technische universität
dortmund

Diplomarbeit

Gamma-Hadron-Separation im MAGIC-Experiment durch verteilungsgestütztes Sampling

Marius Helf
April 2011

Gutachter:
Prof. Katharina Morik
Prof. Wolfgang Rhode

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS 8)
http://www-ai.cs.tu-dortmund.de

In Kooperation mit:
Fakultät für Physik
Experimentelle Physik 5

Gamma-Hadron-Separation im MAGIC-Experiment durch verteilungsgestütztes Sampling

Marius Helf

Kompiliert am 27. April 2011 um 20:21

Inhaltsverzeichnis

1. Einleitung

Im MAGIC-Experiment werden zwei Cherenkov-Teleskope betrieben, deren Beobachtungen eine sehr große Datenmenge erzeugen. Dabei macht jedoch der Anteil der für weitere Analysen interessanten Gamma-Teilchen nur etwa ein Tausendstel aller Ereignisse aus. Diese müssen durch geeignete Verfahren von den übrigen Ereignissen separiert werden. Es liegt also ein Klassifikationsproblem vor, für dessen Lösung zur Zeit ein Random Forest verwendet wird. Die sehr großen Datenmengen, die für das Training des Klassifikators zur Verfügung stehen, lassen jedoch Klassifikationsverfahren attraktiv erscheinen, die nicht nur eine begrenzte, statische Trainingsmenge nutzen, sondern durch Resampling während des Lernens einen beliebig großen Teil der zur Verfügung stehenden Daten verwenden. Dazu wird eine Erweiterung des Boosting-Verfahrens Ada^{2}Boost entwickelt, die in einigen Iterationen die Trainingsmenge durch neu gezogene Daten ersetzt. Weiterhin wird Ada^{2}Boost so modifiziert, dass weiche Basisklassifikatoren verwendet werden können.

Neben des großen Umfangs der Trainingsdaten besteht die Schwierigkeit, dass sich die Verteilungen einiger Merkmale in Abhängigkeit von Umgebungsparametern ändern. Diese Eigenschaft wird derzeit nicht oder nur schwach berücksichtigt. In dieser Arbeit wird ein Verfahren entwickelt, das die Daten so partitioniert, dass die Verteilung innerhalb der Partitionen möglichst homogen ist, um dann auf jeder Partition ein auf die jeweilige Verteilung spezialisiertes Modell zu trainieren. Das Problem nach der Suche einer optimalen Partitionierung wird in eine Merkmalsselektion transformiert, so dass vorhandene Verfahren zur Merkmalsselektion angewendet werden können.

Die entwickelten Verfahren werden zur Evaluierung für die Data Mining-Umgebung RapidMiner implementiert. Weiterhin wird ein Framework vorgestellt, mit dem sich RapidMiner-Prozesse auf ein Rechencluster verteilen lassen.

Teil I beschreibt die physikalischen Grundlagen und die Vorarbeiten, auf denen die neu entwickelten Verfahren aufbauen. Kapitel 3 gibt einen Einblick in die Herkunft kosmischer Teilchen und den Hintergrund des MAGIC-Experiments. Das derzeit etablierte Klassifikationsverfahren wird beschrieben. Kapitel 4 definiert ein formales Gerüst für die Klassifikation und erläutert wichtige Verfahren zur Bewertung von Klassifikationsmodellen. In Kapitel 5 werden einige konkrete Lernverfahren vorgestellt. Darauf aufsetzende Meta-Verfahren werden in Kapitel 6 beschrieben. Dazu gehören insbesondere Ada^{2}Boost und der Random Forest. Schließlich werden Methoden zur Vorverarbeitung und Merkmalsselektion in Kapitel 7.1 beschrieben.

In Teil II werden die neuen Verfahren entwickelt und evaluiert. Kapitel 8 gibt einen kurzen Überblick über diese Verfahren. Das folgende Kapitel beschreibt die Testumgebung, die zur Evaluierung genutzt wird. Eine statistische Voruntersuchung der MAGIC-Daten findet in Kapitel 10 statt und Kapitel 11 stellt den Recall-Chooser vor, ein Hilfsverfahren, das auf der ROC-Analyse basiert. Kapitel 12 erweitert Ada^{2}Boost um die Nutzung weicher Basisklassifikatoren. Dieses Verfahren wiederum wird in Kapitel 13 um ein Resampling während des Lernprozesses erweitert, so dass große Datenmengen verarbeitet werden können. Das Binningverfahren zum Partitionieren der Trainingsmenge wird in Kapitel 14 beschrieben. Zum Vergleich mit dem etablierten Verfahren bewertet Kapitel 15 die neuen Verfahren in Bezug auf den Random Forest. Das letzte Kapitel 16 fasst die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen.

2. Terminologie

Diese Arbeit behandelt ein Problem der Physik mit den Methoden der Informatik und Statistik. Diese unterschiedlichen Disziplinen benennen viele gleiche Dinge mit unterschiedlichen Fachbegriffen. Dabei kommt es auch vor, dass beispielsweise Begriffe aus der Physik in der Informatik gänzlich unbekannt sind. In diesem Abschnitt werden die wichtigsten Begriffe definiert.

In dieser Arbeit bezeichnet ein *Event* oder *Ereignis* das Eintreffen eines Teilchens, unabhängig davon, ob es es sich um ein Gamma-Teilchen oder ein hadronisches Teilchen handelt. Jedes Event entspricht einem *Beispiel* im Sinne des Data Mining.

Im weiteren werden die vorhandenen Events *klassifiziert*, d.h. der Klasse „Gamma-Teilchen“ oder „Hadronen“ zugeordnet. Dies entspricht dem Prozess, der in der Physik in der Regel mit *Separation* bezeichnet wird.

Der Begriff *Parameter* wird gelegentlich in Anlehnung an bestehende Literatur im Bereich der Physik synonym zu den aus dem Data Mining bekannten Begriffen *Merkmal* oder *Feature* gebraucht, da die derzeit in der MAGIC-Analyse gebräuchlichen Merkmale als Hillas-Parameter, Bildparameter etc. bezeichnet werden. Merkmale sind Größen, die ein Event charakterisieren, und anhand derer eine Klassifikation erfolgen kann.

Stratification oder *Stratifizierung* bedeutet im Kontext dieser Arbeit die Transformation der a-priori-Wahrscheinlichkeiten in Beispielmengen. Dies entspricht nicht der Verwendung des Begriffs in RapidMiner, wo es das Sampling unter Beibehaltung der a-priori-Wahrscheinlichkeiten bezeichnet.

Teil I.
Grundlagen

3. Motivation und physikalische Grundlagen

Diese Arbeit befasst sich mit der Klassifikation von Teleskopaufnahmen. Wie in den folgenden Abschnitten erläutert wird, stehen zum Training eines Modells sehr große Datenmengen mit mehreren Millionen Ereignissen zur Verfügung, und es können und werden täglich neue Daten aufgenommen bzw. erzeugt. Dadurch steht dem Analysten eine quasi unbegrenzte Datenmenge zur Verfügung um ein Modell der Daten zu erzeugen. Andererseits ist es mit klassischen Verfahren kaum möglich diese großen Datenmengen zu nutzen. Einerseits muss in der Regel der gesamte Datensatz in den Hauptspeicher geladen werden, was auf den meisten Systemen bereits eine deutliche Einschränkung bedeutet. Desweiteren ist die Laufzeit der meisten Algorithmen größer als $O(n)$, so dass sie bei großen Datenmengen überproportional wächst. Somit werden gängige Verfahren für große Datenmengen ineffizient, bzw. können nur einen sehr geringen Teil aller zur Verfügung stehenden Daten nutzen.

Weiterhin ändern sich die Verteilungen innerhalb der Gamma- und Hadron-Klasse in Abhängigkeit vom Zenitwinkel des Teleskops und der Energie der beobachteten Events. Der Zenitwinkel ist als Metainformation in den Daten enthalten. Die Energie ist zwar a-priori nicht bekannt, kann aber aus anderen Merkmalen abgeschätzt werden.

Die in dieser Arbeit entwickelten Verfahren machen sich eben diese Eigenschaften der Daten, d. h. die große Menge und die veränderlichen Verteilungen, zu Nutze.

Dieses Kapitel skizziert zunächst die Herkunft der zu untersuchenden kosmischen Teilchen, gibt dann einen Überblick über die MAGIC-Teleskope, und erläutert dann das zur Zeit etablierte Analyseverfahren.

3.1. Die Herkunft kosmischer Teilchen

Es gibt im Universum viele Objekte, die hochenergetische Strahlung abgeben. Durch deren Beobachtung und Analyse können Rückschlüsse auf die Entstehung der Strahlung, die aussendenden Objekte und letztendlich über die Entwicklung und Struktur des Universums gezogen werden. Im MAGIC-Experiment[1] soll mit den MAGIC-Teleskopen hochenergetische Gamma-Strahlung beobachtet werden. Diese Strahlung im Bereich von etwa 30 GeV bis über 10 TeV wird sowohl von Objekten innerhalb der Milchstraße als auch von Quellen außerhalb der Galaxie, sogenannten extragalaktischen Quellen, ausgesandt. Zu den extragalaktischen Quellen gehören insbesondere die Kerne aktiver Galaxien (AGNs), zu den galaktischen Quellen Supernovaüberreste (SNRs), Pulsarwindnebel (PWN) und Röntgenbinärsysteme (XRBs) [3].

Supernovaüberreste sind Überreste von Supernovaexplosionen, bei denen sich die Materie des Sterns schalenartig mit sehr hoher Geschwindigkeit ausbreitet. Auch bei einem Pulsarwindnebel handelt es sich um die Überreste einer Supernovaexplosion, wobei der Kern des Sterns jedoch zu einem schnell rotierenden Neutronenstern kollabiert ist. Weicht die Rotationsachse des Neutronensterns von der magnetischen Achse

[1] Major Athmospheric Gamma-Ray Imaging Cherenkov Telescopes

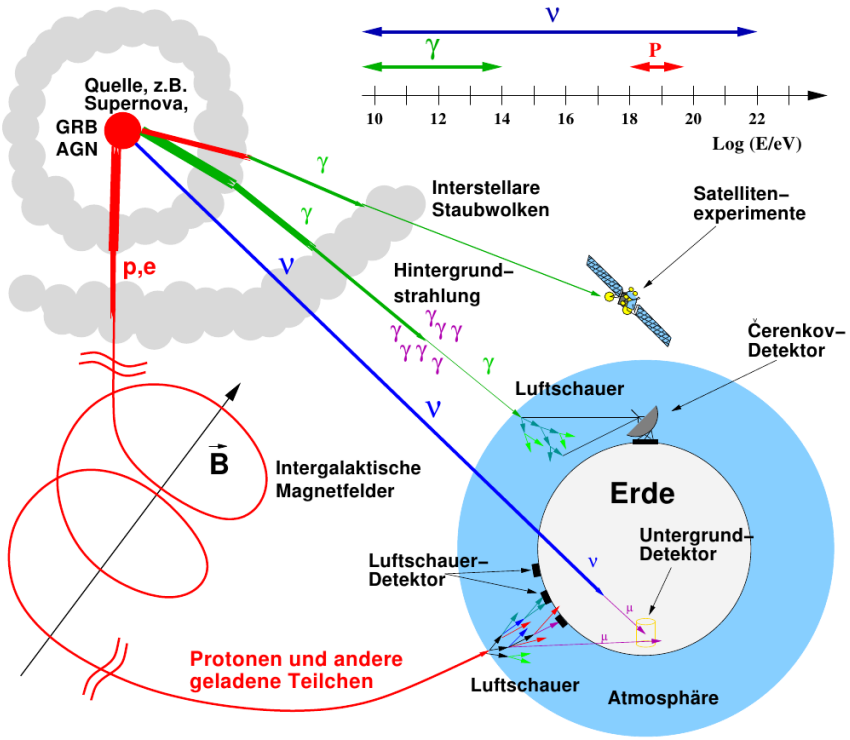


Abbildung 1: Der Weg kosmischer Teilchen von der Quelle bis zum Detektor auf der Erde. Aus [8] nach [40].

ab, entsteht hochenergetische Synchrotronstrahlung, die in Form sogenannter Jets stark gerichtet abgegeben wird. In einem Röntgenbinärsystem umkreisen sich ein kompaktes Objekt wie etwa ein Neutronenstern oder ein schwarzes Loch und ein Riese als Begleiter. Überschreitet der Riese das sogenannte *Roche-Volumen*, so fließt Materie vom Begleiter zu dem kompakten Objekt. Es bildet sich eine Akkretionsscheibe aus, d. h. eine Materie-Scheibe, die auf das kompakte Objekt stürzt. Dabei entstehen ebenfalls Jets, in denen Teilchen durch verschiedene Effekte zu sehr hohen Energien beschleunigt werden. Die Beschleunigungseffekte sind vermutlich denen ähnlich, die auch bei AGNs zum Tragen kommen.

Im Zentrum einer aktiven Galaxie befindet sich supermassives schwarzes Loch mit 10^6 bis 10^9 Sonnenmassen, welches Masse aus der Galaxie akkretiert, d. h. es zieht Masse aus der Galaxie an, bis diese in das schwarze Loch stürzt. Auch hierbei entstehen eine Akkretionsscheibe und zwei Jets. Durch starke, rotierende Magnetfelder werden geladene Teilchen beschleunigt und geben dabei Synchrotronstrahlung ab. Durch den inversen Synchrotron-Self-Compton-Effekt können dann auch Photonen im Hochenergiebereich entstehen, d. h. Gamma-Strahlung, die mit den MAGIC-Teleskopen beobachtet werden kann.

Bei der Beobachtung von verschiedenen AGNs werden verschiedene Energiespektren beobachtet. Anhand der Eigenschaften dieser Spektren werden die aktiven Galaxien in unterschiedliche Klassen unterteilt. Es ist ein offenes Forschungsthema der Astroteilchenphysik, wodurch die Unterschiede zustande kommen. Es wird jedoch vermu-

tet, dass alle AGNs ähnliche Eigenschaften haben, und die unterschiedlichen Spektren daher stammen, dass die Galaxien aus unterschiedlichen Blickwinkeln betrachtet werden. Diese Hypothese wird als *Unification*-Theorie bezeichnet. Danach werden, je nachdem ob ein Jet direkt auf die Erde gerichtet ist oder unter verschiedenen Winkeln beobachtet wird, unterschiedliche Energiespektren beobachtet. Die *Strong Unification*-Theorie geht davon aus, dass alle beobachteten AGNs bezüglich ihrer Struktur und der ablaufenden Prozesse identisch sind, während die *Weak Unification*-Theorie von kleineren Unterschieden ausgeht [39]. Ein Ziel des MAGIC-Experiments ist es, Erkenntnisse zu erlangen, die eine der beiden Unification-Theorien belegen können.

Im Hochenergiebereich entstehen zum einen Gamma-Strahlung, d. h. hochenergetische Photonen, zum anderen hadronische Teilchen, insbesondere Protonen, aber auch Kerne von höherwertigen Elementen, und Leptonen. Da Photonen ungeladen sind, bewegen sie sich weitgehend geradlinig von ihrer Quelle weg, so dass aus ihrer Richtungsinformation auf die Position der Quelle geschlossen werden kann. Die geladenen Teilchen werden jedoch von Magnetfeldern im interstellaren und intergalaktischen Raum abgelenkt, so dass jede Richtungsinformation verloren geht. Abbildung 1 veranschaulicht diesen Sachverhalt.

Treten die hochenergetischen Teilchen in die Erdatmosphäre ein, wechselwirken sie mit den Kernen der Moleküle in der Atmosphäre. Dabei werden kaskadenartige Teilchenschauer ausgelöst, die wiederum Cherenkov-Licht erzeugen. Bei Cherenkov-Licht handelt es sich um Licht, das entsteht, wenn die Geschwindigkeit bewegter Teilchen größer als die Ausbreitungsgeschwindigkeit von Licht im Medium ist. Kosmische Strahlung im Hochenergiebereich erzeugt in der Atmosphäre Cherenkov-Licht im optischen Bereich. Abbildung 2 zeigt Beispiele für solche Schauer.

3.2. Die Ziele des MAGIC-Experiments

Die MAGIC-Kollaboration betreibt im MAGIC-Experiment die zuvor erwähnten MAGIC-Teleskope. Das sind zwei Cherenkov-Teleskope, mit denen das Cherenkov-Licht der zuvor beschriebenen Luftschauer beobachtet wird. Das Ziel der Beobachtungen ist ein tieferer Einblick in die Vorgänge in Quellen hochenergetischer Gamma-Strahlung. Insbesondere sollen Erkenntnisse über die Beschleunigungsmechanismen der kosmischen Teilchen und deren Wechselwirkungen gewonnen werden. Außerdem soll die Herkunft der kosmischen Strahlung, d. h. die Herkunft geladener Teilchen, geklärt werden. Es wird vermutet, dass diese zumindest teilweise von denselben Objekten wie die Gamma-Strahlung stammt. Da die kosmische Strahlung jedoch keine Richtungsinformation trägt, sind tiefergehende Untersuchungen notwendig, um diese Hypothese zu bestätigen.

Seit in jüngster Zeit die Anzahl der bekannten Gamma-Quellen stetig gewachsen ist, können weiterhin sogenannte Populationsstudien durchgeführt werden. Dabei werden die aktiven Objekte katalogisiert und kategorisiert. Es finden nicht nur die Beobachtungen der MAGIC-Teleskope Beachtung, sondern auch Beobachtungen in anderen Energiebereichen wie dem Radio- oder Röntgenbereich. Aus den Beobachtungen werden Lichtkurven und Energiespektren berechnet. Bei einer Lichtkurve wird die Anzahl der beobachteten Teilchen aus einem bestimmten Energiebereich gegen die Zeit aufgetragen, für ein Energiespektrum der Teilchenfluss gegen die Energie.

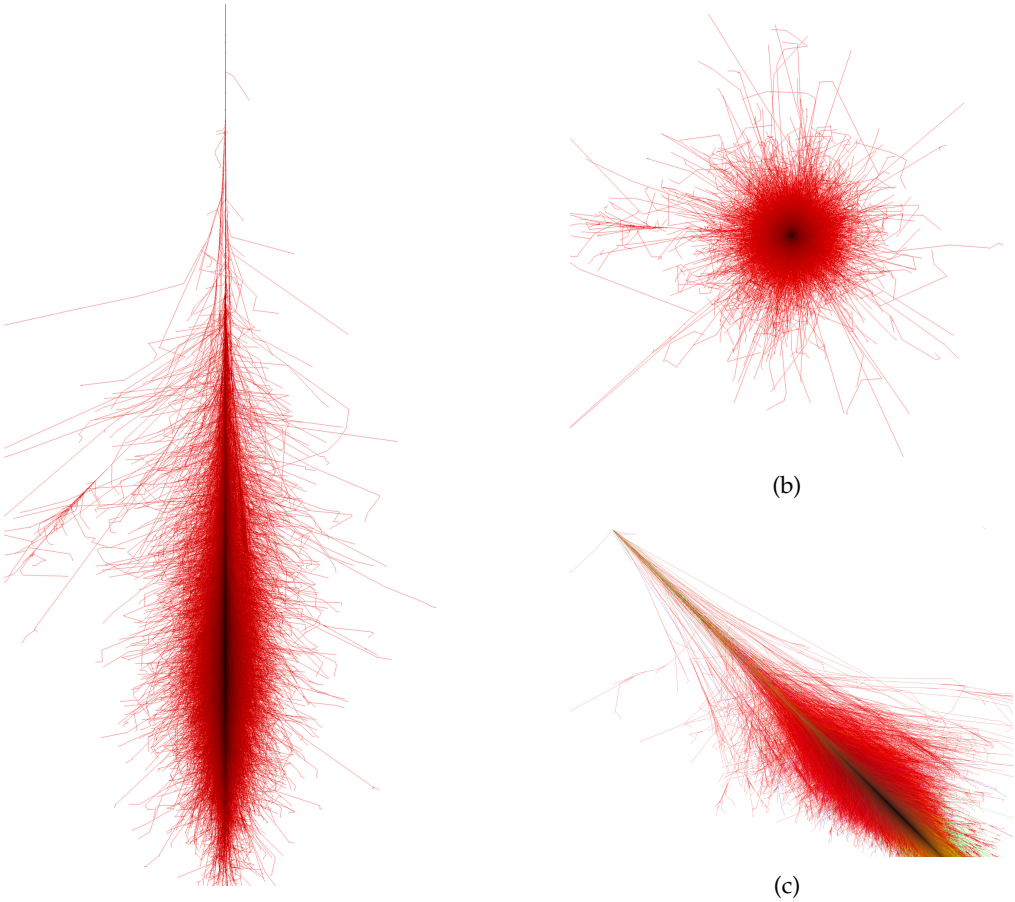


Abbildung 2: Bilder von simulierten Luftschauern. (a) und (c) zeigen die Seitenansicht zweier Luftschauer; (b) ist die Projektion eines Luftschauers in die x-y-Ebene. (a) und (b) wurden von Teilchen erzeugt, das senkrecht in die Erdatmospäre eingedrungen sind. In (c) ist das Teilchen in einem Winkel von 45° in die Erdatmosphäre eingetreten. Quelle: [34].

Die Lichtkurven eines Objekts zeigen häufig periodische Schwankungen. Vergleicht man die Lichtkurven aus verschiedenen Energiebereichen, so zeigen sich bei einigen Objekten, jedoch nicht bei allen, Korrelationen zwischen unterschiedlichen Energiebereichen. Die Ursachen dafür zu ergründen ist ein weiteres Ziel der Beobachtungen.

3.3. MARS

Zur Berechnung der Spektren und Lichtkurven sollen nur Gamma-Teilchen verwendet werden. Bei Ausrichtung auf eine starke Gamma-Quelle wie dem Krebsnebel liegt das Verhältnis der beobachteten Gamma-Ereignisse zu hadronischen Ereignissen in der Größenordnung 1:1000. Daher muss zunächst eine Klassifikation der Ereignisse durchgeführt werden, um hadronische Ereignisse auszusortieren. Dazu müssen die Daten aus der Aufnahmeeinheit der Teleskopkameras zunächst aufbereitet werden, damit sie von Klassifikationsverfahren verarbeitet werden können.

Weiterhin muss aus den Bildinformationen die Energie jedes Ereignisses abgeschätzt

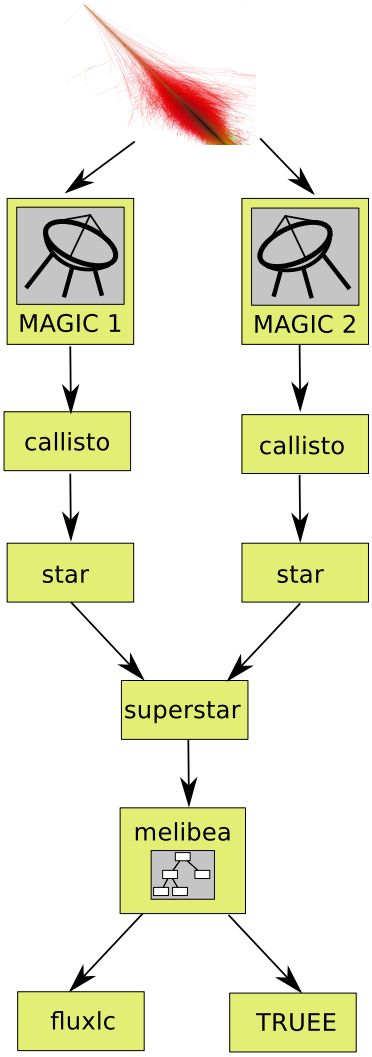


Abbildung 3: Die Analyse-Kette der MARS-Programme

werden, bevor die Lichtkurven erzeugt werden.

Verwendet wird dazu MARS[2] [26], die Standard-Analysesoftware der MAGIC-Kollaboration. Es besteht aus mehreren Programmen, die schrittweise die Rohdaten verarbeiten und letztendlich eine Klassifikation in Gamma-Teilchen und hadronische Teilchen vornehmen und die Lichtkurven und Energiespektren berechnen. Abbildung 3 zeigt schematisch die Verarbeitungsschritte.

MARS ist in C++ geschrieben und basiert auf dem ROOT-Framework [36]. Die Daten werden durchgängig im Datenformat des ROOT-Frameworks eingelesen und ausgegeben. In einem initialen Schritt konvertiert das Programm *merpp* die Teleskopdaten in das ROOT-Format. Dabei werden bereits erste hardwarenahe Vorverarbeitungsschritte vorgenommen. Anschließend werden die eigentlichen Analyseschritte durchgeführt.

[2]MAGIC Analysis and Reconstruction Software

3.3.1. callisto: Kalibrierung

Die Kamera der Teleskope besteht aus mehreren hundert Photomultipliern (PMT), welche die Pixel der Kamera darstellen. In den PMTs erzeugt einfallendes Licht eine Anzahl von Photoelektronen, die von der Aufnahmeelektronik registriert werden [3]. Je größer die Anzahl der Photoelektronen ist, desto größer ist der Ausgabewert dieses PMTs. Ziel von *callisto*[3] ist die Suche nach einer Umrechnungsfunktion von den Ausgabedaten der PMTs in die Anzahl der Photonen, die in ein PMT eingefallen sind. Dabei sollen Schwankungen in der Empfindlichkeit der PMTs ausgeglichen werden: die PMTs können sich produktionsbedingt und durch Alterungsprozesse in ihrer Empfindlichkeit unterscheiden. Daher muss eine Kalibrierung jedes einzelnen PMTs erfolgen. Bei einem *Calibration-Run* werden die PMTs einzeln von Dioden mit bekannten Eigenschaften beleuchtet, und daraus für jedes Pixel ein Korrekturfaktor berechnet.

Eine weitere Fehlerquelle bildet der Nachthimmelhintergrund (NSB). Damit wird Restlicht am Himmel bezeichnet, das in die Kamera fällt. Es stammt z. B. von Mondlicht, uninteressanten Sternen im Blickfeld der Kamera und weiteren Quellen, insbesondere Streulicht von weiteren Himmelskörpern oder auch Sonnenlicht, das an interplanetaren Staubwolken reflektiert wird. Bei einem *Pedestal Run* werden mit hoher Frequenz Aufnahmen des Himmels gemacht, ohne dass der Aufnahmemechanismus durch tatsächliche Teilchen getriggert wird. Dadurch erhält man für jede Position des Beobachtungsfensters quantitative Informationen über die Menge des Restlichts, den sogenannten Pedestal. *callisto* erhält als Eingabe einen Calibration Run und einen Pedestal Run sowie unkalibrierte Rohdaten von der Teleskopkamera und gibt die kalibrierten Daten aus.

3.3.2. star: Bildbereinigung und Merkmalsextraktion

Die Ausgabedaten von *callisto* enthalten noch Helligkeitswerte und Ankunftszeiten für jedes einzelne Pixel der Kamera. *star*[4] extrahiert daraus Merkmale, anhand derer eine Klassifikation der Aufnahmen vorgenommen werden kann. Zuvor werden die Bilder bereinigt, d. h. es werden Informationen entfernt, die höchstwahrscheinlich nicht zu dem triggernden Event gehören. Beim *Image Cleaning* wird nur die Helligkeitsinformation der Pixel verwendet, beim *Time Image Cleaning* zusätzlich die Zeitinformation.

Beim Image Cleaning [6] werden zunächst alle Pixel gesucht, deren Helligkeitswert einen bestimmten Schwellwert überschreitet. Dies sind die *Core Pixel*. Benachbarte Pixel überleben das Image Cleaning, wenn sie einen weiteren, etwas niedrigeren Schwellwert überschreiten; sie werden dann *Boundary Pixel* genannt. Alle anderen Pixel sowie isolierte Core Pixel mit weniger als drei anderen benachbarten Core Pixeln werden aus dem Bild entfernt.

Beim *Time Image Cleaning* werden benachbarte Pixel unterdrückt, wenn die Differenz der Ankunftszeiten einen bestimmten Schwellwert überschreitet [2].

Aus den so von Rauschen befreiten Bildern werden mehrere Dutzend Merkmale extrahiert, die im MAGIC-Kontext meist Parameter genannt werden. Die Bilder werden durch Ellipsen angenähert, aus deren Form und Lage die meisten Parameter extrahiert werden. Den ersten Satz Merkmale hat Hillas in [22] beschrieben. Im Laufe der Zeit

[3] **cal**ibrate **l**ight **s**ignals and **t**ime **o**ffsets
[4] **st**andard **a**nalysis and **r**econstruction

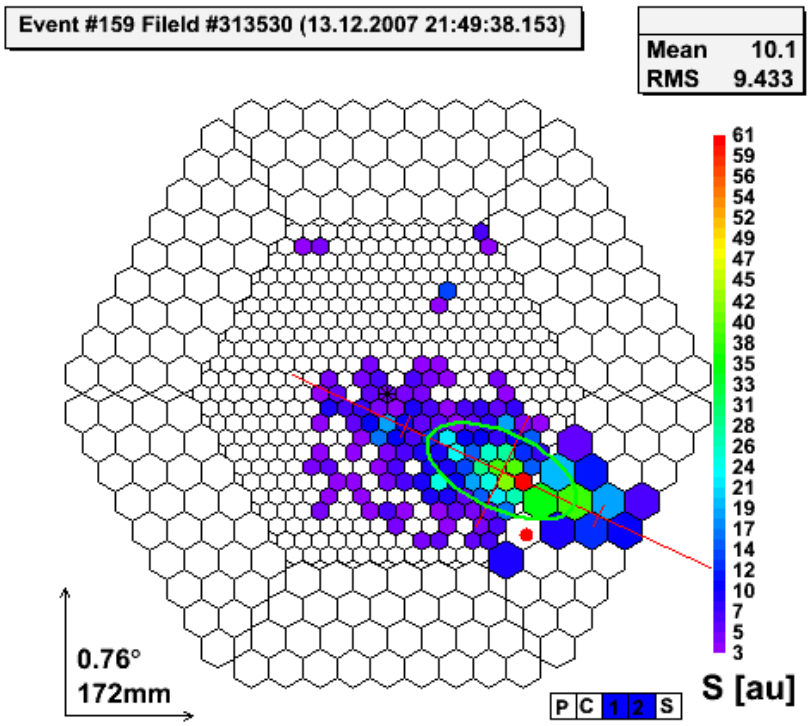


Abbildung 4: Bereinigtes Signal der MAGIC 1-Kamera, Lichtintensität, sowie die von *star* angepasste Ellipse.

wurden jedoch mehrere weitere Parameter eingeführt. Eine Erläuterung der Parameter, die standardmäßig zur Analyse verwendet werden, befindet sich im Kapitel 3.3.6.

3.3.3. superstar: Stereo-Merkmale

callisto und *star* arbeiten jeweils auf den Daten *eines* Teleskops. Bei der Analyse eines Beobachtungszeitraums werden sie also zweimal aufgerufen, einmal für die Daten jedes Teleskops. Beide Datensätze enthalten die gleichen Ereignisse, beobachtet aus verschiedenen Blickwinkeln, da die Teleskope in einem gewissen Abstand voneinander stehen. Dadurch stehen Informationen zur Verfügung, durch die eine dreidimensionale Rekonstruktion einiger Merkmale möglich ist. *superstar* ordnet die Beobachtungen beider Teleskope einander zu und berechnet diese Stereo-Merkmale. Außerdem schätzt *superstar* die Energie jedes Ereignisses ab. Dazu werden Tabellen, *lookup tables*, verwendet, die als Schlüssel die Merkmale *Size*, *Impact*, *MaxHeight* (vgl. Kapitel 3.3.6) sowie den Zenitwinkel tragen und Monte-Carlo berechnete Werte für die zugehörige Energie beinhalten.

3.3.4. melibea: Klassifikation

Anhand der Merkmale wird nun die Separation von Signal und Hintergrund bzw. die Klassifikation in Gamma-Ereignisse und hadronische Ereignisse vorgenommen. Das Programm *melibea* verwendet dazu einen Random Forest [1] (s. a. Kapitel 6.5). Das Training des Random Forest erfolgte für MAGIC 1 durch das Programm *osteria*. Für

die Stereo-Daten stehen das Programm *coach* oder das ROOT-Makro *myTrain* zur Verfügung.

In der Trainingsphase wird eine Trainingsmenge mit positiven (Gamma-) und negativen (hadronischen) Beispielen benötigt. Die negativen Beispiele werden aus sogenannten off-Daten gewonnen. Off-Daten erhält man, indem das Teleskop so ausgerichtet wird, dass sich in seinem Blickfeld keine Gamma-Quelle befindet. Folglich enthalten die beobachteten Ereignisse keine Gamma-Events aus einer Gamma-Quelle, so dass die Beobachtung als negative Beispiele für das Training verwendet werden können. Das Himmelssegment zur Gewinnung der off-Daten sollte nahe der eigentlich zu beobachtenden Quelle liegen und unter ähnlichen Licht- und atmosphärischen Verhältnissen beobachtet werden.

Eine Alternative bietet der *Wobble-Aufnahmemodus*. In diesem Modus wird das Teleskop so ausgerichtet, dass die zu beobachtende Quelle nicht im Kamerazentrum liegt, sondern etwa 0,4 ° außerhalb des Kamerazentrums. Viele Klassifikationsparameter sind stark quellpositionsabhängig. Legt man eine virtuelle Quelle als Bezugspunkt in die gegenüberliegende Hälfte der Kamera und berechnet die Parameter relativ zu dieser Position, ändern sich die Verteilungen der quellpositionsabhängigen Parameter. Für nicht-Gamma-Ereignisse spielt das keine Rolle, da diese ohnehin keine Richtungsinformation tragen. Die Verteilungen der beobachteten Gamma-Ereignisse werden stark verzerrt. Da der Anteil der Gamma-Events nur einen Anteil in der Größenordnung von einem Promille ausmacht, kann das so gewonnene Sample als negative Beispielmenge zum Training des Random Forest genutzt werden. Meist wird nicht nur eine Wobble-Position verwendet, sondern außer der Quelle drei Positionen in unterschiedlichen Quadranten der Kamerafläche (vgl. Abbildung 5) [6].

Der Wobble-Modus hat gegenüber der off-Datenbeobachtung die Vorteile, dass keine Beobachtungszeit für den Erhalt der off-Daten verloren geht und andererseits die verwendeten off-Daten aus demselben Himmelssegment stammen wie die zu analysierenden Daten und somit kein systematischer Fehler durch unterschiedliche Beobachtungspositionen eingeführt werden kann.

Die Gewinnung positiver Beispiele gestaltet sich etwas schwieriger. Für die reine Klassifikationsaufgabe wäre ein beliebiges, reines Gamma-Sample ausreichend. Da der hadronische Hintergrund isotrop aus allen Richtungen beobachtet wird, ist es jedoch unmöglich, durch Himmelsbeobachtungen reine Gamma-Samples zu erhalten. Für die Erzeugung der Energiespektren ist es außerdem unabdingbar, dass die Energie der beobachteten Teilchen abgeschätzt bzw. *rekonstruiert* werden kann. Für beobachtete Gamma-Events ist die Energie jedoch unbekannt, so dass der Algorithmus zur Energierekonstruktion nicht kalibriert werden könnte.

Daher müssen die Daten mit Hilfe von Monte-Carlo-Methoden simuliert werden. Dabei kann die Energie für jedes simulierte Teilchen vorgegeben und als Metadaten an das Beispiel angehängt werden. Weitere Informationen dazu finden sich in Kapitel 3.4.

