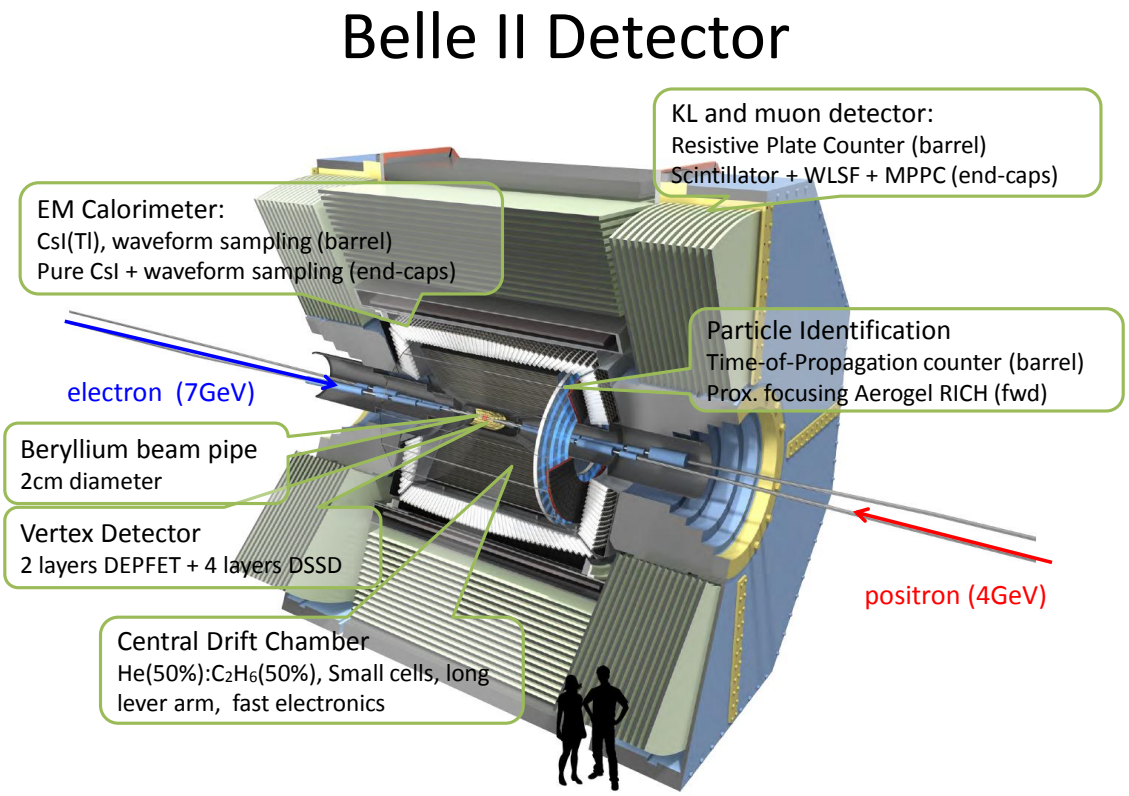


Figure 2.9.: Perspective view of the Belle II detector with comments summarising the most important features [21].

3. Multivariate Classification

Many problems in science, engineering, and economics can be can be reduced to a multivariate classification task. An object described by a feature vector $\vec{x}$ has to be classified as either signal or background.

A multivariate classifier (MVC) maps the multidimensional feature vector $\vec{x} \in \mathbb{R}^n$ onto a one-dimensional test-statistic. The test-statistic is designed to separate signal and background from one another.

The FEI relies heavily on multivariate classification. In particular, for each final-state particle and decay channel a binary classifier is trained, which can distinguish between correctly reconstructed candidates (signal) and incorrectly reconstructed candidates (background). The feature vector usually contains quantities derived from the kinematic and the fitted track of the candidate. This chapter contains an introduction into the used terminology, mathematical tools and classification methods.

3.1. Statistics

The distribution of signal and background is determined by multidimensional probability density functions (PDF). In the remainder of this chapter the distribution of signal and background is denoted by $f(\vec{x}|H_0)$ and $f(\vec{x}|H_1)$ respectively. One can interpret $f(\vec{x}|H)\,\mathrm{d}\vec{x}$ as the probability to encounter a candidate in the hypercube spanned by $\vec{x} \pm \mathrm{d}\vec{x}$. As a consequence, the PDFs are normed to

$$\int f(\vec{x}|H)\,\mathrm{d}\vec{x} = 1.$$

The distribution of signal and background in the test-statistic can again be described in terms of PDFs $T(\vec{x}|H)$. If transformed properly the test-statistic corresponds to the signal-probability. All classifier outputs used by the FEI are transformed to a probability as discussed in section 4.3.3.

To classify a candidate as signal or background a critical value T_c has to be chosen. If the output of the MVC is above T_c the candidate is considered as signal, otherwise

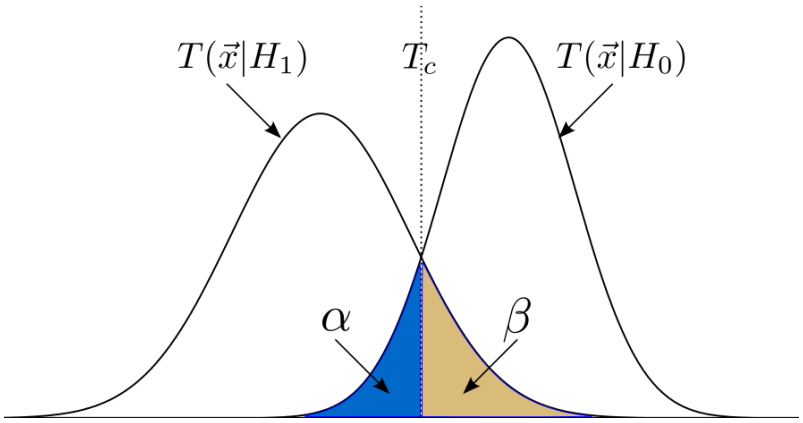


Figure 3.1.: PDFs of signal $T(\vec{x}|H_0)$ and background $T(\vec{x}|H_1)$ candidates in the test-statistic. An arbitrary critical value T_c was chosen to separate signal and background. The areas corresponding to the error of first and second kind are inked in blue and brown respectively.

as background. Yet this is just a convention. Usually the classification is not perfect, hence some candidates are misclassified. The error of the first kind α states the fraction of signal, which is wrongly classified as background

$$\alpha = \int_{-\infty}^{T_c} T(\vec{x}|H_0).$$

Contrastingly, the error of the second kind β states the fraction of background, which is wrongly classified as signal.

$$\beta = \int_{T_c}^{\infty} T(\vec{x}|H_1).$$

Figure 3.1 shows signal and background PDFs as well as the areas corresponding to the errors of first and second kind.

It is useful to quantify the performance of a classifier using the signal-efficiency and signal-purity instead of the error of first and second kind, due to their better interpretability. The signal-efficiency ϵ states the fraction of real signal candidates classified as signal

$$\epsilon = \int_{T_c}^{\infty} T(\vec{x}|H_0) = 1 - \alpha.$$

And the signal-purity p states the fraction of the candidates classified as signal, which are classified correctly

$$p = \frac{N_S \int_{T_c}^{\infty} T(\vec{x}|H_0)}{N_S \int_{T_c}^{\infty} T(\vec{x}|H_0) + N_B \int_{T_c}^{\infty} T(\vec{x}|H_1)},$$

where N_S is the number of signal and N_B is the number of background candidates in the sample. In the same way one can define the background-efficiency and background-purity, however these quantities are not used in this thesis, hence the prefix "signal" is omitted in the following.

The optimal critical value T_c depends on the desired efficiency and purity. Additional information on statistics and choosing T_c can be found in [22].

3.1.1. Optimal and Near-Optimal Test-Statistics

If the PDFs $f(\vec{x}|H_0)$ and $f(\vec{x}|H_1)$ are known in analytical or numerical form, the optimal test-statistic is given by

$$T(\vec{x}) = \frac{f(\vec{x}|H_0)}{f(\vec{x}|H_1)}.$$

The Neyman-Pearson Lemma [23] states that this test-statistic leads to the smallest error of second kind for a fixed error of first kind.

Yet it is not possible to calculate the PDFs in an analytical form in the use case presented later. Nor is it possible to numerically approximate the complete PDFs due to the curse of dimensionality. Between twenty and thirty features are typically employed by the classifiers of the FEI, even with a rough approximation using ten bins in every dimension, one would need to simulate astronomical amounts of data to obtain enough statistics in all 10^{20} to 10^{30} bins.

In consequence, many methods were developed to obtain a near-optimal test-statistic with less computational effort. Simple linear discrimination functions were around as early as 1936 [24], more advanced methods like artificial neural networks followed a few years later [25], were discarded in 1969 because they were inapplicable to simple separation problems like XOR [26] and resurrected in 1986 with the invention of the backpropagation algorithm [27], which lifted this limitation. In the last years the deployment of stochastic gradient-boosted decision trees [28] became popular in high energy physics experiments, for instance in the recent discovery of the Higgs boson [29].

All methods considered in this thesis have to be trained on a so-called training-sample for which the classification is known. The training-sample is created by a Monte Carlo simulation of the underlying physics processes and followed by a detector simulation. The employed method adapts its internal parameters and structure to minimize a given loss-function L on the provided training-sample.

To perform well on an independent test-sample it is crucial that the trained MVC has the ability to generalize. An MVC looses this ability if it uses statistical fluctuations

of the training-sample to minimize the training-error, this is termed as over-training. In consequence, the model is more complex than necessary and the performance on an independent test-sample is often very poor. A well-trained MVC describes all important effects in the data with as few parameters as possible. Many MVC methods allow to adjust the complexity of the model using hyper-parameters[1].

The main difference between the performance of MVC methods arises from their different strategies, so-called regularization techniques, to inhibit over-training. Typically these techniques reduce the effective number of free parameters of the method by smoothing internally used distributions, introducing penalty terms in the loss-function or simplifying the structure of the method.

Although the FEI can utilize an arbitrary MVC, usually it employs either an commercial Artificial Neural Network implementation or a specially adapted Stochastic Gradient Boosted Decision Tree as MVC. In the following both methods are discussed in more detail.

3.2. Artificial Neural Network

Artificial neural networks (ANN), or more precisely multi-layer perceptrons, consist of several layers of neurons. Each neuron is connected to all neurons in the preceding and succeeding layer, each connection has a weight w_{ij}. The neurons in the first layer correspond to the input features, the neuron in the last layer produces the final output of the classifier. The resulting network topology is depicted in Figure 3.2.

The response of the network to an input vector is calculated by propagating the input vector from the first layer to the last. Therefore, this type of ANN is called a feed-forward network. Other kind of ANNs exist, but are not discussed here. During the propagation each neuron transforms its received weighted inputs from the preceding layer into a single output using an activation function.

The weights are adjusted to create a near-optimal test-statistic minimizing the given loss-function for the given training-sample. Initially the weights are randomly chosen using a standard-normal distribution. Afterwards, the output of the network for a candidate from the training-sample is calculated and the induced error in the loss-function is back-propagated through the network using the chain rule. The algorithm corresponds to a steepest-descent minimization of the loss-function in the high-dimensional space of the weights

$$\Delta w_{ij} = -\eta \frac{\mathrm{d}L}{\mathrm{d}w_{ij}}.$$

[1] The term hyper-parameter is used to distinguish parameters of the method from parameters of the model created by the method.

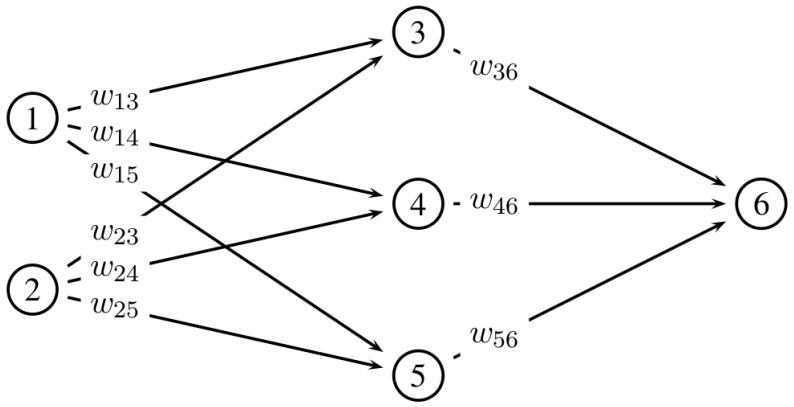


Figure 3.2.: Simple feed-forward network with two input features, three hidden neurons and a single output.

The parameter η is the learn-rate of the network and crucial for a fast but stable convergence into a near-optimal minimum. If the learn-rate is too small the algorithm will slowly converge into a local minimum far away from the optimum. On the other hand, if the learn-rate is too large the algorithm does not converge at all. Different methods were developed to adjust the learn-rate during the training automatically like the BFGS algorithm [22] and resilient back-propagation.

The complexity of an ANN can be adjusted by modifying the maximum number of iterations, the number of layers in between the input and output layer, and the number of neurons in these layers. Furthermore, different activation functions and loss-functions can be employed.

The NeuroBayes package [30], which will be used in the FEI, is based on an ANN. It includes extensive pre-processing of the input features and advanced weight-update strategies like the BFGS algorithm.

3.3. Stochastic Gradient Boosted Decision Tree

A decision tree divides the training-sample into disjunct regions dominated by either signal or background. The training-sample is consecutively divided into two parts by a simple cut on an input feature. Each cut is chosen to maximise the separation gain, as calculated by a separation measure like the entropy difference. Hence, the method can be represented by a tree-structure with a simple cut at each branching-point as shown in Figure 3.3. The final classifier output for a candidate is the signal-fraction of the corresponding region[2]. In the following the branching-points and regions are called nodes and leafs, respectively.

[2]From a mathematical point of view the regions are hyper-cubes in the feature space $\mathbb{R}^n$

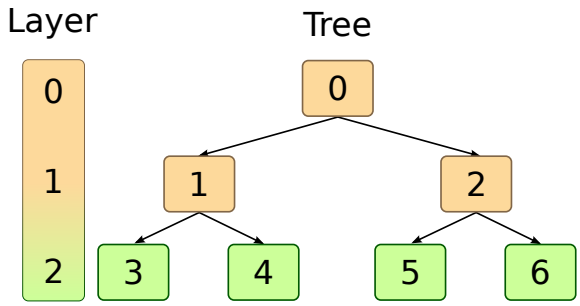


Figure 3.3.: Decision tree with two consecutive cuts at the branching-points/nodes (in brown) and the resulting four regions/leafs (in green).

A single decision tree is very prone to over-training. A boosted decision tree avoids over-training by strongly limiting the depth of a single tree (or equivalently the number of consecutive cuts). In consequence, a single tree is a so-called weak-learner and separates the signal and background only roughly.

By using many weak-learners one can construct a well-regularized classifier with great separation power. A boosting algorithm attributes a weight to each candidate and trains a weak-learner. Afterwards, the weights of the wrongly classified candidates are increased and another weak-learner is constructed. Usually this process is repeated a few hundred times. The final classifier output is calculated as a weighted sum of the outputs of the individual weak-learners.

In the case of decision trees, the boosting algorithm constructs, figuratively speaking, a whole forest. Therefore, boosted decision trees belong to the category of ensemble methods. Different boosting algorithms are available.

For this thesis the gradient boost as described in [28] was implemented. The new weights w_i of the candidates are calculated as the negative gradient of the loss-function L with respect to the current classifier output $\sigma(\vec{x}_i)$

$$w_i = -\frac{\partial L}{\partial \sigma(\vec{x}_i)}.$$

Thereby, the algorithm detects the candidates in the sample which are hard to classify and increases their weight. In effect, the weak-learners concentrate on the difficult candidates and do not learn statistical fluctuations of the other candidates. In fact, after a few boosting iterations most of the candidates become irrelevant and can be ignored, to speed up the training.

To further increase the ability of the boosted decision tree to generalize, only a random subset of the candidates is used for the construction of the individual trees, hence the name stochastic gradient boosted decision trees (SGBDT).

The complexity of SGBDTs can be adjusted by modifying the number of trees, the depth of the trees and the fraction of candidates used to construct the trees.

If NeuroBayes is not available, the FEI uses a speed-optimized SGBDT as MVC.

4. Belle II Analysis Software Framework

The software framework for the Belle II experiment (BASF2) was rewritten from scratch to obtain better usability and code quality with respect to Belle. It is used for online as well as offline data handling. The framework is written in C++11 and Python 2.7. In addition, popular third-party libraries like evtgen [31], Geant4 [32] and ROOT [33] are employed. The basic concept of BASF2 adopts ideas from the GAUDI architecture [34] used by LHCb [35], ATLAS [36] and other high energy physics experiments; as well as from the ILC and Belle software framework. The architecture of BASF2 is visualized in Figure 4.1. The remainder of this chapter describes the structure of BASF2 in general, with emphasis on the parts developed for the FEI.

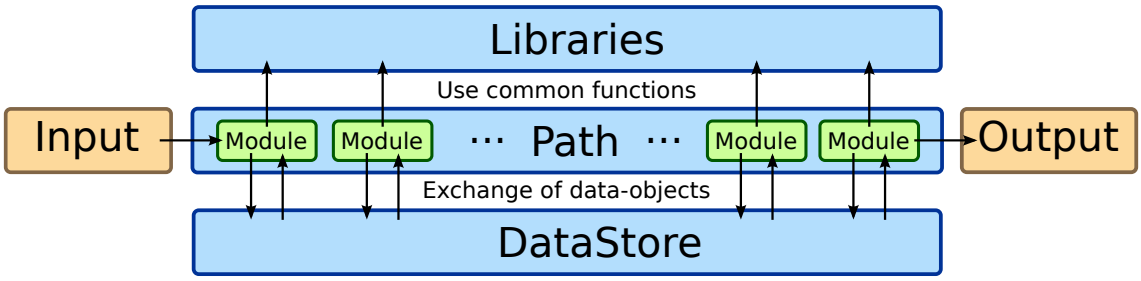


Figure 4.1.: Modules operating event-by-event on the data. They are chained into a path and executed for each event. The modules use functionality provided by libraries and share information encoded in data-objects via the common `DataStore`.

4.1. General Structure

The framework is divided into packages, each covering a different aspect of data processing like:

- data acquisition (**daq**);
- Monte Carlo event generation (**generators**);
- general detector simulation (**simulation**);
- sub-detector simulation, unpacking[1] and reconstruction (**ir, pxd, svd, cdc, ecl, top, arich, bklm**);
- track reconstruction (**tracking**);
- visualization of individual events inside the detector (**display**);
- and physics analysis (**analysis**).

The packages contain libraries, modules and data-objects. The libraries are implemented in C++ and tested with the google unit-test framework. They provide functionality which is shared between different modules.

Modules are small processing blocks built on top of the libraries. They operate on the data event-by-event and perform self-contained tasks. A chain of such modules is called a path. An illustrating example is given in the next section. Most of the modules are implemented in C++, but it is also possible to implement modules in Python.

Information which is shared between the modules is encoded in data-objects. The data-objects are stored in a common `DataStore`, which every module can read and write. Typical examples for data-objects are `Track`, `ECLCluster`, `Particle` and `ParticleList` objects. In addition, the `DataStore` keeps track of relations between the data-objects. For example between associated `Track` and `Particle` objects. The data-objects can be serialized using ROOT. Serialization is important to support persistence, multi-processing, distributed computing and real-time monitoring of the HLT farm.

To use BASF2 the user has to provide a steering file written in Python. In the steering file the path is created and filled with modules, afterwards the path is processed. Input and output data files can be read and written by adding the `RootInput` and `RootOutput` modules at the beginning and end of the path, respectively. The input and output data file names can be either passed to these modules as arguments or specified via command line arguments. One has to distinguish between two levels of code execution. Firstly, the Python steering file is executed and builds the path (**steering level**). Secondly, the path is applied to all specified events (**path level**), usually the code executed at this level is written in C++.

[1] Conversion of detector data into BASF2 data-objects.

4.2. Illustrating Example

The steering file in Listing D.1 shows a simple reconstruction of the decay $D^0 \to K^-\pi^+$. All the basic tools needed for the FEI already appear in this example. The steering file expects generic $B\bar{B}$ Monte Carlo data files as command line argument. The Monte Carlo data is created by a separate steering file and contains among others serialized `Track`, `ECLCluster`, `KLMCluster` and `MCParticle` data-objects. In the following the listing is explained step by step to illustrate the abstract concepts of BASF2 described in the previous section.

```
1 from basf2 import *
2 from modularAnalysis import *
```

At first all identifiers from the `basf2` and `modularAnalysis` Python modules are loaded into the current namespace. `basf2` provides direct access to the modules using the `register_module` function, whereas `modularAnalysis` provides convenience functions to add modules to the current path. In this example only the `RootInput` and `RootOutput` modules are added directly using the `register_module` function. The other modules described below are inserted into the path by corresponding convenience functions provided by `modularAnalysis`.

```
3 # Read data from files given via command line argument.
4 analysis_main.add_module(register_module('RootInput'))
```

The `RootInput` module is registered and added to a path named `analysis_main`. The module transfers the data-objects from the provided Monte Carlo data into the `DataStore`.

```
5 # Load Belle II detector parameters file.
6 loadGearbox()
7 # Create FSP hypotheses for tracks and clusters.
8 loadReconstructedParticles()
```

The Belle II detector parameters are loaded by the `Gearbox` module. The `ParticleLoader` module creates for each `Track` in the `DataStore` particle hypotheses for K, π, p, e and μ, the so-called charged final-state particles (FSP). Furthermore, for each `KLMCluster` a K_L^0 particle hypothesis is created, as well as a γ particle hypothesis for each `ECLCluster` without an associated track. The particle hypotheses are represented by `Particle` data-objects.

```
9 # Collect all kaon and pion hypotheses into
10 # ParticleLists and apply cuts on the PID information.
11 selectParticle('K-','Kid > 0.01')
12 selectParticle('pi+','piid > 0.01')
```

Afterwards, K and π candidates are selected by the `ParticleSelector` module and a soft cut on the PID information given by the likelihood-ratio of the individual outputs of the PID subdetectors is applied. Internally a `ParticleList` data-object, which contains references to the corresponding `Particle` data-objects, is created for each selected particle type. Moreover, associated `ParticleLists` are created, containing references to the charge conjugated `Particle` data-objects.

```
13 # Combines the hypotheses to D0 candidates and
14 # cut on the invariant mass of each candidate.
15 reconstructDecay('D0 -> K- pi+','1.6 < M < 2.0',
16                  persistent=True)
17 # Match the MC Truth of all D0 candidates
18 matchMCTruth('D0')
```

The `ParticleCombiner` module combines the K and π candidates to D^0 candidates if the invariant mass of the candidates falls within a mass range near the nominal mass of the D^0. The candidates are stored in a `ParticleList` data-object. Additionally, the `ParticleList` is marked as `persistent`, thereby it is written into the final ROOT file by the `RootOutput` module. The decay is specified by a `DecayDescriptor` string. Details on the syntax of `ParticleLists` and `DecayDescriptors` can be found in the Appendix in section A.1.

After that the `MCMatching` module sets a relation between the `Particle` data-object of each D^0 candidate to the corresponding `MCParticle` data-object.

Like in all modules the charge conjugate is implicitly implied. It means in effect that `reconstructDecay('D0 -> K- pi+')` creates two `ParticleLists`. One for candidates of the channel $D^0 \to K^-\pi^+$ and another for $\bar{D}^0 \to K^+\pi^-$. Likewise `matchMCTruth('D0')` and all further modules operate automatically on the $\bar{D}^0$ `ParticleList` as well.

```
19 # Fit the secondary vertex of all D0 candidates and
20 # apply a soft cut on the p value of the fit.
21 fitVertex('D0',0.0001)
```

Thereafter, the `Geometry` module loads the geometry of the Belle II detector, which is currently needed to fit the secondary vertex of the D^0 with the following `ParticleVertexFitter` module. BASF2 contains two different vertex-fitting libraries: KFit and Rave [37]. At the moment Rave is used by default. A soft cut on the p value of the fit is applied to remove candidates which are combined out of tracks without a common vertex.

```
22 # Train the standard MVC
23 # If the method is already trained, calculate the
24 # signal probability and save it together with its
25 # mass and mcStatus in an ntuple file.
26 if isTMVAMethodAvailable():
27     applyTMVAMethod('D0')
28     variablesToNTuple('D0',['getExtraInfo(isSignal)',
29                             'mcStatus', 'M'])
30 else:
31     trainTMVAMethod('D0',['M','p','pz','dr','dz',
32                           'chiProb','daughter(0,Kid)',
33                           'daughter(1,piid)'])
```

Then standard[2] MVC is trained by the `TMVATeacher` module. The MVC separates correctly reconstructed D^0 from background candidates using provided `Variables` like the invariant mass `M` or the momentum `p`. Details on `Variables` are described in the Appendix in section A.2.

If the weight-file of the MVC is already available, the `TMVAExpert` module is added to the path instead of `TMVATeacher`. `TMVAExpert` calculates the signal probability for each `Particle` in the given `ParticleList`. As a consequence, the user has to run the steering file twice: in the first pass the MVC is trained and in the second pass the probability for each candidate is calculated.

