Online-Lernen von zufälligen Entscheidungsbäumen

Bachelor-Thesis von Moritz Kulessa aus Frankfurt am Main
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Online-Lernen von zufälligen Entscheidungsbäumen

Vorgelegte Bachelor-Thesis von Moritz Kulessa aus Frankfurt am Main

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 26. Mai 2015

(M. Kulessa)

Inhaltsverzeichnis

1 Einleitung

Im Bereich des maschinellen Lernens werden Algorithmen entwickelt, die die Fähigkeit haben, anhand von Daten zu lernen. Die Aufgabe solcher Algorithmen besteht darin, eine Menge von Objekten zu verarbeiten und in diesen auf unterschiedliche Art und Weise Muster zu erkennen oder Gesetzesmäßigkeiten nachzubilden. Als Hilfestellung verwenden viele Lernalgorithmen hierfür intern ein Modell, das mit den verfügbaren Daten aufgebaut wird. Mit dem erlernten Wissen ist es dem Lernalgorithmus daraufhin möglich, neue Objekte in dem erstellten Modell einzuordnen und für diese bestimmte Aussagen treffen zu können. Eine solche Aussage kann zum Beispiel die Klassifikation sein, bei der das Objekt einer oder mehreren Klassen zugeordnet wird.

Die stetig zunehmende Menge an Daten und die dadurch wachsende Zunahme an Rechenzeit für die Erstellung der Modelle stellt für die meisten Lernalgorithmen ein Problem dar. Oftmals müssen die Daten für viele Algorithmen genau analysiert und ausgewertet werden, bevor das Modell erstellt werden kann. Problematisch ist es, wenn das Modell aus einem nicht endenden Datenstrom von Objekten erstellt werden soll. In diesem Fall müssen die Lernalgorithmen die Objekte inkrementell dem Modell hinzufügen. Ein wichtiger Aspekt spielt außerdem die Verarbeitungszeit der Objekte bei dem ständig anhaltendem Datenstrom. Die Objekte müssen direkt bei ihrer Ankunft verarbeitet werden, weil eine Speicherung der Daten schnell zu Speicherplatzproblemen führen könnte. Nur sehr wenige Lernalgorithmen aus dem Bereich des maschinellen Lernens erfüllen diese Bedingungen. Aus diesem Grund wurden Online-Lernalgorithmen entwickelt, die speziell für die Verarbeitung von nicht endenden Datenströmen konzipiert wurden.

Im Bereich des Online-Lernens stellt das Verfahren der zufälligen Entscheidungsbäume eine attraktive Option dar. Entscheidungsbäume eignen sich optimal für das inkrementelle Lernen und können durch ihren zufälligen Aufbau extrem schnell mit Objekten angepasst werden. Aus diesem Grund besteht hauptsächlich die Motivation dieser Bachelorarbeit darin, einen Lernalgorithmus für das Online-Lernen zu entwickeln und zu implementieren, der auf den Grundlagen von zufälligen Entscheidungsbäumen basiert.

1.1 Ziel der Arbeit

Im Rahmen dieser Bachelorarbeit wird ein modulares Programm entwickelt, mit welchem Ensembles von zufälligen Entscheidungsbäumen erstellt werden können. Der Beweggrund für den Entwurf des Programmes liegt darin, eine Grundlage für die Entwicklung unterschiedlicher Verfahren zur Erstellung von Entscheidungsbäumen schaffen zu können. Durch das Programm können neue Ideen ausprobiert und deren Tauglichkeit getestet werden. Zusätzlich wird eine universelle Struktur der Entscheidungsbäume entworfen, mit welcher es möglich sein soll, unterschiedliche Lernaufgaben gleichzeitig mit einem einzigen Entscheidungsbaum durchführen zu können. Das bedeutet, dass der Algorithmus mehrere Muster in einer Menge von Objekten erkennen und diese auf den gleichen Entscheidungsbaum abbilden kann. Eine solche Funktionalität ist von Vorteil, wenn mehrere Eigenschaften in einer Menge von Daten gleichzeitig untersucht werden sollen.

Anhand des erstellten Programms sollen innerhalb dieser Bachelorarbeit zwei Verfahren für die Erstellung der Entscheidungsbäume entworfen und implementiert werden. Die Motivation für die Entwicklung des ersten Verfahrenes besteht darin, eine schnelle Kreuzvalidierung durchführen zu können. Eine schnelle Kreuzvalidierung kann beispielsweise eingesetzt werden, um die optimalen Parametereinstellungen für ein gegebenes Lernproblem schneller finden zu können. Dazu wird der erste Algorithmus unter der Annahme des Batch-Lernens entwickelt. Das bedeutet, dass für diesen Algorithmus ein kompletter Datensatz vorliegt, dessen Objekte für den Aufbau des Modelles mehrmals verwendet werden können. Unter der Verwendung des dekrementellen Lernens konnte in dieser Arbeit eine Methode entworfen werden, die sich besonders gut für eine schnelle Leave-One-Out-Kreuzvalidierung eignet. Die Einsetzbarkeit des entworfenen Verfahrens ließ sich in den Versuchen bestätigen.

Die grundlegende Motivation für die Entwicklung des zweiten Verfahrens besteht in der Einsetzbarkeit für das Online-Lernen. Dazu soll ein neuartiges Verfahren für die Erstellung der Entscheidungsbäume entwickelt werden, das für die Verarbeitung von nicht endenden Datenströmen entworfen werden soll. In dem Bereich des Online-Lernens stellt die Diskretisierung von numerischen Attributen eine Herausforderung dar, weil die Werteverteilungen nicht zu Beginn bekannt sind. Eine erfolgversprechende Methode stellt die Diskretisierung unter der Verwendung von Quantilen dar. Die Idee ist, während des Aufbaus der Entscheidungsbäume Werte von Attributen zu sammeln, um anschließend durch Quantile die Trennwerte in den Tests der Entscheidungsbäume bestimmen zu können. Des Weiteren soll für den entworfenen Algorithmus ein Verfahren für die Aggregation der erstellten Entscheidungsbäume entwickelt werden. In diesem Bezug sollen unterschiedliche Verfahren der Aggregation bei einer Kreuzvalidierung vorgestellt und diskutiert werden. In unterschiedlichen Experimenten ließ sich die Einsetzbarkeit von Quantilen für die Diskretisierung bestätigen. Besonders auf Datensätzen mit vielen numerischen Attributen konnte das entworfene Verfahren gegenüber dem Hoeffding-Tree in dem Bereich der Laufzeit und der Ergebnisse besser abschneiden.

1.2 Struktur der Arbeit

Zunächst wird in Kapitel 2 auf die Grundlagen eingegangen, die für das Verständnis dieser Arbeit benötigt werden. Anschließend wird in dem Kapitel 3 auf Baumstrukturen eingegangen und wie diese als Entscheidungsbaum genutzt werden können. Des Weiteren wird in diesem Kapitel das Verfahren der zufälligen Entscheidungsbäumen erläutert und es werden Lernalgorithmen vorgestellt, die speziell für das Online-Lernen entwickelt wurden oder mit Ensembles von Entscheidungsbäumen arbeiten. In dem darauffolgenden Kapitel 4 werden die Anforderungen für das zu erstellende Programm aufgelistet und mögliche Lösungen diskutiert. Unter Berücksichtigung dieser Analyse wird anschließend das entworfene Programm vorgestellt. Daraufhin wird in Kapitel 5 auf die entworfenen Algorithmen für die Erstellung von Entscheidungsbäumen eingegangen und der Funktionsweise dargestellt. Daraufhin wird in Kapitel 5.2 auf den in dieser Bachelorarbeit entworfenen Online-Lernalgorithmus eingegangen und dessen Funktionsweise dargestellt. Anschließend werden in Kapitel 6 die beiden implementierten Verfahren für die Erstellung der Entscheidungsbäume anhand von ausgewählten Datensätzen in unterschiedlichen Versuchen evaluiert. Abschließend werden in Kapitel 7 die Ergebnisse dieser Arbeit zusammengefasst und es wird ein Ausblick für zukünftige Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe definiert, die für das Verständnis der folgenden Arbeit nötig sind. Hierbei wird auf die unterschiedlichen Arten der Klassifikation und auf die Regression eingegangen. Des Weiteren wird erklärt, nach welchen Prinzipien und Bedingungen ein Lernverfahren arbeiten kann und mit welchen Problemen ein Lernalgorithmus konfrontiert ist. Zum Schluss dieses Kapitels werden die Möglichkeiten vorgestellt, mit denen ein Lernalgorithmus evaluiert werden kann.

2.1 Instanzen

In dieser Arbeit wird häufig der Begriff der Instanz verwendet. Durch eine Instanz wird ein Objekt beschrieben, welches durch bestimmte Eigenschaften definiert ist. Die Eigenschaften eines Objekts werden auch Attribute genannt. Durch Tabelle 2.1 sind beispielhaft vier verschiedene Instanzen aufgelistet, wobei jede Zeile der Tabelle genau eine Instanz abbildet. In diesem Beispiel stellen *Hubraum*, *Leistung*, *Leergewicht*, *Marke* und *Modell* die Attribute dar. Hierbei unterscheidet man zwischen numerischen und nominalen Attributen. Die Werte numerischer Attribute können durch Zahlen dargestellt werden, wobei der Wertebereich nicht definiert werden muss. Bei unserem Beispiel ist das für die Attribute *Hubraum*, *Leistung* und *Leergewicht* der Fall. Bei nominalen Attributen werden die Werte, die angenommen werden können, durch eine Menge dargestellt. So hat zum Beispiel das Attribut *Marke* die Wertemenge {*Mercedes, Volkswagen, Porsche, Ford*}, wobei jede Instanz genau einem dieser Werte angehört. Bei dem Attribut *Modell* handelt es sich ebenfalls um ein nominales Attribut, dessen Wertemenge ebenfalls aus vier Werten besteht. Um im Bereich des maschinellen Lernens mit Instanzen arbeiten zu können, werden alle Instanzen, die dieselben Attribute haben in einem Datensatz verwaltet. Mit den vier Instanzen aus der Tabelle kann ein Datensatz gebildet werden.

Tabelle 2.1: Instanzen

Hubraum (Kubikzentimeter)	Leistung (Pferdestärken)	Leergewicht (Kilogramm)	Marke	Modell
2497	90	1320	Mercedes	W124
1390	60	960	Volkswagen	Golf 3
2993	260	1195	Porsche	930
4736	220	1109	Ford	Mustang

2.2 Überwachtes Lernen

Im Bereich des maschinellen Lernens unterscheidet man hauptsächlich zwischen dem unüberwachten, dem überwachten und dem bestärkenden Lernen. Unüberwachtes Lernen zeichnet sich dadurch aus, dass beim Lernen nur der Datensatz vorhanden ist und keine weiteren Informationen zu dem Problem gegeben sind. In diesem Fall ist die Aufgabe des Lernalgorithmus den Datensatz zu verarbeiten, um ein Muster in den Daten erkennen zu können. Ein sehr bekanntes Verfahren, welches unter der Bedingung des unüberwachten Lernens arbeitet, ist der Clustering-Algorithmus. Von einem überwachten Lernen ist die Rede, wenn dem Lernalgorithmus, zusätzlich zu dem Datensatz, Expertenwissen zur Verfügung steht. Das bedeutet, dass das Muster der Daten bereits vorgegeben ist, wobei die Aufgabe des Lernalgorithmus darin besteht, eine Funktion zu finden, die auf dieses Muster abbildet. Beispielsweise könnte die Aufgabe eines Lernalgorithmus sein, die Instanzen eines Datensatzes in gute und schlechte Instanzen einzuteilen. In diesem Fall wird das Expertenwissen dadurch ausgedrückt, dass bereits ein Datensatz vorhanden ist, dessen Instanzen in gute und schlechte eingeteilt wurde. Diesen Datensatz kann der Lernalgorithmus verwenden, um die Gesetzmäßigkeit nachzubilden. Bei dem bestärkenden Lernen handelt es sich um eine Mischung von überwachtem und unüberwachtem Lernen. In diesem Fall steht dem Lernalgorithmus der Datensatz ohne zusätzliche Informationen zur Verfügung, jedoch ist es dem Lernalgorithmus möglich nach Expertenwissen zu fragen. In dieser Arbeit beschränken wir uns auf das überwachte Lernen.

2.3 Klassifikation

Der Begriff der Klassifikation wird verwendet, wenn eine Instanz einer oder mehreren Klassen zugeordnet wird. In dem Beispiel aus Tabelle 2.1 ist ersichtlich, dass die Tabelle um ein oder mehrere weitere nominale Attribute erweitert

wird. Die Aufgabe des Lernalgorithmus, welcher bei solch einer Anwendung auch Klassifikationsverfahren genannt wird, ist es, für jedes der hinzugefügten nominalen Attribute den entsprechenden Wert zu bestimmen. Unter der Annahme des überwachten Lernens wird dem Klassifikationsverfahren ein Datensatz übergeben, der Instanzen enthält, bei denen bereits die zu vorhersagenden Klassen angegeben sind. Nachdem das Klassifikationsverfahren den Datensatz verarbeitet und ein Modell für die Klassifizierung erstellt hat, ist es möglich, dem Algorithmus nicht klassifizierte Instanzen zu übergeben, um für diese die Klassen zu bestimmen. In den folgenden Abschnitten werden die Arten der Klassifikationen beschrieben, die sich durch die Anzahl der zu vorhersagenden Klassen und Attribute unterscheiden.

2.3.1 Binäre Klassifikation

Die binäre Klassifikation ist die bekannteste Art der Klassifikation. Ein Klassifikationsverfahren, das binäre Klassifikation vornimmt, ordnet einer Instanz genau eine von zwei möglichen Klassen zu. Das bedeutet, dass das nominale Attribut, welches die Klasse beschreibt, eine Wertemenge besitzt, die aus zwei Werten besteht. Beispielsweise kann man die Autos aus Tabelle 2.1 in die Klassen schnelle und langsame Autos einteilen. Hierfür müssen wir die Tabelle um ein weiteres nominales Attribut erweitern, welches die Wertemenge {*langsam, schnell*} besitzt. In Tabelle 2.2 wird diese Erweiterung abgebildet, wobei *langsam* und *schnell* die Klassen darstellen.

Tabelle 2.2: Instanzen mit den Klassen *{langsam, schnell}*

Hubraum (Kubikzentimeter)	**Leistung (Pferdestärken)**	**Leergewicht (Kilogramm)**	**Marke**	**Modell**	***Geschwindigkeit***
2497	90	1320	Mercedes	W124	*langsam*
1390	60	960	Volkswagen	Golf 3	*langsam*
2993	260	1195	Porsche	930	*schnell*
4736	220	1109	Ford	Mustang	*schnell*

2.3.2 Multiklassen-Klassifikation

Von einer Multiklassen-Klassifikation ist die Rede, wenn ein Klassifikationsverfahren einer Instanz einer von drei oder mehreren Klassen zuordnet. Diese Art der Klassifikation unterscheidet sich von der binären Klassifikation nur in der Anzahl der Klassen, die vorhergesagt werden können. Wie bei der binären Klassifikation wird die Klasse nur durch ein einziges nominales Attribut beschrieben. Für eine solche Klassifikation könnte die Tabelle beispielsweise um ein Attribut erweitert werden, das den Komfort eines Autos angibt. Dieses Attribut könnte in diesem Beispiel die Wertemenge {*niedrig, mittel, hoch*} besitzen, welche aus drei Werten besteht. Tabelle 2.3 stellt ein solches Szenario dar.

Tabelle 2.3: Instanzen mit den Klassen *{niedrig, mittel, hoch}*

Hubraum (Kubikzentimeter)	**Leistung (Pferdestärken)**	**Leergewicht (Kilogramm)**	**Marke**	**Modell**	***Komfort***
2497	90	1320	Mercedes	W124	*hoch*
1390	60	960	Volkswagen	Golf 3	*niedrig*
2993	260	1195	Porsche	930	*mittel*
4736	220	1109	Ford	Mustang	*mittel*

2.3.3 Multilabel-Klassifikation

Bei den beiden bereits vorgestellten Klassifikationsarten wird einer Instanz genau eine Klasse zugeordnet. Bei dem Beispiel der binären Klassifikation aus Tabelle 2.2 ist es nur möglich, einer der Klassen {*langsam, schnell*} anzugehören. Es ist nicht möglich, dass ein Auto gleichzeitig als *schnell* und als *langsam* klassifiziert wird oder dass ein Auto keiner der beiden Klassen angehört. Bei den beiden vorgestellten Klassifikationsarten haben sich die Klassen gegenseitig ausgeschlossen. Anders ist dies bei der Multilabel-Klassifikation, wobei es möglich ist, dass eine Instanz keiner, einem Teil oder allen Klassen gleichzeitig angehören kann. Ein solches Verhalten kann dadurch abgebildet werden, dass jede Klasse durch ein nominales Attribut dargestellt wird, dessen Wertemenge aus zwei Elementen besteht. Durch die zwei Werte wird angegeben, ob die Klasse angenommen wird oder nicht. In dem Beispiel der Autos könnte man beispielsweise die Klassen *Beliebtheit*, *Seltenheit* und *Unfallsicherheit* hinzufügen. In Tabelle 2.4 ist dieses Szenario dargestellt, wobei der

Wert *ja* angibt, dass die Klasse angenommen und der Wert *nein* angibt, dass die Klasse nicht angenommen wird. Beispielsweise gehört die zweite Instanz aus der Tabelle keiner der drei Klassen an. Die Aufgabe eines Klassifikationsverfahren, das Multilabel-Klassifikation durchführt, ist es zu bestimmen, welchen Klassen eine Instanz angehört.

Tabelle 2.4: Instanzen mit den Klassen *Beliebtheit, Seltenheit* und *Unfallsicherheit*

Hubraum (Kubikzentimeter)	**Leistung (Pferdestärken)**	**Leergewicht (Kilogramm)**	**Marke**	**Modell**	***beliebt***	***selten***	***unfallsicher***
2497	90	1320	Mercedes	W124	*ja*	*nein*	*ja*
1390	60	960	Volkswagen	Golf 3	*nein*	*nein*	*nein*
2993	260	1195	Porsche	930	*ja*	*ja*	*nein*
4736	220	1109	Ford	Mustang	*ja*	*nein*	*ja*

2.4 Regression

Der Begriff der Regression wird verwendet, wenn ein numerischer Wert für eine Instanz vorhergesagt werden soll. Im Gegensatz zur Klassifikation ist in diesem Fall das zu vorhersagende Attribut numerisch. Das Beispiel aus Tabelle 2.1 könnte um ein numerisches Attribut erweitert werden, das den Preis darstellt. Tabelle 2.5 bildet dieses Szenario ab. In diesem Fall ist die Aufgabe des Lernalgorithmus den Preis für die Instanzen zu bestimmen. Vergleichbar mit der Multilabel-Klassifikation existiert für die Regression auch ein Verfahren, bei dem mehrere numerische Werte vorhergesagt werden können. In diesem Fall spricht man von einer Multitarget-Regression.

Tabelle 2.5: Instanzen mit Regressionswerten für das Attribut *Preis*

Hubraum (Kubikzentimeter)	**Leistung (Pferdestärken)**	**Leergewicht (Kilogramm)**	**Marke**	**Modell**	***Preis (Euro)***
2497	90	1320	Mercedes	W124	*3.455*
1390	60	960	Volkswagen	Golf 3	*1.885*
2993	260	1195	Porsche	930	*99.900*
4736	220	1109	Ford	Mustang	*19.650*

2.5 Batch-Lernen

Von Batch-Lernen ist die Rede, wenn dem Lernalgorithmus alle Instanzen, die für das Lernen verwendet werden, sofort zur Verfügung stehen. Das bedeutet, dass alle Instanzen vorher gesammelt und in einem Datensatz zusammengefasst werden müssen. Dieser Datensatz wird dem Lernalgorithmus als Ganzes übergeben. Durch die Verfügbarkeit aller Instanzen zum gleichen Zeitpunkt ist es möglich, die Daten genau zu analysieren, um zum Beispiel den Wertebereich von numerischen Attributen bestimmen zu können. In vielen Fällen ist es einfacher ein gutes Modell für ein Lernproblem zu erstellen, wenn der Lernalgorithmus die Daten vorher analysieren kann. Durch die Verfügbarkeit eines kompletten Datensatzes besteht für das Lernen die Möglichkeit, die Instanzen des Datensatzes mehrmals zu verwenden.

2.6 Inkrementelles Lernen

Das inkrementelle Lernen unterscheidet sich von dem Batch-Lernen in der Verfügbarkeit der Instanzen. Bei dem inkrementellen Lernen werden dem Lernalgorithmus die Instanzen nicht als kompletter Datensatz übergeben, sondern werden einzeln nacheinander dem Algorithmus übermittelt. Ein Lernalgorithmus, der inkrementell lernen kann, muss die Fähigkeit haben, sein erstelltes Modell mit jeder ankommenden Instanz neu anpassen zu können. Bei dieser Art des Lernens ist es nicht möglich, die Daten vorher zu analysieren. Dementsprechend muss der Lernalgorithmus während des Lernprozesses nützliche Informationen erkennen, um sein Modell verbessern beziehungsweise an die Gegebenheiten anpassen zu können. So kann zum Beispiel der Wertebereich eines numerischen Attributes mit jeder neuen Instanz angepasst werden. Des Weiteren liegen dem Lernalgorithmus nicht die Wertebereiche der nominalen Attribute vor, weshalb der Algorithmus in der Lage sein muss, auch noch nicht gesehene Werte verarbeiten zu können. Nicht jeder Lernalgorithmus aus dem Bereich des maschinellem Lernen kann diese speziellen Anforderungen erfüllen, wodurch einige Algorithmen nicht für das inkrementelle Lernen geeignet sind.

2.7 Online-Lernen

Bei dem Online-Lernen geht man von einer speziellen Art des inkrementellem Lernen aus. Nach [Khouzam, 2009] unterscheidet man zwischen zwei unterschiedlichen Arten des inkrementellen Lernens. Zum einen gibt es das inkrementelle Lernen, das Offline stattfindet. Hierbei liegt ein kompletter Datensatz von Instanzen vor. Um den Speicherbedarf möglichst gering zu halten, werden die Instanzen des Datensatzes nicht sofort eingelesen, sondern einzeln nacheinander übermittelt. Mit dieser Methode können auch sehr große Datensätze von einem Lernalgorithmus verarbeitet werden, die normalerweise nicht in den Speicher passen würden. Bei einem inkrementellen Lernen, das Online stattfindet, geht man davon aus, dass ein unendlicher Strom an Daten existiert. Hierbei lassen sich die Instanzen nicht in einem Datensatz zusammenfassen und müssen dem Lernalgorithmus direkt einzeln übergeben werden. Kritisch für einen Algorithmus, der Online arbeitet, ist der Speicherplatzbedarf und die Verarbeitungszeit. Online Algorithmen sind dafür konzipiert, alle eintreffenden Instanzen des Datenstroms verarbeiten zu können. Aufgrund des durchgehenden Datenstroms muss der Lernalgorithmus die Fähigkeit haben, schnell zu arbeiten, wodurch meistens die Genauigkeit des erstellten Modells sinkt. Eine Speicherung der Instanzen und eine anschließende Verarbeitung ist in diesem Fall auch nicht möglich, da dies schnell zu Problemen mit dem Speicherplatz und der Verarbeitungszeit führt. Eine weitere Eigenschaft von Algorithmen, die Online lernen können, ist, dass jederzeit ein fertiges Modell für das Lernproblem vorliegt, welches direkt verwendet werden kann.