3.3.5. FLUXLC und TRUEE: Spektrum und Lichtkurve

Nach der Klassifikation durch *melibea* berechnen FLUXLC oder TRUEE das Energiespektrum der als positiv klassifizierten Ereignisse. FLUXLC kann außerdem die Lichtkurve erzeugen (vgl. Kapitel 3.2). Darauf können dann die eingangs erwähnten Highlevel-Analysen durchgeführt werden.

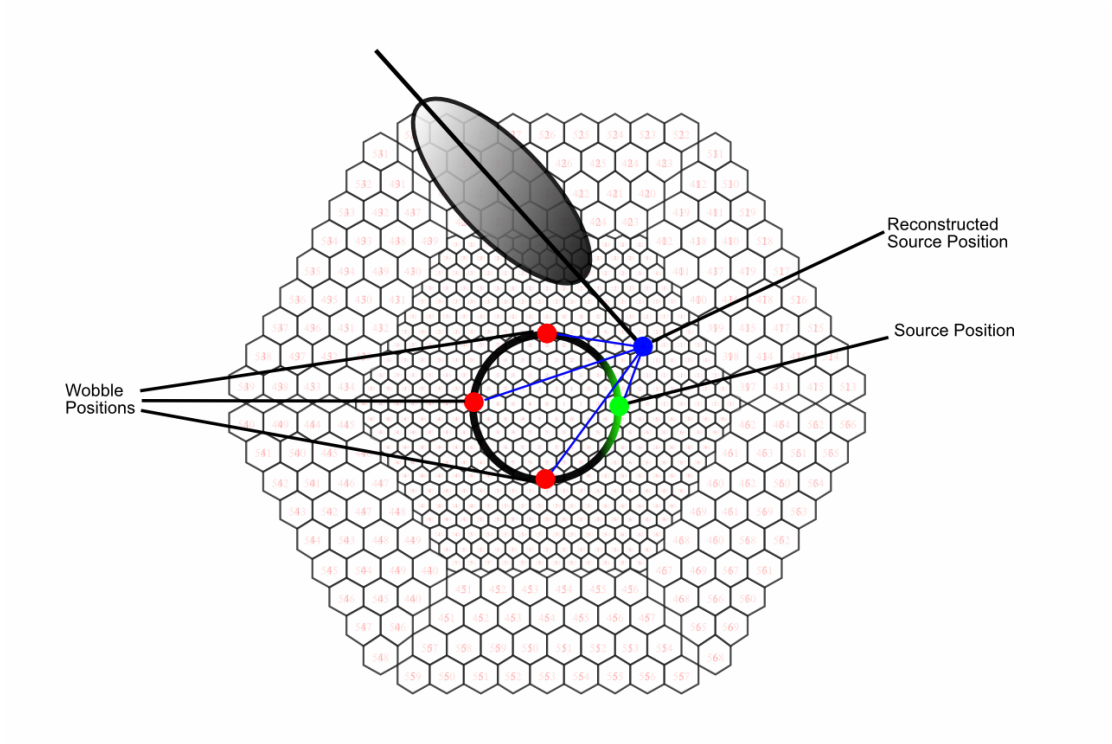


Abbildung 5: Der Wobble-Aufnahmemodus. Schematisch ist die Kamera von MAGIC 1 abgebildet. Der grüne Punkt repräsentiert die tatsächliche Beobachtungsposition, die drei roten Punkte die Bezugspunkte für den Wobble-Modus. Außerdem zu sehen ist ein Ereignis, aus dessen Eigenschaften eine Quellposition rekonstruiert wurde (blau dargestellt). Deutlich zu erkennen ist, dass sich der Parameter θ, der den Abstand der rekonstruierten Quellposition zur Beobachtungs- bzw. Wobble-Position darstellt, zwischen den Bezugspunkten stark unterscheidet.

3.3.6. Die Merkmale in der Standard-Analyse

Die Merkmale, ihre Berechnung und ihre Eigenschaften werden u. a. in [6], [15] und [22] beschrieben. Die folgende Auflistung zeigt die Merkmale, die in der Standard-Analyse zur Klassifikation verwendet werden. 1/2 im Index bedeutet, dass die Parameter separat für jedes Teleskop berechnet werden und somit zweimal im Datensatz vorkommen. P1Grad$_{1/2}$, Length$_{1/2}$, Width$_{1/2}$ und Size$_{1/2}$ werden von *star* berechnet, M$_{1/2}$Impact und MaxHeight von *superstar*.

- *P1Grad*$_{1/2}$. Gradient der Ankunftszeiten entlang der Hauptachse der Ellipse.
- *Length*$_{1/2}$. Länge der Ellipse.
- *Width*$_{1/2}$. Breite der Ellipse.
- *Size*$_{1/2}$. Summe aller Photoelektronen im Bild. Dieser Wert ist in etwa proportional zur Energie des auslösenden Teilchens.
- *M*$_{1/2}$*Impact.* Abstand vom Auftrittspunkt des Zentrums des Schauers auf dem Boden zum Teleskop.

- *MaxHeight.* Höhe des Schauer-Maximums über dem Boden.

3.4. CORSIKA: Produktion von Monte-Carlo-Datensätzen

Das Programm *CORSIKA*[5] [20] simuliert Teilchenschauer, die von hochenergetischer Gamma-Strahlung in der Erdatmosphäre erzeugt werden. Es ist nicht Teil des MARS-Paketes, sondern ein eigenständiges, am Forschungszentrum Karlsruhe entwickeltes Programm. Zur Simulation der Schauer stehen verschiedene Modelle mit unterschiedlichen Anwendungsgebieten, Genauigkeiten und Rechenaufwand zur Verfügung, unter denen der Benutzer wählen kann. Da für die Luftschauer keine analytische Beschreibung existiert, sondern es sich vielmehr um statistische Prozesse handelt, für die lediglich die Verteilungen bekannt sind oder abgeschätzt werden können, bedient sich CORSIKA der Monte-Carlo-Methode. Die Wechselwirkungen der kosmischen Teilchen mit Teilchen in der Erdatmosphäre sowie die Lebensdauer der erzeugten Teilchen wird unter Berücksichtigung der gegebenen Verteilungen ausgewürfelt [19].

CORSIKA beschreibt lediglich die Luftschauer selbst sowie die von ihnen erzeugte Cherenkov-Strahlung. Um die Daten von CORSIKA in MARS verwenden zu können, sind deshalb weitere Verarbeitungsschritte notwendig, die zuletzt zu einer simulierten Aufnahme des erzeugten Cherenkov-Lichts durch die MAGIC-Kamera in dem von MARS verwendeten Datenformat führen. Ein echter Luftschauer wird zunächst von den Spiegeln des Teleskops reflektiert und in die Kamera projiziert. Dort wandeln die PMTs das eintreffende Licht in elektronische Signale um, die wiederum vom Auslesesystem weiterverarbeitet werden. Das Programm *Reflector* simuliert die Reflektion. Die Eigenschaften des Aufnahmesystems, bestehend aus Kamera, Aufnahme- und Auslesesystem, werden von *Camera* nachgebildet. Nach Anwendung von CORSIKA, Reflector und Camera erhält man einen Datensatz im selben Format wie die echten Kameradaten.

4. Grundlagen der Analyseverfahren

Bei der Gamma-Hadron-Separation handelt es sich um ein binäres Klassifikationsproblem: Ziel ist es, eine Funktion oder ein Modell zu finden, das unbekannten Beispielen mit bekannten Merkmalen eine von zwei Klassen oder Labels zuordnet, im konkreten Anwendungsfall „Gamma“ und „nicht-Gamma“.

Die Beispiele stammen aus dem Beispielraum X und werden durch d Merkmale A_j, $j \in \{1, \ldots, d\}$ mit $d \in \mathbb{N}^+$ charakterisiert. Diese können diskrete oder kontinuierliche Werte annehmen, oder, als Spezialfall der diskreten Werte, boolesche Werte. Der Beispielraum ist dann das kartesische Produkt aller Merkmale:

$$X := A_1 \times \ldots \times A_d \tag{1}$$

In dieser Arbeit werden ausschließlich Methoden des *überwachten Lernens* verwendet. Beim überwachten Lernen wird ein Modell h auf einer Trainingsmenge $\mathbb{E}$ mit m

[5] Cosmic Ray Simulation for Kascade

Beispielen trainiert. Jedem Beispiel ist dabei ein Label $y \in Y$ zugeordnet:

$$\mathbb{E}_m = \{(x_1, y_1), \ldots, (x_m, y_m)\} \tag{2}$$

mit $(x_i, y_i) \in X \times Y$.

Gesucht wird eine Funktion oder *Hypothese* $h : X \to \mathbb{R}$, die jedem Beispiel $x \in X$ einen reellen Wert zuordnet, aus dem sich das Label ableiten lässt. Bei binären Klassifikationsproblemen wird eine Klasse als *positiv*, die andere als *negativ* bezeichnet. Die Menge der Labels ist dann $Y = \{-1, +1\}$ und die positive Klasse wird durch $y = +1$, die negative durch $y = -1$ kodiert.

Im Allgemeinen ist der Wertebereich von h die Menge aller reellen Zahlen. Das Vorzeichen sign$h(x)$ entscheidet dann, welches Label einem Beispiel zugeordnet wird. Der Betrag $|h(x)|$ gibt die *Konfidenz* der Entscheidung an: je größer die Konfidenz, desto größer die Wahrscheinlichkeit, dass das wahre Label y dem Ergebnis der Ausgabe der Entscheidungsfunktion entspricht. Allgemein könnte anstelle der Signum-Funktion auch ein Schwellwert benutzt werden, um die Entscheidung für die eine oder die andere Klasse zu treffen - die Signum-Funktion ist ein Spezialfall davon mit dem Schwellwert 0. Durch Veränderung des Schwellwerts kann ein Bias eingeführt werden. Kapitel 4.4 geht näher auf den Einfluss des Schwellwerts ein.

Solche Klassifikatoren, die Konfidenzen in Form einer reellen Zahl ausgeben, heißen *weiche Klassifikatoren*. Im Gegensatz dazu ist der Wertebereich *harter Klassifikatoren* auf $h(x) \in \{-1, 1\}$ beschränkt.

Nachdem eine Funktion h mit Hilfe einer Trainingsmenge gefunden wurde, wird sie in der Regel auf einer Testmenge validiert. Dabei geht man davon aus, dass die Daten einer Verteilung mit der Dichtefunktion $D : X \to \mathbb{R}^+$ unterliegen [35] und beide Mengen nach genau dieser Verteilung unabhängig gezogen wurden. Ist der Beispielraum X endlich oder wird eine endliche Untermenge von X verwendet, so ist die Wahrscheinlichkeit, ein bestimmtes Beispiel x zu ziehen, gegeben durch

$$\Pr_{x \sim D}[x] \quad . \tag{3}$$

Sind aus dem Kontext Variable und Verteilung ersichtlich, so kann auch durch $\Pr_D[x]$ oder $\Pr[x]$ abgekürzt werden.

Ist X nicht endlich, kann zumindest die Wahrscheinlichkeit angegeben werden, ein Element einer Teilmenge von $W \subseteq X$ zu ziehen:

$$\Pr_D[W] = \int_{x \in W} D(x)dx = \int_D I[x \in W]dx \tag{4}$$

Die Indikatorfunktion $I : \{\text{true}, \text{false}\} \to \{0, 1\}$ nimmt genau dann den Wert 1 an, wenn ihr Argument wahr ist.

4.1. Klassifikationsregeln

Zur Klassifikation können beispielsweise Klassifikationsregeln der Form $A \to C$ verwendet werden [35]. Eine solche Regel besagt, dass aus der Vorbedingung A die Bedingung C folgt. Bei Klassifikationsaufgaben bedeutet C zumeist, dass ein Beispiel ein

bestimmtes Label trägt, während A bestimmte Eigenschaften der Merkmale kodiert. Im einfachsten Falle gibt A an, ob ein numerisches Merkmal einen Grenzwert über- oder unterschreitet oder ein diskretes Merkmal einen bestimmten Wert annimmt. In komplexeren Fällen könnte A auch auf die Ausgabe eines anderen Klassifikationsmodells zurückgreifen. Eine solche Regel hätte die Form $h(x) > 0 \rightarrow y = 1$. Anschaulich bedeutet das: wenn das gelernte Modell ein Beispiel als Gamma klassifiziert, dann impliziere dies, dass das Beispiel ein Gamma ist.

Formal definiert Scholz Klassifikationsregeln folgendermaßen:

Definition 4.1 (Klassifikationsregel)**.** *Eine Klassifikationsregel besteht aus einer Bedingung A, die eine Konjugation atomarer Aussagen $A_1, \ldots, A_k$ ist, und einer Folgerung C, welche den Wert eines Zielattributs vorhersagt. Wenn die Bedingung für ein Beispiel erfüllt ist, heißt die Regel* anwendbar *und das Beispiel* abgedeckt*. Wenn außerdem C erfüllt ist, dann heißt die Regel* korrekt*.*

Offensichtlich können mit solchen Regeln lediglich harte Klassifizierer modelliert werden, nicht aber solche mit Konfidenzen oder anderen reellwertigen Ausgaben. Dieser Aspekt wird in Kapitel 12 näher beleuchtet.

Die *Extension* $\mathrm{Ext}(A) \subseteq X$ bzw. $\mathrm{Ext}(C) \subseteq X$ ist die Menge genau der Beispiele, für die A bzw. C erfüllt ist. Weiterhin entspricht $\Pr_D[A]$ der Wahrscheinlichkeit, dass ein zufällig nach der Verteilung D gezogenes Beispiel A erfüllt bzw. in $\mathrm{Ext}(A)$ enthalten ist.

4.2. Bewertung von Klassifikationsregeln

Oft müssen unterschiedliche Klassifikatoren miteinander verglichen und bewertet werden. Grundlegende Gütemaße und Methoden für die Bewertung von Modellen für binäre Klassifikationsprobleme werden in den folgenden beiden Abschnitten vorgestellt. o. B. d. A. wird davon ausgegangen, dass C die Zugehörigkeit zur positiven und $\overline{C}$ zur negativen Klasse bedeutet. Weiterhin wird die positive Klasse durch das Label $+1$, die negative durch das Label -1 kodiert.

4.2.1. Die Konfusionsmatrix

Wenn ein Beispiel $(x, y) \in X \times Y$ durch die Funktion $h : X \rightarrow Y$, $h(x) = \hat{y}$ klassifiziert wird, so können bei der binären Klassifikation genau vier verschiedene Fälle eintreten. Zunächst kann das wahre Label y positiv oder negativ sein. Die Vorhersage $\hat{y}$ kann ebenfalls positiv oder negativ sein. Tatsächlich positive Beispiele, die auch als positiv klassifiziert werden, heißen *True Positive* (TP). Werden sie inkorrekterweise als negativ klassifiziert, heißen sie *False Negative*. Analog heißen echt negative negativ klassifizierte Beispiele *True Negative* (TN) und fälschlich als positiv klassifizierte Beispiele *False Positive* (FP). Tabelle 1 veranschaulicht dieses Schema.

Die Menge aller True Positives wird ebenfalls mit TP bezeichnet, analog wird auch FP, TN und FN verwendet. Diese vier Mengen partitionieren den Beispielraum.

Die (gewichteten) Kardinalitäten werden durch p, $\overline{p}$, $\overline{n}$ und n angegeben. Durch den Querstrich ist kodiert, welchen Wert das wahre Label y trägt: ein Querstrich bedeutet $y = -1$, fehlt er, so ist $y = +1$. Die Buchstaben p und n kodieren die Vorhersage $\hat{y} = h(x)$ des Klassifikators. n bedeutet $h(x) = -1$, p steht für $h(x) = +1$.

	$y = +1$; True C	$y = -1$; True $\overline{C}$
$h(x) = +1$; Pred C	TP: $A \wedge C$ $\lvert TP\rvert = p$	FP: $A \wedge \overline{C}$ $\lvert FP\rvert = \overline{p}$
$h(x) = -1$; Pred $\overline{C}$	FN: $\overline{A} \wedge C$ $\lvert FN\rvert = n$	TN: $\overline{A} \wedge \overline{C}$ $\lvert TN\rvert = \overline{n}$
	$P = p + n$	$N = \overline{p} + \overline{n}$

Tabelle 1: Veranschaulichung der Konfusionsmatrix

Die vier Werte p, $\overline{p}$, $\overline{n}$ und n werden durch

$$p = \sum_{i:x_i \in \text{TP}} w_i \tag{5}$$

$$\overline{p} = \sum_{i:x_i \in \text{FP}} w_i \tag{6}$$

$$n = \sum_{i:x_i \in \text{FN}} w_i \tag{7}$$

$$\overline{n} = \sum_{i:x_i \in \text{TN}} w_i \tag{8}$$

berechnet. w_i gibt dabei das Gewicht des Beispiels (x_i, y_i) an.

Zusammen bilden sie die sogenannte *Konfusionsmatrix* (vgl. bspw. [10]). Diese ist die Basis für viele Gütemaße zur Bewertung von binären Klassifikatoren.

Unter der zuvor getroffenen Annahme, dass C die positive Klasse impliziert, kann für die vier Partitionen ausgesagt werden, ob A und C erfüllt sind oder nicht. Wird ein Beispiel (x, y) mit $y = -1$ von der Regel $A \rightarrow C$ als positiv vorhergesagt, so ist offenbar A erfüllt, so dass die Regel fälschlicherweise C impliziert. Da das wahre Label jedoch negativ ist, ist C in Wahrheit nicht erfüllt. Das Beispiel wird von der Regel also in die Menge FP eingeordnet, in der A erfüllt ist, C aber nicht. Auch dies ist in Tabelle 1 veranschaulicht.

4.2.2. Gütemaße für Klassifikationsregeln

Zur Bewertung von Klassifikationsregeln und anderen Klassifikatoren existieren mehrere Gütemaße. In diesem Kapitel werden die für diese Arbeit relevanten Gütemaße vorgestellt. Die wahrscheinlichkeitsbasierten Formulierungen sind [35] entnommen. In der Praxis werden sie mit Hilfe der Konfusionsmatrix abgeschätzt.

Definition 4.2 (Accuracy)**.** *Die* Accuracy *einer Regel $r = A \rightarrow C$ ist definiert als*

$$Acc_D(r) := Pr_D[A, C] + Pr_D[\overline{A}, \overline{C}] \quad . \tag{9}$$

Das ist die Wahrscheinlichkeit, dass ein zufällig gezogenes Beispiel von der Regel korrekt klassifiziert wird. Analog kann die Wahrscheinlichkeit definiert werden, dass das Beispiel falsch klassifiziert wird. Die *Fehlklassifikationsrate* ist definiert als $1 - \text{Acc}(r)$.

Definition 4.3 (Precision). *Die* Precision *einer Regel* $r = A \to C$ *ist definiert als*

$$Prec_D(r) := Pr_D[C|A] \quad . \tag{10}$$

Das ist die Wahrscheinlichkeit, dass ein als positiv klassifiziertes Beispiel auch das wahre Label „positiv“ trägt, also $\Pr_{(x,y)\sim D}[y = h(x)]$. Offensichtlich ist die Precision stark abhängig von den a-priori-Wahrscheinlichkeiten der Labels.

Die Precision kann analog auch für die übrigen Partitionen berechnet werden.

Definition 4.4 (Recall). *Der* Recall *einer Regel* $r = A \to C$ *ist definiert als*

$$Recall_D(r) := Pr_D[A|C] \quad . \tag{11}$$

Das ist die Wahrscheinlichkeit, dass ein tatsächlich positives Beispiel auch positiv klassifiziert wird. Wie in Kapitel 4.4 gezeigt wird, stehen Recall und Precision in einem sehr engen Verhältnis.

Definition 4.5 (Lift). *Das* Lift-Kriterium *einer Regel* $r = A \to C$ *ist definiert als*

$$Lift_D(r) := \frac{Pr_D[C|A]}{Pr_D[C]} \quad . \tag{12}$$

Das Lift-Kriterium ist ein Maß zur Bewertung von Klassifizierern bzw. den zuvor eingeführten Regeln. Es berechnet die Wahrscheinlichkeit, in der Menge der als positiv klassifizierten Beispiele tatsächlich ein echtes positives Beispiel zu finden, normiert durch die a-priori-Wahrscheinlichkeit der positiven Klasse, oder anders ausgedrückt, eine Art normierter Precision. Durch diese Normierung ist Lift invariant gegen Änderungen der a-priori-Wahrscheinlichkeiten.

Im ROC-Raum (vgl. Kapitel 4.4) verändern sich die Äquivalenzlinien bei Änderung der a-priori-Wahrscheinlichkeiten nicht.

Definition 4.6 (Coverage). *Die* Coverage *einer Regel* $r = A \to C$ *ist definiert als*

$$Cov_D(r) := Pr_D[A] \quad . \tag{13}$$

Das ist die Wahrscheinlichkeit, dass ein zufällig nach der Verteilung D gezogenes Beispiel von der Regel als positiv klassifiziert wird.

Definition 4.7 (Bias). *Der* Bias *einer Regel* $r = A \to C$ *ist definiert als*

$$Bias_D(r) := Pr_D[C|A] - Pr_D[C] \quad . \tag{14}$$

Der Bias ist also die absolute Abweichung zwischen der Precision einer Regel und zufälligem Raten.

4.2.3. Schätzer für Gütemaße

Da die tatsächlichen Wahrscheinlichkeiten, die eigentlich zur Berechnung der meisten Gütemaße benötigt würden, in der Praxis unbekannt sind, werden sie aus der Konfusionsmatrix abgeschätzt. Dieses Unterkapitel listet die Schätzer für die zuvor definierten

Kriterien.

$$\text{Acc}(r) \approx \frac{p + \overline{n}}{m} \tag{15}$$

$$\text{Prec}(r) \approx \frac{p}{p + \overline{p}} \tag{16}$$

$$\text{Recall}(r) \approx \frac{p}{p + n} \tag{17}$$

$$\text{Lift}(r) \approx \frac{\text{Prec}_D(r)}{p + n} \tag{18}$$

$$\text{Cov}(r) \approx \frac{p + \overline{p}}{m} \tag{19}$$

$m = p + \overline{n} + \overline{p} + n$ gibt dabei die Anzahl aller Beispiele an.

4.3. Stratifizierung

Nach [35] werden durch die *Stratifizierung* einer Verteilung die a-priori-Wahrscheinlichkeiten der Klassen verändert. Das Zielverhältnis ist beliebig, wird es jedoch nicht explizit angegeben, so soll jede Klasse die gleiche Wahrscheinlichkeit haben.

Definition 4.8 (Stratifizierung)**.** *Sei $D : X \times Y \to \mathbb{R}^+$ eine Verteilung. Dann ist die* stratifizierte Verteilung *D' mit gleichen a-priori-Wahrscheinlichkeiten definiert als*

$$D'(x, y) := \frac{D(x, y)}{|Y| \cdot Pr_{(x',y')\sim D}[y = y']} \quad . \tag{20}$$

In der Praxis kann Stratifizierung durch Gewichtung der Beispiele oder durch Ziehen von Beispielen erfolgen. Wird mit Beispielgewichten gearbeitet, entspricht die Stratifizierung einer Umgewichtung der Beispiele. Die Summe der Gewichte aller positiven Beispiele vor der Umgewichtung sei P, die der negativen Beispiele N, analog nach der Umgewichtung P' und N'. Erfolgt die Stratifizierung im Kontext des Boostings muss die Bedingung $P \cdot N = P' \cdot N'$ beachtet werden, da beim Boosting aus dem Produkt $P \cdot N$ ein Fehlermaß abgeleitet werden kann [35].

In jedem Fall entspricht die Stratifizierung einer Änderung des Verhältnisses P/N um einen reellen Faktor c

$$w'(x, y) = w(x, y) \cdot \begin{cases} \sqrt{c} & \text{für } y = +1 \\ \frac{1}{\sqrt{c}} & \text{für } y = -1 \end{cases} \quad . \tag{21}$$

Um die Daten zu stratifizieren muss

$$c = N/P \tag{22}$$

gewählt werden. Diese Wahl minimiert zugleich das Gesamtgewicht $P' + N' = \sqrt{c}P + N/\sqrt{c}$, wie durch Nullsetzen der Ableitung und Auflösung nach c leicht gezeigt werden kann.

4.4. ROC-Analyse

Die *ROC-Analyse* wie z. B. in [10] beschrieben, ist ein weiteres Instrument zur Analyse von Klassifikatoren. Außerdem können die zuvor vorgestellten Gütemaße mit Hilfe der ROC-Analyse untersucht und Unterschiede aufgezeigt werden. Auch kann gezeigt werden, dass manche Gütemaße robust gegen bestimmte Verschiebungen in der Verteilung der Beispielmenge sind, während andere dadurch stark beeinflusst werden. Die ROC-Analyse kann außerdem bei der Suche nach Schwellwerten helfen, die weiche Klassifikatoren in harte Klassifikatoren umwandeln.

4.4.1. Die Definition des ROC-Raums

In Kapitel 4.2.1 wurde die Konfusionsmatrix eingeführt (vgl. Tabelle 1). Neben den zuvor vorgestellten Gütemaßen lassen sich weitere Größen daraus ableiten, insbesondere die *True Positive Rate* (TPr) und die *False Positive Rate* (FPr):

Definition 4.9 (True Positive Rate)**.** *Die* True Positive Rate *ist definiert als*

$$TPr(h(x)) := Pr[sign(h(x) = +1)|(x,y) : y = +1] \quad . \tag{23}$$

Definition 4.10 (False Positive Rate)**.** *Die* False Positive Rate *ist definiert als*

$$FPr(h(x)) := Pr[sign(h(x) = +1)|(x,y) : y = -1] \quad . \tag{24}$$

Es gilt

$$\mathrm{TPr}(h(x)) \approx \frac{p}{p+n} \tag{25}$$

$$\mathrm{FPr}(h(x)) \approx \frac{\overline{p}}{\overline{p}+\overline{n}} \tag{26}$$

und es fällt auf, dass die True Positive Rate gleich dem Recall ist:

$$\mathrm{TPr}(h(x)) = \mathrm{Recall}(h(x)) \tag{27}$$

Ein weiteres Gütemaß ist die *Spezifizität*. Sie wird analog zum Recall berechnet, arbeitet jedoch mit den Beispielen der negativen Klasse. Sie gibt also die Wahrscheinlichkeit an, dass ein zufällig gezogenes Beispiel (x, y) mit $y = -1$ von einer Hypothese tatsächlich als negativ klassifiziert wird:

Definition 4.11 (Spezifizität)**.** *Die* Spezifizität *ist definiert als*

$$Specificity(h(x)) = Pr[sign(h(x) = -1)|(x,y) : y = -1] \quad . \tag{28}$$

Die Spezifizität kann durch

$$\mathrm{Specificity}(h(x)) \approx \frac{\overline{n}}{\overline{n}+\overline{p}} \tag{29}$$

angenähert werden.

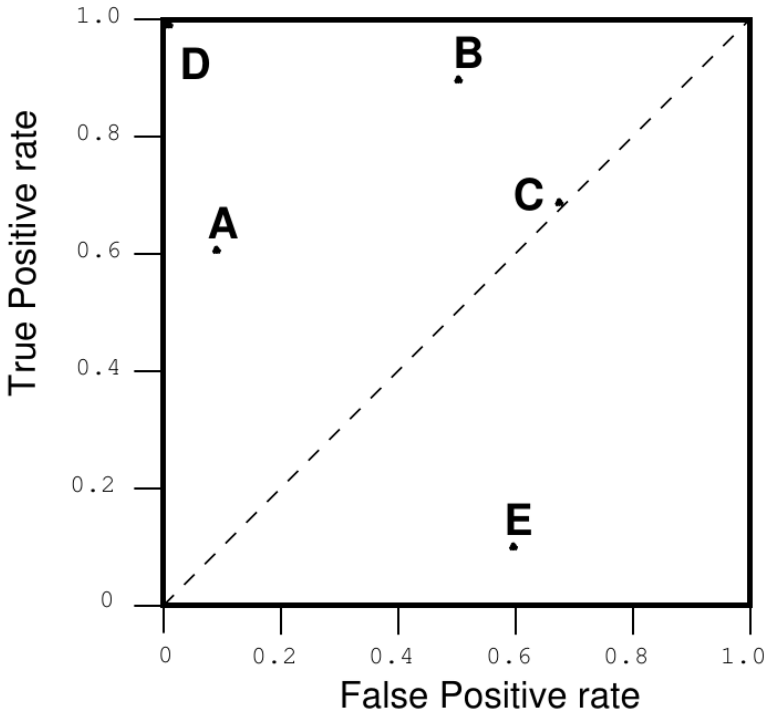


Abbildung 6: ROC-Diagramm mit harten Klassifikatoren. Jeder Klassifikator belegt genau einen Punkt im ROC-Raum. Aus [10].

Es ist

$$\mathrm{FPr}(h(x)) = 1 - \mathrm{Specificity}(h(x)) \quad . \tag{30}$$

TPr und FPr können reelle Werte in $[0, 1]$ annehmen. Der von ihnen aufgespannte zweidimensionale Raum heißt *ROC-Raum*, wobei die True Positive Rate auf die y-Achse und die False Positive Rate auf die x-Achse gelegt wird (vgl. Abbildung 6).

4.4.2. Ausgezeichnete Punkte und Strecken im ROC-Raum

Ein perfekter Klassifikator führt eine optimale Klassifikation durch, d. h. er klassifiziert alle positiven Beispiele als positiv und alle negativen als negativ: $\forall(x, y) \in X \times Y : h(x) = y$. Somit ist die True Positive Rate des perfekten Klassifikators 1 und seine False Positive Rate 0. Dieser Klassifikator wird im ROC-Raum auf den Punkt $(0, 1)$ abgebildet. Analog entspricht der Punkt $(1, 0)$ einem Klassifikator, der nie die richtige Vorhersage trifft, d. h. $\forall(x, y) \in X \times Y : h(x) \neq y$.

Klassifikatoren, die auf der Diagonalen liegen, die diese beiden Punkte trennt und von $(0, 0)$ nach $(1, 1)$ verläuft, entsprechen zufälligen Klassifikatoren, die jedes Beispiel mit einer festen Wahrscheinlichkeit c als positiv vorhersagen: $\forall(x) \in X : \Pr[h(x) = +1] = c$ [10]. Abhängig von c bewegt sich der Klassifikator auf der Diagonalen: je größer c, desto weiter nähert sich der Klassifikator an den Punkt $(1, 1)$ an.

Klassifikatoren, die oberhalb der Diagonalen liegen, liefern bessere Ergebnisse als zufälliges Raten, liegt ein Klassifikator unterhalb der Diagonalen, liefert er schlechte-

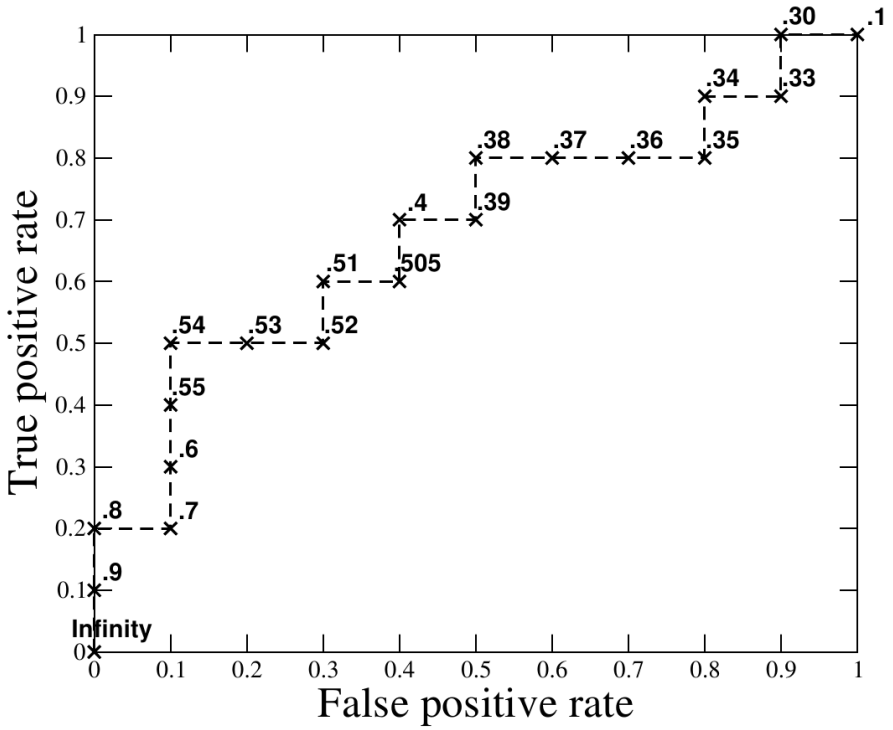


Abbildung 7: ROC-Kurve eines weichen Klassifikators. Aus [10].

re Ergebnisse als zufälliges Raten. Dennoch liefert er nutzbare Information: invertiert man seine Vorhersage, d. h. $h'(x) = -h(x)$, so wird seine Repräsentation im ROC-Raum an der Diagonalen gespiegelt, so dass er in der oberen Hälfte des Raumes zu liegen kommt.

4.4.3. Schwellwertbasierte Kurven im ROC-Raum

Abbildung 6 zeigt lediglich einzelne Punkte, die Klassifikatoren mit diskreten Ausgaben mit $h(x) \in \{0, 1\}$ erzeugen. Viele Klassifikatoren geben jedoch reelle Werte aus. Ein optimaler Klassifikator ordnet jedem positiven Beispiel einen höheren Wert zu als jedem negativen Beispiel, und es gibt einen Schwellwert s, der einen perfekten harten Klassifikator $h_s(x) : X \to \{-1, +1\}$ erzeugt. Meist gibt es jedoch Überschneidungen, und mit einer gewissen Wahrscheinlichkeit werden einige negative Beispiele höher gerankt als positive:

$$\Pr[h(x_1) > h(x_2) | (x_1, +1), (x_2, -1) \in X \times Y] \geq 0 \tag{31}$$

Variiert man den Schwellwert s, so ändert sich die Ausgabe von $h_s(x)$, und unterschiedliche $h_s(x)$ ergeben unterschiedliche Konfusionsmatritzen und somit unterschiedliche True Positve Rates und False Positive Rates. Es ergeben sich also für unterschiedliche Schwellwerte unterschiedliche Punkte im ROC-Raum, die zusammengenommen eine sogenannte *ROC-Kurve* ergeben.

Aus einem weichen Klassifikator lässt sich also eine ROC-Kurve erzeugen, und jeder Punkt auf der Kurve entspricht einem *Operation Point* bzw. einem bestimmten Schwell-

wert s. Abbildung 7 zeigt eine solche Kurve.

Wird die ROC-Kurve durch Evaluierung des Klassifikators auf einer Beispielmenge mit m Beispielen (x, y) erzeugt, so ist diese Kurve bei genauerer Betrachtung nicht kontinuierlich, sondern besteht aus höchstens m diskreten Punkten: der Klassifikator ordnet jedem Beispiel x einen reellen Wert zu, welcher wiederum einem Punkt auf der ROC-Kurve entspricht. Häufig besteht die Kurve sogar aus weit weniger Punkten, da der Klassifikator mehreren Beispielen denselben Wert zuordnen kann.

