Verbesserung der Untergrundunterdrückung eines neuen CAST Detektors mittels multivariater Methoden aus TMVA

Sebastian Schmidt

Bachelorarbeit in Physik
angefertigt im Physikalischen Institut

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

September 2013

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Bonn, ...
Datum Unterschrift

1. Gutachter: Prof. Dr. Klaus Desch
2. Gutachter: Dr. Eckhard von Törne

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Das Standardmodell der Teilchenphysik ist fester Bestandteil der modernen Physik. Es beschreibt drei der vier fundamentalen Wechselwirkungen; die starke Kraft, die elektromagnetische Kraft und die schwache Kraft. Trotz des großen Erfolges des Standardmodells in der Physik hat dieses noch einige Probleme. In der starken Wechselwirkung erwartet man, ähnlich wie in der schwachen Wechselwirkung, bei Umkehr der Ladungen der Teilchen eines Systems bei gleichzeitiger Raumspiegelung einen nicht zum Ausgangssystem symmetrischen Zustand. Während diese Tatsache in der schwachen Wechselwirkung bereits 1964 gemessen wurde, steht der experimentelle Nachweis für die starke Wechselwirkung noch aus. Dies wird als das starke CP-Problem bezeichnet. Um dieses Problem zu lösen gibt es Erweiterungen des Standardmodells, welche theoretisch begründen, warum die sogenannte CP-Verletzung so klein ist. Dabei werden neue Teilchen, sogenannte Axionen, vorhergesagt.

Ein Experiment, welches diese Teilchen nachzuweisen versucht, ist das CERN Axion Solar Telescope. Hierbei wird indirekt versucht diese Axionen über Photonen zu detektieren. Für dieses Experiment ist ein neuer Detektor an der Universität Bonn in Entwicklung. Aufgrund einer sehr geringen Wahrscheinlichkeit Axionen zu detektieren, muss der Detektor in der Lage sein, so viele Untergrundereignisse, wie möglich und gleichzeitigt möglichst wenige der erwarteten Signale zu filtern. Man spricht dabei von einer hohen Untergrundunterdrückung bei hoher Signaleffizienz.

In dieser Arbeit soll die Untergrundunterdrückung dieses Detektors verbessert werden. Dazu werden multivariate Analyseverfahren verwendet, welche im *Toolkit for Multivariate Data Analysis* implementiert sind. Es werden *Artificial Neural Networks*, *Support Vector Machines* und *Boosted Decision Trees* mit unterschiedlichen Parametern und unterschiedlichen Datensätzen untersucht. Als Grundlage dient einerseits eine Messreihe mit einer radioaktiven Eisenquelle zur Simulation von Ereignissen, die einem Signal im Detektor entsprechen. Andererseits eine Untergrundmessung, um den im Experiment vorherrschenden Untergrund darzustellen.

Da die Ereignisse der Eisenquelle im weichen Röntgenbereich von 5,76 keV liegen, wird außerdem versucht auf Basis dieser Daten, niederenergetische Ereignisse künstlich zu erzeugen. Damit soll versucht werden die Software-Effizienz des Detektors im niederenergetischen Bereich von etwa $\sim 1\,\mathrm{keV}$ abzuschätzen.

KAPITEL 2

Axionen

Axionen sind hypothetische, pseudoskalare Teilchen, welche in theoretischen Modellen vorhergesagt werden, um ein Problem der starken Wechselwirkung zu lösen. In der **Q**uanten **C**hromo**d**ynamik (QCD) erwartet man, dass ein System unter Ladungs- und Paritätsumkehr nicht symmetrisch zum Ausgangssystem ist. Nach bisherigen Messungen ist dies allerdings nicht der Fall. Dies wird als das Problem der starken CP-Verletzung bezeichnet.

Es zeigt sich unter anderem anhand eines hypothetischen, elektrischen Dipolmoments des Neutrons. Da das Neutron aus einem up- und zwei down-Quarks zusammengesetzt ist, erwartet man aufgrund ihrer unterschiedlichen Ladungen ein solches Dipolmoment. Sofern das Neutron ein elektrisches Dipolmoment hat, ist die Zeit- (T) und Paritätstransformation (P), wie in Abbildung 2.1 gezeigt, nicht symmetrisch zum Ausgangszustand. Da alle modernen Theorien einer CPT-Symmetrie unterliegen, folgt aus der Verletzung von P und T sofort auch die CP-Verletzung. Insofern ist also die Messung eines elektrischen Dipolmoments des Neutrons äquivalent zur Existenz der starken CP-Verletzung. Nach Messungen beträgt der Wert des elektrischen Dipolmoments des Neutrons $|d_n| < 2{,}9 \cdot 10^{-26}\, e\,\mathrm{cm}$. [1]

In der QCD wird die starke CP-Verletzung durch die folgende Lagrange Dichte dargestellt:

$$\mathcal{L}_{\bar{\theta}} = \bar{\theta}\frac{g^2}{32\pi^2} G^a_{\mu\nu}\tilde{G}^{\mu\nu}_a.$$

Hierbei ist g die Kopplungskonstante, $G^a_{\mu\nu}$ der Feldstärketensor des Gluon-Feldes und $\tilde{G}^{\mu\nu}_a$ der dazu duale Tensor. Der Winkel $\bar{\theta}$ ist

$$\bar{\theta} = \theta + \theta_{\mathrm{weak}},$$

wobei θ_{weak} durch die elektroschwache Wechselwirkung auftritt. θ ist ein freier Parameter mit $\theta \in \{0, 2\pi\}$. Aus dem oben genannten Messwert zum elektrischen Dipolmoment des Neutrons, folgt für $\bar{\theta}$ ein Wert von kleiner als 10^{-10}. Das Problem ist nun, warum dieser Wert gerade so klein ist, dass die starke CP-Verletzung bisher nicht gemessen werden konnte.

Das Problem kann gelöst werden, entweder indem der θ Term aus dem Lagrangian entfernt wird, wofür allerdings Physik jenseits des Standardmodells benötigt wird. Oder indem θ auf 0 gesetzt wird, was jedoch zu Problemen für die schwache CP-Verletzung führt.

Die elegante Lösung des Problems bietet der Peccei-Quinn Mechanismus, welcher im Jahr 1977 vorgestellt wurde [3]. Dazu wird eine globale Symmetrie, $U(1)_{\mathrm{PQ}}$, eingeführt, welche spontan unterhalb einer bestimmten Energieskala f_a gebrochen wird. Bei diesem Bruch der Symmetrie erscheint ein sogenanntes Goldstone Boson, welches Axion genannt wird. Das Axion ist nach einem amerikanischen

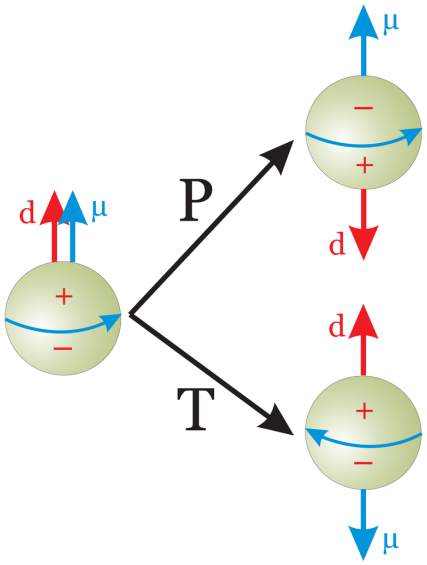


Abbildung 2.1: Darstellung von Zeit- (T) und Paritätstransformation (P) des Neutrons. μ entspricht dem magnetischen Dipolmoment und d dem hypothetischen, elektrischen Dipolmoment. [2]

Waschmittel benannt, weil es das Problem der starken CP-Verletzung aus dem Standardmodell wäscht. Es koppelt an Gluonen:

$$\mathcal{L}_{aG} = -\frac{\alpha_S}{8\pi f_a} a G_b^{\mu\nu} \tilde{G}^b_{\mu\nu}.$$

Hierbei ist α_S die Feinstrukturkonstante. Definiert man für diese Bosonen ein Feld a, kann man eine Kopplung an zwei Photonen durch

$$\mathcal{L}_{a\gamma} = -\frac{1}{4} g_{a\gamma} F_{\mu\nu} \tilde{F}^{\mu\nu} a = g_{a\gamma} \mathbf{E} \cdot \mathbf{B} a$$

beschreiben [4]. $g_{a\gamma}$ ist die Kopplungskonstante von Axionen an Photonen. Da eines der beiden Photonen auch virtuell sein kann, folgt, dass Axionen sich unter Einfluss von elektrischen oder magnetischen Feldern, über ihre Kopplung an Gluonen, in Photonen umwandeln können. Dies geschieht über den inversen Primakoff Effekt (siehe Abbildung 2.2).

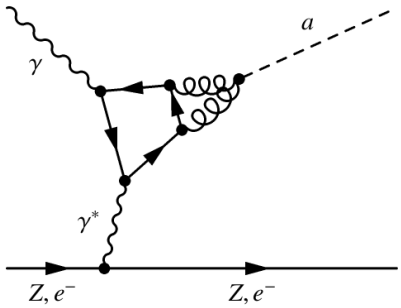


Abbildung 2.2: Feynman Diagramm des inversen Primakoff Effekts zum Übergang zwischen Photon und Axion.

Da der Primakoff Effekt auch invers durch ein Photon, welches mit einem Feld eines Atomkerns wechselwirkt, auftreten kann, geht man davon aus, dass im Zentrum der Sonne eine große Zahl Axio-

nen entsteht. Aufgrund der Temperatur in der Sonne haben die Photonen Energien im keV Bereich. Koppeln zwei Photonen zu einem Axion, hat dieses folglich eine kinetische Energie im keV Bereich. Das erwartete Spektrum ist in Abbildung 2.3 zu finden.

Beim inversen Primakoff-Effekt wechselwirkt ein Axion mit einem virtuellen Photon eines elektrischen oder magnetischen Feldes und es entsteht aufgrund der kinetischen Energie ein Photon im weichen Röntgenbereich.

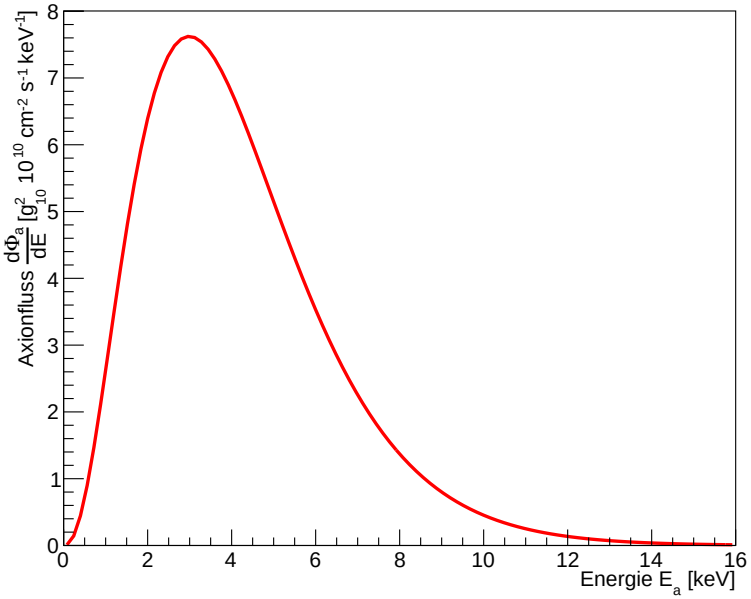


Abbildung 2.3: Fluss in Abhängigkeit der Energie der solaren Axionen, die auf der Erde erwartet werden.

KAPITEL 3

CERN Axion Solar Telescope

Das **C**ERN **A**xion **S**olar **T**elescope (CAST) ist ein Experiment am CERN, welches seit 2003 Daten aufnimmt. Ziel des Experiments ist die Messung des Wirkungsquerschnitts des Übergangs von Photonen zu Axionen, bzw. im Optimalfall der indirekte Nachweis von solaren Axionen mittels Photonen.

Das Experiment besteht aus einem Prototypen-Magneten des **L**arge **H**adron **C**olliders (LHC), welcher nicht gebogen ist. Aufgrund eines relativ kleinen Bewegungsbereichs von horizontal ±40° und vertikal ±8°, kann die Sonne täglich nur etwa 2·1,5 Stunden, während des Sonnenauf- und -untergangs, beobachtet werden [5]. Die restliche Messzeit werden Untergrundmessungen vorgenommen.

Das Konzept ist, dass innerhalb des 9,26 m langen Magneten durch das Magnetfeld von $B = 9\,\mathrm{T}$ Axionen über den inversen Primakoff-Effekt in Photonen umgewandelt werden. Da die solaren Axionen nach der Theorie Energien im Bereich mehrerer keV haben, erwartet man Photonen im weichen Röntgenbereich. Daher sind auf beiden Seiten des Magneten Röntgendetektoren befestigt. Auf der Seite, welche zum Sonnenaufgang blickt, ist ein Röntgenteleskop, baugleich mit dem des XMM-Newton Weltraumobservatoriums befestigt. Der dort verbaute **C**harge-**C**oupled **D**evice Sensor (CCD) wird durch den in [6] entwickelten Detektor ersetzt.

Abbildung 3.1 zeigt das Experiment.

Abbildung 3.1: Foto des CAST-Experiments. Entnommen aus [5].

3.1 Detektoren

Im Folgenden wird kurz auf die bisher verwendeten Detektoren des CAST Experiments eingegangen und mit dem Detektor verglichen, für den die Untergrundunterdrückung in dieser Arbeit vorgenommen wird. Unter anderem wurde zuvor eine **T**ime **P**rojection **C**hamber (TPC) verwendet. Auf diese wird in dieser Arbeit nicht weiter eingegangen.

3.1.1 Micromegas

Bislang wurden am CAST **MicroMe**sh **Ga**seous **S**tructures (Micromegas), eine Art von **M**icro**p**attern **G**aseous **D**etectors (MPGDs), verwendet. Diese wurden im Jahr 1996 von Giomataris und Charpak [7, 8] entwickelt. Wie in Abbildung 3.2 dargestellt kann der Aufbau wesentlich in zwei Bereiche aufgeteilt werden. Nachdem das Photon ein Fenster durchlaufen hat, tritt es in den Konversionsbereich (engl. *conversion gap*) ein. Hier liegt über eine Hochspannung ein elektrisches Feld von $E_D \sim 1\,\mathrm{kV\,cm^{-1}}$ an. Der Bereich ist mit 97,7 % Argon (Ar) und 2,3 % Isobutan ($\mathrm{iC_4H_{10}}$) gefüllt. Hier sollen die Photonen über den Photoeffekt Elektronen herauslösen, welche durch das elektrische Feld in den zweiten Bereich des Detektors driften. Dazu durchqueren sie das *micromesh*, ein metallisches Gitter mit ~20 µm durchmessenden Löchern und einer Periodizität von ~80 µm [9]. In diesem Gasverstärkungsbereich (engl. *amplification gap*) liegt ein elektrisches Feld von etwa $E_A \sim 50\,\mathrm{kV\,cm^{-1}}$ an. Hier werden über Gasverstärkung, ähnlich wie beim Geiger-Müller Zählrohr, durch den Lawineneffekt weitere Elektronen herausgelöst, welche in Richtung Detektor driften und dort detektiert werden können. Die Detektion wird hierbei in der Regel über Streifen oder Pads vorgenommen.

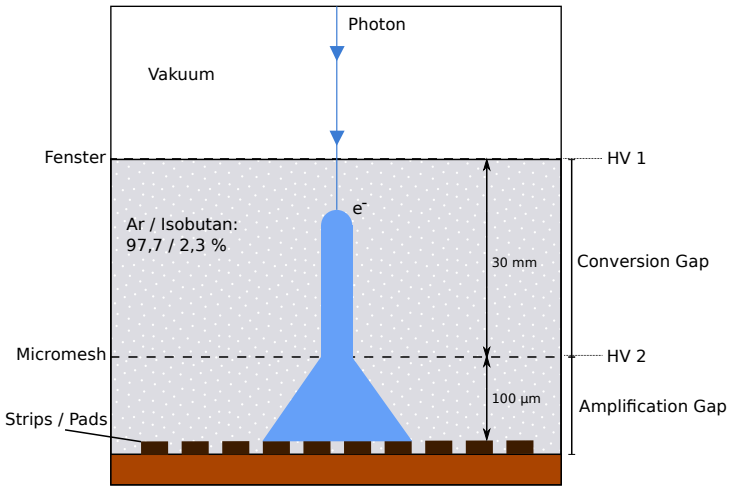


Abbildung 3.2: Skizze zum Aufbau eines Micromegas. Konversionsbereich (*conversion gap*) in dem Photonen Elektronen herauslösen und Verstärkungsbereich (*amplification gap*) mit stärkerem elektrischen Feld zum vervielfachen der Elektronen. Die Auslese wird über Streifen oder Pads vorgenommen.

3.1.2 GridPix

Zur Verbesserung der Ortsauflösung können die Streifen oder Pads durch einen pixelbasierten Chip ersetzt werden. Setzt man ein Micromegas jedoch einfach auf einen Pixelchip, führt dies zu Interferenzmustern, den sogenannten Moiré-Strukturen. Diese treten aufgrund leichter Verschiebungen und

unterschiedlicher Periodizität des Micromesh zu den kleineren Pixeln des Chips auf. Um dies zu umgehen wird die Struktur zur Gasverstärkung direkt mittels photolithographischer Verfahren auf dem Chip integriert. Diese Struktur zur Gasverstärkung wird als **In**tegrated **Grid** (InGrid) [10, 11] bezeichnet. Der gesamte Aufbau aus Pixelchip und InGrid wird dann **GridPix** genannt.

Im Vergleich zu den Streifen oder Pads, welche eine Größe von ~400 µm bis zu mm haben, liegt die Größe eines Pixels in der Größenordnung von ~50 µm. Mit einer geeigneten Gasdiffusion kann bei diesem Auflösungsvermögen in erster Näherung davon ausgegangen werden, dass alle durch Gasverstärkung erzeugten Elektronen auf unterschiedliche Pixel treffen. Dies erlaubt eine Aussage über die Energie direkt anhand der aktivierten Pixel, da die Anzahl der Primärelektronen, und damit die Anzahl der durch Gasverstärkung erzeugten Elektronen, ein Maß für die Energie des Photons ist.

Der Detektor der die Daten dieser Arbeit aufgezeichnet hat, basiert auf einem Timepix ASIC [6]. Ein ASIC ist ein **A**pplication **S**pecific **I**ntegrated **C**ircuit.

3.2 Daten und Spektrum der ^{55}Fe Quelle

Am CAST wird, aufgrund der geringen Wechselwirkung von Axionen mit Photonen, erwartet, dass der Großteil der erzeugten Daten Untergrundereignisse sind. Die meisten Untergrundereignisse kommen aus kosmischer Strahlung und sind beispielsweise hochenergetische Myonen. Diese sind in den meisten Fällen eindeutig in Form einer Spur zu erkennen, wie in Abbildung 3.3 (a) anhand eines Beispiels dargestellt. Es besteht weiterhin die Möglichkeit, dass kosmische Strahlung Atome des Detektormaterials ionisiert, wodurch Röntgenlinien angeregt werden können, welche nicht von Photonen, die durch Axionen entstehen, unterschieden werden können. Generell ist der Untergrund eine kontinuierliche Verteilung über einen großen Energiebereich. Die Photonen, welche erwartet werden, sehen typischerweise aus wie in 3.3 (b) dargestellt.

Durch die hohe Untergrundrate muss eine gute Methode zur Trennung zwischen Signal und Untergrund, mit einem Fokus auf eine gute Untergrundunterdrückung entwickelt werden. Bisher wurde dazu in [6] eine Likelihood-Ratio Methode verwendet. Hier soll diese Unterscheidung mittels künstlichen neuronalen Netzwerken (**A**rtificial **N**eural **N**etworks, ANNs), **S**upport **V**ector **M**achnies (SVMs) sowie **B**oosted **D**ecision **T**rees (BDTs) verbessert werden.

Um mittels multivariater Methoden eine Untergrundunterdrückung durchführen zu können, muss zunächst mit dem Detektor jeweils ein Datensatz mit signalartigen Ereignissen und einer mit untergrundartigen Ereignissen aufgenommen werden. Die Daten für die Untergrundmessung stammen aus einer einwöchigen Messung, bei der etwa 38 000 Ereignisse gemessen wurden, in der der Detektor mit Blei umhüllt wurde. Um signalartige Ereignisse zu erhalten, wurde eine Messung mit einer ^{55}Fe Quelle durchgeführt und etwa 59 000 Ereignisse aufgenommen. Diese Datensätze wurden in [6] erstellt.

Das Isotop ^{55}Fe zerfällt über Elektroneneinfang zu ^{55}Mn, welches sich in einem angeregten Zustand befindet und über das Aussenden eines Photons in den Grundzustand übergeht.

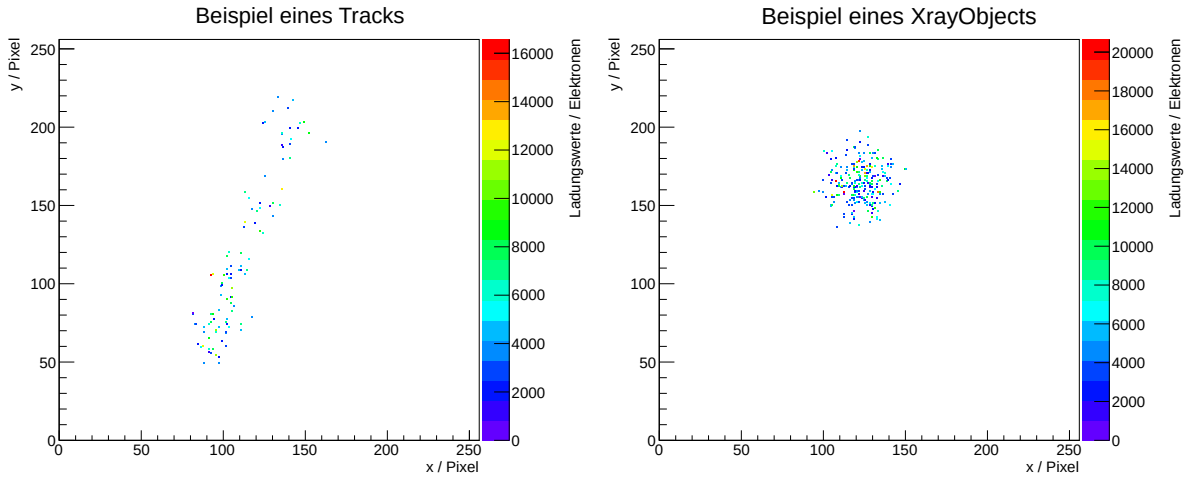
$$^{55}\text{Fe} \longrightarrow {}^{55}\text{Mn}^{*} \longrightarrow {}^{55}\text{Mn} + \gamma$$

Ein solcher Übergang tritt in etwa 28 % der Fällen auf. Davon fällt ~85 % auf den K_α Zerfall mit einer Energie von 5,895 keV und der Rest auf die K_β Linie mit 6,49 keV. Da man nach Möglichkeit zur Energiekalibrierung und Eindeutigkeit der Daten nur eine Linie in den Daten haben möchte, filtert man die K_β Linie mit einer Chromfolie von 10 µm Dicke heraus. Somit bleibt hauptsächlich der sogenannte Photopeak bei 5,9 keV.

Argon besitzt Elektronen in der K-, L- und M-Schale. Beim Photoeffekt mit Photonen von etwa 5,9 keV tragen die Schalen L und M nur etwa ~ 10 % der Fälle bei, in den meisten Fällen also wird

ein Elektron aus der K-Schale herausgelöst. Ein Elektron der K-Schale mit $E_K = -3{,}2\,\text{keV}$, welches von einem 5,9 keV Photon herausgelöst wird, hat zunächst 2,7 keV. Die K-Schale wird zu 13,5 % von einem Elektron aus der L-Schale erneut besetzt. Dabei wird ein Röntgenphoton mit einer Energie von etwa 2,9 keV frei, welches in den meisten Fällen den Detektor verlässt. Bei der weiteren Umverteilung der Elektronen, wird meist ein Auger Elektron mit 0,2 keV frei, welches im Detektor umgesetzt wird. Daher beläuft sich die im Schnitt beim sogenannten Escapepeak deponierte Energie auf etwa 2,9 keV.

In 86,5 % der Fälle, kommt es zu einer Auger Kaskade, bei der zunächst aus der L-Schale und anschließend aus der M-Schale Elektronen die unteren Schalen neu füllen. Hierbei werden zu 75 % der Fälle Elektronen mit 3,06 keV frei, welche detektiert werden. Dies führt für den Photoeffekt auf eine Gesamtenergie von 5,76 keV. [10]



(a) (b)

Abbildung 3.3: (a) Beispiel eines typischen Untergrundereignisses. (b) Beispiel eines typischen Röntgenereignisses der ^{55}Fe Quelle bei einer Energie von ~5,76 keV. In Farbe ist die Ladung der Pixel dargestellt.

KAPITEL 4

TMVA

Das **T**oolkit for **M**ulti**v**ariate Data **A**nalysis (TMVA) ist eine Sammlung von Algorithmen zur Untersuchung multivariater Verteilungen. Es ist ein fester Bestandteil von ROOT, einem Datenanalyseframework für Hochenergiephysik, entwickelt am CERN. Unter multivariater Analyse versteht man zunächst das Untersuchen einer Zufallsverteilung, die von mehreren Variablen abhängt.

In der Physik wird sie z.B. in der Datenanalyse von Detektoren verwendet. Und zwar zur Unterscheidung zwischen signal- und untergrundartigen Ereignissen, da hierbei in der Regel jedes Ereignis mehrere Eigenschaften (Energie, Länge, etc.) besitzt. Diese unterscheiden sich im Idealfall für Signal und Untergrund, wodurch sie mittels multivariater Analyse getrennt werden können.

TMVA bringt hierfür nun eine Reihe verschiedener Algorithmen mit. Angefangen von simpleren Methoden, wie einfachen Schnitten auf die Daten oder einer Likelihood Methode bis zu komplizierten Modellen, z.B. **B**oosted **D**ecision **T**rees (BDT) und künstlichen neuronalen Netzwerken (**A**rtificial **N**eural **N**etwork, ANN). Auf die in dieser Arbeit verwendeten Methoden wird im Abschnitt 4.2 im Detail eingegangen. Alle Methoden erhalten letztlich ein Ereignis mit verschiedenen Eigenschaften, denen sie in der Regel einen Zahlenwert zwischen 0 bis 1 oder -1 bis 1 zuordnen. Darüber wird die Klassifikation durchgeführt. Eine Methode ist dann als erfolgreich anzusehen, wenn sie für signalartige und untergrundartige Ereignisse signifikant unterschiedliche Ausgabewerte errechnet. Anhand des Ausgabewertes kann dann anschließend ein Schnittwert definiert werden, der bei der Klassifizierung mit unbekannten Daten zur Unterscheidung verwendet wird.