Additionally, in the second pass the `VariablesToNtuple` module writes the signal probability, which is stored in the `ExtraInfo` field of each `Particle` data-object, together with the mass and the Monte Carlo status flag of the candidate into an ntuple-file. This is useful if a user is only interested in some properties of the reconstructed candidates.

```
34 # Write out the D0 candidates.
35 analysis_main.add_module(register_module('RootOutput'))
```

The `RootOutput` module on the other hand writes out all data-objects stored in the `DataStore` to a ROOT file if they are marked as persistent, like it is the case for the D^0 `ParticleList`.

```
36 # Execute the path.
37 process(analysis_main)
```

The code execution on steering level ends here, up to this point only Python code was executed. Now the transition to C++ code execution on path level is performed. The generated Python path object is translated to C++ by the boost python library.

[2]Currently the standard MVC is the FastBDT method, which is discussed later in this chapter.

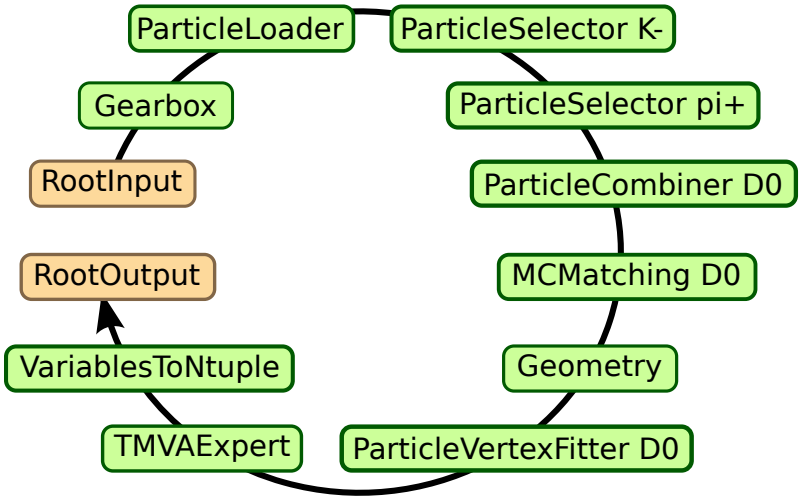


Figure 4.2.: Path which is created and executed by the steering file in Listing D.1 in the second pass.

All modules contained in the path are instantiated using the corresponding C++ classes. Afterwards, the modules are called one after another for every event.

Over ten different modules are employed by the path which is created by the steering file. The whole path is shown in Figure 4.2.

Although the basic tools shown in the example are already sufficient to implement the FEI, it would be a tedious, complicated and error-prone task. Therefore, the FEI is implemented using a meta-framework at steering level which resolves dependencies between decay channels automatically. It is described in Appendix B.

4.3. Multivariate Classification

The Belle II software framework employs TMVA to perform multivariate classifications. The library `TMVAInterface` provides an interface to TMVA inside BASF2. The library was originally developed for the FEI, but is already adopted by other groups for implementing techniques like flavour tagging.

Based on the library two BASF2 modules were developed: the `TMVATeacher` and the `TMVAExpert` module, which train and apply a MVC respectively. Since the FEI relies heavily on MVCs, TMVA and the used MVCs are described in more detail below.

4.3.1. TMVA

TMVA is a standard component of the ROOT library, therefore it is well tested, well documented and widely used in particle physics. In addition, it includes a large number of different data analysis methods, among others Boosted Decision Trees, Artificial Neural Networks and Support Vector Machines.

It is also possible to provide custom TMVA methods, which are automatically loaded at runtime via the plugin interface of ROOT. Therefore, one can easily employ commercial software like NeuroBayes without recompiling BASF2 or ROOT.

TMVA provides a rich set of preprocessing methods for the used variables like: normalization; decorrelation; transformation to a Gaussian distribution; and principal component transformation. Furthermore, it can apply boosting, bagging and categorization on arbitrary methods. The given sample is automatically split into a training and a test sub-sample.

In particular, the FEI uses TMVA to create probability density functions (PDF) of the output of the used classifier for signal and background candidates. TMVA converts the output of the classifier into a probability using these PDFs.

A complete discussion of all TMVA features and options can be found in [38].

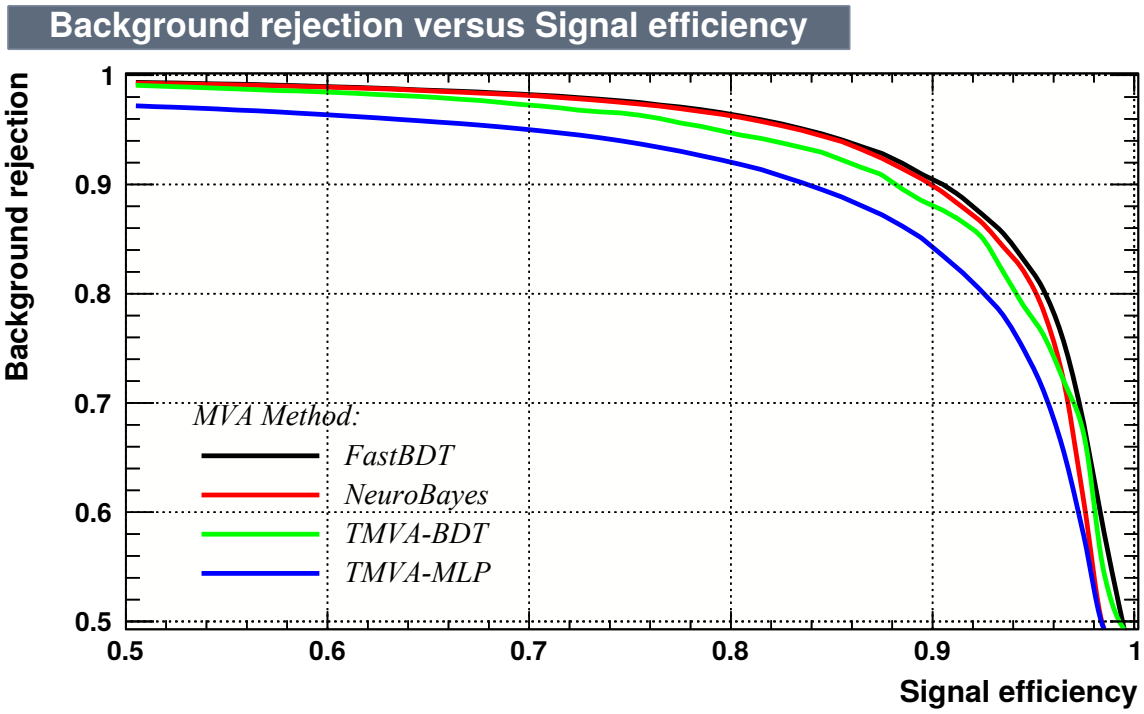
4.3.2. TMVA Control Plots

TMVA automatically creates control plots for each training. The most useful plots are shown in Figure 4.3. The plots were generated by the example steering file in Listing D.1, with small modifications to train further methods in addition to the default one. The receiver operating characteristic and over-training control plots are included in the automatic reporting system of the FEI. Further control plots can be created on demand (see Appendix A.3 for some examples).

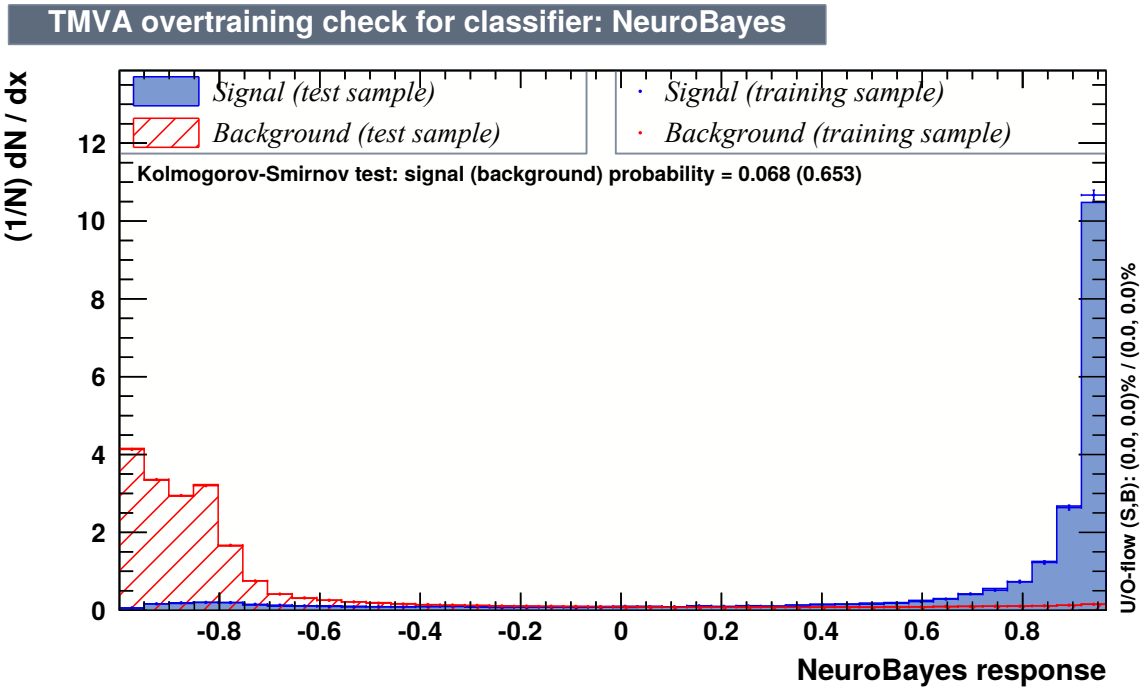
a) Receiver operating characteristic

The receiver operating characteristic (ROC) plot illustrates the performance of binary classifiers. It shows the performance of the classifications in terms of signal efficiency and background rejection. Mathematically the receiver operating characteristic is a curve in the two-dimensional signal efficiency, background rejection space, which is parametrized by the cut applied on the test-statistic of the classifier. The integral below the curve is a measure for the discrimination power of the classifier.

Figure 4.3a shows a comparison of the TMVA built-in Boosted Decision Tree (TMVA-BDT), the TMVA built-in Multi-Layer Perceptron (TMVA-MLP), a speed-optimized



(a) Receiver operating characteristic



(b) Overtraining check

Figure 4.3.: TMVA control plots for Listing D.1

Boosted Decision Tree classifier written especially for the FEI and the commercial NeuroBayes classifier. The closer the ROC curve of a classifier is to the (1,1) point in the plane the bigger its integral and the better its separation. Hence, the FastBDT and NeuroBayes methods offer a better separation than the built-in classifiers of TMVA in this example, at least for signal efficiencies below 95%. For instance, a cut on the classifier output which suppresses 95% of the background leads to a signal efficiency of 80% using TMVA-BDT, 70% using TMVA-MLP and 84% using FastBDT or NeuroBayes.

b) Overtraining check

The overtraining check plot compares the distributions of the classifier output on the training and test sample, each for signal and background. If the classifier in question is overtrained, its output distributions will differ for training and test data. A Kolmogorov-Smirnov test[3] is performed to check if the distributions for training and test sub-samples are statistically compatible. Furthermore, the shapes of the signal and background distributions are a good measure for the quality of the separation. One expects mono-modal[4] and smooth distributions which are well separated from each other.

In the example plot 4.3b the distributions are compatible for a significance level of 5%. The signal and background distributions are well separated and peak at -1 and 1 respectively. Hence, the classifier performs well in this example.

4.3.3. Transformation of the classifier output to a probability

The FEI assumes that the output of each employed classifier is a probability. Hence, it is important to ensure this property. Fortunately, TMVA can create PDFs for the signal S and background B distributions of the classifier output σ, in addition it is possible to apply different smoothing algorithms to these PDFs.

The final transformed classifier output is given by

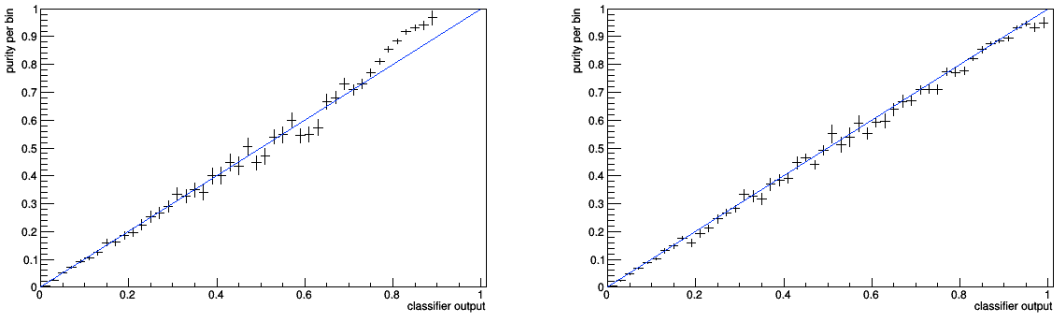
$$p = \frac{S(\sigma)}{S(\sigma) + B(\sigma)},$$

which is a probability by construction. Differences in the signal-fraction of the data used for training and application are taken into account by scaling the PDFs accordingly.

[3] A statistical test calculating the probability to obtain the observed or a greater deviation of two samples under the assumption that both samples are generated from the same distribution.

[4] A distribution is mono-modal if it has only one peak.

The interpretability of the classifier output as a probability is validated by plotting the purity of the candidates over the binned transformed classifier output. The output corresponds to a probability if all data-points in such a plot are compatible with a line through the origin with slope 1. This plot is included in the automatic reporting system of the FEI for each particle and decay channel.



(a) Purity over raw classifier output. (b) Purity over transformed classifier output.

Figure 4.4.: Purity over FastBDT output for example Listing D.1.

Figure 4.4 shows the purity over classifier plot for the FastBDT method. Although the FastBDT output should already be a probability, this is sometimes not fulfilled over the entire output range as can be seen in Figure 4.4a. The TMVA transformation enforces the interpretability as a probability over the entire output range.

4.3.4. FastBDT

For each decay channel the FEI trains a separate MVC with a few million samples for signal and background. In total over 100 methods have to be trained. Unfortunately, the TMVA built-in methods are not designed to minimise the duration of the training. Therefore, the standard MVC in BASF2 is a speed-optimized stochastic gradient-boosted decision tree (FastBDT), which is integrated in TMVA via its plugin mechanism.

The gradient boosting algorithm is described in the original paper [28], boosted decision trees in general are discussed in Chapter 3. The FastBDT implementation is up to 50 times faster than the equivalent stochastic gradient boosted decision tree implemented in TMVA, which is based on the same paper. Differences in the quality of the separation between the FastBDT and TMVA-BDT arise from a different separation measure used for the determination of the cuts. In addition, the calculation of the FastBDT response is up to 10 faster than the TMVA implementation.

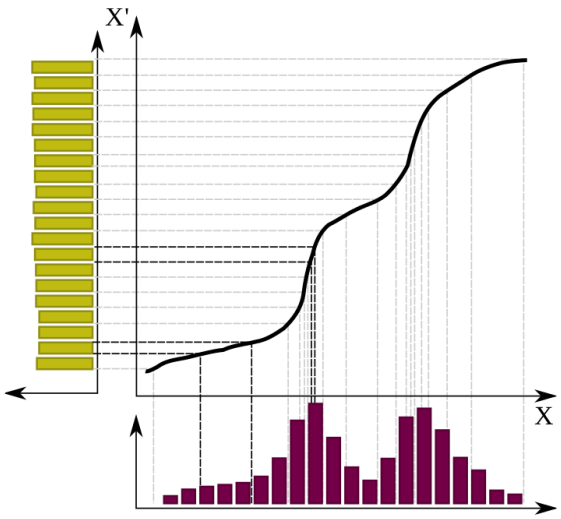


Figure 4.5.: Transformation of an arbitrary distribution to a uniform binning using the cumulative distribution [14].

Speed optimizations in the implementation

The input features are represented by floating point numbers. At first the FastBDT implementation maps every feature onto an integer-based binning, such that every bin contains the same number of entries. After this transformation the algorithm is independent from the shape of the input data and slow floating point calculations can be replaced by fast integer operations. An example mapping of an arbitrary distribution to a uniform binning is visualized in Figure 4.5. Furthermore, the binning defines the grid which is used to search for the optimal cut at each node in the tree.

The FastBDT implementation determines the cuts simultaneously for all nodes in the same layer of the tree, in contrast to a traditional implementation of a decision tree, which determines the cuts node after node. This allows to access the events in linear order, which is faster due to the cache of the CPU. In other words, instead of accessing the events in random order and train node after node, the nodes are accessed randomly whereas the events are processed in a linear order matching the memory layout.

There are further optimisations in the implementation which go beyond the scope of this thesis. However, a common feature of all speed optimisations implemented in the FastBDT algorithm is the principle of locality. Modern CPU caches are optimised for this principle:

- Temporal locality (a value is frequently accessed) – The cut optimisation at each node can be represented as a series of counting and summing operations. These operations frequently update a single value. In the optimal case this value never leaves the CPU cache during the computation.

```
int a = 0;
int b = 0;
for(int i=0; i<1e9; ++i) {
   if(rand()%2 == 0) a++;
   else              b++;
}
cout<<a<<" "<<b<<endl;
```

(a) Straight-forward implementation – Execution time 10.4 sec

```
int a[] = {0,0};

for(int i=0; i<1e9; ++i) {
    a[rand()%2]++;

}
cout<<a[0]<<" "<<a[1]<<endl;
```

(b) If statement replaced by array lookup – Execution time 7.4 sec

Figure 4.6.: Branch locality optimisation example which increments two counters randomly. Both codes were compiled with the highest optimization level available. The optimised version is 30% faster.

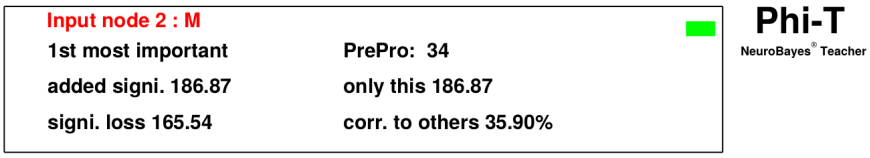
- Spatial locality (a value is accessed after a nearby value was accessed) – Events are always accessed in linear order. Therefore, the spatial locality is maximally exploited.
- Branch locality (few conditional branching instructions) – Conditional branch instructions (e.g. *if* and *switch* statements) are avoided and replaced by array lookups using the integer-based binning of the features. See Figure 4.6 for an example of this technique.

4.3.5. NeuroBayes

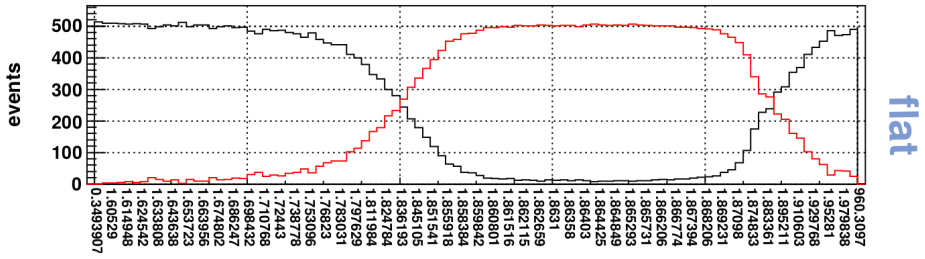
NeuroBayes is a multivariate analysis method based on an artificial neural network (see Chapter 3). It was originally designed for high energy physics. It is orders of magnitude faster than the TMVA built-in artificial neural networks, has an advanced preprocessing mechanism for the input features and is very resistant against over-training due to automatic relevance determination of the input features and internal regularization algorithms. Since it is a commercial product it is not shipped with BASF2, instead it is loaded at runtime via the plugin mechanism of ROOT if it is available.

Moreover, NeuroBayes offers its own control plots with more detailed information on the individual contribution of the input variables. The output produced by NeuroBayes for the reconstructed invariant mass `M` used in the training performed in the steering file in Listing D.1 is shown in Figure 4.7.

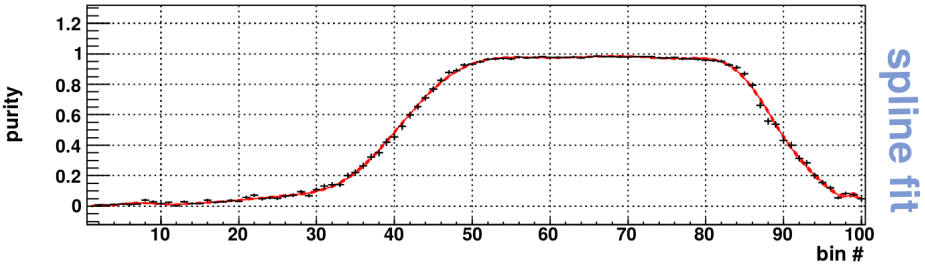
The box in Figure 4.7a contains additional information about the significance of the variable and the correlation to other variables. NeuroBayes automatically determines



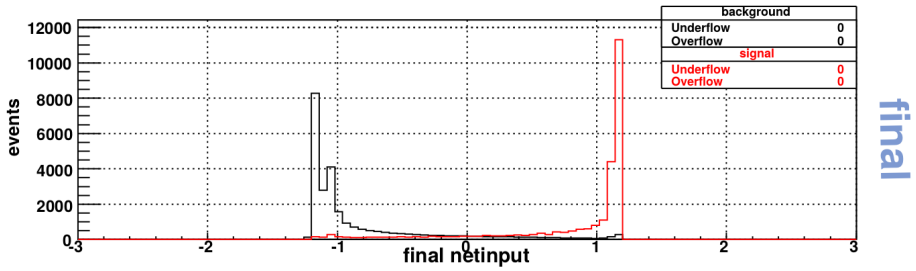
(a) Information about significance and correlation.



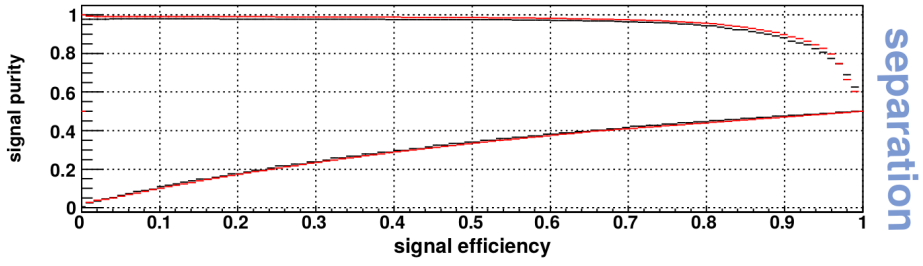
(b) Signal and background distribution of feature after transformation to a uniform binning.



(c) Spline fit on the purity in each bin.



(d) Input into neural network after preprocessing.



(e) Receiver operator characteristic for this feature.

Figure 4.7.: Detailed information on the individual contribution of a input feature provided by NeuroBayes. The plot shows the reconstructed mass variable M of the decay $D^0 \to K^-\pi^+$.

the relevance of each feature and discards features with a significance below a user-defined threshold to avoid over-training. Figure 4.7b shows the `M` distribution of signal and background candidates in a uniform binning, similar to the one used by the FastBDT algorithm. Figure 4.7c shows a spline fit on the purity of each bin. This spline fit regularizes the feature and prevents over-training. The fitted purity on the y-axis is transformed to a distribution with mean 0 and standard deviation 1. Finally, the input features are decorrelated by a principal component analysis. This distribution of the final network input is shown in Figure 4.7d. Figure 4.7e is a ROC plot, similar to the TMVA ROC plot, however showing only the contribution of the `M` feature.

Although the training using NeuroBayes requires more time, it is used in favour of FastBDT if available, because its regularization techniques are better, which yields an overall better performance on an independent test-sample. NeuroBayes was already extensively employed in the FR [12]. In addition, NeuroBayes is constantly developed further in contrast to the TMVA built-in methods and FastBDT. Therefore, users employing NeuroBayes profit from the possibility to retrain the FEI as soon as new versions and features become available.

4.3.6. Speed Comparison

The TMVA built-in gradient boosted decision tree (BDT) and multi-layer perceptron (MLP) were compared with the FastBDT implementation and the NeuroBayes package. The methods were trained using the example steering file in Listing D.1 and one million generic $B\bar{B}$ Monte Carlo events. The hyper-parameters of FastBDT (e.g. the number and depth of the trees) and NeuroBayes (e.g. the maximum number of iterations and preprocessing flags) were optimized to give the best separation. Fortunately, both methods are almost equal in the quality of their separation; in consequence the duration of training is a meaningful quantity for comparison.

Afterwards, the same hyper-parameters were applied to the corresponding TMVA built-in methods if a corresponding parameter was available. The quality of the achieved separation is similar for TMVA-BDT, FastBDT and NeuroBayes as can be seen from the ROC curves in Figure 4.3a. The correlation between the classifier outputs of these methods is above 96 %. TMVA-MLP performed worse and required the longest training and application time.

Both NeuroBayes and FastBDT are faster than their equivalent methods in TMVA, while providing a better separation. If one uses more variables, events and trees the FastBDT implementation is up to 50 times faster than the TMVA-BDT in training and up to 10 times faster in application. NeuroBayes is not as fast as the TMVA-BDT and FastBDT but provides better separation in more complicated trainings. In addition, one can operate NeuroBayes without the neural network,

Table 4.1.: Speed comparison for training and application of FastBDT and Neurobayes versus their respective TMVA-builtin counter-parts TMVA-BDT and TMVA-MLP.

Method	Training time in sec	Application time in sec
NeuroBayes	89.8	1.3
NeuroBayes (without network)	3.9	1.3
FastBDT	3.4	0.3
TMVA-BDT	19.2	1.5
TMVA-MLP	1670	1.9

which is faster than TMVA-BDT while providing a similar separation. The time spent to train and apply the methods is summarized in Table 4.1.