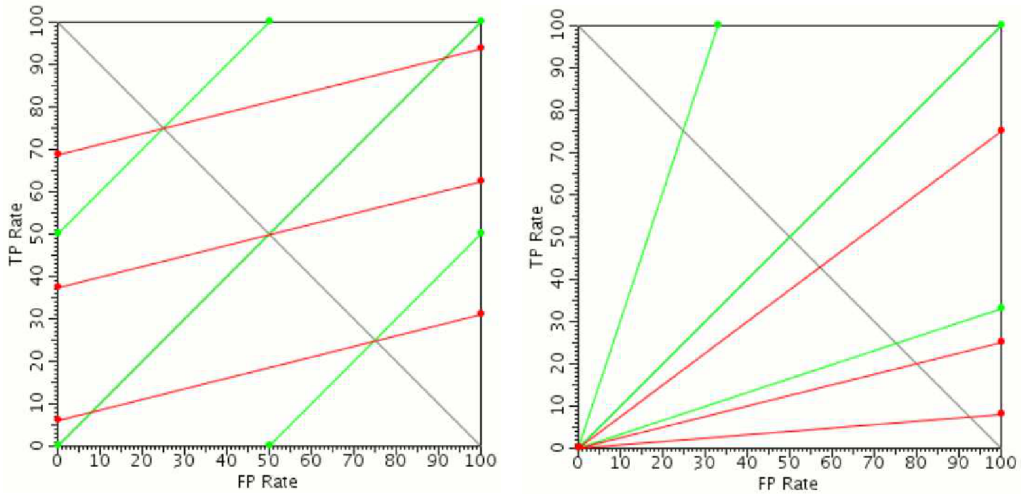
Dass ein Operator vielen Beispielen denselben Wert zuweist, ist nicht ungewöhnlich [10]. Um beispielsweise bei einem Decision Tree eine Konfidenz für ein unbekanntes Beispiel abzuleiten, wird das Klassenverhältnis der Trainingsmenge in dem Blatt betrachtet, in das das unbekannte Beispiel einsortiert wird. Da für einen Baum ein Blatt eine atomare Einheit darstellt, sind alle Beispiele, die in ein Blatt fallen, für den Baum ununterscheidbar, und sie erhalten somit alle dieselbe Konfidenz. Je mehr Beispiele dieselbe Konfidenz erhalten, desto weiter liegt der entsprechende Punkt im ROC-Raum von dem nächstniedrigeren Punkt entfernt: sobald der Schwellwert einen Punkt überquert, dem einige Beispiele zugeordnet sind, so wechseln diese die Partitionen in der Konfusionsmatrix. Dadurch entstehen Sprünge in der True Positive Rate und False Positive Rate. Je mehr Beispiele dem Punkt zugeordnet sind, desto größer ist dieser Effekt.

Durch die Wahl des Schwellwerts kann man den Trade-Off zwischen Precision und Recall einstellen: in der Regel verringert sich der Recall eines Operators, wenn seine Precision größer wird, und umgekehrt. Im ROC-Raum lässt sich das so erklären, dass ein Operator weiter links unten eher konservative Entscheidungen trifft, also nur dann die positive Klasse vorhersagt, wenn die Wahrscheinlichkeit für eine falsche Vorhersage sehr gering ist. Somit werden die meisten negativen Beispiele zuverlässig korrekt klassifiziert, jedoch werden auch viele positive Beispiele mit ausgefiltert, TPr ist klein [10]. Das entspricht einer hohen Precision bei einem kleinen Recall. Nähert man sich hingegen dem Punkt $(1, 1)$, so erkennt der Klassifikator tendenziell viele positive Beispiele korrekt, klassifiziert aber auch viele negative Beispiele falsch. Wie beschrieben läge der optimale Klassifikator im Punkt $(0, 1)$. Da die Punkte im ROC-Raum jedoch nicht beliebig ausgewählt werden können, sondern durch den Schwellwert nur entlang der ROC-Kurve verschoben werden können, besteht immer ein Trade-Off zwischen Precision und Recall.

4.4.4. Verhalten von Gütemaßen bei unterschiedlichen Klassenverhältnissen

Im Anwendungsfall der Gamma-Separation ist das Verhältnis P/N zwischen der Anzahl Gammas und anderer Teilchen sehr klein und bei echten Daten nicht genau bekannt und außerdem von Quelle zu Quelle unterschiedlich. Viele im maschinellen Lernen häufig verwendete Gütemaße sind jedoch in hohem Maße abhängig von eben diesem Verhältnis.

P entspricht der Summe der linken Spalte in der Konfusionsmatrix, N der rechten Spalte. Somit ändert sich jedes Gütemaß, das Werte aus *beiden* Spalten der Konfusionsmatrix verwendet, mit dem Verhältnis P/N [10]. Darunter fallen insbesondere auch Accuracy und Precision (vgl. Kapitel 4.2.2).



(a) (b)

Abbildung 8: Isometrielinien im ROC-Raum für ein Klassenverhältnis von $P/N = 1$ (grün) und $P/N = 4$ (rot). (a) zeigt die Isometrielinien für die Accuracy, (b) für die Precision. Aus [35].

Die Form der ROC-Kurven dagegen ist invariant gegen Änderungen des Klassenverhältnisses, da die True Positive Rate und die False Positive Rate jeweils nur auf einer Spalte der Konfusionsmatrix definiert sind.

Für jedes Gütemaß existieren im ROC-Raum Isometrielinien, d. h. Linien, auf denen der Wert des Gütemaßes konstant ist. Je nachdem, wie sich die jeweiligen Isometrielinien bei Änderungen des Klassenverhältnisses unterscheiden, werden Gütemaße entweder als *schwach Skew-sensitiv*, *stark Skew-sensitiv* oder *Skew-unabhängig* bezeichnet [11].

Ein Gütemaß heißt *Skew-unabhängig*, wenn sich seine Isometrielinien bei einer Änderung des Klassenverhältnisses nicht verändern.

Ein Gütemaß heißt *schwach Skew-sensitiv*, wenn sich bei einer Änderung des Klassenverhältnisses die Form der Isometrielinine nicht verändert, wohl aber der Wert einer Isometrielinie.

Ein Gütemaß heißt *stark Skew-sensitiv*, wenn sich bei einer Änderung des Klassenverhältnisses die Form der der Isometrielinien verändert, wie Beispielsweise eine Änderung der Steigung von Geraden.

Abbildung 8 zeigt die Isometrielinien von Accuracy und Precision. Die Steigung der Linien ergibt sich für die Accuracy direkt aus dem umgekehrten Klassenverhältnis N/P, für die Precision aus dem Quotienten TPr/FPr am jeweiligen Operation Point [11]. Die Accuracy ist also stark Skew-sensitiv, die Precision schwach Skew-sensitiv.

4.4.5. Bewertung und Selektion von Modellen bei unbekanntem Klassenverhältnis

Wurden mehrere weiche Klassifikatoren trainiert, stellt sich die Aufgabe, daraus einen optimalen Klassifikator auszuwählen, mit dem unbekannte Beispiele letztlich klassifiziert werden. Ist das Klassenverhältnis der Daten, auf die der Klassifkator angewendet

wird, bekannt, so wird für jeden Klassifikator ein optimaler Operation Point bezüglich eines Kriteriums ermittelt, und die Klassifikationsgüte der daraus resultierenden harten Klassifikatoren ermittelt. Das Modell mit der höchsten Güte wird verwendet.

Ist das Klassenverhältnis der Daten unbekannt, so muss das verglichene Gütemaß Skew-unabhängig sein. Wird nämlich ein Skew-sensitives Maß verwendet, kann es passieren, dass eine Änderung des Klassenverhältnisses die Güte an einem Operation Point beeinflusst. Skew-sensitve Gütemaße haben somit keine Aussagekraft für die Performance des Klassifikators auf dem unbekannten Klassenverhältnis.

Im MAGIC-Experiment werden weiche Klassifikatoren auf einem festen Klassenverhältnis trainiert, während das Klassenverhältnis in der Anwendung auf echten Daten unbekannt ist. Hier ist es üblich, den Operation Point so zu wählen, dass ein bestimmter Recall nicht unterschritten wird [23]. Da der Recall Skew-unabhängig ist, ist dies eine valide Möglichkeit zur Wahl des Operation Point und des daraus resultierenden Schwellwerts.

Zum Vergleich unterschiedlicher Klassifikatoren wird die Precision bei einem festen Klassenverhältnis und festem Recall genutzt. Dies ist jedoch problematisch, da die Precision Skew-sensitiv ist. Außerdem kann der Recall nicht immer exakt eingestellt werden: wie zuvor beschrieben, wird nämlich häufig mehreren Beispielen von einem Klassifikator derselbe Wert zugeordnet, so dass sie durch einen Schwellwert nicht diskriminierbar sind. Das hat zur Folge, dass der Recall beliebig große Sprünge macht, wenn viele Beispiele denselben Wert erhalten, und die Operation Points weit auseinander liegen. Das hat aber auch Einfluss auf die zugehörige Precision, so dass ein Vergleich der Precision allein wenig Aussagekraft hat. Kapitel 11 geht näher darauf ein.

Besser geeignet ist die im folgenden Abschnitt beschriebene, Skew-unabhängige *Area under the ROC-Curve*.

4.4.6. Area under the ROC-Curve

Die ROC-Kurven selbst sind invariant gegen Änderungen des Klassenverhältnisses. Sie sind somit ein gutes Mittel, um Klassifikatoren zueinander in Relation zu stellen, die auf unterschiedlichen Klassenverhältnissen gut arbeiten sollen. Im Gegensatz zu den bisher vorgestellten Gütemaßen sind die Kurven jedoch zweidimensionale Objekte und keine skalaren Größen, was einen direkten Vergleich schwieriger macht. Eine Möglichkeit, die Eigenschaften einer ROC-Kurve in einem Skalar zusammenzufassen, ist die Fläche unter der ROC-Kurve, bzw. die *Area under the ROC-Curve* (AUC) [4, 17]. Dieses Maß ist ebenso wie die eigentliche Kurve Skew-unabhängig. Zur Berechnung wird außerdem kein zuvor festgelegter Schwellwert benötigt.

Statistisch betrachtet gibt AUC die Wahrscheinlichkeit an, mit der der Klassifikator einem positiven Beispiel einen höheren Wert zuweist als einem negativen Beispiel:

$$\mathrm{AUC}(h) = \Pr[h(x_1) > h(x_2) | (x_1, +1), (x_2, -1) \in X \times Y] \tag{32}$$

Die Area under the ROC-Curve garantiert nicht, dass ein Klassifikator mit höherer AUC an jedem Operation Point besser klassifiziert als einer mit geringerer AUC, sondern dies nur mit höherer Wahrscheinlichkeit tut. Abbildung 9 zeigt ein Beispiel mit den beiden Klassifikatoren A und B. B hat die größere AUC. Für Werte von $\mathrm{FPr} > 0,6$ liefert dennoch A bessere Werte.

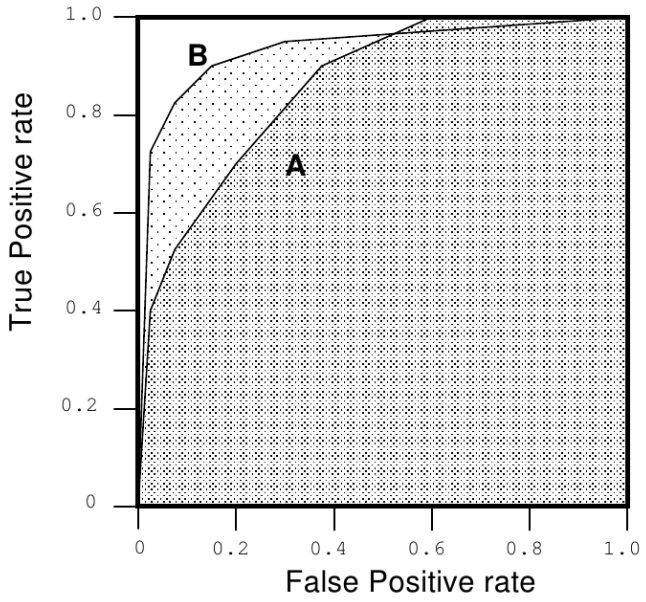
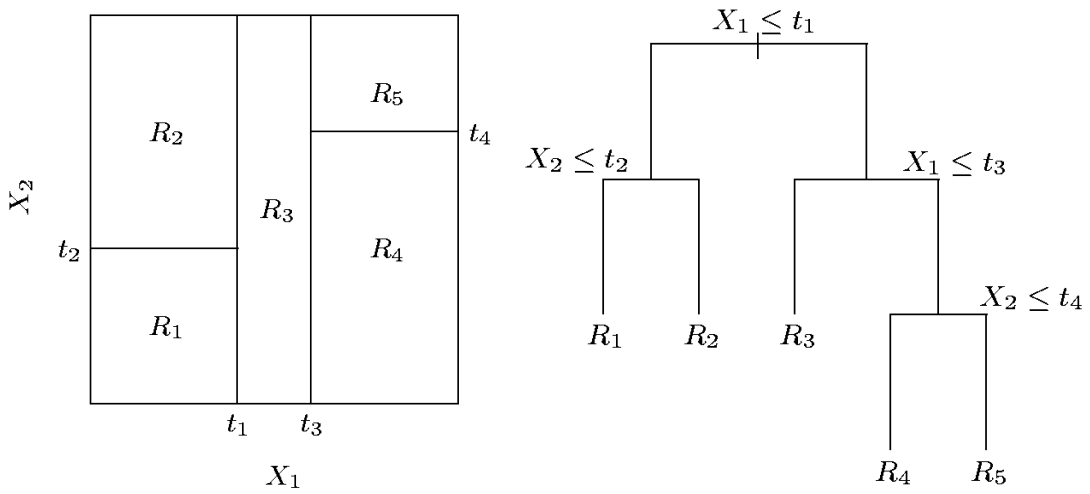


Abbildung 9: Beispiele für unterschiedliche ROC-Kurven. Aus [10].



(a) Partionen des Beispielraums (b) Baumdarstellung

Abbildung 10: Veranschaulichung eines Entscheidungsbaums. Aus [18].

Wegen der Invarianz gegenüber Skew-Änderungen wird in dieser Arbeit weitestgehend die Area under the ROC-Curve zum Vergleich von Operatoren verwendet.

5. Klassifikationsverfahren

5.1. Entscheidungsbäume

Entscheidungsbäume sind Klassifikatoren, die den Beispielraum in eine Menge von Rechtecken partitionieren, indem sie iterativ Schnitte an unterschiedlichen Merkma-

len durchführen [18]. Abbildung 10(a) veranschaulicht die Partitionierung an einem beispielhaften Datensatz mit zwei Merkmalen. 10(b) zeigt die entsprechenden Schnitte in Baumdarstellung. Die Blätter R_i des Baums, die gleichzeitig den Partitionen des Beispielraums entsprechen, sind für das Baummodell atomar, d. h. Elemente einer einzelnen Menge sind für den Baum nicht unterscheidbar. Somit werden bei der Anwendung des Baummodells alle unbekannten Beispiele, die in dieselbe Partition fallen, mit derselben Klasse vorhergesagt. Dies ist in der Regel die Klasse, die beim Training die Mehrheitsklasse in der jeweiligen Partition stellt. Soll anstelle einer harten Klassifikation eine weiche Klassifikation erfolgen, so kann der Anteil der Mehrheitsklasse an allen Beispielen in der Partition ausgegeben werden. Dies entspricht der Precision der Mehrheitsklasse in dem Blatt.

Je stärker der Beispielraum partitioniert wird, desto reiner können die Blätter werden, so dass der Trainingsfehler klein wird. Andererseits besteht dann die Gefahr von Overfitting, so dass der Baum schlecht generalisierbar ist. Beim Training eines Baumlerners ist dieser Trade-off zwischen Trainingsfehler und Generalisierbarkeit zu berücksichtigen. Häufig wird so lange trainiert, bis der Fehler auf der Trainingsmenge verschwindet, und anschließend werden beim sogenannten *Pruning* nach einer Heuristik Blätter zusammengefasst.

Zwei weitere Probleme stellen sich beim Training eines Entscheidungsbaums: in jedem Knoten muss ein geeignetes Merkmal für den Schnitt ausgewählt werden, und für dieses Merkmal muss ein geeigneter Schwellwert gefunden werden. Das folgende Kapitel beschreibt eine Möglichkeit zum Trainieren von Entscheidungsbäumen.

Ein *Decision Stump* ist ein Entscheidungsbaum, der nur aus der Wurzel besteht. Er führt also lediglich einen Schnitt in einem einzigen Merkmal durch. Somit ist er meist ein relativ schwacher Klassifikator, der unter Umständen nur unwesentlich besser als eine Zufallsentscheidung ist.

5.1.1. J48 und C4.5

J48 ist eine Implementierung von *C4.5* [29], einem weit verbreiteten Algorithmus zum Trainieren von Entscheidungsbäumen, für das WEKA-Toolkit [16]. Der Entscheidungsbaum wird ausgehend von der Wurzel aus erzeugt. In jedem Knoten wird die Menge $\mathbb{E}$ in die Untermengen $\mathbb{E}_1$ bis $\mathbb{E}_k$ partitioniert. Die Partitionierung erfolgt wie zuvor beschrieben durch einen oder mehrere Schwellwerte auf einem Attribut. Dabei können durchaus mehr als ein Schwellwert gewählt werden, so dass der trainierte Baum nicht notwendigerweise ein binärer Baum ist. Die Schwellwerte und das Attribut werden so gewählt, dass ein Gütemaß optimiert wird. Bei C4.5 wird hierzu *Information Gain* verwendet. Die folgenden Definitionen führen zur Berechnung des Information Gain.

Definition 5.1 (Precision)**.** *Die* Precision *einer Klasse y auf einer Beispielmenge $\mathbb{E}$ ist definiert als*

$$Prec_{\mathbb{E}}(y) := \frac{|\{(x_i, y_i) \in \mathbb{E} | y_i = y\}|}{|\mathbb{E}|} \quad . \tag{33}$$

Die Information, die daraus abgeleitet werden kann, ist

$$-\log_2 \mathrm{Prec}_{\mathbb{E}}(y) \quad . \tag{34}$$

Die Information, die aus einer Beispielmenge mit mehreren Klassen abgeleitet werden kann, heißt *Entropie*.

Definition 5.2 (Entropie)**.** *Die* Entropie *einer Beispielmenge* $\mathbb{E}$ *ist definiert als*

$$info(\mathbb{E}) := -\sum_{y \in Y} Prec_{\mathbb{E}}(y) \cdot \log_2 Prec_{\mathbb{E}}(y) \quad . \tag{35}$$

Definition 5.3 (Entropie einer Partitionierung)**.** *Die* Entropie einer Partitionierung $\mathbb{E}_1, \ldots \mathbb{E}_k$ *von* $\mathbb{E}$ *ist definiert als*

$$info_X(\mathbb{E}) := \sum_{i=1}^{k} \frac{|\mathbb{E}_i|}{|\mathbb{E}|} \cdot info(\mathbb{E}_i) \quad . \tag{36}$$

Daraus lässt sich der *Information Gain* berechnen:

Definition 5.4 (Information Gain)**.** *Der* Information Gain *einer Partitionierung der Beispielmenge* $\mathbb{E}$ *ist definiert als*

$$gain(\mathbb{E}) = info(\mathbb{E}) - \underset{X}{info}(\mathbb{E}) \quad . \tag{37}$$

Das ist der Anstieg der Entropie, der aus der gewählten Partitionierung resultiert. Zusammen mit der *Split Entropie* wird das *Gain Ratio* berechnet:

Definition 5.5 (Split Entropie)**.** *Die* Split Entropie *einer Partitionierung der Beispielmenge* $\mathbb{E}$ *ist definiert als*

$$split\,info(\mathbb{E}) := -\sum_{i=1}^{k} \frac{|\mathbb{E}_i|}{|\mathbb{E}|} \cdot \log_2 \frac{|\mathbb{E}_i|}{|\mathbb{E}|} \quad . \tag{38}$$

Definition 5.6 (Gain Ratio)**.** *Das* Gain Ratio*-Kriterium ist definiert als*

$$gain\,ratio(\mathbb{E}) := \frac{gain(\mathbb{E})}{split\,info(\mathbb{E})} \quad . \tag{39}$$

Die Schnitte in jedem Knoten werden so optimiert, dass das Gain Ratio-Kriterium maximal wird. Durch die Normalisierung des Information Gain mit der Split Entropie wird Gain Ratio groß, wenn die Entropie der Partitionen $\mathbb{E}_1$ bis $\mathbb{E}_k$ groß ist und die Partitionen viele Beispiele enthalten. So wird verhindert, dass die Beispielmenge zu schnell zu stark aufgeteilt wird und ein schnelles Overfitting stattfindet. Die Partitionierung wird jedoch so lange fortgeführt, bis trotz dieses Kriteriums die Blätter weitestgehend rein sind. Die Laufzeit zum Training eines Baums ist bezüglich der Größe m der Trainingsmenge beschränkt durch $O(m)$.

Enthalten die Merkmale kontinuierliche Werte, so müssen die m' Beispiele in einem Knoten nach diesem Merkmal sortiert werden. Dann wird über jeden Wert des Merkmals, der in dem Knoten vorkommt, iteriert. Diese Werte werden als Schwellwert ge-

testet und das Gain Ratio dafür berechnet. Der Schwellwert, der das beste Gain Ratio liefert, wird für den Split ausgewählt. Bei großen Trainingsmengen kann die Laufzeit zur Erzeugung des Baums von der Sortierung der Beispiele dominiert werden, so dass sie sich auf $O(m \log m)$ erhöht und somit nicht mehr linear in der Anzahl der Beispiele ist.

Nach der Erzeugung des Baums werden einige Blätter durch Pruning zusammengefasst, um Overfitting zu vermeiden. Blätter werden genau dann zusammengefasst, wenn eine Abschätzung des Klassifikationsfehlers auf dem zusammengefassten Blatt geringer ist, als die gewichtete Summe der Fehlerschätzer in den Blättern. Zur Schätzung des Fehlers wird nicht die Güte bei Anwendung auf eine Testmenge verwendet, sondern statistische Methoden: wenn ein Blatt $|\mathbb{E}|$ Beispiele der Trainingsmenge enthält, davon E fälschlicherweise (d. h. E Beispiele gehören zur weniger stark repräsentierten Klasse in diesem Knoten), dann wird dies als E „Ereignisse“ bei $|\mathbb{E}|$ Beobachtungen im Sinne der Binomialverteilung aufgefasst. Die obere Schranke des c-Konfidenzintervalls wird als Fehlerschätzer verwendet. Die Konfidenz c ist ein Parameter des Algorithmus'.

5.2. Support Vector Machine

Die *Support Vector Machine*, kurz SVM, ist ein Lernverfahren, das auf VAPNIK zurückgeht [38]. Sie wird für binäre Klassifikationsaufgaben oder zur Regression genutzt[6]. Daneben gibt es auch Varianten für strukturelle Modelle (bei denen für eine Beobachtung $\vec{x}$ nicht eine Zahl, sondern ein Vektor $\vec{y}$ ausgegeben wird) und für Clustering. Für die Gamma-Separation ist die Lernaufgabe die binäre Klassifikation.

Sei die Trainingsmenge gegeben als N Tupel (x_i, y_i), wobei $y_i \in \{-1; 1\}$ das Label des Beispiels x_i angibt. Geht man zunächst davon aus, dass die Daten linear separierbar sind, so lässt sich eine Hyperebene $\{\vec{x} : f(x) = \vec{x} \cdot \vec{\beta} + \beta_0 = 0\}$ bestimmen, die den Merkmalsraum so in zwei Halbräume unterteilt, dass auf der einen Seite nur positive, auf der anderen nur negative Beispiele liegen.

Wählt man $\vec{\beta}'$ so, dass $|\vec{\beta}'| = 1$, dann kann eine Klassifikation erfolgen durch (vgl. [18]):

$$G(x) = \text{sign}(\vec{x} \cdot \vec{\beta}' + \beta_0') \tag{40}$$

Dabei gibt das Argument des sign-Operators den vorzeichenbehafteten Abstand des Beispiels $\vec{x}$ von der Hyperebene an. Die optimale trennende Hyperebene liegt so, dass sowohl der Abstand M zum nächsten positiven Beispiel als auch zum nächsten negativen Beispiel maximal ist. Abb. 11 skizziert eine solche Ebene. Der Raum ohne Beispiele zwischen der Hyperebene und den nächsten Beispielen heißt *Margin*. Wenn kein Beispiel auf dem Margin liegt, so muss gelten (vgl. [18])

$$\vec{y}_i(\vec{x}_i \cdot \vec{\beta}' + \beta_0') \geq M,\ i = 1, \ldots, N\,, \tag{41}$$

[6] Die Ausführungen in diesem Kapitel orientieren sich an [21].

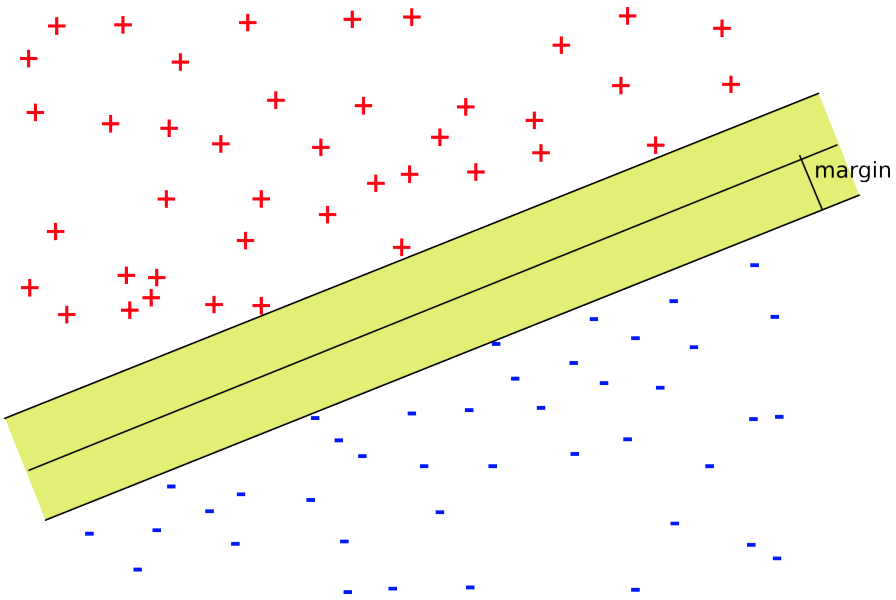


Abbildung 11: Optimale trennende Hyperebene im separierbaren Fall. Keine Beispiele liegen auf dem Margin.

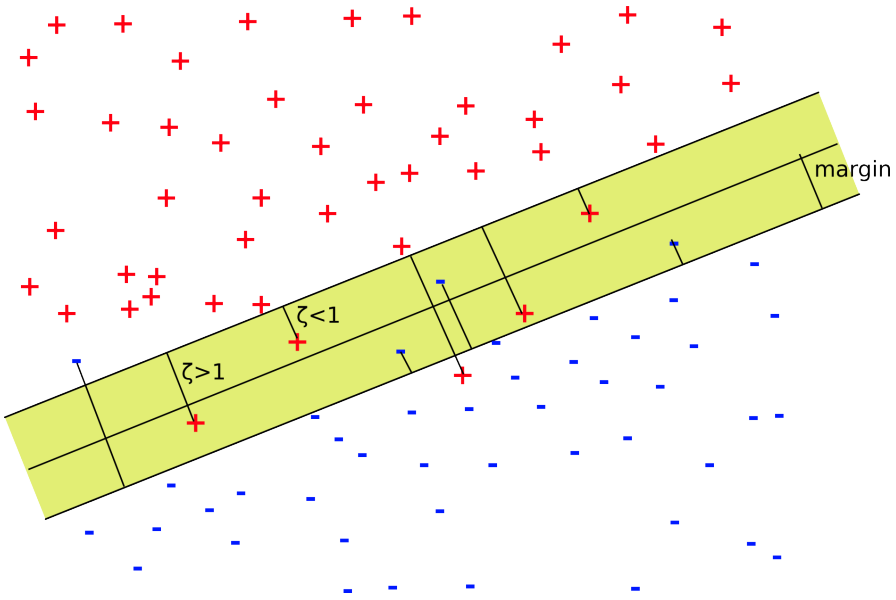


Abbildung 12: Optimale trennende Hyperebene im nicht-separierbaren Fall. Einige Beispiele liegen auf dem Margin und auf der falschen Seite der trennenden Hyperebene. Diese werden durch Slack-Variablen bestraft.

und M soll maximiert werden:

$$\max_{\vec{\beta}',\beta_0',|\vec{\beta}'|=1} M\,. \tag{42}$$

Teilt man diese Gleichung durch M und wählt $\vec{\beta} = \frac{\vec{\beta}'}{M}$ und $\beta_0 = \frac{\beta_0'}{M}$, so erhält man

$$\vec{y}_i(\vec{x}_i \cdot \vec{\beta} + \beta_0) \geq 1,\ i = 1, \ldots, N\,, \tag{43}$$

und es soll $|\vec{\beta}|$ minimiert werden:

$$\min_{\vec{\beta},\beta_0} |\vec{\beta}|\,. \tag{44}$$

Im allgemeinen Fall sind die zu verarbeitenden Daten aber nicht vollständig separierbar. In diesem Fall müssen Ausnahmen in Form von Beispielen, die innerhalb des Margins und unter Umständen sogar auf der falschen Seite der Hyperebene liegen, zugelassen werden (vgl. Abb. 12). Formell wird dies durch die Einführung von *Slack-Variablen* (vgl. [18]) $\xi_i \in \mathbb{R}^+$ erreicht. ξ_i gibt an, wie weit ein Beispiel innerhalb des Margins liegt, in Einheiten der Breite des Margins $\frac{1}{M}$. Ein Beispiel x_i wird somit genau dann fehlklassifiziert, wenn $\xi_i > 1$. Die Summe aller ξ_i ist durch einen frei wählbaren Parameter c nach oben beschränkt. Somit können nicht mehr als c Beispiele fehlklassifiziert werden. Gleichungen (43) und (44) ergeben sich zu

$$\begin{aligned} &\min_{\vec{\beta},\beta_0} |\vec{\beta}| \\ \text{so dass}\quad &\forall i : \vec{y}_i(\vec{x}_i\vec{\beta} + \beta_0) \geq 1 - \xi_i,\ \xi_i \geq 0,\ \sum \xi_i < c\,. \end{aligned} \tag{45}$$

Dies ist äquivalent zu

$$\begin{aligned} &\min_{\vec{\beta},\beta_0} \frac{1}{2}\vec{\beta}^2 \\ \text{so dass}\quad &\forall i : \vec{y}_i(\vec{x}_i\vec{\beta} + \beta_0) \geq 1 - \xi_i,\ \xi_i \geq 0\,. \end{aligned} \tag{46}$$

Hier wurde die Konstante c durch einen Kostenfaktor C ersetzt.

Nach dem Prinzip der quadratischen Programmierung ergibt sich daraus das primale Problem in Lagrange-Formulierung ([18]):

$$L_P = \frac{1}{2}\vec{\beta}^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\vec{x}_i\vec{\beta} + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i\,. \tag{47}$$

Dabei sind die μ_i und α_i die Lagrange-Faktoren. L_P soll nun nach $\vec{\beta}$, β_0 und den ξ_i minimiert werden. Dazu werden die Ableitungen zu Null gesetzt, woraus sich die fol-

genden Gleichungen ergeben:

$$\vec{\beta} = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{48}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \tag{49}$$

$$\forall i : \alpha_i = C - \mu_i \tag{50}$$

Mit Hilfe dieser Gleichungen erhält man aus dem primalen Problem das duale Problem

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \cdot f(\vec{x}_i, \vec{x}_j) \tag{51}$$

mit $f(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$. Der Vorteil dieser Umformung liegt auf der Hand, wenn man sich vor Augen führt, dass das duale Problem nur N freie Parameter in Form der α_i hat, während das primale Problem mit den μ_i und xi_i weitere $2N$ Parameter besitzt. Dies verringert den Aufwand für die Optimierung erheblich.

L_D wird unter Einhaltung von Gl. (48), (49) und (50) sowie Gl. (52), (53) und (54), die sich durch weitere Umformungen ergeben, maximiert. Dies ist analytisch nicht möglich, es existieren jedoch numerische Verfahren, die dieses Problem lösen.

$$\alpha_i [y_i (f(\vec{x}_i, \vec{\beta}) + \beta_0) - (1 - \xi_i)] = 0 \tag{52}$$

$$\mu_i \xi_i = 0 \tag{53}$$

$$y_i (f(\vec{x}_i, \vec{\beta}) + \beta_0) - (1 - \xi_i) \geq 0 \tag{54}$$

Nachdem die α_i mit einer geeigneten Methode bestimmt wurden, kann das optimale $\hat{\vec{\beta}}$ nach Gl. (48) berechnet werden; es ist also eine Linearkombination aus Beispiel-Vektoren. Aus Gl. (52) und (54) geht hervor, dass nur solche Vektoren einen endlichen Beitrag dazu leisten, die auf dem Margins liegen (vgl. [18]). Diese Vektoren heißen *Support-Vektoren* und sind die Namensgeber der SVM.

Die Berechnung der α_i erfolgt in der Regel durch *Sequential Minimal Optimization* (SMO, vgl. [28]). Die Komplexität der SMO liegt theoretisch etwa bei $O(m)$ bis $O(m^{2,3}$ [37], wobei das Verfahren in der Praxis eher zu der höheren Schranke tendiert.

6. Meta-Klassifikationsverfahren

Die zuvor beschriebenen Klassifikationsverfahren arbeiten direkt auf einer Trainingsmenge und erzeugen daraus ein Modell. Meta-Klassifikationsverfahren trainieren eine Reihe von Basisklassifikatoren und fassen diese zu einem sogenannten Ensemble zusammen. Als Basisklassifikatoren können meist beliebige Klassifikationsverfahren wie beispielsweise die zuvor beschriebenen Entscheidungsbäume oder eine SVM verwendet werden. Die Wahl der Basisklassifikatoren ist von den zu untersuchenden Daten und vor allem von der Arbeitsweise des Meta-Klassifikationsverfahren abhängig. Bei

manchen Verfahren wie dem Random Forest ((vgl. Kapitel 6.5) ist der Basisklassifikator sogar fest vorgegeben.

Zwei Untergruppen der Meta-Klassifikationsverfahren sind *Boosting* und *Bagging*. Kapitel 6.1 und 6.2 liefern die Grundlagen für das Boosting-Verfahren *Ada*^{2}*Boost* , das in Kapitel 6.3 beschrieben wird. Kapitel 6.5 stelt den *Random Forest* als Beispiel für ein Bagging-Verfahren vor.

6.1. Wissensbasiertes Sampling

Bei den meisten Lernaufgaben, die iterativ Modelle oder Regeln erzeugen, ist es nicht erwünscht, dass gleiche oder ähnliche Modelle mehr als einmal erzeugt werden. Beim Boosting beispielsweise wird ein Ensemble erzeugt, das in keiner Weise von redundanten Modellen profitiert. Mit Hilfe des *wissensbasierten Samplings* (KBS, vgl. [35]) kann die Verteilung der Trainingsmenge so verändert werden, dass es unmöglich ist, redundante Modelle zu finden . Gleichzeitig wird vermieden, die Verteilung stärker zu verändern, als es zur Erreichung dieses Ziels nötig ist. Scholz definiert mehrere Bedingungen, durch die genau diese Forderungen erfüllt werden.