2.8 Dekrementelles Lernen

Unter der Annahme, dass ein Lernalgorithmus ein Modell durch eines der drei vorgestellten Lernverfahren gebildet hat, gibt es noch eine Art des Lernens, mit der es möglich ist, bereits gelernte Instanzen wieder zu vergessen. In diesem Fall spricht man von dekrementellen Lernen. Dabei handelt es sich nicht um eine Methode, mit der man ein Modell mit einem Lernalgorithmus erstellen kann und ist deshalb nur in Kombination mit einem der drei bereits vorgestellten Lernverfahren anwendbar. Nachdem der Lernalgorithmus mit einem der Lernverfahren ein fertiges Modell erstellt hat, ist es mit dem dekrementellen Lernen möglich, beliebige Instanzen, die für das Lernen verwendet wurden, wieder aus dem Modell zu entfernen. So besteht beispielsweise die Möglichkeit, einen Lernalgorithmus mit einem Datensatz, der vier Instanzen enthält, lernen zu lassen. Das daraus resultierende Modell beruht auf den vier Instanzen. Durch das dekrementelle Lernen kann beispielsweise eine bestimmte Instanz des Datensatzes aus dem Modell entfernt werden, sodass dieses nur noch auf drei Instanzen basiert. Die Anforderung an einen Lernalgorithmus, der dekrementelles Lernen unterstützt, ist, jede gelernte Instanz in dem Modell so zu verwalten, dass sie wieder entfernt werden kann. Aus diesem Grund gibt es nur wenige Lernalgorithmen, die für ein dekrementelles Lernen geeignet sind. Anwendung findet das dekrementelle Lernen beispielsweise bei der Analyse des Kaufverhaltens von Nutzern eines Onlinekaufhauses. In diesem Fall werden die Daten von registrierten Nutzern verwendet, um einen Lernalgorithmus zu trainieren, der den Nutzern Produkte empfiehlt. Tritt nun der Fall ein, dass ein registrierter Nutzer des Onlinekaufhauses sein Benutzerkonto kündigt, so dürfen seine Daten aus rechtlichen Gründen nicht mehr verwendet werden. Das bedeutet, dass die Daten des Nutzers auch aus dem Modell des Lernalgorithmus entfernt werden müssen.

2.9 Fehlende Werte

Von einem fehlenden Wert ist die Rede, wenn bei einer Instanz der Wert für ein Attribut nicht angegeben ist. Tabelle 2.6 stellt einen Datensatz dar, der fehlende Werte enthält. In Datensätzen, die aus der realen Welt stammen, sind fehlende Werte keine Seltenheit, weil die Daten oft durch elektronische Geräte erfasst werden. Bei der Verwendung dieser Geräte kann es zu unterschiedlichen Fehlerquellen kommen. So kann zum Beispiel ein Messfehler oder ein Übertragungsfehler der Ursprung eines fehlenden Wertes sein. Für einen Lernalgorithmus bedeutet dies, dass solche fehlenden Werte berücksichtigt und auf eine bestimmte Art und Weise mit ihnen umgegangen werden muss. In manchen Fällen ist der fehlende Wert für den Lernalgorithmus uninteressant, weil er für das Lernproblem nicht relevant ist. Dies kann zum Beispiel der Fall sein, wenn wir die Instanzen aus Tabelle 2.6 in schnelle und langsame Autos klassifizieren wollen. Hierbei spielt das

Tabelle 2.6: Instanzen mit fehlenden Werten

Hubraum (Kubikzentimeter)	Leistung (Pferdestärken)	Leergewicht (Kilogramm)	Marke	Modell	*Geschwindigkeit*
?	90	1320	Mercedes	W124	*langsam*
1390	60	?	Volkswagen	Golf 3	*langsam*
2993	?	1195	?	930	*schnell*
4736	220	?	?	?	*schnell*

Attribut *Marke* eine untergeordnete Rolle, hingegen sind die Attribute *Hubraum* und *Leistung* für dieses Lernproblem sehr nützlich. Das bedeutet, dass zum Beispiel die vierte Instanz aus der Tabelle, trotz der drei fehlenden Werte, gut für dieses Lernproblem geeignet ist.

2.10 Ensembles

In dem Bereich des maschinellen Lernens versteht man unter einem Ensemble eine Zusammensetzung von mehreren Lernalgorithmen, die für das gleiche Lernproblem eingesetzt werden. Hierbei wird jeder Lernalgorithmus des Ensembles mit einem Teil oder dem ganzen verfügbaren Datensatz trainiert. Um eine Vorhersage für eine Instanz machen zu können, werden die Vorhersagen aller Lernalgorithmen eines Ensembles kombiniert. Ensembles wurden entwickelt, um dem Bias-Varianz-Dilemma[1] entgegen zu wirken. Der Bias ist der Fehler, der durch ein schlecht erstelltes Modell entsteht und die Varianz ist der Fehler, der durch eine schlechte Auswahl des Datensatzes entsteht. Das Bias-Varianz-Dilemma sagt aus, dass Modelle mit einem niedrigen Bias eine hohe Varianz haben, und dass Modelle mit einem hohen Bias eine niedrige Varianz haben. Mit einem Ensemble von Lernalgorithmen ist es möglich, die Varianz und den Bias gleichzeitig zu verringern.

Für die Erstellung eines Ensembles gibt es unterschiedliche Varianten, wie zum Beispiel das *Bagging*. Bei diesem Verfahren wird der vorliegende Datensatz verwendet, um speziell für jeden Lernalgorithmus in dem Ensemble einen Datensatz erstellen zu können. Die erstellten Datensätze werden dabei zufällig mit den Instanzen aus dem vorliegenden Datensatz gefüllt. Mit den erstellten Datensätzen werden anschließend die einzelnen Modelle des Ensembles erzeugt. Die einfachste Variante für die Erstellung von Ensembles ist, dass jedem Lernalgorithmus des Ensembles der vorliegende Datensatz ohne Veränderungen übergeben wird. Allerdings müssen sich in diesem Fall die erzeugten Modelle der Lernalgorithmen unterscheiden, um den Vorteil von Ensembles ausnutzen zu können.

2.11 Evaluierungsarten

In diesem Abschnitt werden unterschiedliche Möglichkeiten vorgestellt, mit denen Lernalgorithmen evaluiert werden können. Die vorgestellten Evaluierungsarten beruhen auf der Annahme, dass ein Datensatz zur Verfügung steht, dessen Instanzen bereits mit den Werten der vorherzusagenden Attributen annotiert sind. Ein Teil dieses Datensatzes kann genutzt werden, um ein Modell für das Lernproblem erstellen zu können. Mit dem anderen Teil der Daten ist es anschließend möglich, die Güte des Lernalgorithmus zu überprüfen, indem für jede Instanz der Lernalgorithmus eine Vorhersage machen muss und diese mit dem tatsächlichen Wert überprüft und bewertet werden kann. In vielen Fällen sind annotierte Datensätze rar, wodurch es für die Evaluierung sinnvoll ist, die Daten möglichst effizient und realitätsgetreu zu verwenden.

2.11.1 Evaluierung auf einer Testmenge

Bei der Evaluierung auf einer Testmenge wird der Datensatz zuerst in eine Trainingsmenge und eine Testmenge aufgeteilt. Mit der Trainingsmenge wird das Modell erstellt, welches anschließend mit der Testmenge evaluiert werden kann. Damit die Evaluierung auf der Testmenge aussagekräftig ist, müssen genügend Instanzen in dieser Menge vorhanden sein. Dies führt dazu, dass für die Erstellung des Modells weniger annotierte Instanzen verwendet werden können, wodurch die Qualität des Modells sinkt. Durch die Aufteilung des Datensatzes können die Instanzen nicht effizient genutzt werden. Verwendet man als Trainings- und Testmenge denselben Datensatz, so sind die Ergebnisse oft nicht realitätsgetreu, da das Modell die zu evaluierenden Instanzen bereits einmal gesehen und verarbeitet hat.

2.11.2 Evaluierung durch eine Kreuzvalidierung

Bei der Kreuzvalidierung wird der Datensatz in n gleich große Teile aufgeteilt. Jeder dieser Teile des Datensatzes kann nun individuell als Trainings- oder Testmenge verwendet werden. Nach dem Prinzip der Kreuzvalidierung wird immer auf n-1 Teilen des Datensatzes das Modell erstellt und auf dem übrig gebliebenen Teil des Datensatzes evaluiert. Dieses Verfahren wird n-mal durchgeführt, sodass jeder Teil des Datensatzes einmal als Evaluierungsmenge verwendet wird. Man spricht von einer n-fachen Kreuzvalidierung, wenn das Verfahren auf einen Datensatz angewendet wird, der in n Teile aufgeteilt wird. In Abbildung 2.1 ist beispielhaft die Aufteilung eines Datensatzes dargestellt, der für eine vierfache Kreuzvalidierung genutzt werden kann. Bei der ersten Iteration wird der erste, der zweite und der dritte Teil des Datensatzes verwendet um das Modell zu erstellen und auf dem vierten Teil wird anschließend evaluiert. In der nächsten

[1] `http://www.ke.tu-darmstadt.de/lehre/ws-14-15/mldm/ensembles.pdf` Abgerufen am: 21.05.2015

Iteration wird der erste, der zweite und der vierte Teil des Datensatzes verwendet um das Modell zu erstellen und auf dem dritten Teil zu evaluieren. Analog wird das Verfahren für die nächsten beiden Iterationen fortgeführt. Dieses Verfahren ist realitätsgetreu, da bei jeder Iteration auf Daten evaluiert wird, die nicht für die Erstellung des Modells verwendet wurden. Auch die Evaluierungsergebnisse dieses Verfahrens sind aussagekräftig, weil jede Instanz des Datensatzes einmal getestet wird. Je größer der Parameter n gesetzt wird, desto realistischer werden die Daten des Datensatzes genutzt, jedoch steigt dadurch auch die Zeit für die Evaluierung, weil der Lernalgorithmus n Modelle erstellen muss.

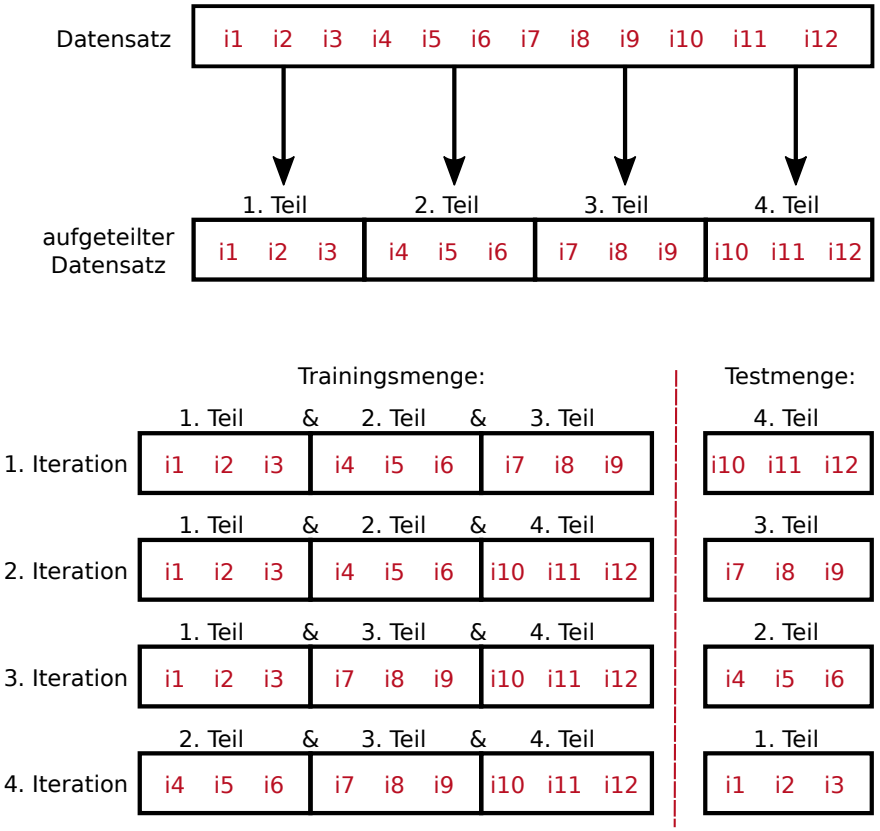


Abbildung 2.1: Kreuzvalidierung mit vier Teilen auf einem Datensatz mit zwölf Instanzen. Die Instanzen sind durch die rote Schrift gekennzeichnet.

2.11.3 Evaluierung durch eine Leave-One-Out-Kreuzvalidierung

Die Leave-One-Out Kreuzvalidierung (LOO-Kreuzvalidierung) ist eine Kreuzvalidierung, bei der die Anzahl der Teile des Datensatzes der Anzahl der Instanzen im Datensatz entspricht. Das bedeutet, dass jeder Teil des Datensatzes, der für die Kreuzvalidierung verwendet wird, aus einer Instanz besteht. Dadurch wird für die Evaluierung jeder Instanz des Datensatzes ein gesondertes Modell erstellt, das mit den restlichen Instanzen des Datensatzes trainiert wird. Dieses Verfahren ist am besten für die Evaluierung geeignet, weil die verfügbare Menge der Daten für die Erstellung des Modells maximal ausgeschöpft wird und trotzdem auf einer Instanz evaluiert werden kann, die nicht für die Erstellung des Modells verwendet wurde. Der Nachteil dieses Verfahrens ist, dass sehr viele Modelle erstellt werden müssen, was je nach Verfahren sehr zeitintensiv sein kann. Außerdem spielt die Größe des Datensatzes eine entscheidende Rolle, weil durch die Anzahl der Instanzen die Anzahl der Iterationen der LOO-Kreuzvalidierung festgelegt wird.

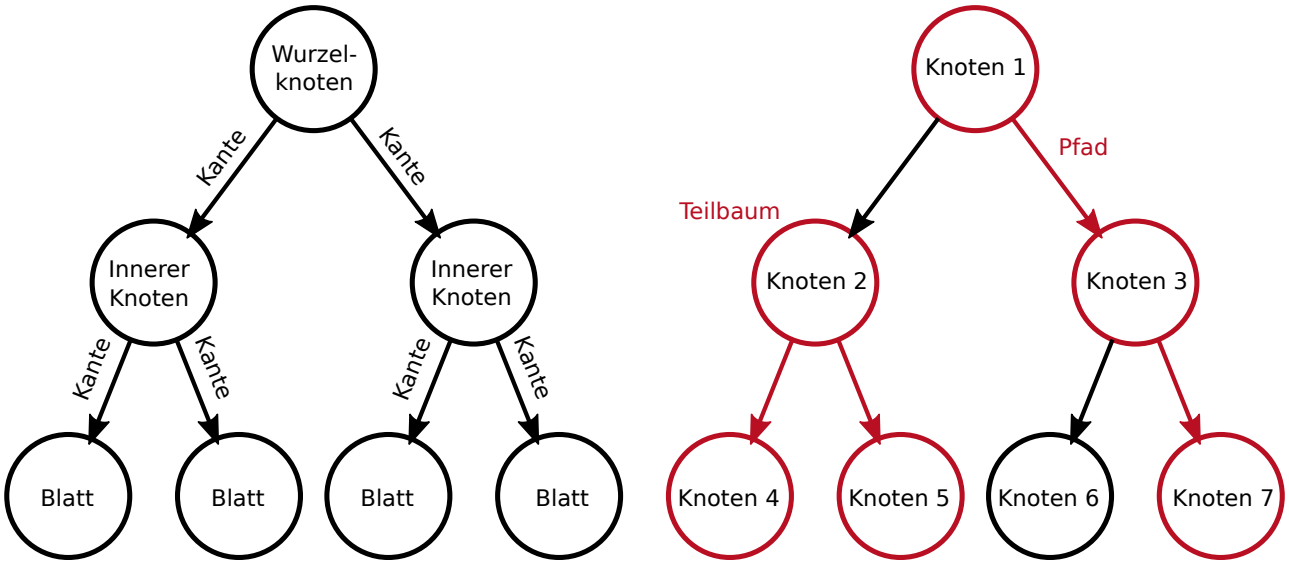
3 Zufällige Entscheidungsbäume und verwandte Arbeiten

In diesem Kapitel werden die Funktionsweise und die Eigenschaften von Entscheidungsbäumen erklärt, wobei zuerst auf die grundlegende Struktur von Bäumen eingegangen wird. In den darauffolgenden Abschnitten wird auf verwandte Arbeiten eingegangen, wobei Algorithmen vorgestellt werden, die Ensembles von Entscheidungsbäumen verwenden. Des Weiteren wird ein Verfahren für die Erstellung der Bäume vorgestellt, das speziell für das Online-Lernen entwickelt wurde.

3.1 Struktur von Bäumen

In der Informatik werden Entscheidungsbäume genutzt, um eine Abfolge von Entscheidungen darstellen zu können. Bei einem Entscheidungsbaum handelt es sich um eine gerichtete Baumstruktur, die aus Knoten und Kanten besteht. Bei den Knoten unterscheidet man zwischen dem Wurzelknoten, den inneren Knoten und den Blättern. Jeder Baum enthält genau einen Wurzelknoten, welcher den Anfang eines Baumes darstellt. Von diesem Knoten aus zeigen zwei oder mehrere Kanten auf weitere Knoten. In diesem Fall werden die weiteren Knoten auch Kindknoten des Wurzelknotens genannt, weil sie die unmittelbaren Nachfolger des Wurzelknotens sind. Die Kindknoten können die Form eines inneren Knotens oder eines Blattes annehmen. Handelt es sich bei dem Kindknoten um ein Blatt, so bedeutet dies, dass von diesem Knoten aus keine weiteren Kanten auf andere Knoten zeigen und der Baum an dieser Stelle nicht weitergeführt wird. Anders ist es, wenn es sich bei dem Kindknoten um einen inneren Knoten handelt. Innere Knoten haben, wie der Wurzelknoten, zwei oder mehrere Kanten, die auf weitere Knoten zeigen. Die Kindknoten des inneren Knoten können wieder die Gestalt von inneren Knoten oder Blättern annehmen. Diese Prozedur kann solange wiederholt werden, bis nur noch auf Blätter gezeigt wird. Bei einem Entscheidungsbaum wird von einer gerichteten Baumstruktur ausgegangen, weswegen keine Zyklen eingebaut werden dürfen. In Abbildung 3.1a ist beispielhaft ein einfach gerichteter Baum abgebildet, an dem die genannten Begriffe annotiert sind.

In dieser Arbeit werden außerdem noch die Begriffe Teilbaum, Pfad und Tiefe eines Baumes verwendet. Als Teilbaum bezeichnet man einen Abschnitt eines Baumes, der selbst einen Baum darstellen kann. In Abbildung 3.1b wird durch die Knoten mit den Ziffern 2, 4 und 5 ein Teilbaum gebildet. Durch diese Knoten ist es möglich, einen separaten Baum darzustellen. Hierbei wäre der Knoten mit der Ziffer 2 der Wurzelknoten und die Knoten mit den Ziffern 4 und 5 Blätter. Durch einen Pfad wird beginnend mit dem Wurzelknoten eine Abfolge von Knoten zu einem Blatt beschrieben. Bei dem Baum aus Abbildung 3.1b bilden beispielsweise die Knoten mit den Ziffern 1, 3 und 7 einen Pfad. Die Tiefe eines Baumes wird durch den längsten Pfad bestimmt. Die Bäume aus Abbildung 3.1 haben eine Tiefe von zwei, weil die längsten Pfade nur zwei Kanten enthalten.



(a) Baum mit annotierten Begriffen

(b) Teilbaum und Pfad

Abbildung 3.1: Struktur von Bäumen

Ein solcher Baum wird erst dann zu einem Entscheidungsbaum, wenn dem Wurzelknoten und den inneren Knoten sogenannte Tests hinzugefügt werden. Wird diesen Knoten ein Objekt übergeben, so ist es mit dem Test möglich, das übergebene Objekt zu überprüfen und eine Entscheidung zu treffen, an welchen Kindknoten es weitergereicht werden soll. Übergibt man dem Wurzelknoten eines Entscheidungsbaumes ein Objekt, so ist es mit diesem Verfahren möglich, dieses Objekt genau einem Blatt des Baumes zuzuordnen. In den Blättern können nützliche Informationen über die ankommenden Objekte gesammelt oder das Objekt einer bestimmten Kategorie zugeordnet werden. Um einen Entscheidungsbaum als Modell für eine Klassifikation oder Regression einsetzen zu können, muss der Baum Instanzen verarbeiten können. In diesem Fall müssen die Tests Instanzen überprüfen und an den entsprechenden Kindknoten weiterleiten. In den meisten Fällen wird bei den Tests der Wert eines Attributes überprüft. So ist es bei dem Beispiel mit den Automodellen möglich, dass ein Test die Instanzen, die eine Leistung von mehr als 150 Pferdestärken haben, an den ersten Kindknoten weiterleitet und alle anderen Instanzen, die weniger oder gleich viele Pferdestärken haben, dem zweiten Kindknoten übergibt. Hierbei handelt es sich um einen Test, der ein numerisches Attribut verwendet, wobei der Wert 150 auch Trennwert genannt wird. Ein Test muss nicht zwangsmäßig immer nur ein Attribut überprüfen, sondern kann im Prinzip jegliche Gestalt annehmen, mit der es möglich ist zwischen Instanzen zu unterscheiden. Zum Beispiel können mit einem Test auch mehrere Attribute gleichzeitig überprüft werden. Die Blätter des Entscheidungsbaumes werden bei einer Klassifikation oder einer Regression genutzt, um Vorhersagen machen zu können. So ist es möglich für jede Instanz, die von dem Wurzelknoten aus einem Blatt zugeordnet wurde, die zu vorhersagenden Attribute zu bestimmen.

In Abbildung 3.2 ist ein Entscheidungsbaum abgebildet, der für eine binäre Klassifikation genutzt werden kann. Durch diesen Baum können die Klassen *schnell* und *langsam* vorhergesagt werden, welche an den Blättern annotiert sind. Der Baum besteht aus drei inneren Knoten, deren Tests die Attribute *Leistung*, *Marke* und *Leergewicht* verwenden. Der Name des jeweiligen Attributs, auf dem der Test bei einem Knoten arbeitet, wurde in den Knoten hineingeschrieben. An den Kanten ist die Bedingung angefügt, die erfüllt werden muss, damit eine Instanz über diese Kante weitergeleitet werden kann. Wird dem Wurzelknoten nun eine Instanz mit einer Leistung von 220 Pferdestärken und einem Leergewicht von 1109 Kilogramm übergeben, so wird diese Instanz bei dem Test von dem Wurzelknoten an den linken Kindknoten weitergeleitet, weil der Wert der Instanz für die Leistung größer als 200 ist. Bei diesem Knoten angekommen, wird ein weiterer Test durchgeführt, wodurch die Instanz wieder dem linken Kindknoten übergeben wird. Bei dem jetzigen Knoten handelt es sich um ein Blatt, mit dem die Klasse *schnell* vorhergesagt wird. Dementsprechend wird die Instanz mit der Klasse *schnell* klassifiziert.

Die Aufgabe eines Lernalgorithmus, der auf einem oder mehreren Entscheidungsbäumen basiert, ist es, die Bäume zu erstellen. Die meisten solcher Lernalgorithmen aus dem Bereich des maschinellen Lernens unterscheiden sich nur in der Art und Weise, wie die Tests in den Knoten erstellt werden. Ein Problem solcher Algorithmen stellt die Überanpassung an die Daten dar, die für die Erstellung des Baumes verwendet werden. Ausschlaggebend für eine Überanpassung ist die Auswahl der verwendeten Instanzen. In vielen Fällen, besonders bei realen Anwendungsszenarien, enthalten die genutzten Daten Rauschen. Das bedeutet, dass bei einigen Instanzen die Werte der Attribute fehlerhaft sind, oder von dem tatsächlichen Wert abweichen. Grund hierfür kann beispielsweise sein, dass die Daten mit elektronischen Geräten

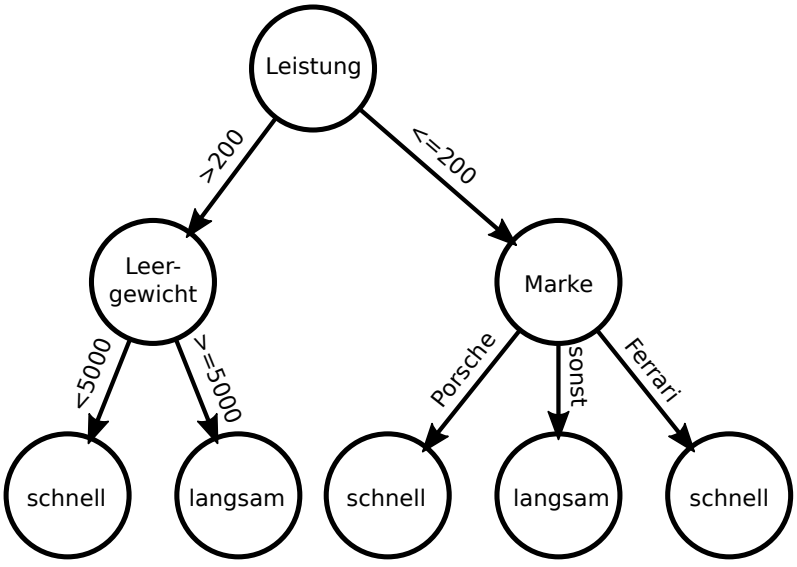


Abbildung 3.2: Entscheidungsbaum für die Klassifikation in *schnelle* und *langsame* Autos

aufgezeichnet wurden und es zu Messfehlern gekommen ist. Mit den meisten Lernalgorithmen wird versucht die genutzten Daten so zu verwenden, dass diese mit dem erstellten Baum perfekt separiert werden können. Das hat zur Folge, dass ein sehr spezifisches Modell entsteht, welches in vielen Fällen bei einer Evaluierung auf andere Daten schlechte Ergebnisse liefert. Somit ist das Modell fehlerhaft und zur Anwendung nicht geeignet. Um der Überanpassung entgegenzuwirken, werden sogenannte Pruning-Verfahren eingesetzt. Mit diesen Verfahren können Entscheidungsbäume während der Erstellung oder im Nachhinein verallgemeinert werden.