4.4. Parallelisation

Processing billions of events is not possible on a single CPU core. In consequence, it is important to exploit parallelization opportunities on all levels. In modern computing there are several parallelization techniques:

- execution of different CPU instructions at the same time using pipelines;
- execution of the same CPU instruction on data aligned consecutively in the memory using vector instructions;
- multi-processing using threads or processes on a multi-core system;
- and distributed computing using multiple computing nodes.

Although the first two techniques are automatically used if a state-of-the-art compiler and a modern CPU is employed, software developers can optimize their time-critical code by avoiding conditional jumps like shown in Figure 4.6. On the other hand, the deployability of multi-processing and distributed computing depends on the implemented algorithms. Fortunately, the processing of independent events is an embarrassingly parallel problem. Hence, using these techniques is comparatively easy.

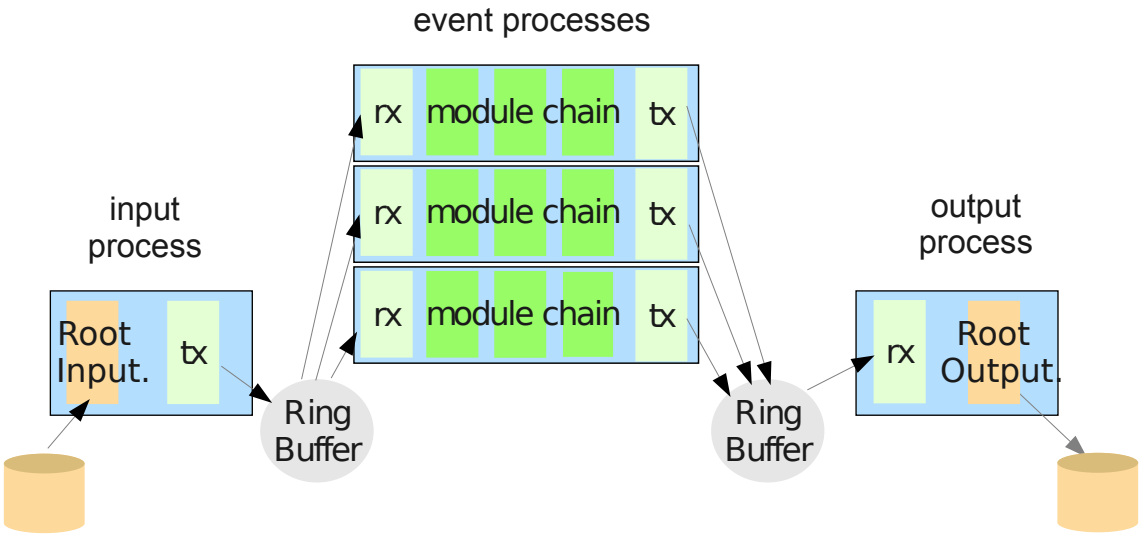


Figure 4.8.: Schematic view of the parallel processing implemented in BASF2.

4.4.1. Multi-Processing

BASF2 supports multi-processing out of the box using processes and shared-memory [39]. However, the `RootInput` and `RootOutput` module cannot run in parallel due to limitations in the ROOT framework. In particular, writing a ROOT file in parallel is not supported. Therefore, the module path provided by the user is split into three parts: an input part running on a single core; a main part running in parallel on multiple cores; and an output part running again on a single core. In between these parts, special modules are inserted which serialize (deserialize) the events into (from) a shared-memory ring-buffer. The final path executed in parallel is visualized in Figure 4.8. The serialize and deserialize modules are named `tx` and `rx` respectively.

Besides the `RootInput` and `RootOutput` module, there are several other modules which perform I/O operations on ROOT files, e.g. `VariableToNtuple` and `TMVATeacher`. The output operations of these modules are synchronized using the `DataStore`. As a result these modules can run in parallel despite their access of ROOT files.

4.4.2. Distributed Computing

BASF2 can also run on multiple computing nodes, yet there are several options available. The simplest option is to run independent BASF2 instances on each node using the same steering file. A dedicated tool named gBasf2 is available for submitting grid-based BASF2 jobs. In the end the ROOT output files of these instances can be merged if needed. This approach works fine for simple steering files for example Monte Carlo production and simple analyses.

However, for non-trivial steering files like the one used by the FEI it is not acceptable to run the steering file multiple times, because the steering file may do event independent calculations like training MVCs. In addition, the steering file may have to be executed multiple times on the same data, like the example in Listing D.1.

A Map-Reduce [40] approach was implemented to take care of these complications, it is visualized in Figure 4.9. At first, the Monte Carlo data is distributed by the Master to the different computing nodes. Afterwards, the Master executes the steering file for the first time and sends the generated BASF2 path to all nodes. Each node applies the path to its part of the Monte Carlo data. The ROOT output file generated by the `RootOutput` module remains on the node and is used as input in the following runs. The remaining ROOT files produced by other modules like `VariableToNtuple` or `TMVATeacher` are sent back to the Master and are merged together. Typically these files are significantly smaller than the ones produced by the `RootOutput` module. If multiple runs over the data are needed the previous steps are repeated, starting with the execution of the steering file. During this second run of the steering file, the files received from the nodes are usually used to train MVCs and to calculate pre-cuts. At the end of execution the steering file generates a new path, which is again sent to the nodes.

The FEI uses this Map-Reduce approach; in consequence its training is performed highly parallel employing up to one-thousand computing nodes at the KEK computing centre simultaneously.

For the sake of completeness it should be mentioned that the DAQ package implements a third possibility for distributed computing. It is used to run different parts of a path on different nodes. In between the parts, modules are added which serialize (deserialize) the data and send (receives) them through (from) a network socket. This is similar to the approach used to implement multi-processing in BASF2. The communication between the event-builder PCs and the high-level trigger farm is based on this approach, as well as the live monitoring of the data acquisition.

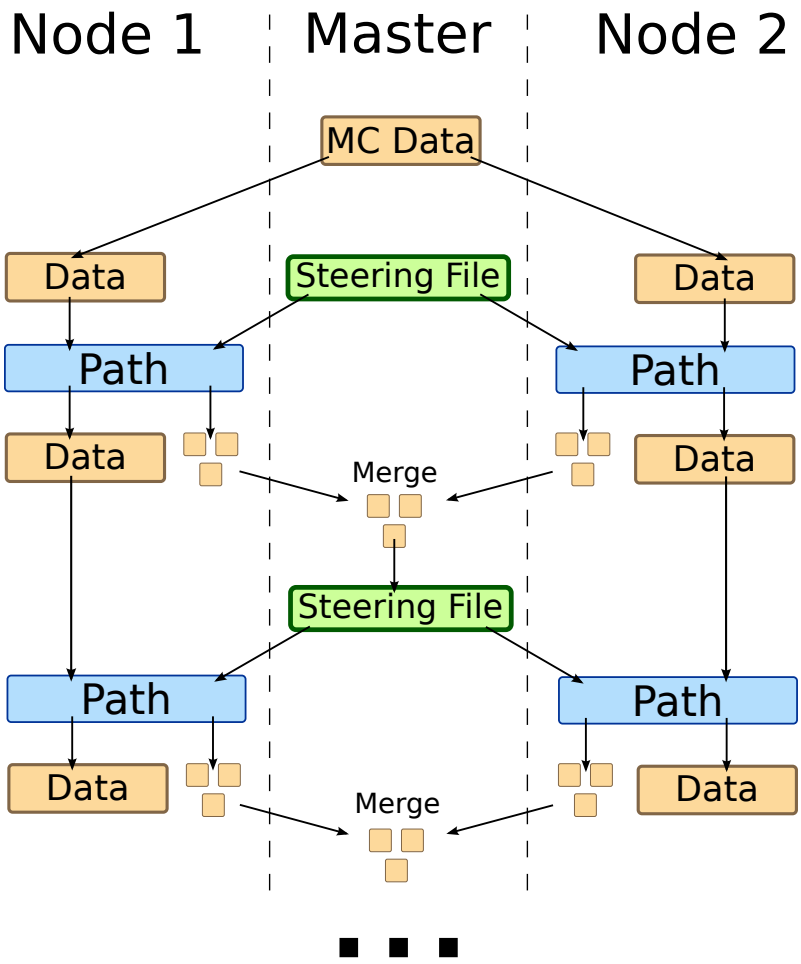


Figure 4.9.: Map-Reduce approach for distributed computing in BASF2.

5. Full Event Interpretation

The Full Event Interpretation (FEI) is part of the Belle II Analysis Software Framework (BASF2) and implemented in the analysis package. It was the first advanced analysis technique implemented in BASF2, hence it served as an extensive test of the whole framework. The implementation of additional features was necessary and existing code had to be developed further. The modifications were driven by the main design goals of the FEI: speed, usability and a high degree of automation.

Furthermore, new features were implemented in a general way. In consequence libraries developed for the FEI like TMVAInterface and VariableManager are already adopted by other groups for implementing techniques like flavour tagging and continuum suppression.

The algorithm itself is implemented purely in Python and takes care of:

- reconstructing a user-defined multi-level decay topology;
- training mutually dependent multivariate classifiers (MVCs) for each decay channel;
- determining sensible channel-specific pre-cuts and particle-specific post-cuts to reduce combinatorics;
- and generating a document[1] summarizing the key performance indicators and control plots of all decay channels and particles used in the FEI.

The reconstructed decay topology, methods and variables used in the multivariate classification as well as desired efficiencies and purities of pre-cuts and post-cuts are defined by the user (see Appendix C).

The multi-level decay topology induces complicated dependencies between the required BASF2 modules. Therefore, the FEI uses a dependency graph meta-framework built on top of BASF2 which runs at steering level. It automatically solves the dependencies and generates the correct BASF2 module path optimized for parallel processing. The technical details of this framework are discussed in Appendix B.

The BASF2 framework is still under development. The efficiencies and distributions stated in this thesis serve only illustrating purposes. In particular, at the time of writing there was no beam-background available, the Monte-Carlo matching code did not work properly in some corner cases and the track as well as the vertex fitting was not finished.

[1] The Full Event Interpretation Report (FEIR)

5.1. Hadronic, Semileptonic and Inclusive Tagging

As previously described, the FEI automatically reconstructs one out of the two B mesons in an $\Upsilon(4S)$ decay to recover information about the remaining B meson. In fact, there is an entire class of analysis methods, so-called tagging-methods, based on this concept. In the past there were three distinct tagging-methods: hadronic, semileptonic and inclusive tagging.

Hadronic tagging solely uses hadronic decay channels for the reconstruction. Hence, the kinematics of the reconstructed candidates are well known and the tagged sample is very pure. Then again, hadronic tagging is only possible for a tiny fraction of the dataset on the order of a few per mille.

Semileptonic tagging uses semileptonic B decays. Due to the high branching fraction of semileptonic decays this approach usually has a higher tagging efficiency. On the other hand, the semileptonic reconstruction suffers from missing kinematic information due to the neutrino in the final state of the decay. Hence, the sample is not as pure as in the hadronic case.

Inclusive tagging combines the four-momenta of all particles in the rest of the event of the signal-side B candidate. The achieved tagging efficiency is usually one order of magnitude above the hadronic and semileptonic tagging. Yet the decay topology is not explicitly reconstructed and cannot be used to discard wrong candidates. In consequence, the methods suffers from a high background and the tagged sample is very impure.

The FEI combines the first two tagging-methods: hadronic and semileptonic tagging, into a single algorithm. Simultaneously it increases the tagging efficiency by reconstructing more decay channels in total. The long-term goal is to unify all three methods in the FEI. The algorithm presented in this thesis is only the first step in this direction.

5.2. Hierarchical Approach

The basic idea of the Full Event Interpretation is to reconstruct the particles and train the MVCs in a hierarchical approach. The approach is depicted in Figure 5.1. At first the final-state particle candidates are selected and corresponding classification methods are trained using the detector information. Building on this, intermediate particle candidates are reconstructed and a multivariate classifier is trained for each employed decay channel. The MVC combines all information about a candidate into a single value – the signal-probability. In consequence, candidates from different decay channels can be treated equally in the following reconstruction steps.

For instance, the FEI currently reconstructs 15 decay channels of the D^0. Afterwards, the generated D^0 candidates are used to reconstruct D^{*0} in 2 decay channels. All information about the specific D^0 decay channel of the candidate is encoded in its signal-probability, which is available to the D^{*0} classifiers. In effect, the hierarchical approach reconstructs $2 \cdot 15 = 30$ exclusive decay channels and provides a signal-probability for each candidate, which makes use of all available information.

Finally, the B candidates are reconstructed and the corresponding classifiers are trained. The final output of the FEI to the user contains four `ParticleLists`: `B+:hadronic`, `B+:semileptonic`, `B0:hadronic` and `B0:semileptonic`.

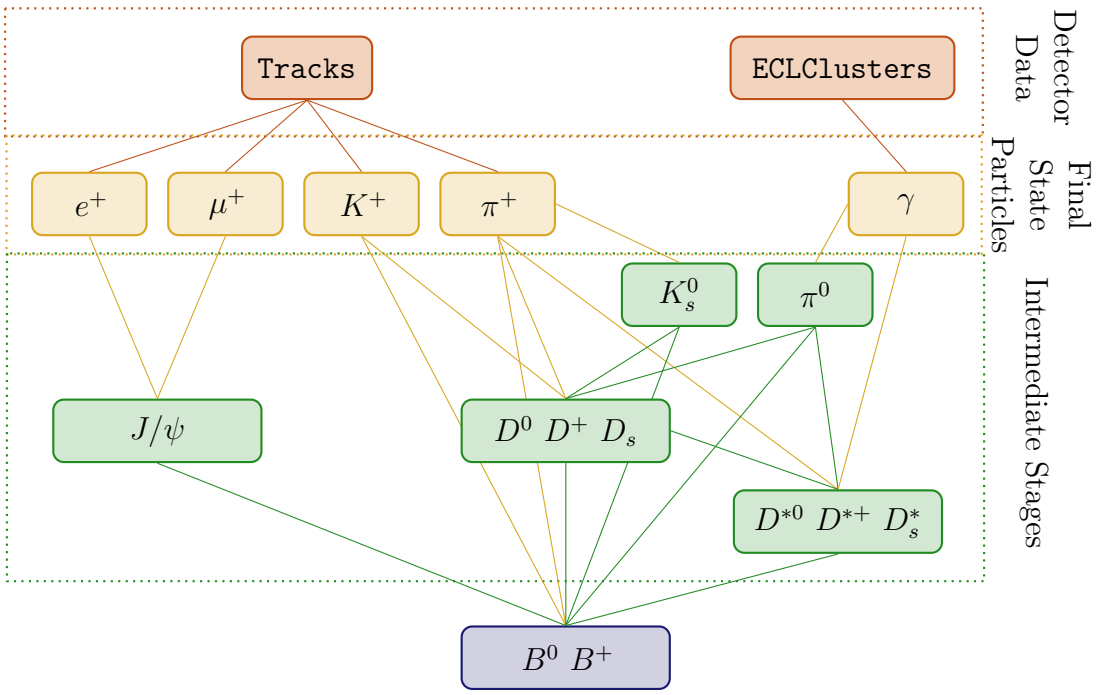


Figure 5.1.: Hierarchical approach of the FEI.

In the remainder of this section the process is explained in detail. All shown control plots and tables are taken from the generic FEI study in Section 6.1. They serve only illustrative purposes.

5.3. Intermediate Cut Determination

A typical event at Belle II contains 12 tracks, hence reconstructing the channel $D^0 \to K^-\pi^+\pi^+\pi^-$ already yields

$$\binom{6}{2}^2 = 225$$

D^0 candidates, assuming 6 positively and 6 negatively charged tracks. A further reconstruction $D^{*+} \to D^0\pi^+$ increases this to

$$225 \cdot \binom{4}{1} = 900$$

D^{*+} candidates. Finally, the reconstruction of $B^0 \to D^{*-}\pi^+\pi^+\pi^-$ yields

$$900 \cdot \binom{4}{2} \cdot \binom{3}{1} = 16200$$

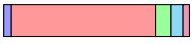
B^0 candidates. On top of that, this is only one out of thousands of exclusive channels and at most two of these candidates are correct as there are only two B mesons in an $\Upsilon(4S)$ event. This problem, rooted in factorial growth, is termed combinatorics. Consequently, it is important to introduce intermediate cuts to reduce combinatorics in order to save computation time.

Table 5.1 is an extract of the CPU time statistics of the generic FEI study in Section 6.1. The effect of the factorial growth is clearly visible in decay channels with high multiplicity like $D^0 \to K^-\pi^+\pi^+\pi^-\pi^0$, which consumed 38 % of the total computation time.

The FEI uses three types of cuts:

- channel-specific pre-cuts are applied directly during the combination of the `Particle` objects and are very fast;
- particle-specific post-cuts are applied after all information about the candidate from the vertex fitting and the MVC is available;
- and user-cuts are additionally applied before all other cuts to introduce a-priori knowledge, which is otherwise not easily employable in the highly automated process.

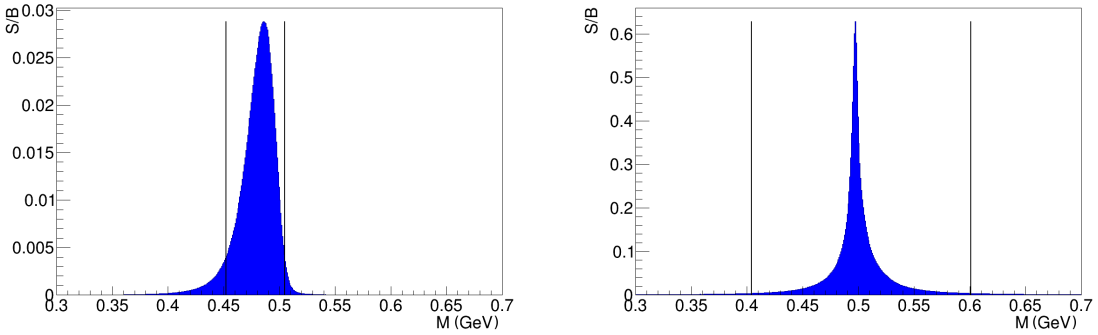
Table 5.1.: Total CPU time spent in event() calls for each channel. Bars show: ParticleSelector, ParticleCombiner, ParticleVertexFitter, MCMatching, TMVAExpert, Other, in this order. Does not include I/O, initalisation, training, post-cuts etc. Taken from the generic FEI study in Section 6.1

Decay	CPU time	By module	Per (true) candidate	Relative time
π^+	17h54m		1ms (3ms)	0.22%
e^+	16h22m		1ms (3ms)	0.20%
μ^+	16h23m		1ms (3ms)	0.20%
K^+	16h37m		1ms (3ms)	0.21%
γ	12h1m		621μs (1ms)	0.15%
$\pi^0 \to \gamma\ \gamma$	227h47m		21ms (43ms)	2.82%
$K^0_S \to \pi^+\ \pi^-$	38h31m		4ms (9ms)	0.48%
$K^0_S \to \pi^0\ \pi^0$	5h31m		816μs (3ms)	0.07%
$D^0 \to K^-\ \pi^+$	13h10m		2ms (11ms)	0.16%
$D^0 \to K^-\ \pi^+\ \pi^0$	293h17m		38ms (138ms)	3.63%
$D^0 \to K^-\ \pi^+\ \pi^0\ \pi^0$	77h49m		12ms (135ms)	0.96%
$D^0 \to K^-\ \pi^+\ \pi^+\ \pi^-$	336h42m		46ms (194ms)	4.17%
$D^0 \to K^-\ \pi^+\ \pi^+\ \pi^-\ \pi^0$	3073h41m		526ms (10s)	38.08%
...				
$B^+ \to \bar{D}^0\ \pi^+$	4h48m		16ms (238ms)	0.06%
$B^+ \to \bar{D}^0\ \pi^+\ \pi^0$	21h20m		3ms (365ms)	0.26%
$B^+ \to \bar{D}^0\ \pi^+\ \pi^+\ \pi^-$	40h7m		8ms (902ms)	0.50%
...				
Total	8070h46m			100.00%

5.3.1. Channel-Specific Pre-Cut

The decay channels of the same particle have different branching ratios and background contributions. A channel-specific pre-cut is applied during the combination of each candidate in order to save computing time and to adjust for differences in the signal-to-background ratio $\frac{S}{B}$ of the decay channels. The cut values are chosen so that the cut boundaries are at the same $\frac{S}{B}$ ratio for each channel. Therefore, channels with more background receive tighter cuts than clean channels.

The pre-cut variable is configurable. Usually the invariant mass `M` is used for intermediate ground-state particles, because its calculation is very fast and signal candidates are only discarded if they are poorly reconstructed. Figure 5.2 shows the chosen pre-cuts for two decay channels of the K_S^0. The $K_S^0 \to \pi^0\pi^0$ has a lower branching fraction and reconstruction efficiency than $K_S^0 \to \pi^+\pi^-$. In addition, it suffers from a higher background contribution due to the large amount of π^0 candidates in each event. In consequence, the signal to background ratio is worse and it receives a tighter cut in comparison with $K_S^0 \to \pi^+\pi^-$.



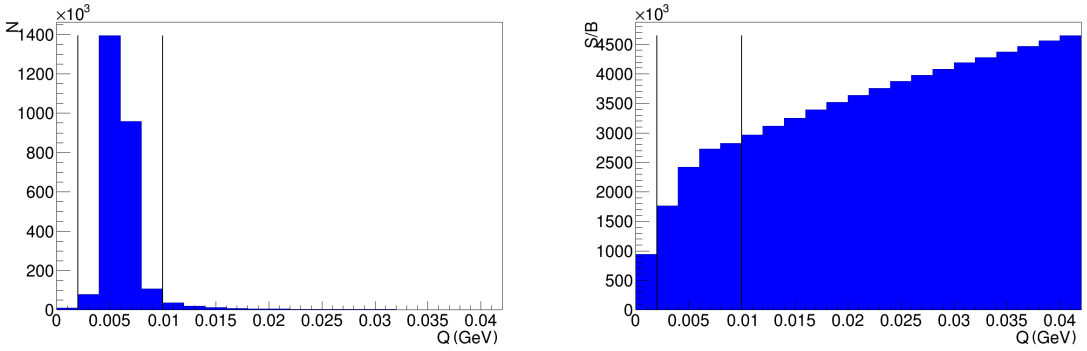
(a) $K_S^0 \to \pi^0\pi^0$ (b) $K_S^0 \to \pi^+\pi^-$

Figure 5.2.: Signal to background ratio over invariant mass and determined channel-specific pre-cuts for K_S^0. Taken from the generic FEI study in Section 6.1

For D^* (excited D) particles the released energy in the decay `Q` is used instead of `M`. Thus, the uncertainty of the D mass reconstruction does not enter the calculation and the signal peaks very sharply, which makes the cut more efficient. Figure 5.3 shows the signal and background distributions used to calculate the pre-cuts for $D^{*+} \to D^0\pi^+$. Noteworthy is the narrow cut, which discards large amounts of background while preserving almost all signal candidates.

The beam-constrained mass[2] M_{bc} of the hadronic B candidates is used in an analysis to extract the signal-fraction on real data, therefore peaking background contributions

[2]The beam-constrained mass is calculated like the invariant mass, with the energy of the particle replaced by the beam-energy.



(a) Signal (b) Background

Figure 5.3.: Pre-cut distributions for $D^{*+} \to D^0\pi^+$. Taken from the generic FEI study in Section 6.1.

due to channel-dependent cuts on M_{bc} or correlated quantities must be avoided. In consequence, the pre-cuts on the hadronic B candidates cannot use kinematic variables like the invariant mass or the momentum. The calculation of vertex variables like the distance to the interaction point on the other hand needs a lot of computing time, hence they are not suitable for fast pre-cuts as well.

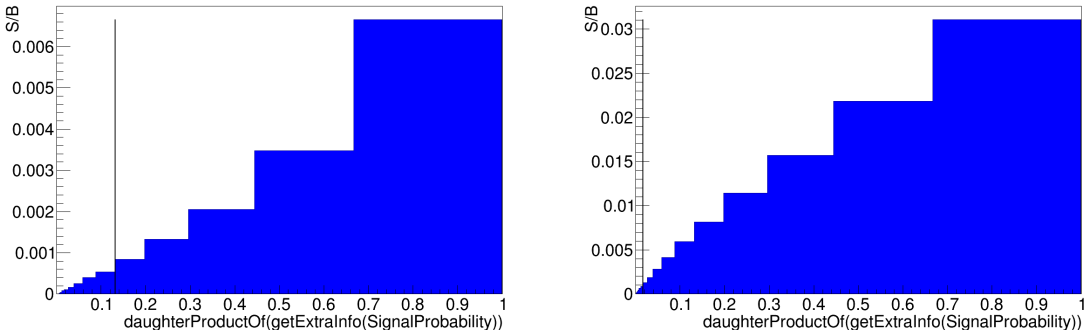
Similar to the hadronic case, the cosine of the angle between the B and the $Dl(\pi)$-system, named $\cos\Theta_{BDl}$, is used to extract the signal-fraction on real data for semileptonic B candidates. Hence, a correlation between this quantity and the applied pre-cuts has to be avoided. Moreover, the distribution of kinematic variables is very broad for semileptonic B candidates due to the missing neutrino.

Therefore, the pre-cuts of the final B candidates employ the product of the daughter signal probabilities $\prod\sigma$, which is fast to calculate and avoids a correlation to M_{bc} and $\cos\Theta_{BDl}$. Figure 5.4 shows the signal to background ratio over the product of the daughter probabilities. Instead of a two-sided cut, only a cut with a lower boundary is applied. It is evident from the Figure that $\prod\sigma$ works well for the hadronic and the semileptonic case.