KBS entfernt Vorwissen in Form einer Regel $r : A \rightarrow C$ aus der ursprünglichen Verteilung D. Da Regeln auch als Klassifikatoren interpretiert werden können, wird dadurch auch Vorwissen in Form von Modellen oder auch Ensembles von Modellen abgedeckt, insbesondere, wenn man weiterhin davon ausgeht, dass $A \rightarrow C$ auch $\overline{A} \rightarrow \overline{C}$ impliziert. Gesucht ist eine Verteilung D', unter der die Regel r keinerlei Information liefert, d. h. nicht besser als zufälliges Raten ist. Das ist genau dann der Fall, wenn A und C unabhängige Ereignisse sind:

$$\Pr_{D'}[C|A] = \Pr_{D'}[C] \tag{55}$$

Das impliziert auch $\text{Lift}_{D'}(r) = 1$.

Abgesehen davon soll die Verteilung möglichst nicht verändert werden. Die a-priori-Wahrscheinlichkeit für A und C sollte also gleich bleiben:

$$\Pr_{D'}[A] = \Pr_{D}[A] \tag{56}$$
$$\Pr_{D'}[C] = \Pr_{D}[C] \tag{57}$$

Die Regel partitioniert den Beispielraum in die bekannten Mengen TP, FP, TN, FN. Innerhalb dieser Partitionen soll die Verteilung unverändert bleiben:

$$\Pr_{D'}[x|A, C] = \Pr_{D}[x|A, C] \tag{58}$$
$$\Pr_{D'}[x|A, \overline{C}] = \Pr_{D}[x|A, \overline{C}] \tag{59}$$
$$\Pr_{D'}[x|\overline{A}, C] = \Pr_{D}[x|\overline{A}, C] \tag{60}$$
$$\Pr_{D'}[x|\overline{A}, \overline{C}] = \Pr_{D}[x|\overline{A}, \overline{C}] \tag{61}$$
$$\tag{62}$$

In jeder dieser Partitionen ist das entsprechend angepasste Lift-Kriterium i. A. ungleich 1. Um beispielsweise den Lift für die FN zu berechnen, wird nicht $\text{Lift}(A \rightarrow C)$ verwendet, sondern $\text{Lift}(\overline{A} \rightarrow C)$.

Gelingt es, die Verteilung so zu ändern, dass der Lift in jeder Partition gleich 1 ist, so liefert die Regel r auf der neuen Verteilung keine Informationen und ist nicht besser als zufälliges Raten, so dass sie bei einem Training auf der neuen Verteilung nicht erneut gefunden wird und Bedingung (55) erfüllt ist. Das wird erreicht, indem man die neue Verteilung D' nach der Regel

$$\Pr_{x\sim D'}[x] = \frac{\Pr_{x\sim D}[x]}{\mathrm{Lift}_D(r,x)} \tag{63}$$

erzeugt. $\mathrm{Lift}_D(r,x)$ ist in der folgenden Definition gegeben.

Definition 6.1. *Sei $r : A \to C$ eine Regel und x ein beliebiges Beispiel. Dann ist das* beispielbasierte Lift-Kriterium *definiert durch*

$$Lift(A \to C, x) := \begin{cases} Lift(A \to C) & x \in Ext(A) \cap Ext(C) \\ Lift(A \to \overline{C}) & x \in Ext(A) \cap Ext(\overline{C}) \\ Lift(\overline{A} \to C) & x \in Ext(\overline{A}) \cap Ext(C) \\ Lift(\overline{A} \to \overline{C}) & x \in Ext(\overline{A}) \cap Ext(\overline{C}) \quad . \end{cases} \tag{64}$$

Das beispielbasierte Lift-Kriterium liefert also das in Gleichung (12) definierte Lift-Kriterium der Partition, in die das Beispiel x durch die Regel r einsortiert wird.

Gleichung (63) erfüllt auch die übrigen Constraints (56) bis (61) (vgl. [35]).

6.2. Boosting

Boosting ist ein Meta-Verfahren, bei dem ein Ensemble aus mehreren einzelnen Klassifikatoren, sogenannten Basismodellen, erzeugt wird [13]. Die Basismodelle für sich betrachtet sind in der Regel eher schwache Klassifizierer, die nur wenig besser als Raten sind. Im Gegensatz zu Bagging-Verfahren wie dem Random Forest, der in Kapitel 6.5 beschrieben wird, werden die Basismodelle nicht unabhängig voneinander trainiert, sondern iterativ, wobei jeweils die Informationen aus den vorangegangenen Iterationen verwertet werden. Insbesondere wird die Verteilung der Trainingsdaten so verändert, dass die Teilmengen der Trainingsmenge, auf denen das Ensemble aus den vorherigen Iterationen Fehler macht, stärker gewichtet, Teilmengen mit korrekten Vorhersagen jedoch schwächer gewichtet werden. Dadurch werden in späteren Iterationen spezialisierte Basismodelle trainiert, die die Fehler aus den vorherigen Iterationen beheben.

Die Klassifizierungsfunktion $h(x)$ des Boosting-Modells ist abhängig von allen Basismodellen. Die genaue Form der Funktion ist abhängig vom Verfahren, ist jedoch oft das gewichtete Mittel aller Basismodelle, wobei die Modellgewichte häufig auf der Güte des Modells auf den Trainingsdaten basiert:

$$h(x) = \left(\sum_{i=1}^{k} \alpha_i\right)^{-1} \cdot \sum_{i=1}^{k} \alpha_i h_i(x) \tag{65}$$

Offensichtlich umfasst der Wertebereich der Funktion nicht nur die diskreten Werte -1 und $+1$, sondern alle reellen Werte in $[-1,+1]$. Das Ensemble ist also ein weicher

Klassifizierer mit Ausgabe der Konfidenzen. Um ihn in einen harten Klassifizierer umzuwandeln, muss ein Schwellwert q gewählt werden. Ein Beispiel x wird genau dann als positiv klassifiziert, wenn $h(x) \geq q$, die Klassifikationsfunktion diesen Schwellwert also überschreitet.

Jeder Schwellwert entspricht einem Punkt auf der ROC-Kurve. Dies kann man sich zunutze machen, wenn durch die Wahl des Schwellwerts beispielsweise ein bestimmter Recall eingestellt werden soll (vgl. Kapitel 11).

Ein weit verbreitetes Boosting-Verfahren ist AdaBoost [12]. AdaBoost wird in Kapitel 6.4 analysiert. Ein neuerer, in weiten Teilen ähnlicher Algorithmus wurde von Scholz unter der Bezeichnung Ada²Boost beschrieben.

6.3. Ada²Boost - ein stratifizierender Boosting Classifier

Ada²Boost [35] ist ein Boosting-Algorithmus, der große Ähnlichkeit zum wissensbasierten Sampling aufweist. Als Eingabe erhält er eine Trainingsmenge $\mathbb{E} =\subseteq X \times Y$ sowie die gewünschte Anzahl k der zu trainierenden Basismodelle, d. h. die Anzahl der Iterationen. Zu Beginn werden die Gewichte für jedes Beispiel (x_i, y_i) mit $w_1(x_i, y_i) = 1$ initialisiert. Dann wird in jeder der k Iterationen ein Basismodell $h_t : X \to \{-1, +1\}$ unter Berücksichtigung der Beispielgewichte auf der Trainingsmenge erzeugt. Wie in Kapitel 4.2.1 beschrieben, partitioniert das Modell die Trainingsmenge in einen abgedeckten Teil $C_t := \{(x, y) \in \mathbb{E} | h_t(x) = +1\}$ und einen nicht abgedeckten Teil $\overline{C}_t := \{(x, y) \in \mathbb{E} | h_t(x) = -1\}$. Für beide Partitionen wird die Fehlklassifikationsrate separat durch

$$\epsilon^+ := \frac{\overline{p}}{p + \overline{p}} \qquad \text{für } C_t \tag{66}$$

$$\epsilon^- := \frac{n}{n + \overline{n}} \qquad \text{für } \overline{C}_t \tag{67}$$

definiert.

Mit Hilfe der Fehlklassifikationsrate wird das Gewicht α_t des Basismodells $h_t(x)$ im Ensemble berechnet. Zunächst wird der Faktor β definiert:

$$\begin{aligned} \beta(+1) &:= \frac{1 - \epsilon^+}{\epsilon^+} = \frac{p}{n} \\ \beta(-1) &:= \frac{1 - \epsilon^-}{\epsilon^-} = \frac{\overline{n}}{\overline{p}} \end{aligned} \tag{68}$$

Das Gewicht des Basismodells ist dann

$$\alpha_t(h(x)) := \frac{\ln \beta(h(x))}{2} \quad . \tag{69}$$

Außerdem werden die Beispielgewichte w_{t+1} für die nächste Iteration berechnet:

$$\begin{aligned} \forall (x, y) \in \mathbb{E} : w_{t+1}(x, y) :=& w_t(x, y) \cdot \exp(-y \cdot \alpha(h_t(x)) \cdot h_t(x)) \\ =& \frac{w_t(x, y)}{\sqrt{\beta_t(h_t(x))}^y} \end{aligned} \tag{70}$$

Nachdem die Basismodelle für alle k Iterationen trainiert worden sind, können neue Beispiele mit Hilfe der Entscheidungsfunktion

$$h(x) = \hat{y} = \text{sign}\left(\sum_{t_1}^{k} \alpha_t(h_t(x)) \cdot h_t(x)\right) \tag{71}$$

klassifiziert werden. Anstelle einer harten Klassifikation kann auch eine Konfidenz bzw. ein Schätzer für die Wahrscheinlichkeit, dass das Label eines Beispiels positiv ist, ausgegeben werden. Mit

$$\hat{\beta}(x) := \prod_{t=1}^{k} \beta_t(h(x)) \tag{72}$$

ist das

$$\hat{\Pr}[y = +1|x] = \frac{\hat{\beta}(x)}{1 + \hat{\beta}(x)} \quad . \tag{73}$$

Anzumerken ist, dass Ada^{2}Boost ausschließlich harte Klassifikatoren als Basismodelle nutzt, selbst wenn das Verfahren zum Training der Basismodelle die Berechnung von Konfidenzen erlaubt. Durch diese Diskretisierung gehen bei Umgewichtung der Beispiele unter Umständen Informationen verloren. Kapitel 12 untersucht mehrere Möglichkeiten, die Konfidenzen der Basismodelle zu nutzen.

Im Gegensatz zu anderen Boosting-Verfahren sind die Gewichte der Basismodelle nicht konstant, sondern abhängig davon, wie das Modell ein Beispiel klassifiziert: $\alpha_t = \alpha_t(h_t(x))$. Auch bei der Neuberechnung der Beispielgewichte spielt die Vorhersage des aktuellen Basismodells eine Rolle: für jede Partition werden die Gewichte separat berechnet. Vergleicht man die Umgewichtungsregel (70) mit Gleichung (21), so sieht man, dass die Umgewichtung auf jeder Partition C, $\overline{C}$ eine Stratifizierung mit dem Umgewichtungsfaktor $c^+ = \beta_t(+1)$ bzw. $c^- = \beta_t(-1)$ vornimmt. Betrachtet man außerdem die Gleichungen (66), (67) und(22), so ist offensichtlich, dass in beiden Partitionen separat die a-priori-Wahrscheinlichkeiten beider Klassen bzw. die Gewichte jeder Klasse angeglichen werden.

Durch die Stratifizierung erfüllt die Umgewichtungsregel von Ada^{2}Boost fast alle Bedingungen des wissensbasierten Samplings. (55) ist erfüllt, da durch die Stratifizierung

$$(\forall y_1, y_2 \in Y) : \Pr_{D_{t+1}}[y = y_1 | h_t(x) = y_2] = \frac{1}{2} \tag{74}$$

gilt, d. h., dass das Basismodell auf der neugewichteten Trainingsmenge nicht besser oder schlechter als Raten ist, und somit A und C unabhängige Ereignisse sind. Durch die Stratifikation ist trivialerweise auch Bedingung (57) erfüllt, da die a-priori-Wahrscheinlichkeiten für alle Klassen per definitionem immer $\frac{1}{2}$ sind. Gleichungen (58) - (61) besagen, dass sich innerhalb der Mengen TP, FP, FN, TN die Verteilungen nicht ändern. Da durch die Stratifizierung alle Beispiele einer dieser Mengen mit demselben Faktor multipliziert werden, die Mengen also atomar behandelt werden, ändern sich

auch die inneren Verteilungen nicht.

Bis auf Gleichung (56) sind also alle Bedingungen des wissensbasierten Samplings erfüllt. Durch Nicht-Erfüllung dieser Gleichung verschiebt Ada²Boost die Verteilung, während KBS lediglich das Vorwissen aus der Trainingsmenge herausfiltert. Dadurch wird das Gesamtgewicht der Trainingsmenge verringert, was wiederum bedeutet, dass die obere Schranke für den Klassifikationsfehler des Ensembles gesenkt wird [35].

Ansonsten ist die Neugewichtung in Ada²Boost identisch zum wissensbasierten Sampling. Dadurch ist gewährleistet, dass Vorwissen effizient ausgefiltert wird und das Training der neuen Basismodelle vermehrt die schwierigen Beispiele einbezieht.

6.4. Vergleich von Ada²Boost zu AdaBoost

Das Boosting-Verfahren *AdaBoost* [12], auf dessen Grundlage Ada²Boost entwickelt wurde, ist bis auf die Regel zur Neugewichtung der Beispiele und Basismodelle identisch zu Ada²Boost. Bei AdaBoost wird α_t durch

$$\alpha_t(x) = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t} \tag{75}$$

berechnet und β_t durch

$$\beta_t(x) := \frac{1 - \epsilon_t}{\epsilon_t} = \exp(2\alpha_t) \quad . \tag{76}$$

Dabei ist ϵ_t die Fehlklassifikationrate von h_t auf dem *gesamten* Sample.

Die Regel zur Neugewichtung lautet

$$\begin{aligned} \forall (x,y) \in \mathbb{E} : w_{t+1}(x,y) :=& w_t(x,y) \cdot \exp(-y \cdot \alpha_t \cdot h_t(x)) \\ =& \frac{w_t(x,y)}{\sqrt{\beta_t}^{y}} \quad . \end{aligned} \tag{77}$$

AdaBoost stratifiziert also nicht jede Partition separat, sondern berechnet eine globale Fehlklassifikationsrate und basiert die Umgewichtung darauf. Das ist vor allem dann nachteilig, wenn sich die Fehlklassifikationsraten für C und $\overline{C}$ stark unterscheiden und die Mengen unterschiedliche Kardinalitäten haben.

Wenn beispielsweise ein Klassifikator eine kleine Untermenge der Trainingsmenge mit einer hohen Precision korrekt als positiv klassifiziert, ist die Fehlklassifikationsrate auf diesem kleinen Sample sehr gering. Wenn das Modell auf der anderen, größeren Partition $\overline{C}$ jedoch nur unwesentlich besser als eine Zufallsentscheidung ist, ist die globale Fehlklassifikationsrate hoch und das Gewicht des Klassifikators in AdaBoost gering. Ada²Boost würde jedoch die gute Performance auf C nutzen, da es die lokale Fehlklassifikationsrate nutzt, und aus demselben Grund die große, ungenaue Partition durch ein geringes Gewicht weitgehend ignorieren. Ada²Boost nutzt also die Information aus den trainierten Basismodellen effizienter, wenn sich die Fehlklassifikationsraten unterscheiden. Sind sie identisch, so verhalten sich auch AdaBoost und Ada²Boost identisch. Ein formaler Beweis findet sich in [35].

6.5. Random Forest

Ein *Random Forest* [5] ist ein Ensemble aus Entscheidungsbäumen, die jeweils mit einer Zufallskomponente erzeugt werden. Die von Breiman beschriebene Variante nutzt in jedem Knoten ein zufällig gewähltes Attribut, anstatt wie C4.5 jedes Attribut zu testen und den Information Gain zu berechnen. Lediglich der Schwellwert wird so gewählt, dass er das Gütekriterium optimiert. Bei der Klassifikation unbekannter Beispiele ist die Ausgabe des Ensembles ein ungewichteter Mehrheitsentscheid aller Entscheidungsbäume.

Formell betrachtet werden k Baumlerner $h_k : X \to Y$ trainiert, die zusätzlich zur Trainingsmenge $\mathbb{E} \subseteq X \times Y$ einen Vektor Θ_i mit Zufallszahlen als Eingabe erhalten, der nach der Verteilung Θ gezogen wurde. Die Zufallszahlen werden während des Trainings zur Auswahl des trennenden Merkmals an jedem Knoten genutzt. Die Ausgabe des Ensembles $h : X \to Y$ ist

$$h(x) = \text{sign}\frac{1}{k}\sum_{i=1}^{k} h_i(x) \quad . \tag{78}$$

Durch Weglassen der Signum-Funktion kann dieser harte Klassifizierer in einen weichen Klassifizierer mit Wertebereich $\mathbb{R}$ umgewandelt werden.

Eine wichtige Eigenschaft des Ramdom Forest ist, dass sein Generalisierungsfehler selbst bei großen k nach oben beschränkt ist und somit selbst bei Verwendung sehr vieler Bäume kein Overfitting stattfinden kann [5]. Diese Schranke ist umso kleiner, je stärker die einzelnen Baummodelle sind, und je stärker sie sich voneinander unterscheiden.

Im Unterschied zu den zuvor beschriebenen Boosting-Verfahren erzeugt der Random Forest basierend auf Zufallszahlen *unabhängige* Basismodelle, während beim Boosting iterativ und deterministisch Modelle erzeugt werden, wobei jedes Modell die Informationen aus den vorherigen Iterationen nutzt.

7. Vorverarbeitung

7.1. Merkmalsselektion

Es kommt häufig vor, dass eine Beispielmenge viele Merkmale beinhaltet, von denen nur wenige für eine gute Klassifikationsleistung benötigt werden. Außerdem können einige Merkmale redundant sein und dieselbe Information kodieren. Solche redundanten Merkmale können die Leistung einiger Klassifikatoren verschlechtern. Auch die mit der Eliminierung von Merkmalen einhergehende Reduzierung der Dimension des Beispielraums kann sich positiv auf das Klassifikationsergebnis auswirken, da hochdimensionale Daten den Lernprozess erschweren. [18] beschreibt dies als *Fluch der hohen Dimension*.

Die folgende Definition gibt eine formale Beschreibung für die Suche nach einer optimalen Merkmalsmenge:

Definition 7.1 (Merkmalsselektion)**.** *Gegeben seien die Merkmale $D := X_1, \ldots, X_d$. Die* Merkmalsselektion *sucht eine Untermenge $D' = \{X'_1, \ldots, X'_d\} \subseteq D$, so dass ein Klassifika-*

```
1  Eingabe:
2    Testmenge 𝔼
3    Merkmalsmenge D
4  Ausgabe:
5    optimale Merkmalsmenge D'
6
7
8  Initialisiere D' := ∅
9  Performance p := −∞
10
11 while D ≠ ∅
12   Wähle X_opt = arg max_{X'∈D} [Performance_𝔼(h : D' ∪ {X'} → Y)]
13   p' := Performance_𝔼(h : D' ∪ {X_opt} → Y)
14   if p' > p
15     p := p'
16     D' := D' ∪ {X_opt}
17     D := D\{X_opt}
18   else
19     return D'
20   end
21 end
22 return D'
```

Algorithmus 1: Forward Feature Selection in Pseudocode

tor auf $X' := X'_1 \times \ldots \times X'_{d'}$ *eine optimale Klassifikationsgüte liefert:*

$$D' = \underset{D' \subseteq D}{\arg\max} \, Performance(h : X' \to Y) \tag{79}$$

In dieser Arbeit werden zur Klassifikation stets die Merkmale der Standard-Analyse in MARS verwendet, so dass keine klassische Merkmalsselektion zum Einsatz kommt. In Kapitel 14 wird jedoch gezeigt, dass sich andere Probleme in eine Merkmalsselektion transformieren lassen und die im folgenden beschriebenen Verfahren dort eingesetzt werden können.

Alle hier beschriebenen Verfahren basieren auf demselben Prinzip: alle generieren unterschiedliche Merkmalsmengen D'. Diese werden bewertet, indem die Güte eines Klassifikators $h : X' \to Y$ abgeschätzt wird, der auf einer Trainingsmenge $\mathbb{E}' \subseteq X'$ trainiert wird. Um die Abschätzung der Güte zu verbessern, kann eine Kreuzvalidierung eingesetzt werden.

7.1.1. Forward Selection und Backward Elimination

Die *Forward Feature Selection* (vgl. bspw. [14]) beginnt mit einer leeren Merkmalsmenge. Iterativ wird das Merkmal der Merkmalsmenge hinzugefügt, welches die Güte des Klassifikators am meisten verbessert. Sobald in einer Iteration kein Merkmal gefunden werden kann, welches die Klassifikationsgüte verbessert, wird die aktuelle Merkmalsmenge ausgegeben. Algorithmus 1 zeigt diese Funktionsweise.

Die *Backward Feature Selection* (vgl. ebenfalls [14]) nutzt den umgekehrten Ansatz. Sie

beginnt mit allen Merkmalen und entfernt iterativ Merkmale. Dabei wird jeweils das Merkmal zur Entfernung ausgewählt, dessen Entfernung den größten Performance-Zuwachs nach sich zieht. Kann die Performance durch Entfernen eines Attributs nicht weiter verbessert werden, terminiert die Suche und die aktuelle Merkmalsmenge wird ausgegeben. Der Algorithmus ist analog zu 1.

Beiden Verfahren ist gemein, dass sie *Greedy* sind, da sie einmal gewählte bzw. verworfene Merkmale nicht erneut betrachten [14]. Varianten stoppen nicht in der ersten Iteration ohne Verbesserung, sondern suchen noch einige Iterationen länger. Tritt eine Verbesserung ein, läuft der Algorithmus weiter. Tritt innerhalb einer definierten Anzahl von Iteration keine Verbesserung ein, so wird diejenige Merkmalsmenge ausgegeben, die zuletzt zu einer Verbesserung geführt hat.

7.1.2. Evolutionäre Algorithmen

Angelehnt an die biologische Evolution nutzen evolutionäre Algorithmen (vgl. bspw. [9]) das Konzept von Individuen mit Genen, Populationen, Kreuzungen und Mutationen sowie einen Selektionsprozess. Der Suchraum S wird so kodiert, dass sich Punkte des Suchraums als Binärstring s bzw. *Genom* darstellen lassen. Einzelne Bits des Strings heißen *Gene*. Diese können entweder ein- oder ausgeschaltet sein. Ein *Individuum* entspricht einem bestimmten Binärstring.

Ziel eines evolutionären Algorithmus ist die Suche nach einem besten Individuum gemäß einer *Fitnessfunktion*.

Definition 7.2 (Optimierungsproblem eines evolutionären Algorithmus)**.** *Gegeben sei eine Fitnessfunktion $f : S \to \mathbb{R}$ und ein Suchraum $S \in \{0,1\}^d$. Dann ist das Optimierungsproblem eines evolutionären Algorithmus*

$$\arg\max_{s \in S} f(s) \quad . \tag{80}$$

Der Ablauf eines evolutionären Algorithmus gliedert sich in eine Initialisierungsphase und eine Schleife über mehrere Iterationen, die im Kontext der evolutionären Algorithmen *Generationen* heißen. Während der Initialisierung wird eine Menge von N Individuen erzeugt, die *Population* genannt wird. Die Größe der Population bleibt während des gesamten Algorithmus konstant. Jedes Individuum besteht zunächst aus einem zufälligen Binärstring.

Ziel ist es, die Binärstrings in den Generationen iterativ so zu verändern, dass die Fitness in jeder Iteration steigt. Dazu werden auf einige Individuen verschiedene Operatoren angewendet. Welche Individuen dazu ausgewählt werden, wird während der *Elternselektion* entschieden. Nach Anwendung der Operatoren wird in einer zweiten Selektionsphase entschieden, welche Individuen überleben und in die nächste Generation übernommen werden[7].

Elternselektion Bei der Elternselektion wird entschieden, welche Individuen der aktuellen Population für die nachfolgend beschriebenen Operationen ausgewählt werden. Meist werden starke Individuen gewählt, d. h. solche mit großer Fitness. Aller-

[7] Die Beschreibung der Operationen während einer Generation orientieren sich an [21].

dings werden mit einer geringen Wahrscheinlichkeit auch schwächere Individuen ausgewählt oder neue, zufällige Individuen erzeugt, um zu verhindern, dass der Algorithmus in ein lokales Optimum läuft.

Mutation und Crossover Zur genetischen Veränderung der Individuen stehen die Operationen *Mutation* und *Crossover* zur Verfügung.

Crossover ist eine binäre Operation, die zwei Individuen als Eingabe erhält und daraus ein neues Individuum erzeugt. Dazu kommt ein sogenanntes *Rekombinationsverfahren* zum Einsatz. Bei Individuen konstanter Länge ist *Single-Point-Crossover* ein gängiges Verfahren. Dabei wird zufällig ein Bit ausgewählt, und alle Bits rechts davon zwischen beiden Individuen vertauscht.

Die Mutation ist eine unäre Operation, sie erhält genau ein Individuum als Eingabe. Mit einer Wahrscheinlichkeit p_m werden einzelne Bits invertiert. Die Mutation ist ein wichtiger Bestandteil eines evolutionären Algorithmus: sie stellt sicher, dass jeder Punkt im Suchraum erreicht werden kann. Trägt ein Gen in allen Individuen der Population denselben Wert, so kann allein durch Crossover dieser Wert nicht geändert werden. Durch die Mutation können jedoch beliebige neue Individuen erzeugt werden. Somit trägt auch die Mutation dazu bei, dass der Algorithmus nicht in ein lokales Maximum läuft.

Selektion überlebender Individuen In diesem Schritt werden aus allen Individuen der aktuellen Generation N Individuen ausgewählt, die in die nächste Generation übernommen werden. Zur Auswahl stehen alle unveränderten Individuen sowie die duch Mutation und Crossover neu hinzugekommenen Individuen. Es überleben nicht notwendigerweise die besten Individuen, d. h. die mit hoher Fitness, sondern es können mit gewisser Wahrscheinlichkeit auch schwächere Individuen ausgewählt werden. Auch dies dient dazu zu verhindern, dass sich die Population um ein lokales Maximum herum verdichtet.

Dieser Prozess wird solange fortgeführt, bis ein Abbruchkriterium erfüllt ist. Üblicherweise ist das das Erreichen einer maximalen Anzahl an Generationen, oder der Durchlauf einer bestimmten Anzahl an Generationen, in denen keine neuen Individuen gefunden werden konnten, die eine bessere Fitness haben als Individuen aus den vorherigen Generationen. Algorithmus 2 veranschaulicht den Ablauf eines evolutionären Algorithmus.

Zur Merkmalsselektion kodieren die Bitstrings die verwendeten Merkmale [31]. Enthält der Beispielraum d Merkmale, so ist der Suchraum ein Binärstring der Länge d: $S = \{0,1\}^d$. Jedes Gen repräsentiert ein Merkmal. Ist das i-te Gen aktiviert, so wird das entsprechende Merkmal verwendet, ist es deaktiviert, so wird das Merkmal für dieses Individuum aus der Merkmalsmenge entfernt. Die Fitnessfunktion entspricht einem beliebigen Gütemaß und wird berechnet, indem ein Lernalgorithmus wie zuvor beschrieben auf einer Testmenge mit den entsprechenden Merkmalen trainiert und evaluiert wird, beispielsweise mit Hilfe einer Kreuzvalidierung.

```
1  Eingabe:
2    Testmenge 𝔼
3    Merkmalsmenge D
4    Populationsgröße N
5    Abbruchbedingung B
6  Ausgabe:
7    optimale Merkmalsmenge D'
8
9  Initialisiere Population P mit N zufälligen Individuen s ∈ {0,1}^|D|
10 Berechne Fitness f(s) für jedes Individuum
11 while !B
12   Führe Elternselektion durch
13   Wende Crossover und Mutation an
14   Berechne Fitness für jedes Individuum
15   Selektiere N Individuen für die nächste Generation
16 end
17 return D' = D'(arg max_{s∈P} f(s))
```

Algorithmus 2: Evolutionärer Algorithmus zur Merkmalsselektion in Pseudocode (nach [31])

Teil II.
Eigene Arbeiten

8. Überblick

In diesem Teil werden die Verfahren vorgestellt und bewertet, die im Rahmen dieser Arbeit entwickelt worden sind. Zunächst wird Fuzzy Ada^{2}Boost entwickelt, welches Ada^{2}Boost so verändert, dass es weiche Basisklassifikatoren mit kontinuierlichem Wertebereich nutzen kann.

Die im MAGIC-Experiment zur Verfügung stehenden Daten haben einen sehr großen Umfang. Um diese große Datenmenge besser nutzen zu können, erweitert Ada^{2}Boost for Very Large Dataset (VLDS-Ada^{2}Boost) das zuvor entwickelte Fuzzy Ada^{2}Boost dahingehend, dass während des Lernprozesses die Trainingsmenge ausgetauscht wird. Dadurch werden während eines einzigen Boosting-Vorgangs mehr Daten genutzt.

Weiterhin ist die Verteilung der Trainingsdaten nicht stationär, sondern ändert sich in Abhängigkeit einiger Parameter. Um ebendiese Änderung der Verteilung auszunutzen, werden die Daten partitioniert, d. h. ein Binning vorgenommen. Das Binning wird so optimiert, dass die Partitionen möglichst groß sind, gleichzeitig jedoch in sich eine möglichst stationäre Verteilung haben.

Um die Verfahren zu evaluieren werden sie als RapidMiner-Operatoren implementiert. Hinzu kommen einige Hilfsoperatoren sowie ein Framework, um Parameter- und Operatorvariationen in RapidMiner auf einem Cluster zu berechnen. Das folgende Kapitel beschreibt die zur Evaluierung genutzte Infrastruktur sowie die Organisation der Daten. Kapitel 10 gibt einen Überblick über die verwendeten Daten. In Kapitel 11 wird der *Recall Chooser* beschrieben, der auf Grundlage eines gewünschten Recalls einen Operation Point auf der ROC-Kurve auswählen kann. Kapitel 12 beschreibt und evaluiert Fuzzy Ada^{2}Boost, Kapitel 13 VLDS-Ada^{2}Boost und Kapitel 14 das Binning-Verfahren. Dann werden die Verfahren in Kapitel 15 mit dem Random Forest verglichen, der in der Standard-MARS-Analyse zum Einsatz kommt. Abschließend gibt Kapitel 16 einen zusammenfassenden Überblick über die neuen Verfahren.

9. Technische Umsetzung der Experimente

Zur Bewertung der neuen und bestehenden Verfahren wurden Testreihen auf den Daten des MAGIC-Experiments durchgeführt. Größtenteils fanden diese Tests im Kern innerhalb der Data Mining-IDE RapidMiner statt. Die benöigte Rechenleistung stellte das Rechencluster *PhiDo* der Fakultät Physik der Technischen Universität Dortmund zur Verfügung. Die Erzeugung von Testreihen und die Verwaltung der Ergebnisse wurde halbautomatisch durch ein vom Autor entwickeltes Framework übernommen. Die in den Experimenten genutzten Daten werden von einer SQL-Datenbank gehalten. Ursprünglich liegen diese Daten im ROOT-Format vor und wurden durch ein ebenfalls im Rahmen dieser Arbeit entwickeltes Programm in die Datenbank übernommen.

9.1. RapidMiner

RapidMiner [25] ist eine Entwicklungsumgebung für Data Mining. Die Entwicklung von RapidMiner begann 2001 unter dem Namen YALE am Lehrstuhl für künstliche Intelligenz an der Technischen Universität Dortmund. Später wurde die Entwicklung von der Firma Rapid-I übernommen, die das Programm unter einem dualen Lizenzmodell vertreibt und unter einer der Open-Source-Lizenz AGPL kostenlos zur Verfügung stellt.

RapidMiner ist Java-basiert und ist auf den gängigen Betriebssystemen lauffähig. Die GUI-Oberfläche von RapidMiner erlaubt die Kombination verschiedener sogenannter Operatoren, die viele Funktionen aus den Bereichen Datenimport, Vorverarbeitung, Modellerzeugung und Evaluierung zur Verfügung stellen. Dadurch wird Rapid Prototyping von Data Mining-Prozessen ermöglicht. Eigene Algorithmen können als zusätzliche Operatoren in Form sogenannter *Extensions* integriert werden.

Die Prozesse werden in einem XML-Format abgespeichert. Sie können direkt in der GUI ausgeführt und ausgewertet werden. Ein Konsolenprogramm ermöglicht weiterhin die Ausführung der Prozesse in einer Umgebung ohne graphische Oberfläche, wie beispielsweise einem Rechnencluster.

Für diese Arbeit wird RapidMiner in der Version 5.1 verwendet, modifiziert durch einige im Rahmen dieser Arbeit entstandene Patches zur Beseitigung von Fehlern (s. Anhang B). Diese Patches führen keine neue Funktionalität ein, sondern dienen lediglich der Korrektur von fehlerhaftem Verhalten. Sämtliche selbst implementierten Algorithmen und Operatoren sind in einer Extension gekapselt. Sie werden in den Kapiteln 11, 13 und 14 beschrieben. Zusätzliche Informationen liefert die integrierte Hilfe der Extension.

9.2. Organisation der Daten

Die Daten, die die MAGIC-Teleskope aufnehmen, werden wie zuvor beschrieben durch eine Kette von Analyseprogrammen verarbeitet. Sowohl die Rohdaten als auch die Ausgabe jedes Programms werden im Format des C++-Frameworks ROOT abgelegt. Daten in diesem Format können nicht direkt in RapidMiner verwendet werden, da kein entsprechender Import-Operator existiert und eine entsprechende, ausreichend flexible Implementierung nicht trivial ist. In früheren Arbeiten([21]) wurden die Daten in ein einfaches ASCII-basiertes Format konvertiert. Das in dieser Arbeit entwickelte Verfahren VLDS-Ada^{2}Boost benötigt jedoch wahlfreien Zugriff auf die gesamte Datenmenge. Daher wird in dieser Arbeit eine SQL-Datenbank als Backend für die Datenhaltung gewählt.

In den folgenden Abschnitten wird die Struktur der von MARS gelieferten Daten erläutert und anschließend das Schema der SQL-Datenbank sowie die Importierung der MARS-Daten in die Datenbank vorgestellt.

9.2.1. Struktur der MARS-Daten

9.2.2. SQL-Datenbank

Die Datenbank enthält eine zentrale Tabelle *melibea* für die gesamten melibea-Daten, sowie einige Tabellen mit Meta-Informationen und Verwaltungsdaten. Letztere spei-

chern unter anderem die Herkunft jedes einzelnen Datensatzes.

Die Tabelle *melibea* hat als Schlüssel einen eindeutigen Index. Die Tabelle *event_set* fasst Datensätze aus *melibea* zu Ereignismengen zusammen. Diesen wiederum wird in der Tabelle *even_set_name* wiederum ein eindeutiger Name zugeordnet. Die Ereignismengen können beispielsweise als Testmenge zur Validierung und Bewertung von gelernten Modellen verwendet werden.