Die Struktur eines Entscheidungsbaumes bietet den Vorteil, dass dieser zu jedem Zeitpunkt erweitert werden kann. Es können beispielsweise neue Teilbäume an einen bereits existierenden Knoten angefügt werden. Hierfür muss nur der Test des existierenden Knoten so angepasst werden, dass dem neuen Kindknoten auch Instanzen weitergeleitet werden können. Ein weiterer Vorteil von Entscheidungsbäumen ist, dass diese für den Menschen sehr gut verständlich sind. So können Probleme schnell erkannt und behoben werden. Durch die übersichtlich Struktur und die einfache Erweiterbarkeit von Entscheidungsbäumen eignen sich diese gut für das inkrementelle Lernen.

3.3 Zufällige Entscheidungsbäume

Die meisten Lernalgorithmen, die auf Entscheidungsbäumen basieren, verwenden für die Erstellung der Tests Algorithmen, mit denen es möglich ist, geeignete Attribute und gute Trennwerte finden zu können. Die Auswahl dieser Algorithmen hängt hierbei von der Aufgabe des Lernalgorithmus ab. Für eine binäre Klassifikation werden beispielsweise Algorithmen verwendet, die ein Attribut bestimmen, mit welchem sich die zu vorhersagenden Klassen am besten separieren lassen. Dies ist in vielen Fällen sehr rechenintensiv, weil die Daten genau analysiert werden müssen, um ein gutes Attribut und einen guten Trennwert finden zu können. Eine Alternative bieten die zufälligen Entscheidungsbäume. Zufällige Entscheidungsbäume verwenden keine rechenintensive Algorithmen für die Erstellung der Tests, sondern wählen die Attribute, auf denen getestet werden sollen, zufällig aus. Dies bietet den großen Vorteil, dass der Rechenaufwand, der für die Verwendung dieser Algorithmen gebraucht wird, eingespart werden kann. Ein weiterer Vorteil ist, dass der erstellte Entscheidungsbaum universell für unterschiedliche Lernaufgaben eingesetzt werden kann. Das bedeutet, dass der gleiche Entscheidungsbaum, der für einen Datensatz aufgebaut wurde, sowohl für die Regression als auch für die Klassifikation eingesetzt werden kann. Ausschlaggebend für die universelle Einsetzbarkeit ist, dass in vielen Fällen bei der Erstellung der Tests durch Algorithmen der Entscheidungsbaum an eine bestimmte Lernaufgabe angepasst wird, dies aber bei zufälligen Entscheidungsbäumen vermieden wird. Aus demselben Grund sind die zufälligen Entscheidungsbäume auch nicht so anfällig für eine Überanpassung.

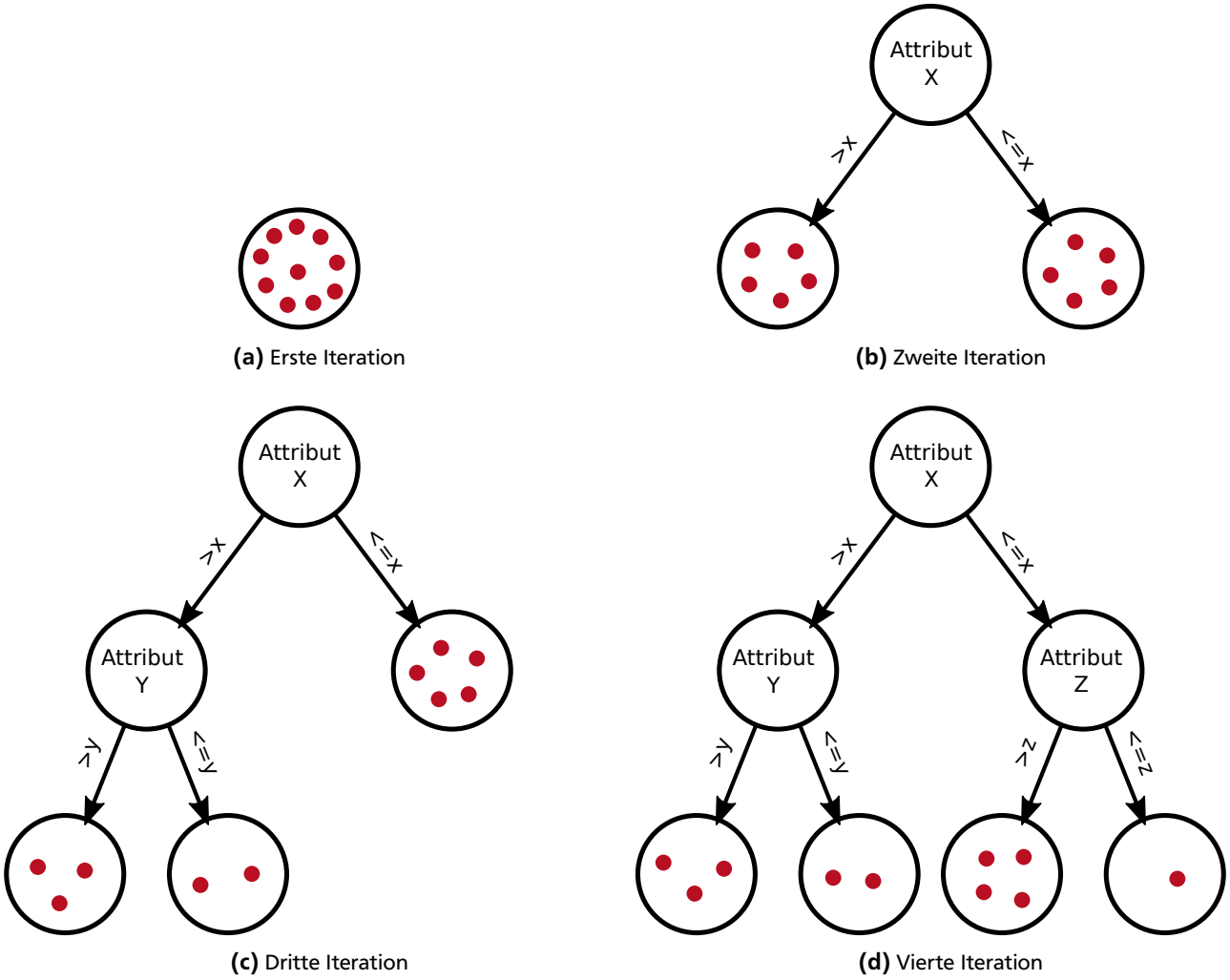
3.4 Dice-Lernalgorithmus

Bei dem Dice-Lernalgorithmus handelt es sich um ein Lernverfahren, das als Modell ein Ensemble von zufälligen Entscheidungsbäumen verwendet. Dieser Lernalgorithmus arbeitet unter der Annahme des Batch-Lernens, wodurch ihm alle Instanzen des Datensatzes beim Lernen gleichzeitig vorliegen. Das Verfahren ist so implementiert, dass die Bäume für unterschiedliche Lernaufgaben eingesetzt werden können. Will man beispielsweise den Algorithmus verwenden, um eine Regression durchzuführen, so kann vor dem Aufbau des Modells eingestellt werden, dass in den Blättern des Entscheidungsbaumes ein numerischer Wert vorhergesagt werden soll. Um die Vorhersagen zu verbessern wird ein Ensemble von zufälligen Entscheidungsbäumen verwendet. Dies ist in diesem Fall sehr vorteilhaft, weil der Aufbau eines Entscheidungsbaumes nicht abhängig von dem Aufbau der anderen Entscheidungsbäume ist. Folglich lässt sich der Rechenaufwand für die Erstellung der Bäume minimieren, indem der Aufbau der Entscheidungsbäume auf mehrere Prozessorkerne aufgeteilt werden kann [Zhang et al., 2010]. Die Funktionalität einer solchen parallelen Arbeitsweise wurde bei diesem Lernalgorithmus noch nicht implementiert. Ein Nachteil dieses Verfahrens ist, dass es nach der Fertigstellung der Entscheidungsbäume nicht möglich ist, das Modell mit weiteren Instanzen anzupassen.

Das Besondere an diesem Algorithmus ist die Art und Weise, wie die Entscheidungsbäume aufgebaut werden. Im ersten Schritt werden alle Instanzen verwendet, um den Wurzelknoten des Entscheidungsbaums zu erstellen. Hierfür wird zufällig eine Instanz ausgewählt und mit dieser ein Test für den Knoten erstellt. Anschließend werden die Instanzen bezüglich des Tests auf die Kindknoten aufgeteilt. In den Kindknoten wird diese Prozedur fortgesetzt, bis die maximale Tiefe des Entscheidungsbaums erreicht oder nicht genügend Instanzen für die Erstellung eines Tests vorhanden sind. In Abbildung 3.3 wird die Erstellung eines Entscheidungsbaumes mit der maximalen Höhe von 2 durch den Dice-Lernalgorithmus grafisch dargestellt. Hierbei werden zehn Instanzen verwendet, die durch die roten Punkte gekennzeichnet sind. Nach der vierten Iteration werden die Instanzen, die den Knoten zugeordnet wurden, verwendet, um die Vorhersage für dieses Blatt zu bestimmen.

Der Vorteil dieses Verfahrens für die Erstellung der Tests ist, dass immer ein Trennwert für das zu testende Attribut bestimmt werden kann, der im Wertebereich des Attributs liegt. Des Weiteren wird der Trennwert von den Instanz ausgewählt, die mit diesem Test separiert werden sollen. Dadurch wird die Wahrscheinlichkeit erhöht, dass die Instanzen,

anhand des erstellten Tests gleichmäßig auf die Kindknoten aufgeteilt werden können. Je besser die Instanzen separiert werden, desto einfacher ist es wichtige Informationen durch die Aufteilung zu gewinnen. Wird beispielsweise ein Trennwert für einen Test ausgewählt, der nicht im Wertebereich des zu testenden Attributs liegt, so werden alle Instanzen an einen einzigen Kindknoten weitergeleitet und der Test ist in diesem Fall überflüssig. Außerdem lässt sich durch dieses Verfahren ein Entscheidungsbaum sehr schnell aufbauen.



(a) Erste Iteration **(b)** Zweite Iteration

(c) Dritte Iteration **(d)** Vierte Iteration

Abbildung 3.3: Schritte bei der Erstellung eines Entscheidungsbaumes durch den Dice-Lernalgorithmus mit einem Datensatz von zehn Instanzen (rote Punkte)

3.5 Hoeffding-Tree

Im Bereich des Online-Lernens ist der Hoeffding-Tree [Jesse Read, 2010] einer der bekanntesten Lernalgorithmen. Die Motivation für die Entwicklung des Hoeffding-Trees war, dass viele Algorithmen, die für das Online-Lernen eingesetzt werden, sehr langsam arbeiten und dabei noch viel Speicherplatz benötigen. Bei manchen dieser Algorithmen beträgt die Zeit für die Erstellung eines Modells unter der Annahme des Online-Lernens länger, als wenn diese durch das Batch-Lernen erstellt werden. Der Hoeffding-Tree wurde speziell für das Online-Lernen entworfen. Die besonderen Merkmale dieses Algorithmus sind, dass das erstellte Modell sehr schnell mit einer neuen Instanz angepasst werden kann und dass der Speicherplatzbedarf gering gehalten wird. Erreicht wird dies, indem die Anpassung des Modells mit einer Instanz eine konstante Zeit benötigt und während des Lernens keine zusätzlichen Informationen im Speicher abgelegt werden, außer dem erstellten Entscheidungsbaum.

Der Hoeffding-Tree besitzt diesen Namen, weil in den Tests die Hoeffding-Ungleichung verwendet wird. Mit dieser Ungleichung ist es möglich, bereits bei einer geringen Anzahl von Werten, den Erwartungswert einer ganzen Verteilung mit einer gewissen Wahrscheinlichkeit abschätzen zu können. Diese Eigenschaft kann genutzt werden, um einen Test für ein bestimmtes Attribut erstellen zu können. Der inkrementelle Aufbau der Bäume wird hierbei so durchgeführt, dass

in den Blättern Statistiken über die zugeordneten Instanzen geführt werden können. Nachdem einem Blatt eine Instanz übergeben wurde, werden die Statistiken aktualisiert. Anschließend wird überprüft, ob die vorliegenden Informationen ausreichen, um einen Test erstellen zu können. Ist dies der Fall, so wird das Blatt durch einen inneren Knoten mit dem ermittelten Test ersetzt und diesem weitere Blätter hinzugefügt, in denen Statistiken geführt werden können. Die besondere Eigenschaft des Hoeffding-Trees ist, dass unter der Annahme des Online-Lernens ungefähr das gleiche Modell erstellt wird, wie wenn unter der Annahme des Batch-Lernens gearbeitet wird. Allerdings ist dies nur der Fall, wenn dem Lernalgorithmus für die Erstellung des Modelles ausreichend viele Instanzen übergeben werden.

3.6 Random Forest

Bei Random Forest [Svetnik et al., 2003] handelt es sich um einen Lernalgorithmus, der ebenfalls ein Ensemble von Entscheidungsbäumen verwendet. Das Ensemble des Random Forest wird durch Verfahren wie zum Beispiel dem *Bagging* aufgebaut. Die Erstellung eines Entscheidungsbaumes durch den Random Forest ähnelt dabei sehr stark dem Verfahren zur Erstellung eines zufälligen Entscheidungsbaumes. Der Unterschied liegt in der Bestimmung der Trennwerte für die Tests. Bei den zufälligen Entscheidungsbäumen wird für die Erstellung eines Tests per Zufall ein Attribut und ein Trennwert ausgewählt, hingegen beim Random Forest der beste Trennwert mit einer Teilmenge von allen Attribute berechnet wird. Um einen Test mit dem Verfahren von Random Forest erstellen zu können, werden nur die Attribute analysiert, die in der Teilmenge vorhanden sind. Anhand eines Algorithmus kann anschließend das wichtigste Attribut und der zugehörige Trennwert bestimmt werden. Dieses Verfahren eignet sich gut für die Verarbeitung von Daten, die sehr viele Attribute besitzen. Je nach verwendetem Algorithmus lässt sich der Random Forest auch für unterschiedliche Lernaufgaben verwenden.

Normalerweise wird dieser Lernalgorithmus unter der Annahme des Batch-Lernens verwendet, jedoch wird nach [Saffari et al., 2009] ein Verfahren vorgestellt, das auch unter der Annahme des Online-Lernens arbeiten kann. In dieser Arbeit wird die Anpassung des Random Forest an die Bedingungen des Online-Lernens beschrieben. Für die Erstellung des Ensembles wird das *Bagging* imitiert, indem dieselbe Instanz des Datenstroms einem Baum mehrmals übergeben werden kann. Dies verletzt jedoch das Kriterium, dass ein Lernalgorithmus, der unter der Annahme des Online-Lernens arbeitet, jede Instanz nur einmal verarbeiten darf, weswegen auf dieses Verfahren nicht weiter eingegangen wird.

3.7 SMART-Lernalgorithmus

Nach [Kong and Yu, 2011] wird ein Verfahren für die Erstellung eines Ensembles von zufälligen Entscheidungsbäume vorgestellt, das unter der Annahme des Online-Lernens arbeitet. Der Algorithmus wurde speziell für die Multilabel-Klassifikation von Datenströmen entwickelt. Die Besonderheit dieses Verfahrens ist, dass das Modell dieses Lernalgorithmus sich an ein veränderndes Szenario anpassen kann. Ein veränderndes Szenario kann beispielsweise sein, dass Automobile in die Klassen *schnell* und *langsam* klassifiziert werden sollen. Vor 20 Jahren wurden Automobile mit einer Leistung von 120 Pferdestärken als *schnell* klassifiziert, hingegen wird heutzutage ein Automobil mit der gleichen Leistung als *langsam* klassifiziert. Über die Jahre haben sich die Bedingungen für eine Klassifikation in *schnelle* und *langsame* Automobile verändert. Dementsprechend muss es mit dem Lernalgorithmus möglich sein, sich an solche Veränderungen anpassen zu können. In der Arbeit wird diese Anpassung anhand eines sogenannten *fading factors* durchgeführt. Bei der inkrementellen Anpassung der Entscheidungsbäume mit einer Instanz werden die Statistiken in den Blättern angepasst, an die die Instanz zugewiesen wird. Mit dem *fading factor* ist es möglich, dass die neu eintreffende Instanz einen größeren Einfluss auf die Statistiken in den Blattknoten hat, als die Instanzen die vorher gesammelt wurden. Dadurch können die gesammelten Statistiken in den Blättern immer an neue Verhältnisse angepasst werden. Bei diesem Algorithmus werden die Entscheidungsbäume komplett ohne Instanzen aufgebaut. Anschließend werden die Instanzen des Datenstroms den Entscheidungsbäumen übergeben, wodurch die Anpassung stattfindet.

4 Modulares Framework zum Aufbau von zufälligen Entscheidungsbäumen

In diesem Kapitel wird das Konzept für das zu erstellende Programm entworfen. Zunächst werden in Abschnitt 4.1 die Anforderungen erfasst und analysiert, wobei die möglichen Lösungen vorgestellt und diskutiert werden. In den darauffolgenden Abschnitten werden die Lösungen detailliert dargestellt, die in diesem Programm umgesetzt wurden.

4.1 Erfassung und Analyse der Anforderungen an das Programm

Innerhalb dieser Bachelorarbeit soll ein Programm entworfen und implementiert werden, mit dem Ensembles von Entscheidungsbäumen erstellt und verwendet werden können. Hierbei soll der Algorithmus für die Erstellung der Entscheidungsbäume so implementiert werden, dass er gegen beliebige Verfahren ausgetauscht werden kann. Durch diese Austauschbarkeit wird die Möglichkeit geboten, neue Ideen für die Erstellung von Entscheidungsbäumen auszuprobieren und somit deren Tauglichkeit testen zu können. Des Weiteren ist es möglich, auf unterschiedliche Probleme im Bereich des maschinellen Lernens einzugehen und speziell für diese einen Algorithmus für die Erstellung der Entscheidungsbäume zu entwerfen. So ist es beispielsweise denkbar, dass ein Algorithmus implementiert werden kann, der besonders gut für den Umgang mit fehlenden Werten oder spärlichen Daten geeignet ist. In den meisten Fällen wird bei der Implementierung des Algorithmus versucht, diesen möglichst so zu entwickeln, dass er mit einer Vielzahl solcher Probleme umgehen kann und somit in vielen Bereichen einsetzbar ist. Für die Erstellung des Ensembles sollen beliebige Verfahren verwendet werden. Aus diesem Grund wird die Erstellung des Ensembles von dem Algorithmus übernommen. Somit ist es möglich, dass speziell für jeden Algorithmus ein Verfahren für die Generierung des Ensembles implementiert und verwendet werden kann. Das heißt, dass es die Aufgabe des Algorithmus ist, ein Ensemble von Entscheidungsbäumen zu erstellen und dieses dem Programm zu übergeben.

Aus diesem Grund ist die Aufgabe des Programmes, die erstellten Bäume zu verwalten und zu verwenden. Dementsprechend ist es wichtig, dass durch die Algorithmen Entscheidungsbäume erstellt werden, die eine einheitliche Struktur haben, sodass jeder Baum in dem Programm unter den gleichen Bedingungen verwendet werden kann. Zu den Aufgaben des Programmes gehört unter anderem, dass die erstellten Entscheidungsbäume genutzt werden können, um eine Vorhersage für eine Instanz machen zu können. Hierbei muss für die Instanz die Vorhersagen der einzelnen Entscheidungsbäume des Ensembles geholt und kombiniert werden. Eine weitere Anforderung an das Programm ist, dass auch Instanzen mit fehlenden Werten verarbeitet werden sollen. So ist es denkbar, dass eine Vorhersage für eine Instanz mit fehlenden Werten gemacht werden könnte. Hierbei kann der Fall eintreten, dass eine Instanz zu einem Test von einem Entscheidungsbaum gelangt, der auf einem fehlenden Wert der Instanz prüft. In diesem Fall ist es nicht möglich, den zugehörigen Kindknoten, an den die Instanz weitergeleitet werden soll, zu bestimmen. Folglich muss ein Verfahren entwickelt werden, das mit einer solchen Situation umgehen kann.

Eine weitere Anforderung an das zu erstellende System ist, dass es möglich sein soll, die erstellten Entscheidungsbäume für mehrere Lernaufgaben gleichzeitig einsetzen zu können. Das bedeutet, dass der erstellte Entscheidungsbaum in den Blättern nicht nur für ein Lernproblem Vorhersagen machen soll, sondern für beliebig viele Lernprobleme gleichzeitig. So kann beispielsweise mit demselben Entscheidungsbaum mittels der Regression ein numerischer Wert für ein Attribut und mittels der Klassifikation ein nominaler Wert für ein anderes Attribut vorhergesagt werden. Des Weiteren soll es möglich sein, in den Blättern unterschiedliche Informationen der Instanzen speichern zu können. So wurde beispielsweise gewünscht, dass bei einer Multilabel-Klassifikation die zugehörigen Instanzen in einem Blatt abgespeichert werden können. Anhand der Instanzen kann im Nachhinein die Verteilung der Klassen per Hand analysiert werden, um zum Beispiel selten aufgetretene Klassenkombinationen ausfindig zu machen. Aus diesen Gründen müssen die Blätter so konzipiert werden, dass beliebig viele Informationen anhand der Instanz gesammelt werden können.

Für die Implementierung kann auf vorhandene Funktionen von WEKA[1] zurückgegriffen werden. Aus diesem Grund müssen die Funktionalitäten, wie zum Beispiel das Einlesen und die Verwaltung der Instanzen, nicht gesondert entworfen und implementiert werden. Ein weiterer Vorteil der Verwendung ist, dass zahlreiche andere Programme auf der Grundlage von WEKA aufbauen, sodass die erstellten Lernalgorithmen mittels Wrapper-Klassen in beliebige Programme eingefügt und übertragen werden können. So ist zum Beispiel die Kompatibilität zu den Programmen MULAN[2], MEKA[3] und MOA[4] gewährleistet, die alle auf Funktionalitäten von WEKA zurückgreifen. Alle genannten Programme verwenden für die Darstellung und Verwaltung der Instanzen die Klassen von WEKA, wobei minimale Anpassungen durchgeführt werden müssen.