Abgesehen von der Klassifikation selbst bietet TMVA viele Optionen, um die Qualität der Methoden beurteilen zu können. Dazu werden die Größen **Signal**- ϵ_S und **Untergrundeffizienz** ϵ_B, **Signalreinheit** und **Signifikanz** verwendet. Schneidet man bei einem bestimmten Wert auf die Klassifikation, beschreibt die Signal- und Untergrundeffizienz, welchen prozentualen Anteil aller Signal- und Untergrundereignisse im Bereich oberhalb des Schnittwerts vorhanden sind. Somit wünscht man eine maximal hohe Signaleffizienz ϵ_S bei minimaler Untergrundeffizienz ϵ_B. Häufig wird statt Untergrundeffizienz der Begriff Untergrundunterdrückung verwendet, welche über

$$\epsilon_{\text{rej}} = 1 - \epsilon_B$$

definiert wird. Unter dem Begriff **Signalreinheit** (engl. *signal purity*) versteht man beim Setzen eines Schnittwerts den prozentualen Anteil der Signalereignisse von allen Ereignissen oberhalb des Schnittwerts.

$$\text{Purity} = \frac{S}{S + B}$$

Dabei sind S und B die Anzahl der Ereignisse oberhalb des Schnittwerts. Die **Signifikanz** ist über

$$\text{Sig} = \frac{S}{\sqrt{S+B}}$$

definiert. Die Signifikanz ist ein Maß für die Untergrundunterdrückung.

Im weiteren Kapitel wird zunächst der Aufbau von TMVA als Programm beschrieben. Dazu wird zuerst darauf eingegangen, in welchem Form die Daten an TMVA übergeben werden. Anschließend wird auf die essentiellen Objekte **Factory** und **Reader** eingegangen. Im weiteren Abschnitt werden die in dieser Arbeit verwendeten Methoden mit ihren Funktionsweisen in Verbindung mit den wählbaren Optionen erläutert.

4.1 Aufbau

In diesem Abschnitt wird der Aufbau von TMVA beschrieben. Hierbei wird nur auf die, für diese Arbeit, relevanten Aspekte eingegangen. Im Handbuch von TMVA ist dies ausführlicher beschrieben [12].

Grundsätzlich muss bei der Verwendung von TMVA zwischen zwei verschiedenen Phasen unterschieden werden. Zunächst die Phase des Trainieren, Testen und Evaluieren der einzelnen Methoden mittels der **Factory**-Klasse basierend auf bekannten Signal- und Untergrund-Daten. Anschließend die Phase der eigentlichen Klassifizierung von einem unbekannten Datensatz über die **Reader**-Klasse. Die beiden Klassen werden in den Abschnitten 4.1.2 und 4.1.3 erläutert.

Der Verlauf der relevanten Schritte ist in Abbildung 4.1 skizzenhaft dargestellt.

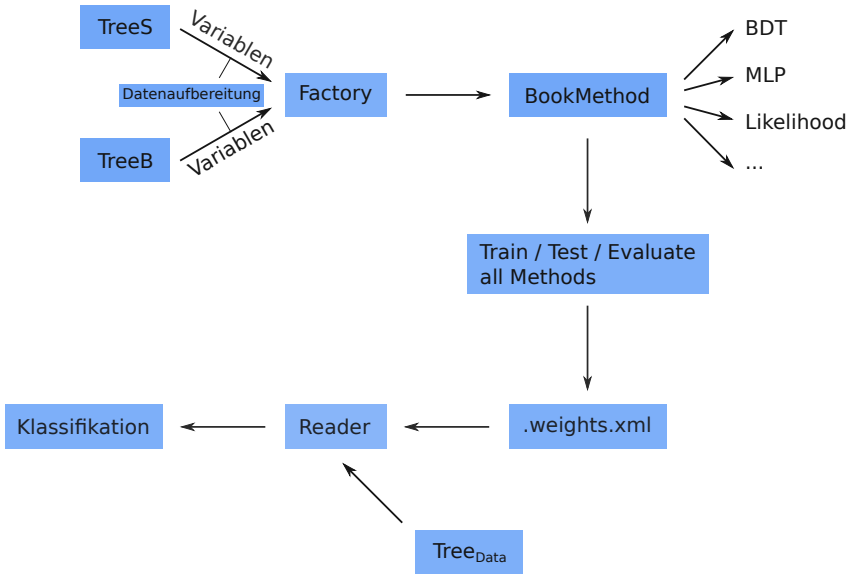


Abbildung 4.1: Skizze des internen Aufbaus von TMVA. Zunächst das Trainieren über das **Factory**-Objekt und anschließende Klassifikation unbekannter Daten mit dem **Reader**-Objekt.

4.1.1 Eingabedaten für TMVA

In beiden Fällen dient als Datengrundlage ein **TTree** Objekt aus ROOT. Dabei handelt es sich um ein Objekt, welches selber Objekte des Typs **TBranch** besitzt. Jedes Objekt des Typs **TBranch** beinhaltet

wiederum Variablen eines bestimmten Typs, gespeichert in sogenannten **TLeaf** Objekte. Für den hier verwendeten Zweck besitzt ein **TTree** für jede vorhandene Variable eines Ereignisses einen **TBranch**. Jedes Ereignis hat in jedem **TBranch** ein **TLeaf**, in dem der Wert der jeweiligen Variable gespeichert ist.

Aus den in Abschnitt 3.2 beschriebenen Daten wird nun für jedes Ereignis versucht ein Röntgenphoton (**XrayObject**) und eine Spur (**Track**) zu finden. In einem Ereignis werden in der Regel nicht alle Pixel zu einem **XrayObject** und **Track** gehören. Es wird der sogenannte MarlinTPC Framework verwendet, welcher auf dem **M**odular **A**nalysis and **R**econstruction for the **L**inear **C**ollider (Marlin) Framework basiert. Marlin wurde erweitert, um die Rekonstruktion für TPCs zu ermöglichen. Verschiedene Aufgaben innerhalb der Rekonstruktion und Analyse der Daten werden von sogenannten *Prozessoren*, welche in C++ programmiert sind, durchgeführt. Grundsätzlich wird nach gewissen Eigenschaften (Dichte der Pixel etc.) entschieden, ob ein **XrayObject** und ein **Track** gefunden werden kann.

Für alle Ereignisse, in denen beide Objekte (**XrayObject** und **Track**) gefunden werden, werden mittels weiterer Prozessoren die relevanten Eigenschaften, wie z.B. Radius, Länge, statistische Momente etc. berechnet. Dies ergibt für beide Datensätze unterschiedliche Verteilungen in den einzelnen Variablen. Diese Daten werden in den oben beschriebenen **TTree** Objekten gespeichert und an TMVA übergeben. Alle in dieser Arbeit verwendeten Variablen sind ausführlich in Kapitel 5 beschrieben.

4.1.2 Factory

Das Objekt **Factory** ist eines der zentralen Objekte von TMVA. Es ist für das Trainieren, Testen und Evaluieren der verschiedenen Methoden verantwortlich. Beim Erstellen des Objekts wird zunächst ausgewählt, welche Transformationen durchgeführt werden sollen. Es stehen die Identität, das Normieren und die Dekorrelation zur Auswahl. Beim Normieren werden die Werte der Variablen auf -1 bis 1 normiert. Dies ist bei einigen Methoden, aufgrund der Berechnung der Gewichtungen verschiedener Variablen notwendig. Bei der Dekorrelation wird versucht Abhängigkeiten zwischen verschiedenen Variablen zu minimieren. Dem **Factory**-Objekt werden Variablen der Ereignisse hinzugefügt. Diese können alternativ als sogenannter Zuschauer hinzugefügt werden. Dies hat zur Folge, dass sie nicht für das Trainieren oder Testen verwendet werden. Sie können allerdings zur Unterscheidung und Darstellung der Ereignisse durch den Benutzer oder zum Schneiden auf die Daten verwendet werden. Anschließend wird jeweils für die Signal- und Untergrund-Daten ein Tree-Objekt aus ROOT übergeben, in welchem die Variablen, welche zuvor hinzugefügt wurden, enthalten sein müssen.

Nun wird über das **Factory**-Objekt das Trainieren und Testen vorbereitet, indem die Daten der Trees in einen gemischten Test- und Trainings-Datensatz aufgeteilt werden. Hierbei kann entschieden werden, wieviel Prozent der Signal- und Untergrunddaten jeweils zum Trainieren und Testen verwendet werden sollen. Standardmäßig wird die eine Hälfte der Daten zum Trainieren und die andere zum Testen genutzt, wobei sie zufällig gewählt werden. Für große Datenmengen oder komplexe Methoden kann es sinnvoll sein zunächst nur eine Teilmenge der Daten zum Trainieren zu nutzen.

Anschließend werden die verschiedenen Methoden gebucht, wobei eine Vielzahl von Optionen gewählt werden kann. Darauf wird genauer im methodenspezifischen Abschnitt 4.2 eingegangen.

Die Ergebnisse werden von TMVA in eine `.root` Datei gespeichert. Für die trainierten Methoden wird eine `.weights.xml` Datei erstellt, in welcher die fertig trainierten Methoden gespeichert sind. Diese können mit dem **Reader**-Objekt zur Klassifikation verwendet werden. Alternativ steht auch für die meisten Methoden eine völlig unabhängige C-Datei zur Verfügung.

Die `.root` Datei kann mittels einiger von TMVA bereitgestellten Macros analysiert werden. Dazu gibt es eine graphische Oberfläche, welche in Abbildung 4.2 dargestellt ist. Es können die Eingabe-Variablen, wie in 4.3 gezeigt, angezeigt werden. Für die getesteten Ereignisse kann unterschieden zwi-

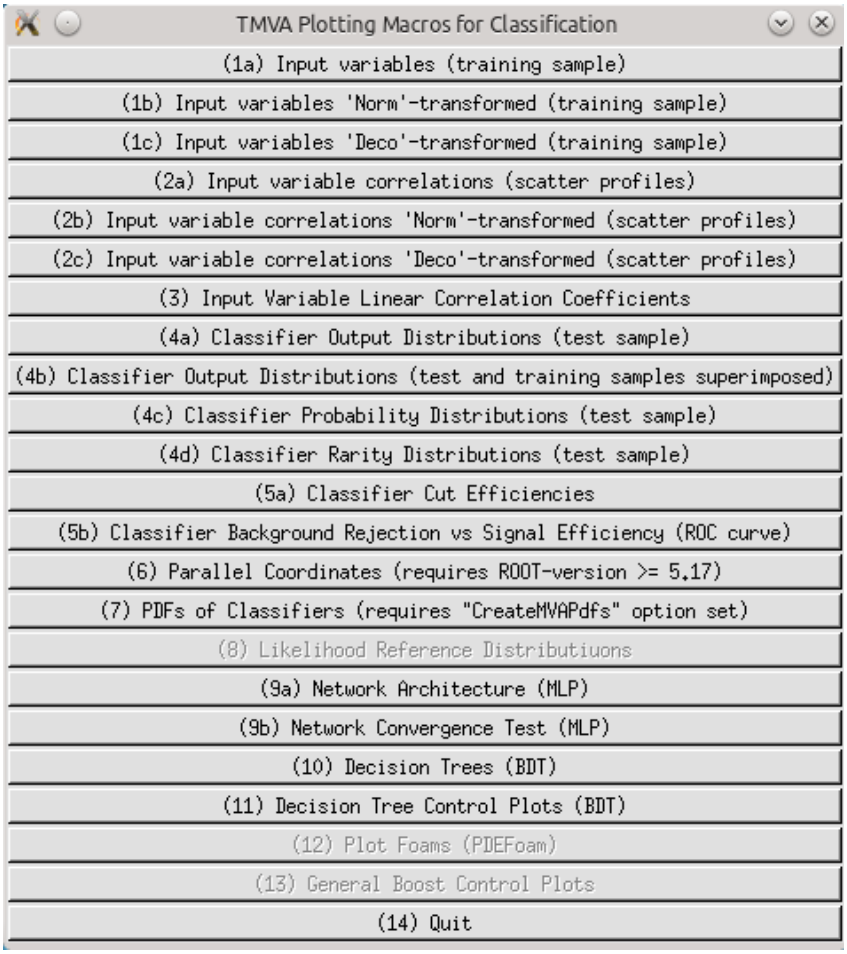


Abbildung 4.2: Graphische Oberfläche von TMVA zum Ausführen verschiedener Macros für das Evaluieren der Methoden.

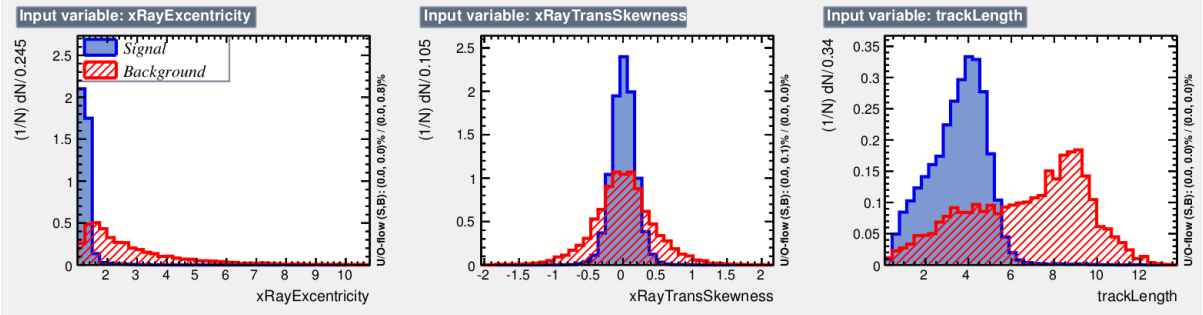


Abbildung 4.3: Beispiel der Verteilungen dreier Eingabe-Variablen. In Blau sind die Werte der Signalereignisse und in Rot die der Untergrundereignisse dargestellt.

schen Signal und Untergrund die Klassifizierung gezeichnet (siehe Abbildung 4.4), die Untergrundunterdrückung betrachtet (siehe Abbildung 4.5) und weitere Methodenspezifische Funktionen gewählt werden (siehe Abschnitt 4.2). In Abbildung 4.6 sind die Signal- und Untergrundeffizienzen, sowie die Signalreinheit in Abhängigkeit eines definierten Schnittwertes auf die Ausgabe einer Methode dargestellt.

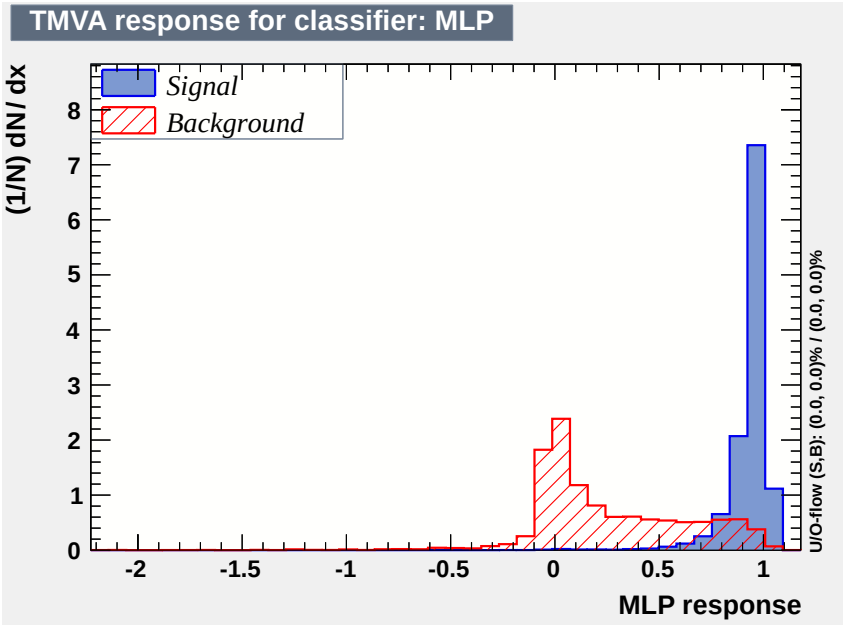


Abbildung 4.4: Beispiel einer Klassifizierung mittels eines neuronalen Netzwerks. In Blau sind die Werte der Signalereignisse aus dem Testdatensatz und in Rot die der Untergrundereignisse gezeichnet.

4.1.3 Reader

Der **Reader** ist das Objekt in TMVA, welches für die Klassifizierung von unbekannten Daten, auf Grund von den durch die **Factory** erzeugten Methoden, verantwortlich ist. Als Dateneingabe wird eine beliebige `.root`-Datei verwendet, welche einen Tree mit Ereignis-Daten enthält. Neben der `.root`-Datei, wird für jede zu verwendende Methode eine `.weights.xml`-Datei, welche zuvor vom **Factory**-Objekt erzeugt wurde, übergeben.

Beim **Reader**-Objekt ist zu beachten, dass alle Variablen, die verwendet werden, bereits im **Factory**-Objekt zum Trainieren und Testen verwendet wurden. Der Tree mit den zu klassifizierenden Daten muss zusätzlich alle diese Variablen enthalten.

Zur Klassifikation wird über alle Einträge des Trees iteriert, wobei für jedes Ereignis einzeln die Klassifikation der Methoden abgefragt wird.

Hierbei kann nun für jede Methode ein Histogramm erstellt werden, in das für jedes Ereignis die Klassifizierung eingetragen wird. Damit erhält man ein Histogramm, wie in Abbildung 4.7 dargestellt.

4.2 Methoden

Im folgenden werden einige Methoden erläutert, die in dieser Arbeit verwendet werden. Viele Erklärungen sind an das Handbuch von TMVA angelehnt, welches alle verfügbaren Methoden ausführlich

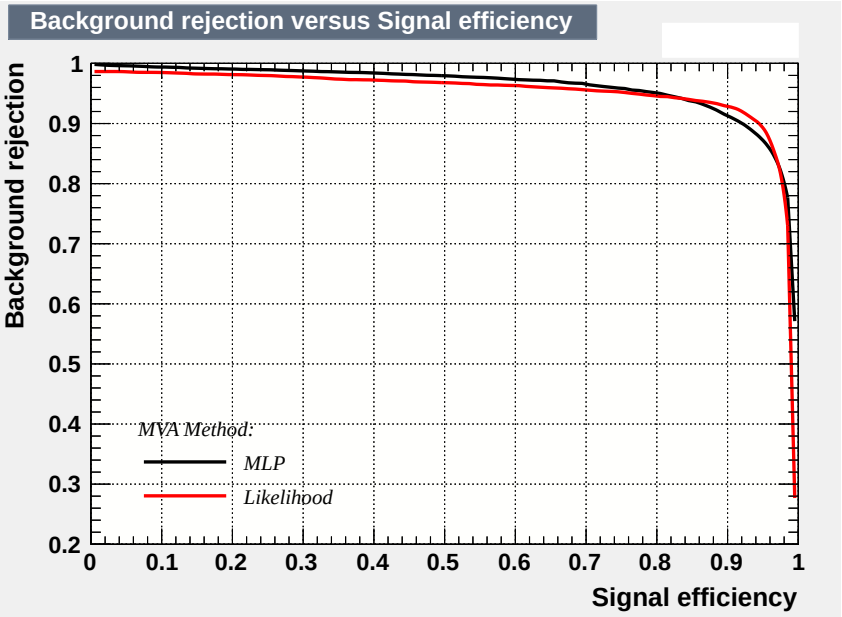


Abbildung 4.5: Beispiel einer Untergrundunterdrückung für eine Likelihood Methode im Vergleich zu einem neuronalen Netzwerk. Die Kurve zeigt, wie hoch die prozentuale Unterdrückung der Untergrundereignisse für eine gegebene Signaleffizienz ist.

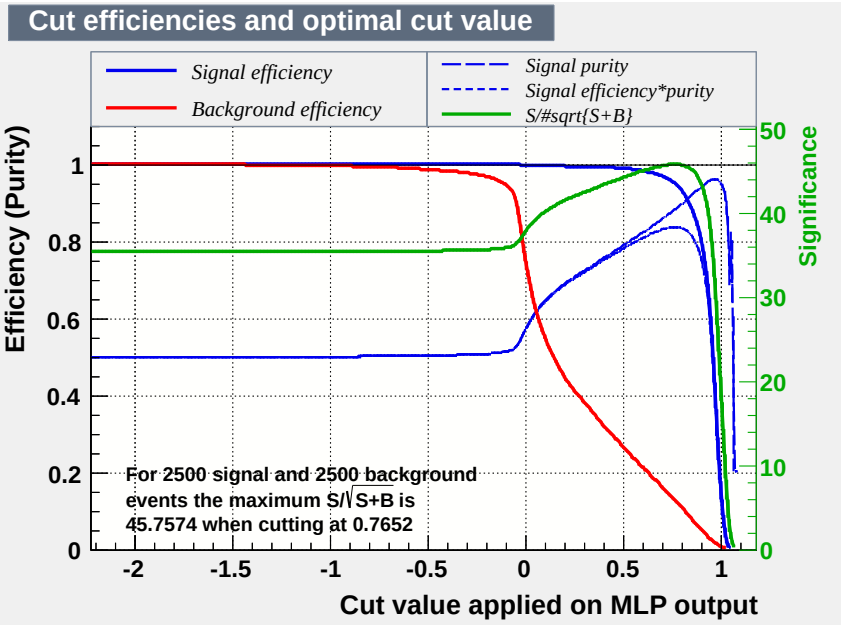


Abbildung 4.6: Effizienzen einer Methode in Abhängigkeit von möglichen Schnittwerten auf die Ausgabe. Blaue Linie: Signal-Effizienz in Abhängigkeit des Schnittwerts, d.h. prozentualer Anteil aller Signalereignisse, die oberhalb dieses Schnittwerts liegen. Rote Linie: Untergrund-Effizienz in Abhängigkeit des Schnittwerts, analog zu Signal-Effizienz. Grob gestrichelte blaue Linie: Signal Reinheit in Abhängigkeit des Schnittwerts; prozentualer Anteil an Signalereignissen an allen Ereignissen oberhalb des Schnittwerts. Fein gestrichelte blaue Linie: Signal Effizienz × Reinheit. Grüne Linie: Signifikanz, Maß für die Untergrundunterdrückung der Methode.

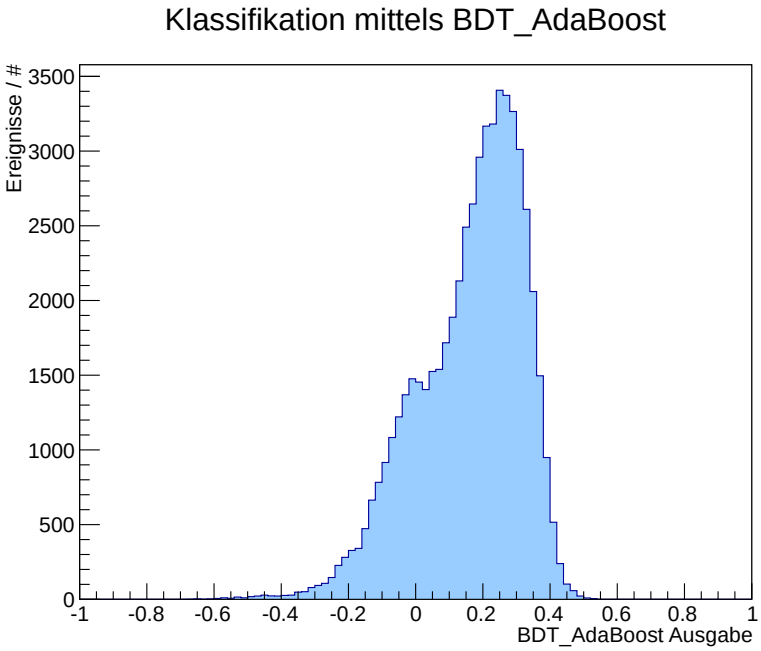


Abbildung 4.7: Beispiel eines Histogramms zur Klassifizierung eines unbekannten Datensatzes mittels eines zuvor trainierten BDTs.

erklärt [12].

Zunächst wird jedoch noch auf ein generelles Problem des Trainierens, und zwar das *Overtraining*, eingegangen.

4.2.1 Overtraining

Je komplexer multivariate Methoden werden, umso anfälliger werden sie für sogenanntes *Overtraining*. Hiermit bezeichnet man die Tatsache, dass ein Modell, wenn es zu komplex wird, nicht nur echte Eigenschaften der Daten erlernt, sondern auch beginnt statistische Schwankungen als Eigenschaft zu verstehen. Dies führt dazu, dass sich die Erkennung für einen unabhängigen Datensatz unter Umständen stark verschlechtert. Verschiedene Methoden sind dafür unterschiedlich stark anfällig. Im allgemeinen tritt es dann auf, wenn für zu viele freie Parameter der Methode, zu wenige Ereignisse vorhanden sind und diese Ereignisse zu oft an die Methode übergeben werden. Bei der Beschreibung der einzelnen Methoden wird jeweils auf *Overtraining* eingegangen. Die meisten Methoden bieten Optionen, um die Gefahr des *Overtraining* zu minimieren.

Um das Maß des *Overtrainings* abschätzen zu können, bietet TMVA ein Makro an, welches die Klassifikation der Trainings- und Testdaten übereinanderlegt. Ist hierbei eine starke Abweichung der beiden Verteilungen zu erkennen, liegt *Overtraining* vor. Um das *Overtraining* quantitativ auszudrücken, führt TMVA den Kolmogorov-Smirnov Test durch. Dieser gibt ein Maß dafür, wie wahrscheinlich es ist, dass zwei unterschiedliche Verteilungen zu einer Verteilung gehören. Für den Kolmogorov-Smirnov Test gibt es kritische Werte, um mit einer gewissen Sicherheit von gleichen Verteilungen ausgehen zu können. Ist folgende Bedingung erfüllt

$$D_{n,n'}(\alpha) > c(\alpha)\sqrt{\frac{n+n'}{nn'}},$$

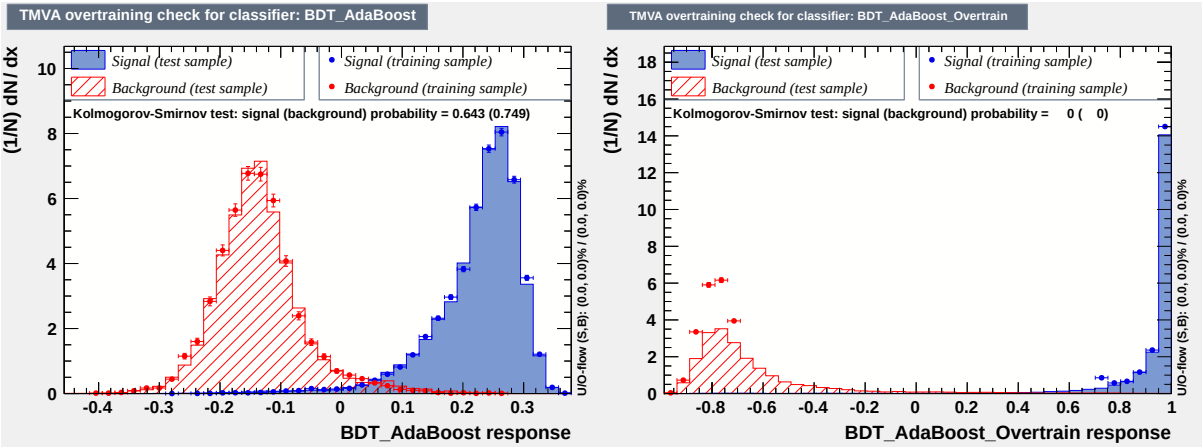


Abbildung 4.8: Überlagerung von Test- und Trainingsdaten zur Überprüfung des *Overtrainings* am Beispiel zweier BDTs. Gefüllt sind die Ausgaben der Methode für den Testdatensatz dargestellt. In Punkten ist zum Vergleich der Trainingsdatensatz eingezeichnet. Links: BDT mit geringem *Overtraining*. Rechts: BDT mit starkem *Overtraining*, da Test- und Trainingsdaten nicht übereinstimmen. Kolmogorov-Smirnov Test gibt quantitative Beschreibung, wie wahrscheinlich es ist, dass beide Datensätze der gleichen Verteilung angehören.

kann von einer mit Sicherheit $1 - \alpha$ davon ausgegangen werden, dass beide Datensätze der gleichen Verteilung angehören. Dabei sind n und n' die Anzahl der Ereignisse in einer Verteilung und $c(\alpha)$ eine Funktion, die von der Signifikanz α abhängt. Für eine Signifikanz von $\alpha = 0{,}001$, folgt für $c(\alpha) = 1{,}95$ und es ergibt sich für 10 000 Ereignissen in beiden Datensätzen ein Wert von $D_{n,n'} = 0{,}0276$. [13] Da im Untergrund- etwa 14 000 Ereignisse und im Signaldatensatz etwa 54 000 Ereignisse enthalten sind, kann der Wert als Größenordnung für geringes *Overtraining* genommen werden. Das in Abbildung 4.8 dargestellte Beispiel mit $D_{n,n'} = 0{,}643$ für die Signaldatensätze und $D_{n,n'} = 0{,}749$ für die Untergrunddatensätze kann somit als Methode mit sehr geringem *Overtraining* angesehen werden.