Einige in dieser Arbeit verwendete Verfahren benötigen zusätzlich zufällige Ereignismengen, so dass statische Mengen allein nicht ausreichend sind. Für effiziente SQL-Ausdrücke zum Ziehen zufälliger Beispielmengen aus der Datenbank ist zwingend ein fortlaufender Index für die zu ziehenden Daten erforderlich. Der Index in *melibea* ist jedoch nicht notwendigerweise fortlaufend. Außerdem soll je nach Szenario nur aus einer Teilmenge der Grundgesamtheit gezogen werden, beispielsweise nur Gamma-Ereignisse oder Ereignisse aus einem bestimmten Zenitwinkelbereich. Daher enthält die Datenbank zusätzliche Hilfstabellen, die jedem Ereignis einer solchen Teilmenge einen Index aus *melibea* zuordnen.

Algorithmus 3 zeigt exemplarisch ein SQL-Statement zum Ziehen eines Samples, welches aus 10.000 zufällig (mit Zurücklegen) gezogenen Gamma-Ereignissen besteht und disjunkt zu der Ereignismenge *t1* ist: *melibea_m_idx* ist die Hilfstabelle, die einen fortlaufenden Index mit Gamma-Ereignissen verknüpft: Zeile 2 bis 12 ziehen 15.000 zufällige Indizes aus der Hilfstabelle und verknüpfen sie mit den tatsächlichen Daten aus *melibea*. Die WHERE-Bedingung ab Zeile 13 sorgt dafür, dass Ereignisse aus der Ereignismenge *t1* nicht ausgewählt werden. Da im Vorhinein nicht abgschätzt werden kann, wieviele Ereignisse davon betroffen sind, muss die zufällige Auswahl zunächst etwas mehr als die angestrebten 10.000 Ereignisse ziehen.

9.2.3. melibea2sql

melibea2sql ist ein in C++ geschriebenes Programm. Es dient dazu, melibea-Daten im ROOT-Format einzulesen und in die zuvor beschriebene Datenbank zu importieren. Zum Einlesen der Daten wird die native ROOT-Api verwendet, die Verbindung zur Datenbank wird mit Hilfe der Qt-Bibliothek hergestellt.

Als Parameter erhält das Programm eine Liste der ROOT-Dateien, welche eingelesen werden sowie die Verbindungsdaten des SQL-Servers.

9.3. PhiDo

PhiDo ist ein Rechencluster der Fakultät Physik. Es wird seit 2010 von den Lehrstühlen Experimentelle Physik 5 a und b sowie Theoretische Physik 1 und 3 betrieben. Es besteht aus 148 Einzelrechnern oder Knoten mit jeweils acht Prozessoren und 32 GB Arbeitsspeicher. Die Knoten sind mit einer Bandbreite von zweimal 1 Gbit untereinander verbunden. Eine weitere 100 Gbit-Leitung verbindet sie mit zentralen Festspeicher-Servern, die insgesamt ein Datenvolumen von 200 TB speichern können.

Um Prozesse wie beispielsweise ein RapidMiner-Experiment im Cluster berechnen zu lassen, schreibt der Benutzer ein sogenanntes Jobfile. Dies ist ein Shell-Skript, welches den eigentlichen Prozess aufruft. Außerdem enthält es weitere Informationen über den Prozess wie den maximalen Speicherbedarf, Anzahl benötigter Prozessoren sowie die maximale Ausführungszeit. Das Jobfile wird an ein Jobverwaltungssystem

```
SELECT idx, l
FROM melibea RIGHT JOIN (
  SELECT melibea_idx
    FROM melibea_m_idx
    RIGHT JOIN (
      SELECT ROUND( RAND() * (SELECT COUNT(*) FROM melibea_m_idx ) )
        AS val
        FROM melibea LIMIT 15000
    ) AS rnd
    ON rnd.val=melibea_m_idx.i
  ) AS rnd_idx
  ON rnd_idx.melibea_idx=melibea.idx
  WHERE rnd_idx.melibea_idx NOT IN (
    SELECT melibea_idx
      FROM event_set
      WHERE melibea_idx=rnd_idx.melibea_idx AND set_id IN (
        SELECT set_id
          FROM event_set_name
          WHERE name LIKE 't1'
      )
  )
  LIMIT 10000
```

Algorithmus 3: SQL-Statement zum Ziehen eines zufälligen Gamma-Samples ohne Überschneidung zur Testmenge t1

übergeben, welches den Job entsprechend seiner veranschlagten Ausführungszeit in eine von mehreren Warteschlangen einfügt und ausführt, sobald die angeforderten Kapazitäten frei sind. Die Ausgaben des Prozesses werden nach der Ausführung an einen zuvor ebenfalls im Jobfile spezifizierten Ort kopiert.

Im Rahmen dieser Arbeit wurde ein Framework entwickelt, mit dem sich RapidMiner-Experimente halbautomatisch erzeugen und auf dem Cluster ausführen lassen. Außerdem können die Ergebnisse aus dem Cluster abgeholt und graphisch dargestellt werden. Dieser *RapidMiner-PhiDo-Adapter* wird in Anhang A beschrieben.

10. Überblick über die verwendeten Daten

10.1. Herkunft der Daten

Bei den Daten, die für die Evaluation der in dieser Arbeit entwickelten Verfahren verwendet werden, handelt es sich um ein Sample aus off-Beobachtungen und Monte-Carlo-generierten Gamma-Ereignissen (vgl. Kapitel 3.3.4 und 3.4). Die off-Daten wurden im November 2009 in der Nähe des Krebsnebels aufgenommen, das Gamma-Sample wurde mit Hilfe von CORSICA generiert.

Der Zenitwinkelbereich beider Mengen reicht von etwa 5° bis 50°. Da jedoch im Bereich unter 10° und über 40° nur wenige Daten zur Verfügung stehen und sich dort außerdem die Zenitwinkelverteilungen von Gammas und off-Daten stark unterscheiden, werden nur Daten aus dem Bereich von 10° bis 40° genutzt. In diesem Bereich liegen etwa 670.000 Gamma-Beispiele und 545.000 off-Beispiele.

10.2. Merkmale

Die Beispielmenge enthält insgesamt 113 Merkmale. Darin enthalten sind alle MARS-Parameter inklusive der Stereo-Parameter sowie Meta-Daten wie ein fortlaufender Index und genaue Information über die Herkunft jedes einzelnen Beispiels. Da die Daten zum überwachten Lernen verwendet werden sollen, gibt ein Label-Attribut die Klassenzugehörigkeit *Gamma* (kodiert als *m* wie *Monte-Carlo*) oder nicht-Gamma (*o* wie *off*) an.

Außerdem wurden die Daten von melibea vorverarbeitet, so dass die Merkmalsmenge die *Hadroness* enthält. Das ist ein reeller Wert in $[0, 1]$, den der in melibea verwendete Random Forest ausgibt. Er gibt die Konfidenz dafür an, dass ein Beispiel ein Hadron ist[8].

Verwendet werden in allen Experimenten, die im Rahmen dieser Arbeit durchgeführt werden, ausschließlich dieselben zwölf Merkmale, die auch in der Standard-Analyse in MARS genutzt werden (vgl. Kapitel 3.3.6).

10.3. Auswahl von Test- und Trainingsmengen

Allen hier verwendeten Lernverfahren ist gemein, dass sie eine Trainingsmenge benötigen, um daraus geeignete Klassifikationsmodelle zu berechnen. Zur Evaluation der Modelle wird eine Testmenge benötigt. Bei einigen Experimenten wird die Testmenge direkt aus der Trainingsmenge generiert, indem die Modelle einer Kreuzvalidierung unterzogen werden. Bei einigen Experimenten ist die Verwendung einer klassischen Kreuzvalidierung nicht möglich, da die Modelle während des Trainings eigenständig neue Beispiele ziehen, und daher eine vorherige Partitionierung der Trainingsmenge nicht möglich ist. In diesem Fall wird zuvor eine Testsmenge gezogen, und das Modell zieht während des Trainings ausschließlich Beispiele, die nicht in der Testmenge enthalten sind.

Beim Ziehen von Test- und Trainingsmengen stellt sich die Frage, ob diese mit oder ohne Zurücklegen gezogen werden. Für die Evaluierung von Modellen ist dies jedoch unerheblich, insbesondere, wenn das verwendete Gütemaß einer Wahrscheinlichkeit entspricht, wie z. B. Accuracy, Precision, Recall und AUC. Wird die Menge mit Zurücklegen gezogen, so folgt die Verteilung einer Binomialverteilung. Wird dagegen ohne Zurücklegen gezogen, so folgt die Wahrscheinlichkeit der hypergeometrischen Verteilung. Beide Verteilungen nähern sich für große Mengen jedoch der Normalverteilung an, und die Approximation beider Verteilungen durch die Normalverteilung ist identisch [33].

Die algorithmische Komplexität des Ziehens einer Menge von m Beispielen ohne Zurücklegen ist $O(m \log m)$, mit Zurücklegen lediglich $O(m)$. Da das Sampling bei einigen im Rahmen dieser Arbeit entwickelten Algorithmen sehr häufig erfolgt, wird hier die schnellere Variante des Ziehens mit Zurücklegen gewählt.

Die Trainingsmenge ist für alle Experimente identisch, ausgenommen sind diejenigen Experimente, die während des Trainings zufällige Mengen nachziehen. Gam-

[8] Da die negative Beispielmenge wie in Kapitel 3 beschrieben nicht nur Hadronen enthält, ist die Bezeichnung Hadroness eigentlich ungenau und müsste korrekterweise *not-Gammaness* lauten. Der Begriff ist jedoch im MAGIC-Experiment etabliert, so dass er auch in dieser Arbeit verwendet wird. Dementsprechend wird die negative Beispielmenge im folgenden gelegentlich auch als *Hadron-Menge* bezeichnet.

mas und Hadronen werden unabhängig voneinander gezogen, um die a-priori-Wahrscheinlichkeiten in der Trainingsmenge garantieren zu können. Beide Mengen werden als Folge zufälliger Beispiele aus einer Datenbank gezogen. Durch einen konstanten Seed des Zufallszahlengenerators wird sichergestellt, dass alle Experimente dieselbe Trainingsmenge verwenden.

Wird zur Evaluierung eine Testmenge verwendet, so ist diese ebenfalls für alle Experimente identisch und entspricht einer im Vorhinein zufällig ausgewählten Menge von jeweils 80.000 Gammas und Hadronen. Meist werden dann, analog zur Kreuzvalidierung, mehrere Modelle trainiert, die auf Partitionen der Testmenge evaluiert werden.

11. Der Recall-Chooser

Bei der Gamma-Separation in der MAGIC-Analyse wird in der Regel ein weicher Klassifikator mit weichen Ausgaben $h : X \to R$ verwendet. Der üblicherweise in melibea benutzte Random Forest macht Ausgaben im Intervall $[0, 1]$. Um ein Event zu klassifizieren, muss ein Schwellwert gefunden werden, oberhalb dessen alle Events als Gamma klassifiziert werden. In der MAGIC-Analyse wird dieser oft so gewählt, dass ein bestimmter Recall v, üblicherweise 60 % oder 70 %, erreicht wird [23].

Wie in Kapitel 4.4 erläutert wurde, entspricht jeder Punkt, genauer jede Parallele zur x-Achse, einem Recall, da TPr $=$ Recall. Wählt man den Schnittpunkt der Geraden $f(x) = v$ mit der ROC-Kurve als Operation Point, so lässt sich daraus der gewünschte Schwellwert ablesen. Dabei treten jedoch zwei Schwierigkeiten auf. Erstens ist die ROC-Kurve im strengen Sinne keine Kurve, sondern lediglich eine Menge von diskreten Punkten, so dass nicht jeder beliebige Punkt als Operation Point gewählt werden kann. Zweitens können einem Punkt auf der Kurve mehrere Beispiele zugeordnet sein, wodurch der Recall an diesen Punkten springen kann. Auch diese beiden Tatsachen wurden in Kapitel 4.4 erläutert.

Der Recall-Chooser kann also den Recall lediglich so einstellen, dass v auf der Testmenge nicht unterschritten wird. Der genaue Wert v kann im Allgemeinen nicht eingestellt werden.

Bei bekannter Anzahl positiver und negativer Beispiele in der Testmenge kann aus jedem Punkt auf der schwellwertbasierten ROC-Kurve die entsprechende Konfusionsmatrix berechnet werden:

$$p = \mathrm{TPr} \cdot P \tag{81}$$
$$\overline{p} = \mathrm{FPr} \cdot N \tag{82}$$
$$n = (1 - \mathrm{TPr}) \cdot P \tag{83}$$
$$\overline{n} = (1 - \mathrm{FPr}) \cdot N \tag{84}$$

Da ein Recall wiederum einem Punkt auf der ROC-Kurve entspricht, können für den gewählten Recall beliebige Gütemaße aus der Konfusionsmatrix berechnet werden. Diese wären dann ein naheliegendes Maß zum Vergleich mehrerer Klassifikatoren. Könnte der Recall exakt eingestellt werden, wäre das Maß nicht nur naheliegend, sondern auch aussagekräftig. So aber kann unter Umständen für zwei verschiedene Klassifikatoren nicht genau der gleiche Recall eingestellt werden, so dass die Gütemaße aus der Konfusionsmatrix unterschiedlich kalibriert sind und somit bestenfalls ungefähre

```
1  Eingabe:
2    Beispielmenge 𝔼 = (x_1, y_1), ..., (x_m, y_m) mit (x_i, y_i) ∈ X × Y
3    Beispielgewichte w_1, ... w_m
4    Klassifikator h : X → ℝ
5    gewünschter Recall v ∈ [0, 1]
6  Ausgabe:
7    Schwellwert s mit Recall(h, s) ≥ v
8
9  P := Σ_{i_1}^{m} w_i · I(y_i = +1)  // Gewicht aller positiven Beispiele
10 sortiere 𝔼, so dass ∀(x_i, y_i), (x_j, y_j), i <= j : h(x_i) <= h(x_j)
11 n = 0
12 for i = 1; i <= m; i++ do
13   n += w_i if y_i = +1
14   return 1/2 (h(x_i) + h(x_{i-1})) if n/P ≥ 1 − v
15 end
16 return h(m)
```

Algorithmus 4: Wahl des Operation Point

Aussagen zulassen.

Eine zulässige Aussage kann jedoch getroffen werden: trotz der ungleichen Kalibrierung kann ausgesagt werden, wie der Klassifikator in der echten MAGIC-Analyse arbeitet, da dort ja genau dieses Verfahren angewendet wird.

11.1. Wahl des Operation Point

Algorithmus 4 zeigt den Algorithmus, mit dem der Schwellwert s gesucht wird. Als Eingabe erhält er die Beispielmenge $\mathbb{E}$, die m Beispiele aus $X \times Y$ enthält. Zu diesen Beispielen erhält er die Gewichte $w_1, \ldots, w_m$. Falls die Beispiele gleichverteilt sind, kann $w_i = 1$ für alle i gewählt werden. h ist der Klassifikator, für den der Schwellwert gesucht wird, und v ist der Recall, der auf der Testmenge mindestens erreicht werden soll.

Zunächst wird P als die Gewichte aller positiven Beispiele berechnet (Zeile 9). Dann werden die Beispiele aufsteigend nach der Vorhersage $h(x_i)$ sortiert und n mit 0 initialisiert (Zeile 10 und 11).

Die for-Schleife iteriert über alle Beispiele, beginnend bei demjenigen mit der kleinsten Vorhersage $h(x_i)$. Dadurch springt man an den Punkt der ROC-Kurve, der dem Schwellwert $s = h(x_i)$ entspricht. Bei Anwendung des Schwellwerts werden alle Beispiele mit $h(x) \leq h(x_i)$ als negativ klassifiziert, also auch das aktuelle Beispiel (x_i, y_i). Ist $y_i = +1$, so würde es in die Partition FN fallen. Somit wird sein Gewicht zu n addiert.

Der Recall ist definiert als

$$\text{Recall}(h) := \frac{p}{P} = 1 - \frac{n}{P}$$

Übersteigt der Recall diesen Wert, so ist der korrekte Operation Punkt gefunden. Um den zuvor beschriebenen Effekt der Sprünge in der ROC-Kurve auszugleichen, wird als Schwellwert der Mittelwert aus der aktuellen Vorhersage $h(x_i)$ und der Vorhersage

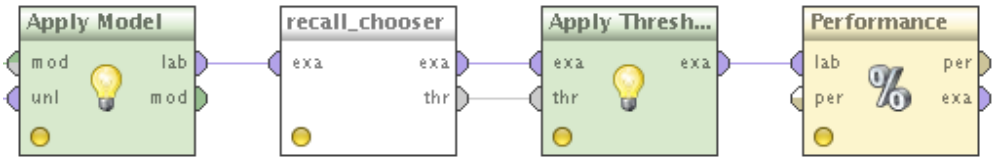


Abbildung 13: Anwendung des Recall-Chooser in einem RapidMiner-Process

für die nächst niedrigere Vorhersage $h(x_{i-1})$ als Schwellwert s zurückgegeben.

Wurde nach Beendigung der for-Schleife kein Schwellwert gefunden, so wird die Vorhersage des letzten Beispiels ausgegeben.

11.2. RapidMiner-Operator

Im Rahmen dieser Arbeit wurde Algorithmus 4 als Operator RECALL CHOOSER für RapidMiner implementiert. Die Beispielmenge erhält er über seinen Eingangsport. Den gewünschten minimalen Recall kann der Benutzer über einen Parameter eingeben. Am Ausgangsport `threshold` liegt nach der Berechnung des Operators der Schwellwert an. Die Beispielmenge wird unmodifiziert an den Port `example set` durchgeschleift.

Da die Eingabemenge gelabelte Daten mit Konfidenzen enthalten muss, wird der Operator in der Regel nach Anwendung eines Modells mit Hilfe des APPLY MODEL-Operators angewendet. Die ausgegebene Threshold kann mit dem Operator APPLY THRESHOLD auf die Daten angewendet werden. Das Ergebnis ist eine Beispielmenge mit harten Labels, d. h. jedem Beispiel ist entweder +1 oder -1 zugeordnet. Enthalten die Daten auch das wahre Label, können beliebige Gütemaße berechnet werden. Abbildung 13 zeigt diesen typischen Anwendungsfall des RECALL CHOOSERs.

12. Konfidenzbasiertes Resampling in Ada²Boost

Ada²Boost verwendet ausschließlich harte Entscheidungsfunktionen. Das bedeutet, dass lediglich das Vorzeichen der Entscheidungsfunktion genutzt wird, selbst wenn das Basismodell Konfidenzen liefert. Wie in Kapitel 6.3 beschrieben, lautet die Umgewichtungsregel in Ada²Boost

$$\begin{aligned} w_{t+1}(x,y) :=& \frac{w_t(x,y)}{\sqrt{\beta_t(h_t(x))}^y} \\ =& w_t(x,y) \cdot \begin{cases} \sqrt{p_t/n_t}^{-y} & \text{für } h_t(x) \geq s \\ \sqrt{\overline{p}_t/\overline{n}_t}^{-y} & \text{für } h_t(x) < s \end{cases} \end{aligned} \tag{85}$$

Diese Regel bietet zwei Ansatzpunkte zur Berücksichtigung von weichen Basisklassifikatoren, die in den beiden folgenden Kapiteln untersucht werden.

In diesem Kapitel sei h eine Funktion $h : X \to [0,1]$. Der Wertebereich ist also beschränkt auf reelle Werte zwischen 0 und 1. Diese Ausgabe soll als Konfidenz dafür interpretierbar sein, dass ein Beispiel der positiven Klasse angehört. Bei Entscheidungsbäumen kann dies beispielsweise sehr leicht erreicht werden: normalerweise gibt ein Entscheidungsbaum das Label der Mehrheitsklasse in dem Blatt aus, in das ein Beispiel

	$y = +1$	$y = -1$
$h(x)$ nahe 1	TP_f $\lvert TP_f \rvert = p_f$	FP_f $\lvert \mathrm{FP}_f \rvert = \overline{p}_f$
$h(x)$ nahe 0	FN_f $\lvert \mathrm{FN}_f \rvert = n_f$	TN_f $\lvert \mathrm{TN}_f \rvert = \overline{n}_f$
	$P = p_f + n_f$	$N = \overline{p}_f + \overline{n}_f$

Tabelle 2: Konfusionsmatrix basierend auf Fuzzy-Mengen

einsortiert wird. Wird stattdessen der Anteil $\frac{p}{p+n}$ der positiven Beispiele in dem Blatt ausgegeben, so liefert $h(x)$ die gewünschte Konfidenz. Ist dabei das Gewicht beider Klassen in dem Blatt etwa gleich, so ist $h(x) \approx 0,5$, so dass die Konfidenzen für beide Klassen etwa identisch sind. Überwiegt stattdessen eine der beiden Klassen, so nähert sich $h(x)$ an 0 oder 1 an, und die Konfidenz der Vorhersage steigt.

Auch viele andere Klassifikatoren lassen sich so modifizieren, dass sie Konfidenzen ausgeben [10, 32].

12.1. Konfidenzbasierte Konfusionsmatrix

Gegeben ein Klassifikator mit Ausgabe der Konfidenz der positiven Klasse. Dann sei

$$\mu_+(x) := h(x) \tag{86}$$

und

$$\mu_-(x) := 1 - h(x) \quad . \tag{87}$$

μ_+ und μ_- geben die Konfidenz für die Zugehörigkeit zur positiven bzw. zur negativen Klasse an. Sie können aber auch als Zugehörigkeitsfunktion zu den Mengen $\mathrm{TP}_f \cup \mathrm{FN}_f$ bzw. $\mathrm{FN}_f \cup \mathrm{TN}_f$ im Sinne der Fuzzy-Logik interpretiert werden (vgl. bspw. [24]). Die Beispiele werden also *ein bisschen* als positiv und *ein bisschen* als negativ klassifiziert. Dadurch werden die eigentlich scharfen Mengen TP, FP, FN und TN zu unscharfen Fuzzy-Mengen TP_f, FP_f, FN_f und TN_f. Ihre Kardinalitäten berechnen sich zu

$$p_f = \sum_{i:y_i=+1} w_i \cdot \mu_+(x_i) \tag{88}$$

$$\overline{p}_f = \sum_{i:y_i=-1} w_i \cdot \mu_+(x_i) \tag{89}$$

$$n_f = \sum_{i:y_i=+1} w_i \cdot \mu_-(x_i) \tag{90}$$

$$\overline{n}_f = \sum_{i:y_i=-1} w_i \cdot \mu_-(x_i) \quad . \tag{91}$$

Daraus ergibt sich dann eine fuzzifizierte Konfidenzmatrix.

Satz 12.1. *Die fuzzifizierte Summen $P_f = p_f + n_f$ und $N_f = \overline{p}_f + \overline{n}_f$ aller positiven bzw. negativen Beispiele identisch zu den nicht fuzzifizierten Summen P bzw. N.*

Der Beweis ergibt sich durch Einsetzen der Definitionen:

$$\begin{aligned} P_f &= p_f + n_f \\ &= \sum_{i:y_i=+1} w_i \cdot [\mu_+(x_i) + \mu_-(x_i)] \\ &= \sum_{i:y_i=+1} w_i \cdot [\mu_+(x_i) + (1 - \mu_+(x_i))] \\ &= \sum_{i:y_i=+1} w_i \\ &= P \end{aligned}$$

Die Rechnung kann analog für N_f durchgeführt werden. □

Aus Satz 12.1 folgt, dass das Gesamtgewicht der Beispielmenge in der konfidenzbasierten Konfusionsmatrix identisch zu dem der in Kapitel 4.2.1 vorgestellten einfachen Konfusionsmatrix ist.

Die einzelnen Partitionen TP_f bis TN_f nutzen jedoch mehr Information, da in die Zuordnung der Beispiele nicht nur die binäre Ausgabe eines harten Klassifikators eingeht, sondern alle Informationen, die der Klassifikator in Form einer reellen Zahl ausgibt.

Zur Berechnung von $\beta_{\mathrm{fss}t}(h_t(x))$ werden nun auch die fuzzifizierten Kardinalitäten verwendet:

$$\beta_{\mathrm{fss}t}(h_t(x)) = \begin{cases} p_{ft}/n_{ft} & \text{für } h_t(x) \geq s \\ \overline{p}_{ft}/\overline{n}_{ft} & \text{für } h_t(x) < s \end{cases} \tag{92}$$

Die Wahl des Schwellwerts s erlaubt hier einen gewissen Freiraum, erfordert jedoch auch eine sinnvolle Wahl von s. In der Praxis wird meist $s = 0,5$ gewählt.

Durch diesen neuen Gewichtungsfaktor werden nun die Fuzzy-Mengen $\mathrm{TP}_f \cup \mathrm{FP}_f$ und $\mathrm{FN}_f \cup \mathrm{TN}_f$ stratifiziert, anstelle der scharfen Mengen TP ∪ FP und FN ∪ TN, wie dies beim unmodifizierten Ada²Boost der Fall ist. Dadurch werden Beispiele, für die der Klassifikator nur unsichere Aussagen treffen kann, weniger stark berücksichtigt als solche mit hoher Konfidenz. Die empirische Auswertung in Kapitel 12.4.1 zeigt, dass dies in der Praxis meistens zu besseren Klassifikationsergebnissen führt.

12.2. Konfidenzbasierte Gewichtung

Ein möglicher Nachteil bei der Berechnung von $\beta_{\mathrm{fss}t}$ ist die Tatasache, dass der Wert s optimiert werden muss. Eine konsequente Weiterführung des Gedankens, die Beispiele Fuzzy-Mengen zuzuordnen, führt dazu, nicht nur bei der Berechnung der Partitionsgrößen auf die Fuzzy-Mengen zurückzugreifen, sondern auch bei der eigentlichen Gewichtung der Beispiele.

Die Zugehörigkeitsfunktionen $\mu_+(x)$ und $\mu_-(x)$ geben an, zu welchen Anteilen ein Beispiel in den Partitionen mit positiv bzw. negativ vorhergesagten Beispielen liegt. Dieses Wissen kann bei der Umgewichtung genutzt werden, indem das neue Gewicht

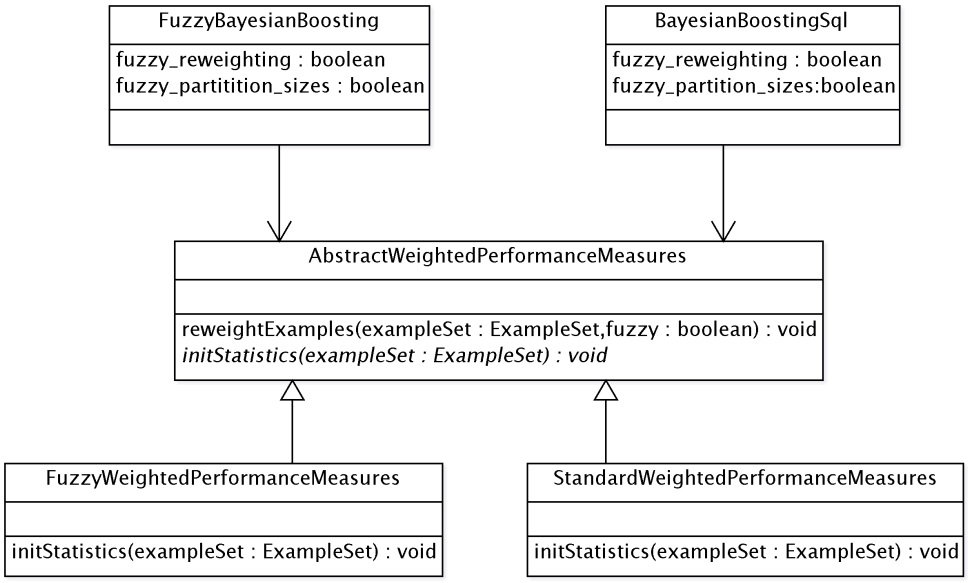


Abbildung 14: Klassendiagramme für die Ada^{2}Boost-Operatoren

anteilig mit $\beta(+1)$ und $\beta(-1)$ berechnet wird. Gleichung (70) wird modifiziert zu

$$\begin{aligned} w_{t+1}(x,y) &= w_t \cdot \left(\mu_+(x) \sqrt{\beta_t(+1)}^{-y} + \mu_-(x) \sqrt{\beta_t(-1)}^{-y} \right) \\ &= w_t \cdot \left(\mu_+(x) \sqrt{\frac{p'_t}{n'_t}}^{-y} + \mu_-(x) \sqrt{\frac{\overline{p}'_t}{\overline{n}'_t}}^{-y} \right) \quad . \end{aligned} \tag{93}$$

Für die Partitionsgrößen p'_t bis $\overline{n}'_t$ können sowohl die scharfe als auch die unscharfe Definition eingesetzt werden.

12.3. Implementierung für RapidMiner

Für RapidMiner wurden zwei Operatoren implementiert, die Ada^{2}Boost mit Fuzzy-Mengen implementieren. Zum einen FUZZY BAYESIAN BOOSTING, zum anderen BAYESIAN BOOSTING (SQL).

FUZZY BAYESIAN BOOSTING ist in der Klasse `FuzzyBayesianBoosting` implementiert. Diese basiert auf der Klasse `BayesianBoosting` aus dem RapidMiner-Kern. Sie wurde um die zwei booleschen Parameter `fuzzy_partition_sizes` und `fuzzy_reweighting` erweitert. `fuzzy_partition_sizes` aktiviert oder deaktivert die Verwendung von Fuzzy-Mengen in der Konfidenzmatrix. `fuzzy_reweighting` gibt an, ob die konfidenzbasierte Gewichtung aus Gleichung (93) oder die ursprünglich in Ada^{2}Boost verwendete Gewichtung aus Gleichung (70) benutzt wird.

Intern wird die Berechnung der Konfusionsmatrix und die Berechnung der Beispielgewichte durch Unterklassen von `AbstractWeightedPerformanceMeasures` übernommen. Je nach Wahl von `fuzzy_partition_sizes` wird entweder eine fuzzifizierte Version oder die ursprüngliche Version mit scharfen Mengen verwendet. Die Be-

rechnung von β findet direkt in `AbstractWeightedPerformanceMeasures` statt; die verwendete Definition wird durch einen Parameter gesteuert. Abbildung 14 zeigt die Beziehung der Klassen und die relevanten Parameter.

BAYESIAN BOOSTING (SQL) wird in Kapitel 13 näher erläutert. Bezüglich der in diesem Kapitel beschriebenen Gewichtungsfunktion verhält er sich jedoch genauso wie FUZZY BAYESIAN BOOSTING.

12.4. Experimentelle Evaluierung

Zur Evaluierung von Fuzzy Ada^{2}Boost werden auf Datensätzen unterschiedlicher Größe Modelle mit allen vorgestellten Varianten trainiert: Fuzzy Ada^{2}Boost mit und ohne konfidenzbasierter Konfusionsmatrix jeweils mit und ohne konfidenzbasierter Neugewichtung[9]. Jede Variante wird sowohl mit einem Decision Stump (WEKA-Implementierung) als auch mit J48 als innerem Lernern auf die Trainingsmenge angewendet.

Die Trainingsmengen unterschiedlicher Größe werden wie in Kapitel 10.3 beschrieben zufällig aus einer Datenbank gezogen, wobei jedoch immer derselbe Seed für den Pseudozufallszahlengenerator verwendet wird, so dass kleinere Trainingsmengen stets Teilmenge jeder größeren Trainingsmenge sind.

Die Area under the ROC-Curve wird in einer zehnfachen Kreuzvalidierung berechnet.

12.4.1. Ergebnisse

Die Abbildungen 15 bis 17 zeigen die Entwicklung der AUC gegen die Anzahl der Boosting-Iterationen für Trainingsmengen mit 10.000, 60.000 und 210.000 Beispielen. Die Legende zeigt, welche Kurve mit welcher Variante erzeugt wurde. Dabei steht frw für *Fuzzy Reweighting*, also die konfidenzbasierte Neugewichtung der Beispiele nach Gleichung (93), und fss für *Fuzzy Set Sizes*, d. h. die konfidenzbasierte Konfusionsmatrix. Weiterhin steht weka-ds für den Decision Stump.

In jedem einzelnen Diagramm fällt auf, dass die Lernkurve bei allen Varianten mit J48 als innerem Lerner erheblich schneller die Sättigung erreichen als alle Varianten mit dem Decision Stump. Bei J48 ist die Sättigung bereits nach etwa 25 Iterationen erreicht, beim Decision Stump geht die erste Kurve erst nach etwa 200 Iterationen in die Sättigung. Dies erklärt sich dadurch, dass J48 ein deutlich stärkeres Lernverfahren ist, das in jeder Iteration mehr Informationen nutzt als ein Decision Stump, und somit in weniger Schritten alle möglichen Informationen aus der Trainingsmenge nutzt.

Beim Vergleich der Fuzzy-Varianten ist ersichtlich, dass der geboostete Decision Stump offenbar nicht von der Fuzzification profitiert. Diese wirkt sich im Gegenteil stark nachteilig aus - die Kurven flachen ab und erreichen in einer akzeptablen Lernzeit und Anzahl Iterationen keine guten Performance-Werte.

J48 hingegen profitiert von der Verwendung der konfidenzbasierten Konfusionsmatrix. Bei allen Trainingsmengen werden dadurch bessere Ergebnisse erzielt: die Kurven erreichen größere Werte, flachen aber nicht ab, so dass die Sättigung nahezu genauso

[9] Die Variante ohne konfidenzbasierte Konfusionsmatrix und ohne konfidenzbasierte Neugewichtung ist identisch zu Ada^{2}Boost nach [35].

schnell Eintritt wie beim unmodifizierten Ada^{2}Boost. Wird zusätzlich die konfidenzbasierte Neugewichtung aktiviert, so werden zumindest bei großen Trainingsmengen minimal höhere AUC-Werte erreicht, die jedoch statistisch nicht signifikant sind. Bei kleinen Trainingsmengen ist die maximal erreichte AUC geringer. Außerdem flacht die Lernkurve im vorderen Teil leicht ab. Wird die konfidenzbasierte Gewichtung ohne die konfidenzbasierte Neugewichtung verwendet, geht die Kurve bei einem deutlich geringeren Wert in die Sättigung.

Vergleicht man das Verhalten des geboosteten J48 mit dem Decision Stump bei Veränderung der Trainingsgröße, so sieht man, dass J48 stärker von einer Vergrößerung der Trainingsmenge profitiert. Bei kleinen Trainingsmengen erreicht der geboostete Decision Stump größere AUC-Werte als J48. Wird auf größeren Mengen trainiert, so verbessert sich zwar die Performance beider Verfahren, die Zuwächse bei J48 sind jedoch größer, so dass die maximale AUC bei einer Trainingsgröße von etwa 60.000 Beispielen zwischen beiden Verfahren identisch ist und bei sehr großen Mengen J48 sogar deutlich höhere Werte erreicht.