5.3.2. Particle-Specific Post-Cut

The post-cut employs all available information about a candidate by cutting on the signal-probability calculated by the MVC. Since the interpretation of the signal-probability is the same for all candidates independent of their decay channels the cut is channel-independent.

It is important to choose this cut tight enough, otherwise one loses a lot of signal candidates in the consecutive reconstruction steps due to a bad signal-to-noise ratio.



(a) Semileptonic $B^+ \to D^-\pi^+e^+$ (b) Hadronic $B^0 \to \bar{D}^0\pi^+\pi^-$

Figure 5.4.: Signal to background ratio for final B candidates. Taken from the generic FEI study in Section 6.1.

This is especially true for the final-state particles. The post-cuts for intermediate particles are typically not as tight as the ones for the final-state particles. However, it is essential to choose a tight cut for π^0 and K^0_S candidates to reduce combinatorics in consecutive reconstruction steps, because both particles have a high average number of candidates per event.

The resulting efficiencies and purities after applied pre- and post-cuts are stated in Tables 5.3, 5.4 and 5.5 for the final-state, intermediate and B particles, respectively. They are discussed in more detail below.

5.3.3. User-Cut

The previously described cuts are automatically determined, yet sometimes it is necessary to apply cuts requiring a-priori knowledge about the employed data or physics analysis. The FEI allows the user to introduce user-cuts, which are applied before the automatic cuts are determined and applied.

For instance, usually in a physics analysis one is only interested in well reconstructed tag-side B mesons. The energy of the B mesons is determined by the beam-energy. In consequence the difference between the center of mass beam-energy and the reconstructed center of mass energy of the B meson itself is a measure for the quality of the reconstruction. Hence, a user-cut for hadronic B candidates on this energy difference $|\Delta E| < 0.5\,\text{GeV}$ makes sense. From the ΔE distribution in Figure 5.5 it is evident that this cut is very loose.

Furthermore, the beam-constrained mass M_{bc} is used to extract the signal-fraction on data, therefore the automatic cuts avoid modifying the shape of this quantity. However, usually only the region above $5.2\,\text{GeV}$ is considered for this extraction.

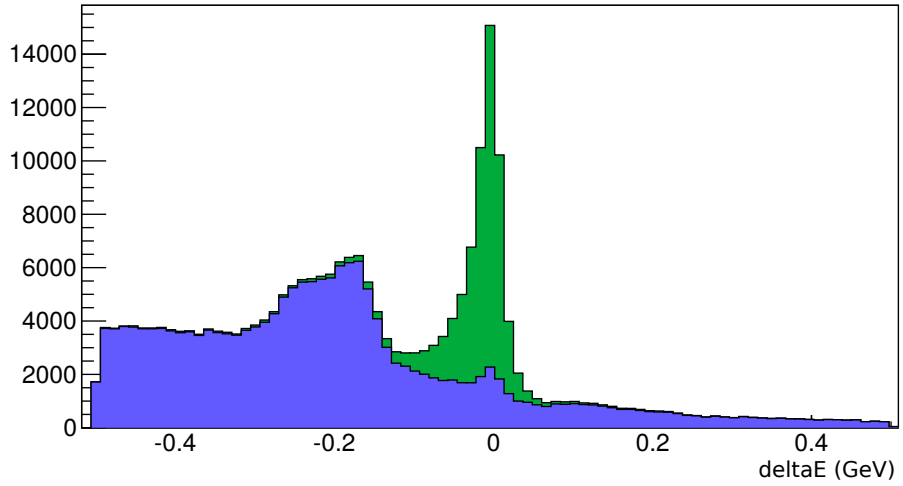


Figure 5.5.: Distribution of ΔE for signal in green and background in blue for B^+. The signal and background yields are normalized to the same order of magnitude.

Hence, a user-cut for hadronic B candidates on the beam-constrained mass $M_{bc} > 5.2\,\text{GeV}$ makes sense as well. See Figure 5.10.

Besides, the user-cut could be used to introduce cuts to suppress continuum background. For instance, D mesons from continuum background exhibit a greater momentum in the center of mass frame than D mesons from an $\Upsilon(4S)$ event. The oldFull Reconstruction (FR) applied a cut at $p_{cms} < 2.6\,\text{GeV}$, to suppress continuum events. Currently the FEI does not employ continuum events in its training and does not use this cut.

Finally, the user-cut is used to enforce the constraint that no additional tracks are left after the signal-side and tag-side reconstruction. From Figure 5.6 it is evident that this cut is a very strong constraint, which removes large amounts of background, while preserving nearly all signal candidates.

The FEI automatically exploits this constraint during the reconstruction of the tag-side if the user provides his analysis-specific signal-side selection. Consequently, the computation time for events with high multiplicity is drastically reduced and at the same time the MVCs are adjusted to the specific signal-side selection of the analysis.

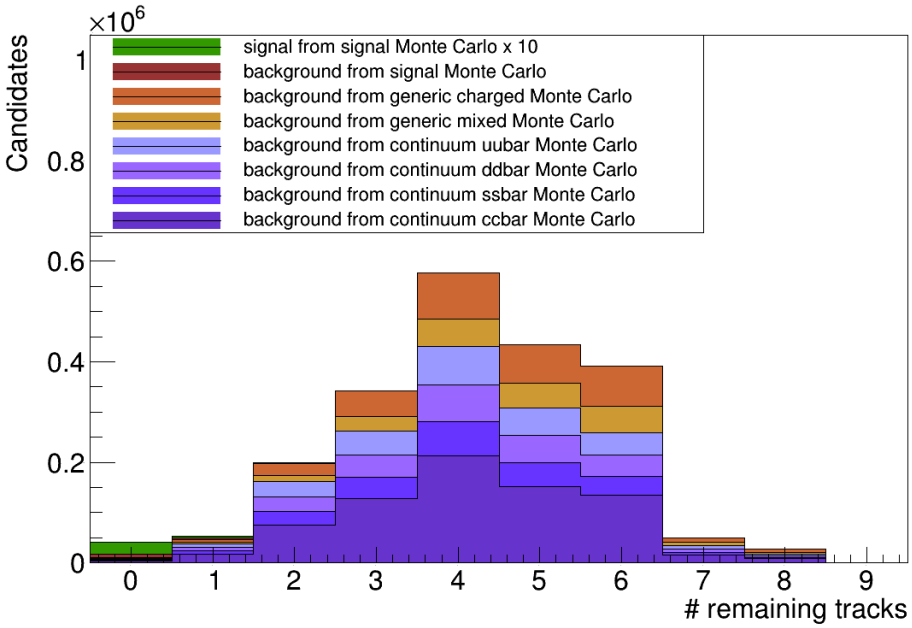


Figure 5.6.: Remaining tracks in the rest of event of the final $\Upsilon(4S)$ candidate. The candidates are reconstructed from one million signal, generic charged, generic mixed, continuum $c\bar{c}$, continuum $s\bar{s}$, continuum $s\bar{s}$, amd continuum $s\bar{s}$ Monte Carlo data. The signal component is scaled up by a factor of ten. Generated using the generic FEI study in Section 6.1.

5.4. Multi-Level Decay Topology Reconstruction

This section describes the hierarchical reconstruction in detail. Currently over 100 distinct channels are automatically reconstructed. This amounts to over 12000 exclusive B decay channels. The resulting covered branching fractions are shown in Table 5.2.

Table 5.2.: Covered branching fractions by the generic FEI study in Section 6.1. The inclusive is the sum of the branching fractions of all used decay channels. The exclusive branching fractions take the covered branching fractions of the daughter particles into account.

Particle	# decay channels	Branching Fraction in %	
		Inclusive	Exclusive
π^0	1	98.8	98.8
K_S^0	2	99.7	99.0
D^0	15	31.4	31.0
D^{*0}	2	100	30.7
D^+	11	22.2	22.0
D^{*+}	3	100	28.0
D_s^+	10	6.6	6.5
D_s^{*+}	2	100	6.5
J/psi	2	11.9	11.9
B^+ (semileptonic)	21	17.4	5.2
B^+ (hadronic)	8	9.0	1.7
B^0 (semileptonic)	19	15.3	4.0
B^0 (hadronic)	8	9.8	1.1

Final-State Particles

At first the `Particle` data-objects representing the final-state particle hypotheses are generated for each `Track`, `ECLCluster` and `KLMCluster`.

Subsequently, the `Particle`s are gathered up into `ParticleLists`. Usually only $e^+, \mu^+, \pi^+, K^+, \gamma$ are considered, because B decay channels containing baryons like p (protons) and n (neutrons) in their final-state have only a small branching-fraction. Moreover, the energy and momentum of K_L^0 particles are measured poorly, therefore they are not suitable for hadronic reconstruction.

A post-cut is applied on the signal-probability of each final-state particle candidate. The efficiency and purity of the final-state particles before and after the applied cut are shown in Table 5.3. The loss in signal-efficiency is small compared to the gain in purity. The efficiency is normalized by the amount of primary particles at generator level. Therefore, muons produced by pions decaying in-flight are not considered in the denominator of the muon efficiency, in consequence the efficiency for muons is above 100 %.

Furthermore, these efficiencies are likely to increase in the future, due to expected improvements in the tracking software. In particular, the VXD standalone tracking, which recovers low-momentum tracks, is not finished at the time of writing.

Table 5.3.: Final-state particle efficiency and purity before and after the applied post-cut. Calculated during the training of the generic FEI study in Section 6.1.

Final-state particle	Efficiency in %		Purity in %	
	detector	post-cut	detector	post-cut
π^+	82.62	82.25	68.964	84.866
e^+	88.88	80.23	5.972	64.274
μ^+	182.79	147.37	8.643	35.794
K^+	80.36	78.44	12.988	67.082
γ	50.30	50.37	54.849	54.934

Intermediate Particles

The standard FEI configuration reconstructs the following intermediate particles: π^0, K^0_S, J/ψ, D^0, D^{0*}, D^+, D^{+*}, D^+_s and D^{+*}_s.

Intermediate resonances like K^*, ρ and ω are not explicitly reconstructed, because it is not easy to account for possible interferences between resonant and non-resonant decays with the same final state. Instead it is assumed that the Monte Carlo data uses the correct Dalitz models to describe these decays, hence information about the Dalitz structure of decays is automatically exploited to separate signal and background candidates by the MVCs if the necessary kinematic information is provided.

Each decay channel of an intermediate particle is reconstructed separately. All generated candidates for the channel are stored in a channel-specific `ParticleList`. During the combination of the daughter lists a fast channel-specific pre-cut is applied. Afterwards, the Monte Carlo truth of the remaining candidates is matched.

Finally, all channel-specific `ParticleLists` of a particle are combined into a single `ParticleList` and a particle-specific post-cut is applied.

The efficiencies and purities of all intermediate particles are shown in Table 5.4. Noteworthy are the tight post-cuts for π^0 and K_S^0, which increase the purity by a factor 6 and 15, respectively. Again the efficiencies are likely to increase in the future, due to better final-state particle efficiencies and improved vertexing. For instance, the vertexing for K_S^0 will be improved by a dedicated vertex finding algorithm, which takes advantage of the peculiar vertex structure of the K_S^0.

Table 5.4.: Per-particle efficiency and purity before and after the applied pre- and post-cut. Calculated during the training of the generic FEI study in Section 6.1.

Particle	Efficiency in %			Purity in %		
	recon.	pre-cut	post-cut	recon.	pre-cut	post-cut
π^0	89.06	68.59	49.37	2.508	5.534	30.912
K_S^0	47.44	45.05	34.85	0.745	3.969	62.213
D^0	21.75	19.45	13.69	0.018	0.098	5.825
D^+	14.66	11.62	8.15	0.010	0.100	5.600
D^{*+}	5.53	5.10	5.08	0.087	15.116	16.808
D^{*0}	8.45	8.40	4.99	0.059	0.121	3.808
D_s^+	9.77	6.93	4.42	0.009	0.099	4.149
D_s^{*+}	4.07	4.03	2.31	0.086	0.178	3.739
J/ψ	9.82	8.69	8.69	0.256	22.177	22.177

Final B candidates

The final charged B^+ and neutral B^0 mesons are reconstructed separately in hadronic and semileptonic channels. There is no fundamental difference to the reconstruction of an intermediate particle. However, an additional user-cut is applied in the hadronic case to restrict the parameter space to the region of interest $5.2\,\text{GeV} < M_{bc}$ and $|\Delta E| < 0.5\,\text{GeV}$.

The final efficiencies are stated in Table 5.5. In the hadronic case roughly half of the maximally reachable efficiency given by the covered exclusive branching fraction in Table 5.2 is achieved. In the semileptonic case roughly a third. Hence, the loss of signal events due to the applied cuts is acceptable in comparison with the savings of computation time.

Table 5.5.: Final tagging efficiency and purity before and after the applied user- and pre-cut. Calculated during the training of the generic FEI study in Section 6.1.

Particle	Efficiency in %			Purity in %		
	recon.	user-cut	pre-cut	recon.	user-cut	pre-cut
B^+ (hadronic)	0.87	0.80	0.76	0.002	0.120	0.411
B^+ (semileptonic)	1.34	1.34	1.25	0.116	0.116	0.488
B^0 (hadronic)	0.60	0.55	0.52	0.004	0.354	0.911
B^0 (semileptonic)	1.60	1.60	1.50	0.064	0.064	0.264

5.5. Multivariate Analysis Method Training

For each final-state particle and each decay channel an MVC is trained and applied. In general a candidate is considered signal if it is reconstructed correctly ignoring intermediate resonances and final-state radiation. Semileptonic decay channels additionally allow for missing neutrinos. The secondary vertices of all candidates are automatically fitted if the training contains `Variable`s which require a previous vertex fit. The classifier outputs of all daughter particles are included as `Variable`s in the training of the mother particle. Therefore, the training of the MVCs follows the same hierarchical approach as the reconstruction.

In this thesis the speed-optimized stochastic gradient boosted decision tree implementation was used as MVC method. The NeuroBayes package was unfortunately not applicable due to main-memory restrictions at the KEK computing centre. This restriction will be circumvented in the near future by using a custom-built NeuroBayes version with less memory consumption.

5.5.1. Final-State Particles

The MVCs for the charged final-state particles receive: the PID information from the TOP, ARICH, CDC and ECL sub-detectors; kinematic variables in the center of mass system and the laboratory frame; and the p-value calculated by the track-fit. The photon MVC contains information about the shape and magnitude of the ECL cluster and kinematic variables.

Currently each charged final-state particle is trained against all other charged FSPs as background. Yet one could try to improve the separation further by performing a multi-class classification using error-correction output-codes or by separating the particles pairwise from one another. However, the achieved separation for the charged

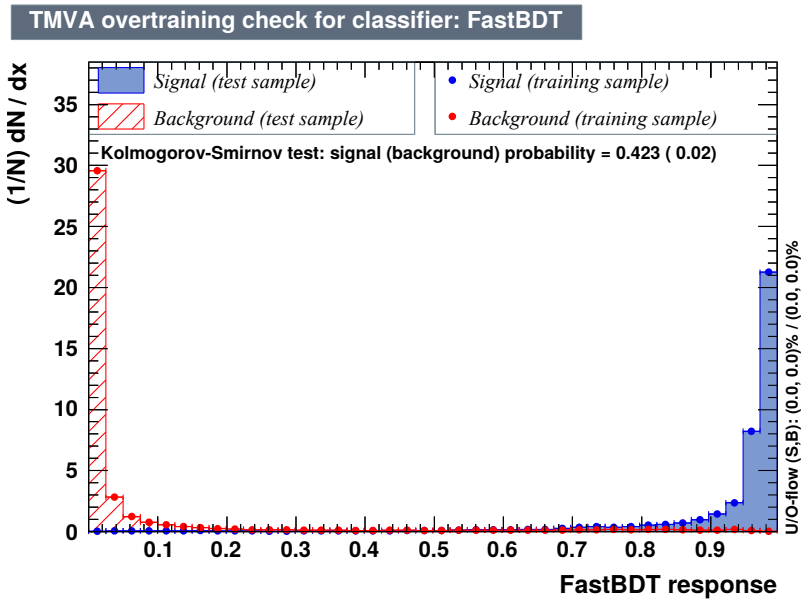


Figure 5.7.: Signal and background distribution of the K MVC. Taken from the generic FEI study in Section 6.1.

final-state particles is already excellent, as can be seen in the example of the kaon MVC in Figure 5.7.

In contrast, the γ MVC has more potential for improvement, in particular since the current training was done without beam-background. Hence, more detailed information about the shape and isolation of the corresponding ECL cluster should be provided to the MVC in the future. Figure 5.8 shows the test-statistic produced by the γ MVC.

5.5.2. Intermediate Particles

For each decay channel of an intermediate particle a separate MVC is trained. Each MVC incorporates kinematic variables, vertex variables and the signal-probabilities of the daughter particles. Additionally, one can include the invariant mass and decay angle of pairs of daughter particles in order to implicitly take resonances like K^* , ρ and ω into account. However, this was not utilized in this thesis, because it was not clear if MC data already described the distribution of these variables correctly. In the future MC data will use the correct Dalitz models to simulate intermediate resonances and consequently the MVCs will be able to take advantage of the peculiar kinematics of decays including intermediate resonances.

In general the intermediate MVCs separate well if enough signal candidates are available. If less than one thousand signal candidates are available for training, the

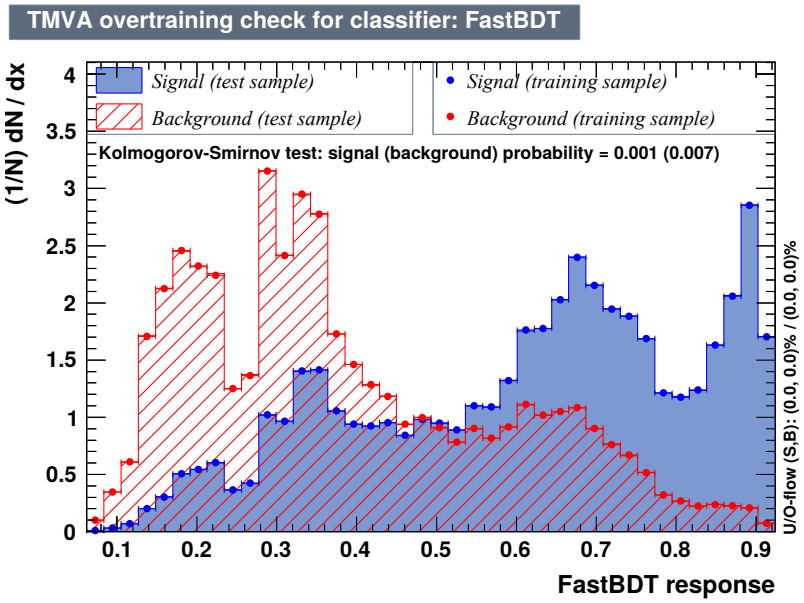


Figure 5.8.: Signal and background distribution of the γ MVC. Taken from the generic FEI study in Section 6.1.

channel is automatically discarded. Usually the most important variables are the distance of the secondary vertex to the IP, the p-value of the vertex fit, and the daughter probabilities. Yet for some channels like $D^0 \to K^-\pi^+\pi^0\pi^0$, which include multiple π^0 the vertex fit cannot be performed, due to restrictions in the current implementation of the vertex fitter. Therefore the corresponding MVCs can be further improved. Moreover, the vertex fitter is not well optimized at the moment and consumes most of the computation time. Both problems are evident from Table 5.1.

Because the different decay channels of daughter particles are not distinguished in the training, it is important that the signal-probability of all daughters is really a probability. Hence, the classifier output of each channel is transformed to ensure this property as described in Section 4.3.3.

5.5.3. Final B candidates

As mentioned earlier, the beam-constrained mass is used to calculate the fraction of correctly reconstructed B candidates from hadronic decay channels on data. On that score, it is important to avoid correlations between the signal probability and the beam-constrained mass M_{bc}. Hence, variables which are strongly correlated to M_{bc} are not included in the training of the MVC.

Typical examples for correlated variables are: the momentum of the particle, decay angles of daughter particles, and invariant mass of combinations of daughter particles.

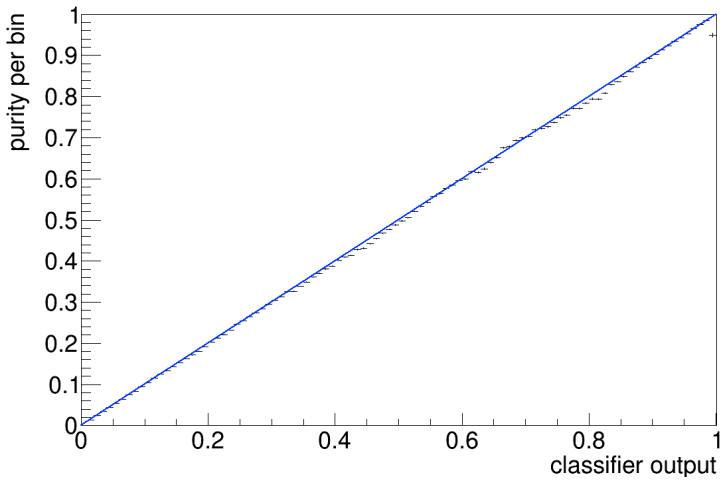


Figure 5.9.: Purity over classifier output after the transformation of the individual classifier outputs of each channel to a probability. Taken from the generic FEI study in Section 6.1.

Therefore, the MVCs include only vertex variables, ΔE and some decay angles which are not correlated to M_{bc}. Similarly, the MVCs for the semileptonic channels have to avoid variables correlated to $\cos\Theta_{BDl}$.

In this thesis only a subset of the variables used in the FR are utilized. More detailed studies are needed to discover variables which are uncorrelated to M_{bc} and $\cos\Theta_{BDl}$, and are well described in MC data.

The final distribution of M_{bc} and $\cos\Theta_{BDl}$ is shown for different cuts on the signal-probability in Figure 5.10 and 5.11.

In the M_{bc} plot the signal peaks at the nominal B mass. The background covers the whole range up to the kinematic limit at 5.29 GeV. The M_{bc} plots exhibit a clear peaking background component. This component was extensively studied. It is not caused by a correlation between M_{bc} and the employed variables in the MVC training. A cross-check training using the same variables to predict M_{bc} showed no significant correlations. Neither is it caused by intermediate cuts, because $\prod\sigma$, which is used to perform the intermediate cuts at B level, has no correlation to M_{bc}.

Detailed investigations showed that the peak consists of B candidates which miss a π^0 or have a misidentified track. This component was not visible in previous studies using the FR, because the FR was always trained on generic and continuum events. The continuum background evens out and covers up the peak. In contrast the FEI does not utilize continuum events in its training.

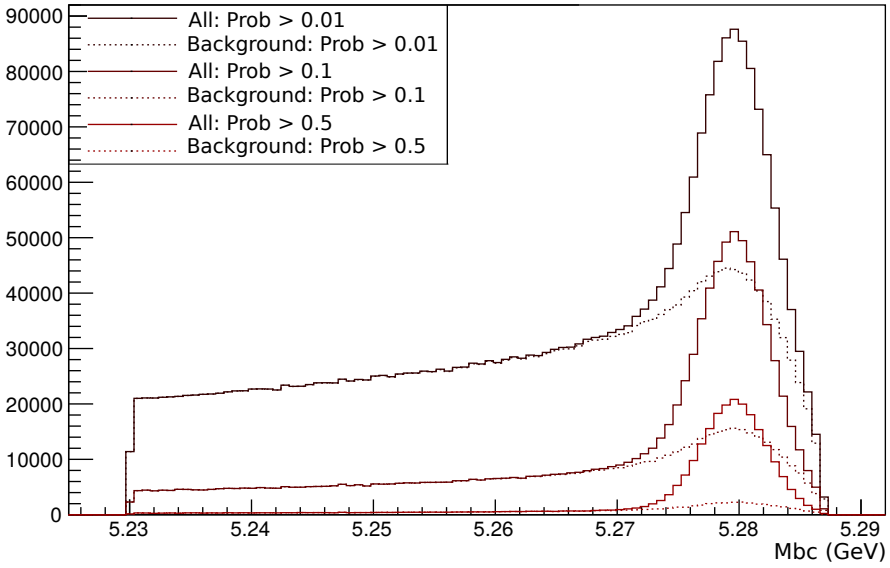
Fortunately, the peaking component can successfully be suppressed by applying a tighter cut on the signal-probability of the candidate, which is evident from the plots

in Figure 5.10. Furthermore, the peaking component becomes less visible if the FEI is applied to data including continuum, in particular real-data.