Ein Beispiel ist in Abbildung 4.8 dargestellt.

4.2.2 Likelihood-Ratio

Die Likelihood-Ratio Methode, in TMVA unter `kLikelihood` aufzurufen, basiert auf dem Verhältnis zweier Likelihood-Funktionen. Eine Likelihood-Funktion $\mathcal{L}$ gibt die Wahrscheinlichkeit an, dass eine Variable unter Annahme eines gewissen Modells einen bestimmten Wert annimmt. Das Konzept der Methode ist nun mittels Wahrscheinlichkeitsdichtefunktionen ein Modell zu erstellen, welches die Eingabevariablen für Signal und Untergrund reproduziert.

$$Q = \frac{\mathcal{L}_S(i)}{\mathcal{L}_S(i) + \mathcal{L}_B(i)}$$

Hierbei steht S für Signal und B für Untergrund. Mittels dieser Definition erhält man über den Wert von Q eine Aussage über die Güte eines Modells im Vergleich zum anderen. Dabei ist $\mathcal{L}$ definiert über

$$\mathcal{L}_{S/B}(i) = \prod_{k=1}^{n_{\text{var}}} p_{S/B,k}\left(x_k(i)\right);$$

die Likelihood-Methode ist also das Produkt der einzelnen Wahrscheinlichkeiten aller Eingabe Variablen. Die Wahrscheinlichkeitsdichtefunktion $p_{S/B,k}$ einer Variable wird folgendermaßen normalisiert:

$$\int_{-\infty}^{+\infty} p_{S/B,k}(x_k)\ \mathrm{d}x_k = 1$$

Parameter Die Likelihood-Methode besitzt wenige wählbare Parameter, welche beim Aufruf gebucht werden können. Es gibt die Option `TransformOutput=True(False)`, welche eine Transformation auf die Ausgabe durchführt:

$$y_L(i) \to y'_L(i) = -\tau^{-1} \ln\left(y_L^{-1}(i) - 1\right).$$

Diese vergrößert die Ausgabe, da die Likelihood-Methode sonst dazu neigt etwa die Hälfte der Ereignisse bei 0 und die andere Hälfte bei 1 zu klassifizieren.

4.2.3 Artificial Neural Networks - ANN

Der Begriff der künstlichen neuronalen Netzwerken bezeichnet eine Reihe von Modellen, die eine vereinfachte Abstraktion von biologischen neuronalen Netzwerken bilden. Grundsätzlich versteht man darunter einen Graph aus Knotenpunkten (Neuronen) mit Verbindungen. Die Verbindungen haben verschiedene Gewichtungen. Auf Basis der Signale, welche am Eingang der Neuronen anliegen, folgt eine Ausgabe der Neuronen. Im Laufe des Trainierens werden die Gewichtungen über einen Lernalgorithmus variiert, um ein gegebenes Problem zu approximieren. Allgemein kann jedes Neuron beliebig viele Verbindungen zu anderen Neuronen haben.

Innerhalb TMVAs kommen nur Feed-Forward Networks vor, genauer gesagt **M**ulti**l**ayer **P**erceptrons (MLP). Diese zeichnen sich dadurch aus, dass es mehrere Ebenen von Neuronen gibt, aber die Neuronen einer Ebene immer nur mit Neuronen der nächsten Ebene verbunden sind. Es gibt jedoch keine Verbindungen zurück. Die Ein- und Ausgabeebene ist jeweils festgelegt. In der Eingabeebene ist für jede Eingabevariable ein Neuron plus zusätzlich ein sogenanntes *Bias*-Neuron. Dieses hat immer den Ausgabewert 1. Mithilfe eines solchen Neurons, kann der Ausgabewert der weiteren Neuronen verschoben werden, indem die Gewichtung dieses Neurons verändert wird. Dies liegt darin begründet, dass die Ausgabe eines Neurons eine gewichtete Linearkombination aller Eingänge ist. Die Ausgabeebene besteht aus einem einzelnen Neuron, welches die Gesamtausgabe der neuronalen Netzwerks darstellt. Die Ebenen zwischen Eingabe- und Ausgabe-Ebene werden als versteckte Ebenen (engl. *hidden layers*) bezeichnet. In jeder versteckten Ebene ist ein weiteres *Bias*-Neuron enthalten. Eine Skizze eines solchen Netzwerks ist in Abbildung 4.9 dargestellt. Diese Art eines neuronalen Netzwerks hat einerseits den Vorteil für n Neuronen nicht n^2 Verbindungen zu haben und somit bleibt die Rechenleistung auch für größere Netzwerke im Rahmen. Andererseits kann mittels eines MLP nach dem Cybenko Theorem [14] mit einer *hidden layer* mit genügend Neuronen bereits jedes andere Netzwerk zu beliebiger Genauigkeit angenähert werden. Somit bietet es sich an, die Anzahl der Neuronen auf der einzelnen versteckten Ebene so weit zu erhöhen, bis eine gewisse Genauigkeit, in Abhängigkeit von der verfügbaren Rechenleistung, erreicht ist. In der Anwendung bietet es sich an etwa einige Neuronen mehr zu nutzen, als Variablen verwendet werden. Zu viele Neuronen können bei kleinen Datensätzen leicht zu *Overtraining* führen.

Das Netzwerk wird mehrfach mit den selben Trainingsdaten gefüllt und so werden über häufige Iterationen die Gewichtungen (positiv oder negativ) und die Stärke der Verbindung modifiziert. Dafür gibt es verschiedene Lernmethoden für die Neuronen. Es wurde einmal das sogenannte **B**ack**p**ropagation Verfahren (BP) und das **B**royden-**F**letcher-**G**oldfarb-**S**hannon Verfahren (BFGS) getestet.

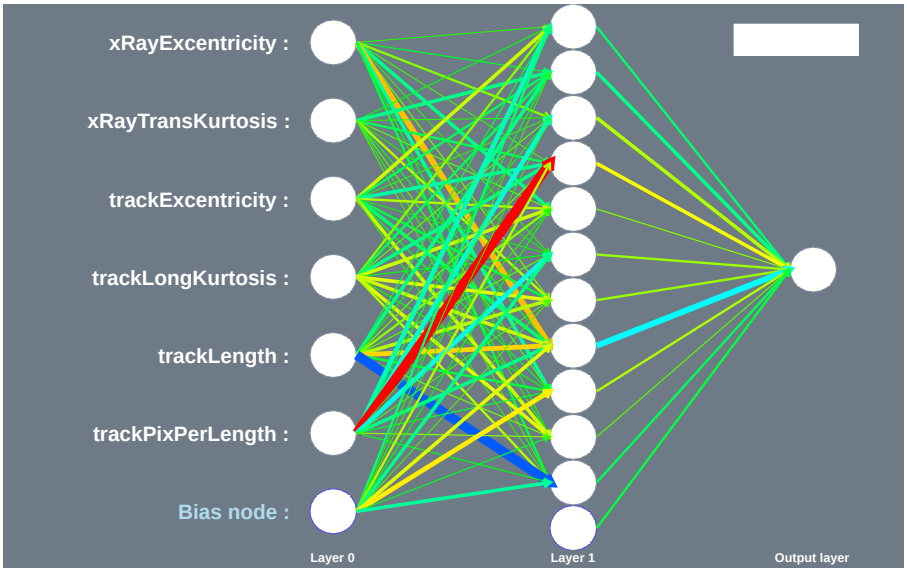


Abbildung 4.9: Skizze eines Feed-Forward ANNs (Beispiel aus TMVA). Die Dicke der Linien gibt die Stärke des Gewichts an, rote Färbung steht für positive Gewichtung, blau für negativ und grün für geringes absolutes Gewicht.

Parameter Für das neuronale Netzwerk gibt es eine große Anzahl wählbarer Parameter. Zunächst sollte immer die Option `VarTransform=Norm` ausgewählt sein. Diese führt eine Normalisierung der Daten auf -1 bis 1 durch. Für ein neuronales Netzwerk ist dies unbedingt notwendig, da im Prinzip die Ausgabe der Neuronen direkt von den Eingangssignalen abhängt. Hat man also Variablen, welche mehrere Größenordnungen auseinander liegen, führt dies zwangsläufig zu einem nicht repräsentativen Modell. Die Option `Sampling` gibt den prozentualen Anteil an, welcher vom Trainingsdatensatz letztlich wirklich für das Trainieren verwendet wird. Dies ist eine wichtige Option, da aufgrund der Komplexität eines neuronalen Netzwerks, die Rechenzeit mit der Anzahl der Ereignisse stark ansteigt. Entweder können die Ereignisse zufällig ausgewählt werden, dann muss `SamplingImportance=1` gesetzt werden. Mit einem Wert kleiner 1 wird die Wahrscheinlichkeit einer Variablen erneut ausgewählt zu werden, durch den Wert von `SamplingImportance` geteilt, wenn sie den Fehler der Testdaten vergrößert und multipliziert, wenn sie ihn verkleinert. Damit wird gewährleistet, dass Ereignisse, die zu *Overtraining* führen können, weniger berücksichtigt werden, um dies zu vermindern.

Die Architektur des Netzwerks wird über den Parameter *HiddenLayer* bestimmt. Hierbei können beliebig viele Ebenen mit beliebig vielen Neuronen festgelegt werden. Allerdings ist eine Ebene aufgrund des Cybenko Theorems für die meisten Anwendungen ausreichend. Die Anzahl der Neuronen kann über die Anzahl der Eingabevariablen `N` angegeben werden.

Der Parameter `ConvergenceTests` gibt an, dass nach jeweils `n` Durchläufen ein Konvergenz-Test durchgeführt wird. Die Anzahl der Durchläufe wird mit `TestRate=n` festgelegt. Hierbei wird überprüft, ob sich in einer Iteration die Ausgabe y_{ANN} weniger als der Parameter `ConvergenceImprove` ändert. Ist dies der Fall, wird das Trainieren beendet. Mit `ConvergenceTests=m` wird festgelegt, dass `m` Iterationen hintereinander eine kleinere Änderung erfahren müssen. Alternativ kann auch die Anzahl der Durchläufe insgesamt mit `NCycles` festgelegt werden.

Backpropagation - BP

Allgemein kann die Ausgabe eines Neurons folgendermaßen ausgedrückt werden

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right). \tag{4.1}$$

Dabei bezeichnet φ die Aktivierungsfunktion des Neurons. Diese hat meistens eine sigmoide Form, z.B. der Tangens Hyperbolicus, kann aber auch linear sein. In jedem Fall ist sie monoton steigend. w_{kj} ist das Gewicht des j-ten Neuron der vorigen Ebene auf Neuron k. Die Ausgabe aller Neuronen ergibt sich dann zu:

$$y_{\text{ANN}} = \sum_{k=1}^{n_h} y_k w_{j1}.$$

Dabei ist n_h die Anzahl Neuronen in der versteckten Ebene und w_{j1} die Gewichte der Neuronen in der versteckten Ebene zum Ausgabeneuron.

Es wird eine Fehlerfunktion (engl. *error function*) E definiert, welche es zu minimieren gilt. Diese wird über

$$E(\mathbf{x}_1, \ldots, \mathbf{x}_N|\mathbf{w}) = \sum_{a=1}^{N} E_a(\mathbf{x}_a|\mathbf{w}) = \sum_{a=1}^{N} \frac{1}{2}(y_{\text{ANN},a} - \hat{y}_a)^2$$

beschrieben. Dabei ist $\mathbf{x}_a$ ein Vektor aus den Input Variablen für das Ereignis a, $\mathbf{w}$ ist der Vektor der veränderbaren Gewichten innerhalb des Netzwerks. Für jedes Ereignis a wird der Output des neuronalen Netzwerks $y_{\text{ANN},a}$ als Summe aller Ausgaben nach Gleichung 4.1, unter Berücksichtigung der Gewichtungen zum Output, berechnet. Dieser wird mit dem gewollten Output von $\hat{y}_a \in \{0, 1\}$ verglichen.

Der Lernalgorithmus basiert nun darauf diese Fehlerfunktion zu minimieren. Dies geschieht iterativ folgendermaßen:

$$\mathbf{w}^{n+1} = \mathbf{w}^n - \eta \nabla_w E.$$

Hierbei ist η die *learning rate*. Das heißt es werden die Gewichte so variiert, dass E am schnellsten kleiner wird.

Die Gewichte, welche mit der Ausgabe-Ebene verbunden sind werden folgendermaßen geändert

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

während die mit den versteckten Ebenen verbundenen Neuronen über

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

variiert werden. Im letzten Schritt wird eine Eigenschaft der Aktivierungsfunktion φ, welche eine Sigmoidfunktion ist, verwendet: $\text{sig}'(t) = \text{sig}(t)(1 - \text{sig}(t))$.

Broyden-Fletcher-Goldfarb-Shannon - BFGS

Diese Methode wurde unabhängig von den Mathematikern Broyden, Fletcher, Goldfarb und Shannon im Jahr 1970 beschrieben [15–18]. Die BFGS Methode gehört zu den Quasi-Newton-Verfahren. Dies bedeutet, dass es die Hesse-Matrix, welche bei der Berechnung der neuen Gewichte auftritt, nicht explizit

berechnet, sondern approximiert. Die Hesse-Matrix ist die Matrix aller zweiten, möglichen Ableitungen einer Funktion $f(x)$

$$H_f(x) := \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{i,j=1,\ldots,n}.$$

Zunächst werden zwei Vektoren D und Y berechnet. Der Vektor D beinhaltet die Änderung der Gewichtungen von einer Iteration zur nächsten. Der Vektor Y besteht aus der Änderung der Gradienten der Gewichte.

$$\begin{aligned} D_i^{(k)} &= w_i^{(k)} - w_i^{(k-1)} \\ Y_i^{(k)} &= g_i^{(k)} - g_i^{(k-1)} \end{aligned}$$

Der Index i bezeichnet das jeweilige Gewicht und der Index k den Iterationsschritt. w_i ist das i-te Gewicht, g_i der Gradient des i-ten Gewichts. Aus diesen Vektoren kann die inverse Hesse Matrix $H^{-1(k)}$ angenähert werden.

Mittels der inversen Hesse Matrix wird nun der Vektor D neu berechnet:

$$D^k = -H^{-1(k)} Y^{(k)}.$$

Nun werden erneut über eine Fehlerfunktion die neuen Gewichte berechnet, indem die Fehlerfunktion lokal mit einer Parabel angenähert wird.

4.2.4 Support Vector Machines - SVM

Unter den **S**upport **V**ector **M**achines (SVM) versteht man eine mit neuronalen Netzwerken verwandte Methode. Bereits in den 1960er Jahren mathematisch beschrieben, dauerte es 30 Jahre bis die Methode auf nichtlineare Probleme erweitert werden konnte. Seitdem werden sie in vielen Anwendungsbereichen verwendet.

Das Konzept der SVM besteht darin die Eingabevariablen in einem vieldimensionalem Raum abzubilden und dort eine Hyperebene zu finden, die beide Datensätze maximal trennt, d.h. die Anzahl der sogenannten *support vectors* zu minimieren und den Abstand zu selbigen gleichzeitig zu maximieren. In zwei Dimensionen kann dies für den linear trennbaren Fall sehr bildlich, wie in Abbildung 4.10, dargestellt werden.

Die Datensätze bilden jedoch nicht zwangsläufig einen linear trennbaren Hyperraum. In diesem Fall wird über die *kernel function*, welche gaußisch ist, in einen anderen Raum transformiert, in dem eine Trennung über eine lineare Ebene möglich ist.

Parameter Trotz ihrer Komplexität bieten SVMs vergleichsweise wenige Parameter. Aufgrund der Funktionsweise der SVM kann `VarTransform=None` gesetzt werden, d.h. es wird keine Transformation auf die Eingabedaten durchgeführt. Der Parameter `C` ist der *cost parameter*. Dieser gibt ein Maß für die Toleranz falsch klassifizierte Ereignisse an. Der Abstand von Ereignissen, die auf der falschen Seite der Hyperebene liegen, wird mit dem *cost parameter* multipliziert. Wird der Wert zu hoch gewählt, dürfen also nur sehr wenige Ereignisse falsch klassifiziert sein, führt dies zu *Overtraining*. Der Toleranz Parameter `Tol` ist die Präzision für Verbesserung der Hyperebene mit jeder Iteration. Der nächste Parameter, `Gamma` steht im Zusammenhang mit der Breite σ der Gauß-Kernel Funktion

$$\Gamma = \frac{1}{2\sigma^2}.$$

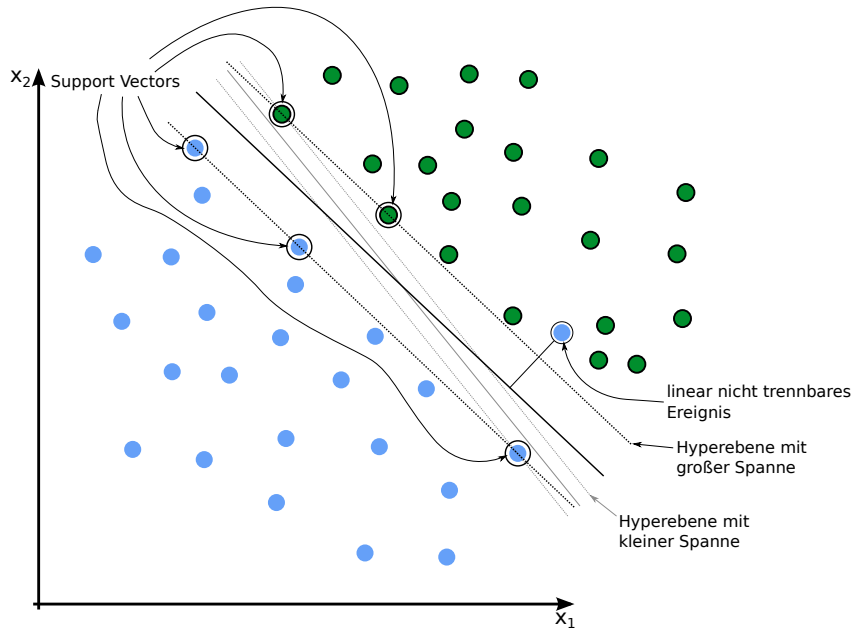


Abbildung 4.10: 2D Darstellung der Trennung des Signal- und Untergrunddatensatzes mittels einer Linie für lineare Separation. *support vectors* bilden die Basis der Hyperebene. SVM Algorithmus versucht die Spanne zu maximieren. Nicht trennbares Ereignis wird mit *cost parameter* multipliziert.

Diese drei Parameter bieten sehr viel Spielraum und müssen dem Problem entsprechend angepasst werden.

4.2.5 Boosted Decision Trees - BDT

Unter Boosted Decision Trees versteht man eine spezielle Sammlung, einen Wald (engl. *forest*), von Decision Trees. Ein Decision Tree ist ein Modell zur multivariaten Analyse, welches auf häufigen Ja/Nein Entscheidungen basiert und damit die Datensätze (Signal und Untergrund) in immer kleinere Teilmengen aufteilt.

Bei einem Decision Tree wird zunächst die Variable gesucht, welche die beste Trennung zwischen Signal und Untergrund liefert. Ist diese gefunden, wird der Schnittwert festgelegt, bei dem diese Trennung am besten funktioniert. Dies erzeugt den ersten Knotenpunkt des Decision Trees. Darüber wird die Menge der Signal- und Untergrunddaten in zwei Teilmengen aufgespaltet. Nun wird jede Teilmenge erneut als Ausgangsdatensatz betrachtet für welchen wieder die beste Separationsvariable gesucht wird. Dies geschieht so lange, bis die Teilmenge in einem Knotenpunkt eine bestimmte Anzahl von Ereignissen, welche als Parameter mitgegeben wird, unterschreitet. Ein solcher letzter Knotenpunkt wird entweder als Signal- oder Untergrundartig bezeichnet, woraus sich die Klassifikation eines Ereignisses in diesem Knotenpunkt ergibt. Über diesen Algorithmus wird eine Struktur, ein Decision Tree, aufgebaut, welche wie in Abbildung 4.11 dargestellt werden kann.

Es ist wichtig die richtige Größe für den Parameter, welcher die Minimalanzahl von Ereignissen für den letzten Knotenpunkt setzt, für den gegebenen Datensatz herauszufinden. Ist der Parameter zu klein gewählt, wird der Tree zu komplex werden und zu *Overtraining* neigen, d.h. er wird Eigenschaften des Datensatzes erlernen, die durch statistische Schwankungen auftreten. Wird der Parameter jedoch zu klein gewählt kann der Tree wichtige Eigenschaften der Daten nicht mehr auflösen.

Alternativ wird das Wachstum gestoppt, wenn das Verhältnis von Signal zu Untergrund in einem

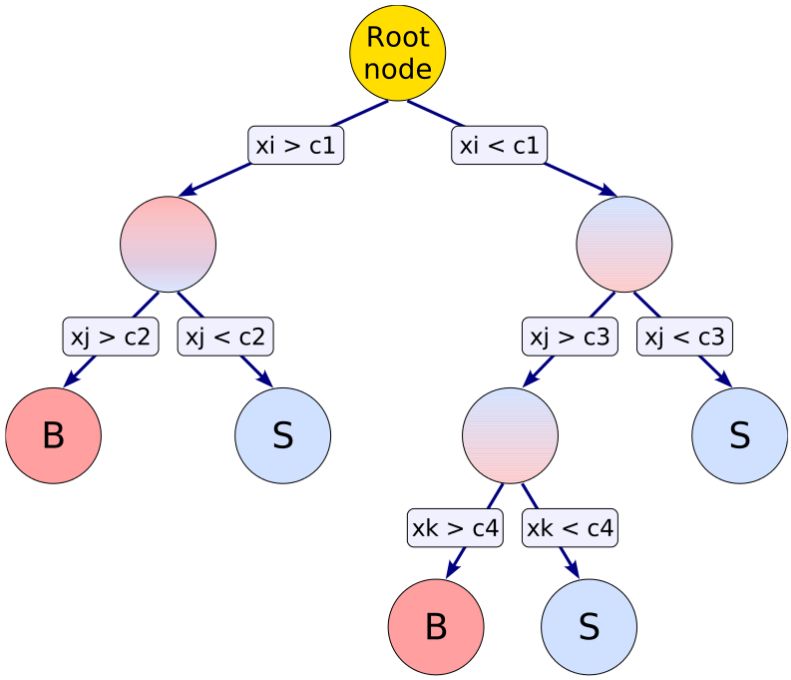


Abbildung 4.11: Skizze einer Darstellung eines Decision Trees. Entnommen aus [12]. In jedem Knotenpunkt wird der Datensatz aufgespaltet, aufgrund eines Schnittwerts auf eine Variable x_i. Bei jedem Knotenpunkt wird neu entschieden, welche Variable verwendet wird. Daher kann eine Variable häufiger auftreten. Die untersten Knotenpunkte werden entweder mit S für Signal oder B für Background, also Untergrund bezeichnet, abhängig davon welche Ereignisse häufiger enthalten sind.

Knotenpunkt einen gewissen Wert übersteigt oder unterschreitet.

Nun wird jedoch nicht nur ein einzelner Tree gebaut, sondern ein ganzer Wald von Trees, ein *forest*. Üblicherweise werden mindestens 200 Trees aufgebaut. Für jeden Tree wird wieder erneut der gesamte Datensatz verwendet. Beim Aufbau der weiteren Trees kommt nun sogenanntes Boosting zum Einsatz, damit sich die Trees unterscheiden. Bei der Klassifikation mit dem BDT, verwendet man den Mittelwert aller Trees aus dem *forest*, um einen Ausgabewert für ein Ereignis zu erhalten.

Boosting Allgemein bezeichnet Boosting die Neugewichtung von Ereignissen aufgrund ihrer Klassifizierung im vorherigen Durchlauf. Die meistverbreiteste Methode des Boosting ist das Adaptive Boosting (AdaBoost) [19]. Dabei werden Ereignisse, die zuvor fälschlicherweise als Signal oder Untergrund klassifiziert wurden, höher gewichtet, als jene, die richtig eingestuft wurden. Dies wird beim Erzeugen jedes neuen Trees durchgeführt.

Bevor ein neuer Tree aufgebaut wird, wird zunächst die *misclassification rate* des vorherigen Trees bestimmt. Sie ist ein Maß für die Anzahl der falsch klassifizierten Ereignisse in einem Tree.

$$\mathrm{err}_k = \frac{\sum_{i=1}^{N} w_i I(x_i)}{\sum_{i=1}^{N} w_i}$$

Dies ist die *misclassification rate* für den k-ten Tree. Sie ist per Definition $\leq 0{,}5$. Hierbei ist w_i die Gewichtung des i-ten Ereignisses, N die Gesamtzahl der Ereignisse. Die Funktion $I(x_i)$ ist dann 1, wenn Ereignis x_i falsch eingestuft wurde und 0 sonst. Nun wird der *Boosting Faktor* α_k über

$$\alpha_k = \left(\frac{1 - \mathrm{err}_k}{\mathrm{err}_k}\right)^{\beta},$$

definiert. Dabei ist β ein Faktor, der als Parameter der Methode wählbar ist. Im $k+1$-ten Tree werden die Gewichte aller falsch klassifizierten Ereignisse mit α_k multipliziert. Da $\alpha_k \geq 1$, wird ihre Gewichtung damit erhöht. [20]

Mittels des sich ergebenden *forest* wird ein Ereignis nun darüber klassifiziert, wie es im Mittel über alle Trees einsortiert wird, unter Berücksichtigung des Boosting Faktors. Die Klassifikation $T(x_i)$ wird über

$$T(x_i) = \sum_{k=1}^{N_{\mathrm{Tree}}} \alpha_k T_k(x_i)$$

berechnet. $T_k(x_i)$ ist die Ausgabe eines einzelnen Trees und ist 1, wenn das Ereignis als Signal eingestuft wird und -1, wenn es als Untergrund klassifiziert wird.

Aus der Funktionsweise ergibt sich gleich ein Vor- und Nachteil. Eine Variable die eine sehr schlechte Trennung bietet kann nicht dazu führen, dass die Methode schlechter arbeitet. Allerdings wird auch nicht die gesamte Separationsfähigkeit aller Variablen berücksichtigt.

Parameter Für die Boosted Decision Trees gibt es viele einstellbare Parameter. Zunächst bietet es sich an über `BoostType` die Art des *boosting* zu wählen.

Die Struktur des *forest*, bzw. die Anzahl der Trees, wird über `NTrees` bestimmt. Der einzelne Tree kann einerseits entweder über `MaxDepth` in der Tiefe beschränkt werden, über `NNodes` anhand der Knotenpunkten im Tree, oder mit `nEventsMin` über die Anzahl der Ereignisse, die mindestens in einem Knotenpunkt sein müssen, damit dieser sich ein weiteres Mal aufspaltet.