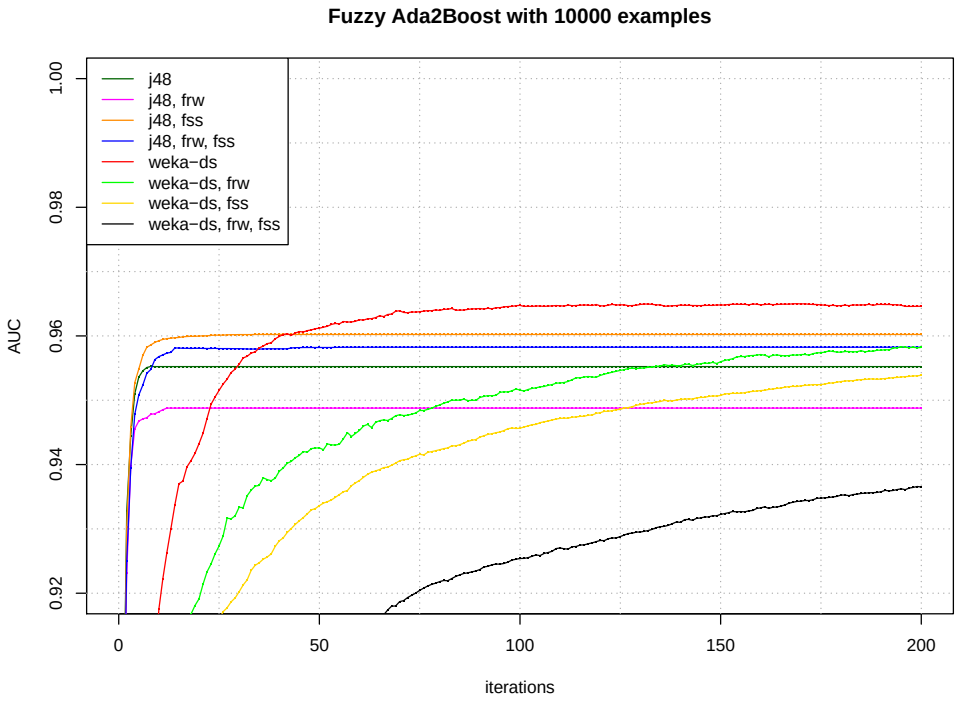


Abbildung 15: AUC von Fuzzy Ada^{2}Boost mit J48 und $m = 10.000$

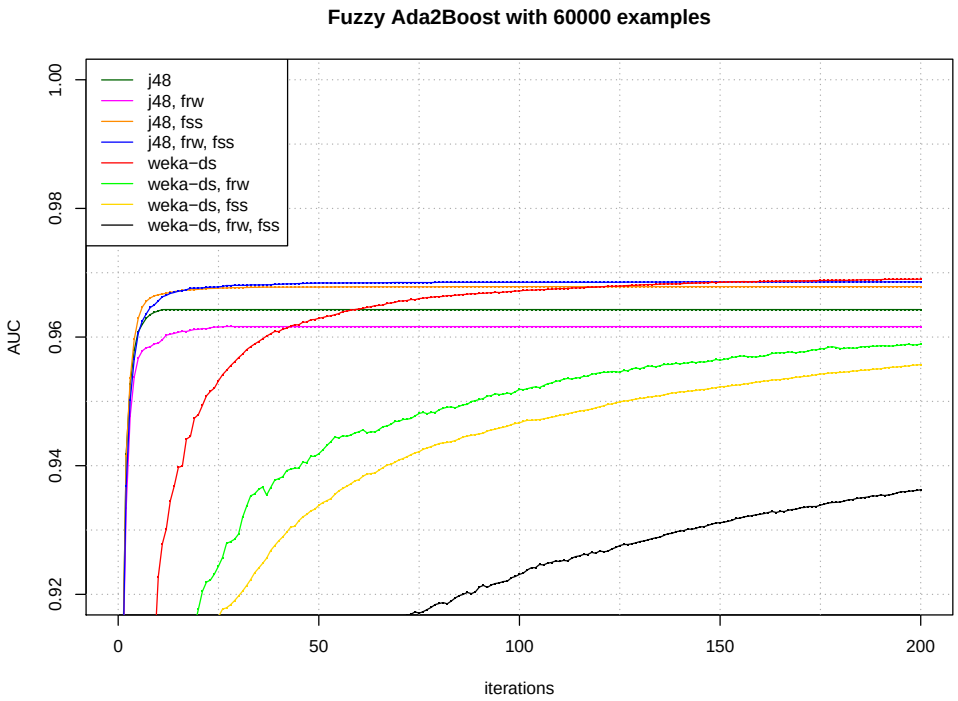


Abbildung 16: AUC von Fuzzy Ada^{2}Boost mit J48 und $m = 60.000$

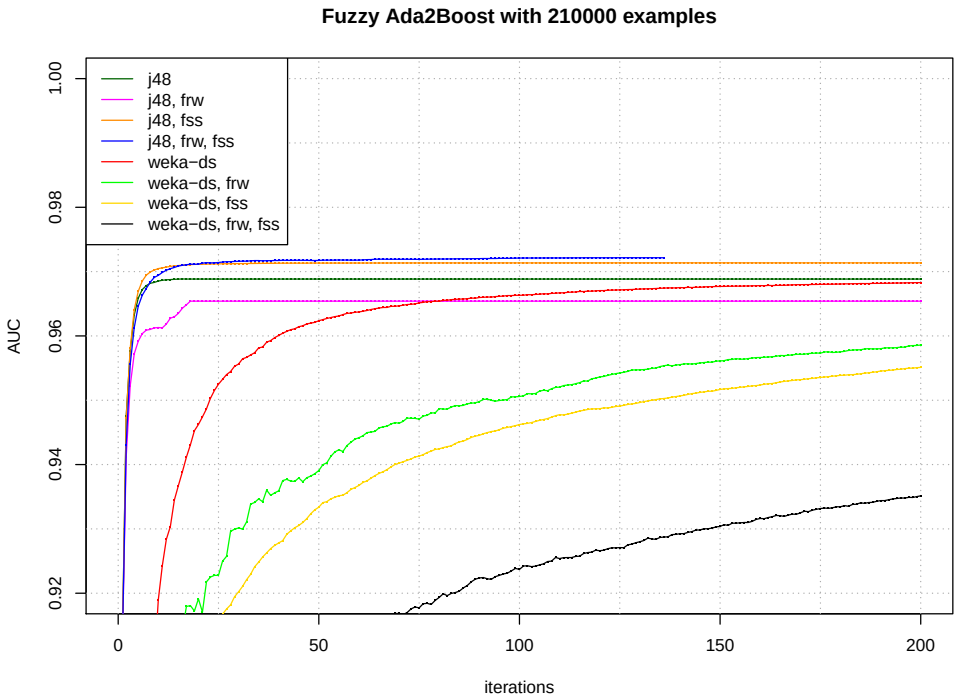


Abbildung 17: AUC von Fuzzy Ada^{2}Boost mit J48 und $m = 210.000$

13. Ada^{2}Boost for Very Large Datasets

Ein Aspekt der Ergebnisse des vorherigen Kapitels ist die Bestätigung der im maschinellen Lernen bekannten Tatsache, dass größere Trainingsmengen in der Regel eine bessere Klassifikationsgüte ergeben. In der Praxis kann die Trainingsmenge jedoch nicht beliebig vergrößert werden, selbst wenn ausreichend Daten vorhanden sind. Die Eigenschaften der Hardware, vor allem die Kapazität des Arbeitsspeichers, beschränken die Trainingsmenge. Ein Ausweg ist die Implementierung von Verfahren, die einen Teil der Daten tiefer in der Speicherhierarchie halten, beispielsweise auf der Festplatte, und nur die gerade benötigten Daten in den Arbeitsspeicher laden. Das bedeutet jedoch einen enormen Implementierungsaufwand, da jeder Algorithmus manuell an diese Art der Speicherverwaltung angepasst werden muss.

Komplexere Lernverfahren, deren Komplexität größer als $O(m)$ ist (mit m als Größe der Trainingsmenge), werden nicht nur durch die Größe des Speichers beschränkt, sondern auch die Laufzeit steigt überproportional an, so dass das Training eines Modells ab einer gewissen Trainingsgröße inakzeptabel lange dauert.

Ada^{2}Boost for Very Large Datasets, kurz *VLDS-Ada^{2}Boost* , nutzt den Ansatz, nicht alle Daten gleichzeitig in den Arbeitsspeicher zu laden, ermöglicht aber gleichzeitig die Benutzung unmodifizierter Lernalgorithmen zum Training. Die einzige Voraussetzung ist, dass die Daten so vorliegen, dass ein wahlfreier Zugriff auf einzelne Beispiele möglich ist. Somit bieten sich Datenbanksysteme für die Datenhaltung an.

VLDS-Ada^{2}Boost setzt auf Ada^{2}Boost auf und nutzt dieselben Verfahren zur Umgewichtung der Beispiele in jeder Iteration wie Ada^{2}Boost und beinhaltet auch die Methoden zur konfidenzbasierten Neugewichtung, die im vorherigen Kapitel beschrieben wurden. Um jedoch mehr Beispiele nutzen zu können, verwirft VLDS-Ada^{2}Boost die aktuelle Trainingsmenge alle R Iterationen und zieht ein neues Sample aus der Datenbank. Bevor das nächste Basismodell trainiert wird, werden die Beispielgewichte der neuen Trainingsmenge nach den bekannten Umgewichtungsregeln neu berechnet. Algorithmus 5 zeigt den Ablauf von VLDS-Ada^{2}Boost in Pseudocode.

Als Eingabe erhält der Algorithmus eine große Trainingsmenge, genauer einen Zeiger darauf, so dass nicht die gesamten Daten in den Arbeitsspeicher geladen werden. Daraus werden an anderer Stelle im Algorithmus zufällige Arbeitsmengen der Größe m gezogen. Dabei werden niemals Elemente aus der Trainingsmenge entfernt; das Ziehen der Arbeitsmenge erfolgt also mit Zurücklegen. Wie bei anderen Boosting-Algorithmen auch enthält die Eingabe die Anzahl der Iterationen. Außerdem wird eine Gewichtungsregel übergeben, die eine der Varianten von Fuzzy Ada^{2}Boost implementiert und aus einem Beispiel, dessen Label und der Vorhersage eines Modells einen Gewichtungsfaktor berechnet (vgl. Kapitel 6.3 und 12). Wie oben beschrieben, gibt R an, wie oft eine neue Arbeitsmenge generiert wird. Die Ausgabe des Algorithmus' ist ein Ensemble-Modell, das die Basismodelle entsprechend ihrer Gewichte und wie in den zuvor genannten Kapiteln beschrieben kombiniert.

Zunächst zieht der Algorithmus eine zufällige Arbeitsmenge $\mathbb{E}_0$ mit der Größe m aus der Menge $\mathbb{E}$ aller Beispiele und initialisiert die Gewichte auf 1 (Zeile 11 und 12). Dann durchläuft der Algorithmus nacheinander alle Iterationen. Ist die Iterationsnummer t *kein* ganzzahliges Vielfaches des Resampling-Intervalls, so verhält sich der Algorithmus identisch zu Fuzzy Ada^{2}Boost (Zeile 21–24): die Arbeitsmenge bleibt unverändert und alle Beispiele der Arbeitsmenge werden entsprechend der Gewichtungsregel in

```
1  Eingabe:
2    Zeiger auf große Beispielmenge 𝔼
3    Größe m der Arbeitsmenge
4    Anzahl der Iterationen k
5    Resampling−Intervall R
6    Gewichtungsregel W : X × Y × ℝ → ℝ
7  Ausgabe:
8    Modell h : X → ℝ
9
10
11 Initialisiere Arbeitsmenge 𝔼_0 := random_subset(𝔼, m)
12 Initialisiere Gewichte w_{0,i} := 1 ∀i ∈ {1,...,m}
13 for t = 1,...,k
14   if t/R ∈ ℕ
15     𝔼_t := random_subset(𝔼, m)
16     w_{0,i} := 1 ∀i ∈ {1,...,m}
17     for j = 1,...,t−1
18       ∀(x_i, y_i) ∈ 𝔼_t : w_{j,i} := w_{j−1,i} · W(x_i, y_i, h_j(x_i))
19     end
20   else
21     𝔼_t := 𝔼_{t−1}
22     if t > 1
23       ∀(x_i, y_i) ∈ 𝔼_t : w_{t,i} := w_{t−1,i} · W(x_i, y_i, h_{t−1}(x_i))
24     end
25   end
26   Trainiere neues Basismodell h_t : X → ℝ auf 𝔼_t
27 end
28 return h : X → ℝ mit h(x) = h(h_1(x),...,h_k(x))
```

Algorithmus 5: Ablauf von VLDS-Ada²Boost in Pseudocode

Abhängigkeit vom wahren Label und der Vorhersage des zuletzt trainierten Basismodells gewichtet.

Im Unterschied zu den bisher beschriebenen Verfahren wird aber alle R Iterationen ein neues Sample gezogen und die alte Arbeitsmenge verworfen (Zeile 15). Die Gewichte werden wieder zunächst mit 1 initialisiert. Danach iteriert eine Schleife über alle bisher trainierten Basismodelle und gewichtet jeweils die Beispiele um. Dadurch wird die neue Arbeitsmenge genauso behandelt, wie es der Fall wäre, wenn kein Resampling stattfinden würde.

Bereits trainierte Basismodelle werden *nicht* neu trainiert, sondern bleiben unverändert. Nur die neuen Basismodelle nutzen Informationen aus der neuen Arbeitsmenge. Die in den vorherigen Iterationen gewonnenen Informationen bleiben erhalten, so dass das Ensemble nach und nach Wissen über eine beliebig große Datenmenge sammelt.

Dadurch, dass nicht nur Beispiele umgewichtet und dadurch die Verteilungen angepasst werden, sondern den Basislernern auch beliebig häufig neue Beispiele zugeführt werden, lassen sich auch stärkere Klassifikatoren boosten, die normalerweise nicht oder nur gering vom Boosting profitieren. Der Performance-Gewinn basiert dann weniger auf der Umgewichtung, als vielmehr auf dem Austausch der Arbeitsmengen,

was einer Vergrößerung der Trainingsmenge gleichkommt. In Kapitel 13.4 wird dies am Beispiel der Support Vector Machine demonstriert.

13.1. Algorithmische Komplexität

Bei Basisklassifikatoren, deren Laufzeit größer als $O(m)$ ist, bringt das iterative Boosten von Klassifikatoren auf kleinen Arbeitsmengen gegenüber dem Training eines einzelnen Modells auf einer entsprechend großen einzelnen Trainingsmenge eine enorme Reduzierung der Laufzeit mit sich.

Sei $O(f(m')) > O(m')$ die Laufzeit für das Training eines Basisklassifikators auf m' Beispielen. Dann ist die Laufzeit auf einer einzelnen Trainingsmenge der Größe M entsprechend $O(f(M))$. Bei VLDS-Ada^{2}Boost werden stattdessen in k Iterationen k Basisklassifikatoren auf Arbeitsmengen der Größe m trainiert. Die Arbeitsmenge wird alle R Iterationen neu gezogen. R wird dabei so gewählt, dass die Zahl der genutzten Beispiele der Größe der großen Trainingsmenge entspricht, also $R = \frac{k}{M} \cdot m$. Die Komplexität zum Training des Ensembles ist dann $O(k \cdot f(m))$. Hinzu kommt jeweils die Zeit, die zum Ziehen der Trainings- bzw. Arbeitsmengen benötigt wird. Diese ist beim Ziehen mit Zurücklegen ebenfalls durch $O(m)$ bzw. $O(M)$ beschränkt, so dass durch das Resampling die asymptotische Laufzeit nicht beeinflusst wird.

Die Betrachtung der asymptotischen Laufzeiten zeigt, dass die Reduzierung umso stärker ausfällt, je größer die Differenz zwischen M und m ist und je schneller $f(m')$ wächst.

Es profitiert beispielsweise die SVM mit ihrer quadratischen Laufzeit bei größeren Trainingsmengen sehr stark von VDSL-Ada^{2}Boost. Es zeigen sich aber (neben der Steigerung der Klassifikationsgüte) auch deutliche Laufzeitverbesserungen bei der Verwendung von J48 mit VLDS-Ada^{2}Boost.

13.2. Vergleich zu bestehenden Verfahren

Wie auch Datenstrom-Algorithmen verarbeitet VLDS-Ada^{2}Boost große Datenmengen, indem iterativ kleinere Untermengen betrachtet werden. Im Gegensatz zu Datenstrom-Algorithmen hat VLDS-Ada^{2}Boost jedoch wahlfreien Zugriff auf die gesamte Datenmenge. Ziel von VLDS-Ada^{2}Boost ist die Erzeugung eines globalen Modells, welches jedes Beispiel, das derselben Verteilung entstammt wie die Trainingsdaten, mit hoher Wahrscheinlichkeit korrekt klassifiziert. Durch den wahlfreien Zugriff auf die Daten und die Verwendung zufällig generierter Datensätze ist gewährleistet, dass die Verteilung der Vereinigung aller zum Training verwendeten Arbeitsmengen $\bigcup_{t=1}^{k} \mathbb{E}_t$ der Verteilung aller zur Verfügung stehenden Daten $\mathbb{E}$ stark ähnelt, auch wenn einzelne Teilmengen von $\mathbb{E}$ Abweichungen von dieser Verteilung zeigen.

In einem Datenstrom dagegen sind die Daten häufig so sortiert, dass sich die Verteilungen der Beispiele vom Anfang des Stroms bis zum Ende hin verändern und nicht stationär sind. Das kann beispielsweise dann der Fall sein, wenn die Daten aus unterschiedlichen Quellen stammen und entsprechend sortiert abgespeichert sind, und der Datenstrom die Beispiele der Reihenfolge im Speicher entsprechend liefert. Werden in einem solchen Fall nicht alle zur Verfügung stehenden Daten zum Training verwendet, sondern nur der Anfang des Stroms, so wird unter Umständen ein lokales Modell trainiert, das lediglich aus der Verteilung der vorderen Beispiele lernt. Beispiele,

die der Verteilung vom Ende des Stroms entsprechen, werden dann unter Umständen nur schlecht klassifiziert. Wenn es möglich ist, einen Datenstrom zu randomisieren, dann entsprechen Batches dieses Stroms jedoch genau den Arbeitsmengen von VLDS-Ada^{2}Boost.

Andere Stream-Boosting-Verfahren wie beispielsweise *OzaBoost* [27] aktualisieren während des Boostings bereits trainierte Modelle. Das setzt entweder aktualisierbare Basismodelle voraus, oder aber die Streamdaten müssen gecached werden, wodurch eine Voraussetzung von Stream-Algorithmen verletzt wird.

KBS-Stream [35] ist ein Datenstrom-Algorithmus zum Erzeugen von Ensembles auf Streams, der wie Ada^{2}Boost und VLDS-Ada^{2}Boost wissensbasiertes Sampling verwendet. Dieser Algorithmus ist jedoch darauf ausgelegt, sich nicht-stationären Verteilungen anzupassen und verwirft unter Umständen bereits gelernte Modelle, wenn sich die Verteilungen im Datenstrom verschieben. Dadurch wird aber gerade kein globales Modell erzeugt, sondern ein an die aktuellen Verteilungen angepasstes lokales Modell.

13.3. Implementierung für RapidMiner

VLDS-Ada^{2}Boost wurde im Rahmen dieser Arbeit als BAYESIAN BOOSTING (SQL) für RapidMiner implementiert. Das bevorzugte Backend zum Zugriff auf die Daten ist das SQL-Interface von RapidMiner, jedoch können auch beliebige andere Schnittstellen zum Datenzugriff verwendet werden.

Der Operator besitzt zwei Unterprozesse. Der eine dient zum Ziehen neuer Arbeitsmengen und wird initial zu Beginn der Berechnung sowie alle R Iterationen aufgerufen. Darin können beliebige Operatoren zum Einsatz kommen, die ein zufälliges ExampleSet liefern können. R wird über den Parameter `new sample interval` eingestellt. Die übrigen Parameter entsprechen denen von FUZZY BAYESIAN BOOSTING.

13.4. Experimentelle Evaluierung

Zur Evaluierung wird VLDS-Ada^{2}Boost mit unterschiedlichen Parametern auf der Datenbank mit den Gamma- und Hadron-Ereignissen angewendet. Insbesondere wird das Resampling-Intervall R von 1 bis 20 sowie die Größe der Trainingsmenge von 2.000 bis 20.000 variiert. Als innerer Lerner kommen erneut J48 und ein Decision Stump zum Einsatz.

Um zu zeigen, dass auch Verfahren, die normalerweise nicht vom Boosting profitieren, eine Verbesserung der Klassifikationsgüte erfahren, wird auch eine Support Vector Machine mit radialem Kern eingesetzt. Aufgrund der etwa quadratischen und somit hohen Laufzeit der SVM (vgl. Kapitel 5.2) wird sie nur mit einem kleinen Datensatz von $m = 400$ trainiert. Es wird die Implementierung libSVM [7] verwendet. Diese Implementierung nutzt die Sequential Minimal Optimization zur Optimierung.

Da der Lernalgorithmus selbstständig Beispiele aus der Datenbank zieht, kann die Kreuzvalidierung von RapidMiner nicht genutzt werden. Stattdessen werden nacheinander zehn Modelle mit identischen Parametern trainiert. Diese sind höchstwahrscheinlich nicht vollkommen identisch, da die Arbeitsmengen mit großer Wahrscheinlichkeit jedesmal unterschiedlich sind. Beim Ziehen der Arbeitsmengen wird garantiert, dass eine Testmenge $\mathbb{E}_V$ von jeweils 80.000 Gammas und Hadronen nicht verwendet wird. Das sind etwa 12 % aller Gamma-Events und 14,5 % aller Hadron-Events.

Jedes Modell wird auf eine andere Teilmenge $\mathbb{E}_{V,i}$ der Testmenge angewendet, wobei garantiert wird, das jedes Beispiel der Testmenge von mindestens einem Modell benutzt wird: $\bigcup_{i=1}^{10} \mathbb{E}_{V,i} = \mathbb{E}_V$. Durch das Training mehrerer Modelle und einer großen Testmenge, die für jedes Modell variiert wird, wird das Verhalten einer Kreuzvalidierung angenähert.

13.4.1. Ergebnisse und Diskussion

Abbildung 18 zeigt die Area under the ROC-Curve von VLDS-Ada^{2}Boost mit J48. Die Größe der Arbeitsmenge beträgt $m = 2.000$ Beispiele, das Resampling-Intervall ist auf $R = 3$ festgesetzt. Der Lerner wurde mit allen vier Varianten von Fuzzy Ada^{2}Boost trainiert. Zum Vergleich enthält das Diagramm die Performance von Fuzzy Ada^{2}Boost ohne Resampling mit 20.000 bzw. 30.000 Beispielen. Dargestellt ist nur die in diesem Fall beste Variante von Fuzzy Ada^{2}Boost mit konfidenzbasierter Konfusionsmatrix und konfidenzbasierter Neugewichtung. Weiterhin sind die Iterationen markiert, in denen VLDS-Ada^{2}Boost durch Resampling aus der Datenbank insgesamt erstmals ebensoviele Daten zur Verfügung hat, wie Fuzzy Ada^{2}Boost. Da $m = 2.000$ und $R = 3$ ist dies in der 30. Iteration (20.000 Beispiele) bzw. in der 45. Iteration (30.000 Beispiele) der Fall.

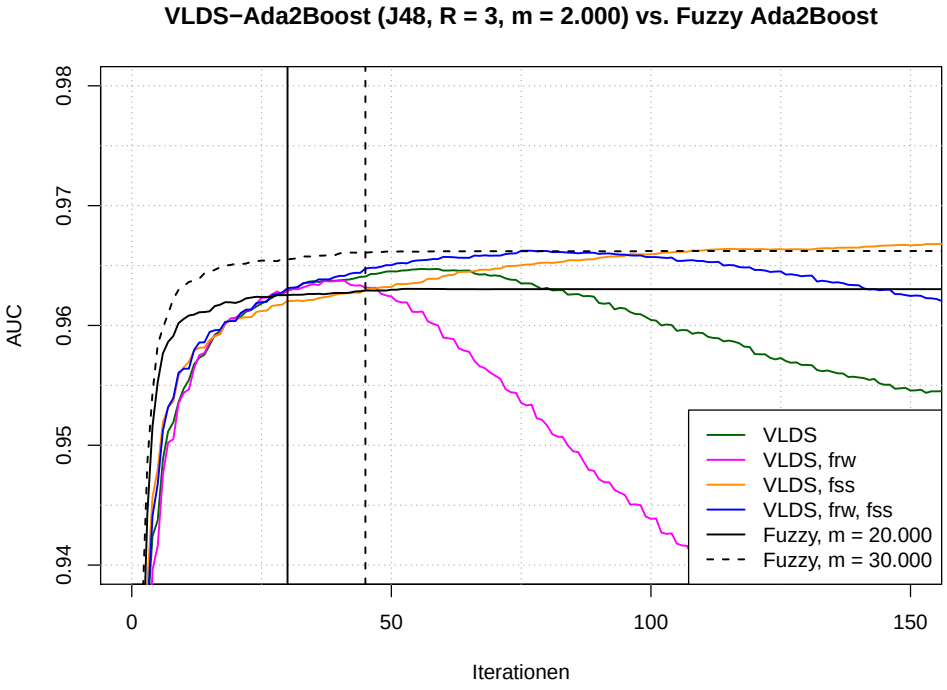


Abbildung 18: VLDS-Ada^{2}Boost mit J48 mit unterschiedlichen Resampling-Intervallen im Vergleich zu Fuzzy Ada^{2}Boost mit J48, trainiert auf 20.000 bzw. 30.000 Beispielen. Die gezeigten Kurven zu Fuzzy Ada^{2}Boost wurden bei aktivierter konfidenzbasierter Konfusionsmatrix und konfidenzbasierter Neugewichtung erzeugt.
Die vertikalen Linien geben an, bei welcher Iteration die Zahl aller von VLDS-Ada^{2}Boost verwendeten Beispiele erstmals 20.000 bzw. 30.000 Beispiele erreicht.

Es fällt auf, dass die Varianten von VLDS-Ada^{2}Boost ein bestimmtes Niveau errei-

chen, in weiteren Iterationen aber stark abfallen. Dies steht im Zusammenhang mit der Gewichtung der Base Classifier in Ada^{2}Boost und wird weiter unten diskutiert.

Das Maximum aller Kurven liegt oberhalb der Sättigungslinie des unmodifizierten Fuzzy Ada^{2}Boost mit 20.000 Beispielen. VLDS-Ada^{2}Boost mit FSS erreicht sogar das Niveau von Fuzzy Ada^{2}Boost mit 30.000 Beispielen. Die Basislerner in VLDS-Ada^{2}Boost verwenden jedoch stets nur 2.000 Beispiele gleichzeitig, wodurch die Laufzeit und der Speicherbedarf dieses Verfahrens deutlich unter dem Bedarf des monolithischen Verfahrens liegt. Da die Arbeitsmenge jedoch regelmäßig ausgetauscht wird, kann dennoch die Güte eines Klassifikators erreicht werden, der etwa 10 bis 15 mal mehr Beispiele im Speicher hält.

Die Kurven in Abbildung 19 zeigen ein ähnliches Verhalten. Sie zeigen die AUC von VLDS-Ada^{2}Boost mit J48 mit $m = 20.000$ und unterschiedlichen Resampling-Intervallen R. Zum Vergleich ist hier die Performance von Ada^{2}Boost mit 210.000 Beispielen mit Fuzzification (FRW und FSS, gestrichelte schwarze Linie) und ohne Fuzzification (durchgehende schwarze Linie) gezeigt. Das Größenverhältnis der Arbeitsmenge von VLDS-Ada^{2}Boost und der Trainingsmenge in Fuzzy Ada^{2}Boost beträgt ebenfalls etwa 1:10. Das Niveau des unmodifizierten Ada^{2}Boost wird bei jedem R erreicht, während die Güte des fuzzifizierten Ada^{2}Boost nicht erreicht wird. Von den VLDS-Varianten liefert VLDS-Ada^{2}Boost ohne Fuzzification außer bei sehr kleinem R die beste Klassifikationsgüte.

Zwar liegt die Klassifikationsgüte von Fuzzy Ada^{2}Boost bei gleicher Anzahl betrachteter Beispiele meist über der von VLDS-Ada^{2}Boost. Bei der Bewertung der Verfahren sollte jedoch nicht nur die Klassifikationsgüte, sondern auch die Laufzeit der Verfahren betrachtet werden. Es wurde bereits gezeigt, dass die asymptotischen Laufzeiten von VLDS-Ada^{2}Boost unter denen von Ada^{2}Boost mit nur einer großen Trainingsmenge liegen. Bei großen Datensätzen bestätigt sich dies in der Praxis: das Training des Ensembles mit $m = 20.000$ Beispielen, Resampling-Intervall $R = 3$ und $k = 32$ Iterationen, benötigt auf einem Referenzrechner etwa 540 s. In den 32 Iterationen werden etwa 210.000 Beispiele verwendet. Das Ensemble liefert eine AUC von 0,969. Das Training eines *einzelnen* J48-Baumlerners auf ebenfalls 210.000 Beispielen beansprucht hingegen etwa 900 s. Der erzeugte Baum liefert eine AUC von nur 0,944 und liegt damit deutlich unter VLDS-Ada^{2}Boost. Ein Ensemble aus solchen J48-Baumlernern mit FRW und FSS erreicht nach etwa 25 Iterationen die Sättigung, ohne Fuzzification nach etwa 12 Iterationen. Die Trainingszeit dieser Ensembles liegt entsprechend bei etwa 11.000 s bzw. 22.500 s und damit 20–40 mal höher als VLDS-Ada^{2}Boost bei nur unwesentlich besserer oder sogar schlechterer AUC.

Wahl des Resampling-Intervalls Die Wahl des Resampling-Intervalls beeinflusst nicht nur die horizontale Streckung oder Stauchung der Kurven, sondern auch die Höhe des Maximums. Bei Betrachtung der Kurven für $R = 20$ fällt auf, dass die Performance nach jedem Resampling zunächst stark ansteigt, dann aber einige Iterationen benötigt, um die Sättigung zu erreichen. Wird R zu klein gewählt, so fällt das nächste Resampling in die Steigungsphase der Lernkurve, so dass die alte Arbeitsmenge verworfen wird, bevor alle nutzbaren Informationen von den Basismodellen aufgenommen wurden (vgl. Kurven für $R = 1$). Bei zu großem R wird die Sättigungsphase

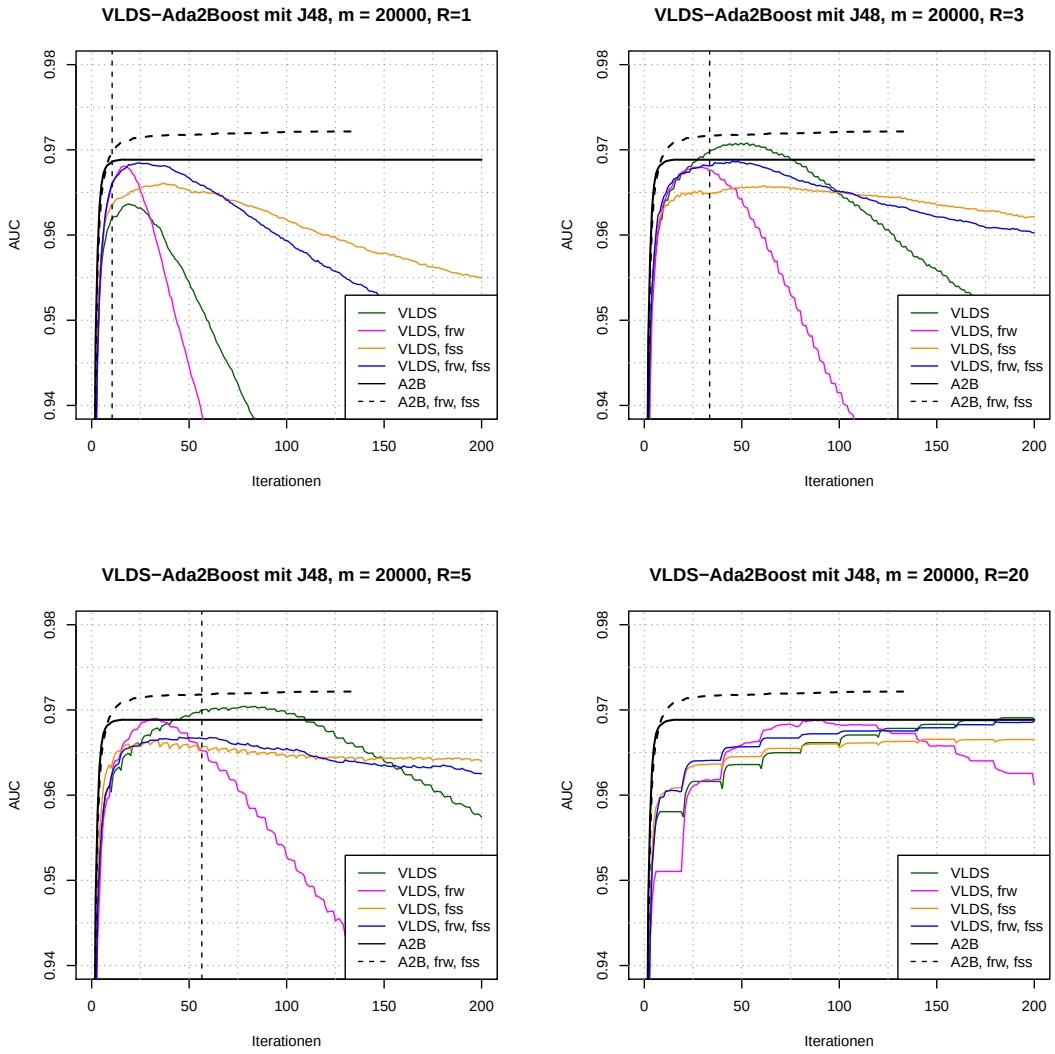


Abbildung 19: Performance von VLDS-Ada^{2}Boost mit J48 mit 20.000 Beispielen bei verschiedenen Resampling-Intervallen. Zum Vergleich enthalten die Diagramme die Performance von Ada^{2}Boost mit 210.000 Beispielen mit und ohne konfidenzbasierter Neugewichtung als schwarze Linien. Die vertikalen Linien geben an, in welcher Iteration VLDS-Ada^{2}Boost erstmals insgesamt 210.000 Beispiele betrachtet hat, also genausoviele Beispiele wie Ada^{2}Boost ohne Resampling.

bei jedem Resampling verlängert, so dass Basismodelle trainiert werden, ohne dass es zu einem Performanceanstieg kommt. Dadurch wird die Laufzeit zum Training des Boosting-Ensembles unnnötig verlängert. R sollte also so gewählt werden, dass sofort bei Erreichen des Sättigungsniveaus ein neues Sample gezogen wird, ohne aber die Steigungsphase zu unterbrechen.