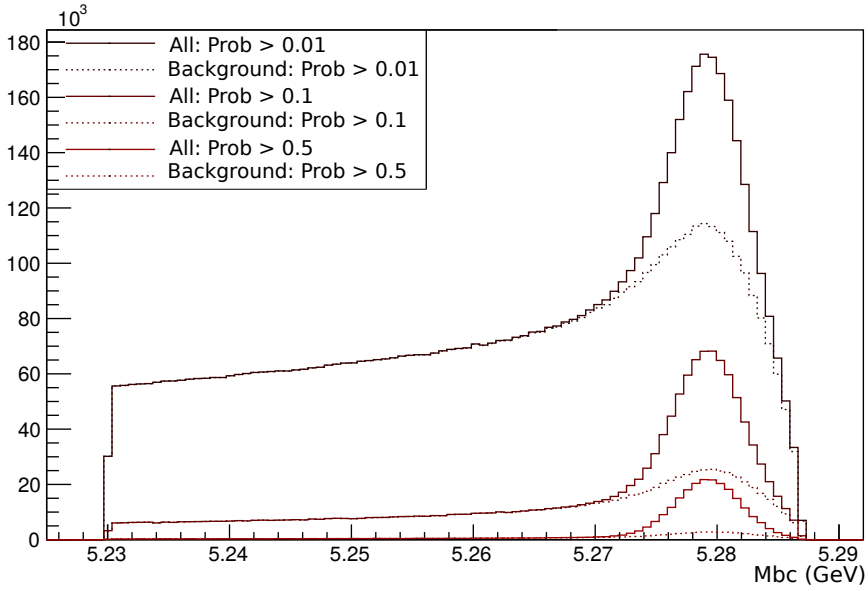
However, this raises the question if the beam-constrained mass is the best way to extract the signal-fraction on real-data. Other methods using well known control-channels similar to the signal-channel may be superior, especially because the MVC methods could use a lot more information if the limitation to variables independent of M_{bc} is dropped. Further studies are necessary to clarify this point.

The signal component in the $\cos\Theta_{BDl}$ peaks between -1 and 1 under the assumption that only one massless particle was not included in the reconstruction of the B candidate. In contrast, background candidates are not subjected to this constraint. As can be seen from Figure 5.11 the semileptonic candidates are not as pure as the hadronic candidates, even if a very tight signal-probability cut is applied.

In summary, the MVCs at B level will be further improved in the near-future by adopting all variables already employed in the FR and maybe by discarding the limitation to variables independent of M_{bc}.

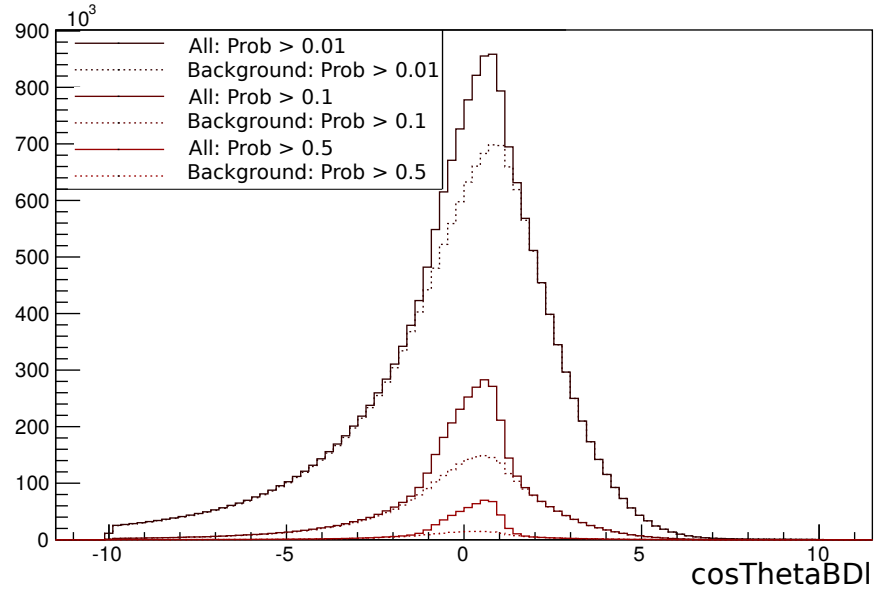


(a) B^0

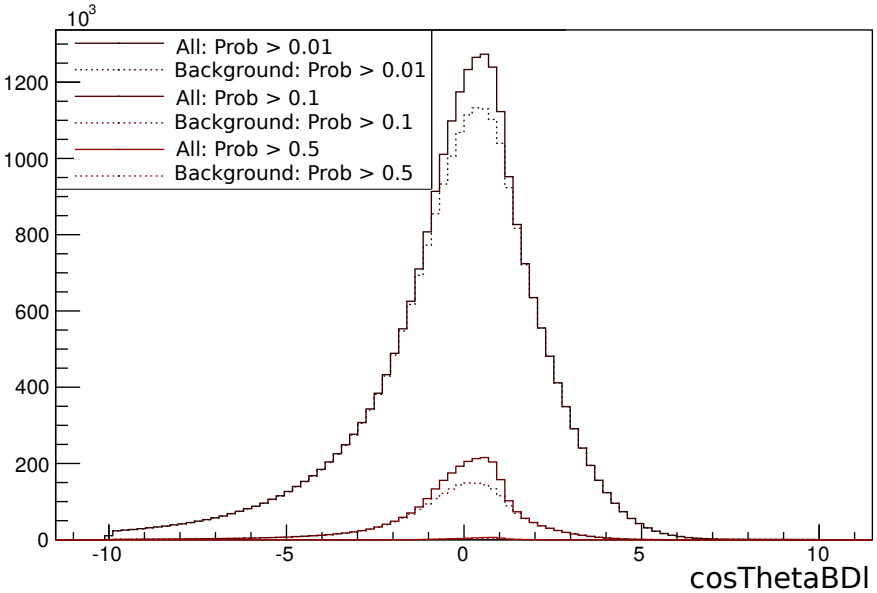


(b) B^+

Figure 5.10.: Beam-constrained mass M_{bc} of all hadronic B candidates after the pre-cut. Dashed lines correspond to background only. Solid lines to all candidates. Taken from the generic FEI study in Section 6.1.



(a) B^0



(b) B^+

Figure 5.11.: `cosThetaBDl` of all semileptonic B candidates after the pre-cut. Dashed lines correspond to background only. Solid lines to all candidates. Taken from the generic FEI study in Section 6.1.

5.6. Automatic Reporting

During the training the algorithm generates automatically a Full Event Interpretation Report (FEIR). A FEIR summarizes the key performance indicators and control plots of every particle and decay channel used in the reconstruction. In particular it includes:

- configuration of all particles;
- efficiency and purity of each particle before any cuts are applied;
- efficiency and purity of each particle after the user-defined cut;
- efficiency and purity of each particle and decay channel after channel-specific pre-cuts;
- efficiency and purity of each particle after particle-specific post-cuts;
- signal, background and $\frac{S}{B}$ distribution of the pre-cut variable for each decay channel;
- calculated channel-specific pre-cuts;
- ROC and overtraining control plots for each MVC training;
- used `Variable`s with ranking and description for each MVC training;
- diagonal control plots for signal probability of each particle and decay channel;
- calculated particle-specific post-cuts;
- beam-constrained mass plots for B mesons reconstructed from hadronic decay channels;
- `cosThetaBDl` plots for B mesons reconstructed from semileptonic decay channels;
- and CPU statistics partitioned by decay channels and BASF2 modules.

In addition, the automatic reporting has a built-in email notification system, which informs the user via email about the completion of the training and the achieved performance.

6. Training and Application of the Full Event Interpretation

The FEI has to be trained on Monte Carlo data and is applied subsequently on real data after an analysis-specific signal-side selection. There are three different types of events one has to consider in the training and application of the FEI:

- double-generic events – $e^+e^- \to \Upsilon(4S) \to B\bar{B}$ with equal branching fractions for charged and neutral B pairs, where both B mesons decay generically[1];
- continuum events – $e^+e^- \to c\bar{c}, s\bar{s}, d\bar{d}, u\bar{u}$;
- and signal events – $e^+e^- \to \Upsilon(4S) \to B\bar{B}$ where one B decays generically and the other decays in an analysis-specific signal-channel like $B^+ \to \tau^+\nu_\tau$.

The final classifier output for the B_{tag} mesons has to identify correctly reconstructed B_{tag} mesons in the signal events of the analysis and reject background B_{tag} mesons from double-generic, continuum and signal events efficiently.

To accomplish a high efficiency for correctly reconstructed B_{tag} in signal events a training on pure signal Monte Carlo after the signal-side selection would be appropriate, but in this scenario background components from double-generic and continuum events would not be considered in the training and therefore could not be rejected efficiently. On the other hand, a training on double-generic and continuum Monte Carlo after signal-side selection suffers from low statistics especially for correctly reconstructed B_{tag} mesons, because the constraint that the reconstructed candidate has to use all remaining tracks is very strict. Moreover, it is not clear if D mesons from continuum background should be considered as signal in the corresponding trainings.

In this thesis the background components are factorized into background from $\Upsilon(4S)$ decays and from continuum events. It is assumed that the continuum events can be suppressed efficiently with the `ContinuumSuppression` module, therefore no Monte Carlo data for continuum events is used in the training of the FEI. Further studies have to be performed to test this assumption.

The FR of Belle was trained on double-generic and continuum Monte Carlo without considering the signal-side selection. In consequence, the background distributions

[1]The B mesons decay in all possible final states with the appropriate branching fractions.

were fundamentally different in training and application. For example, most of the CPU time in the training was used for events with more than 12 tracks, yet these events never led to a valid B_{tag} meson in an analysis with only one track on the signal-side like $B \to \tau\bar{\nu}_\tau$.

Two example trainings of the FEI were performed and are discussed in the following sections. The first training was done on double-generic Monte Carlo without signal-side selection, which corresponds to the FR of Belle. The second training was optimized for the signal-side $B \to \tau\bar{\nu}_\tau$ and trained on double-generic and signal Monte Carlo, in order to get enough signal statistics despite the no-remaining-tracks constraint.

Both studies were performed without beam-background, because unfortunately there was no usable beam-background available. Furthermore, there are known problems in the MC-matching code of BASF2 which can lead to multiple correct tag-side candidates. The effect on the stated efficiencies is of the order of 5 - 10 %. In consequence, the studies below are preliminary. Both FEI trainings were applied to an independent test-sample of one million:

- double-generic charged B^+B^- MC;
- double-generic mixed $B^0\bar{B}^0$ MC;
- continuum $c\bar{c}$ MC;
- continuum $s\bar{s}$ MC;
- continuum $d\bar{d}$ MC;
- continuum $u\bar{u}$ MC;
- and $B \to \tau(\to \mu\nu_\tau\bar{\nu}_\mu)\bar{\nu}_\tau$ signal MC data events.

6.1. Generic Full Event Interpretation

For the generic study, the FEI was trained on 100 million double-generic $B\bar{B}$ events. The training took three days at the KEK computing centre. In total BASF2 consumed 8070 hours of computing time. Additionally, the training of the MVCs consumed 6 hours, however this is the raw time consumed by the FastBDT algorithm. Loading and preprocessing the training-data via TMVA required additional computing time in the same order of magnitude.

Due to the hierarchical approach it is necessary to run multiple times over the training-data. The presented training required 14 iterations, however all interim results like `ParticleLists`, vertex fit results and calculated signal-probabilities were

cached between the runs. As a result the training required 13 terabyte of temporary storage. Unfortunately, this amount of free space was only available on an hierarchical storage management (HSM) data-storage. As a consequence, most of the time was wasted while waiting for the magnetic tape drives.

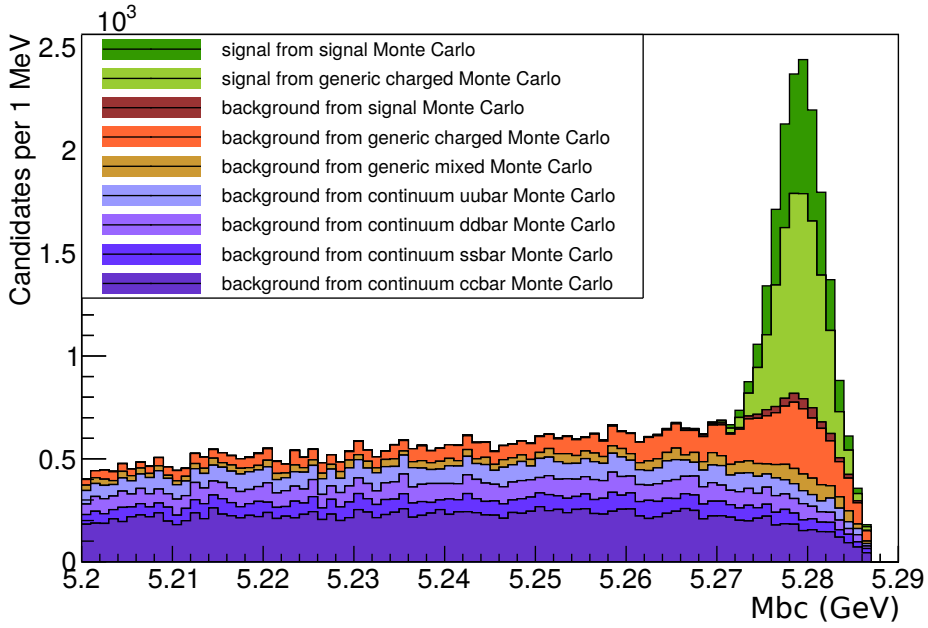
No analysis-specific signal-side selection was used. Hence, the performance can be compared to the FR. The final tagging efficiencies on the test-sample are shown in Table 6.1. The calculation of the efficiencies and purities was restricted to the signal-region: $M_{bc} > 5.27$ for hadronic and $|\cos\Theta_{BDl}| < 1$ for semileptonic B^+ candidates. All 8 semileptonic B^+ channels and 19 out of 21 hadronic B^+ channels were employed by the FEI. The remaining two hadronic channels were discarded because of low statistics.

Table 6.1.: Tagging efficiency on the independent test-sample in percent calculated in the signal-region.

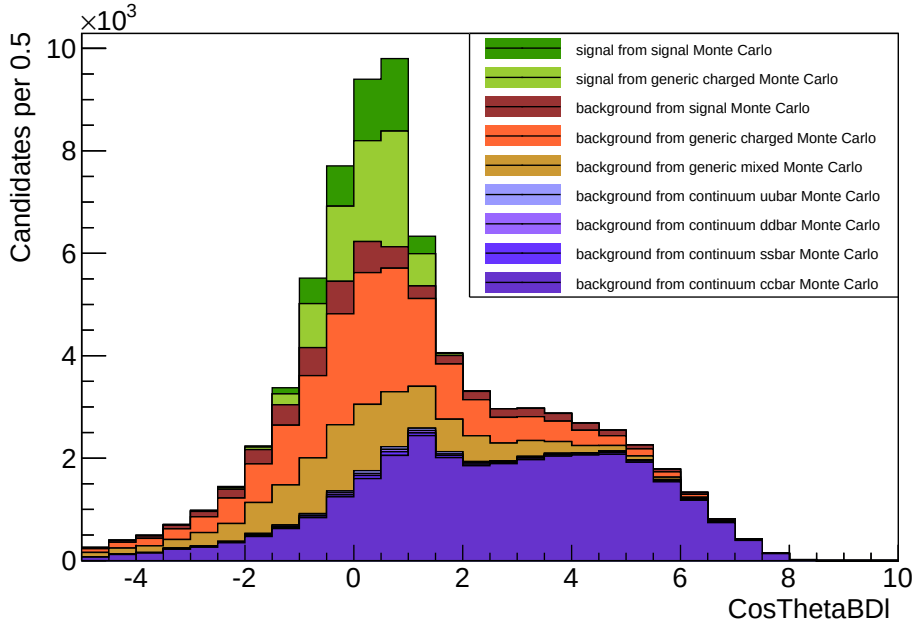
	Signal MC	Charged Generic MC	Overall
Hadronic	0.81	0.71	0.74
Semileptonic	1.13	0.99	1.03

The tagging efficiency on signal MC is higher than on generic charged MC. This demonstrates the influence of the signal-side on the tag-side efficiency. The analysis-specific FEI training can take this influence into account. The overall efficiency is two to three times higher compared to the FR. However, this result is preliminary, further studies have to be conducted, after the known problems in the BASF2 frameworks have been fixed, to confirm these results.

The resulting M_{bc} and $\cos\Theta_{BDl}$ plots on the test-sample are shown in Figure 6.1. The peaking background component is not as prominent as in the training data. Furthermore, it is evident that the continuum background components can be efficiently suppressed by the MVC despite their absence in the training data.



(a) Hadronic candidates: M_{bc} distribution with a purity of 50 % and a tagging efficiency of 0.35 %.



(b) Semileptonic candidates: $\cos\Theta_{BDl}$ distribution with a purity of 32 % and a tagging efficiency of 0.35 %.

Figure 6.1.: Distribution of final B^+ candidates calculated on the independent test-sample.

6.2. Specific Full Event Interpretation

The specific FEI study investigates the gain in performance due to consideration of an analysis-specific signal-side selection. The rare tauonic decay

$$B \to \tau\nu_\tau$$
$$\hookrightarrow \mu\nu_\tau\nu_\mu$$

was chosen as signal-side and the corresponding signal-side selection path was specified in the FEI configuration:

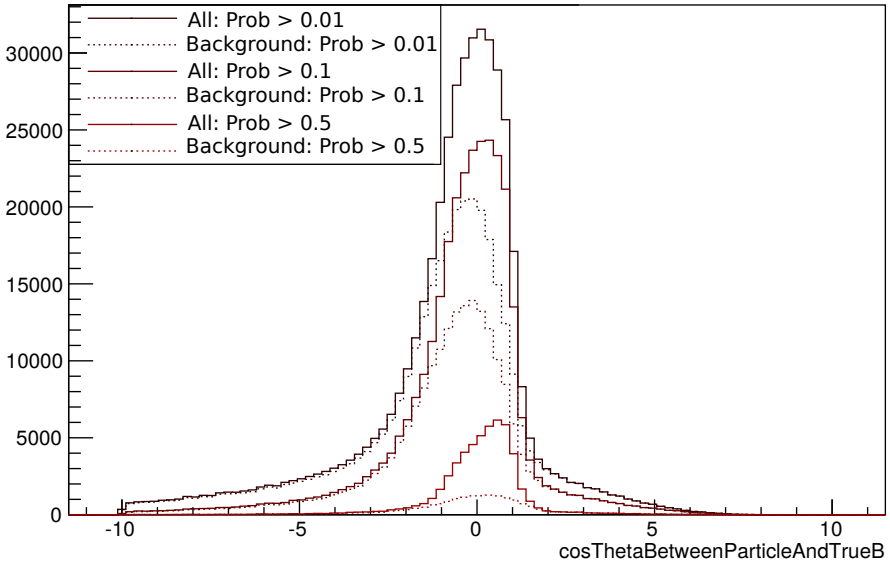
```
1 selectParticle('mu+', 'muid > 0.6 and nTracks <= 12')
2 reconstructDecay('tau+ ->  mu+')
3 reconstructDecay('B+:sig -> tau+')
4 matchMCTruth('B+:sig')
5 buildRestOfEvent('B+:sig')
```

The FEI was trained on 40 million double-generic $B\bar{B}$ and 20 million signal events. Thereby only the rest of event of the B_{sig} candidates were considered in the reconstruction. In particular, events with more than 12 tracks were immediately discarded, because no exclusive B decay channel with a multiplicity of 11 or more tracks was considered in the training. Hence, it is not possible to reconstruct a valid[2] $\Upsilon(4S)$ candidate in these events.

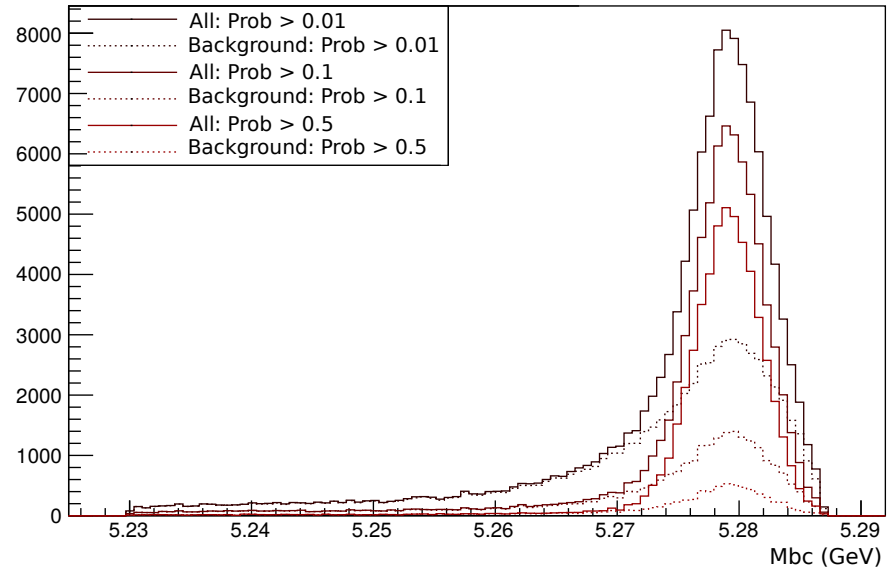
In addition, a user-cut for the B_{tag} candidates was applied, which required that the candidate results in a valid $\Upsilon(4S)$ together with the signal-side selection. This cut discarded nearly all candidates from double-generic $B\bar{B}$ events, therefore it is crucial to include enough signal Monte Carlo events to obtain a reasonable statistic for the training of the B classifiers.

The training took two days at the KEK computing centre. In total BASF2 consumed only 585 hours of computing time and the training of the MVCs only 3 hours. Like in the generic FEI study, the HSM data-storage was the limiting resource in terms of training duration, yet the specific FEI training required only 1.5 terabyte of free space for caching. To summarize, the computing time and the required storage space is reduced by one order of magnitude due exploitation of the signal-side selection.

Figure 6.2 shows M_{bc} and $\cos\Theta_{BDl}$ distributions for the hadronic and semileptonic B^+ candidates respectively. Neutral B^0 were automatically discarded during the training of the FEI due to low statistics after the user-cut. This behaviour was expected and means that background from neutral B^0 can be efficiently suppressed by the user-cut. Since only 20 million signal events were used for the training, only 9 out of 21 hadronic B^+ channels were successfully trained and employed by the FEI.



(a) Distribution of $\cos\Theta_{BDl}$ of all semileptonic B^+ candidates.



(b) Beam-constrained mass M_{bc} distribution of all hadronic B^+ candidates.

Figure 6.2.: Distribution of final B^+ candidates after the pre-cut calculated on the training-data. Dashed lines correspond to background only, whereas solid lines correspond to all candidates.

The remaining channels were discarded because of low statistics. In contrast, all 8 semileptonic B^+ channels were successfully trained.

The user-cut reduces the background by two to three orders of magnitude. A considerable amount of signal from double-generic charged events is also discarded. Yet these candidates would be rejected anyway later in the analysis, because one is only interested in tag-candidates which are compatible with the signal signature of the analysis. In consequence combinatorics are drastically reduced and the B classifiers exhibit a better performance, because they are specifically trained to identify tag-candidates, which are usable in the analysis.

Besides, the classifiers of the intermediate particles also exhibit a better performance, due to the absence of background from high multiplicity events with more than 12 tracks. Hence, the specific training has a higher efficiency even before applying the cuts on the final B signal-probability, because the post-cuts on the intermediate candidates are more efficient.

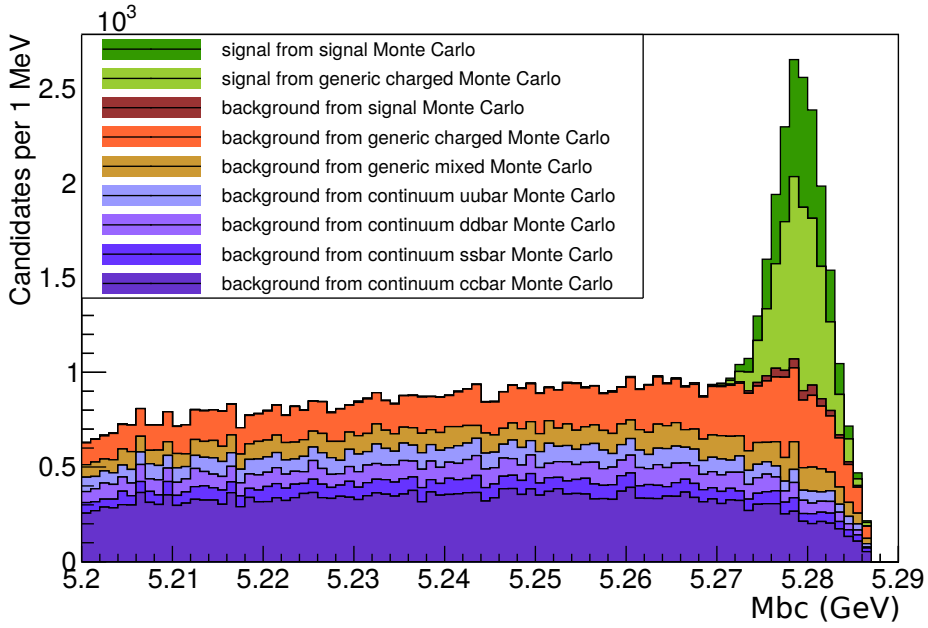
The final tagging efficiencies on the test-sample are shown in Table 6.2, Again the calculation of the efficiencies and purities was restricted to the signal-region: $M_{bc} > 5.27$ for hadronic and $|\cos\Theta_{BDl}| < 1$ for semileptonic B^+ candidates. Yet the constraint to candidates which lead to a valid $\Upsilon(4S)$ was not applied, to obtain distributions and efficiencies which are comparable to the generic FEI training.

Table 6.2.: Tagging efficiency on the independent test-sample in percent calculated in the signal-region.