Eine weitere Reihe Parameter beziehen sich auf das sogennante *pruning*. Damit bezeichnet man ein

Beschneiden der Trees, nachdem sie gewachsen sind. Darüber soll mögliches *Overtraining* der Trees verhindert wird. Es wird dabei von Ende des Trees begonnen und jene Knotenpunkte weg geschnitten, die insignifikant sind. Dafür können über `PruneMethod` verschiedene Methoden gewählt werden. Über `PruneStrength` wird ein Grenzwert festgelegt, mit dem entschieden wird, ob etwas abgeschnitten wird oder nicht. Wählt man den Wert -1, sucht TMVA selber den optimalen Wert.

KAPITEL 5

Vorbereitung der Daten

Als Datensatz dienen die beiden Messreihen, welche bereits in Abschnitt 3.2 erwähnt wurden. Um für diese Ereignisse eine Untersuchung mit TMVA durchführen zu können, müssen zunächst die Eigenschaften berechnet werden. Dazu wurde jeweils ein MarlinTPC Prozessor für die Photonen, sogenannte **XrayObjects**, und für die Untergrunddaten, **Tracks**, geschrieben. Dieser berechnet die Eigenschaften, welche in Tabelle 5.1 dargestellt sind.

Varianz σ Unter der Varianz σ versteht man das zweite zentrale Moment einer Verteilung. Man beschreibt sie für eine diskrete Verteilung über

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 .$$

Dabei bezeichnet $\bar{x}$ den Mittelwert der Verteilung. Sie ist ein Maß für die Breite der Verteilung.

Schiefe - Skewness Unter Skewness versteht man die Schiefe einer Verteilung. Sie beschreibt, ob die Verteilung links und rechts unterschiedlich schnell abfällt. Hierbei handelt es sich in der Statistik um das dritte Moment der Verteilung.

$$S = \frac{1}{N\sigma^3} \sum_{i=1}^{N} (x_i - \bar{x})^3$$

Kurtosis Die Kurtosis bzw. Wölbung einer Verteilung gibt an wie spitz oder flach der Gipfel der Verteilung ist. Sie ist das vierte zentrale Moment einer Verteilung. Meist wird jedoch die Exzess-Kurtosis verwendet, welche folgendermaßen definiert ist:

$$K = \frac{1}{N\sigma^4} \sum_{i=1}^{N} (x_i - \bar{x})^4 - 3.$$

Der Term -3 kommt daher, dass man die Kurtosis um 0 verteilt haben möchte. Eine Gaußverteilung, von der die Abweichung betrachtet wird, hat selbst eine Kurtosis von 3.

Eigenschaft	Erklärung
Allgemein	
runNumber	Nummer des Laufs, aus dem das Ereignis ist
eventNumber	Nummer des Ereignisses innerhalb des Laufs
XrayObjects	
xRayRmsX	Varianz entlang der langen Achse
xRayRmsY	Varianz entlang der kurzen Achse
xRayLongSkewness	Skewness entlang der langen Achse
xRayTransSkewness	Skewness entlang der kurzen Achse
xRayLongKurtosis	Kurtosis entlang der langen Achse
xRayTransKurtosis	Kurtosis entlang der kurzen Achse
xRayExcentricity	Excentrizität der Objekte
xRayRadius	mittlerer Radius der Objekte
xRayLength	Länge entlang der langen Achse
xRayWidth	Länge entlang der kurzen Achse
xRayNpx	Anzahl aktiver Pixel im Ereigniss
xRayCharge	Gesamtladung des Ereignisses
xRayEnergy	Energie des Ereignisses
Tracks	
trackLongRms	Varianz entlang der langen Achse
trackTransRms	Varianz entlang der kurzen Achse
trackLongSkewness	Skewness entlang der langen Achse
trackTransSkewness	Skewness entlang der kurzen Achse
trackLongKurtosis	Kurtosis entlang der langen Achse
trackTransKurtosis	Kurtosis entlang der kurzen Achse
trackExcentricity	Excentrizität der Objekte
trackLength	Länge einer Spur entlang der langen Achse
trackWidth	Länge der Spur entlang der kurzen Achse
trackNpx	Anzahl aktiver Pixel im Ereignis
trackCharge	Gesamtladung des Ereignisses
trackPixPerLength	Zahl aktiver Pixel pro Länge der Spur

Tabelle 5.1: Eigenschaften der **XrayObjects** und **Tracks** und dazugehörige Erklärung.

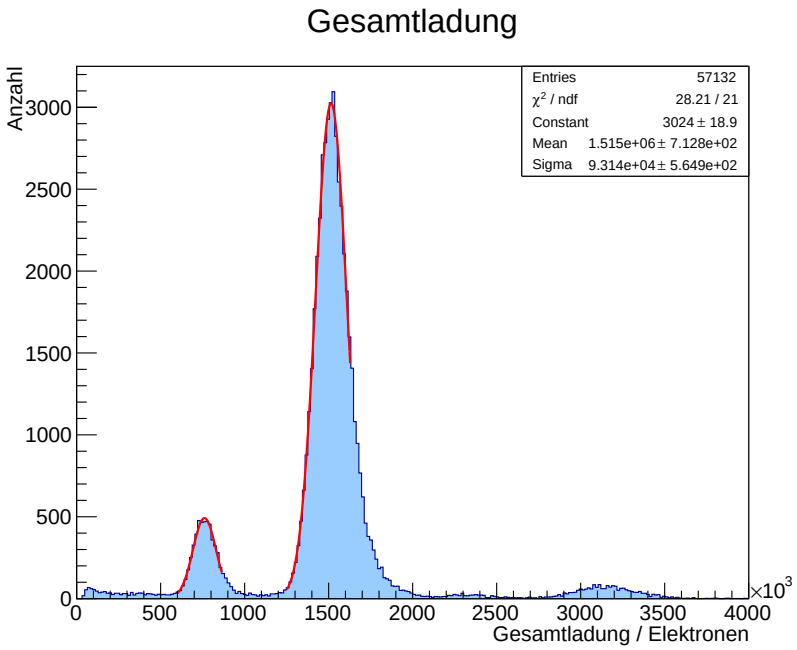


Abbildung 5.1: Spektrum der ^{55}Fe Quelle in Gesamtladungen der Ereignisse. Angepasste Gaußkurven zur Energiekalibrierung. Fit Parameter gehören zum Photopeak.

5.1 Energiekalibrierung

Zusätzlich zu den geometrischen Eigenschaften, welche von den oben genannten Prozessoren berechnet werden, wird für jedes Ereignis die Energie berechnet. Dazu muss jedoch zuerst eine Energiekalibrierung durchgeführt werden.

Aufgrund der zuvor in Abschnitt 3.1.2 beschriebenen Eigenschaft des Detektors, dass die Anzahl aktiver Pixel direkt proportional zur Energie des einfallenden Photons ist, lässt sich über das Spektrum der ^{55}Fe-Quelle eine solche Kalibrierung vornehmen. Diese geschieht letztlich jedoch nicht über die Anzahl der aktiven Pixel (auch wenn dies möglich wäre), sondern über die Ladungswerte der Pixel. Nachdem eine Ladungskalibration wie in [6] durchgeführt wurde, kann für jeden Pixel ein Ladungswert in Elektronen angegeben werden. In Abbildung 5.1 ist ein Histogramm der Gesamtladung aller Pixel für den gesamten Datensatz der ^{55}Fe Quelle gezeigt.

Der erste Peak von links entspricht dem Escapepeak bei einer Energie von 2,9 keV. Rechts daneben liegt der Photopeak mit einer Energie von 5,76 keV. Der kleine Peak bei knapp unterhalb $2{,}5 \cdot 10^6\, e$ entspricht dem Fall, dass zwei Photonen in den Detektor gelangt sind, allerdings eines Energie über ein Escape Photon verloren hat. Der letzte Peak bei oberhalb von $3 \cdot 10^6\, e$ entspricht dem Fall zweier Photonen, die so überlappen, dass der MarlinTPC Prozessor, welcher **XrayObjects** sucht, nicht in der Lage war sie als zwei Objekte zu erkennen.

An den Escape- und Photopeak wurde eine Gaußkurve angepasst:

$$f(x) = A \exp\left(-\frac{\left(x - x_{\text{pos}}\right)^2}{2\sigma^2}\right).$$

Anhand der Positionen der Peaks wird nun eine Gerade bestimmt. Für den Escapepeak bei 2,9 keV

erhält man eine Ladung von:

$$Q = 7{,}6103 \cdot 10^5\, e \pm 1{,}2961 \cdot 10^3\, e.$$

Für den Photopeak bei 5,76 keV:

$$Q = 1{,}5145 \cdot 10^6\, e \pm 7{,}1275 \cdot 10^2\, e.$$

Mit diesen Werten wurde folgende Gerade berechnet:

$$f(Q) = 3{,}811 \cdot 10^{-6} Q - 1{,}221 \cdot 10^{-2}.$$

Hiermit kann nun für jedes Ereignis über die Gesamtladung Q die Energie berechnet werden. Die Konstante der Funktion ist klein genug, sodass sie mit Null verträglich ist. Dies ist sinnvoll, da ohne Ladung keine Energie im Detektor deponiert werden kann.

5.2 Erzeugung künstlicher, niederenergetischer Photondaten

Der neue GridPix Detektor kann prinzipiell bis zu Energien unterhalb von 1 keV messen. Da dafür jedoch keine Messdaten mit einer radioaktiven Quelle existieren, macht man sich zunutze, dass die Energie direkt proportional zu der Anzahl der aktiven Pixel in den Ereignissen ist. Es ist naheliegend statistisch einen gewissen Prozentsatz der Pixel wegzulassen, um damit künstlich niederenergetische Ereignisse zu simulieren. Mit diesen Ereignissen kann die Effizienz des Detektors in diesem Energiebereich simuliert werden.

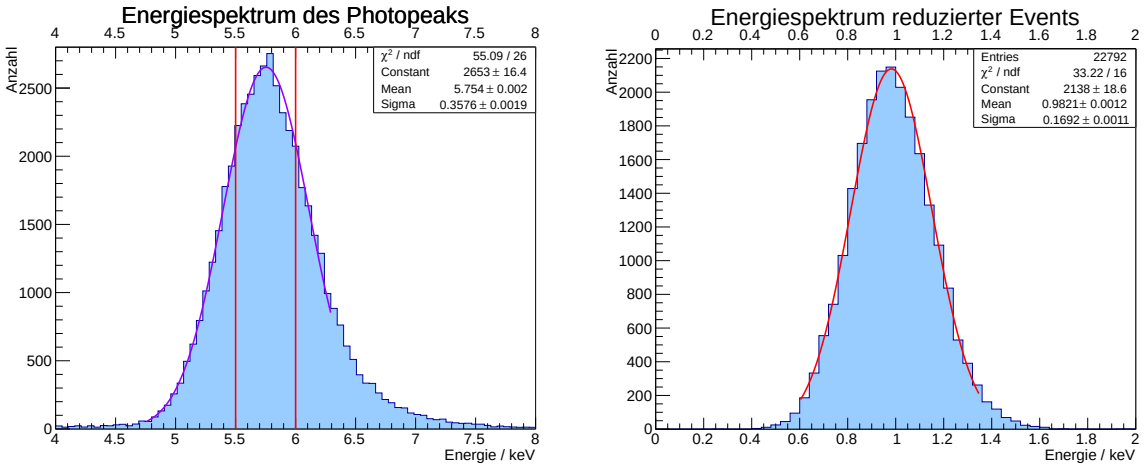
Folglich wurde ein Prozessor geschrieben, welcher diese Aufgabe erledigt. Die Daten der Messung mit der ^{55}Fe Quelle werden eingelesen und es werden von jedem Ereignis, welches eine Energie von ~5,76 keV hat, die aktiven Pixel geholt. Weiterhin wird ein Faktor f definiert

$$f = \frac{E_{\text{Target}}}{E_{\text{Photo}}},$$

welcher das Verhältnis aus der zu erreichenden Energie E_{Target} und der Energie des Photopeaks E_{Photo} ist. Nun läuft man für jedes Ereignis über alle getroffenen Pixel. Dabei wird das **TRandom3** Objekt aus ROOT verwendet, um eine Zufallszahl zwischen 0 bis 1 zu erzeugen. Es wird **TRandom3** verwendet, da er auf dem *Mersenne Twister* basiert und somit eine extrem geringe Korrelation aufweist [21]. Hier wird über eine einfach `if` Abfrage überprüft, ob die erzeugte Zufallszahl kleiner ist, als der Faktor f. Falls dies der Fall ist, wird der Pixel behalten, sonst wird er gelöscht.

Mit diesem einfachen Prozessor erreicht man Ergebnisse, welche durchaus einer Messung mit einer niederenergetischeren Quelle entsprechen könnten. Sowohl das Energiespektrum, als auch die Ereignisse selber sehen aus wie von einem solchen Ereignis erwartet wird. Wichtig ist, dass Eigenschaften wie die statistischen Momente unverändert bleiben, da sie nicht von der Energie, sondern der Diffusion abhängen. Hierbei wurden Ereignisse verwendet mit einer Energie oberhalb von 5,5 keV und unterhalb von 6 keV. Aus 57 165 Ereignissen mit einem **XrayObject** aus den Signaldaten, werden durch den Schnitt auf die Energie 22 806 energetisch reduzierte Ereignisse. In Abbildung 5.2 ist das Energiespektrum der künstlichen Ereignisse dargestellt.

Weiterhin sind in Abbildung 5.3 Beispiele von Ausgangsereignissen und den daraus erzeugten Ereignissen dargestellt.



(a) (b)

Abbildung 5.2: (a) Energiespektrum des Photopeaks. In Rot sind die verwendeten Schnitte für das Erstellen des Spektrums in (b) eingezeichnet. (b) Energiespektrum der energetisch reduzierten Ereignisse. Es wurde eine Energie von $E_{\text{Target}} = 1\,\text{keV}$ angestrebt.

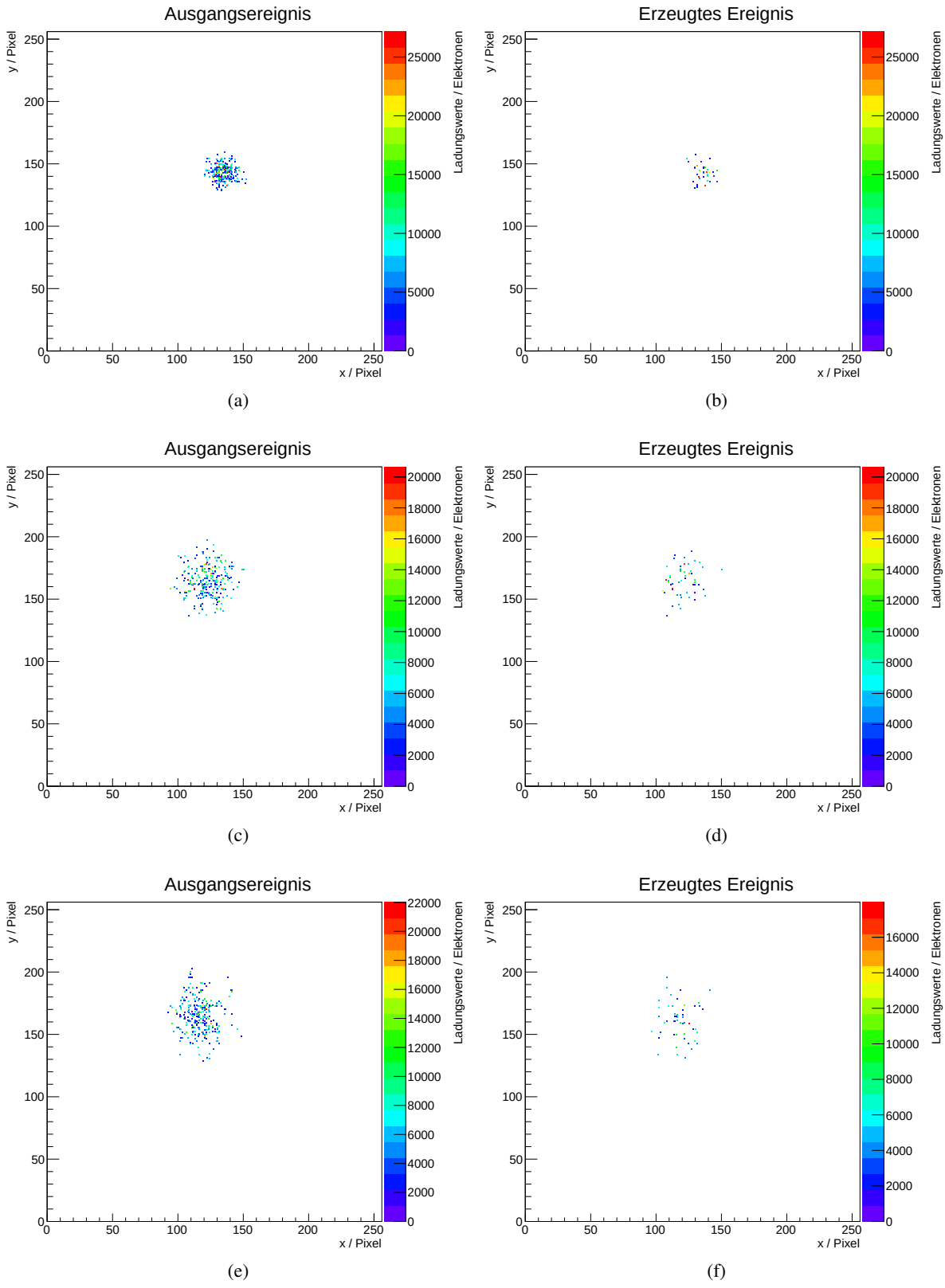


Abbildung 5.3: Beispiele für die Erzeugung künstlicher, niederenergetischer **XrayObjects**. Links das Ausgangsereignis und rechts das gleiche Ereignis reduziert. Die Ladungswerte der Pixel sind farbig dargestellt.

KAPITEL 6

Klassifikation mittels TMVA

In diesem Kapitel wird auf das Trainieren und Testen der verschiedenen Methoden unter gewissen Gesichtspunkten eingegangen. Zunächst wird das Training mit allen verfügbaren Daten vorgestellt. Anschließend wird auf das Training mit einem Schnitt auf die Energie um den Photopeak eingegangen. Zuletzt wird das Training der künstlich erzeugten Ereignisse behandelt. Es wird dabei jeweils beschrieben welche Parameter verwendet wurden und wie erfolgreich die Methoden sind. Diese werden mit den Ergebnissen aus [6] verglichen. Für alle Klassifikationen gilt, dass der Untergrunddatensatz vollständig verwendet wird. Dieser hat eine nahezu kontinuierliche Energieverteilung, während die Signaldaten auch ohne einen Schnitt bereits eine Häufung um den Escape- und Photopeak haben. Zunächst werden die Ergebnisse aus [6] vorgestellt.

Die bisherige Klassifikation basiert auf den folgenden Eigenschaften:

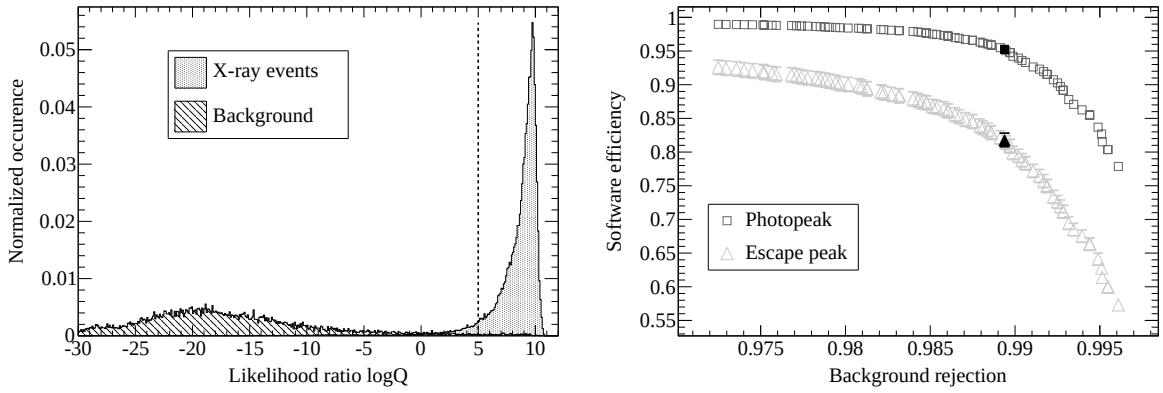
- trackLength
- trackPixPerLength
- trackLongKurtosis
- trackExcentricity
- xRayTransKurtosis
- xRayExcentricity.

Die Bedeutung der Bezeichungen wurde in Abschnitt 5 in Tabelle 5.1 erläutert. Mittels einer logarithmischen Likelihood Methode wurde die Klassifikation erreicht, wie sie in Abbildung 6.1 gezeigt ist.

Für den Schnittwert von $\log Q = 5$ ergibt sich für die Ereignisse bei 5,76 keV eine Software-Effizienz von (95,3 ± 0,3) % und für die des Escapepeaks bei 2,9 keV eine Effizienz von (85,8 ± 1,0) %. Die Untergrundunterdrückung liegt bei (98,9 ± 0,6) % [22].

6.1 Trainieren mit allen Daten

Die einfachste Möglichkeit des Trainierens der Methoden ist es, alle Daten die zur Verfügung stehen zur Separation zu verwenden. Dies wurde für verschiedene Parameter aller Methoden durchgeführt. Die Methoden, welche letztlich verwendet wurden, sind im Folgenden aufgelistet. Für alle Methoden, wenn



(a) (b)

Abbildung 6.1: (a) Bisherige Klassifikation aus [22] mittels einer logarithmischen Likelihood Methode. Die gestrichelte Linie markiert den Schnittwert. X-ray events = Signal und Background = Untergrund. (b) Die zu (a) gehörige Untergrundunterdrückung. Eingezeichnet sind die Signaleffizienzen für den Schnittwert $\log Q = 5$.

nicht anders beschrieben, gilt, dass die Hälfte der Daten zum Trainieren und die andere Hälfte zum Testen der Methoden verwendet wird. Die Verteilungen aller Eingabevariablen sind im Anhang A.2 dargestellt.

Die Optionen der gebuchten Methoden sind in Tabelle 6.1 zusammengefasst.

MLP Es wurde jeweils einmal das **Backpropagation** und das **BFGS** Verfahren trainiert. Wie zuvor beschrieben, werden die Eingabedaten normiert. Um die Rechenzeit zu minimieren, werden nur 30 % der Trainingsdaten verwendet; diese werden zufällig ausgewählt. Beide Netzwerke haben eine Zwischenebene mit `N+5` Neuronen, wobei `N` die Anzahl der Variablen ist. Es wurden 20 Variablen verwendet. `N+5` bietet einen Kompromiss aus Genauigkeit und Rechenzeit. Alle 5 Durchläufe sollen vier (`MLP_BFGS`) bzw. sechs (`MLP_BP`) aufeinanderfolgende Konvergenztests durchgeführt werden. Die Lernrate wird auf 0,05 gesetzt.

SVM Da die SVM, anders als die neuronalen Netzwerke, keine Parameter zur Beschränkung des Trainingsdatensatzes bieten, laufen sie unter einem weiteren **Factory** Objekt. In diesem werden 10000 Ereignisse aus dem Signaldatensatz und die Hälfte des Untergrunddatensatz zum Training verwendet. Die drei verfügbaren Parameter (beschrieben in 4.2.4) der SVM bieten sehr viel Spielraum und müssen dem Problem entsprechend angepasst werden. Die Werte in Tabelle 6.1 bilden einen guten Kompromiss zwischen Genauigkeit und geringem *Overtraining*.

BDT Es wurden zwei verschiedene Methoden für Boosted Decision Trees verwendet. Einerseits eine Sammlung simpler Trees, mit einer maximalen Tiefe von 3 und andererseits eine erlaubte Tiefe von 15. Beide BDTs bestehen aus einem *forest* mit 300 Trees und einer minimalen Anzahl von Ereignissen pro Knotenpunkt von 43. In beiden Fällen wird *pruning* angewendet, wobei TMVA den optimalen Schnittwert selber sucht. Der beim Boosting auftretende Parameter β wird auf 0.6 gesetzt.

Mit diesen Einstellungen wird nun das Training durchgeführt. In Abbildung 6.3 sind die Verteilungen der einzelnen Methoden für die Trainings- und Testdatensätze dargestellt.

MLPs		
Option	MLP_BFGS	MLP_BP
VarTransform	Norm	Norm
TrainingMethod	BFGS	BP
Sampling	0.3	0.3
SamplingImportance	1	1
HiddenLayers	N+5	N+5
ConvergenceTests	4	6
TestRate	5	5
LearningRate	0.05	0.05
BDTs		
Option	BDT_AdaBoost	BDT_AdaBoost_Depth
VarTransform	Deco	Deco
BoostType	AdaBoost	AdaBoost
NTrees	300	300
MaxDepth	3	15
nEventsMin	43	43
PruneStrength	-1	-1
PruneMethod	CostComplexity	CostComplexity
AdaBoostBeta	0.6	0.6
SVM		
Option	SVM	
VarTranfsorm	None	
C	4	
Tol	0.05	
Gamma	0.25	
MaxIter	1000	

Tabelle 6.1: Tabelle der gebuchten Methoden mit ihren gewählten Optionen. Jeweils zwei MLPs und BDTs mit leicht unterschiedlichen Optionen und eine SVM.

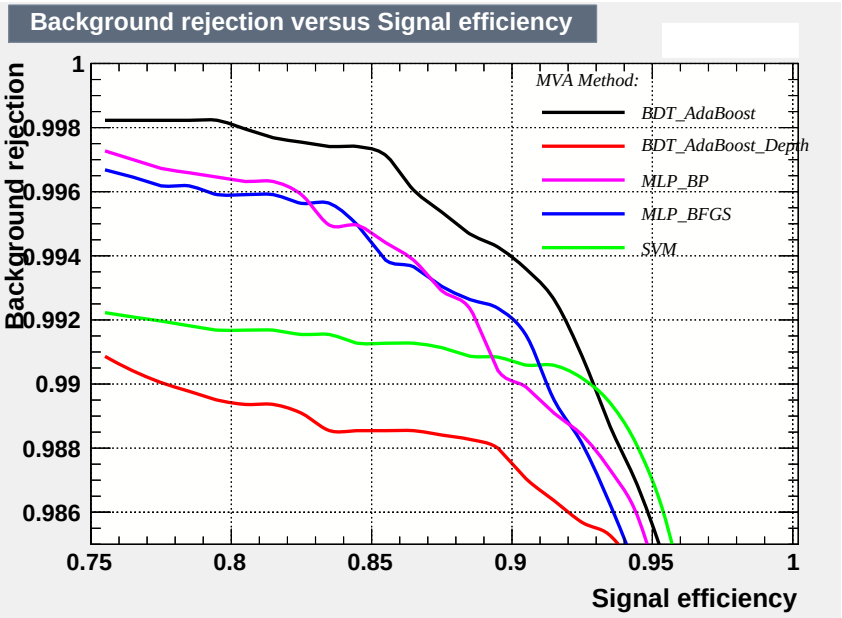


Abbildung 6.2: Untergrundunterdrückungen aller Methoden für das Trainieren mit allen Daten.

Nun werden zwei verschiedene Fälle betrachtet. Zum einen wird die Signaleffizienz und Untergrundunterdrückung für alle Methoden beim vom TMVA bestimmten optimalen Schnittwert betrachtet. Dieser optimale Schnittwert ergibt sich aus der höchsten Signifikanz, wie sie in Abschnitt 4 definiert wurde. Um die Methoden direkter mit denen aus [6] zu vergleichen, wird zusätzlich die Signaleffizienz bei einer Untergrundunterdrückung von 98,9 % betrachtet.