[1] http://www.cs.waikato.ac.nz/ml/weka/ Abgerufen am: 23.03.2015
[2] http://mulan.sourceforge.net/starting.html Abgerufen am: 23.03.2015
[3] http://meka.sourceforge.net/ Abgerufen am: 23.03.2015
[4] http://moa.cms.waikato.ac.nz/ Abgerufen am: 23.03.2015

Zusätzlich soll es mit dem Programm möglich sein, dekrementelles Lernen auf den Entscheidungsbäumen durchführen zu können. Hierbei ist besonders darauf zu achten, dass nicht jeder Entscheidungsbaum für ein dekrementelles Lernen eingesetzt werden kann. Zum Beispiel kann ein Algorithmus, der inkrementell mit Instanzen angepasst wird, über die Zeit des Lernens die Trennwerte existierender Tests verändern. Würden wir nun eine Instanz aus dem Entscheidungsbaum entfernen wollen, könnte nicht mehr gewährleistet werden, dass die Instanz aus demselben Blatt entfernt wird, dem sie beim Lernen hinzugefügt wurde. Aus diesem Grund darf die Funktionalität nur auf Entscheidungsbäumen angewendet werden, die für ein dekrementelles Lernen geeignet sind.

Zusammenfassung der Anforderungen an das Programm

- Austauschbarkeit des Algorithmus für die Erstellung der Entscheidungsbäume
- Verwaltung der Entscheidungsbäume
- Mehrere Lernaufgaben mit einem einzigen Entscheidungsbaum durchführen
- Durchführen von dekrementellen Lernen auf den Entscheidungsbäumen

4.2 Struktur der Entscheidungsbäume

Eine Anforderung an das zu erstellende Programm ist, dass der Algorithmus für die Erstellung der Entscheidungsbäume ausgetauscht werden kann. Damit die unterschiedlich erstellten Entscheidungsbäume in dem Programm verwaltet und verwendet werden können, ist es wichtig, dass alle Bäume auf derselben Struktur aufbauen. Die grundlegende Struktur besteht aus inneren Knoten und Blättern. In Abschnitt 3.2 wurde bereits erläutert, dass sich die unterschiedlichen Arten der Entscheidungsbäume nur durch die verwendeten Tests unterscheiden. Hierbei haben die unterschiedlichen Tests immer dieselbe Funktionsweise. Anhand einer übergebenen Instanz muss ein Kindknoten bestimmt werden, an den die Instanz weitergegeben werden soll. Aus diesem Grund eignet sich eine abstrakte Klasse besonders gut für den Test, weil diese die grundlegende Funktion vorgibt. Aufbauend auf dieser abstrakten Klasse können beliebige Tests implementiert werden.

Des Weiteren wird gefordert, dass mehrere Lernaufgaben gleichzeitig durchgeführt werden sollen. Dies stellt für die normalen Entscheidungsbäume ein Problem dar, weil bei diesen die Blätter verwendet werden, um genau für eine Lernaufgabe eine Vorhersage machen zu können. Gelöst werden kann dieses Problem dadurch, dass in den Blättern beliebig viele sogenannte Kollektoren verwaltet werden können. Ein Kollektor repräsentiert hierbei genau eine Lernaufgabe. Bei der Erstellung des Entscheidungsbaum für zwei unterschiedliche Lernaufgaben, müssen jedem Blatt des Entscheidungsbaumes die Kollektoren dieser Lernaufgaben hinzugefügt werden. Hierbei ist wichtig, dass jedes Blatt des Entscheidungsbaumes für jede gewünschte Lernaufgabe genau einen Kollektor enthält. Wird nun die Vorhersage für eine der beiden Lernaufgaben verlangt, so kann die Instanz über den Wurzelknoten einem Blatt zugewiesen werden. Anschließend wird der entsprechende Kollektor des Blattes für diese Lernaufgabe ausgewählt, durch welchen die Vorhersage für diese Instanz erstellt werden kann. In Abbildung 4.1 ist dargestellt, wie n Kollektoren in den Blättern verwaltet werden. Die Verwendung von Kollektoren bietet diverse weitere Anwendungsmöglichkeiten, auf welche in dem folgenden Abschnitt eingegangen wird.

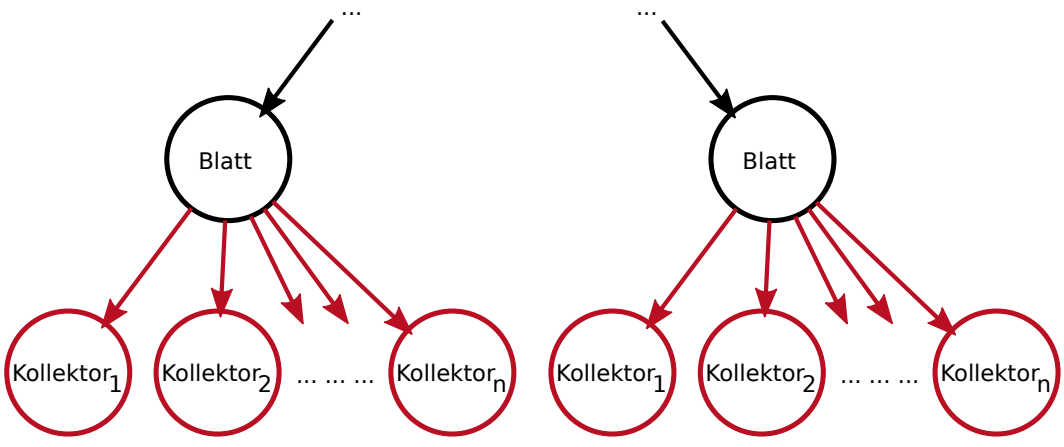


Abbildung 4.1: Verwendung von Kollektoren in Blättern

4.3 Kollektoren

Wie schon im vorherigen Abschnitt erwähnt, werden in den Blättern eines Baumes Kollektoren verwaltet, wobei durch jeden Kollektor eine bestimmte Lernaufgabe übernommen wird. So kann beispielsweise ein Kollektor verwendet werden, der eine binäre Klassifikation vornimmt. Die Aufgabe dieses Kollektors ist es, den entsprechenden Wert eines nominalen Attributes für eine gegebene Instanz vorhersagen zu können. Damit die unterschiedlichsten Kollektoren entwickelt und verwendet werden können, ist es notwendig für diese eine gewisse Struktur festzulegen. Damit mit einem Kollektor eine Vorhersage gemacht werden kann, müssen diesem zuerst Informationen übergeben werden. Die Idee ist, dass bei der Erstellung des Entscheidungsbaumes den Kollektoren Instanzen übergeben werden können, die bereits mit den zu vorhersagenden Klassen annotiert sind. Anhand der übergebenen Instanzen kann der Kollektor die gewünschten Informationen für seine Lernaufgabe extrahieren und abspeichern. Mit den gesammelten Informationen kann eine Vorhersage generiert werden. Ein Kollektor, der eine binäre Klassifikation durchführt, kann beispielsweise die Information speichern, wie oft welcher Wert des vorherzusagenden Attributs in den übergebenen Instanzen gesehen wurde. Um mit diesem Kollektor eine Vorhersage generieren zu können, wird die Klasse vorhergesagt, die am häufigsten gesehen wurde. Der Vorteil dieses Verfahren ist, dass durchgehend Informationen in den Kollektoren gesammelt werden können und zu jedem Zeitpunkt eine Vorhersage anhand der momentan gesammelten Informationen gemacht werden kann. Aus diesem Grund eignen sich die Kollektoren besonders gut für ein inkrementelles Lernen. Zu beachten ist aber auch, dass Kollektoren, denen keine Instanz übergeben wurde, keine gültige Vorhersage machen können. Kollektoren können aber auch für andere Aufgaben eingesetzt werden. So besteht zum Beispiel die Möglichkeit, einen Kollektor zu entwickeln, der die übergebenen Instanzen abspeichert, um diese im Nachhinein analysieren zu können.

Des Weiteren werden die Entscheidungsbäume als Ensemble eingesetzt, wobei das Zusammenfügen der Vorhersagen der einzelnen Entscheidungsbäume eine wichtige Rolle spielt. Idealerweise eignen sich Kollektoren gut, um die unterschiedlichen Vorhersagen der Bäume kombinieren zu können. In den Kollektoren kann auf alle gesammelten Informationen zugegriffen und diese für die Kombination berücksichtigt werden. Aus diesem Grund werden durch die Entscheidungsbäume keine Werte vorhergesagt, sondern die Kollektoren des Blattes, denen die Instanz zugeordnet wurde, werden zurückgegeben. Um beispielsweise eine Vorhersage für eine Instanz mit einem Ensemble von Entscheidungsbäumen machen zu können, wird die Instanz jedem Entscheidungsbaum übergeben. In dem jeweiligen Entscheidungsbaum wird die Instanz an ein Blatt weitergeleitet, von dem die Kollektoren zurückgegeben werden. Anschließend werden die Kollektoren der einzelnen Entscheidungsbäume miteinander kombiniert. Außerdem bietet die Kombination der Kollektoren den Vorteil, dass speziell für jeden Kollektor ein Verfahren entwickelt werden kann, mit der die Vorhersagen kombiniert werden können.

Grundsätzlich wird zwischen zwei Arten von Kombinationen unterschieden. Zum einen gibt es das *Vereinen* der Kollektoren. Bei dieser Kombination werden die Kollektoren so zusammengefügt, dass jeder Kollektor gleich gewichtet ist. Hierbei hat ein Kollektor, dem eine Instanz übergeben wurde das gleiche Gewicht, wie ein Kollektor dem 100 Instanzen übergeben wurden. Eine solche Kombination wird beispielsweise verwendet, um die Kollektoren der Vorhersagen der einzelnen Entscheidungsbäume im Ensemble kombinieren zu können. Zum anderen gibt es das *Addieren* von Kollektoren. Bei dem Addieren von Kollektoren werden die Kollektoren nicht gewichtet zusammengefügt. Das heißt, dass in diesem Fall die gesammelten Informationen der zu kombinierenden Kollektoren addiert werden. Addiert man beispielsweise einen Kollektor, der auf einer Instanz basiert, mit einem Kollektor der auf 100 Instanzen basiert, so erhält man einen Kollektor, der auch durch die Übergabe dieser 101 Instanzen entstehen würde. Eine Addition von Kollektoren wird beispielsweise eingesetzt, um zwei Blätter zu einem Blatt zusammenfassen zu können.

Um die Funktionalität des dekrementellen Lernens zu unterstützen, muss die Möglichkeit bestehen, Instanzen von Kollektoren wieder entfernen zu können. Hierbei wird die zu entfernende Instanz den Kollektoren übergeben. Anhand dieser Instanz können die Statistiken der Kollektors angepasst werden, sodass diese Instanz bei einer Vorhersage des Kollektors nicht mehr involviert ist. Bei einem Kollektor, der beispielsweise die Anzahl eines bestimmten Wertes für ein Attribut speichert, kann die Anzahl um eins verringert werden. Außerdem besteht die Möglichkeit, in den Kollektoren spezielle Funktionalitäten zu implementieren, sodass zum Beispiel speziell auf gewichtete Instanzen reagiert werden kann. Auch die im Abschnitt 3.7 vorgestellte Variante des *fading factors* kann in den Kollektoren implementiert werden. Für die Evaluation in Kapitel 6 werden zwei Kollektoren verwendet, welche in den folgenden Abschnitten vorgestellt werden.

4.3.1 Kollektor für die Multiklassen-Klassifikation

Ein Kollektor, der für eine Multiklassen-Klassifikation eingesetzt wird, muss den Wert eines nominalen Attributes vorhersagen. Aus diesem Grund werden für das zu vorhersagende Attribut die Vorkommnisse der Klassen abgespeichert. Muss durch einen solchen Kollektor eine Vorhersage gemacht werden, so wird immer die Klasse vorhergesagt, die am häufigsten gesehen wurde. Beim Vereinen von diesen Kollektoren werden ihre gesammelten Werte vor der Kombination

bezüglich der Anzahl der gesehenen Instanzen normiert. Bei einer Addition von Kollektoren wird nur die Anzahl der gesehenen Werte der unterschiedlichen Kollektoren addiert.

4.3.2 Kollektor für die Multilabel-Klassifikation

Der Kollektor, der eine Multilabel-Klassifikation durchführen kann, muss die Werte mehrerer nominaler Attribute vorhersagen. Hierbei wird für jede zu vorhersagende Klasse ein Zähler verwendet, der die Anzahl der gesehenen Klassen abspeichert. Zusätzlich wird noch abgespeichert, wie viele Instanzen dem Kollektor übergeben wurden. Anhand dieser Daten kann der Kollektor eine Vorhersage machen, wobei zuerst bestimmt wird, wie viele Klassen vorhergesagt werden sollen. Hierfür addiert man die Werte aller Zähler der einzelnen Klassen und dividiert diese durch die Anzahl der gesehenen Instanzen. Enthält das Ergebnis eine Zahl mit Nachkommastellen, so muss diese auf eine ganze Zahl gerundet werden. Anschließend werden die Klassen vorhergesagt, die den größten Wert in ihrem Zähler haben. Auch hier werden bei dem Vereinen von Kollektoren die Werte vorher mit der Anzahl der gesehenen Klassen normiert. Bei einer Addition werden die gesammelten Werte nur addiert.

Dieses Verfahren stellt jedoch nicht die einzige Möglichkeit dar, mit der eine Vorhersage bei einer Multilabel-Klassifikation gemacht werden kann. Eine andere Variante ist das *Thresholding*. Auch bei diesem Verfahren wird für jede vorherzusagende Klasse ein Zähler verwaltet, der das Auftreten der jeweiligen Klasse zählt. Für die Erstellung einer Vorhersage wird jeder Zähler durch die Anzahl der gesehenen Instanzen geteilt, um die Relevanz der Klassen zu bestimmen. Es werden alle Klassen vorhergesagt, deren Relevanz größer ist als ein bestimmter Trennwert. Dieser Trennwert kann beliebig gewählt werden. Wählen wir beispielsweise einen Trennwert von 50%, so werden alle Klassen vorhergesagt, die eine Relevanz von mindestens 50% haben. Für die Evaluation auf den Multilabel-Klassifikations-Datensätze in Kapitel 6 wird die erste vorgestellte Variante des Kollektors für die Multilabel-Klassifikation verwendet.

4.4 Erstellung der Entscheidungsbäume

Bei der Erstellung der Entscheidungsbäume soll darauf geachtet werden, dass beliebige Verfahren verwendet werden können. Das bedeutet, dass das Programm eine Schnittstelle bereitstellen muss, an der beliebige Algorithmen für die Erstellung der Entscheidungsbäume verwendet werden können. Die erstellten Entscheidungsbäume müssen hierbei auf der vorgestellten Struktur basieren. Es gehört nicht zu den Aufgaben des Programmes das Ensemble zu erstellen. Diese Funktionalität wird dem Algorithmus für die Erstellung der Entscheidungsbäume überlassen. Dies bietet die Möglichkeit, für jeden Algorithmus spezifisch ein Verfahren für die Erstellung des Ensembles zu implementieren. In Kapitel 5 wird auf zwei Verfahren für die Erstellung von Entscheidungsbäumen eingegangen, die innerhalb dieser Bachelorarbeit entwickelt wurden.

4.5 Instanzen dem Entscheidungsbaum hinzufügen und entfernen

Ein Vorteil der einheitlichen Struktur der Entscheidungsbäume und der Verwendung von Kollektoren ist, dass die Bäume zu jedem Zeitpunkt mit weiteren Instanzen trainiert werden können. Hierbei spielt es keine Rolle, mit welchem Algorithmus die Bäume erstellt wurden, weil die Struktur der Bäume durch dieses Hinzufügen nicht verändert wird. Die zu hinzufügenden Instanzen werden jedem Baum übergeben, wodurch sie den Blättern zugeordnet werden. In den Blättern werden die ankommenden Instanzen wiederum den Kollektoren hinzugefügt. Für Algorithmen, die inkrementell lernen und ihre Struktur mit jeder ankommenden Instanz anpassen, wird das Verfahren für die Erstellung der Entscheidungsbäume aufgerufen, um den Baum zu erweitern beziehungsweise anzupassen.

Aus den gleichen Gründen ist es möglich, dass ein dekrementelles Lernen durchgeführt werden kann. Hierbei kann eine Instanz, die für die Erstellung oder durch nachträgliches Training hinzugefügt wurde, wieder aus dem Entscheidungsbaum entfernt werden. Bei diesem Verfahren wird die zu entfernende Instanz dem Wurzelknoten übergeben, wodurch sie einem Blatt zugeteilt wird. Durch die Funktionalität der Kollektoren, dass Instanzen wieder herausgenommen werden können, ist es möglich, dass die entsprechende Instanz von den Kollektoren des Blattes wieder entfernt werden kann. Auch in diesem Fall wird die Baumstruktur nicht verändert. Voraussetzung für das dekrementelle Lernen ist, dass durch das Trainieren die Trennwerte in den Tests nicht verändert werden. Gehen wir dagegen von einer Änderung der Trennwerte in den Tests aus, so kann nicht gewährleistet werden, dass die Instanz aus demselben Kollektor entfernt werden kann, zu welchem sie hinzugefügt wurde.

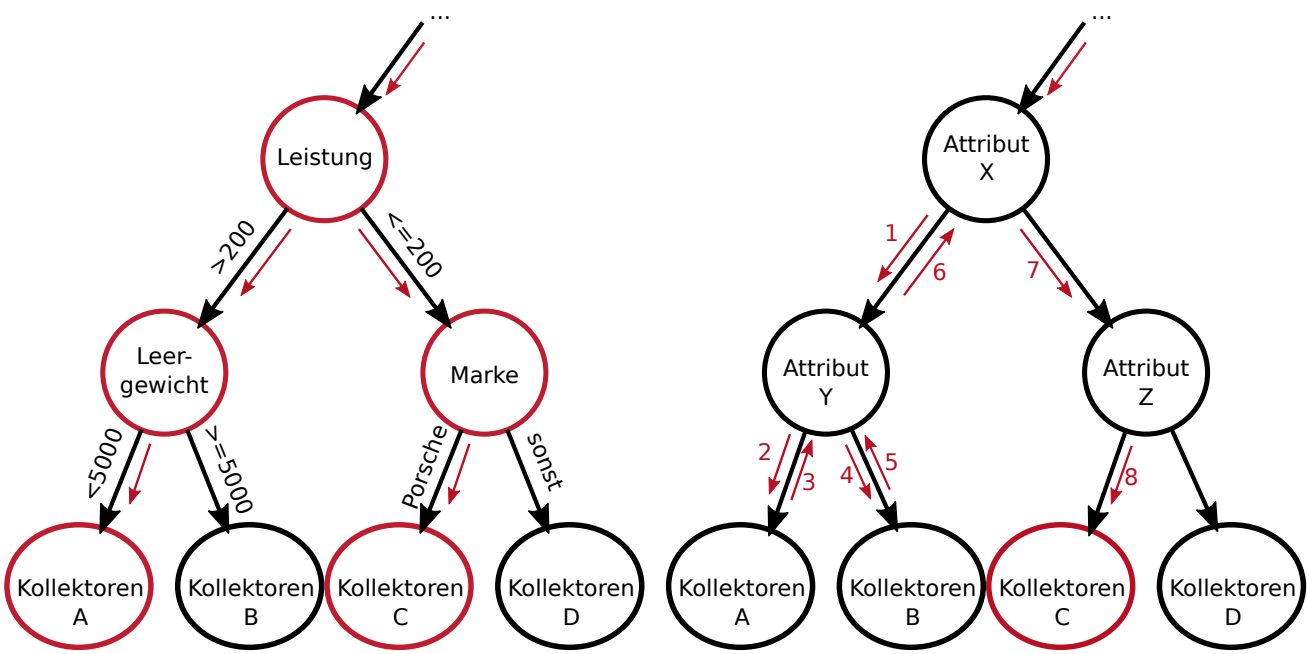
4.6 Vorhersage durch einen Entscheidungsbaum

Aufgrund der Verwendung von Kollektoren werden die Vorhersagen in einem Entscheidungsbaum auf eine spezielle Art und Weise generiert. In den Blättern des Entscheidungsbaumes werden keine endgültigen Werte vorhergesagt, sondern

die Kollektoren des Blattes zurückgegeben. Dies bietet den Vorteil, dass bei einer späteren Kombination der Kollektoren auf die gesammelten Werte der Kollektoren zugegriffen werden kann. In einem Blatt können mehrere Kollektoren enthalten sein, die für unterschiedliche Aufgaben eingesetzt werden können. Bei der Kombination von Kollektoren unterschiedlicher Blätter werden nur die Kollektoren miteinander kombiniert, die dieselbe Aufgabe haben. Verwenden wir beispielsweise in einem Entscheidungsbaum die Kollektoren K_1 und K_2, so enthält jedes Blatt in dem Entscheidungsbaum einen Kollektor vom Typ K_1 und einen Kollektor vom Typ K_2. Müssen nun die Kollektoren der Vorhersagen zweier unterschiedlicher Entscheidungsbäume kombiniert werden, so werden nur die Kollektoren desselben Typs miteinander kombiniert. Das bedeutet, dass der Kollektor K_1 des einen Baumes mit dem Kollektor K_1 des anderen Baumes kombiniert wird. Dasselbe wird für den Kollektor K_2 durchgeführt. Um den Wert für ein zu vorhersagendes Attribut erhalten zu können, muss anschließend nur der entsprechende Kollektor ausgewählt und die Vorhersage generiert werden.

4.6.1 Umgang mit fehlenden Werten

Unter der Annahme, dass der Algorithmus für die Erstellung der Entscheidungsbäume beliebig ausgetauscht werden kann, ist es nötig, auf unterschiedliche Sachverhalte beim Ablauf einer Vorhersage zu achten. Eine Anforderung an das Programm ist, dass auch Algorithmen für die Erstellung der Entscheidungsbäume unterstützt werden sollen, die mit fehlenden Werten umgehen können. Das bedeutet auch, dass für Instanzen mit fehlenden Werten Vorhersagen auf der einheitlichen Struktur gemacht werden sollen. Hierbei kann bei einer Vorhersage der Fall eintreten, dass bei der Zuordnung einer Instanz zu einem Blatt, ein Test auf einem fehlenden Wert der Instanz prüft. In diesem Fall kann die Instanz nicht eindeutig einem Kindknoten zugewiesen werden. Um mit diesem Problem umgehen zu können, wird die Instanz an alle Kindknoten weitergegeben. In den Kindknoten kann der Ablauf der Vorhersage normal weitergeführt werden. Anschließend erhalten wir für jeden Kindknoten die Kollektoren, die für die Vorhersage verwendet werden sollen. Diese Kollektoren werden daraufhin miteinander vereint. Bei einem Test, der ein fehlenden Wert überprüft, werden quasi die Vorhersagen aller Kindknoten kombiniert.



(a) Vorhersage für eine Instanz, deren Wert für die Leistung unbekannt ist

(b) Vorhersage für eine Instanz, wobei die *Kollektoren A* und die *Kollektoren B* keine gültige Vorhersage machen können

Abbildung 4.2: Vorhersage durch Entscheidungsbäume

In Abbildung 4.2a wird beispielhaft eine Vorhersage für eine Instanz mit dem *Leergewicht* von 1119 Kilogramm und der *Marke* Porsche dargestellt, deren Wert für die *Leistung* unbekannt ist. Hierbei wird die Instanz vom inneren Knoten, der die *Leistung* überprüft, an beide Kindknoten weitergeleitet. In den Kindknoten kann das Verfahren für die Vorhersage normal fortgeführt werden. So erhalten wir durch den einen Kindknoten die *Kollektoren A* und durch den anderen Kindknoten die *Kollektoren C*. Die Kollektoren der beiden Kindknoten werden anschließend vereint.

4.6.2 Umgang mit leeren Blättern

Außerdem ist bei der Vorhersage zu beachten, dass der Entscheidungsbaum unter Umständen Blätter enthalten kann, deren Kollektoren keine gültigen Vorhersagen machen können. Dies wäre zum Beispiel der Fall, wenn bei der Erstellung der Bäume ein Blatt erzeugt wurde, dem aber keine Instanzen zugeordnet wurden, welche den Kollektoren übergeben werden konnten. Des Weiteren ist es möglich, dass durch das dekrementelle Lernen so viele Instanzen von einem Blatt entfernt werden, dass die Kollektoren dieses Blattes keine Instanzen mehr enthalten. Die Folge ist, dass keine Informationen verfügbar sind, um eine Vorhersage machen zu können. Um mit diesem Problem umgehen zu können, wird ein Verfahren verwendet, mit dem es möglich ist, das nächstliegende Blatt zu finden, deren Kollektoren eine gültige Vorhersage machen können.