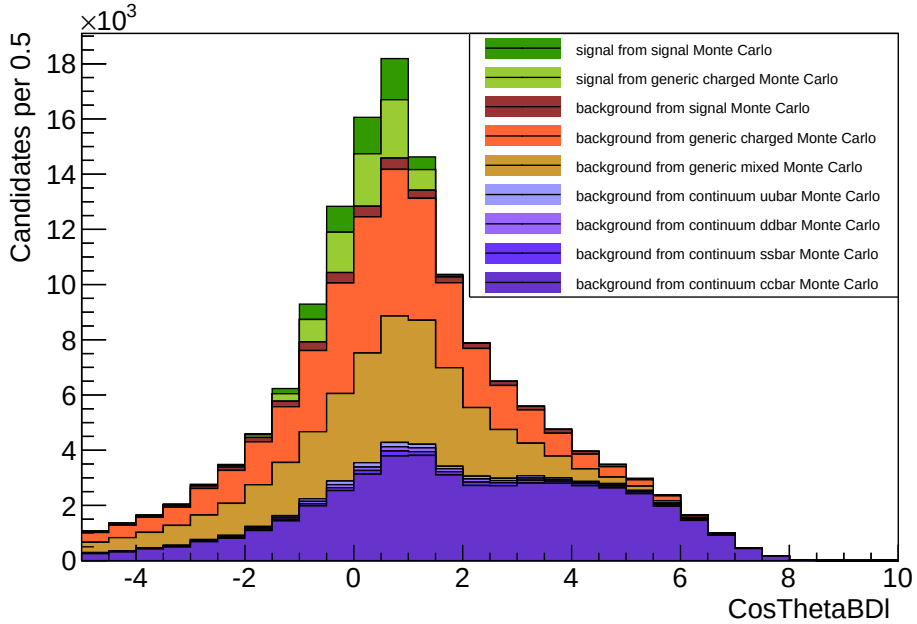
	Signal MC	Charged Generic MC	Overall
Hadronic	0.82	0.72	0.75
Semileptonic	1.26	1.11	1.16

The resulting M_{bc} and $\cos\Theta_{BDl}$ plots on the test-sample are shown in Figure 6.3. They look very similar to the generic case, however for the same efficiency in the signal region their purity is worse, because as, already mentioned, the constraint to candidates which lead to a valid $\Upsilon(4S)$ was not applied. The effects of this constraint is investigated in the next section.

[2] A $\Upsilon(4S)$ candidate is considered as valid if its rest of event contains no tracks.



(a) Hadronic candidates: M_{bc} distribution with a purity of 43 % and a tagging efficiency of 0.35 %.



(b) Semileptonic candidates: $\cos\Theta_{BDl}$ distribution with a purity of 19 % and a tagging efficiency of 0.35 %.

Figure 6.3.: Distribution of final B^+ candidates calculated on the independent test-sample.

6.3. Comparison of generic and specific FEI study

The number of correctly reconstructed $\Upsilon(4S)$ in the signal MC data sample was chosen as the key performance indicator to compare both studies. In contrast, to the previous sections the efficiency of the signal-side selection $\epsilon_{sig} = 65$ % enters the overall efficiency of the correctly reconstructed $\Upsilon(4S)$. In addition, candidates with remaining tracks in their rest of event are discarded. This cut has an efficiency of $\epsilon_{cut,had} = 87$ % and $\epsilon_{cut,sem} = 90$ % for hadronic and semileptonic candidates respectively.

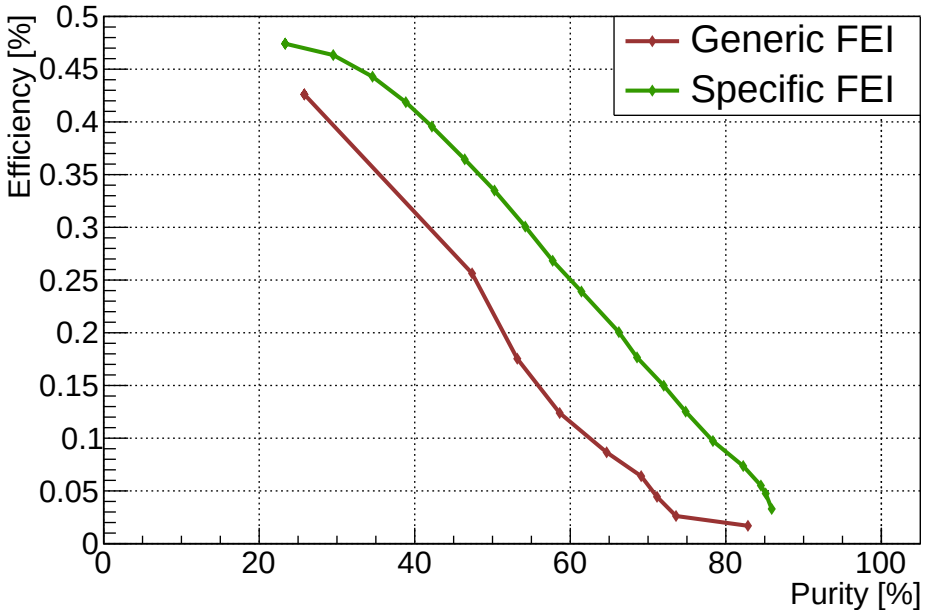
The ROC curves in Figure 6.4 show the efficiency and purity of the reconstructed $\Upsilon(4S)$ as a function of the signal-probability of the B_{tag} meson. The curves were calculated on the independent test-sample. The restriction to valid $\Upsilon(4S)$ with no remaining tracks in their respective rest of event was applied for both studies.

It is noteworthy that both generic FEI and specific FEI have roughly the same hadronic tagging efficiency, yet the specific FEI could not take advantage of all available hadronic B decay channels due to low statistics. Thus, the hadronic tagging efficiency of the specific FEI can be further improved by using more signal MC data in the training. The semileptonic tagging efficiency is higher for the analysis-specific training.

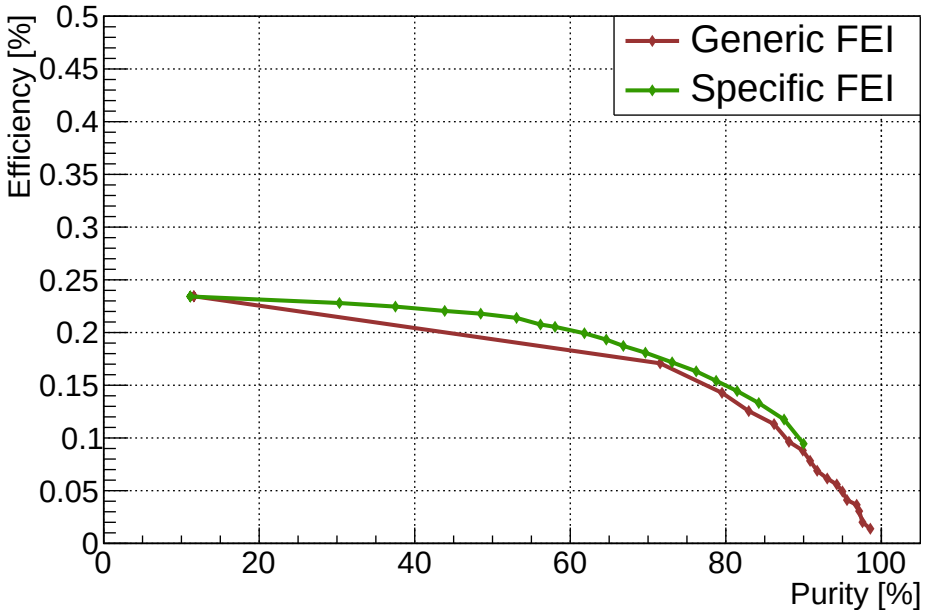
However, in both studies the final reconstruction efficiencies for $\Upsilon(4S)$ is lower than expected from the tag-side and signal-side efficiencies. This effect was not investigated yet.

Figures 6.5 and 6.6 show the distribution of the signal-probability fo the B_{tag} meson on the independent test-sample in the specific and generic training respectively. It is evident that the analysis-specific training exhibits a better performance for signal-events. Furthermore, the specific FEI requires roughly one order of magnitude less computation time and temporary storage space.

In conclusion, the analysis-specific training is superior to the generic training in terms of computation time, storage space and tagging efficiency.

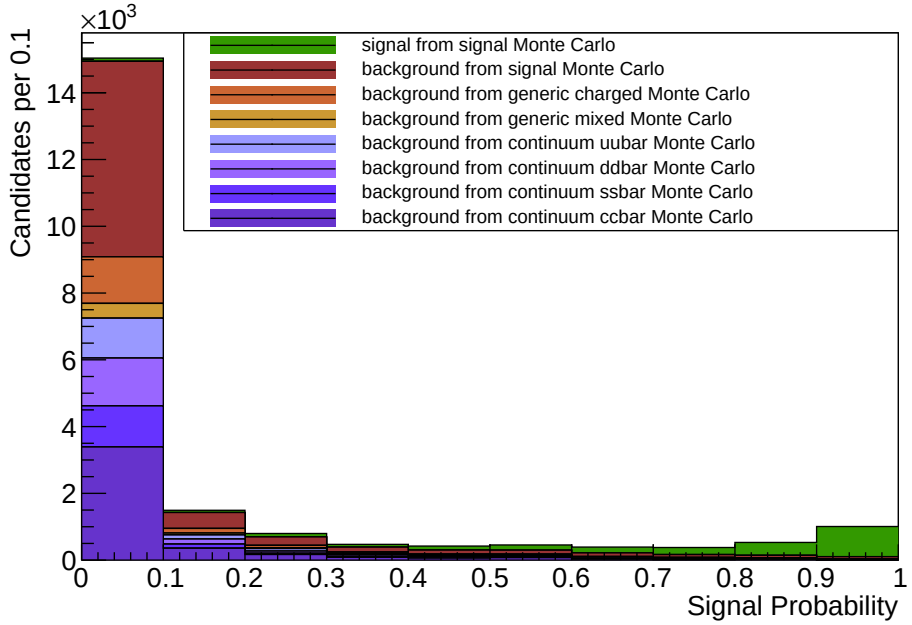


(a) Semileptonic B channels.

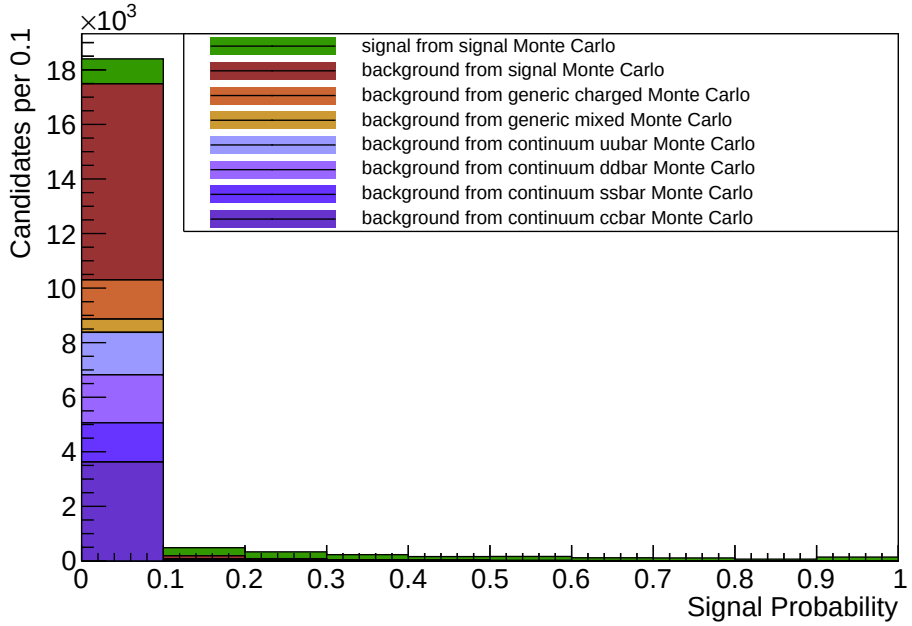


(b) Hadronic B channels.

Figure 6.4.: Comparison of the ROC curves calculated on the independent test-sample. Each point represents the efficiency and purity of the reconstructed $\Upsilon(4S)$ mesons as a function of the cut on the signal-probability of the tag-side B mesons. The efficiency corresponds to the fraction of correctly reconstructed $\Upsilon(4S)$ mesons in the signal events. The specific FEI is superior to the generic FEI even though its training sample was smaller and consequently it employed less hadronic channels in the reconstruction.

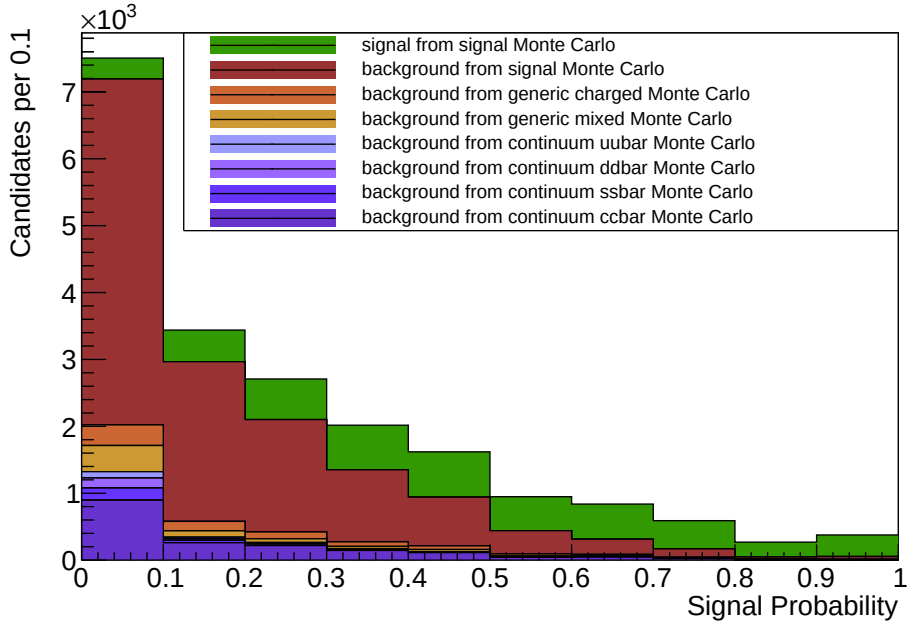


(a) Specific FEI

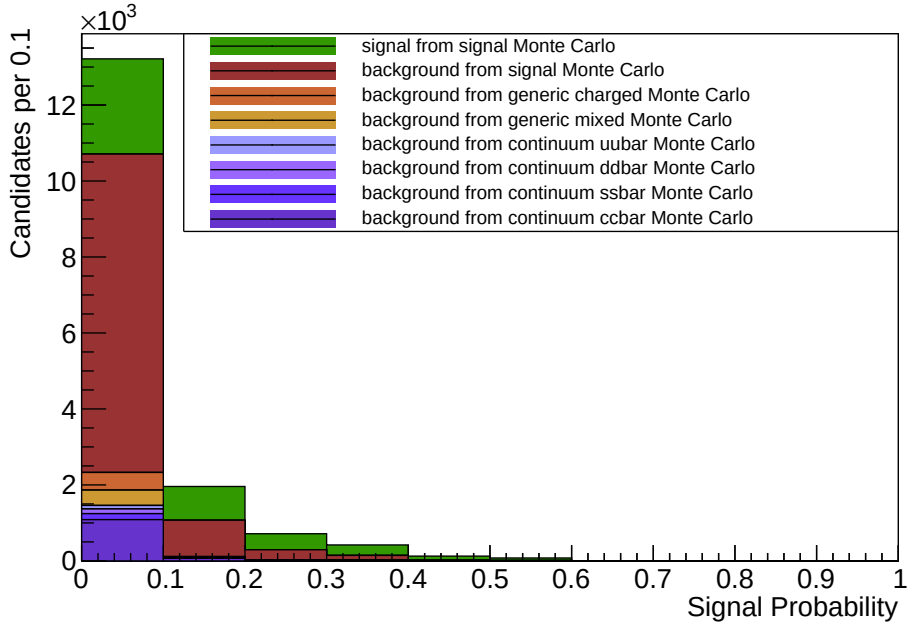


(b) Generic FEI

Figure 6.5.: Hadronic signal probability output distribution.



(a) Specific FEI



(b) Generic FEI

Figure 6.6.: Semileptonic signal probability output distribution.

7. Summary

The FEI combines hadronic and semileptonic tagging into a joint method and offers the unique possibility to be trained on an analysis-specific signal-side selection. This analysis-specific optimization reduces the required computation time and storage space by one order of magnitude. Furthermore, it improves the final efficiency and purity because the training of the multivariate classifiers take constraints arising from the signal-side into account.

The FEI was the first advanced analysis method implemented in the BASF2 framework and had a great impact on the code basis. Several libraries originally developed for the FEI are already adopted by other groups. Many analysis modules were redesigned and optimized.

First preliminary performance studies indicate a fair improvement in the tagging efficiency with respect to the FR. Consequently, the FEI will play a major role in future measurements of decays including neutrinos at Belle II.

Improvements like:

- using advanced vertex finding algorithms provided by the RAVE package [37] to reduce combinatorics;
- discarding D candidates which won't lead to a valid $\Upsilon(4S)$ candidate;
- signal-fraction extraction using dedicated control channels instead of M_{bc} and $\cos\Theta_{BDl}$;
- reconstructing π^0 candidates from single photons;
- reconstructing D candidates from semileptonic decay channels;
- and determining the four-momentum of incompletely reconstructed B candidates using the known nominal mass of intermediate particles

are under consideration, and are already anticipated in the design of the FEI algorithm. Configuration details like hyper-parameters and variables used in the multivariate analysis methods, efficiency and purity limits for intermediate cuts as well as used channels and employed Monte Carlo data are not fixed yet and will be optimized in the future. A best candidate selection was already implemented but not employed in this thesis. In addition, the influence of continuum and beam-background on the performance will be studied, as soon as the continuum suppression algorithm and the needed Monte Carlo data become available.

8. Acknowledgment

First of all, I thank Christian Pulvermacher, who is responsible for the ongoing development of the BASF2 framework and co-author of the FEI. The implementation of integral parts of the FEI was only possible due to his creative programming skills. Furthermore, I thank Bastian Kronenbitter for his reliable advice in physics questions and multivariate statistic problems. Moreover, I give thank to Dr. Thomas Kuhr and Dr. Martin Heck for inspiring discussions resulting in lots of ideas and novel approaches, which are now part of the FEI. In addition, to the above mentioned, I thank Moritz Gelb and Dr. Pablo Goldenzweig for the proofreading of my thesis. Finally, I thank Prof. Dr. Michael Feindt who awakened my interest for the topic of my thesis three years ago and constantly stimulated it further since then.

A. BASF2 Libraries

As can be seen from Listing D.1 the reconstruction of particles in BASF2 can be represented as a series of operations on `ParticleLists`. All modules use a common syntax for identifying `ParticleLists`, describing decay channels and performing cuts. The main design goal was to create intuitive, readable and consistent syntactic conventions throughout the framework. On that score, the relevant code is provided by two libraries. The `DecayDescriptor` library is responsible for the syntax of decay channels and names of the generated `ParticleLists`, whereas the `VariableManager` library provides the syntax of `Variables` and `Cuts`.

There are further libraries used by the analysis modules like the `ParticleCombiner` and `MCMatching` library. Both were reimplemented during the development of the FEI to cover previously unconsidered aspects. The description of these libraries goes beyond the scope of this thesis and is not necessary for the understanding of the FEI.

A.1. DecayDescriptor Library

The `DecayDescriptor` library contains auxiliary functions which parse `ParticleList` names and `DecayDescriptors`.

Each `ParticleList` data-object can be accessed with a unique name. All modules operating on a `ParticleList` like `ParticleSelector`, `MatchMCTruth` and `ParticleVertexFitter` require this name as a parameter. The name consists of a particle name like `e+`, `K_S0`, `D*+_s`, `J/psi`, `gamma`[1] and an optional user-defined label separated by a colon. Hence, valid `ParticleList` names are `e+`, `B0:sig` and `B+:tag`. As mentioned earlier, modules also automatically operate on the charge-conjugated `ParticleLists` e.g. `e-`, `anti-B0:sig`, `B-:tag`. Therefore, the following example code

```
selectParticle('e-:uncut')
selectParticle('e-:pidcut','eid > 0.5')
```

[1]The naming convention of evtgen is used. A list of valid names can be found in the file $BELLE2_EXTERNALS_DIR/share/evtgen/evt.pdl

creates four ParticleLists: e-:uncut, e+:uncut, e-:pidcut and e+:pidcut.

A decay channel can be encoded in a DecayDescriptor string. Modules which operate on decay channels like ParticleCombiner, PreCutHistMaker and NTupleMaker require a DecayDescriptor as a parameter. A DecayDescriptor consists of the ParticleList name of the mother particle, followed by an arrow "->" and a list of the ParticleList names of all daughter particles separated by whitespace. Hence, valid DecayDescriptors are B+:tag -> D*+ pi- and B-:sig -> tau+ nu_tau. The following example code

```
reconstructDecay('B+:sig -> D0 e+:pidcut')
```

creates two ParticleLists B+:sig and B-:sig by creating unique combinations of all particles in the D0, e+:pidcut and anti-D0, e-:pidcut ParticleLists, respectively.

A.2. VariableManager Library

The VariableManager library was originally developed for the FEI, but is now an essential part of the framework which is used in nearly all modules provided by the analysis package.

A Variable is a C++ function which takes a Particle data-object as its argument and calculates a floating-point number as its return value. In order to achieve consistency and reduce error-proneness throughout the framework all Variables are registered in the VariableManager and are automatically available to all modules. Some examples of simple Variables used in the FEI are:

- Kinematic Variables like momentum in laboratory and center of mass coordinates p, pz, pt, p_CMS, pz_CMS,...; total energy E, released energy in decay Q, invariant mass M and beam-constrained mass Mbc;
- PID Variables based on likelihood-ratios Kid, eid, muid, piid and prid as well as the contribution of the individual sub-detectors eid_dEdx, Kid_ARICH, muid_TOP, ...;
- Vertex fitting Variables like secondary vertex coordinates dx, dy, dz and the vertex/track fit probability chiProb;
- and finally target Variables like isSignal and isSignalAcceptMissingNeutrino, where the former includes candidates with missing final-state radiation and missing intermediate resonances and the later allows additionally for missing neutrinos.

Moreover, there are so-called MetaVariables, these Variables take additional parameters. Some examples of MetaVariables used in the FEI are:

- `getExtraInfo` returns the value which is stored under a given key in the extra-info field of the `Particle` data-object e.g. `getExtraInfo(SignalProbability)`,
- `daughter` returns properties of daughter particles like the invariant mass of the first daughter `daughter(0, M)` or the signal probability of the second `daughter(1,getExtraInfo(SignalProbability))`,
- `decayAngle` returns the decay angle between two daughter particles e.g. `decayAngle(0, 1)`,
- `invariantMass` returns the invariant mass of two or more daughter particles, thereby providing information about intermediate resonances like ρ or ω mesons.

As can be seen from the examples, `Variable`s are extensively used in the FEI. A nice application aside from MVC training is the determination of cuts performed on the B candidates to reduce combinatorics. Both the Full Reconstruction (FR) and the FEI use the product of the signal probabilities of all daughter particles of the B candidate to perform this cut. This calculation was hard-coded in the source code of the FR. In contrast the FEI configuration contains just an assignment of the expression `daughterProductOf(getExtraInfo(SignalProbability))` to the pre-cut `Variable` of the B particle.