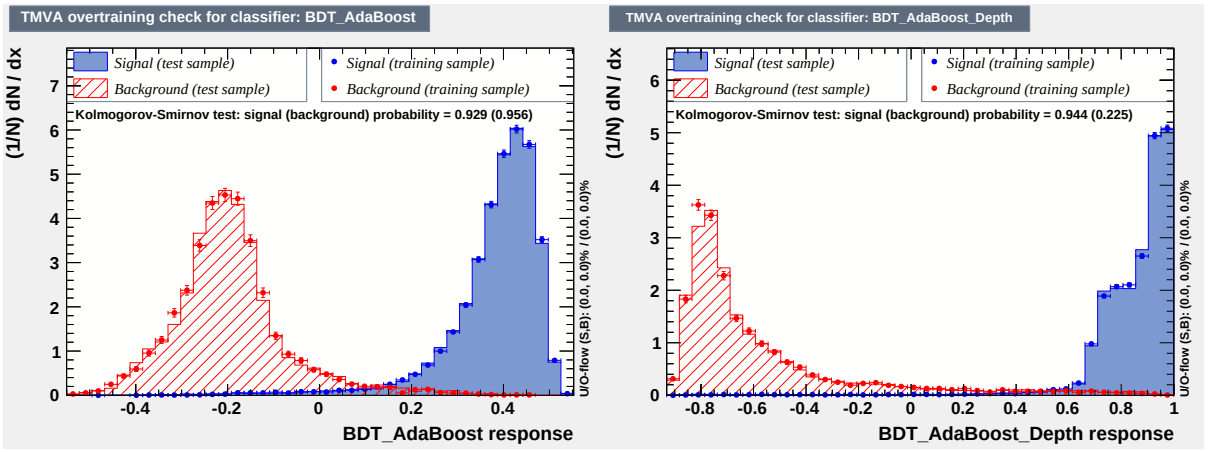
Wie anhand der Verteilungen in Abbildung 6.3 zu erkennen ist, schaffen alle Methoden eine gute Trennung zwischen Signal und Untergrund.

Weiterhin lässt sich anhand der Verteilungen das *Overtraining* abschätzen. Es ist zu erkennen, dass die neuronalen Netzwerke mit beiden Lernalgorithmen und `BDT_AdaBoost` am robustesten zu sein scheinen. Im Fall der SVM und des zweiten BDTs tritt für die Untergrunddaten leichtes *Overtraining* auf. Vergleicht man die Werte der Kolmogorov-Smirnov Tests mit den kritischen Werten für eine Signifikanz von $\alpha = 0{,}001$, stellt man fest, dass alle Methoden insgesamt geringes *Overtraining* aufweisen (siehe Abschnitt 4.2.1).

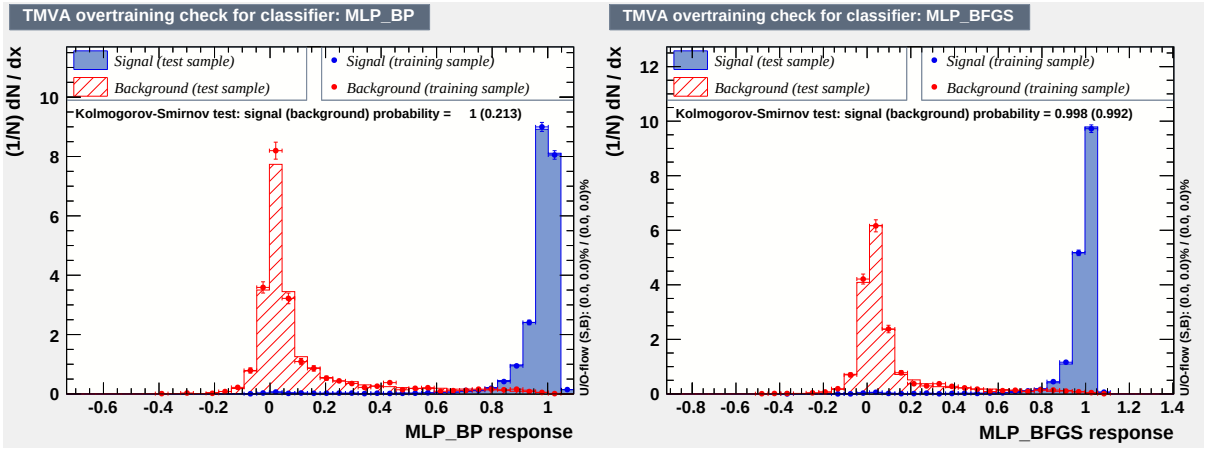
In Abbildung 6.2 ist die Untergrundunterdrückung für alle Methoden dargestellt.

Betrachtet man für die jeweiligen Methoden nun die Signaleffizienz beim optimalen Schnittwert, d.h. bei der höchsten Signifikanz (zu entnehmen aus den Graphen in Anhang A.3.1), erhält man die Werte, wie sie in Tabelle 6.2 dargestellt sind. In Verbindung mit der jeweiligen Untergrundunterdrückung arbeitet die SVM am effizientesten, da sie bei der höchsten Untergrundunterdrückung von 97,72 % noch eine Signaleffizienz von 97,09 % erreicht. Insgesamt liegen alle Methoden nah beieinander.

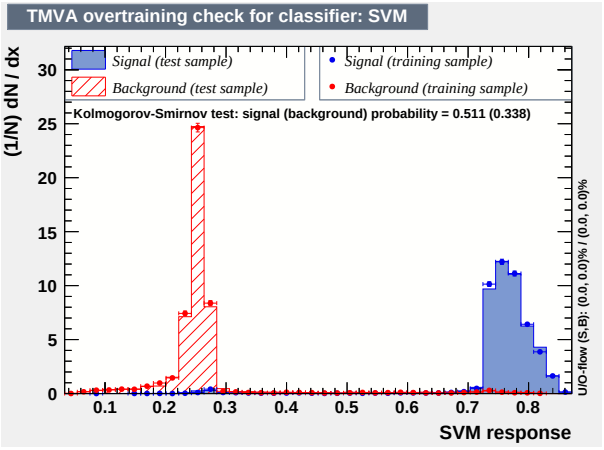
Die Signaleffizienzen für eine Untergrundunterdrückung von 98,9 % sind in Tabelle 6.3 zu finden. Dabei ist zu erkennen, dass die SVM mit 93,8 % die höchste Signaleffizienz hat, während `BDT_AdaBoost_Depth` mit 82,7 % bereits stark nachgelassen hat. Die Signaleffizienz von 95,3 % der Likelihood Methode erreicht jedoch keine der Methoden.



(a) (b)



(c) (d)



(e)

Abbildung 6.3: Ausgabe der Methoden für die Testdaten in gefüllten Verteilungen. (a) `BDT_AdaBoost` (b) `BDT_AdaBoost_Depth` (c) `MLP_BP` (d) `MLP_BFGS` (e) `SVM` Ausgabe. Blau entspricht dem Signal, Rot dem Untergrund. In Punkten sind zusätzlich die Trainingsdaten eingetragen, um das *Overtraining* zu beurteilen.

Methode	Schnittwert	Signaleffizienz	Untergrundunterdrückung
BDT_AdaBoost	0,1392	0,9686	0,9756
BDT_AdaBoost_Depth	0,5883	0,9675	0,9769
SVM	0,6477	0,9709	0,9772
MLP_BFGS	0,7836	0,9691	0,9753
MLP_BP	0,7280	0,9749	0,9704

Tabelle 6.2: Signaleffizienzen bei optimalem Schnittwert der einzelnen Methoden basierend auf allen Trainingsdaten.

Methode	Signaleffizienz
BDT_AdaBoost	0,934
BDT_AdaBoost_Depth	0,827
SVM	0,938
MLP_BFGS	0,919
MLP_BP	0,916

Tabelle 6.3: Signaleffizienzen bei einer Untergrundunterdrückung von 98,9 % aller Methoden basierend auf allen Trainingsdaten.

6.2 Trainieren mit Schnitt auf Energie

Es kann nicht verhindert werden, dass in den Signaldaten auch Untergrundereignisse enthalten sind. Aufgrund der Quelle ist bekannt, dass die meisten Ereignisse eine Energie um den Photopeak haben müssen. Damit die Methoden nicht fälschlicherweise versuchen Untergrundereignisse aus den Signaldaten als signalartig zu interpretieren, kann man versuchen diese über einen Schnitt auf die Energie herauszufiltern. In diesem Abschnitt wird ein Schnitt auf die Energie zwischen 5,5 keV bis 6 keV vorgenommen. Es werden die gleichen Methoden verwendet, wie zuvor.

Da die Verteilungen konzeptionell die gleichen sind, wie im Abschnitt zuvor, sind diese zusammen im Anhang A.3.2 zu finden. In Tabelle 6.4 sind die Signaleffizienzen beim optimalen Schnittwert für diesen Fall zu finden.

Methode	Schnittwert	Signaleffizienz	Untergrundunterdrückung
BDT_AdaBoost	0,0576	0,9952	0,9926
BDT_AdaBoost_Depth	0,3032	0,9929	0,9933
SVM	0,6255	0,9946	0,9947
MLP_BFGS	0,7223	0,9906	0,9879
MLP_BP	0,6813	0,993	0,9879

Tabelle 6.4: Signaleffizienzen bei optimalem Schnittwert der einzelnen Methoden basierend auf Daten mit Schnitt auf die Energie.

In diesem Fall zeigt erneut keine der Methoden nennenswertes *Overtraining*. Auch in diesem Fall ist die SVM mit einer Signaleffizienz von 99,46 % bei einer Untergrundunterdrückung von 99,47 % die effizienteste Methode. Es arbeiten jedoch alle Methoden sehr effizient. In Abbildung 6.4 sind die Untergrundunterdrückung aller Methoden dargestellt.

Insgesamt führt der Schnitt auf die Energie dazu, dass die Klassifikation insgesamt deutlich besser wird und nahezu alle Ereignisse richtig klassifiziert werden. Dies kann damit begründet werden, dass die Ereignisse des Photopeaks klarer die Eigenschaften der echten **XrayObjects** haben.

Zusätzlich wird erneut die Signaleffizienz bei einer Untergrundunterdrückung von 98,9 % betrachtet. Die sich ergebenen Werte sind in Tabelle 6.5 zu finden. Abgesehen von den neuronalen Netzwerken, liegen die Signaleffizienzen der Methoden bei dieser Untergrundunterdrückung 100 %. Somit übertrifft jede Methode mit dieser Art des Trainierens die Likelihood Methode.

Methode	Signaleffizienz
BDT_AdaBoost	1
BDT_AdaBoost_Depth	1
SVM	1
MLP_BFGS	0,986
MLP_BP	0,989

Tabelle 6.5: Signaleffizienzen bei einer Untergrundunterdrückung von 98,9 % aller Methoden basierend auf dem Trainieren mit Schnitt auf die Energie.

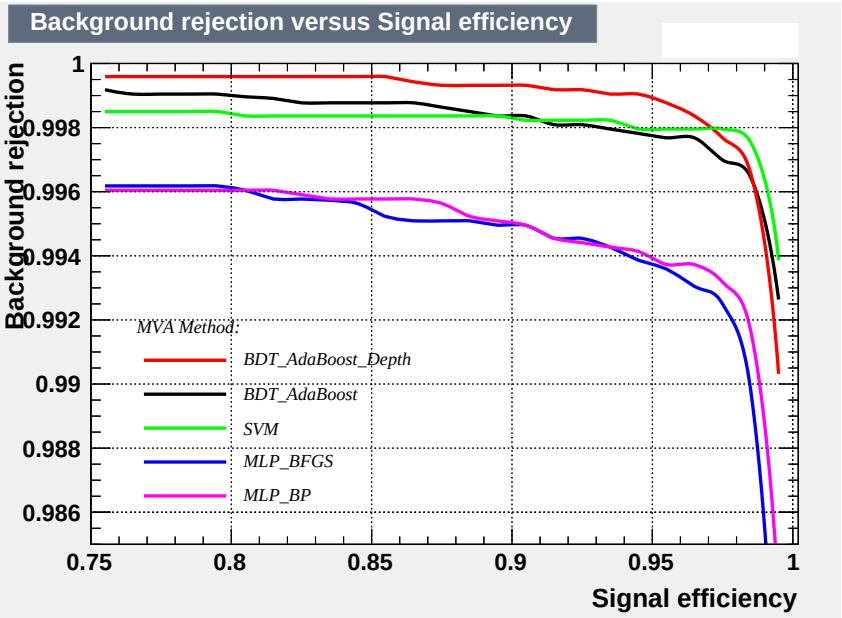


Abbildung 6.4: Untergrundunterdrückungen aller Methoden für das Trainieren mit einem Schnitt auf die Energie.

6.3 Vergleich mit Likelihood Methode aus TMVA

Der Vergleich der Signaleffizienzen für eine Untergrundunterdrückung von 98,9 %, welcher in den Abschnitten 6.1 und 6.2 durchgeführt wurde, muss kritisch betrachtet werden.

Um die Vergleichbarkeit besser einschätzen zu können, wird eine Likelihood Methode in TMVA auf Basis der gleichen Variablen trainiert, welche in [6] verwendet wurden und am Anfang von Kapitel 6 aufgeführt sind. Dies wird ohne einen Schnitt auf die Eingabedaten durchgeführt.

Für diesen Fall ist die Untergrundunterdrückung in Abbildung 6.5 zu finden. Anhand dieser Abbildung ist zu sehen, dass die Ergebnisse aus [22] nicht mit der Likelihood Methode aus TMVA reproduziert werden können. Daher ist es schwierig die Werte für Signal- und Untergrundeffizienzen der hier trainierten Methoden direkt mit bisherigen Ergebnissen zu vergleichen.

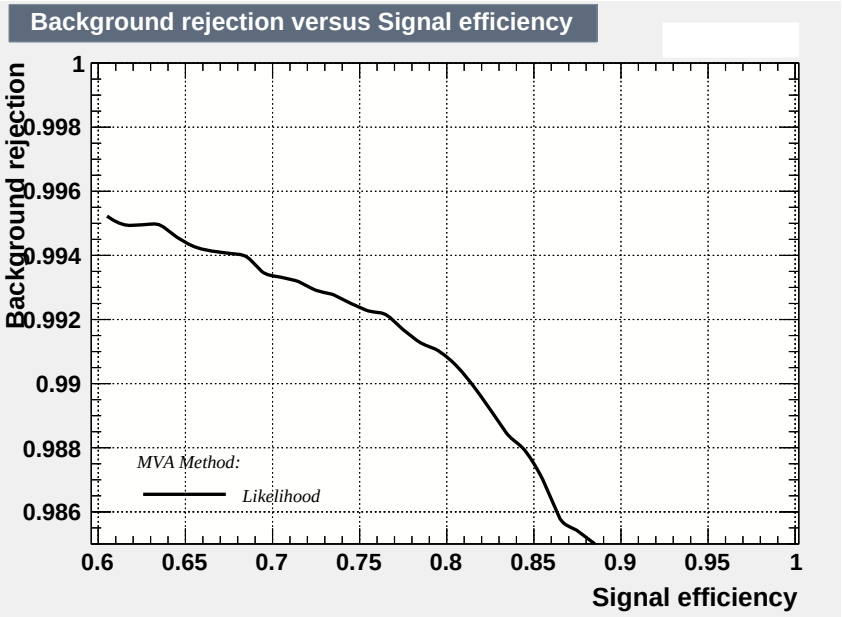


Abbildung 6.5: Vergrößerung auf den Bereich um eine Unterdrückung von 98,9 % einer Likelihood Methode ohne einen Schnitt auf die Daten.

6.4 Klassifikation der künstlich erzeugten Daten

Für die zuvor in Abschnitt 5.2 künstlich erzeugten, niederenergetischen Daten werden zunächst neue Methoden unter Verwendung dieser Daten trainiert und getestet. Im Anschluss wird untersucht, wie effizient die in den vorigen Abschnitten trainierten Methoden sind, wenn die künstlich erzeugten Daten zur Klassifikation gegeben werden.

6.4.1 Training neuer Methoden

Die Verwendung der künstlichen niederenergetischen Daten entspricht stets auch einem Schnitt auf die Energie, da für die Erzeugung der künstlichen Ereignisse nur Ereignisse aus einem Energiebereich um 5,76 keV verteilt, verwendet wurden. Es werden erneut die gleichen Methoden wie in Abschnitt 6.1 verwendet.

Die Graphen der Ausgabeverteilungen und die Schnitteffizienzen sind erneut im Anhang A.3.3 zu finden.

Anhand der Signaleffizienzen in Tabelle 6.6 ist zu erkennen, dass in diesem Fall alle Methoden eine extrem gute Trennung erreichen. Das künstliche neuronale Netzwerk `MLP_BFGS` erreicht mit einer Effizienz von 99,69 % bei einem Schnittwert von 0,6522 die höchste Effizienz. Unter Berücksichtigung der Untergrundunterdrückung erzielt erneut die SVM die besten Ergebnisse mit einer Signaleffizienz von 99,57 % bei einer Untergrundunterdrückung von 99,56 %. *Overtraining* spielt nach dem Kolmogorov-Smirnov Test erneut bei einer Signifikanz $\alpha = 0.001$ keine Rolle.

In Abbildung 6.6 sind die Untergrundunterdrückungen aller Methoden dargestellt. Es ist zu erkennen, dass alle Methoden selbst bei sehr hohen Untergrundunterdrückungen von über 99 % noch Signaleffizienzen von mehr als 99 % erreichen. Die hohen Effizienzen sind darauf zurückzuführen, dass für die Erzeugung des Datensatzes ein Schnitt auf die Energie vorgenommen wurde. Daher müssen die Signaleffizienzen mit denen des Trainierens mit einem Schnitt auf die Energie verglichen werden. Es kann geschlussfolgert werden, dass das Trainieren für geringere Energien nicht schlechter wird, sondern die geringere Energie dazu führt, dass die Verteilungen der Variablen zwischen Signal und Untergrund stärker getrennt wird.

Methode	Schnittwert	Signaleffizienz	Untergrundunterdrückung
`BDT_AdaBoost`	0,0407	0,9962	0,9945
`BDT_AdaBoost_Depth`	0,3562	0,9926	0,9956
`SVM`	0,6715	0,9957	0,9956
`MLP_BFGS`	0,6522	0,9969	0,9926
`MLP_BP`	0,6848	0,9958	0,9925

Tabelle 6.6: Signaleffizienzen bei optimalem Schnittwert der einzelnen Methoden basierend auf den künstlichen Daten.

6.4.2 Klassifikation mittels vorhandener Methoden

Eine andere Fragestellung ist, inwieweit es mit Methoden, welche für höhere Energien trainiert wurde, möglich ist die niederenergetischen Ereignisse zu klassifizieren. Dies in von Relevanz, da die Methoden natürlich zuverlässiger mit den Daten der ^{55}Fe Quelle trainiert werden können, als mit den künstlichen

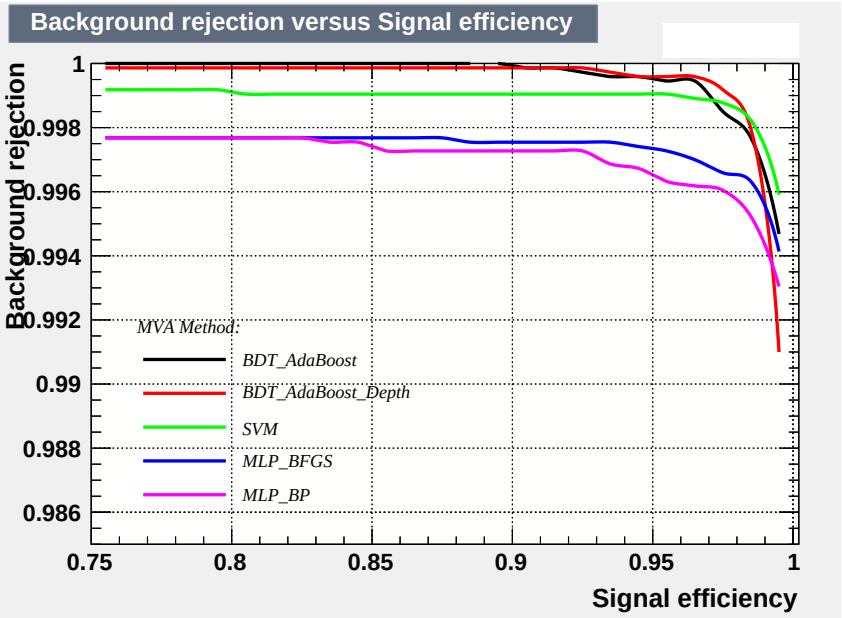


Abbildung 6.6: Untergrundsunterdrückung aller Methoden für das Trainieren mit den künstlich erzeugten 1 keV Daten.

Daten. Um also die Effizienz der Methoden für hypothetisches Ereignisse im niederenergetischen Bereich abschätzen zu können, werden die erzeugten Daten über die **Reader** Klasse klassifiziert und es wird über den optimalen Schnittwert die Signaleffizienz berechnet.

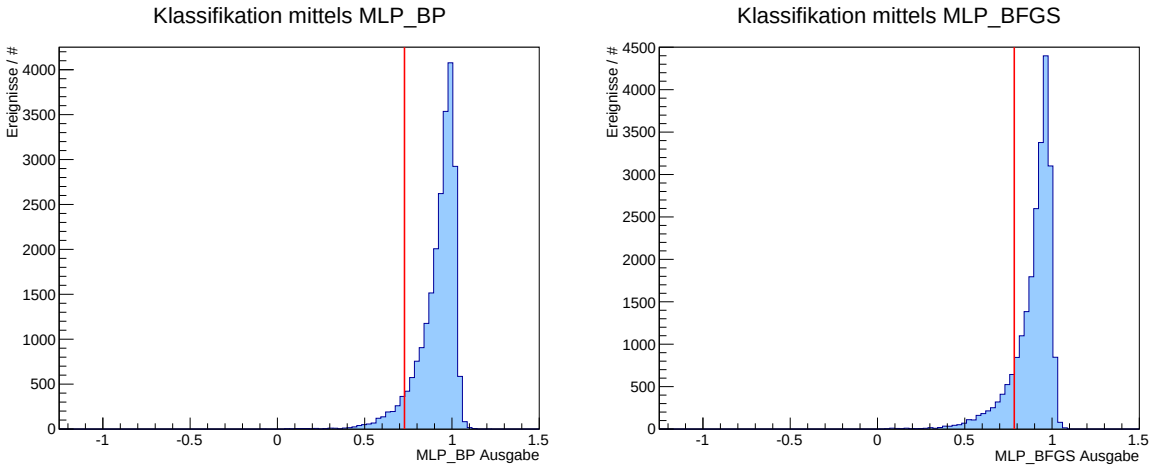
Klassifiziert man die 1 keV Daten mit den Methoden, welche in Abschnitt 6.1 trainiert wurden, ergeben sich die Klassifikationen, welche in Abbildung 6.7 dargestellt sind. Die sich ergebenden Effizienzen sind in Tabelle 6.7 zu finden. Da die gleichen Schnittwerte verwendet werden, bleibt die Untergrundunterdrückung die Gleiche, wie in Abschnitt 6.1.

Die Methoden unterscheiden sich stark in der Signaleffizienz. Während das neuronale Netzwerk `MLP_BFGS` nur eine Signaleffizienz von 85,87 % erreicht, ist die SVM mit 99,44 % hier sogar effizienter als beim gesamten Datensatz.

Methode	Schnittwert	Signaleffizienz	Untergrundunterdrückung
`BDT_AdaBoost`	0,1392	0,9622	0,9756
`BDT_AdaBoost_Depth`	0,5883	0,9261	0,9769
`SVM`	0,6477	0,9944	0,9772
`MLP_BFGS`	0,7836	0,8587	0,9753
`MLP_BP`	0,7280	0,9314	0,9704

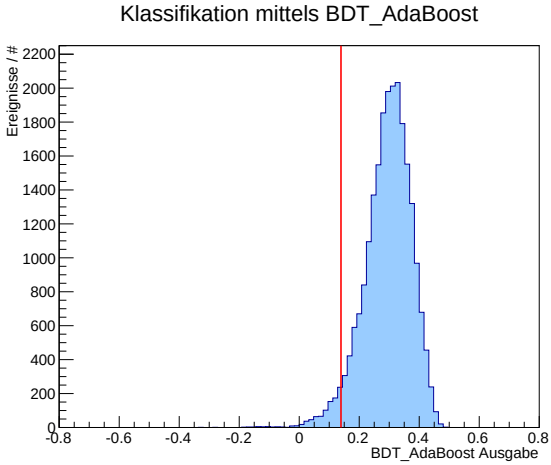
Tabelle 6.7: Signaleffizienzen der Klassifikation der 1 keV Daten mit den für alle Daten bestimmten Methoden, beim zuvor bestimmten optimalem Schnittwert. Die Untergrundunterdrückungen sind unverändert, da der Schnittwert gleich geblieben ist.

Darüber hinaus stellt sich die Frage, wie sich die Methoden allgemein als Funktion der Energie der zu klassifizierenden Daten verhalten. Deshalb wurden aus dem Photopeak des Gesamtdatensatzes weitere Datensätze erzeugt. Und zwar werden die Energien 2 keV, 1,5 keV, 1,25 keV, 1 keV, 0,75 keV und 0,5 keV betrachtet. Zusätzlich wird der Escapepeak einzeln mit den gleichen Methoden klassifiziert.

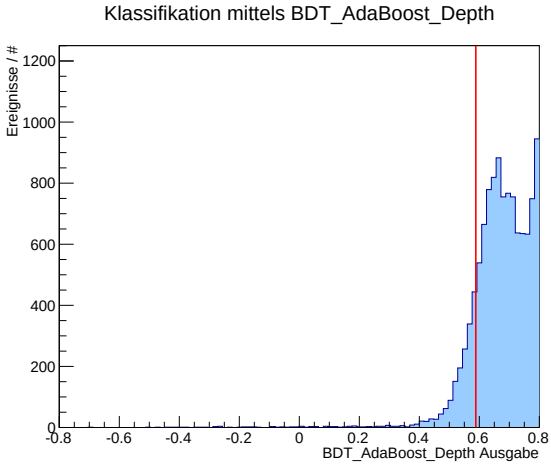


(a)

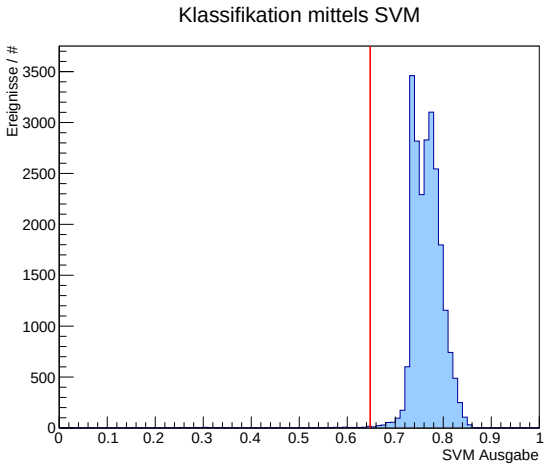
(b)



(c)



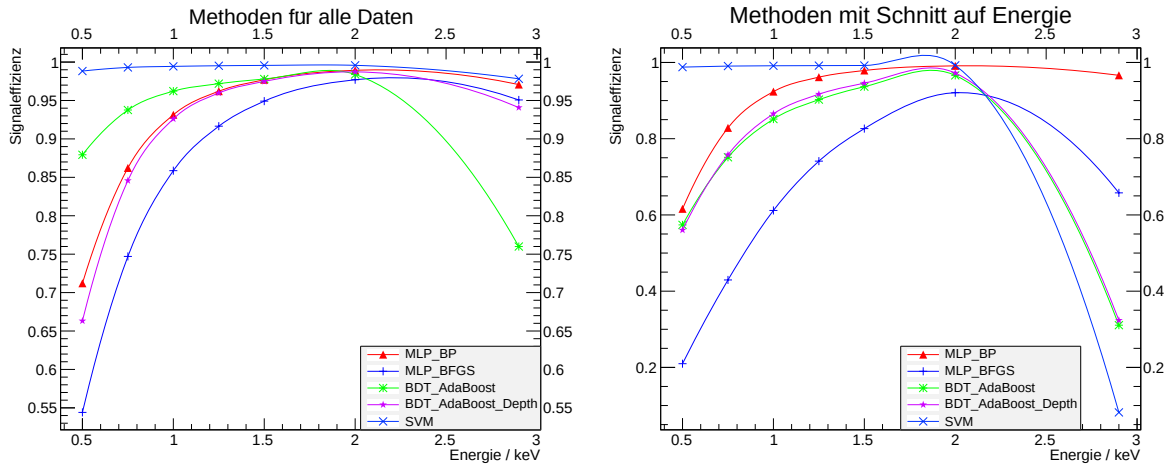
(d)



(e)

Abbildung 6.7: Klassifizierungen des künstlichen 1 keV Datensatzes mit den Methoden, welche in Abschnitt 6.1 entworfen wurden. (a) `MLP_BP` (b) `MLP_BFGS` (c) `BDT_AdaBoost` (d) `BDT_AdaBoost_Depth` (e) `SVM` Ausgabe. In Rot ist der optimale Schnitt aus Tabelle 6.2 eingetragen.