Gelangt man zu einem Blatt, das keine gültige Vorhersage machen kann, so wird mit dem Verfahren in dem Baum um einen Knoten zurückgegangen. Von diesem Knoten aus wird in allen anderen Kindknoten nach einer gültigen Vorhersage gesucht. Werden hierbei eine oder mehrere gültige Vorhersagen gefunden, so können diese vereint und als Vorhersage zurückgegeben werden. Möglich ist aber auch, dass alle anderen Kindknoten ebenfalls keine gültigen Vorhersagen machen können. In diesem Fall wird bei diesem Verfahren wieder auf einen vorhergehenden Knoten in der Baumstruktur zurückgegangen, von wo aus in allen anderen Kindknoten nach gültigen Vorhersagen gesucht wird. Eine Wiederholung dieser Prozedur erfolgt solange, bis eine gültige Vorhersage gefunden worden ist. Enthält ein Baum keine gültigen Vorhersagen, so kann keine Vorhersage gemacht werden. Da das Verfahren bei sehr vielen Blättern, die keine gültige Vorhersage machen können, sehr rechenaufwendig ist, sollte bei der Implementierung des Algorithmus zur Erstellung der Entscheidungsbäume vermieden werden, viele Blätter mit leeren Kollektoren zu erstellen.

In Abbildung 4.2b wird ein Beispiel vorgestellt bei dem die *Kollektoren A* und *Kollektoren B* keine gültigen Vorhersage machen können. Die *Kollektoren C* und *Kollektoren D* können gültige Vorhersagen machen. Tritt nun der Fall ein, dass eine Instanz über den ersten und den zweiten inneren Knoten an den linken Kindknoten weitergeleitet wird, jeweils gekennzeichnet durch die roten Pfeile mit den Ziffern 1 und 2, so gelangt die Instanz zu einem Blatt, welches keine gültige Vorhersage machen kann. Dementsprechend wird ein Schritt in der Baumstruktur zurückgegangen, dargestellt durch den roten Pfeil mit der Ziffer 3. Von diesem Knoten aus wird in allen anderen Kindknoten nach einer gültigen Vorhersage gesucht. Da der Knoten aber nur zwei Kindknoten enthält und in dem linken Kindknoten keine gültige Vorhersage gemacht werden konnte, wird im nächsten Schritt im rechten Kindknoten nach einer gültigen Vorhersage gesucht, gekennzeichnet durch den roten Pfeil mit der Ziffer 4. Da wieder keine gültige Vorhersage gefunden werden konnte und der innere Knoten, der das Attribut *Y* überprüft, keine gültige Vorhersage machen kann, wird über die Pfeile mit den Ziffern 5 und 6 nochmals ein Schritt im Baum zurückgegangen. Von diesem Knoten aus, der das Attribut *X* überprüft, wird diese Prozedur wiederholt. In diesem Fall wird die Instanz an den Kindknoten übergeben, dessen Test das Attribut *Z* überprüft, dargestellt durch den roten Pfeil mit der Ziffer 7. Über den Test wird die Instanz von dort aus an den linken Kindknoten weitergeleitet, gekennzeichnet durch den roten Pfeil mit der Ziffer 8. Nun sind wir zu einem Blatt gelangt, dass eine gültige Vorhersage machen kann. Aus diesem Grund werden für die Vorhersage der Instanz die *Kollektoren C* verwendet.

4.7 Verwendung der Entscheidungsbäume als Ensemble

Eine weitere Anforderung an das Programm ist, dass die erstellten Entscheidungsbäume auch als Ensemble verwendet und verwaltet werden sollen. In einem der vorherigen Abschnitten wurde bereits beschrieben, dass es möglich ist, die Vorhersagen von einzelnen Entscheidungsbäumen unter der Verwendung von Kollektoren kombinieren zu können. Die Aufgabe des Programmes ist in diesem Fall nur, eine Funktion bereitzustellen, mit der die Vorhersagen der einzelnen Bäume geholt und kombiniert werden können. Auch die anderen Funktionalitäten, wie zum Beispiel das dekrementelle Lernen, müssen in dem Programm so verwendet werden, dass diese auf allen Entscheidungsbäumen des Ensembles angewendet werden können. Die Aufgabe des Programmes beschränkt sich bei der Verwendung von Ensembles nur auf die Verwaltung und ihre Verwendung.

5 Trainieren von zufälligen Entscheidungsbäumen

In diesem Kapitel wird auf zwei Verfahren für die Erstellung von Entscheidungsbäumen eingegangen, die im Rahmen dieser Bachelorarbeit entwickelt und implementiert werden. Als Grundlage beider Algorithmen dient das in dem vorherigen Kapitel vorgestellte Framework. Der erste Algorithmus soll unter der Annahme des Batch-Lernens arbeiten und wird in Abschnitt 5.1 vorgestellt. Der zweite Algorithmus soll unter der Annahme des Online-Lernens entworfen und implementiert werden und wird in Abschnitt 5.2 vorgestellt.

5.1 Batch-Lernen von zufälligen Entscheidungsbäumen

In diesem Abschnitt wird der Algorithmus für die Erstellung der Entscheidungsbäume beschrieben, der unter der Annahme des Batch-Lernens arbeitet. Hierbei wird sich an der Funktionsweise des Dice-Lernalgorithmus orientiert. In Abschnitt 5.1.1 werden die Anforderungen erfasst und analysiert und in den darauffolgenden Abschnitten wird der entworfene Algorithmus vorgestellt. Abschließend wird in Abschnitt 5.1.4 auf eine schnelle Evaluierung eingegangen, die speziell für diesen Algorithmus entworfen wird.

5.1.1 Erfassung und Analyse der Anforderungen

Um sich mit der Materie der zufälligen Entscheidungsbäume vertraut machen zu können, soll der erste Algorithmus unter der Annahme des Batch-Lernens entworfen und implementiert werden. Der Dice-Lernalgorithmus soll bei der Entwicklung dieses Algorithmus als Orientierung dienen. Hierbei ist nicht verlangt, dass die Funktionalitäten des Dice-Lernalgorithmus exakt nachgebaut werden sollen, sondern es besteht die Möglichkeit, eigene Ideen in die Entwicklung einfließen zu lassen. Anhand des erstellten Algorithmus soll ein Verfahren entwickelt werden, mit dem es möglich ist, die Evaluation durch eine Kreuzvalidierung beschleunigen zu können. Ein besonderes Augenmerk soll hierbei auf der LOO-Kreuzvalidierung liegen, weil diese am meisten Rechenaufwand benötigt. Eine schnelle Evaluierung kann beispielsweise nützlich sein um die optimalen Parametereinstellungen für ein Lernverfahren finden zu können. In diesem Fall kann die schnelle Evaluierung eingesetzt werden um unterschiedliche Parametereinstellungen schneller testen und deren Tauglichkeit bestimmt zu können.

5.1.2 Aufbau der Entscheidungsbäume

Zuerst widmen wir uns dem Aufbau eines einzigen Entscheidungsbaumes. Für die Erstellung eines Entscheidungsbaumes werden alle Instanzen des Datensatzes verwendet. Um den ersten Knoten des Baumes erstellen zu können, wird zufällig ein Attribut des Datensatzes ausgewählt. Handelt es sich bei dem zufälligen Attribut um ein numerisches Attribut, so wird zusätzlich noch per Zufall eine Instanz aus der Menge der verfügbaren Instanzen ausgewählt. Mit der ausgewählten Instanz wird der Trennwert des Tests bestimmt, indem der Wert des zufälligen Attributes von dieser Instanz genommen wird. Anhand des Trennwertes können die Instanzen auf zwei Kindknoten aufgeteilt werden. Handelt es sich bei dem zufällig ausgewähltem Attribut um ein nominales Attribut, so wird ein Test erstellt, der für jeden Wert der Wertemenge des Attributes einen Kindknoten enthält. In diesem Fall wird auf beliebig viele Kindknoten aufgeteilt. Nachdem der erste Test für den Wurzelknoten erstellt wurde, werden alle Instanzen bezüglich des Tests auf die Kindknoten aufgeteilt. In den Kindknoten wird die Prozedur mit den zugewiesenen Instanzen solange wiederholt, bis die maximale Höhe des Baumes erreicht ist oder zu wenige Instanzen für die Bestimmung eines Tests vorhanden sind. Tritt einer dieser beiden Fälle ein, so wird anstatt eines inneren Knoten mit einem Test, ein Blatt erstellt. Die Instanzen die zu diesem Blatt gelangt sind, werden anschließend den Kollektoren des Blattes übergeben. Bei der zufälligen Auswahl eines Attributes für einen Test wird außerdem noch berücksichtigt, dass nominale Tests auf dem gleichen Attribut nicht mehrmals auf einem Pfad durchgeführt werden. Der Grund hierfür ist, dass bei der zweiten Selektion der Instanzen auf dem gleichen nominalen Attribut, alle Instanzen nur noch genau einem Wert dieses Attributes angehören und nur einem Kindknoten weitergegeben werden könnten.

Der Aufbau eines Entscheidungsbaumes ist noch einmal durch den Pseudocode von Algorithmus 2 und 3 abgebildet. Bei dem Algorithmus 2 handelt es sich um ein rekursives Verfahren, dass mit allen Instanzen des Datensatzes und der Höhe 0 angestoßen werden muss. Als Rückgabe erhält man den Wurzelknoten des erstellten Entscheidungsbaumes. Für den Aufbau des Ensembles von Entscheidungsbäumen werden vier Parameter benötigt. Über die Parameter lassen sich die maximale Höhe der Bäume, die Anzahl der Bäume im Ensemble, der Seed für den Zufallsgenerator und die minimale Anzahl an Instanzen für die Erstellung eines Tests einstellen. Für die Erstellung des Ensembles wird kein spezielles Verfahren verwendet, was bedeutet, dass jeder Entscheidungsbaum unter der Verwendung des ganzen Datensatzes unabhängig

Algorithmus 1 $buildTrees(insts)$

Eingangsvariablen:

$insts$: Die Instanzen, die für den Aufbau genutzt werden sollen

Ablauf:

for Jeden Baum b_i in dem Ensemble **do**
    Erstelle b_i mit $buildNodes(insts, 0)$
end for

Algorithmus 2 $buildNodes(insts, height)$

Eingangsvariablen:

$insts$: Die Instanzen, die für den momentan zu erstellenden Knoten genutzt werden können
$height$: Die momentane Höhe des zu erstellenden Knotens

Ablauf:

if Die maximale Höhe des Baumes ist erreicht oder es liegen nicht genügend Instanzen vor **then**
    Erstelle ein neues Blatt $leaf$ und füge den Kollektoren dieses Blattes die Instanzen $insts$ hinzu
    return $leaf$
else
    $height++$;
    Erstelle ein neuen inneren Knoten $node$ mit $buildNode(insts)$
    for Für jeden Kindknoten k_i von $node$ **do**
        Bestimme die Instanzen $insts_i$ von $insts$, die durch den Test von $node$ dem Kindknoten k_i zugeordnet werden
        Erstelle k_i mit $buildNodes(insts_i, height)$
        Füge k_i dem Knoten $node$ als Kindknoten hinzu
    end for
    return $node$
end if

Rückgabewert:

$node$: Der momentan erstellte Knoten

Algorithmus 3 $buildNode(insts)$

Eingangsvariablen:

$insts$: Die Instanzen, die für die Erstellung des Knotens genutzt werden können

Ablauf:

Bestimme das zu testende Attribut a
if a ist nominal **then**
    Erzeuge Test $test$ auf nominalen Attribut a
else
    Wähle zufällig eine Instanz aus $insts$ aus und nehme den Wert $value$ des Attributes a
    Erzeuge Test $test$ auf numerischen Attribut a mit Trennwert $value$
end if
Erstelle inneren Knoten $node$ mit $test$
return $node$

Rückgabewert:

$node$: Der erstellte Knoten

aufgebaut wird. Dies bietet den Vorteil, dass sich die Erstellung der Entscheidungsbäume sehr gut parallelisieren lässt. Der Algorithmus für die Erstellung des Ensembles wird in Algorithmus 1 dargestellt.

Viele Eigenschaften des Dice-Lernalgorithmus wurden für den Aufbau dieser Entscheidungsbäume übernommen. Der entworfene Algorithmus unterscheidet sich von dem Dice-Lernalgorithmus hauptsächlich in der zufälligen Auswahl des Trennwertes für die numerischen Tests. Bei dem Dice-Lernalgorithmus werden zusätzliche Operationen auf den verfügbaren Instanzen durchgeführt. So werden beispielsweise alle Werte der Instanzen für das zufällig zu testende Attribut untersucht und doppelte Werte gelöscht. Nehmen wir an, dass vier Instanzen für die Erstellung eines Tests vorhanden sind, die für das zu prüfende numerische Attribut die Werte {*10, 10, 10, 15*} haben. Mit dem Dice-Algorithmus werden vor der zufälligen Auswahl des Trennwertes aus dieser Menge zuerst die doppelten Werte eliminiert. So erhält man nur noch die folgenden Werte {*10, 15*}. Aus den resultierenden Werten wird anschließend ein zufälliger Wert ausgewählt. In dem entworfenen Algorithmus wird diese Selektion nicht durchgeführt, wodurch zwar die Güte der Vorhersagen sinken

kann, jedoch die Rechenzeit für die Erstellung eines solchen Tests deutlich minimiert werden kann. In manchen Fällen ist es sogar von Vorteil, dass Werte als Trennwert genommen werden, die oft in den Daten auftreten.

5.1.3 Umgang mit fehlenden Werten

Eine weitere Anforderung an diesen Algorithmus ist, dass auch Datensätze mit fehlenden Werten verarbeitet werden sollen. Um mit einen solchen Szenario umgehen zu können, müssen bei dem Aufbau der Entscheidungsbäume zwei mögliche Fälle berücksichtigt werden. So kann es bei der Erstellung eines numerischen Tests vorkommen, dass der zufällig ausgewählte Trennwert ein fehlender Wert ist. In diesem Fall kann der Test nicht funktionieren, weil die Instanzen nicht separiert werden können. Deswegen muss bei der zufälligen Auswahl des Trennwertes darauf geachtet werden, dass der Trennwert kein fehlender Wert ist. Wird festgestellt, dass der zufällig ausgewählte Wert ein fehlender Wert ist, so wird der Wert einer anderen Instanz genommen. Enthalten alle Instanzen für das zu testende Attribut einen fehlenden Wert, so kann kein Test für dieses numerische Attribut erstellt werden. Tritt dieser Fall ein, so wird der Baum an dieser Stelle nicht fortgeführt und es erfolgt die Erstellung eines Blattes, dem alle zugehörigen Instanzen übergeben werden.

Des Weiteren kann bei der Aufteilung der Instanzen auf die Kindknoten auf einem fehlenden Wert getestet werden. In diesem Fall kann die Instanz nicht eindeutig einem Kindknoten zugewiesen werden. Um mit diesem Problem umgehen zu können, werden die Instanzen, bei denen auf einem fehlenden Wert getestet wird, allen Kindknoten übergeben. Hierbei wird das Gewicht der Instanz bezüglich der Anzahl der Kindknoten angepasst. Wird beispielsweise auf einem fehlenden Wert einer Instanz getestet und der Knoten mit dem Test enthält vier Kindknoten, so wird das Gewicht der Instanz geviertelt und jedem Kindknoten wird die Instanz übergeben. In Abbildung 5.1 ist ein solcher Fall abgebildet.

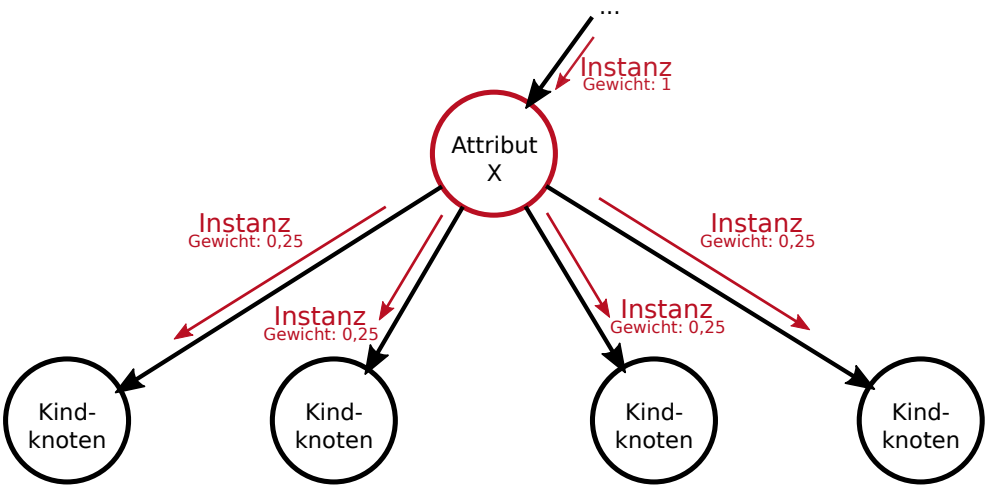


Abbildung 5.1: Aufteilung einer Instanz auf alle Kindknoten

5.1.4 Spezielle Leave-One-Out-Kreuzvalidierung

Um eine schnelle Evaluierung durch eine Kreuzvalidierung entwickeln zu können, müssen die Kriterien für eine Kreuzvalidierung untersucht werden. Wie schon in Abschnitt 2.11.2 erklärt, wird der Datensatz bei einer Kreuzvalidierung in n-Teile aufgeteilt. Für die Evaluierung eines jeden Teils des Datensatzes muss durch den Lernalgorithmus ein Modell erstellt werden. Je größer der Parameter n gewählt wird, desto mehr Modelle müssen erstellt werden, weshalb viel Zeit beansprucht wird. Die Idee ist, ein Modell mit allen Instanzen des Datensatzes zu erstellen und durch dekrementelles Lernen, die zu evaluierenden wieder Instanzen aus dem Modell entfernen zu können. Wird beispielsweise eine Kreuzvalidierung mit vier Teilen durchgeführt, so wird das Modell auf allen vier Teilen erstellt. Um in der ersten Iteration den ersten Teil evaluieren zu können, müssen die Instanzen dieses Teils des Datensatzes aus dem Modell entfernt werden. Anschließend können die entfernten Instanzen genutzt werden, um das Modell zu testen. Damit die anderen Teile des Datensatzes auch getestet werden können, müssen die entfernten Instanzen nach der Evaluierung wieder dem Modell hinzugefügt werden. Nach diesem Prinzip können die anderen Iterationen der vierfachen Kreuzvalidierung durchgeführt werden. Durch dieses Verfahren muss für die Kreuzvalidierung nur ein Modell erstellt werden und es ist sichergestellt, dass auf Instanzen evaluiert wird, die nicht für die Erstellung des Modells verwendet wurden.

Die Idee dieses Verfahrens soll auf den, in den vorherigen Abschnitten dargestellten, Algorithmus angewendet werden. Das Problem hierbei ist, dass mit dem Algorithmus kein striktes dekrementelles Lernen durchgeführt werden kann. Das bedeutet, dass die Instanzen aus den Blättern entfernt, jedoch die Tests in den Entscheidungsbäumen nicht angepasst

werden können. Der Grund liegt in der Auswahl des Trennwertes für einen Test auf einem numerischen Attribut. Bei dieser Auswahl wird per Zufall eine Instanz bestimmt, dessen Wert für den Test verwendet wird. Das heißt, dass die zufällig ausgewählte Instanz für die Erstellung des Modells genutzt wurde. Folglich muss ein Verfahren entwickelt werden, mit dem es möglich ist, dieses Problem zu umgehen.

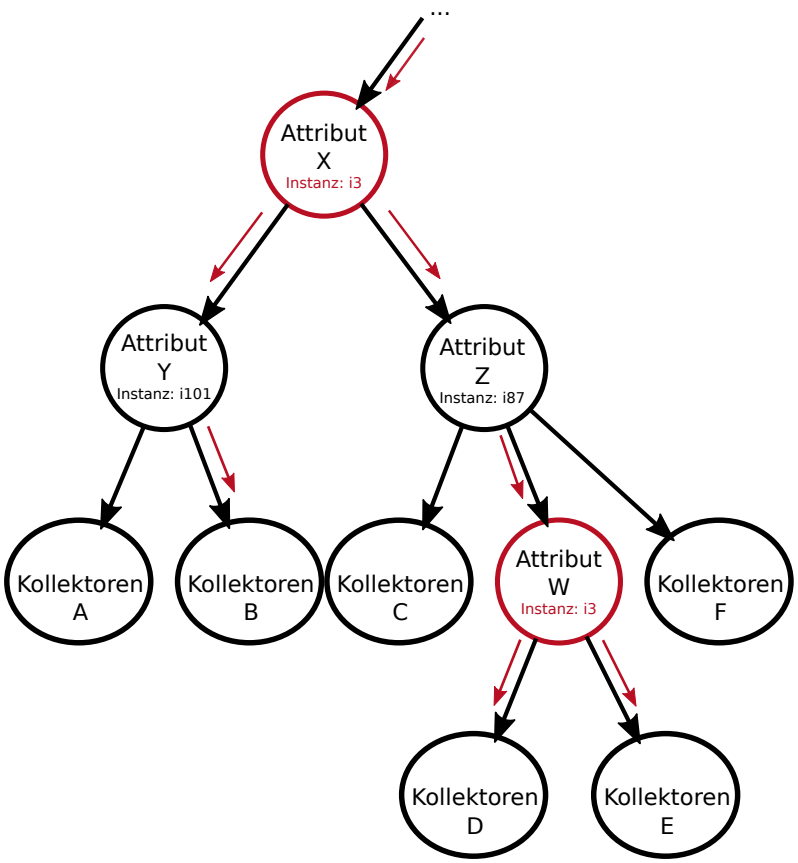


Abbildung 5.2: Überspringen der inneren Knoten, die mit der Instanz *i3* erstellt wurden

Die Lösung für dieses Problem besteht darin, dass bei der Erstellung der Tests auf numerischen Attributen zusätzlich die zufällig ausgewählte Instanz in dem Test mit abgespeichert wird. Durch die Speicherung der Instanz ist es möglich, bei der späteren Verwendung des Entscheidungsbaumes zu erkennen, welche Instanz für die Erstellung des Tests genutzt wurde. Tritt nun der Fall ein, dass ein Test bei einer Vorhersage einer Instanz nicht genutzt werden darf, so kann dies erkannt und der Test übersprungen werden. Um einen Test überspringen zu können, wird die Instanz nicht mit dem Test überprüft, sondern direkt allen Kindknoten weitergegeben. Die Kollektoren der Vorhersagen der einzelnen Kindknoten werden anschließend addiert. Durch die Addition wird gewährleistet, dass keine zusätzliche Information durch die Aufteilung auf die Kindknoten gewonnen werden kann. In Abbildung 5.2 wird beispielhaft eine Vorhersage dargestellt, bei der die Instanz *i3* nicht genutzt werden darf. Hierbei werden die Tests der inneren Knoten, die die Attribute *X* und *Y* überprüfen, übersprungen. Aus diesem Grund werden für die Vorhersage die Kollektoren *B*, *D* und *E* ausgewählt, welche anschließend addiert werden, um eine Vorhersage generieren zu können.