Abfall der Kurven Bei erneuter Betrachtung der unteren Diagramme in Abbildung 19 ist zu erkennen, dass die Güte nach dem Resampling für eine Iteration sinkt, be-

vor die zusätzlichen Beispiele eine Performancesteigerung bewirken. Insbesondere bei den grünen und gelben Kurven, die ohne FRW erzeugt wurden, ist dieser *Resampling-Schock* durchgehend zu beobachten. Bei den anderen beiden Kurven tritt der Effekt erst nach Erreichen des Maximums auf. Ursache für den Resampling-Schock ist ein Overfitting an die vorherige Arbeitsmenge, gepaart mit der Gewichtungsregel für die Basismodelle: die Arbeitsmengen werden zwar nach derselben Verteilung aus der Grundgesamtheit gezogen. Da es sich um einen statistischen Prozess handelt, unterscheiden sich die Stichprobenverteilungen einzelner Arbeitsmengen jedoch leicht voneinander. Haben die ersten $t-1$ Basismodelle die Stichprobe $\mathbb{E}_{t-1}$ sehr detailliert beschrieben

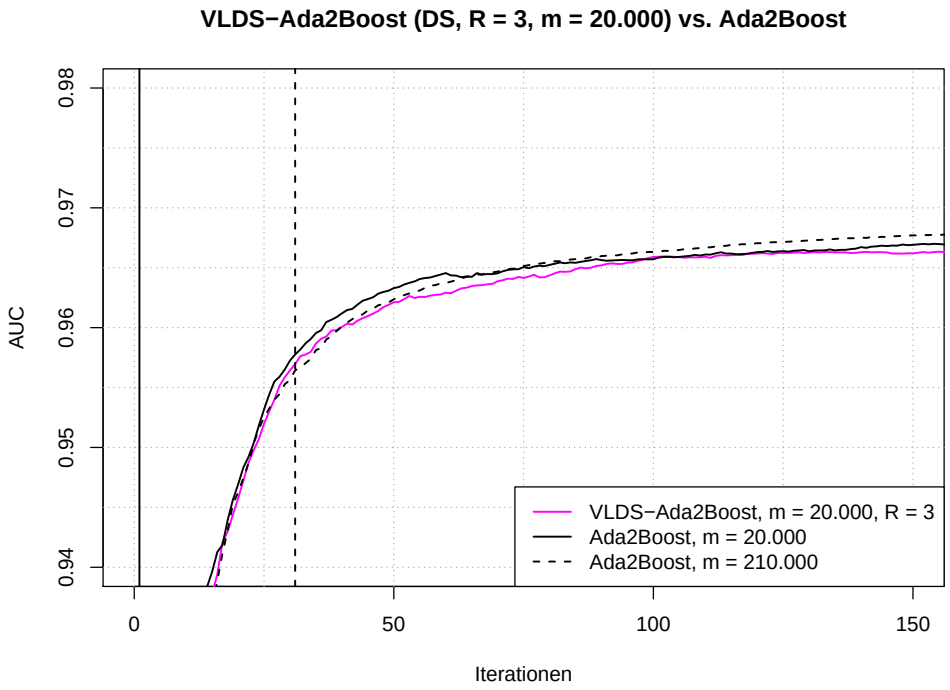


Abbildung 20: VLDS-Ada^{2}Boost mit Decision Stump im Vergleich zu unmodifiziertem Ada^{2}Boost mit Decision Stump, trainiert auf 20.000 bzw. 210.000 Beispielen. Die vertikalen Linien geben an, bei welcher Iteration die Zahl aller von VLDS-Ada^{2}Boost verwendeten Beispiele erstmals 20.000 bzw. 210.000 Beispiele erreicht.

und dadurch komplesxes Vorwissen generiert, so versucht VLDS-Ada^{2}Boost die neue Arbeitsmenge $\mathbb{E}_t$ so umzugewichten, dass dieses Vorwissen entfernt wird. Je komplexer dieses Vorwissen ist, desto größer ist die Gefahr, dass ein hoher Generalisierungsfehler auftritt. Gleichzeitig sind die (ungewichteten) Partitionen FP und FN der Konfusionsmatrix jedoch relativ klein, da sie nur falsch klassifizierte Beispiele enthalten, deren Zahl nach einigen Iterationen aufgrund der Funktionsweise des Boostingverfahrens stark abnimmt. Durch den Stratifizierungsschritt wird diesen wenigen Beispiele ein hohes Gewicht zugeordnet.

Durch den Generalisierungsfehler des Vorwissens auf der neuen Arbeitsmenge $\mathbb{E}_t$ kommt es nun dazu, dass mehr Beispiele in FP und FN fallen. Die Partitionen sind meist trotzdem immer noch deutlich kleiner als die „guten“ Partitionen TP und TN, so

dass die Beispiele weiterhin ein hohes Gewicht haben. Wird nun ein neues Basismodell h_t trainiert, so wird es einige der falschklassifizierten Beispiele korrekt klassifizieren und erhält wegen des großen Gewichts der korrigierten Beispiele selbst ein hohes Gewicht. Da es aber in Wirklichkeit lediglich wenige Einzelbeispiele korrigiert, neigt auch h_t zu einem hohen Generalisierungsfehler. Durch sein großes Gewicht wirkt es sich daher negativ auf die Performance des Ensembles aus.

In der nächsten Iteration $t + 1$ wird das Vorwissen aus h_t dem gesamten Vorwissen hinzugefügt. Damit ist das Vorwissen dann an die geänderte Stichprobenverteilung von $\mathbb{E}_t$ angepasst, und neue Basismodelle erbringen wieder eine Performance-Steigerung.

Wird kein Resampling verwendet, so ändert sich die Stichprobenverteilung der Trainingsmenge nicht plötzlich, und die ungewichtete Kardinalität von FP und FN steigt nicht plötzlich stark an, um dann korrigiert zu werden, so dass die oben beschriebenen schlecht generalisierenden Basismodelle mit hohem Gewicht nicht erzeugt werden.

Durch Verwendung von FSS, also der konfidenzbasierten Konfusionsmatrix, kann der Resampling-Schock abgemildert werden, da sich die Partitionen nicht so stark verändern, wenn sich das vorhergesagte Label von Beispielen nur mit geringer Konfidenz ändert, da sich die Zugehörigkeitsfunktionen für die Zugehörigkeit zu den entsprechenden Mengen nur geringfügig ändert, die Partitionsgrößen dadurch weniger stark variieren und h_t ein geringeres Gewicht erhält.

VLDS-Ada^{2}Boost mit Decision Stump und SVM Abbildung 20 zeigt die AUC von Ada^{2}Boost mit einem Decision Stump als Basisklassifikator. Dargestellt sind die Kurven eines unmodifzierten Ada^{2}Boost mit 20.000 bzw. 210.000 Beispielen, sowie VLDS-Ada^{2}Boost mit $m = 20.000$ und Resampling-Intervall $R = 3$. Auf die Darstellung der Kurven mit konfidenzbasierter Gewichtung wird verzichtet, da diese ähnlich schlechte Ergebnisse liefern wie in Kapitel 12.4.1.

Es zeigt sich, dass Ada^{2}Boost mit Decision Stump nicht vom Ziehen neuer Arbeitsmengen profitiert. Dies ist die Konsequenz daraus, dass dieses Verfahren an sich kaum von einer Vergrößerung der Trainingsmenge profitiert, was sich auch darin äußert, dass sich die beiden Kurven für Ada^{2}Boost ohne Resampling trotz des großen Größenunterschieds der Trainingsmenge kaum unterscheiden. Auch dies entspricht den Ergebnissen aus Kapitel 12.4.1.

Andererseits ist auch kein Resampling-Schock erkennbar. Das liegt daran, dass der Decision Stump ein sehr schwacher Klassifikator ist und daher nicht zu Overfitting neigt. Wie im vorherigen Abschnitt erläutert, tritt der Resampling-Schock aber umso stärker auf, je komplexer der Basisklassifikator ist.

Abbildung 21 bestätigt diese Erklärung. Dort ist die AUC für VLDS-Ada^{2}Boost mit SVM dargestellt. Die SVM ist ein sehr starker Klassifikator und wird in der Regel nicht als Basisklassifikator in Boosting-Verfahren verwendet, da sie kaum von Boosting profitiert und selbst ein Boosting-ähnliches Verhalten zeigt [13]. Dies in in dem Diagramm für $R = 10$ ersichtlich: die Performance bleibt zwischen den Resampling-Schritten nahezu konstant.

Man sieht jedoch auch, dass der Resampling-Schock so schnell auftritt, dass die Kurven fast sofort stark abfallen, sofern keine konfidenzbasierte Konfusionmatrix verwendet wird. Erst die Kombination von konfidenzbasierter Konfusionsmatrix und Resamp-

ling ermöglicht das Boosting der SVM. Die Verwendung der konfidenzbasierten Konfusionsmatrix wirkt sich deshalb so stark aus, weil die SVM für jedes Beispiel individuelle Konfidenzen ausgibt. Dadurch kann der Resamling-Schock so weit abgefangen werden, dass das Verfahren selbst bei $R = 1$ sehr gute Ergebnisse liefert, die die Güte von größeren R sogar übersteigen.

Die AUC-Kurven der gezeigten Ensembles erreichen ihr Maximum bei etwa 50 Iterationen. Das Ensemble mit $R = 1$ hat bis dahin 20.000, das Ensemble mit $R = 5$ 4.000 Beispiele genutzt. Die Berechnung von 50 Iterationen bei $R = 1$ dauert auf einem Referenzrechner etwa 140 s, bei $R = 5$ etwa 40 s. Das Training einer einzelnen SVM auf 4.000 Beispielen dauert 6 s und liefert dieselbe Performance wie das entsprechende Ensemble. Das Training auf 20.000 Beispielen benötigt zwar mit 480 s etwa 3,5 mal mehr Zeit als das Training des Ensembles mit $R = 1$, liefert mit einer AUC von 0,821 jedoch auch eine deutlich bessere Performance.

Auf Datensätzen dieser Größenordnung macht sich der Vorteil in der asymptotischen Laufzeit also aufgrund des Overhead durch das Nachziehen und die anschließende Umgewichtung der Arbeitsmengen noch nicht bemerkbar.

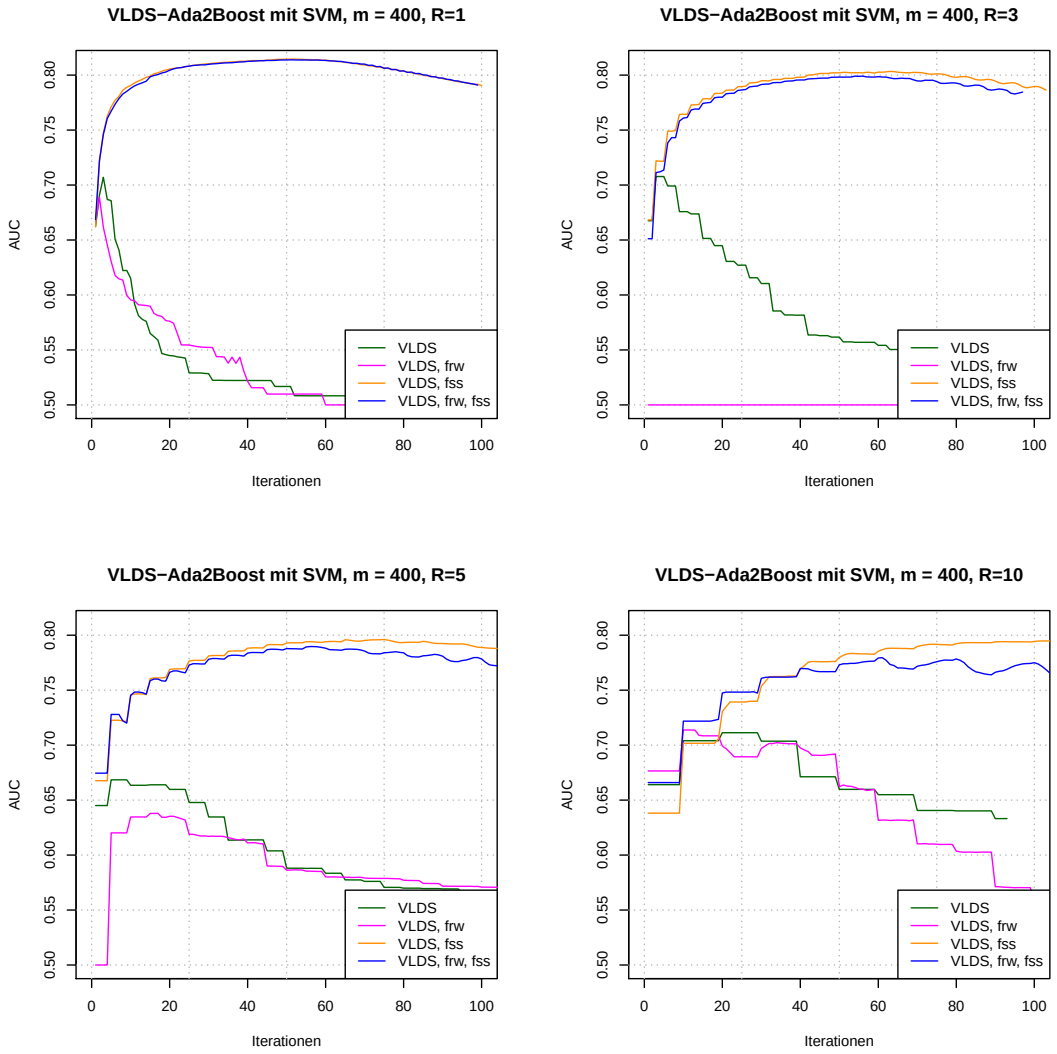
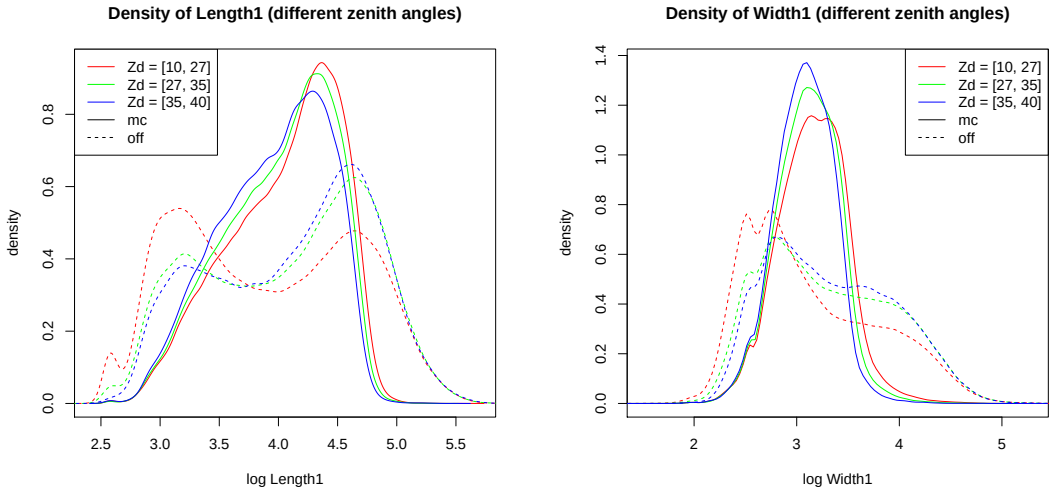


Abbildung 21: Performance von VLDS-Ada^{2}Boost mit einer Support Vector Machine mit radialem Kern, angewendet auf 400 Beispielen bei verschiedenen Resampling-Intervallen.



(a) (b)

Abbildung 22: Dichtefunktionen in verschiedenen Zenitwinkelbereichen. (a) Length; (b) Width. Die gestrichelten Linien zeigen die Verteilung der off-Daten, die durchgehenden die des Gamma-Samples.

14. Binning

14.1. Veränderliche Merkmalsverteilungen

Die zur Verfügung stehenden Daten umfassen einen Zenitwinkelbereich von 10° bis 40°. Der Zenitwinkel φ gibt den Abstand der Beobachtungsachse des Teleskops von der senkrechten, d. h. vom Zenit, an.

Das Teleskop befindet sich auf der Erdoberfläche und nutzt die Atmosphäre als Detektor. Ändert sich der Zenitwinkel, so ändert sich dadurch die Strecke, die das Licht eines Teilchenschauers in der Atmosphäre zurücklegen muss, bevor es in die Kamera fällt. Außerdem ändert sich unter Umständen der Winkel zwischen der Teleskopachse und den Achsen einfallender Schauer. Daher unterscheiden sich die Kamerabilder, die unter verschiedenen Zenitwinkeln aufgenommen werden. Daraus ergibt sich, dass sich auch die Verteilungen der von *star* berechneten Merkmale ändern.

Abbildung 22 zeigt die Verteilung der Merkmale *Length* und *Width* des Magic-1-Teleskops für unterschiedliche Zenitwinkelbereiche. Diese beiden Merkmale geben an, wie groß ein Schauer in der Teleskopkamera erscheint. Die Winkelintervalle sind so gewählt, dass sie in $(\cos^2(\varphi))^{-1}$ äquidistant sind. Zur Abschätzung der Dichtefunktion aus den empirischen Daten wird die Implementierung eines Kerndichteschätzers aus dem R-Project [30] verwendet, die auf alle zur Verfügung stehenden Daten angewendet wird.

Deutlich zu erkennen ist, dass die Verteilung des Gamma-Samples über verschiedene Zenitwinkelbereiche nahezu konstant ist. Sie zeigt lediglich eine geringe Verschiebung zu geringeren Ausdehnungen für größere Winkel. Die off-Daten verschieben sich dagegen deutlich zu größeren Ausdehnungen bei höheren Zenitwinkeln.

Physikalisch überrascht diese Beobachtung nicht. Die Gamma-Events werden stets

aus demselben Blickwinkel beobachtet, da das Teleskop auf eine Gamma-Quelle ausgerichtet ist. Auch bei Monte-Carlo-generierten Daten wird eine virtuelle Kamera auf eine virtuelle Quelle ausgerichtet. Da fast alle beobachteten Gamma-Ereignisse aus der anvisierten Quelle stammen, werden die Schauer stets von unten beobachtet. Bei hohen Zenitwinkeln ist der Abstand der Schauer zum Teleskop größer, so dass die Ereignisse in der Kamera kleiner dargestellt werden.

Hadronische Teilchen dringen jedoch isotrop aus allen Richtungen in die Atmosphäre ein und erzeugen Schauer. Somit steigt bei großen Zenitwinkeln die Wahrscheinlichkeit, solche hadronischen Schauer auch aus anderen Winkeln, beispielsweise von der Seite, zu beobachten. Von der Seite aus betrachtet haben die Ereignisse dann eine größere Ausdehnung, was die Verschiebung der Verteilungen erklärt.

Die Verteilungen verschieben sich also, wenn man die Beispiele nach einem Attribut, dem Zenitwinkel, sortiert und dann darüber iteriert. Diese Eigenschaft heißt *Concept Drift*[10] [35].

14.2. Ausnutzung der Concept Drift bei der Modellgenerierung

Da die Gamma-Verteilungen nahezu konstant sind, sich die Hadron-Verteilungen jedoch verschieben, liegt auch eine relative Verschiebung beider Verteilungen vor. Daher könnte es vorteilhaft sein, die Trainingsmenge in mehrere Intervalle zu partitionieren. Diese Intervalle werden im folgenden *Bins* genannt.

Im Allgemeinen kann das Binning nicht nur auf dem Zenitwinkelbereich erfolgen, sondern auf einem beliebigen Binning-Attribut b mit dem Wertebereich $B \subseteq \mathbb{R}$. Im allgemeinen ist b nicht Teil der Merkmalsmenge, die zum Training eines Modells genutzt wird, da es oft Meta-Informationen repräsentiert, die beispielsweise die Art der Datennahme kodieren und keine Eigenschaften der eigentlichen Beispiele sind. Wenn für unterschiedliche Werte von b unterschiedlich viele Daten vorliegen und sich die a-priori-Wahrscheinlichkeiten der Klassen ändern, würde ein Lernalgorithmus möglicherweise ein Modell erzeugen, dass die Daten (auch) anhand der Datennahme klassifiziert, statt die eigentlichen Eigenschaften der Beispiele zu nutzen.

Es wird nun anstelle eines globalen Modells auf dem gesamten B ein Meta-Modell erzeugt, welches für jedes Bin ein separates Modell trainiert. Der Trainingsalgorithmus erhält als Eingabe eine Trainingsmenge

$$\mathbb{E} = \{(x_1, y_1, b_1), \ldots, (x_m, y_m, b_m)\} \text{ mit } (x_i, y_i, b_i) \in X \times Y \times B \tag{94}$$

sowie eine Menge

$$L = \{l_1, \ldots, l_q\} \text{ mit } l_j \subseteq \mathbb{R}, \bigcap_{j=1}^{q} l_j = \emptyset \wedge \bigcup_{j=1}^{q} l_j = B \quad , \tag{95}$$

welche die Binning-Intervalle angibt und den Wertebereich des Binning-Attributs partitioniert. Die Trainingsmenge enthält für jedes Beispiel neben den üblichen Merkmalsausprägungen x und dem Label y auch den zugehörigen Wert b des Binning-Attributs.

[10] In der Regel wird Concept Drift als Verschiebung der Verteilung über die Zeit aufgefasst. Hier ist der Parameter „Zeit“ durch den Zenitwinkel substituiert.

```
1  Eingabe:
2    Beispielmenge 𝔼
3    Binning−Intervalle L
4  Ausgabe:
5    Modell h : X × B → ℝ
6
7  Partitioniere 𝔼 in {𝔼_1,...𝔼_q}, so dass ∀j ∈ {1,...q} : ∀(x,y,b) ∈ 𝔼_j : b ∈ l_j
8  for j in {1,...,q}
9    Trainiere Modell h_j : X → ℝ auf 𝔼_j
10 end
11 Erzeuge Meta−Modell h : X × B → ℝ, h(x,b) = Σ_{j=1}^{q} h_j(x,b) · I(b ∈ l_j)
12 return h
```

Algorithmus 6: Algorithmus zum Training eines Binning-Modells

Algorithmus 6 zeigt den Trainingsalgorithmus in Pseudocode. Zunächst partitioniert der Algorithmus die Eingabemenge in q disjunkte Teilmengen, die jeweils nur Beispiele mit Werten aus genau einem Binning-Intervall enthalten. Auf jeder Teilmenge wird dann ein Modell h_j trainiert. Dazu kann ein beliebiger Lernalgorithmus verwendet werden, der sich zur Lösung der Klassifikationsaufgabe eignet. Jedes gelernte Modell ist dann spezialisiert auf ein bestimmtes Binning-Intervall und kann, im Gegensatz zu einem globalen Modell, die Merkmalsverteilungen in diesem bestimmten Intervall optimal zum Lernen nutzen.

Alle gelernten Modelle werden dann zu einem Meta-Modell zusammengefasst, dem Binning-Modell. Dieses kann unbekannte Beispiele aus dem gesamten Wertebereich B des Binning-Attributs klassifizieren. Erhält es als Eingabe ein Beispiel (x, b), so wählt es das entsprechende spezialisierte Modell h_j aus, so dass das Binning-Attribut b in der entsprechende Partition l_j liegt. Die Ausgabe $h(x, b)$ des Meta-Modell ist die Ausgabe von $h_j(x)$.

Da die Trainingsmenge durch das Binning aufgeteilt und die spezialisierten Modelle auf kleineren Teilmengen trainiert werden, ergibt sich beim Training der spezialisierten Modelle mit derselben Argumentation wie in Kapitel 13.1 eine Reduzierung der Laufzeit.

14.3. Wahl der Binning-Intervalle

Für das Training der spezialisierten Modelle stehen weniger Beispiele zur Verfügung, als für das Training eines globales Modell, da die Beispielmenge der m Trainingsbeispiele in q Teilmengen partitioniert wird, so dass für die spezialisierten Beispiele im Mittel nur m/q Beispiele zur Verfügung stehen. In der Regel bedeuten weniger Trainingsbeispiele jedoch eine geringere Klassifikationsgüte des Modells. Auf der anderen Seite erlaubt eine größere Anzahl an Bins die Erzeugung spezialisierterer Modelle. Ein Binning mit kleinen Intervallen liefert also nicht automatisch das beste Binning-Modell (s. Abbildung 23).

Stattdessen müssen die Intervalle so gewählt werden, dass sie möglichst viele Beispiele enthalten, gleichzeitig die Concept Drift vom Beginn des Intervalls bis zum Ende des Intervalls jedoch möglichst gering ist, d. h. die Verteilung innerhalb des Intervalls

möglichst homogen ist.

Das Binning L enthält die Binning-Intervalle als Teilmengen von B. Statt Teilmengen zu wählen, können auch die Bin-Grenzen $Z' = \{z_0, \ldots, z_q\} \in \mathbb{R}^{q+1}$ dargestellt werden, die diese Intervalle voneinander trennen:

$$z_j = \begin{cases} b : \forall b' \in l_1 : b \leq b' & \text{falls } j = 0 \\ b : \forall b' \in l_q : b > b' & \text{falls } j = q \\ b : \forall b' \in l_j : b > b' \wedge \forall b' \in l_{j+1} : b \leq b' & \text{sonst} \end{cases} \tag{96}$$

Dabei wird vorausgesetzt, dass die Intervalle sortiert sind, d. h.

$$\forall b \in l_j, b' \in l_j + 1 : b < b' \quad . \tag{97}$$

Das Problem, ein optimales Binning auszuwählen, ist dann ähnlich zu einer Merkmalsselektion (s. Definition 7.1). Zur vereinfachten Darstellung werden zunächst die oberste und die unterste Bin-Grenze z_0 und z_q weggelassen, da sie stets die Unter- bzw. Obergrenze von B darstellen und daher implizit stets für das Binning verwendet werden. Es sei nun

$$Z = Z' \backslash \{z_0, z_q\} \quad , \tag{98}$$

dann ist das Optimierungsproblem wie folgt definiert:

Definition 14.1 (Optimales Binning)**.** *Ein* optimales Binning $Z_{opt} \subseteq Z$ *ist ein Binning, bei dem ein darauf trainiertes Binning-Modell h eine optimale Performance liefert:*

$$Z_{opt} := \arg\max_{Z'' \subseteq Z} (Performance(h, Z'')) \tag{99}$$

Als Performance kann ein beliebiges Gütemaß gewählt werden. Sollten beim gewählten Maß kleine Werte eine hohe Güte bedeuten, so muss die Performance in obiger Definition mit -1 multipliziert werden.

14.4. Implementierung für RapidMiner

Im Rahmen dieser Arbeit wurde ein Binning-Framework für RapidMiner implementiert. Dieses greift auf das Clustering-Framework von RapidMiner zurück und die Bins werden dabei als Cluster aufgefasst. Der Operator DISCRETIZE BY BINNING aus dem RapidMiner-Kern kann beispielsweise aus einem reellwertigen Attribut ein Clustering erzeugen. Neben einer Beispielmenge mit Clusterattribut gibt der Operator auch ein *Discretization Model* aus, das auf unbekannte Beispielmengen angewendet werden kann.

14.4.1. Training und Anwendung eines Binning-Modells

Der Operator CREATE COMBINED MODEL ON CLUSTERS, der im Rahmen dieser Arbeit entwickelt wurde, erhält als Eingabe eine Beispielmenge mit dem Spezialattribut *cluster*. Ein innerer Prozess trainiert ein spezialisiertes Modell auf jedem Cluster. Alle Modelle aus dem inneren Prozess werden in einem Meta-Modell zusammengefasst. Wird

das Meta-Modell durch den APPLY MODEL-Operator auf neue Beispiele angewendet, so wird anhand des Clusterattributs jedes Beispiels das entsprechende spezialisierte Modell zur Klassifikation ausgewählt.

Das Binning-Modell selbst kennt also abweichend von der Definition im vorherigen Kapitel die Bin-Grenzen nicht. Diese müsssen stattdessen in einem Vorverarbeitungsschritt auf die Beispielmenge angewendet werden. Es wird gezeigt, dass dadurch mit geringem Aufwand viel bestehende Funktionalität auf das neue Binning-Framework angewendet werden kann.

Mit den bislang beschriebenen Operatoren lässt sich ein Binning erzeugen und darauf ein Modell trainieren, mit dem sich unbekannte Daten klassifizieren lassen.

14.4.2. Auswahl der Binning-Intervalle als Merkmalsselektion

Wie in Kapitel 14.3 gezeigt wurde, kann das Optimierungsproblem der Intervallauswahl als Merkmalsselektion aufgefasst werden. Da RapidMiner über ein umfangreiches Instrumentarium zur Merkmalsselektion verfügt, werden zwei Operatoren implementiert, die die Intervallauswahl so transformieren, dass dieses Instrumentarium genutzt werden kann.

Wie zuvor beschrieben gibt der DISCRETIZE-Operator ein *Discretization Model* aus. Dieses enthält die Informationen, die benötigt werden, um das Binning auf neue Daten anzuwenden. Das sind insbesondere die Bin-Grenzen. Der Operator DISCRETIZATION MODEL 2 EXAMPLE SET erzeugt aus einem *Discretization Model* eine Beispielmenge mit genau einer Zeile und q Attributen. Jedes Attribut enthält den Wert einer Bin-Grenze. Der Operator EXAMPLE SET 2 DISCRETIZATION MODEL transformiert ein solches *ExampleSet* zurück in ein *Discretization Model*.

Die Intervallauswahl entspricht nun einer Merkmalsselektion auf der von DISCRETIZATION MODEL 2 EXAMPLE SET generierten Beispielmenge. Somit können alle Verfahren zur Merkmalsselektion verwendet werden, die in RapidMiner implementiert sind.

14.5. Experimentelle Evaluierung

Die Evaluierung der beschriebenen Binning-Methode erfolgt in zwei Schritten. Zunächst werden mit unterschiedlichen Vefahren und Parametern mehrere Binning-Intervalle berechnet. Auf den Intervallen werden dann Binning-Modelle mit unterschiedlichen Parametern trainiert.

Die Auswahl der Binning-Intervalle wird durch die zuvor beschriebenen Operatoren in eine Merkmalsselektion transformiert. Danach werden in unterschiedlichen Experimenten die Forward Feature Selection und Backward Feature Selection sowie eine evolutionäre Merkmalsselektion angewendet (vgl. Kapitel 7.1). Jedes dieser Verfahren wird auf Datensätze unterschiedlicher Größe und mit unterschiedlichen initialen Binning-Intervallen angewendet. Die Datensätze umfassen jeweils 4000, 10000, 15000, 30000 und 60000 Beispiele, in denen die a-priori-Wahrscheinlichkeiten von Gamma- und Hadron-Ereignissen jeweils 50 % beträgt. In einem Vorverarbeitsungsschritt wird ein lineares Binning in $(\cos^2 Zd)^{-1}$ mit 15, 30, 45, 60, 75 und 90 Intervallen durchgeführt. Jedes Verfahren zur Merkmalsauswahl wird auf jeden Datensatz angewendet.

Innerhalb der Merkmalsauswahl müssen die aktuellen Intervall-Grenzen bewertet

werden. Dazu wird in einer Kreuzvalidierung die AUC des J48-Klassifikators berechnet. Jedes Experiment gibt die besten gefundenen Intervall-Grenzen aus.

Aufgrund der hohen Laufzeit findet die Merkmalsselektion auf relativ kleinen Datensätzen mit einem nicht geboosteten Klassifikator statt. Es wurde aber gezeigt, dass sich die Klassifikationsgüte von J48 durch größere Trainingsmengen und Boosting deutlich verbessern lässt. Auf kleineren Trainingsmengen, wie den den spezialisierten Klassifikatoren im Binning-Modell, liefert häufig ein geboosteter Decision Stump bessere Ergebnisse als J48.

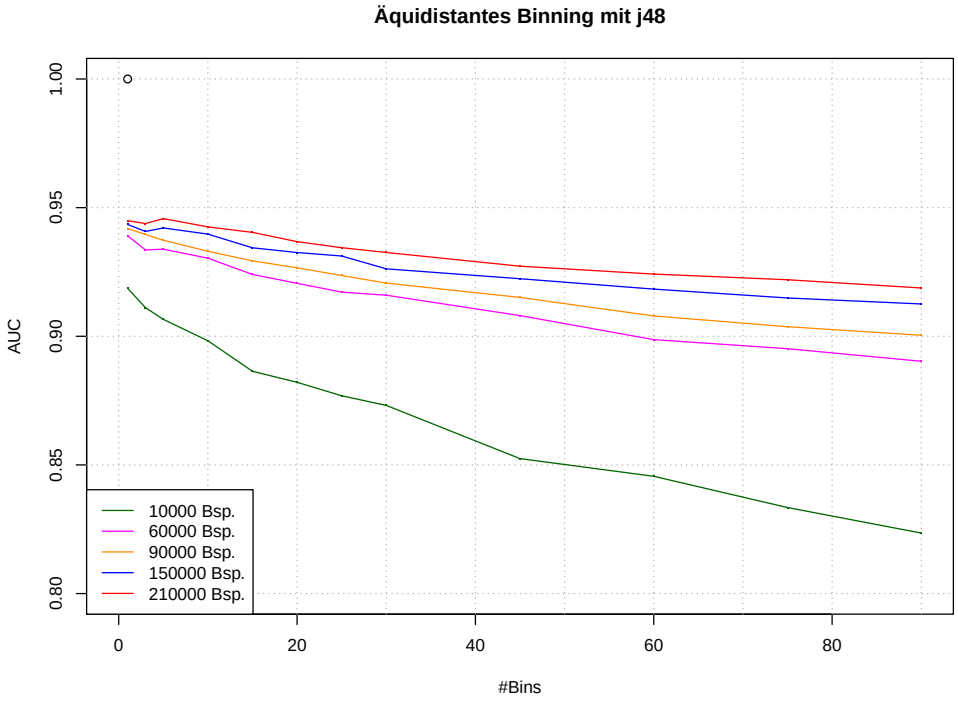


Abbildung 23: Äquidistantes Binning über $(cos^2(Zd))^{-1}$. Gezeigt ist die Area under the ROC-Curve eines Binning-Modells mit J48 gegen die Anzahl der Intervalle für unterschiedliche Größen der Trainingsmenge.
Zu erkennen ist, dass die Güte für kleinere Trainingsmengen stark abnimmt, da in den einzelnen Intervallen nicht mehr ausreichend viele Beispiele zur Verfügung stehen. Bei größeren Trainingsmengen wird ein Maximum bei fünf Bins erreicht. Steigt die Anzahl der Bins, nimmt die Güte ebenfalls ab, da die Anzahl der Beispiele in den Intervallen sinkt.

Daher wird zur abschließenden Bewertung der gefundenen Bin-Grenzen ein Binning-Modell mit geboostetem J48 bzw. geboostetem Decision Stump trainiert und in einer zehnfachen Kreuzvalidierung validiert. Beim Boosting kommt der zuvor beschriebene Algorithmus Fuzzy Ada^{2}Boost zum Einsatz. Zum Boosten des Decision Stumps werden 200 Iterationen verwendet, für J48 nur 25, da in Kapitel 12 gezeigt wurde, dass Ada^{2}Boost mit J48 schon nach wenigen Iterationen die Sättigung erreicht.

14.5.1. Ergebnisse

Abbildung 24 zeigt die AUC von Fuzzy Ada^{2}Boost auf den vier besten gefundenen Binnings gegen die Größe der Trainingsmenge. Tabelle 3 zeigt die zugehörigen Binning-Intervalle. Es fällt auf, dass drei der gezeigten vier besten Binnings eine Grenze bei $1,17$ und alle vier eine Grenze um $1,26$ setzen. Dass das beste Binning unterschiedlicher Verfahren mit unterschiedlichen Parametern jeweils diese Grenzen wählt, deutet darauf hin, dass einerseits an diesen Punkten die Concept Drift hoch zu sein scheint und andererseits alle Verfahren geeignet sind, dies zu entdecken und eine entsprechende Intervallgrenze zu setzen.