Einmal werden die Methoden für die gesamten Daten und einmal für die Daten mit einem Schnitt auf die Energie verwendet. Alle Klassifikationen werden erneut mit den Schnittwerten aus Tabelle 6.2 und 6.4 ermittelt. Die Ergebnisse sind in Abbildung 6.8 zu finden.



(a) (b)

Abbildung 6.8: (a) Signaleffizienzen beim optimalen Schnittwert aus Tabelle 6.2 für künstliche Datensätze zwischen 0,5 und 2 keV mittels der Methoden für den gesamten Datensatz. (b) Signaleffizienzen beim optimalen Schnittwert aus Tabelle 6.4 mittels der Methoden für die Daten mit einem Schnitt auf die Enerige. Zusätzlich ist in (a) und (b) bei 2,9 keV der Escapepeak eingezeichnet.

Für den in Abbildung 6.8 (a) dargestellten Fall verhalten sich die Methoden stark unterschiedlich. Die Methode `MLP_BFGS` verhält sich für niedrige Energien am schlechtesten. `BDT_AdaBoost_Depth` und `MLP_BP` bleiben etwas effizienter. Die SVM ist die mit Abstand konstanteste Methode. Bei ihr ist nahezu keine Abhängigkeit der Energie festzustellen und sie hat überall eine Signaleffizienz von oberhalb 97 %. Beim Betrachten des Escapepeaks fällt weiterhin auf, dass die Methoden trotz höherer Energie alle ineffizienter sind. Die Methode `BDT_AdaBoost` wird sehr ineffizient. Die Abweichungen liegen an den unterschiedlichen Eigenschaften der Objekte des Escapepeaks und den, aus dem Photopeak erzeugten künstlichen Daten.

Im Fall der Methoden mit Schnitt auf die Energie, in Abbildung 6.8 (b) dargestellt, sehen die Abhängigkeiten der Signaleffizienz von der Energie für die künstlichen Daten zwischen 0,5 keV und 2 keV ähnlich aus. Die Methode `MLP_BP` arbeitet in diesem Bereich etwas besser, `BDT_AdaBoost` etwas schlechter. Für den Escapepeak sind abgesehen von `MLP_BP` jedoch alle Methoden deutlich ineffizienter. Allen voran fällt die SVM auf, welche hier nahezu keine Signalereignisse oberhalb des Schnittwerts klassifizert.

KAPITEL 7

Betrachtung falsch klassifizierter Ereignisse

In diesem Kapitel wird versucht die falsch klassifizierten Ereignisse der verschiedenen Methoden zu verstehen. Für sehr hohe Untergrundunterdrückungen wird erwartet, dass ein gewisser Anteil der Ereignisse falsch klassifiziert werden muss. Einerseits, weil die signalartigen Trainingsdaten der ^{55}Fe Quelle zu geringem Anteil auch Untergrundereignisse enthalten können. Interessanter aber andererseits, weil manche Untergrundereignisse den Detektor von vorne nach hinten durchqueren, sodass ihre Spur senkrecht auf dem Chip des Detektors steht. Dies führt dazu, dass sie in erster Näherung aussehen, wie durch Photonen erzeugten Ereignisse. Erst bei genauerer Betrachtung müssen Unterschiede deutlich werden, die durch die andere Energiedeposition von Photon und beispielsweise Myon auftreten. Dies könnte sich anhand der *Skewness* oder *Kurtosis* der Ereignisse zeigen. Der zweite Fall ist skizzenhaft in Abbildung 7.1 dargestellt.

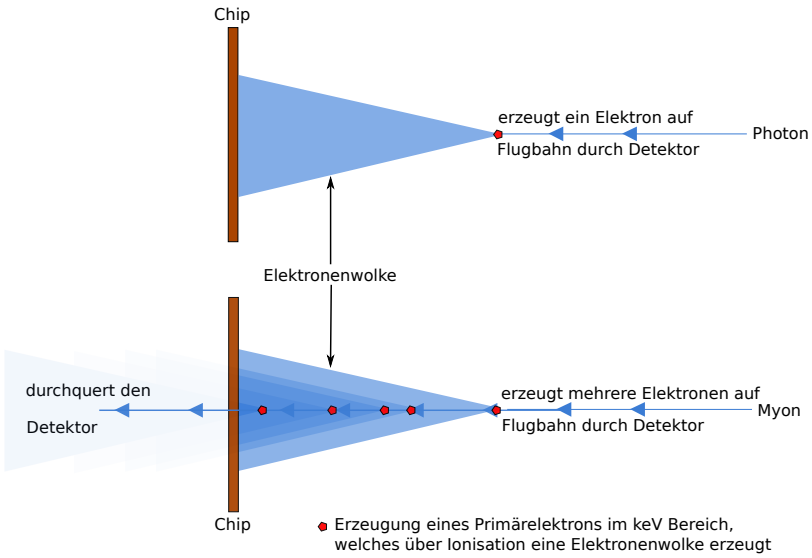


Abbildung 7.1: Skizzenhafte Darstellung der Entstehung und Diffusion der Elektronen im Detektor für Photonen und Myonen. Das Photon erzeugt ein Primärelektron, welches über Ionisation eine Elektronenwolke erzeugt. Diese driftet zum Chip. Das Myon ionisiert auf seiner Flugbahn viele Elektronen, welche eine Elektronenwolke bilden. Aufgrund gleicher Diffusion aller Elektronenwolken, aber unterschiedlichem Entstehungspunkt, ergibt sich im Zentrum eine höhere Ladungsdichte

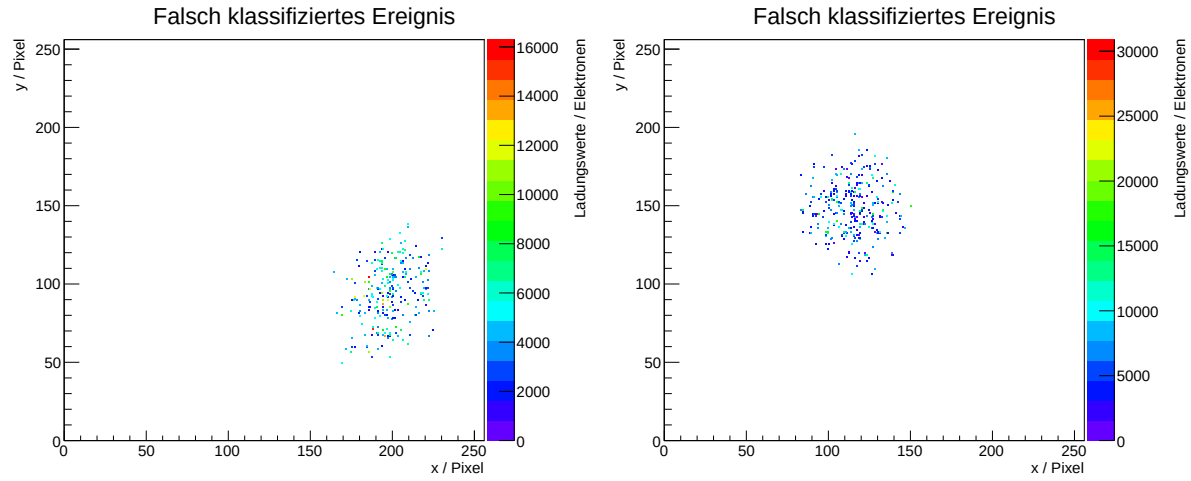


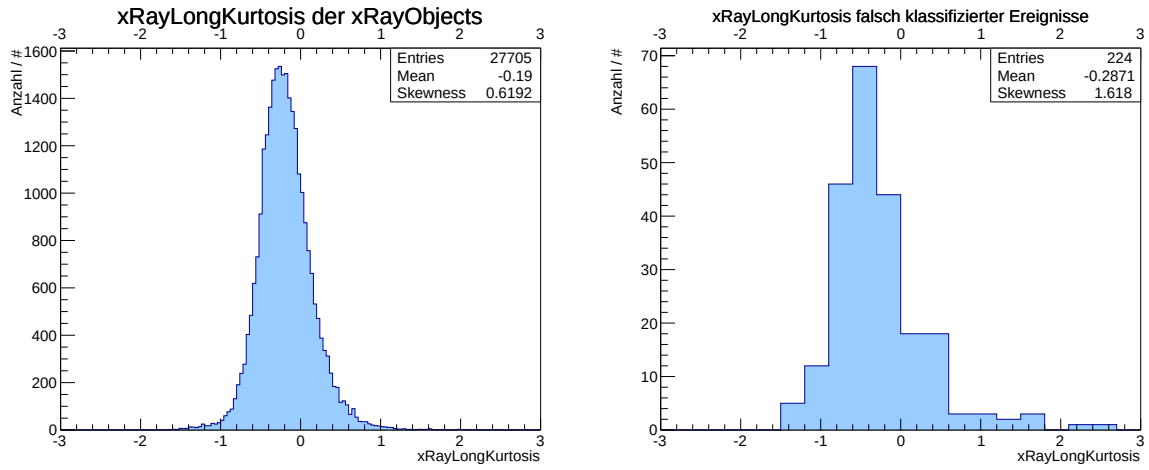
Abbildung 7.2: Typische Beispiele von Untergrundereignissen, die von der `MLP_BP` Methode mit einem Wert oberhalb des Schnittwerts bei 0,728 klassifiziert wurden.

Zunächst werden aus den klassifizierten Ereignissen, jene herausgefiltert, welche zum Untergrunddatensatz gehören, aber einen Wert der Klassifikation im Bereich des Signals haben. TMVA speichert die Informationen über den Datensatz und den Ausgabewert der Methoden für jede Variable im Tree ab.

Dies wird für die `MLP_BP` Methode aus Abschnitt 6.1 betrachtet. Und zwar werden alle Untergrundereignisse, welche einen Ausgabewert größer als den optimalen Schnittwert von 0,728 haben, herausgefiltert. In Abbildung 7.2 sind zwei Beispiele typischer Ereignisse aus diesem Datensatz zu erkennen.

Nun wird von diesen, insgesamt 208 Ereignissen, die *Skewness* und *Kurtosis* betrachtet. Zum Vergleich wird jeweils die gleiche Verteilung aller Signaldaten betrachtet, die auch einen Ausgabewert $> 0{,}728$ haben. Die Histogramme sind im Anhang A.4 zu finden. Während die *Skewness* für beide Fälle ähnlich ist, fällt vor allem für die longitudinale *Kurtosis* auf, dass die falsch klassifizierten Ereignisse rechts einen stärkeren Auslauf haben. Dies ist daran zu erkennen, dass die *Skewness* der Verteilung falsch klassifizierter Ereignisse größer als die der Photopeak Ereignisse ist und beide größer null sind. Eine *Skewness* größer null steht für eine Verteilung mit stärkerem Auslauf auf der rechten Seite. Diese Verteilung ist zusätzlich in Abbildung 7.3 zu finden. Daraus folgt, dass es tendenziell mehr falsche Ereignisse mit einer größeren *Kurtosis* gibt, als für den Photopeak. Da eine hohe *Kurtosis* für eine spitzere Verteilung steht, lässt sich schlussfolgern, dass eine größere Zahl der falsch klassifizierten Ereignisse ihre Energie auf einen kleineren Bereich deponieren. Dies kann damit erklärt werden, dass ein hochenergetisches Teilchen auf seinem Weg zum Detektor viele Atome ionisiert, welche alle einzeln diffundieren. Im Falle eines Photons geht dies von einem einzelnen Elektron aus, wodurch eine stärkere Diffusion auftritt.

Aufgrund der geringen Datenmenge und dem relativ kleinen Unterschied der *Skewness*-Werte kann dies nur als Hinweis gesehen werden, dass die Unterschiede zwischen frontalen Untergrundereignissen und Photonen erkennbar sind.



(a) (b)

Abbildung 7.3: Longitudinale *Kurtosis* für die **XrayObjects** in (a) und die falsch klassifizierten Untergrundereignisse in (b). Die Verteilung der Untergrundereignisse hat eine höhere *Skewness* und damit stärkeren Auslauf nach rechts.

KAPITEL 8

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es die Untergrundunterdrückung für einen neuen Röntgendetektor des CAST Experiments, welcher an der Universität Bonn in Entwicklung ist, zu verbessern. Dies wurde, im Vergleich zur bisher verwendeten Likelihood Methode, über multivariate Verfahren mittels TMVA durchgeführt. Die Werte werden dabei mit der Signaleffizienz von 95,3 % bei einer Untergrundunterdrückung von 98,9 % der Likelihood Methode vergleichen. Das zweite Ziel der Arbeit war die Effizienz des Detektors für geringe Energien zu simulieren.

Zunächst wurden künstliche neuronale Netzwerke, Boosted Decision Trees und eine Support Vector Machine mit allen vorhandenen Daten und Eigenschaften der Ereignisse trainiert, getestet und evaluiert. Bereits mit der einfachsten Trainingsmethode sind die Methoden sehr effizient. Die SVM erreicht eine Signaleffizienz von 93,8 % bei der gleichen Untergrundunterdrückung, wie die Likelihood Methode. Anschließend wurden die gleichen Methoden erneut, mit einem Schnitt auf Energie der Signalereignisse, trainiert. In diesem Fall sind alle Methoden effizienter als die Likelihood Methode. Bei der Untergrundunterdrückung von 98,9 % erreichen die beiden BDTs sowie die SVM eine Signaleffizienz von 100 %. Im nächsten Schritt wurde versucht abzuschätzen wie gut der Vergleich zwischen den zuvor trainierten Methoden und der bisherigen Likelihood Methode möglich ist. Dazu wurde eine Likelihood Methode mit TMVA bei gleichen Bedingungen (gleiche Variablen und keine Schnitte auf die Daten) wie bisher trainiert. Dabei musste festgestellt werden, dass die Likelihood Methode in TMVA eine deutlich schlechtere Effizienz bei gleicher Untergrundunterdrückung wie bisher aufweist. Dies erschwert den Vergleich der Methoden.

Insgesamt konnten die bisherigen Ergebnisse der Likelihood Methode reproduziert und sogar übertroffen werden.

Um die Effizienz des Detektors für niedrigere Energien abschätzen zu können, wurden aufgrund nicht vorhandener echter Daten, niederenergetische Daten simuliert. Mit künstlich erzeugten 1 keV Daten wurden neue Methoden trainiert, welche alle eine extrem hohe Effizienz aufweisen. Die SVM erreicht eine Signaleffizienz von 99,57 % bei einer Untergrundunterdrückung von 99,56 %. Mit weiteren Datensätzen wurde die Effizienz des Detektors in Abhängigkeit der Energie simuliert. Einmal mit den zuvor für alle Daten trainierten Methoden und für die Methoden mit einem Schnitt auf die Signaldaten. Die Support Vector Machine ist in beiden Fällen für die künstlich erzeugten Daten bis herunter zu 0,5 keV signaleffizienter als 98,5 %. Wie zu erwarten, funktioniert die Klassifikaton des Escapepeaks

mit den Methoden, welche mit Schnitt auf den Photopeak erstellt wurden, deutlich schlechter. Einzig das neuronale Netzwerk, welches per *backpropagation* trainiert wurde, erreicht hier eine nennenswerte Signaleffizienz von 96,6 %. Diese Ergebnisse lassen darauf schließen, dass der neue Detektor auch im Energiebereich bis 1 keV effizient arbeiten wird.

Zuletzt wurde versucht abzuschätzen, ob es prinzipiell möglich ist, signalartige Untergrundereignisse, wie Myonen welche frontal durch den Chip des Detektors fliegen, von Photonen zu unterscheiden. Dazu wurden falsch klassifizierte Untergrundereignisse der `MLP_BP` Methode betrachtet und auf ihre *Skewness* und *Kurtosis* untersucht. Es konnte bestätigt werden, dass zumindest eine Mehrzahl der falsch klassifizierten Ereignisse eine größere longitudinale Kurtosis besitzt, als Photonen. Dies wurde aufgrund unterschiedlicher Ladungsdeposition erwartet.

Zusammenfassend können die Ergebnisse alle als sehr erfolgreich betrachtet werden. Dennoch könnten noch einige Maßnahmen getroffen werden, um diese noch zu verbessern.

Bezüglich des Trainierens der Methoden sollte der Einfluss von Korrelationen der Variablen in Hinblick auf neuronale Netzwerke untersucht werden. Außerdem könnte versucht werden, die Seperationsfähigkeit vieler Variablen zu verbessern, z.B. indem man sie quadriert. Dadurch könnte die Effizienz der Methoden unter Umständen stark gesteigert werden. Eine weiterer möglicher Ansatzpunkt wäre das sogenannte **Category**-Objekt aus TMVA, welches es ermöglicht für verschiedene Teilmengen eines Datensatzes unterschiedliche Methoden zu trainieren. Vielleicht kann die Effizienz der `MLP_BP` Methode für den Escapepeak mit der Signaleffizienz für die künstlichen Daten der SVM kombiniert werden.

Um den Unterschied zwischen signalartigen Untergrundereignissen und Photonen genauer zu untersuchen, bietet es sich an eine Messreihe mit einem Betastrahler aufzunehmen, um die Eigenschaften hochenergetischer Teilchen im Detektor besser zu verstehen.

ANHANG A

Anhang

A.1 Quellcode

Der folgende Quellcode ist der, der für die Klassifikation aller Daten verwendet wurde. Er ist ausführlich kommentiert, um den Ablauf und die Funktionsweise von TMVA klar zu machen.

```
void TMVAXray_Klass_All( )
{

  // Definiert den Name der .root Datei, in den die Ergebnisse der Methoden
      geschrieben wird
  TString outfileName( "/home/basti/data/background/TMVA/TMVAXray_Klass_All.root"
       );
  // Definiert, dass die Datei erstellt wird, wenn sie nicht vorhanden ist, und u
      berschrieben wird,
  // wenn sie vorhanden ist.
  TFile* outputFile = TFile::Open( outfileName, "RECREATE" );

  // Zunachst wird ein Pointerarray auf Factory Objekte erstellt. Die
      verschiedenen Methoden laufen nicht
  // alle unter einer Methode, da aufgrund der unterschiedlichen Komplexitat
      unterschiedlich viele
  // Ereignisse zum Trainieren verwendet werden konnen.
  TMVA::Factory *factory[4];
  // Fur die verschiedenen Methoden werden unterschiedliche Transformationen
      gebucht.
  // I: Identitat, N: Normierung, D: Dekorrelation
  // I gibt die Daten unverandert an die Factory uber
  // N normiert die verschiedenen Variablen zwischen -1 und 1. Fur Methoden wie
      MLP notwendig, da
  // Werte der Variablen direkten Einflus auf Gewichtung haben
  // D fuhrt Rotation der Daten durch, um Korrelationen zwischen Variablen zu
      minimieren
  factory[0] = new TMVA::Factory( "MVXrayAnalysis", outputFile,"!V:
      Transformations=I;N;D");
  factory[1] = new TMVA::Factory( "MVXrayAnalysis", outputFile,"!V:
      Transformations=I");
  factory[2] = new TMVA::Factory( "MVXrayAnalysis", outputFile,"!V:
      Transformations=I;N");
```

```
factory[3] = new TMVA::Factory( "MVXrayAnalysis", outputFile,"!V:
    Transformations=I;N");

// Liest die .root Dateien ein, welche die Trees mit den Signal- und
    Untergrunddaten enthalten.
TFile *inputS = TFile::Open("/home/basti/data/background/lciodata/
    all_fe55_350V_cr/ana/TMVATree.root");
TFile *inputB = TFile::Open("/home/basti/data/background/lciodata/all_lead/ana/
    TMVATree.root");

// Variablen werden fur alle Factory Objekte hinzugefugt. Im ersten String muss
     ein Variablen-Name stehen,
// der auch in beiden Eingabe-Trees zu finden ist. Zweiter String 'I' oder 'F'
    gibt den Datentyp der
// Variable an. I: Integer, F: Float (Double mit eingeschlossen, aber: Reader
    Objekt kann nur mit
// Float Variablen umgehen!)
for (int i=0; i<4; i++){
  // XRay Variablen
  factory[i]->AddVariable("xRayRmsX", 'F');
  factory[i]->AddVariable("xRayRmsY", 'F');
  factory[i]->AddVariable("xRayExcentricity", 'F');
  factory[i]->AddVariable("xRayLongSkewness", 'F');
  factory[i]->AddVariable("xRayTransSkewness", 'F');
  factory[i]->AddVariable("xRayLongKurtosis", 'F');
  factory[i]->AddVariable("xRayTransKurtosis", 'F');
  factory[i]->AddVariable("xRayRadius", 'F');
  factory[i]->AddVariable("xRayLength", 'F');
  factory[i]->AddVariable("xRayWidth", 'F');

  // Track Variablen
  factory[i]->AddVariable("trackLongRms", 'F');
  factory[i]->AddVariable("trackTransRms", 'F');
  factory[i]->AddVariable("trackExcentricity", 'F');
  factory[i]->AddVariable("trackLongSkewness", 'F');
  factory[i]->AddVariable("trackTransSkewness", 'F');
  factory[i]->AddVariable("trackLongKurtosis", 'F');
  factory[i]->AddVariable("trackTransKurtosis", 'F');
  factory[i]->AddVariable("trackLength", 'F');
  factory[i]->AddVariable("trackWidth", 'F');
  factory[i]->AddVariable("trackPixPerLength := trackNpx / trackLength", 'F');
      // Pixels per Track length

  // Zuschauer Variablen. Diese werden nicht zum Training und der
      Klassifikation verwendet, konnen aber
  // zur Seperation der Eingabedaten verwendet werden (Bsp: Schnitt auf die
      Energie). Im Allgemeinen zur
  // Erkennung spezieller Ereignisse sinnvoll eventNumber und runNumber
      mitzugeben.
  factory[i]->AddSpectator("eventNumber", 'I');
  factory[i]->AddSpectator("runNumber", 'I');
  factory[i]->AddSpectator("xRayEnergy", 'F');
  factory[i]->AddSpectator("xRayNpx", 'F');
  factory[i]->AddSpectator("xRayCharge", 'F');

  // Hier werden die zuvor gelesenen Trees dem Factory Objekt ubergeben. Alle
      Variablen-Namen
```

```
    // mussen in beiden Trees enthalten sein.
    factory[i]->AddSignalTree ( (TTree*)inputS->Get("T") );
    factory[i]->AddBackgroundTree ( (TTree*)inputB->Get("T") );
}

// Ein TCut Objekt aus ROOT (String basierend auf mitgegebenen Variablen,
    boolschen Operatoren und
// Ausdrucken) kann definiert werden, um auf die Eingabedaten zu schneiden.
// TCut myCut = "xRayEnergy > 5.5 && xRayEnergy < 6";

// Hier werden dem Factory Objekt die Eingabedaten ubergeben und angegeben, ob
    auf die Daten geschnitten wird.
// Erster String: Schnitt auf Signaldaten (kann TCut Objekt sein)
// Zweiter String: Schnitt auf Untergrunddaten (kann TCut Objekt sein)
// Dritter String: Optionen, die mitgegeben werden:
// nTrain_Signal, nTrain_Background, nTest_Signal, nTest_Background: jeweils
    prozentualer Anteil, der von den
// verfugbaren Signal oder Untergrunddaten zum Testen und Trainieren verwendet
    werden soll. Wird einer der
// Parameter auf 0 gesetzt, wird 50% der Daten verwendet. SplitMode=Random
    erzwingt die zufallige Wahl von
// Ereignissen fur die Datensatze, SplitSeed=0 sorgt dafur, dass jedes Mal
    andere Ereignisse gewahlt werden.
factory[0]->PrepareTrainingAndTestTree( "", "",
                                        "nTrain_Signal=0:nTrain_Background=0:
                                            nTest_Signal=0:nTest_Background=0:
                                            SplitMode=Random:SplitSeed=0:!V" );

factory[1]->PrepareTrainingAndTestTree( "", "",
                                          "nTrain_Signal=10000:nTrain_Background
                                              =0:nTest_Signal=0:nTest_Background
                                              =0:SplitMode=Random:SplitSeed=0:!V"
                                              );
factory[2]->PrepareTrainingAndTestTree( "", "",
                                        "nTrain_Signal=0:nTrain_Background=0:
                                            nTest_Signal=0:nTest_Background=0:
                                            SplitMode=Random:SplitSeed=0:!V" );

factory[3]->PrepareTrainingAndTestTree( "", "",
                                        "nTrain_Signal=0:nTrain_Background=0:
                                            nTest_Signal=0:nTest_Background=0:
                                            SplitMode=Random:SplitSeed=0:!V" );


// Nun folgt das Buchen der eigentlichen Methoden.
// Zunachst wird eine TMVA interne Bezeichnung fur eine der Methoden mitgegeben
    , z.B. kBDT fur einen Boosted
// Decision Tree.
// Erster String: benutzerdefinierte, eindeutige Bezeichnung der Methode.
// Zweiter String: Methodenspezifische Optionen.

// Boosted Decision Tree mit Namen BDT_AdaBoost wird gebucht.
// Boosting Methode: AdaBoost. Bestehend aus 300 Trees, maximale Tiefe 3,
    mindestens 43 Ereignisse in jedem Knotenpunkt.
// PruneStrength=-1 gibt an, dass TMVA die optimale Starke des Pruning, also
    Beschneiden der Trees,
// berechnen soll. Das Pruning geschieht uber die Methode CostComplexity.
```

```
factory[0]->BookMethod( TMVA::Types::kBDT, "BDT_AdaBoost",
                        "H:!V:BoostType=AdaBoost:AdaBoostBeta=0.6:NTrees=300:
                            MaxDepth=3:nEventsMin=43:PruneStrength=-1:PruneMethod=
                            CostComplexity" );

// Von der Tree Tiefe abgesehen gleiche Methode wie BDT_AdaBoost. Komplexerer
    Tree Aufbau (maximale Tiefe 15)
// neigt zu starkerem Overtraining. Mittels Pruning wird versucht dies zu
    minimieren.
factory[0]->BookMethod( TMVA::Types::kBDT, "BDT_AdaBoost_Depth",
                        "H:!V:BoostType=AdaBoost:NTrees=300:MaxDepth=10:nEventsMin
                            =43:PruneStrength=-1:PruneMethod=CostComplexity" );

// Support Vector Machine. Gebucht ohne Transformation auf Eingabedaten.
    CostParameter=2.5, Toleranz bei 0.001
// und der Parameter, der mit der Breite des Gaus-Kernels zusammenhangt Gamma
    =0.005.
factory[1]->BookMethod( TMVA::Types::kSVM, "SVM",
                         "H:!V:VarTransform=None:C=2.5:Tol=0.001:Gamma=0.005");

// Kunstliche neuronale Netzwerke: Variablen werden auf -1 bis 1 normiert, um
    gleiche Gewichtung der Variablen
// zu gewahrleisten. Sampling=0.05, es werden also 30% der Trainingsdaten zum
    Trainieren verwendet, diese werden aufgrund
// SamplingImportance=1 zufallig ausgewahlt. Auf einer Zwischenebene haben
    beide Netzwerke N+5 Neuronen. Alle 5 Durchlaufe
// werden 4, bzw. 6 Konvergenztests durchgefuhrt. Die LearningRate betragt
    0.05.

// MLP_BFGS verwendet das BFGS-Verfahren zum Trainineren.
factory[2]->BookMethod( TMVA::Types::kMLP, "MLP_BFGS",
             "H:!V:VarTransform=Norm:TrainingMethod=BFGS:Sampling=0.3:
                 SamplingImportance=1:HiddenLayers=N+5:TestRate=5:
                 ConvergenceTests=4:LearningRate=0.05" );

// MLP_BP verwendet das Backpropagation-Verfahren zum Trainieren.
factory[3]->BookMethod( TMVA::Types::kMLP, "MLP_BP",
             "H:!V:VarTransform=Norm:TrainingMethod=BP:Sampling=0.3:
                 SamplingImportance=1:HiddenLayers=N+5:TestRate=5:
                 ConvergenceTests=4:LearningRate=0.05" );


// Fur alle Methoden wird nacheinander zunachst das Trainieren aufgerufen,
    anschliesend das Testen und zuletzt das
// Evaluieren der Methoden.
for(int i=0; i<4; i++){
  factory[i]->TrainAllMethods();
  factory[i]->TestAllMethods();
  factory[i]->EvaluateAllMethods();
}

// Die .root Datei, in welche geschrieben wird, wird geschlossen.
outputFile->Close();

// Die Factory Objekte werden geloscht.
delete factory[0];
delete factory[1];
```

```
  delete factory[2];
  delete factory[3];

  // Startet die graphische Oberflache, welche das Aufrufen vieler Macros ermo
      glicht und ladt die
  // Datei, in welche soeben geschrieben wurde.
  gROOT->LoadMacro("$ROOTSYS/tmva/test/TMVAGui.C");
  if (!gROOT->IsBatch()) TMVAGui( outfileName );
}
```