Mit dem Überspringen der Tests und dem nicht strikten dekrementellen Lernen des Algorithmus ist es nun möglich, eine schnelle Kreuzvalidierung durchführen zu können. Problematisch hierbei ist, je mehr Instanzen aus dem Modell entfernt werden, desto mehr Tests müssen in einem Entscheidungsbaum übersprungen werden. Durch das Überspringen der Tests wird die Anzahl der Tests auf einem Pfad verringert, mit dem Instanzen separiert werden können, wodurch in vielen Fällen die Güte des Entscheidungsbaumes sinken kann. Dementsprechend ist es sinnvoll so wenige Tests wie möglich überspringen zu müssen. Aus diesem Grund ist die LOO-Kreuzvalidierung für dieses Verfahren am besten geeignet. Bei dieser Kreuzvalidierung muss für die Evaluierung einer Instanz in jeder Iteration genau eine Instanz aus dem Entscheidungsbaum entfernt werden. Das bedeutet, dass nur die Tests, die mit der zu evaluierenden Instanz erstellt wurden, übersprungen werden müssen. Zusätzlich spielt die Anzahl der Instanzen, die für die Erstellung des Entscheidungsbaums verwendet werden eine wichtige Rolle. Verwendet man beispielsweise 1.000 Instanzen für die Erstellung eines Entscheidungsbaumes mit der maximalen Höhe von zehn, so ist die Wahrscheinlichkeit, dass eine spezielle Instanz in einem oder mehreren Tests benutzt wird höher, als wenn der Baum mit 10.000 Instanzen aufgebaut wird. Ein weiterer Vorteil ist, dass nominale Tests nicht übersprungen werden müssen, weil für diese bei der Erstellung des Tests nicht explizit eine In-

stanz verwendet wird. Tritt beispielsweise der Fall ein, dass nur für die zu evaluierende Instanz bei einem nominalen Test ein gesonderter Kindknoten angelegt wird, so enthält dieser Teilbaum nur diese Instanz. Vor der Evaluierung wird diese Instanz aus dem Entscheidungsbaum entfernt, wodurch an dieser Stelle ein leerer Teilbaum entsteht. Bei der Vorhersage dieser Instanz, wird diese an diesen leeren Teilbaum weitergeleitet, wodurch nach dem Verfahren aus Abschnitt keine gültige Vorhersage gefunden werden kann und in dem Entscheidungsbaum zurückgegangen werden muss. In diesem Fall kann zusätzliche Information durch diesen Kindknoten erreicht werden, weshalb alleine durch das dekrementelle Lernen gewährleistet werden kann, dass diese Instanz nicht bei der Erstellung des Modells involviert ist. Aus diesem Grund eignet sich die spezielle LOO-Kreuzvalidierung gut für Datensätze, die viele nominale Attribute verwenden. Bei Datensätzen mit vielen numerischen Attributen kann es wahrscheinlich zu Differenzen zwischen den Ergebnissen der speziellen und der normalen LOO-Kreuzvalidierung kommen. Des Weiteren spielt die maximale Tiefe der zu evaluierenden Entscheidungsbäume eine wichtige Rolle. Je tiefer ein Baum gebaut wird, desto mehr Tests müssen verwendet werden. Unter der Annahme, dass keine fehlenden Werte verwendet werden, kann eine Instanz maximal in zehn Tests verwendet werden, wenn der Baum eine maximale Höhe von zehn hat.

Algorithmus 4 stellt den Ablauf der speziellen LOO-Kreuzvalidierung dar. Ein weiterer Vorteil der speziellen LOO-Kreuzvalidierung ist, dass nach der Evaluierung direkt ein fertiges Modell vorliegt, dass mit allen Instanzen des Datensatzes erstellt wurde.

Algorithmus 4 $specialCV(insts)$

Eingangsvariablen:

$insts$: Die Instanzen, die für Kreuzvalidierung benutzt werden

Ablauf:

1: Erstelle mit allen Instanze $insts$ das Ensemble von Entscheidungsbäume mit dem Algorithmus
2: **for** Jede Instanz $inst_i$ von $insts$ **do**
3: Entferne $inst_i$ mit dekrementellen Lernen aus den Entscheidungsbäumen des Ensembles
4: Benutze das Ensemble um eine Vorhersage für $inst_i$ zu machen, wobei die Tests, die mit $inst_i$ erstellt wurden, übersprungen werden
5: Überprüfe die Vorhersage mit dem tatsächlichem Wert für $inst_i$
6: Füge $inst_i$ wieder den Entscheidungsbäumen des Ensembles hinzu
7: **end for**

5.2 Online-Lernen von zufälligen Entscheidungsbäumen

In diesem Abschnitt wird der Algorithmus für die Erstellung der Entscheidungsbäume vorgestellt, der für das Online-Lernen eingesetzt werden kann. Das Hauptmerkmal dieses Algorithmus besteht darin, dass für die Tests auf numerischen Attributen Quantile verwendet werden. In Abschnitt 5.2.1 wird erklärt, was Quantile sind und wie diese verwendet werden können. In dem darauffolgenden Abschnitt 5.2.2 werden die Anforderungen erfasst und analysiert und in den anschließenden Abschnitten wird der entworfene Algorithmus vorgestellt.

5.2.1 Quantile

Durch ein Quantil wird ein Trennwert für eine Menge von numerischen Werten angegeben, mit dem die Menge in zwei Teilmengen aufgeteilt werden kann. Hierbei werden alle Werte der Menge, die kleiner als der Trennwert sind, der ersten Teilmenge und der Rest der zweiten Teilmenge zugeordnet. Die Besonderheit des Quantils ist, dass die Menge prozentual zu der Anzahl der Werte aufgeteilt wird. In vielen Fällen spricht man von einem p-Quantil, wobei p einen Wert zwischen 0 und 1 annehmen kann. Mit dem p wird ausgedrückt, wie viel Prozent der Werte in der ersten Teilmenge enthalten sein sollen. Die restlichen Werte werden der zweiten Teilmenge zugeordnet. Zum Beispiel kann durch das 0,3-Quantil eine Menge von Werten in eine Teilmenge mit 30% und in eine Teilmenge mit 70% der Werte aufgeteilt werden. Hierbei spricht man auch von einer Diskretisierung eines numerische Attributes, weil jeder Wert genau einer von den zwei Teilmengen zugeordnet wird. Das bekannteste Quantil ist das 0,5-Quantil, welches auch als Median bezeichnet wird. Die Berechnung des Trennwertes eines Quantils wird durch die Formel 5.1 ausgedrückt [Falk et al., 2014, S. 16f.]. Hierbei wird von einer geordneten Menge von Werten $x_1, x_2, \ldots, x_n$ ausgegangen, wobei durch n die Anzahl der Werte in der Menge angegeben ist.

$$Trennwert = \begin{cases} \frac{1}{2}(x_{n \cdot p} + x_{(n \cdot p+1)}), & \text{falls } n \cdot p \in \mathbb{N}, \\ x_{\lceil n \cdot p \rceil}, & \text{falls } n \cdot p \notin \mathbb{N} \end{cases} \tag{5.1}$$

$$A = \{3, 7, 9, 10, 12, 15, 100, 100\} \tag{5.2}$$

Nehmen wir an, wir haben eine geordnete Menge von Werten, die durch die Formel 5.2 dargestellt ist. Wollen wir nun den Trennwert für das 0,5-Quantil berechnen, so erhalten wir durch die Formel 5.1 den Trennwert 11. Für das 0,3-Quantil wird der Trennwert 9 berechnet.

5.2.2 Erfassung und Analyse der Anforderungen

Im Rahmen der Bachelorarbeit soll ein Algorithmus für die Erstellung von Entscheidungsbäumen entwickelt werden, der unter der Annahme des Online-Lernens arbeitet. Auch dieser Algorithmus soll mit fehlenden Werten umgehen können. Des Weiteren sollen für die Bestimmung der Trennwerte in den numerischen Tests Quantile verwendet werden. Hierbei soll der Wert p des p-Quantils für jeden numerischen Test zufällig bestimmt werden. Wie schon in Abschnitt 5.2.1 beschrieben, ist für die Berechnung des Trennwertes eine Menge von Werten vonnöten. Problematisch ist, dass bei dem Online-Lernen vor dem Aufbau der Entscheidungsbäume keine Informationen über die Werte der Attribute vorliegen. Die Informationen über die Werte von numerischen Attributen können erst bei dem inkrementellen Aufbau der Entscheidungsbäume erfasst werden. Dementsprechend können die Trennwerte, die in den Tests verwendet werden sollen, nicht direkt bestimmt werden.

Nach [Webb, 2014] wird eine Möglichkeit vorgestellt, mit der eine Diskretisierung von numerischen Attributen unter der Verwendung von Quantilen vorgenommen werden kann. In dieser Arbeit wird besonders auf die Verwendung der Diskretisierung von inkrementellen Lernverfahren eingegangen. Hierbei stellte sich heraus, dass die Verwendung von Quantilen eine effiziente Methode für die Diskretisierung von numersichen Attributen darstellt. Das in der Arbeit vorgestellte Verfahren arbeitet ebenfalls unter der Annahme des Online-Lernens. Die Lösung des Problems besteht darin, die Werte für das ausgewählte numerische Attribut während des inkrementellen Aufbaus zu sammeln. Wird ein neuer Wert der gesammelten Menge eines Attributes hinzugefügt, so wird der Trennwert anhand der aktualisierten Menge neu bestimmt. Der Vorteil dieses Verfahren ist, dass für die Bestimmung des Trennwertes nicht erst alle Werte der Menge gesammelt werden müssen sondern inkrementell hinzugefügt werden können. Der Nachteil dieses Verfahrens besteht darin, dass, durch die ständige Neuberechnung, der Trennwert sich oft verändert. Besonders, wenn noch nicht sehr viele Werte für ein Attribut gesammelt wurden, kann sich der neu berechnete Trennwert deutlich von dem alten Trennwert unterscheiden. Je mehr Werte der Menge hinzugefügt werden, desto genauer kann der exakte Trennwert für die Verteilung eines numerischen Attributes bestimmt werden. Aus diesem Grund eignet sich das Sammeln der Werte besonders gut für das Online-Lernen, weil hier ein unendlicher Datenstrom an Instanzen vorliegen kann.

Durch den permanenten Datenstrom entsteht jedoch noch ein weiteres Problem, denn die gesammelten Werte für das entsprechende numerische Attribut müssen gespeichert werden. Werden 1000 Instanzen verarbeitet, so müssen 1000 Werte für die Bestimmung des Quantils eines Attributes gespeichert werden. Dementsprechend müssen bei einem unendlichen Datenstrom unendlich viele Werte gespeichert werden. Dies widerspricht jedoch dem Kriterium für das Online-Lernen, das aussagt, dass der Speicherplatzbedarf möglichst gering gehalten werden soll. Eine Lösung dieses Problems wird ebenfalls in [Webb, 2014] vorgestellt. Hierbei kann über einen Parameter die maximale Anzahl an Werten eingestellt werden, die für ein Attribut gesammelt werden dürfen. Dies hat den Grund, dass oft schon mit einer geringen Anzahl von Werten der Trennwert gut bestimmt werden kann. Nachdem die maximale Anzahl von Instanzen gesammelt wurde, werden keine weiteren Werte zusätzlich gespeichert, so dass nur eine geringe Anzahl von Werten verwaltet werden muss.

In Abschnitt 3.7 wurde vorgestellt, dass es Lernalgorithmen gibt, die sich an ein veränderndes Szenario anpassen können. Nach [Webb, 2014] können hierfür auch Quantile eingesetzt werden. Nehmen wir beispielsweise an, dass Personen in die Klassen *geringes Einkommen* und *hohes Einkommen* klassifiziert werden sollen. In diesem Fall kann das geringe Einkommen dadurch definiert sein, dass 50% der Personen mit dem geringsten *Gehalt* der Klasse *geringes Einkommen* und der andere Teil der Personen der Klasse *hohes Einkommen* zugeordnet werden. Für eine solche Klassifizierung eignet sich ein 0,5-Quantil für das Attribut *Gehalt* gut, weil hiermit ein Trennwert gefunden werden kann, mit dem die Personen genau in zwei Mengen mit jeweils 50% der Personen aufgeteilt werden können. Problematisch wird es, wenn sich über die Zeit das Verhältnis ändert und der alte bestimmte Trennwert nicht mehr gültig ist. Zum Beispiel kann durch Inflation das *Gehalt* aller Personen über die Zeit stetig steigen. Aufgrund dieser Steigerung kommt es zu einer Verschiebung des Trennwertes und damit zu einer anderen Klassifikation in *geringes Einkommen* und *hohes Einkommen*.

Um sich an ein solch veränderndes Szenario anpassen zu können, werden die gesammelten Werte für ein Attribut ständig durch neue Werte ausgetauscht. Ist die maximale Anzahl an Instanzen schon gesammelt, so wird bei jeder neu ankommenden Instanz per Zufall ein Wert aus den gesammelten Werten gelöscht. Anschließend wird der Wert der Instanz für das entsprechende Attribut den gesammelten Werten hinzugefügt. Durch dieses Verfahren werden die gesammelten Werte immer aktuell gehalten, anhand derer die Trennwerte für die Quantile bestimmt werden können. In Algorithmus 5 ist eine vereinfachte Form des Verfahrens für die Erstellung der Mengen aus [Webb, 2014] abgebildet.

Algorithmus 5 $updateSamples(inst)$

Globale Variablen:

$values$: Ein Array, das für jedes Attribut eine Menge von Werten speichert

$maxValues$: Die maximale Anzahl an Werten, die für ein Attribut gespeichert werden dürfen

Eingangsvariablen:

$inst$: Die Instanz, die hinzugefügt werden soll

Ablauf:

```
1: for Jedes Attribut a_i do
2:     if Der Wert value der Instanz für das Attribut a_i ist kein fehlender Wert then
3:         if Die Anzahl der Werte in values_i entspricht maxValues then
4:             Entferne zufällig einen Wert aus values_i
5:         end if
6:         Füge value der Menge values_i hinzu
7:     end if
8: end for
```

Des Weiteren wird gefordert, dass die durch den Online-Algorithmus erstellten Entscheidungsbäume aggregierbar sein sollen. Das bedeutet, dass mehrere Entscheidungsbäume, die dieselben Attribute und Kollektoren verwenden, zu einem einzigen Entscheidungsbaum zusammengefasst werden können. Dies bietet den Vorteil, dass mehrere Ensembles von Entscheidungsbäumen auf unterschiedlichen Prozessorkernen erstellt und anschließend zu einem Ensemble zusammengefasst werden können. Will man beispielsweise ein Ensemble mit 10000 Instanzen erstellen, so lässt sich die Erstellung der Entscheidungsbäume auf fünf Prozessorkerne aufteilen, die jeweils 2000 Instanzen für die Erstellung des Ensembles verwenden. Als Resultat erhält man fünf Ensembles von Entscheidungsbäumen, die durch eine Aggregation zu einem Ensemble zusammengefügt werden können. Durch diese Parallelisierung kann der Aufbau der Entscheidungsbäume beschleunigt werden.

Ein weiterer Vorteil der Aggregation ist, dass diese bei einer Kreuzvalidierung verwendet werden kann. Normalerweise haben die durch die Kreuzvalidierung erstellten Modelle keine Verwendung, weil sie nur mit einem Teil der Daten eines Datensatzes erstellt wurden. In vielen Fällen will man jedoch nach der Kreuzvalidierung ein Modell erhalten, dass mit allen Instanzen des Datensatzes aufgebaut wurde. Mit einer Aggregation ist es beispielsweise möglich, die erstellten Modelle für jeden Teil der Kreuzvalidierung zusammenzufassen und somit ein Modell zu erzeugen, dass alle Instanzen des Datensatzes enthält.

Nach [Andrzejak et al., 2013] wird ein Verfahren vorgestellt, mit dem es möglich ist, Entscheidungsbäume zu aggregieren. Hierbei können unterschiedlich aufgebaute Entscheidungsbäume zu einem Baum zusammengefasst werden. Bei diesem Verfahren werden die Vorhersagen der Blätter der zu aggregierenden Entscheidungsbäume als Entscheidungsregionen auf einen n-dimensionalen Raum abgebildet, wobei der Parameter n die Anzahl der Attribute angibt. Für eine Aggregierung zweier Entscheidungsbäume werden die Entscheidungsregionen beider Entscheidungsbäume übereinander gelegt und anhand dieser neue Regionen mit Vorhersagen bestimmt. Mit den Regionen, die durch die Kombination entstanden sind, wird anschließend wieder ein Entscheidungsbaum erstellt. Das Problem dieses Verfahrens ist, dass die Güte des aggregierten Entscheidungsbaumes deutlich sinkt, je mehr Entscheidungsbäume miteinander aggregiert werden. Aus diesem Grund eignet sich dieses Verfahren nicht für unseren Algorithmus.

Um trotzdem eine Aggregation von Entscheidungsbäumen durchführen zu können, muss nach einer alternativen Lösung gesucht werden. Die Idee ist, schon bei dem Aufbau der Entscheidungsbäume darauf zu achten, dass die Bäume im Nachhinein ohne große Probleme aggregiert werden können. Die Aggregation von zwei Entscheidungsbäumen wird beispielsweise deutlich vereinfacht, wenn beide Entscheidungsbäume die gleichen Knoten und die gleichen Tests an denselben Stellen im Baum verwenden. In diesem Fall müssen nur die Trennwerte in den Tests der inneren Knoten und die Vorhersagen in den Blättern angepasst werden.

5.2.3 Aufbau eines Entscheidungsbaumes

Um einen Entscheidungsbaum erstellen zu können, müssen zuerst die zu verwendenden Tests entworfen werden. Für die Tests auf numerischen Attributen sollen Quantile benutzt werden. Wie schon in den Anforderungen in Abschnitt 5.2.2 beschrieben, müssen für die Bestimmung des Trennwerts durch ein Quantil die Werte des entsprechenden Attributes gesammelt werden.

Das Sammeln der Werte kann durch zwei unterschiedliche Verfahren erfolgen. Zum einen können die Werte der numerischen Attribute global gesammelt werden. Bei jeder Übergabe einer Instanz an das Ensemble werden die Wertemengen der numerischen Attribute aktualisiert und der Trennwert für die Tests kann neu bestimmt werden. Der Vorteil dieses Verfahrens ist, dass für jedes numerische Attribut nur eine Wertemenge verwaltet werden muss. Zum anderen kann in

jedem Test, der ein numerisches Attribut überprüft, eine Wertemenge verwaltet werden. Dieses Verfahren ist besonders geeignet, weil der Trennwert nur mit den Instanzen erstellt wird, die zu diesem Test gelangt sind. Diese Instanzen werden dann bezüglich des p-Quantils aufgeteilt, wodurch sichergestellt werden kann, dass p% der ankommenden Instanzen an den linken Kindknoten und der Rest an den rechten Kindknoten weitergeleitet werden kann. Durch die erste Variante, kann ein solches Verhalten nicht immer gewährleistet werden. Es kann beispielsweise der Fall eintreten, dass die Instanzen, die mit einem Test separiert werden sollen, nur einem bestimmten Wertebereich des zu überprüfenden Attributes angehören. Problematisch wird es, wenn durch die globale Bestimmung des Trennwertes ein Trennwert berechnet wird, der außerhalb dieses Wertebereichs liegt. In diesem Fall werden alle Instanzen nur an einen Kindknoten weitergeleitet und es können keine nützlichen Information durch diesen Test gewonnen werden. Aus diesem Grund wird die zweite Variante verwendet, obwohl ein erhöhter Speicherbedarf erforderlich ist.

Folglich muss in jedem Test, der ein numerisches Attribut überprüft, eine Menge von Werten verwaltet werden. Zusätzlich werden in dem Test noch die Variablen verwaltet, die angeben, wie viele Werte maximal gespeichert werden dürfen, welches Attribut überprüft und welcher Wert für p beim Quantil verwendet werden soll. Mit diesen Variablen ist es nun möglich, den Trennwert für diesen Test zu bestimmen. Damit der Test auch für das Online-Lernen einsetzbar ist, muss der Test bei dem inkrementellen Aufbau der Entscheidungsbäume aktualisiert werden. Dies geschieht immer dann, wenn eine neue Instanz zu dem Test gelangt. Bevor die Instanz an einen Kindknoten weitergeleitet wird, wird die Instanz dem Test übergeben, damit dieser die gesammelten Werte aktualisieren und den Trennwert neu berechnen kann. In Algorithmus 6 wird die Methode vorgestellt, mit der ein numerischer Test aktualisiert werden kann. Über eine weitere Variable ist es möglich einzustellen, ob sich die Werte sich an ein veränderndes Szenario anpassen sollen.

Algorithmus 6 $addInstance(inst)$

Globale Variablen:

$values$: Die Menge der gespeicherten Werte
$maxValues$: Die maximale Anzahl an Werten, die gespeichert werden dürfen
a : Das Attribut, für das die Werte gesammelt werden
$quantil$: Der p-Wert für das Quantil
$split$: Der Trennwert
$conceptDrift$: Boolische Variable mit dem angegeben werden kann, ob sich der Test an ein veränderndes Szenario anpassen soll

Eingangsvariablen:

$inst$: Die Instanz, die hinzugefügt werden soll

Ablauf:

```
1: if Enthält values weniger als maxValues Werte then
2:     if Der Wert value der Instanz inst für das Attribut a ist kein fehlender Wert then
3:         Füge value der Menge values hinzu
4:         Berechne den Trennwert split neu
5:     end if
6: else
7:     if conceptDrift then
8:         Entferne zufällig einen Wert aus values
9:         Füge den Wert value der Instanz inst für das Attribut a der Menge values hinzu
10:        Berechne den Trennwert split neu
11:    end if
12: end if
```

Für die Tests, die nominale Attribute überprüfen, wird ein ähnliches Verfahren benutzt wie es auch bei dem vorgestellten Algorithmus, der unter der Annahme des Batch-Lernens arbeitet, verwendet wird. Problematisch ist, dass unter der Annahme des Online-Lernens, nicht im Voraus angegeben ist, wie viele Werte die Wertemenge eines nominalen Attributes enthält. Aus diesem Grund muss ein Test entwickelt werden, der ebenfalls mit jeder neu ankommenden Instanz aktualisiert werden kann. Bei der Aktualisierung wird überprüft, ob für den Wert ein Kindknoten existiert. Wenn dies der Fall ist, wird die Instanz an diesen Knoten weitergeleitet. Ist dies nicht der Fall, so wird für diesen Wert ein neuer Kindknoten erstellt, an den die Instanz weitergeleitet werden kann.

Damit die erstellten Ensembles von Entscheidungsbäume mit wenig Aufwand aggregiert werden können, wurde in Abschnitt 5.2.2 bereits erwähnt, dass die Entscheidungsbäume die gleichen Tests verwenden sollen. Das heißt, dass der Aufbau der Entscheidungsbäume, die aggregiert werden sollen, identisch sein muss. Nur die Instanzen, die für das Training der Entscheidungsbäume genutzt werden, können unterschiedlich sein. Um denselben Aufbau von Entscheidungsbäumen zu gewährleisten, werden die Attribute, die in den Tests überprüft werden sollen, vorher per Zufall bestimmt. Hierbei wird für jede Höhe des Entscheidungsbaumes ein Attribut festgelegt. Entscheidungsbäume, die mit

demselben Seed für den Zufallsgenerator erstellt werden, verwenden auf jeder Höhe des Baumes dieselben Attribute. Aus diesem Grund müssen die Attribute vor der Erstellung der Entscheidungsbäume ausgewählt werden. Wird ein numerisches Attribut für eine Höhe eines Entscheidungsbaumes ausgewählt, so wird zusätzlich noch ein Wert p per Zufall zwischen 0 und 1 generiert, welcher für das Quantil verwendet werden soll. Durch Algorithmus 7 wird die Initialisierung der Entscheidungsbäume abgebildet. Hierbei werden auch die Parameter, die für den Aufbau der Entscheidungsbäume notwendig sind, übergeben.

Algorithmus 7 Initialisierung

Eingangsvariablen:

$numTrees$: Die Anzahl der Bäume
$maxHeight$: Die maximale Höhe der Bäume
$maxValues$: Die maximale Anzahl an Werten, die bei den numerischen Tests gespeichert werden
$randomSeed$: Seed für den Zufallszahlengenerator

Ablauf:

1: **for** Jeden Baum b_i in dem Ensemble **do**
2: Bestimme für b_i die Attribute, die für jede Höhe in den Tests überprüft werden sollen
3: Erstelle für jeden Baum b_i ein Blatt als Wurzelknoten
4: **end for**

Um die Aggregation von Entscheidungsbäumen weiter zu vereinfachen, werden bei dem Aufbau der Bäume die Instanzen nur den Blättern hinzugefügt, die sich auf der maximalen Tiefe des Baumes befinden. Wird beispielsweise ein Entscheidungsbaum mit einer Instanz aktualisiert, so wird diese bis zur maximalen Tiefe des Baumes weitergereicht. Tritt der Fall ein, dass die Instanz zu einem Blatt gelangt, dass sich nicht auf der maximalen Tiefe des Baumes befindet, so wird dieses Blatt in einen inneren Knoten umgewandelt. Der Test des Knotens wird dabei mit dem vorher definierten Attribut für diese Höhe erstellt. Dem neu erstellten inneren Knoten wird anschließend für alle Kindknoten ein Blatt hinzugefügt. In den Kindknoten kann die Prozedur solange wiederholt werden, bis die maximale Tiefe des Baumes erreicht ist. Durch dieses Verfahren können trotzdem Blätter entstehen, die sich nicht auf der maximalen Tiefe befinden. Die Kollektoren dieser Blätter können jedoch keine gültigen Vorhersagen machen, weil ihnen keine Instanzen zugewiesen wurden. Aus diesem Grund müssen diese Blätter bei einer Aggregation nicht berücksichtigt werden. Auf das genaue Verfahren der Aggregation wird in Abschnitt 5.2.5 eingegangen.