On top of that, all `Variable`s are accessible in cut strings, which are accepted by many modules. The cut strings are translated by the `VariableManager` library to `Cut` C++-objects. These objects are then used by the modules to perform the cuts. The `Cut` syntax supports logic and numeric conditions. Some valid examples are:

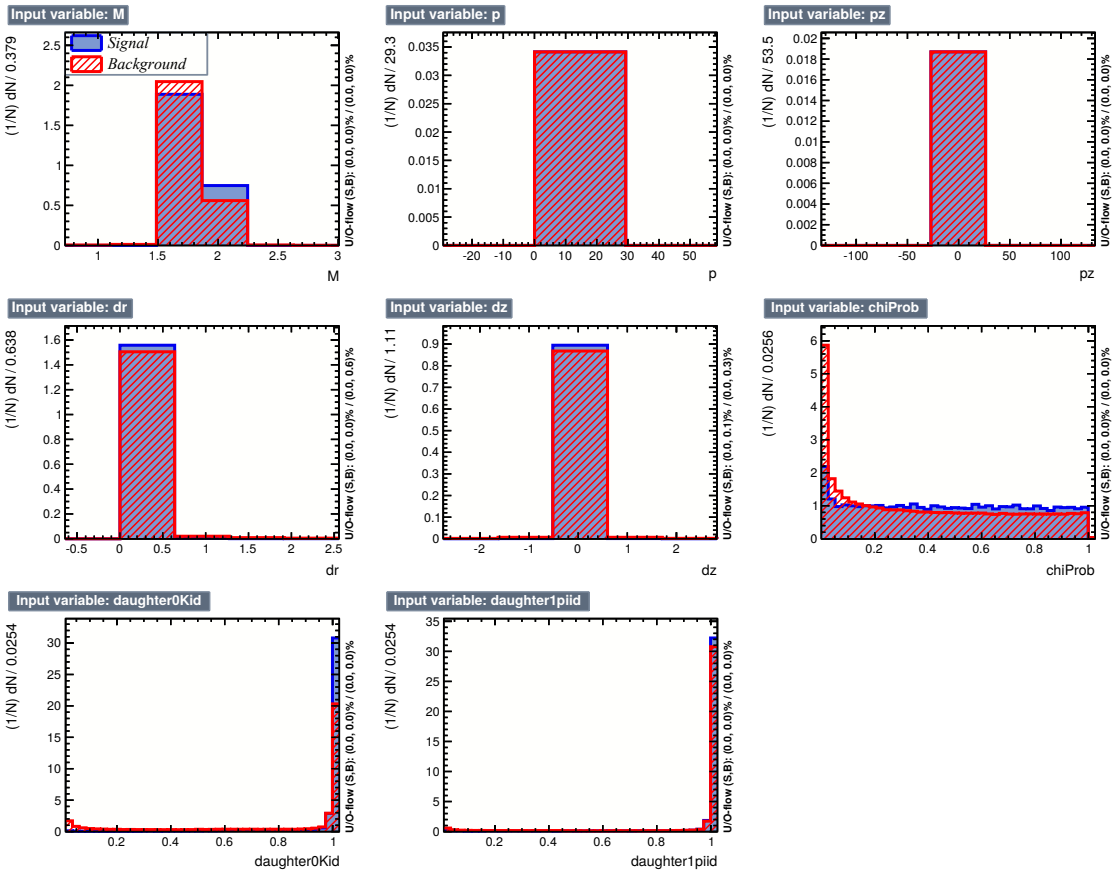
- `1.2 < M < 1.5 and isSignal == 1`
- `daughter(0, M) <= daughter(1, M) <= daughter(2, M)`
- `0.5 < getExtraInfo(SignalProbability) and [M > 1.5 or M < 0.5]`

A.3. Further TMVA control plots

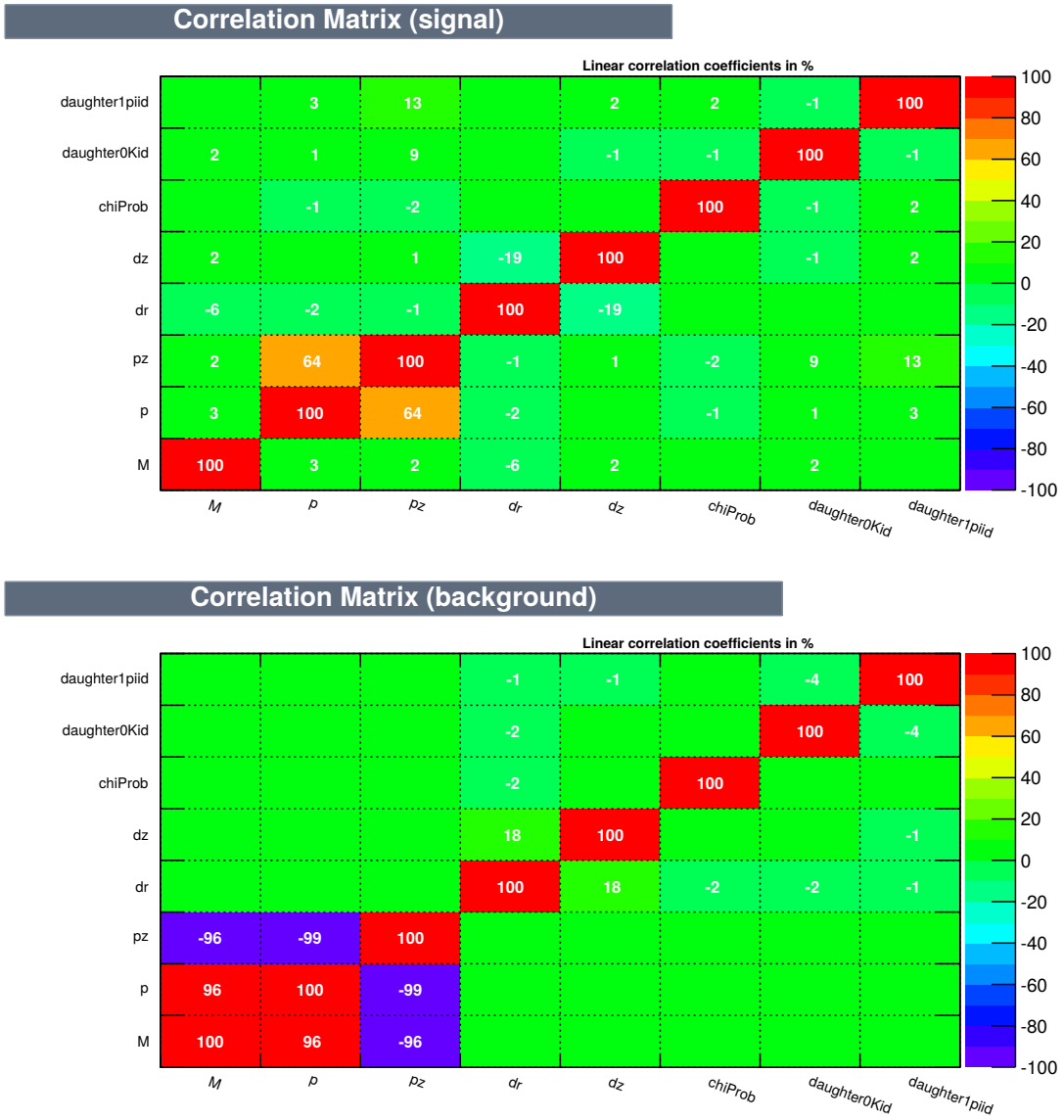
a) Input feature distributions

The separation power of individual input features as well as basic consistency checks can be inferred from the signal and background distribution of the input features.

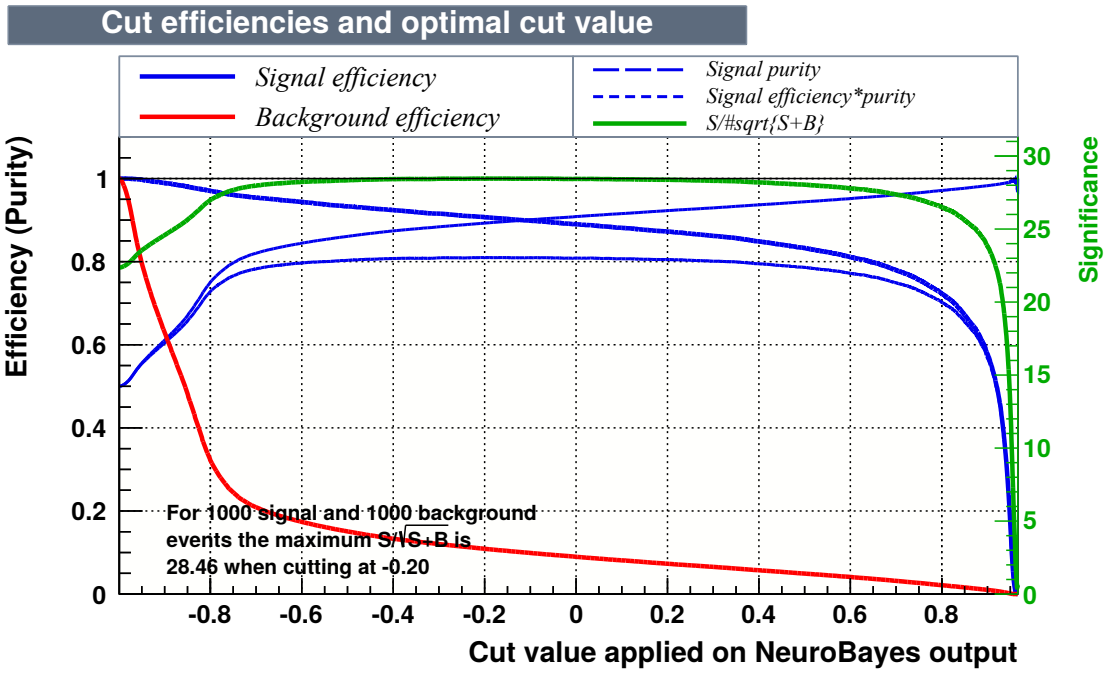
Although the default binning of TMVA (which unfortunately cannot be changed easily) is not optimal, the distributions in Figure A.1a shows some interesting properties. For example, the signal distribution of `chiProb` exhibits deviations from the theoretical expected uniform signal distribution. Furthermore, the separation power of the charged pion PID information is significantly worse than the PID information of the kaon. Both effects are worth investigating further in a real analysis.



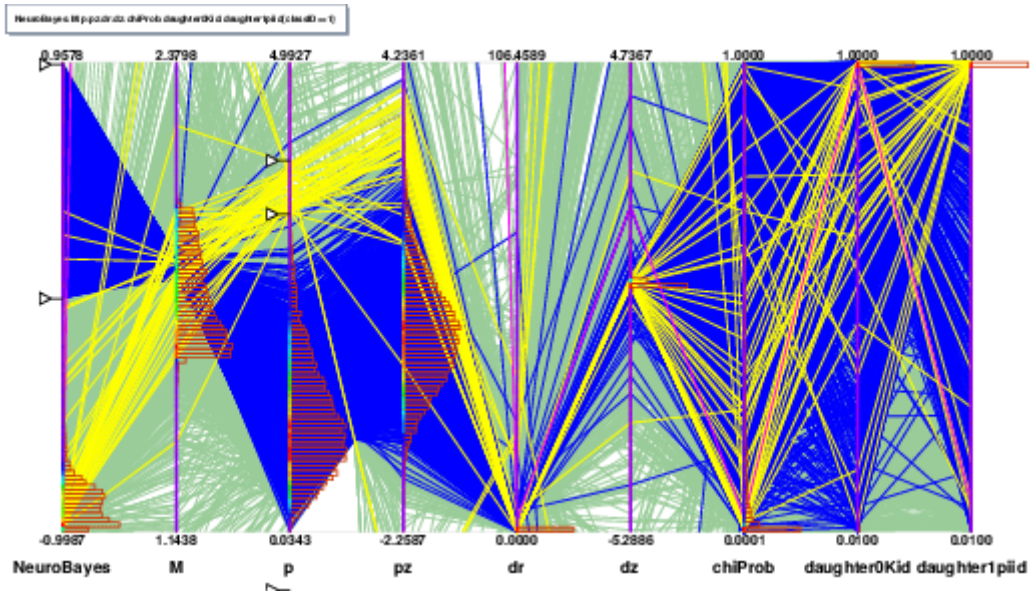
(a) Signal and background distribution for input features



(b) Correlation matrix for signal and background candidates.



(c) Cut efficiencies



(d) Parallel coordinates

Figure A.1.: TMVA control plots, which can be created on-demand from the FEI training files.

b) Correlation Matrix

The correlation matrix displays Pearson's correlation coefficient between all pairs of input features. The values are color-encoded using blue for negative correlations and red for positive correlations, whereas green indicates no correlation. The main reason to employ multivariate classification methods is to take linear and non-linear correlations between input variables into account. On the other hand, strongly linear-correlated variables pose a threat to some statistical classification methods like Fisher's linear discriminant.

The correlation matrices in Figure A.1b for signal and background exhibit strong correlations between the momentum in z direction `pz` and the total momentum `p`, which can be explained by the Lorentz boost due to the asymmetric energy of the colliding beams. Moreover, `dr`[2] and `dz` exhibit a negative correlation for signal and positive correlation for background candidates. This indicates that the difference of these quantities can distinguish between signal and background candidates.

c) Cut efficiencies

A common task is optimizing the cut on the classifier output to a figure-of-merit like the significance $\frac{S}{\sqrt{S+B}}$, where S and B are the number of signal and background events after the cut, respectively. The cut efficiency plot of TMVA displays this and other efficiency variables as a function of the cut on the classifier output.

The cut efficiencies for the example steering file are shown in Figure A.1c. The trained classifier shows a significance near to 28 over a wide range of cuts. Therefore, the performance of the classification changes only slightly if the cut on the classifier output is changed inside this region. A cut on this classifier output is stable.

d) Parallel coordinates

In parallel coordinates the classifier response and the high-dimensional input data is projected into a plane. Each dimension corresponds to a parallel coordinate axis and each data-point is represented by a line connecting the coordinates axes. Clusters and correlations between features can be identified in this presentation. Distributions in all features can be seen at once with possible cuts on multiple features. In fact, the produced plot is interactive and data-points falling into a user-defined range in arbitrary features can be highlighted with different colours by manipulating interactive sliders.

[2] `dr` is the transverse distance of the secondary vertex with respect to the IP

In Figure A.1d the parallel coordinates for the background candidates of the example $D^0 \to K^-\pi^+$ training are shown. The first axis on the left shows the output of the trained classifier. The two blue arrow-shaped markers select the wrongly classified candidates with a classifier output above 0. Due to this selection all blue lines in the plot correspond to candidates with a classifier output above 0. One can see that all blue lines converge to a small region around the D^0 mass on the `M` feature-axis. So these candidates are wrongly classified due to their small deviation from the D^0 mass.

The two yellow markers on the third axis `p` select candidates with high momentum. Most of the selected yellow lines run in parallel to each other and cross the `pz` axis at high values. This illustrates the linear correlation between `p` and `pz` for the selected high momentum tracks due to the asymmetric energy of the colliding beams.

B. Dependency Graph Meta Framework

The dependency graph meta framework (`actorFramework`) allows to connect parameters of functions to returned values of other functions. The resulting dependencies are modelled using a directed acyclic graph (DAG). This type of graph is commonly used in computer science to model data processing networks and causal relations, because a DAG always has a topological ordering such that every node occurs before a node it is connected to. This ordering is not necessarily unique, however the remaining freedom can be used to optimize the concurrency of the system.

B.1. General Structure

The `actorFramework` is motivated by the structure of a theatre play. Actors interact with properties and other actors in the play producing complex dependencies among themselves. The framework uses so-called Resources as basic building blocks to model dependencies. A Resource is an arbitrary Python object stored by the framework in an associative container using a unique identifier. The `actorFramework` defines three Python classes which operate on these Resources:

- `Property` adds a new Resource to the framework using an unique identifier and a Python object;
- `Collection` joins a list of required Resources to a new Resource with a common identifier;
- `Actor` passes required Resources as arguments to a function and provides new Resources extracted from the return value of the function.

In addition, the framework defines a `Play` class which stores instances of the above classes, solves the dependencies between them and calls the `Actor` functions in the correct order.

The example in Listing B.1 incorporates the basic concept of the `actorFramework`.

Listing B.1: Academic example

```
1 play = Play()
2 play.addProperty('a', 2)
3 play.addProperty('b', 3)
4 play.addActor(lambda a,b: {'sum': a + b})
5 play.addActor(lambda a,b: {'diff': a - b})
6 play.addActor(lambda x,y: {'z': x/y},x='diff',y='sum')
7 play.run(...)
```

First a new `Play` is created and then the values a=*2* and b=*3* are added as `Property` objects to the `Play`. Afterwards, `Actor`s calculating the sum and difference of the two values are added. Finally, the quotient of the difference and sum is calculated by another `Actor`. The dependency of the sum and difference on a and b is implicitly derived by the framework using the parameter names of the function. Whereas the dependency of the quotient on *sum* and *diff* is explicitly stated, by connecting the parameters x and y with the identifiers of the required Resources *diff* and *sum*. As soon as the run method of the `Play` is called, the dependencies between the `Property` objects and `Actor`s are solved and all `Actor`s are called with their required parameters.

The framework adds an additional layer of abstraction and offers several advantages over using BASF2 directly:

- dependencies between modules are explicit, therefore errors are already noticed at steering level instead of path level;
- modules added to the path which are not needed in the current run are discarded;
- the generated path is optimized for parallel processing;
- a hash of all dependencies is provided which can be used as prefix for weight-files;
- the developer is forced to split up her code into small independent parts, which can easily be unit-tested;
- and these independent parts of the steering file can automatically run in parallel.

B.2. Illustrating Example

The following example uses the `actorFramework` to reconstruct the decay chain

$$\begin{aligned}\bar{B} &\to D^{*+}K^- \\ &\hookrightarrow D^0\pi^+ \\ &\hookrightarrow K^-\pi^+\pi^0 \\ &\hookrightarrow \gamma\gamma\end{aligned}$$

and to train a multivariate analysis method for each appearing particle. Thereby, the signal probabilities of the daughters particles calculated by the methods are used as input for the method of the mother particle. In the following Listing D.2 is explained step by step.

```
1  from basf2 import *
2  from modularAnalysis import *
3  import actorFramework
```

At first all identifiers from `basf2` and `modularAnalysis` Python modules are loaded into the current namespace. In addition, the `actorFramework` Python module is loaded.

```
4  # Define needed final-state particles and
5  # decay topology.
6  my_fsps= ['K-','pi+','gamma']
7  my_decays = ['pi0 -> gamma gamma',
8               'D0 -> K- pi+ pi0',
9               'D*+ -> D0 pi+',
10              'anti-B0 -> D*+ K-']
11 # Extract particle name from ParticleList name.
12 def extract_name(plist):
13   return plist.split(':')[0]
```

The used final-state particles and the decays which should be reconstructed are defined using the `DecayDescriptor` syntax. Thereafter, an auxiliary function is defined, which returns the name of a particle by splitting a provided `ParticleList` name into the name of the corresponding particle and the user-defined label.

```
14 # Select all Particles with given name into
15 # a ParticleList.
16 def select(path, hash, name):
17   plist = name + ':' + hash
18   selectParticle(plist, path=path)
19   return {'Particle_' + name: plist}
20
```

```
21 # Reconstruct given particle out of its daughters.
22 def reconstruct(path, hash, name, daughters):
23   plist = name + ':' + hash
24   reconstructDecay(plist+' -> '+' '.join(daughters),
25                    path=path)
26   matchMCTruth(plist, path=path)
27   return {'Particle_' + name: plist}
```

The reconstruction part is split into selection of final-state particles and reconstruction of a decay. Both functions generate a unique `ParticleList` name using the provided hash argument, add the necessary modules, which generate these `ParticleLists` to the provided path and return finally a Resource containing the name of the generated `ParticleList`. These functions are later promoted to `Actors`.

```
28 # Train standard MVA method with given ParticleList.
29 def train(path, hash, plist, variables, probabilities):
30   if isTMVAMethodAvailable(hash):
31       return {'Prefix_' + extract_name(plist): hash}
32   trainTMVAMethod(plist, variables, prefix=hash,
33                   path=path)
34   return {}
35
36 # Apply standard MVA method on given ParticleList.
37 def apply(path, hash, plist, prefix):
38   applyTMVAMethod(plist, prefix=prefix, path=path)
39   return {'Probability_' + extract_name(plist): hash}
```

The multivariate analysis part is split into training and application of the method. The `train` function checks if the weight-file, identified by a unique prefix, of the requested TMVA training is already available. If so, a Resource containing the prefix of the weight-file is provided. Otherwise the `TMVATeacher` module is added to the path and the function returns an empty dictionary. In consequence, `Actors` requiring the Resource provided by `train`, cannot run until the corresponding method was trained.

The `apply` function requires the prefix of a weight-file and adds the corresponding `TMVAExpert` module to the provided path. The `TMVAExpert` module calculates the signal probability of every `Particle` in the passed `ParticleList` and stores it in the `ExtraInfo` field of the `Particle` data-object under the key `isSignal`. Both functions are later promoted to `Actors` as well.

The unique prefix is provided by the special *hash* Resource. It contains a SHA-1 hash[1] of all Resources required by the `Actor`. If a Resource changes, the hash changes

[1]The secure hash algorithm (SHA) was developed in 1994 by the National Institute of Standards and Technology in collaboration with the National Security Agency. Up to now there are no known collisions.

and the MVA methods in the example are automatically retrained, because the hash is used as a prefix for the weight-files.

```
40 # Create new Play and add MVA variables as properties.
41 play = actorFramework.Play()
42 play.addProperty('variables_K-',['Kid', 'p', 'pt'])
43 play.addProperty('variables_pi+',['piid', 'p', 'pt'])
44 play.addProperty('variables_gamma',['clusterReg',
45     'clusterE9E25'])
46 play.addProperty('variables',['M','p', 'pt', 'p_CMS',
47     'E', 'Q', 'daughter(0, getExtraInfo(isSignal))',
48     'daughter(1, getExtraInfo(isSignal))'])
```

A new `Play` is created using the `actorFramework` module. The `Variables` which should be used in the training for the final-state particles and in the training of the decay channels are added as a `Property` to the `Play`. All `Variables` registered in the `VariableManager` library can be used here. For simplicity only a small number of `Variables` are used in this example.

In particular, the decay channel `Variables` depend on the MVA method output of the daughter particles via `daughter(0, getExtraInfo(isSignal))` and `daughter(1, getExtraInfo(isSignal))`. Therefore, it is crucial that all daughter particle MVA methods are already trained and applied before the training of the mother particle begins.

```
49 # Add actors to play, which take care of
50 # final-state particles.
51 for fsp in my_fsps:
52   play.addProperty(fsp, fsp)
53   play.addActor(select, name=fsp)
54   play.addActor(train, plist='Particle_'+fsp,
55             variables='variables_'+fsp,
56             probabilities='None')
57   play.addActor(apply,plist='Particle_'+fsp,
58                     prefix='Prefix_'+fsp)
59
60 # Add actors to play, which take care of
61 # combined particles.
62 for decay in my_decays:
63   mother = decay.split(' ')[0]
64   daughters = decay.split(' ')[2:]
65   play.addProperty(mother, mother)
66   play.addActor(reconstruct, name=mother,
67    daughters=['Particle_'+d for d in daughters])
68   play.addActor(train, plist='Particle_'+mother,
69    probabilities=['Probability_'+d for d in daughters])
```

```
70    play.addActor(apply, plist='Particle_'+mother,
71                  prefix='Prefix_'+mother)
```

Now the dependencies between the defined functions are declared by promoting the functions to `Actor`s. For every particle three `Actor`s are added to the `Play`:

- `select`/`reconstruct` generates the `ParticleList` containing candidates of the particle;
- `train` trains a multivariate method on the `ParticleList`;
- `apply` calculates the signal probability for each candidate in the `ParticleList`.

The Resource keys assigned to the parameters of the functions define the dependencies between the `Actor`s. The resulting dependency graph is visualized in Figure B.1.

Although this seems complicated at first, the additional abstraction level provided by the `actorFramework` pays off as soon as more decay channels and additional reconstruction steps, like the computation of pre-cuts and vertex fitting, are considered.

```
72 # Define output of B0 MVA method as needed.
73 play.addNeeded('Probability_anti-B0')
```

The final $\bar{B}^0$ signal probability is marked as a needed Resource, thereby the framework can automatically infer which BASF2 modules in the final path are actually required to create this Resource. Other modules are discarded. This strategy is important in more complicated situations frequently encountered in the FEI. For example, if all J/ψ channels in the B reconstruction have to be discarded due to low training statistics, this mechanism discards the reconstruction of the J/ψ altogether.

```
74 # Add input modules and run play.
75 path = create_path()
76 path.add_module(register_module('RootInput'))
77 path.add_module(register_module('ParticleLoader'))
78 play.run(path, nProcesses=2)
79 # Now run the generated path with BASF2.
80 process(path)
```

Finally, a BASF2 path is created and passed to the run method of `play`. The dependencies between the `Actor`s are solved, and all needed modules are added to the given path. As can be seen from the `nProcesses` argument, the `Actor`s can be executed in parallel. However, the presented example does not profit from this, because CPU-intensive operations like the training of the MVA methods are done on path level. In contrast, the FEI trains its multivariate analysis methods on steering level inside an Actor. Hence, the parallel execution of Actors automatically runs these trainings in parallel.

In contrast to the previous example in Listing D.1 which reconstructed the decay $D^0 \to K^-\pi^+$ using the basic BASF2 modules, the example using the `actorFramework` can easily be extended to hundreds of decay channels and additional reconstruction steps.

B.3. Code Quality

The FEI algorithm consists of approximately 2500 lines of Python code. To ensure a good code quality throughout the project the `actorFramework` and every `Actor` is tested using the Python `unittest` package. The test-cases for the `Actor`s cover the correctness of:

- returned Resources;
- parameters of modules added to the path;
- and conditional jumps depending on the existence of weight-files.