Es folgt der Quellcode für die Anwendung der Methoden auf einen unbekannten Datensatz.

```
void TMVApp_muster()
{

  // Erzeugt einen Pointer auf ein neues Reader Objekt.
  TMVA::Reader *reader = new TMVA::Reader( "Color:!Silent" );

  // Create a set of variables and declare them to the reader
  // - the variable names MUST corresponds in name and type to those given in the
       weight file(s) used

  // Es wird ein Satz Variablen erzeugt, der anschliesend dem Reader ubergeben
      wird. Die Namen und Typen der Variablen
  // mussen (!) denen entsprechen, die zuvor beim Trainieren dem Factory Objekt u
      bergeben wurden.
  Float_t xRayRmsX, xRayRmsY, xRayExcentricity, xRayLongSkewness,
      xRayTransSkewness, xRayLongKurtosis, xRayTransKurtosis, xRayRadius,
      xRayWidth, xRayLength, trackLongRms, trackTransRms, trackExcentricity,
      trackLongSkewness, trackTransSkewness, trackLongKurtosis,
      trackTransKurtosis, trackLength, trackWidth, trackPixPerLength, trackNpx;
  // Zuschauer-Variablen:
  Int_t eventNumber, runNumber, xRayNpx;
  Float_t xRayEnergy, xRayCharge;

  // XRay-Variablen
  reader->AddVariable("xRayRmsX", &xRayRmsX);
  reader->AddVariable("xRayRmsY", &xRayRmsY);
  reader->AddVariable("xRayExcentricity", &xRayExcentricity);
  reader->AddVariable("xRayLongSkewness", &xRayLongSkewness);
  reader->AddVariable("xRayTransSkewness", &xRayTransSkewness);
  reader->AddVariable("xRayLongKurtosis", &xRayLongKurtosis);
  reader->AddVariable("xRayTransKurtosis", &xRayTransKurtosis);
  reader->AddVariable("xRayRadius", &xRayRadius);
  reader->AddVariable("xRayLength", &xRayLength);
  reader->AddVariable("xRayWidth", &xRayWidth);

  // Track Variablen
  reader->AddVariable("trackLongRms", &trackLongRms);
  reader->AddVariable("trackTransRms", &trackTransRms);
  reader->AddVariable("trackExcentricity", &trackExcentricity);
  reader->AddVariable("trackLongSkewness", &trackLongSkewness);
  reader->AddVariable("trackTransSkewness", &trackTransSkewness);
  reader->AddVariable("trackLongKurtosis", &trackLongKurtosis);
  reader->AddVariable("trackTransKurtosis", &trackTransKurtosis);
  reader->AddVariable("trackLength", &trackLength);
  reader->AddVariable("trackWidth", &trackWidth);
```

```
reader->AddVariable("trackPixPerLength := trackNpx / trackLength", &
    trackPixPerLength);

// Zuschauer-Variablen. Alle Zuschauer-Variablen des Factory-Objekts mussen an
    den Reader ubergeben werden.
reader->AddSpectator("eventNumber", &eventNumber);
reader->AddSpectator("runNumber", &runNumber);
reader->AddSpectator("xRayEnergy", &xRayEnergy);
reader->AddSpectator("xRayNpx", &xRayNpx);
reader->AddSpectator("xRayCharge", &xRayCharge);


// Es werden die verschiedenen Methoden gebucht. Der erste String ist ein
    benutzerdefinierter, eindeutiger
// Name. Der zweite String gibt den Pfad zu einer .weights.xml Datei an, welche
     zuvor von der Factory erstellt
// wurde.
reader->BookMVA( "MLP_BFGS", "./weights/Klass_All/MVXrayAnalysis_MLP_BFGS.
    weights.xml");
reader->BookMVA( "MLP_BP", "./weights/Klass_All/MVXrayAnalysis_MLP_BP.weights.
    xml");
reader->BookMVA( "BDT_AdaBoost_Depth", "./weights/Klass_All/
    MVXrayAnalysis_BDT_AdaBoost_Depth.weights.xml");
reader->BookMVA( "BDT_AdaBoost", "./weights/Klass_All/
    MVXrayAnalysis_BDT_AdaBoost.weights.xml");
reader->BookMVA( "SVM", "./weights/Klass_All/MVXrayAnalysis_SVM.weights.xml");


// Hier werden die Histogramme erstellt, in die die Klassifikation der
    Ereignisse fur die verschiedenen
// Methoden eingetragen wird.
UInt_t nbin = 100;
TH1F   *histMLP(0), *histBDT(0), *histBDT_Depth(0), *histSVM(0), *histMLP_BP(0)
    ;
histMLP = new TH1F("MVA_MLP_BFGS", "MVA_MLP_BFGS", nbin, -1.25, 1.5);
histBDT = new TH1F("MVA_BDT_AdaBoost", "MVA_BDT_AdaBoost", nbin, -0.8, 0.8);
histBDT_Depth = new TH1F("MVA_BDT_AdaBoost_Depth", "MVA_BDT_AdaBoost_Depth",
    nbin, -0.8, 0.8);
histMLP_BP = new TH1F("MVA_MLP_BP", "MVA_MLP_BP", nbin, -1.25, 1.5);
histSVM = new TH1F("MVA_SVM", "MVA_SVM", nbin, 0, 1.0);

// Datei mit unbekannten Ereignissen wird eingelesen. In ihr ist erneut ein
    Tree, der die Daten
// speichert.
TFile *input = new TFile;
input = TFile::Open( "./TMVATree.root");

// Ein neuer Tree wird aus dem Tree der .root Datei erstellt.
TTree* theTree = (TTree*)input->Get("T");
// Hier wird die Verbindung zwischen dem Tree und dem Reader hergestellt, indem
     beide auf
// die gleichen Variablen zugreifen. Hierbei werden keine Zuschauer-Variablen
    mitgegeben.
// Xray-Variablen
theTree->SetBranchAddress("xRayRmsX", &xRayRmsX);
theTree->SetBranchAddress("xRayRmsY", &xRayRmsY);
theTree->SetBranchAddress("xRayExcentricity", &xRayExcentricity);
```

```
theTree->SetBranchAddress("xRayLongSkewness", &xRayLongSkewness);
theTree->SetBranchAddress("xRayTransSkewness", &xRayTransSkewness);
theTree->SetBranchAddress("xRayLongKurtosis", &xRayLongKurtosis);
theTree->SetBranchAddress("xRayTransKurtosis", &xRayTransKurtosis);
theTree->SetBranchAddress("xRayRadius", &xRayRadius);
theTree->SetBranchAddress("xRayLength", &xRayLength);
theTree->SetBranchAddress("xRayWidth", &xRayWidth);
// Track-Variablen
theTree->SetBranchAddress("trackLongRms", &trackLongRms);
theTree->SetBranchAddress("trackTransRms", &trackTransRms);
theTree->SetBranchAddress("trackExcentricity", &trackExcentricity);
theTree->SetBranchAddress("trackLongSkewness", &trackLongSkewness);
theTree->SetBranchAddress("trackTransSkewness", &trackTransSkewness);
theTree->SetBranchAddress("trackLongKurtosis", &trackLongKurtosis);
theTree->SetBranchAddress("trackTransKurtosis", &trackTransKurtosis);
theTree->SetBranchAddress("trackLength", &trackLength);
theTree->SetBranchAddress("trackWidth", &trackWidth);
theTree->SetBranCHAddress("trackNpx", &trackNpx);

// Ein TStopwatch Objekt wird verwendet, um die Zeit zu messen, die fur die
    Klassifikation
// benotigt wird.
TStopwatch sw;
sw.Start();

// Die eigentliche Klassifikation findet in dieser for-Schleife statt. T ist
    der Tree, welcher in der
// eingelesenen .root Datei gespeichert ist. Es wird also uber alle Eintrage
    des Trees gelaufen.
for (Long64_t ievt=0; ievt < theTree->GetEntries(); ievt++) {
  // Alle 1000 Ereignisse wird diese Anzahl ausgegeben.
  if (ievt%1000 == 0) std::cout << "--- ... Processing event: " << ievt << std
      ::endl;

  // Holt den aktuellen Eintrag des Trees
  theTree->GetEntry(ievt);
  // Berechnet die Variable trackPixPerLength, da diese nicht direkt im Tree
      enthalten ist
  trackPixPerLength = trackNpx / trackLength;
  // Fullt die Histogramme, mit der Klassifikation der Methoden fur das
      Ereignis ievt.
  histMLP->Fill( reader->EvaluateMVA( "MLP_BFGS"));
  histBDT->Fill( reader->EvaluateMVA( "BDT_AdaBoost"));
  histMLP_BP->Fill( reader->EvaluateMVA( "MLP_BP"));
  histBDT_Depth->Fill (reader->EvaluateMVA( "BDT_AdaBoost_Depth"));
  histSVM->Fill( reader->EvaluateMVA( "SVM"));
}

// Stopt die Zeitnahme und gibt sie aus.
sw.Stop();
std::cout << "--- End of event loop: "; sw.Print();

// Erzeugt oder uberschreibt die Zieldatei.
TFile *target  = new TFile( "TMVApp_muster.root","RECREATE" );
// Schreibt die gefullten Histogramme in die .root Datei.
histMLP->Write();
histBDT->Write();
```

```
    histMLP_BP->Write();
    histBDT_Depth->Write();
    histSVM->Write();

    // Schliest die Datei und loscht den Reader.
    target->Close();
    delete reader;
    std::cout << "--- Created root file: TMVApp_muster.root containing the MVA
        output histograms" << std::endl;
    std::cout << "==> TMVAClassificationApplication is done!" << endl << std::endl;
}
```

A.2 Eingabe-Daten aller Variablen

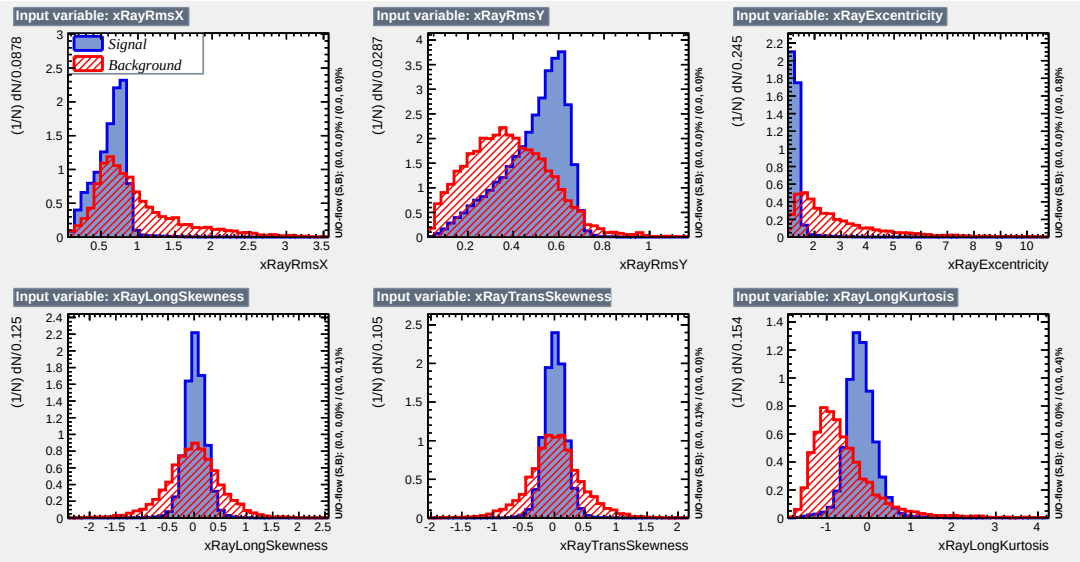


Abbildung A.1: Eingabe-Daten aller Variablen ohne Schnitte auf die Daten. Teil 1.

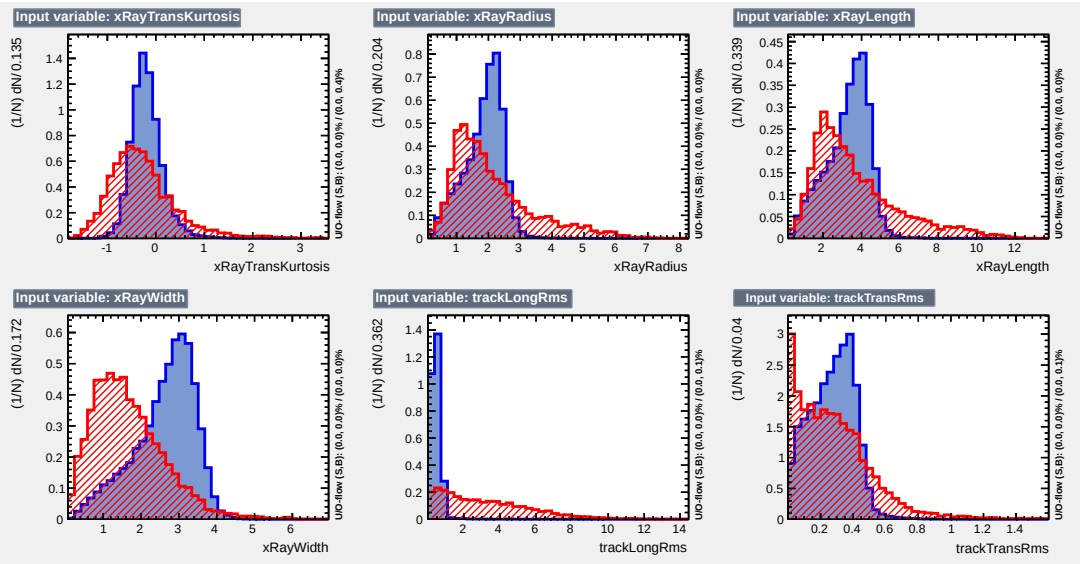


Abbildung A.2: Eingabe-Daten aller Variablen ohne Schnitte auf die Daten. Teil 2.

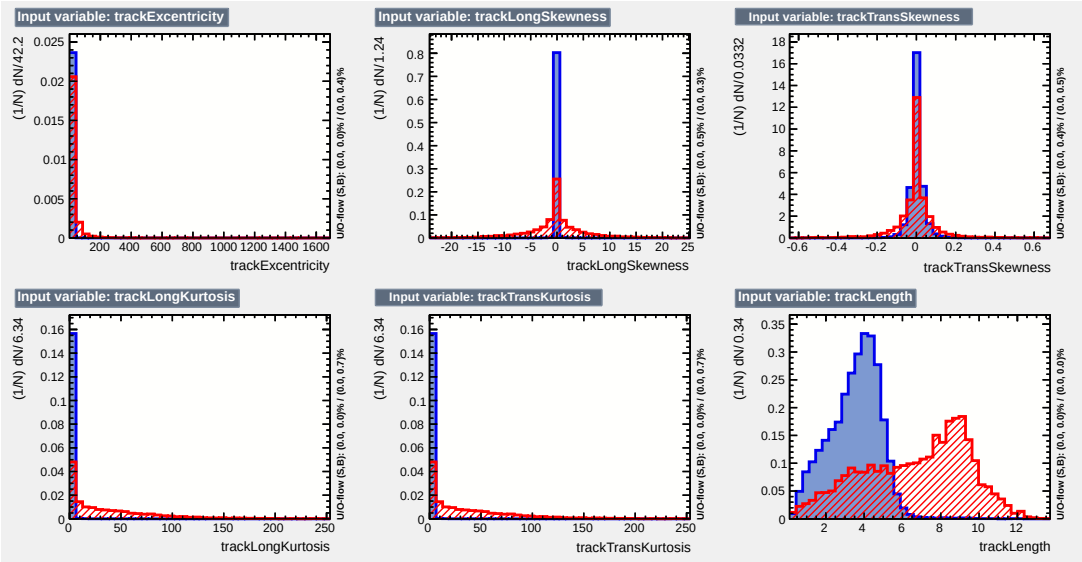


Abbildung A.3: Eingabe-Daten aller Variablen ohne Schnitte auf die Daten. Teil 3.

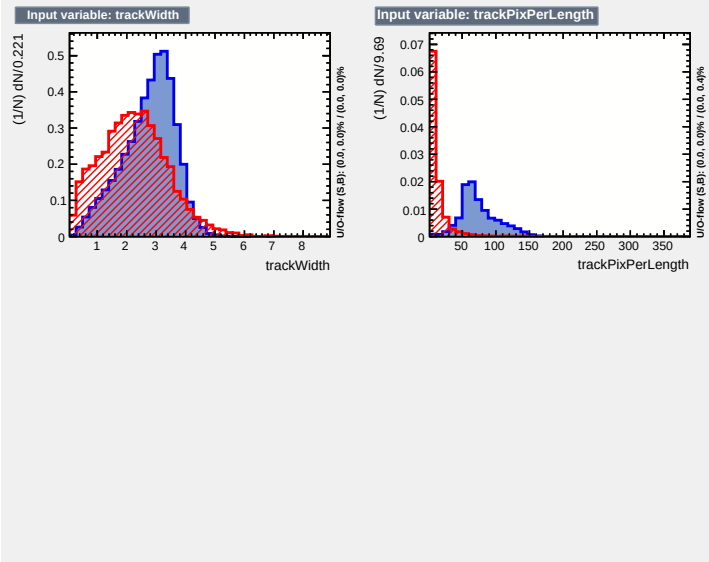
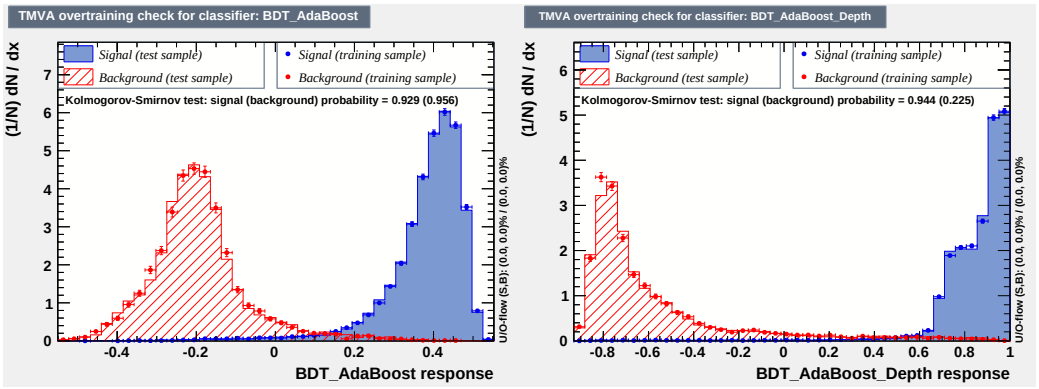


Abbildung A.4: Eingabe-Daten aller Variablen ohne Schnitte auf die Daten. Teil 4.

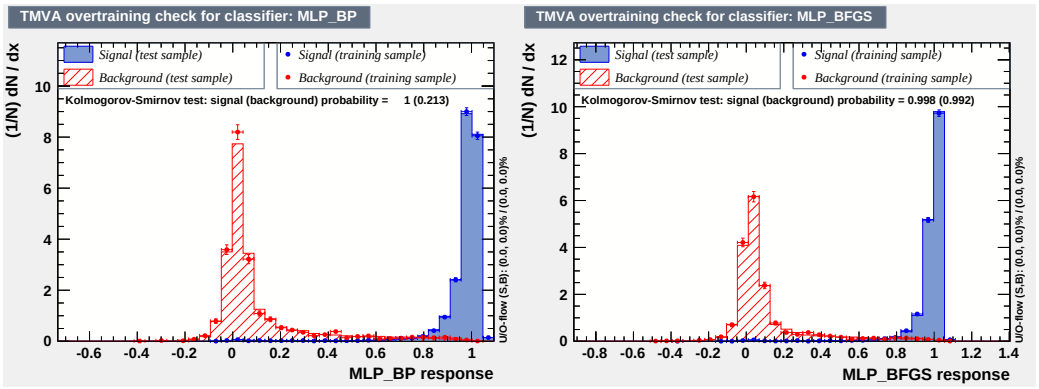
A.3 Klassifikation

A.3.1 Alle Ereignisse

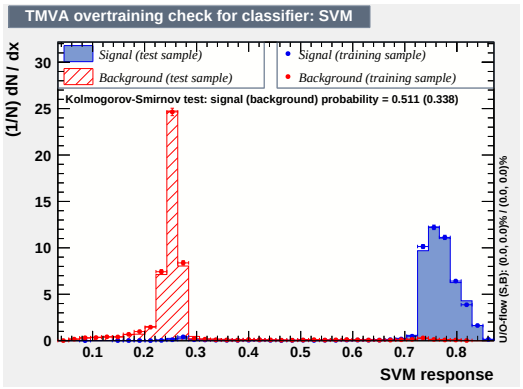
Overtraining



(a) (b)



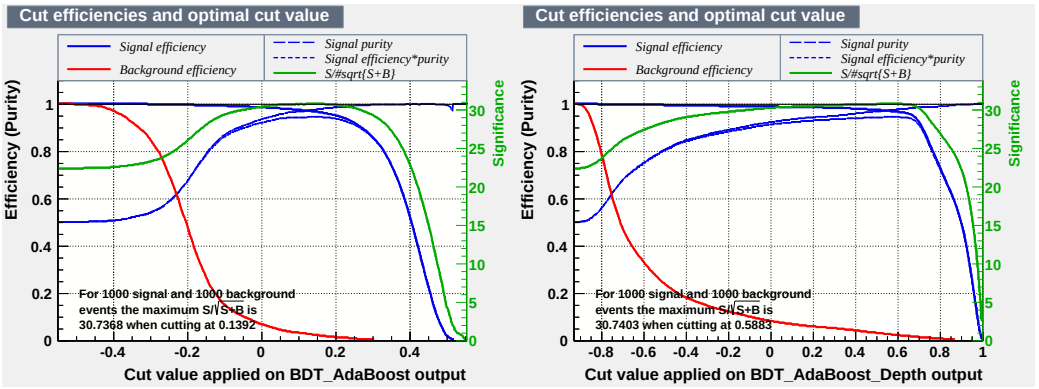
(c) (d)



(e)

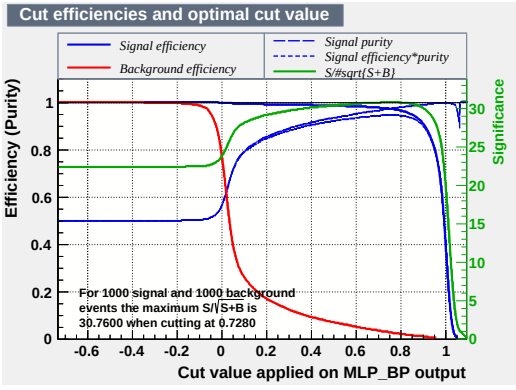
Abbildung A.5: Klassifikation aller Methoden für das Trainieren mit allen Testdaten (gefüllte Bereiche). Zusätzlich Trainingsdaten eingezeichnet (Punkte) zur Evaluierung des *Overtrainings*.