Das Aktualisieren eines Entscheidungsbaumes mit einer Instanz wird in Algorithmus 8 dargestellt. Hierbei handelt es sich um einen rekursiven Algorithmus, der mit dem Wurzelknoten, der Instanz und der Höhe 0 angestoßen werden muss.

Algorithmus 8 $buildWithInstance(node, inst, height)$

Eingangsvariablen:

$node$: Der momentan betrachtete Knoten
$inst$: Die Instanz
$height$: Die momentane Höhe im Baum

Ablauf:

1: **if** $node$ ist ein innerer Knoten **then**
2: Aktualisiere den Knoten $node$ mit $updateInnerNode(node, inst)$
3: Bestimme mit dem Test von $node$ den Kindknoten k an den die Instanz übergeben werden soll
4: $height++$;
5: Übergebe die Instanz an den Kindknoten mit $buildWithInstance(k, inst, height)$
6: **else** ($node$ ist ein Blatt)
7: **if** Die maximale Höhe des Baumes ist erreicht **then**
8: Füge den Kollektoren des Blattes $node$ die Instanz $inst$ hinzu
9: **else**
10: Wandel das Blatt $node$ in einen inneren Knoten um
11: Erstelle einen neuen Test für $node$
12: Aktualisiere $node$ mit $updateInnerNode(node, inst)$
13: Bestimme mit dem Test von $node$ den Kindknoten k an den die Instanz übergeben werden soll
14: $height++$;
15: Übergebe die Instanz an den Kindknoten mit $buildWithInstance(k, inst, height)$
16: **end if**
17: **end if**

Algorithmus 9 $updateInnerNode(node, inst)$

Eingangsvariablen:

$node$: Der momentan betrachtete Knoten
$inst$: Die Instanz
$height$: Die momentane Höhe im Baum

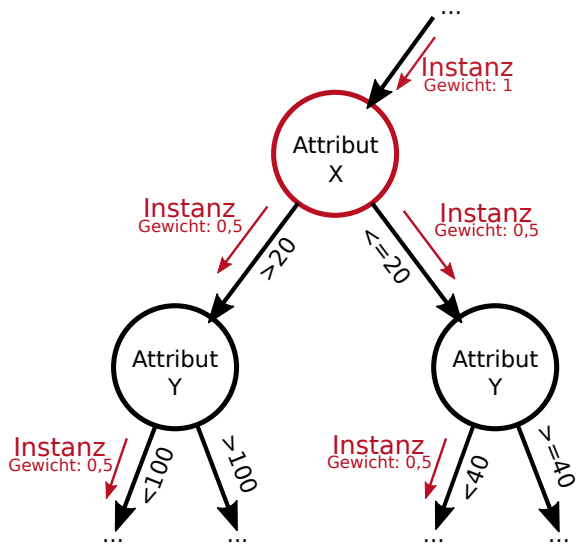
Ablauf:

1: Aktualisiere den Test von $node$ mit $inst$
2: **if** Durch die Aktualisierung des Test können an ein neuen Kindknoten Instanzen weitergeleitet werden **then**
3: Füge $node$ ein Blatt hinzu, dass den neuen Kindknoten repräsentiert
4: **end if**

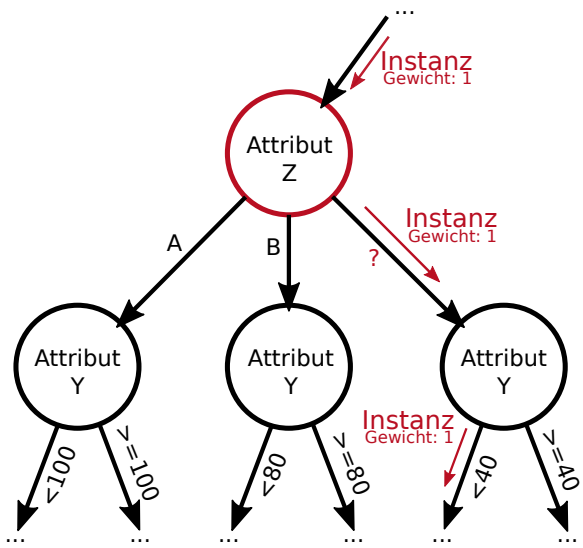
5.2.4 Umgang mit fehlenden Werten

Des Weiteren wird gefordert, dass der Algorithmus mit fehlenden Werten umgehen soll. Dementsprechend muss beim Aufbau eines Entscheidungsbaumes darauf geachtet werden, dass fehlende Werte verarbeitet werden können. Bei dem Aktualisieren eines Entscheidungsbaumes wird die Instanz über die Tests der inneren Knoten einem Blatt zugewiesen. So kann es zum Beispiel vorkommen, dass ein Test einen fehlenden Wert einer Instanz überprüft. Wird ein fehlender Wert mit einem numerischen Test überprüft, so wird die Instanz an alle Kindknoten weitergegeben. Dies geschieht, weil die Anzahl der Kindknoten durch die Aktualisierung des numerischen Tests nicht verändert werden kann. Bei der Aufteilung wird das Gewicht der Instanz bezüglich der Anzahl der Kindknoten angepasst. Hat beispielsweise ein numerischer Test zwei Kindknoten, so wird das Gewicht der Instanz halbiert und diese wird beiden Kindknoten übergeben. Bei diesem Verfahren handelt es sich um dasselbe Verfahren, welches für den Umgang mit fehlenden Werten bei dem Batch-Algorithmus verwendet wird und in Abbildung 5.1 dargestellt ist. In Abbildung 5.3a ist zur Veranschaulichung hier noch einmal die Aufteilung einer Instanz auf zwei Kindknoten abgebildet. Des Weiteren darf bei der Aktualisierung des numerischen Tests der fehlende Wert nicht den gesammelten Werten hinzugefügt werden.

Unter der Annahme des Online-Lernens ist das Verfahren für den Umgang mit fehlenden Werten, wie es bei den numerischen Tests durchgeführt wird, nicht auf nominale Tests anwendbar. Durch die Aktualisierung eines nominalen Tests kann sich die Anzahl der Kindknoten verändern. Folglich kann eine Instanz mit einem fehlenden Wert nicht an alle Kindknoten weitergegeben werden, wenn noch nicht bekannt ist, wie viele Kindknoten vorhanden sein werden. Aus diesem Grund wird bei nominalen Tests für fehlende Werte ein gesonderter Kindknoten angelegt. An diesen werden alle Instanzen weitergeleitet, die für das zu überprüfende Attribut einen fehlenden Wert haben. In dem gesonderten Kindknoten kann der Aufbau des Entscheidungsbaumes normal fortgeführt werden. In Abbildung 5.3b ist dieser Fall noch einmal grafisch dargestellt. Hierbei stellt das *Attribut Z* ein nominales Attribut mit den Werten *A* und *B*. Das Gewicht der Instanz wird bei fehlenden Werten auf einem nominalen Test nicht angepasst.



(a) Weiterreichung einer Instanz, die für das numerische Attribut *X* einen fehlenden Wert hat

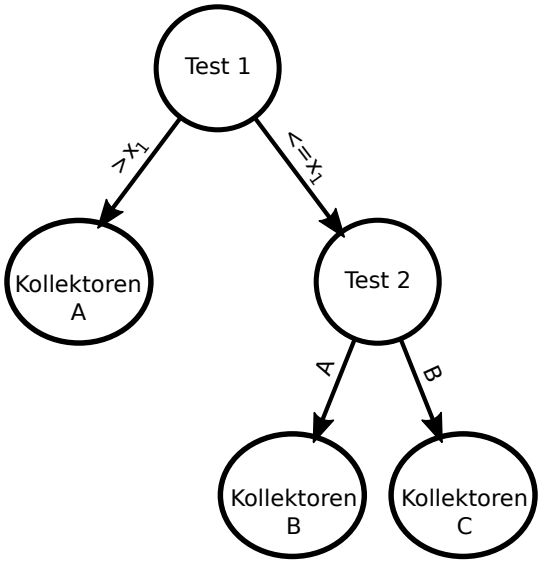


(b) Weiterreichung einer Instanz, die für das nominale Attribut *Z* einen fehlenden Wert hat

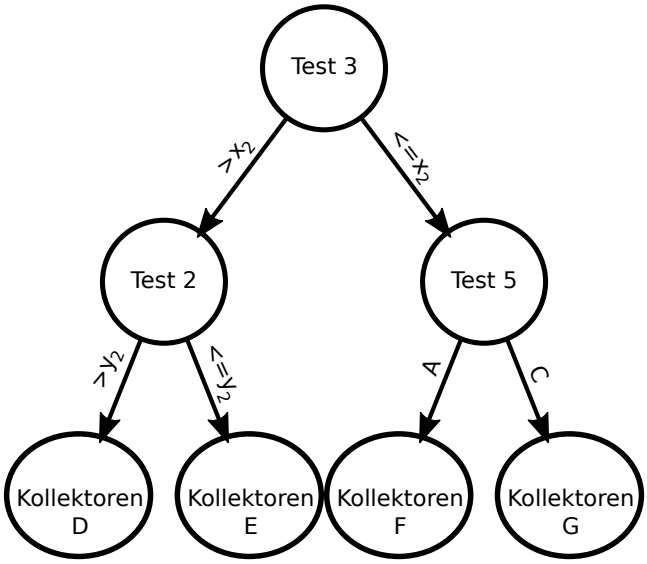
Abbildung 5.3: Umgang mit fehlenden Werten

5.2.5 Aggregation von Ensembles

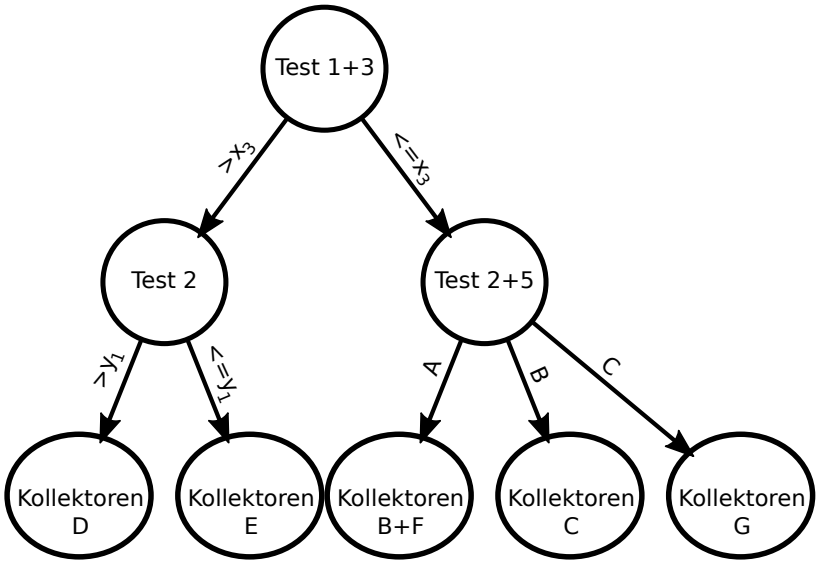
Wie in dem Abschnitt 5.2.3 beschrieben, werden die Entscheidungsbäume nach einem speziellen Verfahren erstellt, damit diese im Nachhinein mit wenig Aufwand aggregiert werden können. Hierbei verwenden die zu aggregierenden Entscheidungsbäume auf jeder Höhe des Entscheidungsbaumes dieselben Attribute und die Instanzen werden nur den Blättern hinzugefügt, die sich auf der maximalen Tiefe des Baumes befinden. Des Weiteren müssen die Entscheidungsbäume mit denselben Parametern erstellt werden und dieselben Kollektoren verwenden. Durch diese Eigenschaft ist es möglich, zwei Entscheidungsbäume zu aggregieren, indem jeder einzelne Knoten des einen Baumes mit demselben Knoten des anderen Baumes zusammengefügt wird. In Abbildung 5.4a und 5.4b sind zwei Entscheidungsbäume abgebildet, die mit dem Algorithmus erstellt wurden.



(a) Entscheidungsbaum 1



(b) Entscheidungsbaum 2



(c) Aggregation der Entscheidungsbäume 1 und 2

Abbildung 5.4: Aggregation von zwei Entscheidungsbäumen

Um diese beiden Entscheidungsbäume aggregieren zu können, werden beide Bäume parallel durchgegangen. Beginnend mit den Wurzelknoten wird bei diesem parallelen Durchgang jeder einzelne Knoten mit dem Knoten des anderen Baumes, der sich an der gleichen Stelle befindet, kombiniert. Bei der Kombination von Knoten müssen drei unterschiedliche Fälle beachtet werden. Bei dem ersten Fall werden zwei Blätter miteinander aggregiert. Hierbei müssen nur die Kollektoren der beiden Blätter addiert werden, um das aggregierte Blatt zu erstellen. Bei dem zweiten Fall werden innere Knoten miteinander aggregiert. Hierbei müssen die Tests der beiden Knoten kombiniert werden. Da beide Entscheidungsbäume dieselben Attribute auf jeder Höhe verwenden, kann davon ausgegangen werden, dass die zu aggregierenden Tests

auf demselben Attribut arbeiten. Bei der Kombination von numerischen Tests werden die gesammelten Wertemengen beider Tests zusammengefügt. Hierbei werden alle Werte der beiden Mengen in eine neue Wertemenge kopiert. Diese neue Menge wird anschließend verwendet um den Trennwert für den aggregierten Test zu bestimmen. Zu beachten ist, dass durch die Aggregierung von numerischen Tests sich der Trennwert ändern kann. Dies hat zur Folge, dass manche Instanzen, die für den Aufbau der zu aggregierenden Entscheidungsbäume verwendet wurden, mit dem alten Trennwert anderen Blättern zugewiesen wurden als mit dem neuen Trennwert. Bei der Kombination von nominalen Tests müssen die Werte, mit denen die Tests die Instanzen separieren können, verglichen werden. Hierbei werden alle unterschiedlichen Werte die in beiden Tests auftauchen, dem aggregierten Test hinzugefügt. Wird beispielsweise ein Test, der anhand der Werte {*A,B*} separiert, mit einem Test, der anhand der Werte {*B,C*} separiert, aggregiert, so entsteht ein Test, der anhand der Werte {*A,B,C*} separieren kann. Da in diesem Fall nur der Kindknoten, an den die Instanzen mit dem Wert *B* weitergeleitet werden, in beiden Bäumen vorhanden ist, können die Teilbäume der Werte *A* und *C* dem aggregierten Entscheidungsbaum einfach angefügt werden. Nachdem zwei innere Knoten miteinander kombiniert wurden, muss die Aggregation in den Kindknoten, die in beiden Bäumen vorhanden sind, fortgeführt werden. Des Weiteren kann der Fall eintreten, dass ein innerer Knoten mit einem Blatt kombiniert werden muss. Dieser Fall entsteht, weil bei dem Aufbau der Entscheidungsbäume auch Blätter erzeugt werden können, die sich nicht auf der maximalen Höhe des Baumes befinden. Diese Blätter können bei der Aggregation jedoch vernachlässigt werden, weil durch den Algorithmus für den Aufbau der Entscheidungsbäume gewährleistet werden kann, dass nur Blätter, die sich auf der maximalen Höhe des Entscheidungsbaumes befinden, Informationen für eine Vorhersage enthalten. Aus diesem Grund wird bei dem Kombinieren eines inneren Knoten mit einem Blatt, immer dem aggregierten Baum der innere Knoten angehängt und das Blatt nicht beachtet. In Abbildung 5.4c ist der aggregierte Entscheidungsbaum dargestellt, der durch Aggregation der Bäume aus Abbildung 5.4a und 5.4b entstanden ist.

Algorithmus 10 stellt das beschriebene Verfahren als Pseudocode dar. Hierbei wird dem Baum, der als erster Parameter übergeben wird, die Eigenschaften des zweiten Baumes angefügt. Es handelt sich um einen rekursiven Algorithmus, der mit den Wurzelknoten der beiden zu aggregierenden Entscheidungsbäumen angestoßen werden muss.

Algorithmus 10 $mergeTrees(node, otherNode)$

Eingangsvariablen:

$node$: Der momentan betrachtete Knoten
$otherNode$: Der momentan betrachtete Knoten des anderen Baums

Ablauf:

```
1: if node ist ein innerer Knoten und otherNode ist ein innerer Knoten then
2:     Kombiniere den Test von node mit dem Test von otherNode
3:     for Jeden Kinknoten a_i von node und den zugehörigen Kindknoten b_i von otherNode do
4:         Kombiniere die Kindknoten mit combineNodes(a_i, b_i)
5:     end for
6:     Füge node die Kindknoten von otherNode hinzu, die noch nicht vorhanden sind
7: else
8:     if node ist ein Blatt und otherNode ist ein Blatt then
9:         for Jeden Kollektor a_i von node und jeden Kollektor b_i von otherNode do
10:            Addiere Kollektor a_i mit Kollektor b_i
11:        end for
12:    else
13:        if node ist ein Blatt und otherNode ist ein innerer Knoten then
14:            Ersetze node durch otherNode
15:        end if
16:    end if
17: end if
```

5.2.6 Aggregation bei einer Kreuzvalidierung

Die Aggregation kann bei einer Evaluation durch eine Kreuzvalidierung eingesetzt werden, um eine komplettes Modell nach der Evaluierung erstellen zu können. Hierbei gibt es drei unterschiedliche Ansätze für die Verwendbarkeit der Aggregation.

1. Post-Aggregation: Bei dieser Evaluierung wird für jeden Teil der Kreuzvalidierung ein Ensemble mit den Instanzen des Teils erstellt. Jedes Ensemble wird hierbei mit den gleichen Parametereinstellungen erstellt, wodurch sichergestellt werden kann, dass die Ensembles aggregiert werden können. Um einen Teil der Kreuzvalidierung evaluieren

zu können, müssen die Ensembles der anderen Teile verwendet werden. Um nicht für die Evaluation jedes Teils der Kreuzvalidierung die Ensembles aggregieren zu müssen, wird ein vereinfachtes Verfahren für die Erstellung einer Vorhersage einer Instanz verwendet. Hierbei wird jeder Entscheidungsbaum des Ensembles der anderen Teile einzeln betrachtet. In der ersten Iteration wird der erste Entscheidungsbaum jedes Ensembles verwendet, um eine Vorhersage generieren zu können. Diese Vorhersagen werden anschließend kombiniert und stellt die Vorhersage des ersten Entscheidungsbaums im Ensemble dar. In der zweiten Iteration wird der zweite Entscheidungsbaum jedes Ensembles verwendet, um die Vorhersage für den zweiten Entscheidungsbaum erstellen zu können. Dies wird für jeden Baum des Ensembles durchgeführt. Durch dieses Verfahren wird die Aggregation der Entscheidungsbäume imitiert, indem nur die Vorhersagen der Bäume miteinander kombiniert werden. Anschließend werden die imitierten Vorhersagen jedes Baumes miteinander kombiniert, um die finale Vorhersage für eine Instanz machen zu können. Dieses Verfahren wird für jede zu evaluierende Instanz durchgeführt. Nachdem alle Instanzen evaluiert sind, können alle Ensembles zu einem Ensemble aggregiert werden, das alle Instanzen aller Teile enthält. Bei einer dreifachen Kreuzvalidierung wird der Datensatz beispielsweise in die Teile *A*, *B* und *C* aufgeteilt. Das erste Ensemble wird mit den Instanzen des Teils *A*, das zweite Ensemble mit den Instanzen des Teils *B* und das dritte Ensemble mit den Instanzen des Teils *C* erstellt. Für die Evaluierung der Instanzen des Teils *C* werden die Ensembles verwendet, die mit den Teilen *A* und *B* aufgebaut wurden. Für die Erstellung einer Vorhersage einer Instanz aus dem Teil *C* wird zuerst der erste Entscheidungsbaum des Ensembles *A* und der erste Entscheidungsbaum des Ensembles *B* verwendet. Mit beiden Entscheidungsbäumen wird für die Instanz eine Vorhersage generiert, welche anschließend kombiniert wird. Dieses Verfahren wird für alle Entscheidungsbäume in den Ensembles wiederholt. Anschließende werden die so erstellten Vorhersagen nochmals kombiniert, um die finale Vorhersage für die Instanz erstellen zu können. Auf diese Art und Weise werden alle Instanzen des Teils *C* evaluiert. Dasselbe Verfahren wird für die Evaluierung der anderen Teile durchgeführt. Der Vorteil dieses Verfahrens ist, dass für jeden Teil der Evaluierung ein Ensemble erstellt wird, das nur mit den Daten eines Teils aufgebaut wird. Bei der normalen Kreuzvalidierung müssen n-1 Teile für die Erstellung eines Ensembles verwendet werden. Der Nachteil des Verfahrens der Post-Aggregation besteht darin, dass die Kombinationen der Vorhersagen der unterschiedlichen Ensembles für jede Instanz zusätzliche Zeit beanspruchen. Des Weiteren werden die Ensembles nur mit einem Teil der Daten aufgebaut, wodurch die Güte der Entscheidungsbäume sinken kann.

2. Pre-Aggregation: Bei dieser Evaluierung wird ebenfalls für jeden Teil der Kreuzvalidierung ein Ensemble mit den darin enthaltenen Instanzen erstellt. Um nun einen Teil zu evaluieren, werden zuerst die Ensembles der anderen Teile der Kreuzvalidierung miteinander aggregiert. Mit dem aggregierten Ensemble können anschließend die Instanzen des zu evaluierenden Teils überprüft werden. Bei einer dreifachen Kreuzvalidierung mit den Teilen *A*, *B* und *C* würde dies bedeuten, dass für die Evaluierung des Teils *C* zuerst die Ensembles der Teile *A* und *B* aggregiert werden müssen. Mit dem aggregierten Ensemble können anschließend die Instanzen des Teils *C* evaluiert werden. Diese Prozedur wird für die Evaluierung jedes Teils der Kreuzvalidierung durchgeführt. Nach der Evaluierung werden alle Ensembles miteinander aggregiert, sodass ein Modell entsteht, dass alle Instanzen aller Teile enthält. Der Vorteil dieses Verfahrens ist, dass die Vorhersagen der einzelnen Entscheidungsbäume nicht mehr miteinander kombiniert werden müssen, sondern für die Evaluation eines Teils ein aggregiertes Ensemble zu Verfügung steht. Der Nachteil besteht darin, dass die Aggregation der Ensembles zusätzlich Zeit beansprucht und dass durch die Aggregierung der Ensembles die Güte der Entscheidungsbäume sinken kann.