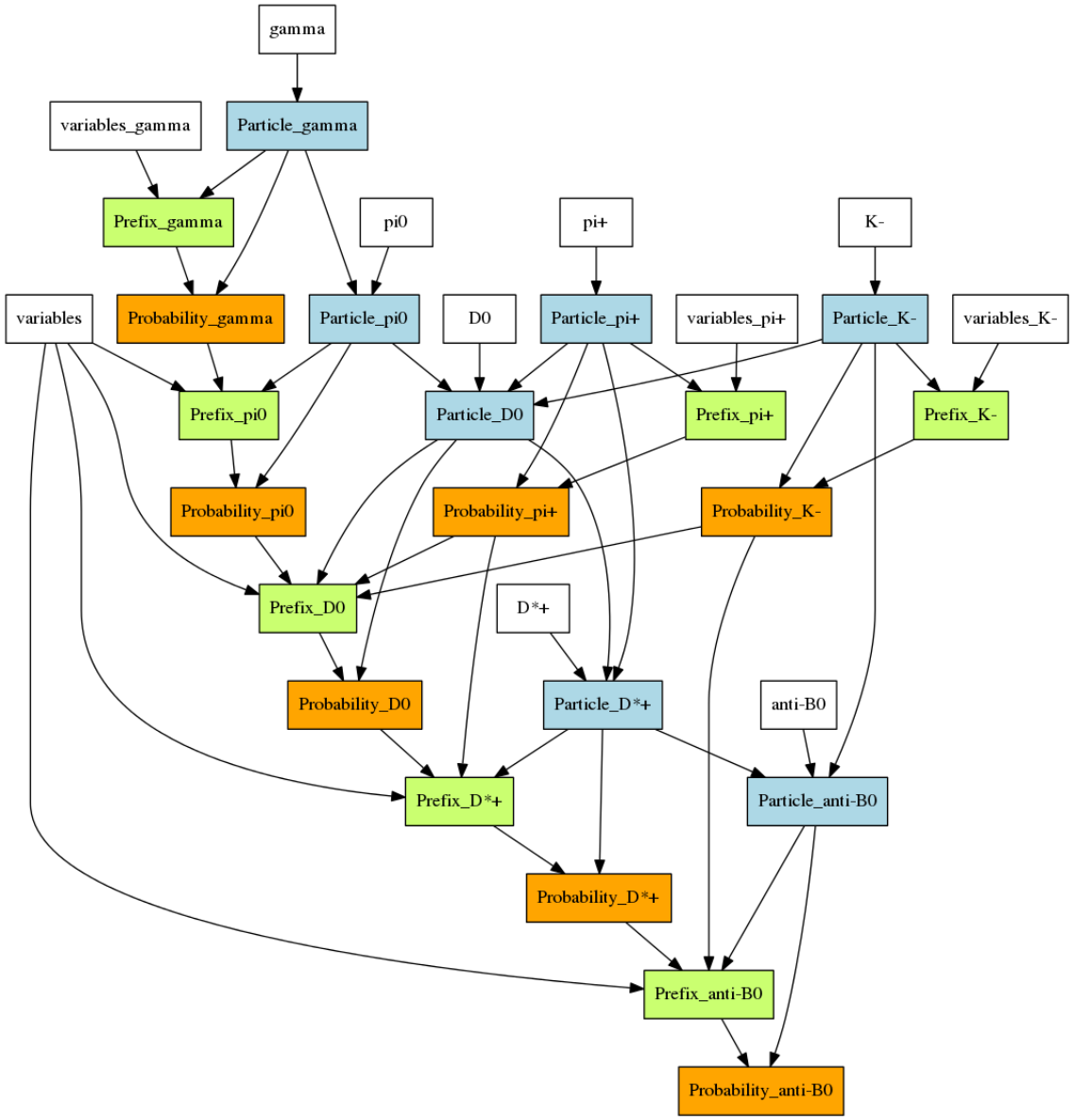


Figure B.1.: Dependency Graph generated by the `actorFramework` for Listing D.2. Resources created by `Property` objects are white. Resources created by `select` and `reconstruct` are blue, `apply` are orange and `train` are green.

C. Full Event Interpretation Configuration

The decay topology which is reconstructed by the FEI is defined by the user in terms of `Particle` Python objects[1]. A `Particle` object is defined by:

- a `list` of decay channels;
- an `MVAConfiguration` object for the used MVC;
- a `PreCutConfiguration` object for the pre-cut determination;
- and a `PostCutConfiguration` object for the post-cut determination.

For each `Particle` the algorithm creates signal and background distributions for the defined pre-cut `Variable`, calculates individual pre-cuts for every channel based on provided efficiency and purity parameters and trains a MVC to calculate a signal probability for each reconstructed candidate.

Listing D.3 shows a simple example configuration for the FEI. Some parts like `Variable`s used in the MVC and TMVA config strings are omitted (marked with ...) for simplicity.

C.1. Final-State Particle Configuration

```
from FullEventInterpretation import *
particles = []
# Define charged final-state particles
mva_fsp = Particle.MVAConfiguration(
                name='FastBDT',
                type='Plugin',
                config='...',
                variables=[...],
                target='isSignal')
postCut_fsp = Particle.PostCutConfiguration(
```

[1]Not to be confused with the `Belle2::Particle` C++ data-objects used on path level to store information about particle candidates in the `DataStore`.

```
11                  value=0.1)
12 particles.append(Particle('e+', mva_fsp, postCut_fsp))
13 particles.append(Particle('mu+', mva_fsp, postCut_fsp))
14 particles.append(Particle('pi+', mva_fsp, postCut_fsp))
15 particles.append(Particle('K+', mva_fsp, postCut_fsp))
```

At first all final-state particles which shall be considered are added to the `list` of `Particles`. The first parameter of the `Particle` object must be a valid `ParticleList` name. In particular, one can add a user label to this name. In addition, each `Particle` object receives an `MVAConfiguration` object. This configuration object contains:

- the name of the MVC e.g. FastBDT, NeuroBayes or TMVA-BDT;
- the type of the method, which is Plugin for FastBDT and NeuroBayes;
- a configuration string, where among other things the default values of the hyper-parameters of the methods can be overwritten;
- a `list` of `Variables`, which are used in the MVC;
- the `Variable` which defines signal and background by returning 1 for a signal and 0 for a background candidate.

The last parameter is an optional `PostCutConfiguration` object. All candidates with a signal probability below the value defined in this configuration are discarded.

C.2. Intermediate Particle Configuration

```
16 # Define D0
17 mva_D0 = Particle.MVAConfiguration(
18                  name='FastBDT',
19                  type='Plugin',
20                  config='...',
21                  variables=[...],
22                  target='isSignal')
23 pre_D0 = Particle.PreCutConfiguration(
24                  variable='M',
25                  binning=(500,1.5,2.0),
26                  efficiency=0.95,
27                  purity=0.001,
28                  userCut='')
29 p = Particle('D0', mva_D0, pre_D0)
30 p.addChannel(['K-', 'pi+'])
31 p.addChannel(['K-', 'pi+', 'pi+', 'pi-'])
32 particles.append(p)
```

Subsequently, all intermediate particles are added to the `Particles list`. In this example only the D^0 is added for simplicity. Just as the final-state `Particle`, each intermediate `Particle` receives an `MVAConfiguration` which is used to train a MVC for each of its channels. Furthermore, an intermediate `Particle` expects a `PreCutConfiguration` object which contains:

- the `Variable` on which the pre-cut is performed,
- the binning which is used to create the PDFs for signal and background in the pre-cut `Variable`, the lower and upper boundary of this binning (the second and third number respectively) define the minimum cut which is applied to all channels,
- the desired efficiency of the pre-cut,
- the minimal purity after the pre-cut,
- an additional user-defined cut which is always applied even before the signal and background PDFs are created.

Again the last parameter could be an optional `PostCutConfiguration` object, which is not used in this example. The decay channels of the `Particle` are added with the `addChannel` method, which expects a `list` of valid `ParticleList` names. Optionally the method accepts an additional `MVAConfiguration`-object, which is used for this channel instead of the default `MVAConfiguration` passed to the `Particle` constructor.

C.3. Final Particle Configuration

```
33 # Define B+
34 mva_BPlus = Particle.MVAConfiguration(
35                  name='FastBDT',
36                  type='Plugin',
37                  config='...',
38                  variables=[...],
39                  target='isSignal')
40 pre_BPlus = Particle.PreCutConfiguration(
41     variable='daughterProductOf('+
42              'getExtraInfo(SignalProbability))',
43     binning=list(...),
44     efficiency=0.95,
45     purity=0.0001,
46     userCut='Mbc > 5.2 and abs(deltaE) < 0.5')
47 p = Particle('B+', mva_BPlus, pre_BPlus)
48 p.addChannel(['anti-D0', 'pi+'])
```

```
49 p.addChannel(['anti-D0', 'pi+', 'pi+', 'pi-'])
50 particles.append(p)
```

The final B^+ `Particle`, which is reconstructed from hadronic channels, is defined. As mentioned before, the final B mesons differ in their reconstruction only in the `userCut` and the pre-cut `Variable`. Instead of an equidistant binning, a user-defined binning is used with a smaller bin-width for smaller values, which is more suitable for the used pre-cut `Variable`. The binning is defined by a `list` of bin boundary values, which is omitted for simplicity.

```
51 mva_BPlusSemileptonic = Particle.MVAConfiguration(
52                  name='FastBDT',
53                  type='Plugin',
54                  config='...',
55                  variables=[...],
56                  target='isSignalAcceptMissingNeutrino')
57
58 pre_BPlusSemileptonic = Particle.PreCutConfiguration(
59     variable='daughterProductOf('+
60              'getExtraInfo(SignalProbability))',
61     binning=list(...),
62     efficiency=0.95,
63     purity=0.0001,
64     userCut='')
65
66 p = Particle('B+:semileptonic', mva_BPlusSemileptonic,
67                                 pre_BPlusSemileptonic)
68 p.addChannel(['anti-D0', 'e+'])
69 p.addChannel(['anti-D0', 'mu+'])
70 particles.append(p)
```

Another B^+ `Particle` is defined, which is reconstructed from semileptonic channels. The label semileptonic is used to distinguish it from the other B. In comparison with the hadronic B configuration, the target `Variable` allows additionally for a missing neutrino in the reconstruction and no `userCut` is applied.

C.4. Analysis-Specific Training

After all `Particles` are defined and configured the FEI can be employed in an analysis.

```
71 selection = create_path()
72 ...
73 analysis = create_path()
74 ...
```

```
75 main = FullEventInterpretation(selection, analysis,
76                                particles)
77 process(main)
```

The user can define an analysis-specific `selection` path. This path has to reconstruct the signal-side B_{sig} candidates and create an `RestOfEvent` data-object for these candidates using the `RestOfEventBuilder`. Afterwards, the FEI is automatically trained on the rest of the event. This means only `Track`s, `ECLCluster`s and `KLMCluster`s are considered which are not used for the signal-side candidate. Moreover, the reconstructed B_{tag} candidates have to use up all remaining `Track`s otherwise they are discarded. If no `selection` path is provided, the FEI is trained without taking the signal-side into account.

Furthermore, the user has to provide an `analysis` path, which is executed as soon as the FEI is completely trained. In this path the user can access the B_{tag} meson `ParticleList` and perform her analysis.

D. Listings

D.1. Basic Example using BASF2 directly

Listing D.1: Illustrating example reconstructing $D^0 \to K^-\pi^+$

```
from basf2 import *
from modularAnalysis import *
# Read data from files given via command line argument.
analysis_main.add_module(register_module('RootInput'))
# Load Belle II detector parameters file.
loadGearbox()
# Create FSP hypotheses for tracks and clusters.
loadReconstructedParticles()
# Collect all kaon and pion hypotheses into
# ParticleLists and apply cuts on the PID information.
selectParticle('K-','Kid > 0.01')
selectParticle('pi+','piid > 0.01')
# Combines the hypotheses to D0 candidates and
# cut on the invariant mass of each candidate.
reconstructDecay('D0 -> K- pi+','1.6 < M < 2.0',
                 persistent=True)
# Match the MC Truth of all D0 candidates
matchMCTruth('D0')
# Fit the secondary vertex of all D0 candidates and
# apply a soft cut on the p value of the fit.
fitVertex('D0',0.0001)
# Train the standard MVC.
# If the method is already trained, calculate the
# signal probability and save it together with its
# mass and mcStatus in an ntuple file.
if isTMVAMethodAvailable():
    applyTMVAMethod('D0')
    variablesToNTuple('D0',['getExtraInfo(isSignal)',
                            'mcStatus', 'M'])
else:
    trainTMVAMethod('D0',['M','p','pz','dr','dz',
                          'chiProb','daughter(0,Kid)',
```

```
33                              'daughter(1,piid)'])
34 # Write out the D0 candidates.
35 analysis_main.add_module(register_module('RootOutput'))
36 # Execute the path.
37 process(analysis_main)
```

D.2. Advanced Example using the actorFramework

Listing D.2: Actor framework example

```
from basf2 import *
from modularAnalysis import *
import actorFramework
# Define needed final-state particles and
# decay topology.
my_fsps= ['K-','pi+','gamma']
my_decays = ['pi0 -> gamma gamma',
             'D0 -> K- pi+ pi0',
             'D*+ -> D0 pi+',
             'anti-B0 -> D*+ K-']
# Extract particle name from ParticleList name.
def extract_name(plist):
  return plist.split(':')[0]
# Select all particles with given name into
# a ParticleList.
def select(path, hash, name):
  plist = name + ':' + hash
  selectParticle(plist, path=path)
  return {'Particle_' + name: plist}

# Reconstruct given particle out of its daughters.
def reconstruct(path, hash, name, daughters):
  plist = name + ':' + hash
  reconstructDecay(plist+' -> '+' '.join(daughters),
                   path=path)
  matchMCTruth(plist, path=path)
  return {'Particle_' + name: plist}
# Train standard MVC method with given ParticleList.
def train(path, hash, plist, variables, probabilities):
  if isTMVAMethodAvailable(hash):
      return {'Prefix_' + extract_name(plist): hash}
  trainTMVAMethod(plist, variables, prefix=hash,
                  path=path)
  return {}

# Apply standard MVA method on given ParticleList.
def apply(path, hash, plist, prefix):
  applyTMVAMethod(plist, prefix=prefix, path=path)
  return {'Probability_' + extract_name(plist): hash}

# Create new Play and add MVA variables as properties.
```

```
42 play = actorFramework.Play()
43 play.addProperty('variables_K-',['Kid', 'p', 'pt'])
44 play.addProperty('variables_pi+',['piid', 'p', 'pt'])
45 play.addProperty('variables_gamma',['clusterReg',
46      'clusterE9E25'])
47 play.addProperty('variables',['M','p', 'pt', 'p_CMS',
48      'E', 'Q', 'daughter(0, getExtraInfo(isSignal))',
49      'daughter(1, getExtraInfo(isSignal))'])
50 # Add actors to play, which take care of
51 # final-state particles.
52 for fsp in my_fsps:
53   play.addProperty(fsp, fsp)
54   play.addActor(select, name=fsp)
55   play.addActor(train, plist='Particle_'+fsp,
56                 variables='variables_'+fsp,
57                 probabilities='None')
58   play.addActor(apply, plist='Particle_'+fsp,
59                        prefix='Prefix_'+fsp)
60
61 # Add actors to play, which take care of
62 # combined particles.
63 for decay in my_decays:
64   mother = decay.split(' ')[0]
65   daughters = decay.split(' ')[2:]
66   play.addProperty(mother, mother)
67   play.addActor(reconstruct, name=mother,
68    daughters=['Particle_'+d for d in daughters])
69   play.addActor(train, plist='Particle_'+mother,
70    probabilities=['Probability_'+d for d in daughters])
71   play.addActor(apply, plist='Particle_'+mother,
72                 prefix='Prefix_'+mother)
73 # Define output of B0 MVA method as needed.
74 play.addNeeded('Probability_anti-B0')
75 # Add input modules and run play.
76 path = create_path()
77 path.add_module(register_module('RootInput'))
78 path.add_module(register_module('ParticleLoader'))
79 play.run(path, nProcesses=2)
80 # Now run the generated path with BASF2.
81 process(path)
```

D.3. Example Configuration of the FEI

Listing D.3: Full Event Interpretation configuration file

```
from FullEventInterpretation import *
particles = []
# Define charged final-state particles
mva_fsp = Particle.MVAConfiguration(
                name='FastBDT',
                type='Plugin',
                config='...',
                variables=[...],
                target='isSignal')
postCut_fsp = Particle.PostCutConfiguration(
                value=0.1)
particles.append(Particle('e+', mva_fsp, postCut_fsp))
particles.append(Particle('mu+', mva_fsp, postCut_fsp))
particles.append(Particle('pi+', mva_fsp, postCut_fsp))
particles.append(Particle('K+', mva_fsp, postCut_fsp))
# Define D0
mva_D0 = Particle.MVAConfiguration(
                name='FastBDT',
                type='Plugin',
                config='...',
                variables=[...],
                target='isSignal')
pre_D0 = Particle.PreCutConfiguration(
                variable='M',
                binning=(500,1.5,2.0),
                efficiency=0.95,
                purity=0.001,
                userCut='')
p = Particle('D0', mva_D0, pre_D0)
p.addChannel(['K-', 'pi+'])
p.addChannel(['K-', 'pi+', 'pi+', 'pi-'])
particles.append(p)
# Define B+
mva_BPlus = Particle.MVAConfiguration(
                name='FastBDT',
                type='Plugin',
                config='...',
                variables=[...],
                target='isSignal')
pre_BPlus = Particle.PreCutConfiguration(
    variable='daughterProductOf('+
```

```
42                'getExtraInfo(SignalProbability))',
43      binning=list(...),
44      efficiency=0.95,
45      purity=0.0001,
46      userCut='Mbc > 5.2 and abs(deltaE) < 0.5')
47  p = Particle('B+', mva_BPlus, pre_BPlus)
48  p.addChannel(['anti-D0', 'pi+'])
49  p.addChannel(['anti-D0', 'pi+', 'pi+', 'pi-'])
50  particles.append(p)
51  mva_BPlusSemileptonic = Particle.MVAConfiguration(
52                  name='FastBDT',
53                  type='Plugin',
54                  config='...',
55                  variables=[...],
56                  target='isSignalAcceptMissingNeutrino')
57
58  pre_BPlusSemileptonic = Particle.PreCutConfiguration(
59      variable='daughterProductOf('+
60                'getExtraInfo(SignalProbability))',
61      binning=list(...),
62      efficiency=0.95,
63      purity=0.0001,
64      userCut='')
65
66  p = Particle('B+:semileptonic', mva_BPlusSemileptonic,
67                                   pre_BPlusSemileptonic)
68  p.addChannel(['anti-D0', 'e+'])
69  p.addChannel(['anti-D0', 'mu+'])
70  particles.append(p)
71  selection = create_path()
72  ...
73  analysis = create_path()
74  ...
75  main = FullEventInterpretation(selection, analysis,
76                                 particles)
77  process(main)
```

Bibliography

[1] A.J. Bevan et al. „The Physics of the B Factories“. In: *not published* (2014). arXiv: 1406.6311 [hep-ex].

[2] K. Abe et al. „Observation of large CP violation in the neutral B meson system“. In: *Phys.Rev.Lett.* 87 (2001), p. 091802. DOI: 10.1103/PhysRevLett.87.091802. arXiv: hep-ex/0107061 [hep-ex].

[3] K. Abe et al. „Observation of mixing induced CP violation in the neutral B meson system“. In: *Phys.Rev.* D66 (2002), p. 032007. DOI: 10.1103/PhysRevD.66.032007. arXiv: hep-ex/0202027 [hep-ex].

[4] Bernard Aubert et al. „Observation of CP violation in the B^0 meson system“. In: *Phys.Rev.Lett.* 87 (2001), p. 091801. DOI: 10.1103/PhysRevLett.87.091801. arXiv: hep-ex/0107013 [hep-ex].

[5] K. Abe et al. „Measurement of time dependent CP violating asymmetries in B0 → phi K0(s), K+ K- K0(s), and eta-prime K0(s) decays“. In: *Phys.Rev.Lett.* 91 (2003), p. 261602. DOI: 10.1103/PhysRevLett.91.261602. arXiv: hep-ex/0308035 [hep-ex].

[6] A. Poluektov et al. „Evidence for direct CP violation in the decay B → D(*)K, D → Ks Pi+ Pi- and measurement of the CKM phase phi3“. In: *Phys.Rev.* D81 (2010), p. 112002. DOI: 10.1103/PhysRevD.81.112002. arXiv: 1003.3360 [hep-ex].

[7] A. Abdesselam et al. „Measurement of the branching fraction of $B^+ \to \tau^+\nu_\tau$ decays with the semileptonic tagging method and the full Belle data sample“. In: *not published* (2014). arXiv: 1409.5269 [hep-ex].

[8] S.-K. Choi et al. „Observation of a Narrow Charmoniumlike State in Exclusive $B^\pm \to K^\pm\pi^+\pi^- J/\psi$ Decays“. In: *Phys. Rev. Lett.* 91 (26 Dec. 2003), p. 262001. DOI: 10.1103/PhysRevLett.91.262001.

[9] B.R. Ko et al. „Observation of $D^0 - \bar{D}^0$ Mixing in e^+e^- Collisions“. In: *Phys.Rev.Lett.* 112 (2014), p. 111801. DOI: 10.1103/PhysRevLett.112.111801. arXiv: 1401.3402 [hep-ex].

[10] A.G. Akeroyd et al. „Physics at super B factory“. In: *not published* (2004). arXiv: hep-ex/0406071 [hep-ex].

[11] C. Schwanda. „SuperKEKB machine and Belle II detector status“. In: *Nuclear Physics B - Proceedings Supplements* 209.1 (2010), pp. 70–72. DOI: http://dx.doi.org/10.1016/j.nuclphysbps.2010.12.012.

[12] M. Feindt et al. „A hierarchical NeuroBayes-based algorithm for full reconstruction of B mesons at B factories“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 654.1 (2011), pp. 432–440. DOI: http://dx.doi.org/10.1016/j.nima.2011.06.008.

[13] S. Kurokawa and E. Kikutani. „Overview of the {KEKB} accelerators“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 499.1 (2003), pp. 1–7. DOI: http://dx.doi.org/10.1016/S0168-9002(02)01771-0.

[14] S. Neubauer. „Search for $B \to K^{(*)}\nu\bar{\nu}$ Decays Using a New Probabilistic Full Reconstruction Method“. In: *not published* (2011). URL: https://ekp-invenio.physik.uni-karlsruhe.de/record/48215.

[15] J. Beringer et al. „Review of Particle Physics“. In: *Phys. Rev. D* 86 (2012), p. 010001.

[16] A. Abashian et al. „The Belle detector“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 479.1 (2002). Detectors for Asymmetric B-factories, pp. 117–232. DOI: http://dx.doi.org/10.1016/S0168-9002(01)02013-7.

[17] Edited by: Z. Doležal and S. Uno. „Belle II Technical Design Report“. In: *not published* (2010). arXiv: 1011.0352 [hep-ex].

[18] In: *online* (1.09.2014). URL: http://belle2.desy.de/sites2009/site_belle2/content/e103206/e103207/SuperKEKB:BelleII.jpg.

[19] Christian Pulvermacher. „dE/dx particle identification and pixel detector data reduction for the Belle II experiment“. 2012. URL: http://ekp-invenio.physik.uni-karlsruhe.de/record/48263.

[20] S. Lee, R. Itoh, N. Katayama, and S. Mineo. „Development of high level trigger software for Belle II at SuperKEKB“. In: *J.Phys.Conf.Ser.* 331 (2011), p. 022015. DOI: 10.1088/1742-6596/331/2/022015.

[21] In: *online* (1.09.2014). URL: http://belle2.kek.jp/images/Belle2.pdf.

[22] V. Blobel and E. Lohrmann. *Statistische und numerische Methoden der Datenanalyse*. Teubner-Studienbücher : Physik. 1998. URL: http://books.google.de/books?id=NcIxajwHOZIC.

[23] J. Neyman and E. S. Pearson. „On the Problem of the Most Efficient Tests of Statistical Hypotheses“. In: *Royal Society of London Philosophical Transactions Series A* 231 (1933), pp. 289–337. DOI: 10.1098/rsta.1933.0009.

[24] R. A. Fisher. „The use of multiple measurements in taxonomic problems“. In: *Annals of Eugenics* 7.2 (1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[25] Walter Pitts and WarrenS. McCulloch. „How we know universals the perception of auditory and visual forms“. In: *The bulletin of mathematical biophysics* 9.3 (1947), pp. 127–147. DOI: 10.1007/BF02478291.

[26] Marvin L. Minsky and Seymour Papert. *Perceptrons : an introduction to computational geometry*. Expanded ed., 3. print. 1988.

[27] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. „Learning representations by back-propagating errors“. In: *Nature* 323.6088 (1986), pp. 533–536.

[28] Jerome H. Friedman. „Stochastic gradient boosting“. In: *Computational Statistics & Data Analysis* 38.4 (2002), pp. 367–378. DOI: http://dx.doi.org/10.1016/S0167-9473(01)00065-2.

[29] Serguei Chatrchyan et al. „Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC“. In: *Phys.Lett.* B716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021.

[30] M. Feindt and U. Kerzel. „The NeuroBayes neural network package“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 559.1 (2006), pp. 190–194. DOI: http://dx.doi.org/10.1016/j.nima.2005.11.166.

[31] Anders Ryd et al. „EvtGen: A Monte Carlo Generator for B-Physics“. In: *not published* (2005).

[32] S. Agostinelli et al. „GEANT4: A Simulation toolkit“. In: *Nucl.Instrum.Meth.* A506 (2003), pp. 250–303. DOI: 10.1016/S0168-9002(03)01368-8.

[33] 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.

[34] G. Barrand et al. „GAUDI - A software architecture and framework for building HEP data processing applications“. In: *Comput.Phys.Commun.* 140 (2001), pp. 45–55. DOI: 10.1016/S0010-4655(01)00254-5.

[35] G. Barrand et al. „GAUDI - The software architecture and framework for building LHCb data processing applications“. In: *not published* (2000), pp. 92–95.

[36] P. Calafiura et al. „The StoreGate: A Data model for the Atlas software architecture“. In: *eConf* C0303241 (2003), MOJT008. arXiv: cs/0306089 [cs-se].

[37] W. Waltenberger. „RAVE – A Detector-Independent Toolkit to Reconstruct Vertices“. In: *Nuclear Science, IEEE Transactions on* 58.2 (Apr. 2011), pp. 434–444. DOI: 10.1109/TNS.2011.2119492.

[38] Andreas Hocker et al. „TMVA - Toolkit for Multivariate Data Analysis“. In: *not published* ACAT (2007), p. 040. arXiv: physics/0703039 [PHYSICS].

[39] Ryosuke Itoh et al. „Implementation of parallel processing in the basf2 framework for Belle II“. In: *J.Phys.Conf.Ser.* 396 (2012), p. 022026. DOI: 10.1088/1742-6596/396/2/022026.

[40] Jeffrey Dean and Sanjay Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (2004), pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1251254.1251264.