Schnitteffizienzen

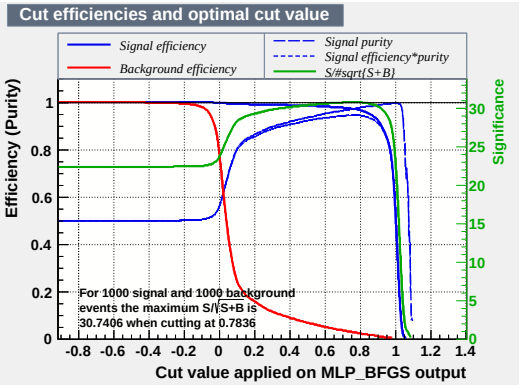


(a)

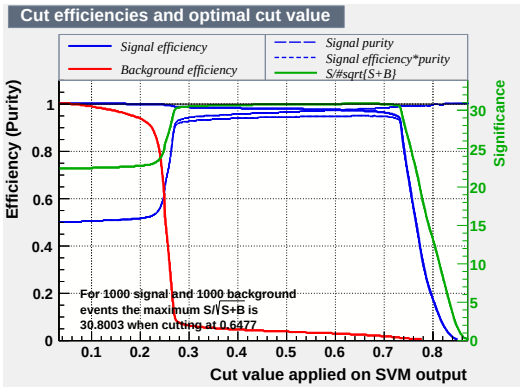
(b)



(c)



(d)

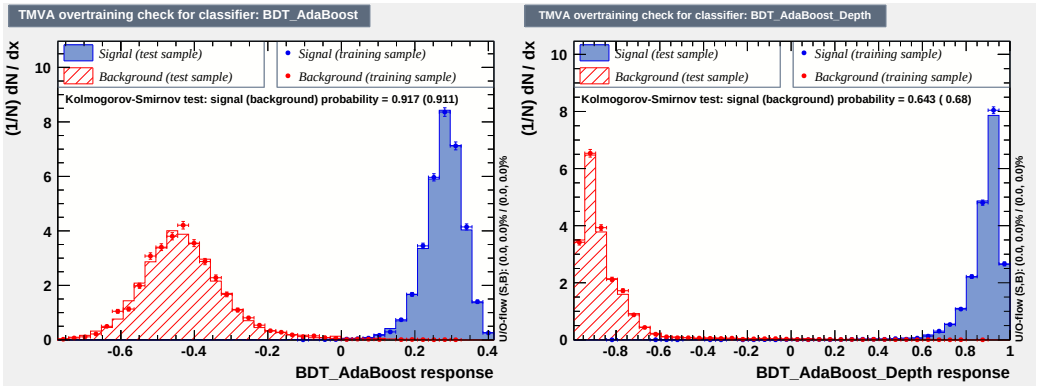


(e)

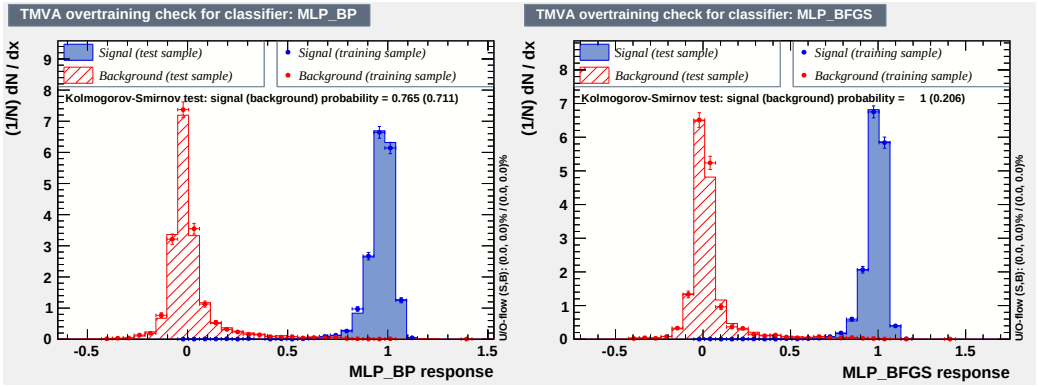
Abbildung A.6: Signal- und Untergrundeffizienzen für die verschiedenen Methoden in Abhängigkeit des Schnittwerts. Methoden basieren auf allen Trainingsdaten.

A.3.2 Schnitt auf die Energie

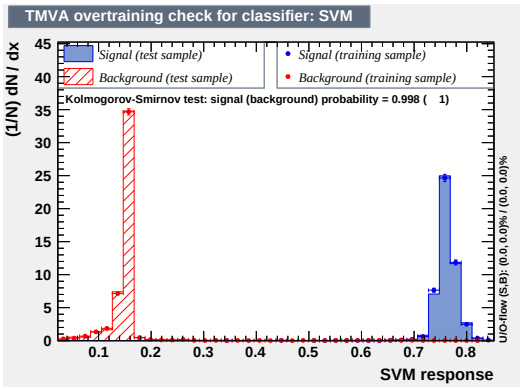
Overtraining



(a) (b)



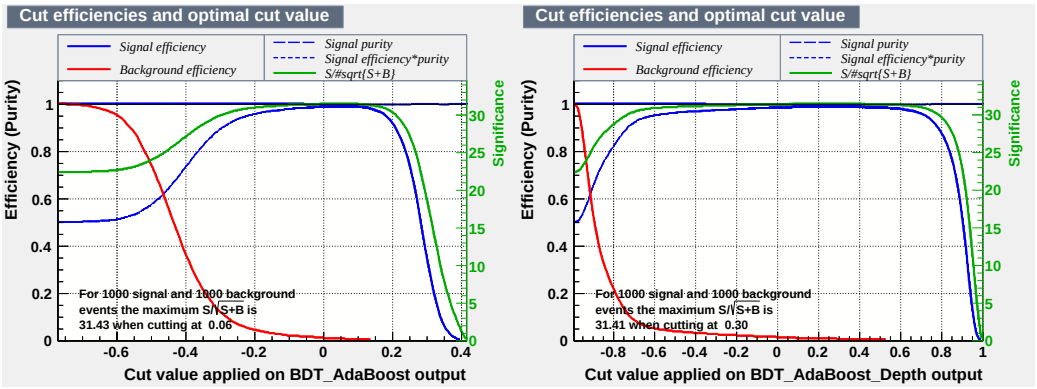
(c) (d)



(e)

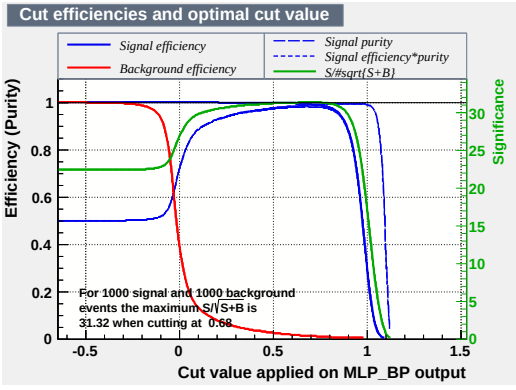
Abbildung A.7: Klassifikation aller Methoden für das Trainieren mit Testdaten (gefüllte Bereiche) bei Schnitt auf die Energie. Zusätzlich Trainingsdaten eingezeichnet (Punkte) zur Evaluierung des *Overtrainings*.

Schnitteffizienzen

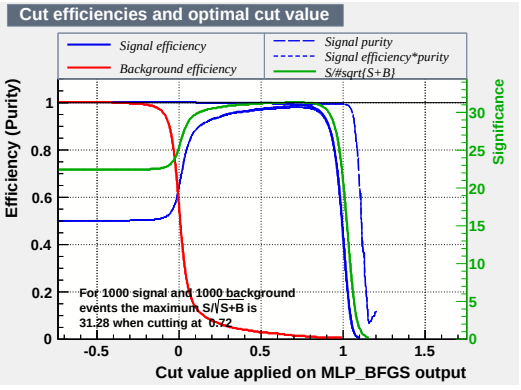


(a)

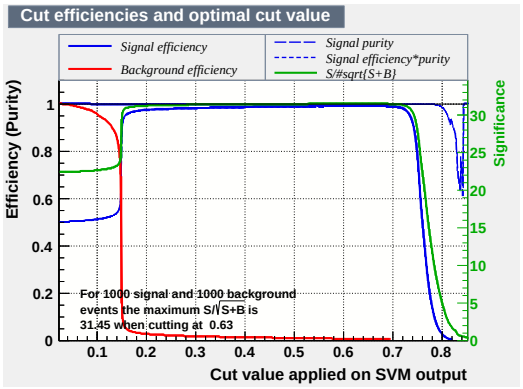
(b)



(c)



(d)

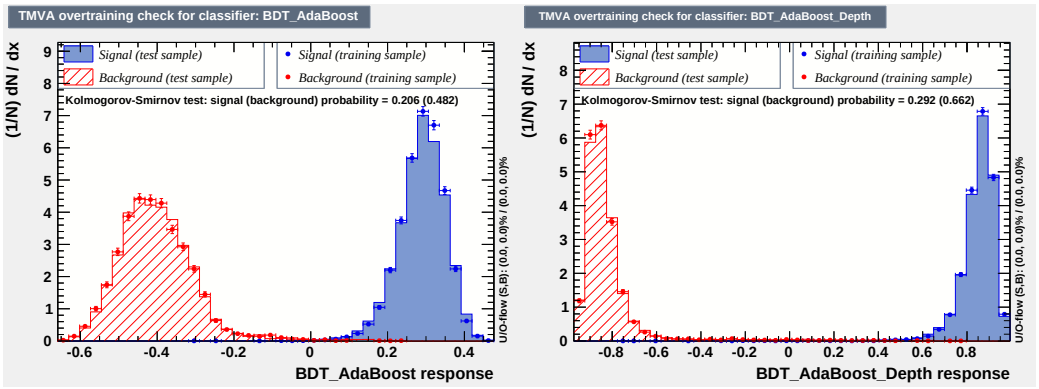


(e)

Abbildung A.8: Signal- und Untergrundeffizienzen für die verschiedenen Methoden in Abhängigkeit des Schnittwerts. Methoden basieren auf Trainingsdaten mit Schnitt auf Energie.

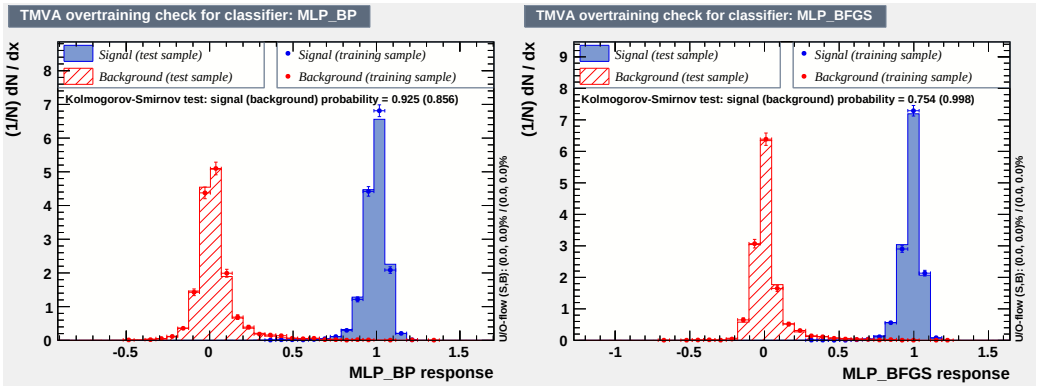
A.3.3 Energetisch reduzierte Daten - eigenes Training

Overtraining



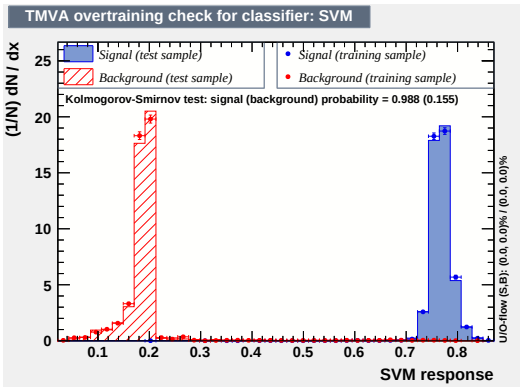
(a)

(b)



(c)

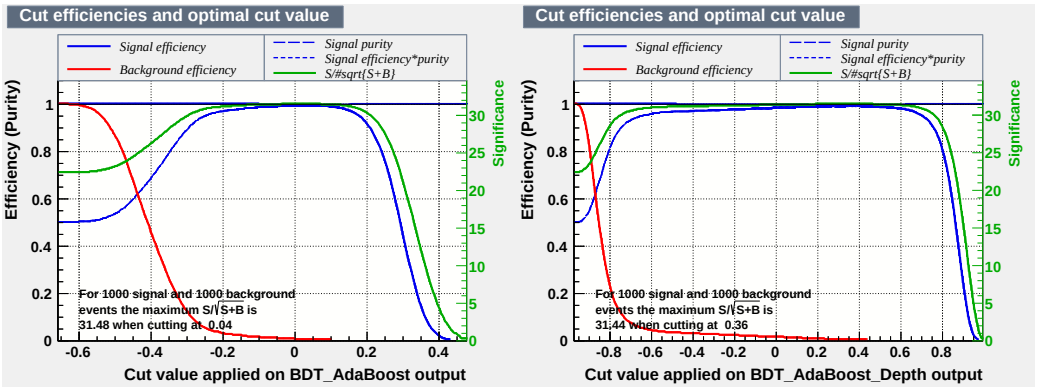
(d)



(e)

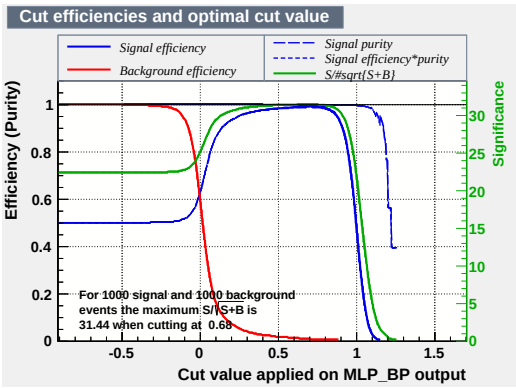
Abbildung A.9: Klassifikation aller Methoden für das Trainieren auf Basis der künstlichen, niederenergetischen Testdaten (gefüllte Bereiche). Zusätzlich Trainingsdaten eingezeichnet (Punkte) zur Evaluierung des *Overtrainings*.

Schnitteffizienzen

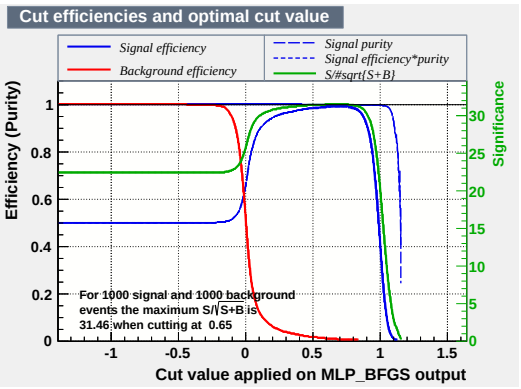


(a)

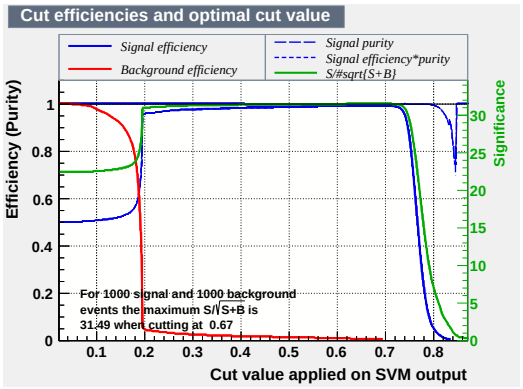
(b)



(c)



(d)



(e)

Abbildung A.10: Training für die energetisch reduzierten Daten. Signal- und Untergrundeffizienzen für die verschiedenen Methoden in Abhängigkeit des Schnittwerts.

A.4 Falsch klassifizierte Ereignisse

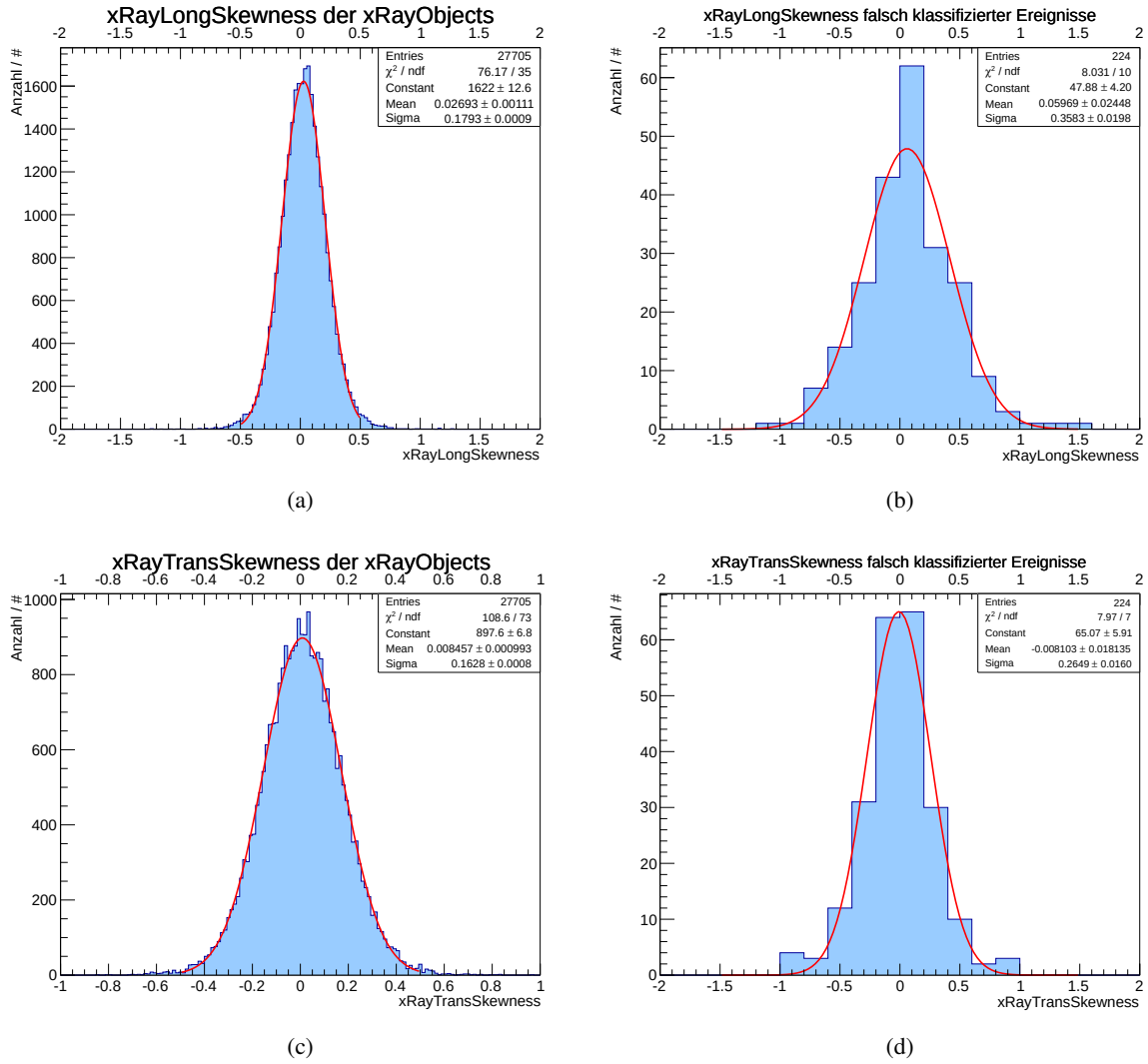
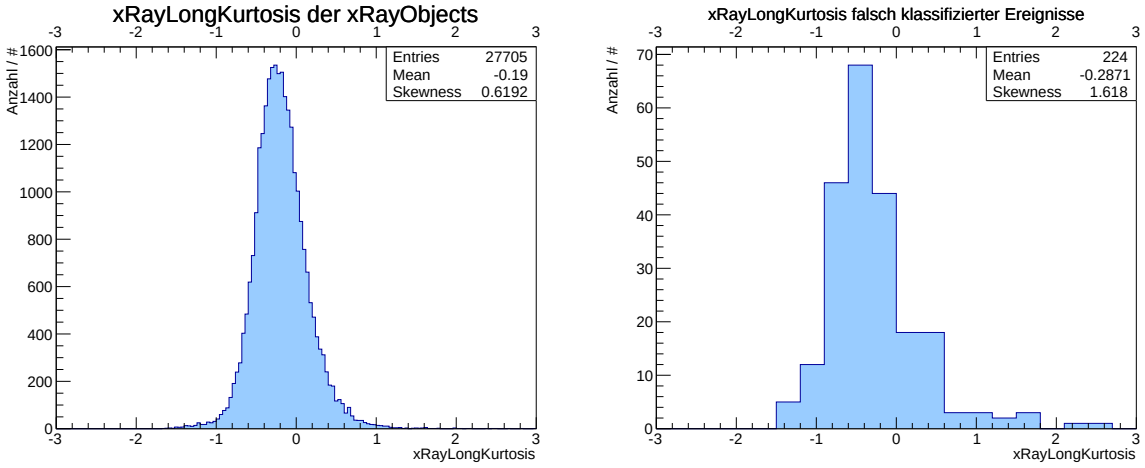
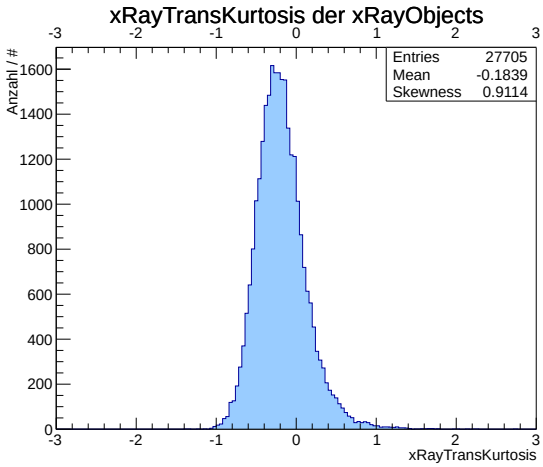


Abbildung A.11: Longitudinale und transversale *Skewness* der **XrayObjects** (links) und falsch klassifizierten Untergrundereignisse (rechts).

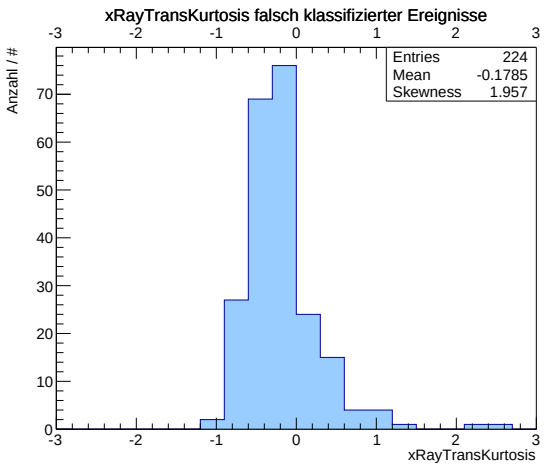


(a)

(b)



(c)



(d)

Abbildung A.12: Longitudinale und transversale *Kurtosis* der **XrayObjects** (links) und falsch klassifizierten Untergrundereignisse (rechts).

Literatur

[1] C. A. Baker u. a.,
„Improved Experimental Limit on the Electric Dipole Moment of the Neutron“,
Phys. Rev. Lett. 97 (13 Sep. 2006) 131801, DOI: 10.1103/PhysRevLett.97.131801,
URL: http://link.aps.org/doi/10.1103/PhysRevLett.97.131801.

[2] A. Knecht, *Parity (P) and time reversal (T) violation in the presence of an electric dipole moment in addition to the magnetic moment of the neutron*, Wikimedia, Public Domain, 2008.

[3] R. D. Peccei und H. R. Quinn, „CP Conservation in the Presence of Pseudoparticles“,
Phys. Rev. Lett. 38 (25 Juni 1977) 1440–1443, DOI: 10.1103/PhysRevLett.38.1440,
URL: http://link.aps.org/doi/10.1103/PhysRevLett.38.1440.

[4] E. Arik u. a., „Probing eV-scale axions with CAST“, *JCAP* 0902 (2009) 008,
DOI: 10.1088/1475-7516/2009/02/008, arXiv: 0810.4482 [hep-ex].

[5] J. K. Vogel,
„Searching for Solar Axions in the eV-Mass region with the CCD Detector at CAST“,
dissertation: Albert-Ludwigs-Universität Freiburg, Mai 2009,
URL: http://pcfr31.physik.uni-freiburg.de/arbeiten/theses/vogel.pdf.

[6] C. Krieger, „Construction and First Measurements of a GridPix based X-ray Detector“,
BONN-IB-2012-05, Magisterarb.: Physikalisches Institut der Universität Bonn, Feb. 2012.

[7] Y. Giomataris u. a., „MICROMEGAS: a high-granularity position-sensitive gaseous detector for high particle-flux environments“, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 376.1 (1996) 29–35,
ISSN: 0168-9002, DOI: http://dx.doi.org/10.1016/0168-9002(96)00175-1,
URL: http://www.sciencedirect.com/science/article/pii/0168900296001751.

[8] Y. Giomataris, „Development and prospects of the new gaseous detector “Micromegas”“,
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 419.2–3 (1998) 239–250, ISSN: 0168-9002,
DOI: http://dx.doi.org/10.1016/S0168-9002(98)00865-1,
URL: http://www.sciencedirect.com/science/article/pii/S0168900298008651.

[9] S. Andriamonje u. a., „Development and performance of Microbulk Micromegas detectors“,
Journal of Instrumentation 5.02 (2010) P02001,
URL: http://stacks.iop.org/1748-0221/5/i=02/a=P02001.

[10] M. A. Chefdeville, „Development of Micromegas-like gaseous detectors using a pixel readout chip as collecting anode“ (2009), URL: http://dare.uva.nl/document/123168.

[11] M. Chefdeville u. a.,
„An electron-multiplying ‘Micromegas’ grid made in silicon wafer post-processing technology“,
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 556.2 (2006) 490–494, ISSN: 0168-9002,
DOI: http://dx.doi.org/10.1016/j.nima.2005.11.065,
URL: http://www.sciencedirect.com/science/article/pii/S0168900205021418.

[12] A. Hoecker u. a., „TMVA - Toolkit for Multivariate Data Analysis“,
ArXiv Physics e-prints (März 2007), eprint: arXiv:physics/0703039.

[13] E. Pearson und H. Hartley, *0.(1972). Biometrika tables for statisticians, Vol. 2*.

[14] G. Cybenko, „Approximation by superpositions of a sigmoidal function“, English,
Mathematics of Control, Signals and Systems 2.4 (1989) 303–314, ISSN: 0932-4194,
DOI: 10.1007/BF02551274, URL: http://dx.doi.org/10.1007/BF02551274.

[15] C. G. BROYDEN, „The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations“, *IMA Journal of Applied Mathematics* 6.1 (1970) 76–90,
DOI: 10.1093/imamat/6.1.76,
eprint: http://imamat.oxfordjournals.org/content/6/1/76.full.pdf+html,
URL: http://imamat.oxfordjournals.org/content/6/1/76.abstract.

[16] R. Fletcher, „A new approach to variable metric algorithms“,
The Computer Journal 13.3 (1970) 317–322, DOI: 10.1093/comjnl/13.3.317,
eprint: http://comjnl.oxfordjournals.org/content/13/3/317.full.pdf+html,
URL: http://comjnl.oxfordjournals.org/content/13/3/317.abstract.

[17] D. Goldfarb, „A family of variable-metric methods derived by variational means“,
Math. Comp. 24 (1970) 23–26, ISSN: 0025-5718.

[18] D. F. Shanno, „Conditioning of quasi-Newton methods for function minimization“,
Math. Comp. 24 (1970) 647–656, ISSN: 0025-5718.

[19] Y. Freund und R. Schapire,
„A desicion-theoretic generalization of on-line learning and an application to boosting“,
Computational Learning Theory, hrsg. von P. Vitányi, Bd. 904,
Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1995 23–37,
ISBN: 978-3-540-59119-1, DOI: 10.1007/3-540-59119-2_166,
URL: http://dx.doi.org/10.1007/3-540-59119-2_166.

[20] B. P. Roe u. a.,
„Boosted decision trees as an alternative to artificial neural networks for particle identification“,
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 543.2–3 (2005) 577–584, ISSN: 0168-9002,
DOI: http://dx.doi.org/10.1016/j.nima.2004.12.018,
URL: http://www.sciencedirect.com/science/article/pii/S0168900205000355.

[21] M. Matsumoto und T. Nishimura, „Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator“,
ACM Trans. Model. Comput. Simul. 8.1 (Jan. 1998) 3–30, ISSN: 1049-3301,
DOI: 10.1145/272991.272995, URL: http://doi.acm.org/10.1145/272991.272995.

[22] C. Krieger, J. Kaminski und K. Desch,
„InGrid-based X-ray detector for low background searches“,
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment (2013) pages, ISSN: 0168-9002,
DOI: http://dx.doi.org/10.1016/j.nima.2013.08.075,
URL: http://www.sciencedirect.com/science/article/pii/S0168900213012163.