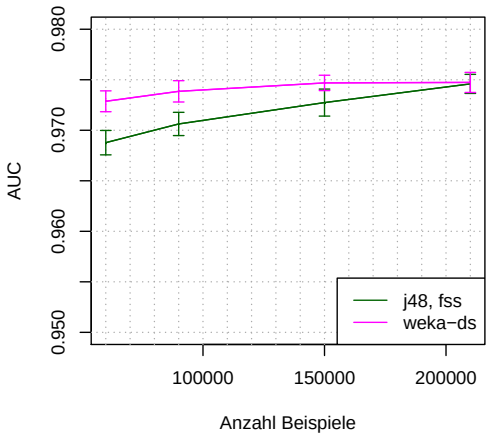
Da Ada^{2}Boost mit Decision Stump ohne konfidenzbasierte Neugewichtung in den Experimenten zum Binning-Modell stets eine höhere AUC liefert als die Varianten mit konfidenzbasierter Gewichtung, wird nur diese Kurve gezeigt. Für J48 liefert die Variante mit der konfidenzbasierten Konfusionsmatrix die besten Ergebnisse. Ob zusätzlich die konfidenzbasierte Gewichtung eingesetzt wird oder nicht, macht keinen signifikanten Unterschied, so dass hier nur die Variante ohne konfidenzbasierte Neugewichtung gezeigt wird.

Wie zu erwarten, verbessert sich die Güte, je größer die Trainingsmenge ist. Dieser Effekt fällt für den geboosteten Decision Stump schwächer als aus für J48. Dies entspricht den Beobachtungen in Kapitel 12. Außerdem ist der Decision Stump stets besser als J48. Lediglich bei sehr großen Beispielmengen ist J48 mit dem Decision Stump vergleichbar. Berücksichtigt man, dass in den einzelnen Bins relativ wenig Beispiele zum Training zur Verfügung stehen, deckt sich auch dies mit den Ergebnissen aus Kapitel 12, wo bereits gezeigt wurde, dass der geboostete Decision Stump bei kleinen bis mittleren Trainingsmengen besser als der geboostete J48 ist. Die Spezialisten im Binning-Modell sind beim Decision Stump also besser und somit auch das gesamte Binning-Modell. Die Güte der Modelle für die größte benutzte Trainingsmenge mit 210.000 Beispielen ist ebenfalls Tabelle 3 zu entnehmen.

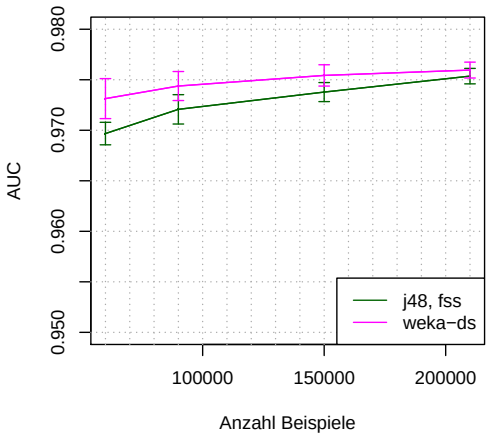
Im Vergleich mit Fuzzy Ada^{2}Boost ohne Binning zeigt sich, dass das Binning bei allen getesteten Trainingsmengen eine Vergrößerung der AUC liefert.

Verfahren	#Beispiele	#Bins	Bin-Grenzen	AUC DS	AUC J48
forward	2000	30	1.14, **1.28**, 1.39, 1.70	0.972 ± 0.0007	0.975 ± 0.0009
backward	5000	15	**1.17**, 1.21, **1.26**, 1.48, 1.52, 1.70	0.973 ± 0.0010	0.975 ± 0.0008
evolutionär	2000	15	**1.17**, **1.26**, 1.61, 1.70	0.973 ± 0.0009	0.975 ± 0.0006
evolutionär	15000	30	**1.17**, 1.21, 1.23, **1.26**, 1.46, 1.50	0.973 ± 0.0008	0.976 ± 0.0006

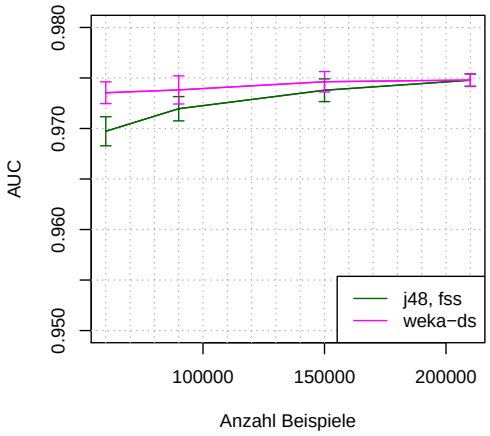
Tabelle 3: Güte und Binning-Intervalle nach Anwendung unterschiedlicher Selektionsverfahren. Die erste Spalte gibt das Verfahren an, das zur Intervallgrenzenselektion verwendet wurde, die nächsten beiden die Anzahl der während der Selektion verwendeten Beispiele und die Anzahl der Bins zu Beginn der Selektion. Die Spalte *Bin-Grenzen* listet die gefundenen Grenzen in $(\cos^2(Zd))^{-1}$ auf. Die beiden letzten Spalten zeigen die AUC bei Anwendung von (Fuzzy) Ada^{2}Boost mit einem Decision Stump (DS) und J48.



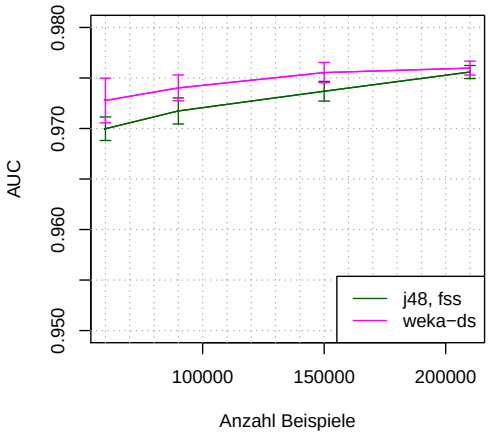
(a) Forward Feature Selection, initial 30 Bins, 2000 Beispiele



(b) Backward Feature Selection, initial 15 Bins, 5000 Beispiele



(c) evolutionäre Feature Selection, initial 15 Bins, 2000 Beispiele



(d) evolutionäre Feature Selection, initial 30 Bins, 15000 Beispiele

Abbildung 24: Performance verschiedener Binning-Modelle. Die Bin-Grenzen wurden mit unterschiedlichen Methoden berechnet. Die Fehlerbalken entsprechen einer Standardabweichung.

15. Vergleich zum Random Forest

In diesem Kapitel werden die im Rahmen dieser Arbeit entwickelten Verfahren mit dem Random Forest verglichen, wie er in der klassischen MAGIC-Analyse durch das MARS-Toolkit zum Einsatz kommt.

Zum Vergleich der Ada^{2}Boost-Varianten mit einem Random Forest wird ein Random Forest mit unterschiedlich vielen Bäumen auf verschieden großen Trainingsmengen erzeugt. Abbildung 25 zeigt die AUC für die erzeugten Modelle. Die Ergebnisse sind zehnfach kreuzvalidiert.

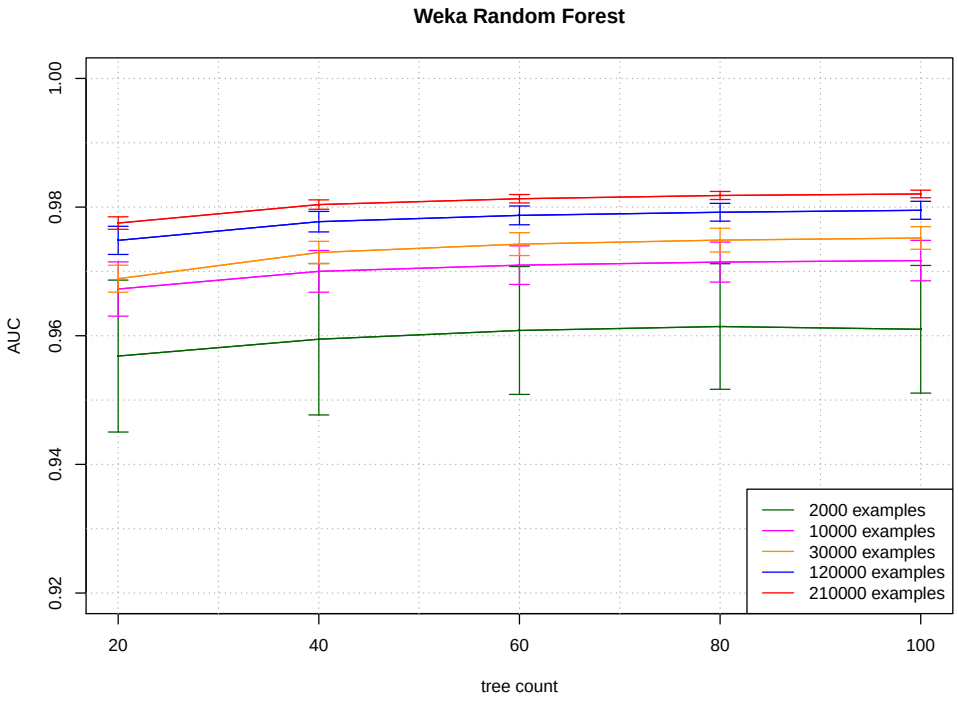


Abbildung 25: AUC für verschiedene Random Forests. Auf der x-Achse ist die Anzahl der Bäume aufgetragen, auf der y-Achse die AUC. Die Fehlerbalken entsprechen einer Standardabweichung.

Es zeigt sich, dass der Random Forest von einer Vergrößerung der Trainingsmenge profitiert. Auch die Erhöhung der Anzahl der Bäume bringt eine gewisse Performance-Steigerung. Ab etwa 60 bis 80 Bäumen wird jedoch eine Sättigung erreicht, und eine weitere Erhöhung der Baumzahl bringt keine Verbesserung mit sich.

Die AUC bei einer Trainingsmenge mit $m = 210.000$ Beispielen liegt bei etwa 0.982 ± 0.0001 und damit geringfügig höher als der beste mit Ada^{2}Boost-Varianten erzeugte Wert. Dieser wurde durch das Binning erzielt und liegt bei etwa 0.976 ± 0.0006. Bezüglich der AUC scheint ein Random Forest also geringfügig besser für den untersuchten Datensatz zu sein. Allerdings ist es möglich, dass die Ada^{2}Boost-Varianten trotz der etwas geringeren AUC auf echten Daten eine bessere Performance erreichen, da die AUC lediglich die Wahrscheinlichkeit angibt, dass an einem zufälligen Operation Point zwei zufällig gezogene Beispiele korrekt gerankt werden. Es ist jedoch durchaus möglich,

dass ein Klassifikator mit geringerer AUC an einigen Operation Points eine bessere Klassifikationsgüte erzielt. Da das wahre Klassenverhältnis auf echten Teleskopdaten einerseits je nach beobachteter Quelle variiert und andererseits niemals genau bekannt ist, kann über die Güte von Random Forest und den Ada^{2}Boost-Varianten wegen des nur geringen Unterschiedes in den AUC-Werten keine Aussage getroffen werden.

16. Fazit

Mit Fuzzy Ada^{2}Boost, VLDS-Ada^{2}Boost und dem Meta-Verfahren des Binnings wurden drei Methoden vorgestellt, die das binäre Klassifikationsproblem der Gamma-Hadron-Separation auf unterschiedliche, sich ergänzende Weise angehen. Allen drei ist gemein, dass sie Daten samplen. Zum einen kommt dabei Sampling durch Umgewichtung von Beispielen zum Einsatz, zum anderen aber auch Sampling durch Ziehen von Stichproben aus der Gesamtheit der Daten. Als Erweiterung zu Ada^{2}Boost trainiert Fuzzy Ada^{2}Boost iterativ Basismodelle, und gewichtet dabei in jeder Iteration die Daten um. Bei diesem Sampling wird das Wissen über die Verteilung genutzt, das durch die Anwendung der zuvor trainierten Basismodelle gewonnen wurde. Ada^{2}Boost wurde dabei insofern verfeinert, als in Fuzzy Ada^{2}Boost auch reellwertige Basisklassifikatoren genutzt werden können, so dass bei der Umgewichtung detaillierteres Wissen über die Verteilungen genutzt werden kann.

Ada^{2}Boost for Very Large Datasets, VLDS-Ada^{2}Boost, erneuert zusätzlich in regelmäßigen Abständen die Arbeitsmenge durch Ziehen neuer Beispiele. Dadurch ermöglicht VLDS-Ada^{2}Boost die Analyse sehr großer Datenmengen durch die iterative Verarbeitung kleiner Arbeitsmengen: auch wenn die Anzahl der Beispiele, die zum Trainieren des Basismodelle verwendet wird, klein ist, vergrößert sich durch das Ziehen neuer Beispiele die Stichprobe, die insgesamt zum Lernen genutzt wird, so dass sich die empirische Verteilung der Trainingsmenge immer mehr an die wahre Verteilung der Daten annähert. Das unmodifizierte Ada^{2}Boost und auch Fuzzy Ada^{2}Boost müssen zum Erreichen derselben Klassifikationsgüte wesentlich mehr Beispiele im Speicher halten und zum Training verwenden als VLDS-Ada^{2}Boost. Durch die Aufteilung der Daten in mehrere Arbeitsmengen, die iterativ verarbeitet werden, reduziert VLDS-Ada^{2}Boost die Speicheranforderungen und auch die Laufzeit zur Erzeugung des Boosting-Ensembles, da pro Iteration weniger Beispiele betrachtet müssen. Bei Verwendung von J48 als Basisklassifikator kann die Laufzeit bei ähnlicher Klassifikationsgüte um den Faktor 20–40 reduziert werden.

Durch den Resampling-Schock sind der Verkleinerung der Arbeitsmenge jedoch Grenzen gesetzt. Dieser verhindert, dass die Klassifikationsgüte des Ensembles durch Nachziehen von Beispielen beliebig gesteigert werden kann. Nachfolgende Arbeiten sollten Methoden untersuchen, die den Resampling-Schock reduzieren. Da er durch Overfitting hervorgerufen wird, sollten diese Methoden an der Verringerung des Overfittings ansetzen.

Trotz der Auswirkungen des Resampling-Schocks gelingt es mit VLDS-Ada^{2}Boost, große Trainingsmengen effektiv zu nutzen, indem eine große Trainingsmenge in mehrere kleine Arbeitsmengen aufgeteilt wird. Auch das Binning-Verfahren teilt die Trainingsmenge in kleinere Untermengen auf. Dabei steht jedoch nicht nur die Beschleunigung des Lernprozesses im Vordergrund, sondern vor allem die Homogenisierung

nicht-stationärer Verteilungen: die Verteilung der MAGIC-Daten verschiebt sich bei Änderung des Zenitwinkels. Indem Zenitwinkelbereiche gesucht werden, in denen die Verteilung möglichst homogen ist, können dann spezialisierte Basismodelle für ebendiese Verteilungen trainiert werden. Diese sind in ihrem Bereich meist genauer als Klassifikatoren, die über den gesamten Zenitwinkelbereich trainiert werden, obwohl die Anzahl der Beispiele im Training durch das Aufteilen der Menge geringer ist.

Das Problem der Binning-Optimierung wurde in eine Merkmalsselektion transformiert. Dadurch können bestehende Verfahren direkt und ohne Modifikationen auf dieses neue Problem angewendet werden.

Die Evaulierung der Verfahren auf realistischen Datenmengen wird durch die Benutzung eines Rechenclusters vereinfacht und beschleunigt. Mit dem PhiDo-Adapter wurde eine Infrastruktur geschaffen, mit der auch Parametervariationen, wie sie bei der Bewertung neuer Verfahren oft zum Einsatz kommen, hochparallel ausgeführt werden können.

Insgesamt wurde also ein Instrumentarium geschaffen, das unter Verwendung von Sampling-Methoden die speziellen Anforderungen der MAGIC-Daten erfüllt: das auf Fuzzy Ada^{2}Boost basierende VLDS-Ada^{2}Boost löst das Problem der großen Datenmengen. Die Binning-Methode geht das Problem der nicht-stationären Verteilungen an. In Kombination mit Fuzzy Ada^{2}Boost werden dabei gute Ergebnisse erzielt. Die Klassifikationsgüte sowohl der Binning-Methode als auch von VLDS-Ada^{2}Boost liegt im Bereich der Güte des Random Forest, der in der MAGIC-Analyse etabliert ist. Die Güte des Ausgangsverfahrens Ada^{2}Boost wird deutlich übertroffen. Eine genauere Analyse des Resampling-Schocks könnte weitere Verbesserungen erbringen.

Anhang

A. RapidMiner-PhiDo-Adapter

Das Potenzial eines Rechenclusters wie des PhiDo, das sich durch eine große Menge von Knoten auszeichnet, die jeweils nicht überdurchschnittlich schnell sind, wird am besten dann genutzt, wenn die durchgeführten Berechnungen möglichst stark in unabhängige, parallel ausführbare Prozesse aufgeteilt werden können. Ein häufiger Anwendungsfall in RapidMiner ist eine sogenannte *Parametervariation*. Dabei wird ein Prozess für mehrere Kombinationen von Operator-Parametern ausgeführt. Dadurch können Performance-Kurven in Abhängigkeit der Parameter erzeugt oder einfach die beste Parameterkombination gefunden werden. Diese Funktionalität ist durch entsprechende Operatoren in RapidMiner integriert. Diese integrierte Variante ist jedoch nicht direkt für die effiziente Ausführung auf dem Cluster geeignet, da alle Parameterkombinationen seriell oder schwach parallelisiert innerhalb eines Prozesses ausgeführt werden.

Daher wurde im Rahmen dieser Arbeit ein Framework implementiert, welches ebenfalls Parameter variieren kann und für jede mögliche Kombination eine RapidMiner-Prozessdatei erzeugt. Außerdem können innerhalb einer Variation verschiedene Operatoren iteriert werden, die jeweils unterschiedliche Parameter oder auch Parameterkombinationen enthalten können. Da für jede Parameterkombination ein eigener Prozess erzeugt wird, können sie parallel im Cluster berechnet werden. Das Framework erzeugt dazu entsprechende Jobfiles sowie Skriptdateien, welche die Jobs in die Warteschlangen des Clusters einreihen.

Durch Kommunikation mit dem Cluster kann jederzeit ein Überblick darüber erzeugt werden, welchen Status die Prozesse und Parameterkombinationen haben, d. h. ob sie bereits eingereiht sind, gerade in Ausführung befindlich oder erfolgreich oder fehlerhaft beendet wurden.

Die Ergebnisse der Berechnungen werden in menschen- und maschinenlesbarer Form in das Dateisystem des Clusters geschrieben. Da das Ergebnis jeder Parameterkombination von einem eigenen Prozess erzeugt wurde, liegen die Ergebnisse jedoch nicht zentral in einer Datei, sondern sind nach einem bestimmten Schema an mehreren Orten des Dateisystems verteilt.

Das Framework sammelt diese Ergebnisse und fasst sie in einem zentralen Objekt zusammen. Dieses *ResultsObject* kann nach bestimmten Parameterwerten gefiltert und sortiert werden. Außerdem können die Ergebnisse graphisch als Diagramme ausgegeben werden. Durch eine Gruppierungsfunktion kann der Benutzer bestimmen, welche Graphen in einem Diagramm zusammengefasst bzw. in unterschiedliche Diagramme gezeichnet werden.

A.1. Erzeugung von Prozessen und Jobs

Algorithmus 7 zeigt beispielhaft die Erzeugung einer Parameter- und Modellvariation über verschiedene Boosting-Operatoren mit unterschiedlichen inneren Lernern.

Die Funktion `job_creator_ada2boost` (Zeile 1–30) erzeugt eine Instanz von `JobFileCreator` mit einer Variation über verschiedene Basisklassifikatoren in einem

Ada²Boost-Operator. Diese Klasse dient dazu, PhiDo-Jobfiles zu erzeugen und erhält als Eingabe eine Instanz der Klasse `Variation`, welche eine Parametervariation beschreibt.

In Zeile 2–5 wird zunächst ein `RMProcess`-Objekt erzeugt. Dieses Objekt repräsentiert einen abgespeicherten RapidMiner-Prozess, der als Grundlage für die Variation verwendet wird. In Zeile 8 wird ein `Variation`-Objekt auf diesem Prozess erzeugt, welches die eigentlich Variation repräsentiert. In den Zeilen 11–18 wird eine Instanz `ParameterValues` erzeugt, die angibt, welche Prozesse zum Training des inneren Lerners verwendet werden.

Im Anschluss werden die Eigenschaften der zu erzeugenden Jobs konfiguriert und anschließend ein `JobFileCreator` mit der erzeugten Variation und den Parametern zurückgegeben.

Die Funktion `job_creator_bbcxval` erzeugt ebenfalls ein `JobFileCreator`-Objekt, welches den Prozess repräsentiert, der letztendlich in das Cluster geschickt wird. Der Basisprozess enthält eine Kreuzvalidierung, in der verschiedene Boosting-Modelle trainiert werden sollen, sowie einen Operator zum Einlesen einer Beispielmenge.

Die Zeilen 35–37 zeigen exemplarisch, wie unterschiedliche Trainingsmengen eingelesen werden können. Die Auslassungen müssen durch entsprechende Prozesse zum Einlesen der Mengen ersetzt werden.

Ab Zeile 39 wird dargestellt, wie verschiedene `JobFileCreator`s geschachtelt werden können: hier wird die zuvor definierte Funktion `job_creator_ada2boost` sowie eine weitere, analog implementierte Funktion `job_file_creator_adaboost` aufgerufen, die jedoch Prozesse für ein anderes Boosting-Verfahren, nämlich AdaBoost, erzeugt. Die Ausgaben dieser Funktionen wird als Parameter des TRAIN MODEL-Operators genutzt. Der am Ende erzeugte `JobFileCreator` kann dann Jobs für alle Kombinationen der eingestellten Beispielmengen-Leseoperatoren sowie die beiden Boosting-Verfahren mit ihren unterschiedlichen Basislernern erzeugen und ins Cluster schicken.

B. Liste der Fehlerbehebungen für RapidMiner 5.1

Die Version RapidMiner 5.1.001 (Entwicklungsversion vom 29.12.2010), die in dieser Arbeit zur Berechnung der Ergebnisse verwendet wird, weist einige Fehler auf. Um dennoch zuverlässig korrekte Ergebnisse zu erhalten, wurden entsprechende Bugfixes implementiert.

- *Boosting-Modelle akzeptieren keine Parameter.* Die Operatoren ADABOOST, BAYESIAN BOOSTING und die Stream-Variante des letzteren Operators erzeugen Ensemble-Modelle, wobei jedes Mitglied bzw. jeder *Base Classifier* des Ensembles aus einer Iteration resultiert. Im Normalfall werden bei Anwendung des Modells durch den Operator APPLY MODEL alle Mitglieder des Ensembles zur Klassifikation verwendet. Zur Erzeugung von Diagrammen, die die Güte des Modells gegen die Anzahl der Iterationen auftragen, ist es jedoch hilfreich, wenn von einem Ensemble nur die ersten k Basismodelle verwendet werden. Die Modelle ignorieren diesen Parameter jedoch. Dieser Fehler wurde behoben.

```
1  def job_creator_ada2boost conf, namespace
2    base_process =
         "#{conf[:local_base_repository]}/prozesse/model_creation/bbc.rmp"
3
4    # create process:
5    process = RMProcess.new base_process
6
7    # create variation:
8    variation = Variation.new process, conf[:repository],
         conf[:file_separator]
9    variation.namespace = namespace
10
11   # create variation values:
12   vo = variation.variation_objects
13   inner_learners = [
14     "//repo/prozesse/model_creation/j48",
15     "//repo/prozesse/model_creation/weka-ds",
16   ]
17   v = ParameterValues.new "Inner Learner", "process_location",
         inner_learners
18   vo.push v
19
20   # init job:
21   pbs_job = PBSJob.new
22   pbs_job.rm_executable = conf[:rapidminer_executable]
23   pbs_job.log_directory = conf[:remote_log_directory]
24   pbs_job.mail_address = conf[:mail_address]
25   pbs_job.walltime = "05:50:00"
26   pbs_job.memory = "4096"
27   pbs_job.cores = 1
28
29   return JobFileCreator.new [...], variation, pbs_job
30 end
31
32 def job_creator_bbcxval conf, namespace
33   base_process =
         "#{conf[:local_base_repository]}/prozesse/bbc_xval.rmp"
34
35   [...]
36   example_readers = [...]
37   v = ParameterValues.new "Read Examples", "process_location",
         example_readers
38
39   adaboost_creator = job_creator_adaboost conf, namespace
40   ada2boost_creator = job_creator_ada2boost conf, namespace
41   learners = Array.new
42   learners += adaboost_creator.variation.filenames :repository_name,
         false
43   learners += ada2boost_creator.variation.filenames
         :repository_name, false
44   v = ParameterValues.new "Train Model", "process_location", learners
45   [...]
46   return JobFileCreator.new [...], variation, pbs_job
47 end
```

Algorithmus 7: Beispiel-Code zum Erzeugen einer Variation über verschiedene innere Lerner

- READ DATABASE-*Operator schließt Verbindung zur Datenbank nicht.* Der Operator READ DATABASE stellt eine Verbindung zu einem SQL-Server her, führt eine beliebige Anfrage aus und liefert das Ergebnis als ExampleSet an RapidMiner. Die Verbindung zur Datenbank wird nur zur Ausführung der Anfrage benötigt, danach sollte sie wieder geschlossen werden. Das ist jedoch nicht der Fall. Im Betrieb einer RapidMiner-Instanz auf einem Rechner fällt der Fehler nicht auf, da die Verbindung nach einer gewissen Zeit automatisch vom SQL-Server beendet wird. Greifen jedoch viele RapidMiner-Instanzen gleichzeitig auf einen Server zu, die unter Umständen auch den READ DATABASE-Operator mehrmals ausführen, erreicht der Datenbankserver häufig die maximale Anzahl zulässiger Verbindungen und unterbindet weitere Anfragen. Auch dieser Fehler wurde behoben; der Operator schließt nun ordnungsgemäß die Verbindung zum Server.
- *Mittelwert von AUC kann nicht gebildet werden.* Der Operator AVERAGE erhält als Eingabe mehrere gleichartige Objekte vom Typ Averageable wie zum Beispiel die Ausgabe eines PERFORMANCE-Operators, d. h. *PerformanceVector*s. Enthält der PerformanceVector den Wert AUC (Area under the Curve), so bricht der Operator AVERAGE mit einer Fehlermeldung ab. Dieser Fehler beruht auf fehlenden Clone-Konstruktoren in den Unterklassen von `AreaUnderCurve` und wurde behoben.

C. Tabellenverzeichnis

D. Abbildungsverzeichnis

E. Verzeichnis der Algorithmen

F. Literatur

[1] ALBERT, J. ; ALIU, E. ; ANDERHUB, H. u. a.: Implementation of the Random Forest method for the Imaging Atmospheric Cherenkov Telescope MAGIC. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 588 (2008), Nr. 3, S. 424–432. – ISSN 0168–9002

[2] ALIU, E. ; ANDERHUB, H. ; ANTONELLI u. a.: Improving the performance of the single-dish Cherenkov telescope MAGIC through the use of signal timing. In: *Astroparticle Physics* 30 (2009), Januar, S. 293–305

[3] BACKES, M.: *Langzeitbeobachtung von TeV-Blazaren - Qullenanalyse mit MAGIC und Konzeption eines dedizierten Teleskops*, TU Dortmund, Diplomarbeit, 2008

[4] BRADLEY, P. A. The use of the area under the ROC curve in the evaluation of machine learning algorithms. In: *Pattern Recognition* 30 (1997), S. 1145–1159

[5] BREIMAN, L.: Random forests. In: *Machine learning* 45 (2001), Nr. 1, S. 5–32

[6] BRETZ, T.: *Observations of the Active Galactic Nucleus 1ES 1218+304 with the MAGIC-telescope*, Julius-Maximilians-Universität Würzburg, Diss., 2006

[7] CHANG, Chih-Chung ; LIN, Chih-Jen: *LIBSVM: a library for support vector machines*, 2001. – Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[8] DREYER, J.: *Hard- und Softwareentwicklung im Rahmen der Experimente AMANDA und IceCube*, Techscnische Universität Dortmund, Diplomarbeit, 2005

[9] EIBEN, A.E. ; SMITH, J.E.: *Introduction to Evolutionary Computing*. 2. Springer, 2007 (Natural Computing Series). – ISBN 978–3–540–40184–1

[10] FAWCETT, T.: ROC graphs: Notes and practical considerations for researchers. In: *Machine Learning* 31 (2004), S. 1–38

[11] FLACH, P. A.: The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics. In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*. Washington DC, 2003

[12] FREUND, Y. ; SCHAPIRE, R.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Journal of Computer and System Sciences* (1997)

[13] FREUND, Y. ; SCHAPIRE, R. ; ABE, N.: A short introduction to boosting. In: *Journal of Japanese Society for Artificial Intelligence* 14 (1999), S. 771–780. – ISSN 0912–8085

[14] GUYON, I. ; ELISSEEFF, A.: An introduction to variable and feature selection. In: *The Journal of Machine Learning Research* 3 (2003), S. 1157–1182. – ISSN 1532–4435

[15] HADASCH, D.: *Study of the MAGIC performance at high zenith angles and application of the results on a very high energy gamma ray flare of the blazar PKS 2155-304*, TU Dortmund, Diplomarbeit, 2008

[16] HALL, M. ; FRANK, E. u. a.: The WEKA Data Mining Software: An Update. In: *SIGKDD Explorations* Bd. 11, 2009

[17] HANLEY, J. A. ; MCNEIL, B. J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. In: *Radiology* 143 (1982), S. 29–36

[18] HASTIE, T. ; TIBSHIRANI, R. ; FRIEDMAN, J.: *The Elements of Statistical Learning*. Springer, 2009

[19] HECK, D.: Das Luftschauer-Simulationsprogramm CORSIKA und hadronische Wechselwirkungsmodelle. In: *FZKA-Nachrichten 33* (2001)

[20] HECK, D. ; PIERONG, T.: *Extensive Air Shower Simulation with CORSIKA: A User's Guide, Version 6.9xx*. Karlsruhe: Forschungszentrum Karlsruhe GmbH, 2009

[21] HELF, M. ; MORIK, K. ; RHODE, W.: Genetische Merkmalsselektion für die Gamma-Hadron-Separation im MAGIC-Experiment / Technische Universität Dortmund, Fakultät für Informatik. 2009 (828). – Forschungsbericht

[22] HILLAS, A. M.: Cerenkov Light Images of EAS Produced by Primary Gamma Rays and by Nuclei. In: *19th International Cosmic Ray Conference ICRC*, Jones, F.C., 1985

[23] JOGLER, T.: *Detailed study of the binary system LS I +61°303 in VHE gamma-rays with the MAGIC telescope*, Technische Universität München, Diss., 2009

[24] KRUSE, R. ; GEBHARDT, J. ; KLAWONN, F.: *Fuzzy-Systeme*. 2. Auflage. Stuttgart, Deutschland : Teubner, 1995

[25] MIERSWA, I. ; WURST, M. ; KLINKENBERG, R. ; SCHOLZ, M. ; EULER, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: UNGAR, L. (Hrsg.) ; CRAVEN, M. (Hrsg.) ; GUNOPULOS, D. (Hrsg.) ; ELIASSI-RAD, T. (Hrsg.): *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, August 2006. – ISBN 1–59593–339–5, 935–940

[26] MORALEJO u. a.: MARS, the MAGIC Analysis and Reconstruction Software. In: *31st International Cosmic Ray Conference*. Łodz, 2009

[27] OZA, N.C.: Online bagging and boosting. In: *Systems, man and cybernetics, 2005 IEEE international conference on* Bd. 3 IEEE, 2006, S. 2340–2345

[28] PLATT, J.: Sequential minimal optimization: A fast algorithm for training support vector machines / Microsoft Research. 1998 (MSR-TR-98-14). – Forschungsbericht

[29] QUINLAN, J.R.: *C4.5: programs for machine learning*. Morgan Kaufmann, 1993. – ISBN 1558602380

[30] R DEVELOPMENT CORE TEAM: *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2011. `http://www.R-project.org`. – ISBN 3-900051-07-0

[31] RITTHOFF, O. ; KLINKENBERG, R. ; FISCHER, S. ; MIERSWA, I.: A Hybrid Approach to Feature Selection and Generation Using an Evolutionary Algorithm / Collaborative Research Center 531, University of Dortmund. Dortmund, Germany, 2002 (CI-127/02). – Forschungsbericht. – ISSN ISSN 1433–3325

[32] SCHAPIRE, R.E. ; SINGER, Y.: Improved boosting algorithms using confidence-rated predictions. In: *Machine learning* 37 (1999), Nr. 3, S. 297–336. – ISSN 0885–6125

[33] SCHEFFER, Tobias ; WROBEL, Stefan: Finding the most interesting patterns in a database quickly by using sequential sampling. In: *J. Mach. Learn. Res.* 3 (2002), S. 833–862. – ISSN 1532–4435

[34] SCHMIDT, F.: *CORSIKA Shower Images*. `http://www.ast.leeds.ac.uk/~fs/showerimages.html`. Version: 2005

[35] SCHOLZ, M.: *Scalable and Accurate Knowledge Discovery in Real-World Databases*. Dortmund, Technische Universität Dortmund, Diss., 2007

[36] THE ROOT TEAM: *ROOT User's Guide*. Geneva: CERN, 2009

[37] TSANG, I.W. ; KWOK, J.T. ; CHEUNG, P.M.: Core Vector Machines: Fast SVM Training on Very Large Data Sets. In: *Journal of Machine Learning Research* 6 (2005), S. 363–392

[38] VAPNIK, V.: *The Nature of Statistical Learning Theory*. New York : Springer, 1996

[39] WAGNER, R. M.: *Measurement of Very High Energy Gamma-Ray Emission from Four Blazars Using the MAGIC Telescope and a Comparative Blazar Study*, Technische Universität München, Diss., 2006

[40] WAGNER, W.: *Design and Realisation of a new AMANDA Data Acquisition System with Transient Waveform Recorders*, Technische Universität Dortmund, Diss., 2004

Erklärung

Hiermit erkläre ich, dass ich die Diplomarbeit mit dem Titel:

Gamma-Hadron-Separation
im MAGIC-Experiment
durch verteilungsgestütztes Sampling

selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Dortmund, den 27. April 2011

_____

Marius Helf

Danksagung

- Zuallererst möchte ich mich bei Prof. Katharina Morik vom Lehrstuhl für Künstliche Intelligenz und Prof. Wolfgang Rhode vom Lehrstuhl Experimentelle Physik E5b dafür bedanken, dass sie durch ihre fächerübergreifende Zusammenarbeit diese Diplomarbeit möglich gemacht haben.
- Weiterer Dank gilt Benjamin Schowe, meinem Betreuer für Fragen der Informatik, der nicht nur für fachlichen Rat stets zur Verfügung stand, sondern auch stets motivierende Worte bereit hielt.
- Ebenso danke ich Michael Backes, meinem Betreuer in der Physik, für viele interessante und anregende Diskussionen nicht nur über das Thema Physik.
- Der gesamten Arbeitsgruppe E5b und insbesondere Natalie Milke, Nikola Strah und Marlene Doert danke ich für die angenehme Arbeitsatmosphäre und die kurzweiligen Mittagspausen.
- Allen Mitarbeitern des Lehrstuhls für künstliche Intelligenz gilt mein Dank für die vielen Ratschläge und Anregungen zu RapidMiner und anderen Belangen des maschinellen Lernens.
- Zu guter Letzt danke ich meinen Freunden und meiner Familie für Unterstützung und Motivation während des gesamten Studiums und darüber hinaus.

Danke!