3. Finale-Aggregation: Bei dieser Evaluierung wird eine normale Kreuzvalidierung durchgeführt, jedoch werden die erstellten Modelle für die Evaluierung jedes Teils nicht verworfen. Die Idee ist, die erstellten Modelle nach der Evaluierung zu aggregieren. Hierbei muss darauf geachtet werden, dass bei einer Kreuzvalidierung mit n Teilen n Modelle entstehen, welche jeweils auf n-1 Teilen der Daten basieren. Zum Beispiel kann eine dreifache Kreuzvalidierung durchgeführt werden. Hierbei wird der Datensatz in die Teile *A*, *B* und *C* aufgeteilt. Das erste Modell wird mit den Teilen *A* und *B*, das zweite mit den Teilen *B* und *C* und das dritte mit den Teilen *A* und *C* erstellt. Würden wir nun die drei Modelle aggregieren, so würden wir ein Modell erhalten, dass zweimal den Teil *A*, zweimal den Teil *B* und zweimal den Teil *C* enthält. Daraus folgt, dass das aggregierte Modell nach einer dreifachen Kreuzvalidierung den Datensatz doppelt enthält. Für ein beliebiges n kann man auch sagen, dass das aggregierte Modell immer n-1 mal den Datensatz beinhaltet. Aus diesem Grund muss ein Verfahren entwickelt werden, in dem das aggregierte Modell nur einmal den Datensatz enthält. Die Idee ist, die gesammelten Informationen der Kollektoren der aggregierten Entscheidungsbäume durch n-1 zu dividieren, um die Entscheidungsbäume wieder auf die Verwendung eines Datensatzes zu normalisieren. Der Vorteil dieses Verfahrens besteht darin, dass keine Modifikation bei der Durchführung der Kreuzvalidierung vorgenommen wird. Das heißt, dass das Ergebnis dieser Evaluation dasselbe ist, wie das Ergebnis der normalen Kreuzvalidierung. Des Weiteren werden durch die Verwendung von n-1 Teilen des Datensatzes für jedes Ensemble bessere Entscheidungsbäume erstellt, als bei der Pre- und der Post-Aggregation. Der Nachteil dieses Verfahren ist, dass für die Erstellung der Modelle immer n-1 Teile verwendet werden müssen.

Tabelle 5.1 vergleicht für die drei vorgestellten Verfahren die Kosten für die Erstellung, die Evaluierung und die anschließende Erstellung des Modells, das auf allen Teilen basiert. Zusätzlich sind noch die Kosten der normalen Kreuzvalidierung dargestellt. Hierbei wird eine n-fache Kreuzvalidierung mit einem Datensatz durchgeführt, der k Instanzen enthält. Es wird die Annahme gemacht, dass das Trainieren des Modells mit einer Instanz die Kosten 1 beträgt und dass das Evaluieren einer Instanz ebenfalls die Kosten 1 beträgt. Anhand der Tabelle lässt sich erkennen, dass die Verfahren der Post- und Pre-Aggregation mehr Rechenzeit für die Evaluation benötigen, für die Erstellung der Modelle jedoch weniger Zeit beanspruchen. Die Pre-Aggregation eignet sich besonders gut für eine Kreuzvalidierung mit wenigen Teilen und einem sehr großen Datensatz. Die Post-Aggregation eignet sich nicht besonders gut für große Datensätze, weil für jede Instanz, die durch das Modell getestet wird, die Vorhersagen der Entscheidungsbäume der Ensembles kombiniert werden müssen. Aus diesem Grund beansprucht jede Instanz, die evaluiert werden muss, mehr Rechenaufwand als bei den anderen Verfahren. Der Nachteil der Post- und der Pre-Aggregation ist, dass die Verfahren nur für eine Kreuzvalidierung auf wenigen Teilen gut funktioniert. Ausschlaggebend hierfür ist, dass durch die Erstellung der Ensembles mit wenigen Instanzen, die Qualität der Entscheidungsbäume sinkt. Aus diesem Grund kann die Evaluierung für eine Kreuzvalidierung mit vielen Teilen nicht aussagekräftig sein. Einen Kompromiss bietet die Finale-Aggregation. Dieses Verfahren hat dieselben Kosten wie die normale Kreuzvalidierung, es ist jedoch möglich, die Ensembles der unterschiedlichen Teile am Ende zu aggregieren. Des Weiteren werden die Ergebnisse der Evaluierung nicht durch schlechte Modelle beeinflusst. Aufgrund des eingegrenzten Zeitrahmens der Bachelorarbeit konnten keine Versuche für die unterschiedlichen Verfahren der Aggregation gemacht werden.

Tabelle 5.1: Kosten für die unterschiedlichen Verfahren der Aggregation bei einer Kreuzvalidierung

Verfahren	**Erstellung der Modelle**	**Evaluierung**	**Erstellung eines Modells, das auf allen Instanzen basiert**
Normale Kreuzvalidierung	$\frac{n-1}{n} \cdot k \cdot n$	$\frac{1}{n} \cdot k \cdot n$	k
Post-Aggregation	$\frac{1}{n} \cdot k \cdot n$	$n \cdot k \cdot \frac{1}{n} \cdot (1 + \textit{Kombinieren der Vorhersagen})$	n Ensembles aggregieren
Pre-Aggregation	$\frac{1}{n} \cdot k \cdot n$	$n \cdot (\frac{1}{n} \cdot k + \textit{Aggregation von n-1 Ensembles})$	n Ensembles aggregieren
Finale-Aggregation	$\frac{n-1}{n} \cdot k \cdot n$	$\frac{1}{n} \cdot k \cdot n$	n Ensembles aggregieren

6 Evaluation

In diesem Kapitel werden die beiden implementierten Verfahren für die Erstellung der Entscheidungsbäume evaluiert. In Abschnitt 6.1 werden die Metriken erklärt, die für die Evaluation verwendet werden. In dem folgenden Abschnitt 6.2 werden die für die Evaluation genutzten Datensätze vorgestellt. In den darauffolgenden Abschnitten werden die Ergebnisse der Experimente dargestellt, die innerhalb dieser Arbeit durchgeführt wurden.

6.1 Evaluationsmetriken

In diesem Abschnitt werden Metriken vorgestellt, die für die Evaluation verwendet wurden. Hierbei wird zwischen Metriken für die Multilabel-Klassifikation und der binären Klassifikation beziehungsweise der Multiklassen-Klassifikation unterschieden.

6.1.1 Metrik für die binäre Klassifikation und Multiklassen-Klassifikation

Für die Versuche, bei denen durch eine Klassifikation nur ein Wert für eine nominales Attribut vorhergesagt wird, wird die Metrik *Accuracy* verwendet. Um die *Accuracy* berechnen zu können, wird die Anzahl der richtig klassifizierten Instanzen durch die Anzahl aller zu klassifizierenden Instanzen dividiert. Eine richtige Vorhersage ist dadurch gekennzeichnet, dass die Vorhersage für eine Instanz durch das Klassifikationsverfahren die Klasse vorhersagt, der die Instanz tatsächlich angehört. Durch die Formel 6.1 wird diese Berechnung dargestellt. Die *Accuracy* kann einen Wert zwischen 0 und 1 annehmen. Je höher der Wert ist, desto besser ist der Klassifizierer.

$$Accuracy = \frac{Richtige\ Vorhersagen}{Anzahl\ aller\ Vorhersagen} \tag{6.1}$$

6.1.2 Metriken für die Multilabel-Klassifikation

Für die Multilabel-Klassifikation müssen spezielle Metriken verwendet werden, weil die Werte von mehreren nominalen Attributen vorhergesagt werden. Im ersten Abschnitt wird auf die Metriken *Subset-Accuracy* und *Hamming Loss* eingegangen. In dem darauffolgenden Abschnitt werden die Micro- und die Macro-Evaluationsmetriken vorgestellt.

Subset-Accuracy und Hamming Loss

Die folgenden beiden Metriken wurden aus [Tsoumakas et al., 2010] entnommen. Hierbei enthält der Datensatz, der für die Evaluation verwendet wird, N Instanzen. Durch den Vektor Y_i werden die richtigen Werte der zu vorhersagenden Klassen der i-ten Instanz dargestellt. Analog wird die Vorhersage dieser Klassen der i-ten Instanz durch den Klassifizierer mit dem Vektor Z_i dargestellt. Durch L wird die Anzahl der zu vorhersagenden Klassen angegeben.

Ähnlich wie bei der *Accuracy*, gibt es bei der Multilabel-Klassifikation auch eine Metrik, die angibt, wieviele Instanzen richtig klassifiziert werden. Damit eine Instanz bei einer Multilael-Klassifikation als richtig klassifiziert werden kann, muss der Klassifizierer genau die Klassen vorhersagen, denen die Instanz tatsächlich angehört. Durch die Formel 6.2 wird diese Metrik dargestellt, wobei $I(wahr) = 1$ und $I(falsch) = 0$ gilt. Je höher der Wert der *Subset-Accuracy* ist, desto besser ist der Klassifizierer.

$$Subset\text{-}Accuracy = \frac{1}{N}\sum_{i=0}^{N} I(Y_i = Z_i) \tag{6.2}$$

Des Weiteren wird für die Evaluation die Metrik *Hamming Loss* verwendet, welche durch die Formel 6.3 dargestellt ist. Bei dieser Metrik wird für jede Instanz einzeln die Vorhersage für jede Klasse mit der tatsächlichen Klasse verglichen. Mit dieser Metrik ist es möglich, die Vorhersage überprüfen zu können, auch wenn nur ein Teil der vorhergesagten Klassen mit den tatsächlichen Klassen übereinstimmt. Je niedriger der Wert von *Hamming Loss* ist, desto weniger Klassen werden falsch vorhergesagt. Folglich ist ein niedriger Wert für *Hamming Loss* optimal. Durch das Zeichen $\oplus$ wird das Exklusiv-Oder dargestellt.

$$HammingLoss = \frac{1}{N}\sum_{i=0}^{N} \frac{|Y_i \oplus Z_i|}{L} \tag{6.3}$$

Micro und Macro Evaluationsmetriken

Bei der Verwendung von Micro und Macro Evaluationsmetriken für die Multilabel-Klassifikation wird auf die Evalutionsmetriken für eine binäre Klassifikation zurückgegriffen. Bei einer binären Klassifikation wird für ein nominales Attribut einer von zwei Werten vorhergesagt. Nehmen wir an, die Wertemenge für das zu vorhersagende Attribut besteht aus den Werten {*ja, nein*}. Bei der Evaluierung einer Vorhersage kann es nun zu folgenden Fällen kommen:

1. Durch das Klassifikationsverfahren wird die Klasse *ja* vorhergesagt und die Instanz gehört der Klasse *ja* an. Hierbei spricht man auch von richtig positiven Vorhersagen (rp).

2. Durch das Klassifikationsverfahren wird die Klasse *nein* vorhergesagt und die Instanz gehört der Klasse *nein* an. Hierbei spricht man auch von richtig negativen Vorhersagen (rn).

3. Durch das Klassifikationsverfahren wird die Klasse *ja* vorhergesagt, jedoch gehört die Instanz der Klasse *nein* an. Hierbei spricht man auch von falsch positiven Vorhersagen (fp).

4. Durch das Klassifikationsverfahren wird die Klasse *nein* vorhergesagt, jedoch gehört die Instanz der Klasse *ja* an Hierbei spricht man auch von falsch negativen Vorhersagen (fn).

Mit diesen Fällen können unterschiedliche Eigenschaften eines Klassifikationsverfahren gemessen werden. In dieser Arbeit verwenden wir die Evaluationsmetirken *Precision*(6.4), *Recall*(6.5) und *F1-Measure*(6.6). Durch die Metrik *Precision* werden die durch das Klassifikationsverfahren mit *ja* klassifizierten Instanzen untersucht. Hierbei wird der Anteil der Instanzen berechnet, die tatsächlich der Klasse *ja* angehören. Hingegen werden bei der Metrik *Recall* die Instanzen untersucht, die tatsächlich der Klasse *ja* angehören. Hierbei wird der Anteil der Instanzen berechnet, die von dem Klassifikationsverfahren mit *ja* klassifiziert wurden. Mit der Metrik *F-Measure* werden die Metriken *Precision* und *Recall* kombiniert. Die Metrik *F1-Measure* stellt ein harmonisches Gleichgewicht zwischen *Precision* und *Recall* her. Für alle drei Metriken gilt, je näher der Wert der Metriken an 1 ist, desto besser ist der Klassifizierer.

$$\textit{Precision(rp,fp,rn,fn)} = \frac{rp}{rp + fp} \tag{6.4}$$

$$\textit{Recall(rp,fp,rn,fn)} = \frac{rp}{rp + fn} \tag{6.5}$$

$$\textit{F1-Measure(rp,fp,rn,fn)} = \frac{2 \cdot rp}{2 \cdot rp + fn + fp} \tag{6.6}$$

Um diese Metriken für die Evaluation einer Multilabel-Klassifikation nutzen zu können, müssen spezielle Formeln verwendet werden [Tsoumakas and Vlahavas, 2007]. Hierbei unterscheidet man zwischen der Micro- und der Macro-Formel, welche durch 6.7 und 6.8 dargestellt sind. Bei diesen Formeln stellt die Funktion $B(tp, fp, tn, fn)$ eine der vorgestellten Metriken für eine binäre Klassifikation dar und L gibt die Anzahl der zu vorhersagenden Klassen an. Bei der Micro-Formel werden die richtig positiven, falsch positiven, richtig negativen und falsch negativen Vorhersagen für jede Klasse vorher bestimmt. Für jeden Fall der Vorhersage wird anschließend die Summe über alle Klassen gebildet und inn die Metrik $B(tp, fp, tn, fn)$ eingesetzt. Bei der Macro-Formel hingegen wird die Metrik $B(tp, fp, tn, fn)$ für jede zu vorhersagende Klasse angewendet. Anschließend wird die Summe der erhaltenen Werte gebildet und durch die Anzahl der Klassen geteilt wird.

$$Micro = B(\sum_{i=0}^{L} rp_i, \sum_{i=0}^{L} fp_i, \sum_{i=0}^{L} rn_i, \sum_{i=0}^{L} fn_i) \tag{6.7}$$

$$Macro = \frac{1}{L} \sum_{i=0}^{L} B(rp_i, fp_i, rn_i, fn_i) \tag{6.8}$$

6.2 Auswahl der Datensätze

Der Fokus bei der Evaluierung liegt auf Datensätzen, bei denen eine Multilabel-Klassifikation oder eine Multiklassen-Klassifikation durchgeführt werden soll. Für die Multilabel-Klassifikation werden sechs Datensätze verwendet, die sich in der Anzahl der Instanzen, Anzahl der Attribute und Anzahl der zu vorhersagenden Klassen unterscheiden. Alle Multilabel-Klassifikations-Datensätze wurden von der MULAN-Webseite[1] entnommen. In Tabelle 6.1 werden die Eigenschaften der verwendeten Datensätze für die Multilabel-Klassifikation abgebildet. Für die Multiklassen-Klassifikation werden ebenfalls sechs unterschiedliche Datensätze verwendet. Die Datensätze *Airline* und *Electricity* wurden von MOA-Webseite[2] entnommen. Hierbei handelt es sich um Datensätze, die sehr viele Instanzen enthalten. Des Weiteren werden die Datensätze *Mushroom*[3], *Eye State*[4], *Letter Recognition*[5] und *Thyroid Disease*[6] verwendet. In Tabelle 6.2 sind die Eigenschaften aller verwendeten Datensätze für die Multiklassen-Klassifikation abgebildet.

Tabelle 6.1: Multilabel-Klassifikations-Datensätze

Name	Instanzen	Attribute (nominal)	Attribute (numerisch)	Anzahl Klassen	Fehlende Werte
TMC2007	28596	49060	0	22	nein
Corel5k	5000	499	0	374	nein
Flags	194	9	10	7	nein
Mediamill	43907	0	120	101	nein
Scene	2407	0	294	6	nein
Yeast	2417	0	103	14	nein

Tabelle 6.2: Klassifikations-Datensätze

Name	Instanzen	Attribute (nominal)	Attribute (numerisch)	Anzahl Klassen	Fehlende Werte
Mushroom	8124	22	0	2	ja
Eye State	14980	0	14	2	nein
Letter Recognition	20000	0	16	26	nein
Thyroid Disease	7200	21	7	6	ja
electricity	45312	4	2	2	nein
airline	539383	3	4	2	nein

6.3 Details zur Durchführung der Experimente

Für die Evaluierung eines Datensatzes für eine bestimmte Parametereinstellung wird jedes mal eine 10-fache Kreuzvalidierung durchgeführt. Da die erstellten Modelle der evaluierenden Algorithmen stark von dem zufälligen Seed abhängen, wird diese 10-fache Kreuzvalidierung zehn mal mit unterschiedlichen Seeds durchgeführt. Die Ergebnisse der zehn Wiederholungen werden anschließend gemittelt. Zusätzlich werden zu den gemittelten Ergebnissen die Standardabweichung der zehn Wiederholungen in den Tabellen mit angegeben. Eine Ausnahme bildet der Versuch, bei dem die spezielle Kreuzvalidierung des Batch-Algorithmus evaluiert wird. Hier wird, statt einer 10-fachen Kreuzvalidierung, eine LOO-Kreuzvalidierung mit zehn Wiederholungen durchgeführt. In den Abschnitten 6.4 und 6.5 wird der entworfene Algorithmus evaluiert, der unter der Annahme des Batch-Lernens arbeitet. Hierbei wird der Algorithmus mit dem Dice-Lernalgorithmus verglichen und die spezielle Kreuzvalidierung evaluiert. In den Abschnitten 6.6, 6.7 und 6.8 wird der entworfene Online-Algorithmus evaluiert. In diesen Versuchen wird der Online-Algorithmus mit dem Hoeffding-Tree verglichen, unterschiedliche Parametereinstellungen analysiert und die Auswirkungen des Online-Lernens auf den Online-Algorithmus betrachtet.

[1] `http://mulan.sourceforge.net/datasets-mlc.html` Abgerufen am: 24.04.2015

[2] `http://moa.cms.waikato.ac.nz/datasets/` Abgerufen am: 24.04.2015

[3] `https://ml-dolev-amit.googlecode.com/svn-history/r56/trunk/weka/required_datasets/mushroom.arff` Abgerufen am: 24.04.2015

[4] `https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State` Abgerufen am: 24.04.2015

[5] `https://github.com/renatopp/arff-datasets/blob/master/classification/letter.arff` Abgerufen am: 24.04.2015

[6] `http://sourceforge.net/projects/meka/files/Datasets/thyroid-L7.arff` Abgerufen am: 24.04.2015

6.4 Vergleich des Batch-Algorithmus mit dem Dice-Lernalgorithmus

In diesem Versuch wird der Dice-Lernalgorithmus mit dem entworfenen Batch-Algorithmus aus Abschnitt 5.1 verglichen. In diesem Experiment wird überprüft, welche Auswirkungen das modifizierte Verfahren für die Bestimmung des Trennwertes in einem numerischen Test des Batch-Algorithmus gegenüber dem Dice-Lernalgorithmus hat. Leider konnten die Datensätzen *Airline*, *Letter Recognition* und *Thyroid Disease* bei diesem Experiment nicht verwendet werden, weil bei der Verarbeitung dieser Datensätze der Dice-Algorithmus Fehler geworfen hat. In den Tabellen 6.3 und 6.4 sind die Evaluationsergebnisse der anderen Datensätze abgebildet. Hierbei wurde ein Ensemble von 50 Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests verwendet. Als Repräsentation der Qualität der Algorithmen werden die Metriken *Micro F1-Measure* für die Multilabel-Klassifikations-Datensätze und *Accuracy* für die Multiklassen-Klassifikations-Datensätze verwendet.

Tabelle 6.3: Evaluationsergebnisse der Multilabel-Klassifikations-Datensätze mit den Parametereinstellungen von 50 Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests

Datensatz	Laufzeit Batch-Algorithmus (Sekunden)	Laufzeit Dice-Algorithmus (Sekunden)	Micro F1-Measure Batch-Algorithmus	Micro F1-Measure Dice-Algorithmus
Flags	0,230	1,996	0,7428±0,0039	0,7078±0,0062
Yeast	4,847	37,321	0,6365±0,0023	0,6371±0,0015
Scene	14,575	45,726	0,4522±0,0054	0,4561±0,0056
Corel5k	46,142	143,097	0,2013±0,0003	0,0325±0,0003
Mediamill	295,611	1869,651	0,5494±0,0003	0,5618±0,0003
TMC2007	1014,431	674,633	0,4159±0,0001	0,4158±0,0001

Tabelle 6.4: Evaluationsergebnisse der Multiklassen-Klassifikations-Datensätze mit den Parametereinstellungen von 50 Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests

Datensatz	Laufzeit Batch-Algorithmus (Sekunden)	Laufzeit Dice-Algorithmus (Sekunden)	Accuracy Batch-Algorithmus	Accuracy Dice-Algorithmus
Mushroom	32,929	20,271	1,0000±0,0000	1,0000±0,0000
Eye State	28,597	125,808	0,5148±0,0036	0,5038±0,0050
Electricity	92,303	406,989	0,7704±0,0014	0,7656±0,0019

Aus den Tabellen lässt sich erkennen, dass der Batch-Algorithmus in vielen Fällen schneller die Daten verarbeiten kann, als der Dice-Algorithmus. Nur die Bewältigung der Datensätze *TMC2007* und *Mushroom*, welche ausschließlich aus nominalen Attributen bestehen, werden durch den Dice-Algorithmus schneller durchgeführt. Des Weiteren unterscheiden sich die Ergebnisse der verschiedenen Evaluationsmetriken des Batch-Algorithmus und des Dice-Algorithmus für diese beiden Datensätze kaum. Dieses Verhalten lässt sich wahrscheinlich auf die iterative Implementierung der Erstellung der Entscheidungsbäume des Dice-Lernalgorithmus zurückführen. Fragwürdig sind jedoch die Evaluationsergebnisse des Datensatzes *Corel5k*, welche in Tabelle 6.5 dargestellt sind.

Tabelle 6.5: Evaluationsergebnisse für den Datensatz *Corel5k* mit den Parametereinstellungen von 50 Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests

Metrik	Batch-Algorithmus	Dice-Algorithmus
Laufzeit (Sekunden)	46,142	143,097
Subset-Accuracy	0,0004±0,0001	0,0000±0,0000
Hamming Loss	0,0158±0,0000	0,0169±0,0000
Micro Precision	0,1919±0,0002	0,0353±0,0003
Micro Recall	0,2116±0,0003	0,0301±0,0003
Micro F1-Measure	0,2013±0,0003	0,0325±0,0003
Macro Precision	0,0059±0,0007	0,0004±0,0000
Macro Recall	0,0109±0,0000	0,0081±0,0001
Macro F1-Measure	0,0045±0,0001	0,0006±0,0000

Obwohl der Datensatz ebenfalls nur aus nominale Attributen besteht, arbeitet in diesem Fall der Batch-Algorithmus deutlich schneller. Auch die Evaluationsergebnisse aller verwendeten Metriken sind deutlich besser, als die des Dice-Algorithmus. Die Ursache dieses Verhaltens wird wahrscheinlich an der Anzahl der zu vorhersagenden Klassen liegen, welche bei diesem Datensatz deutlich höher ist, als bei den anderen Datensätzen. Folglich lässt sich hieraus schließen, dass der Batch-Algorithmus für die Vorhersage von sehr vielen Klassen bei einer Multilabel-Klassifikation besser geeignet ist, als der Dice-Algorithmus.

Wie man schon an den Metriken *Micro F1-Measure* und *Accuracy* in den Tabellen 6.3 und 6.4 erkennen kann, unterscheiden sich für die anderen Datensätze die Ergebnisse des Batch-Algorithmus kaum von den Ergebnissen des Dice-Algorithmus. Bei der Analyse der Ergebnisse der anderen Evaluationsmetriken ist jedoch aufgefallen, dass es noch deutliche Unterschiede zwischen dem Batch-Algorithmus und dem Dice-Algorithmus bei den Datensätzen *Mediamill* und *Flags* gibt. Die Versuche mit dem Datensatz *Mediamill* haben ergeben, dass der Dice-Algorithmus für die Metriken *Micro-Recall* und *Macro-Precision* eindeutig bessere Werte erzielt, als der Batch-Algorithmus. Auf der anderen Seite ist jedoch auch aufgefallen, dass der Batch-Algorithmus bei der gleichen Parametereinstellungen für die Metriken *Micro-Precision* und *Macro-Recall* eindeutig bessere Ergebnisse liefert, als der Dice-Algorithmus. Die Evaluationsergebnisse dieser vier Metriken für eine unterschiedliche Anzahl an Entscheidungsbäumen sind in Abbildung 6.1 abgebildet. Hierbei wurden Entscheidungsbäume mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests verwendet. In dieser Grafik fällt außerdem auf, dass für die Metrik *Macro-Precision* die besten Werte erzielt werden, wenn nur fünf bis zehn Entscheidungsbäume im Ensemble verwendet werden.

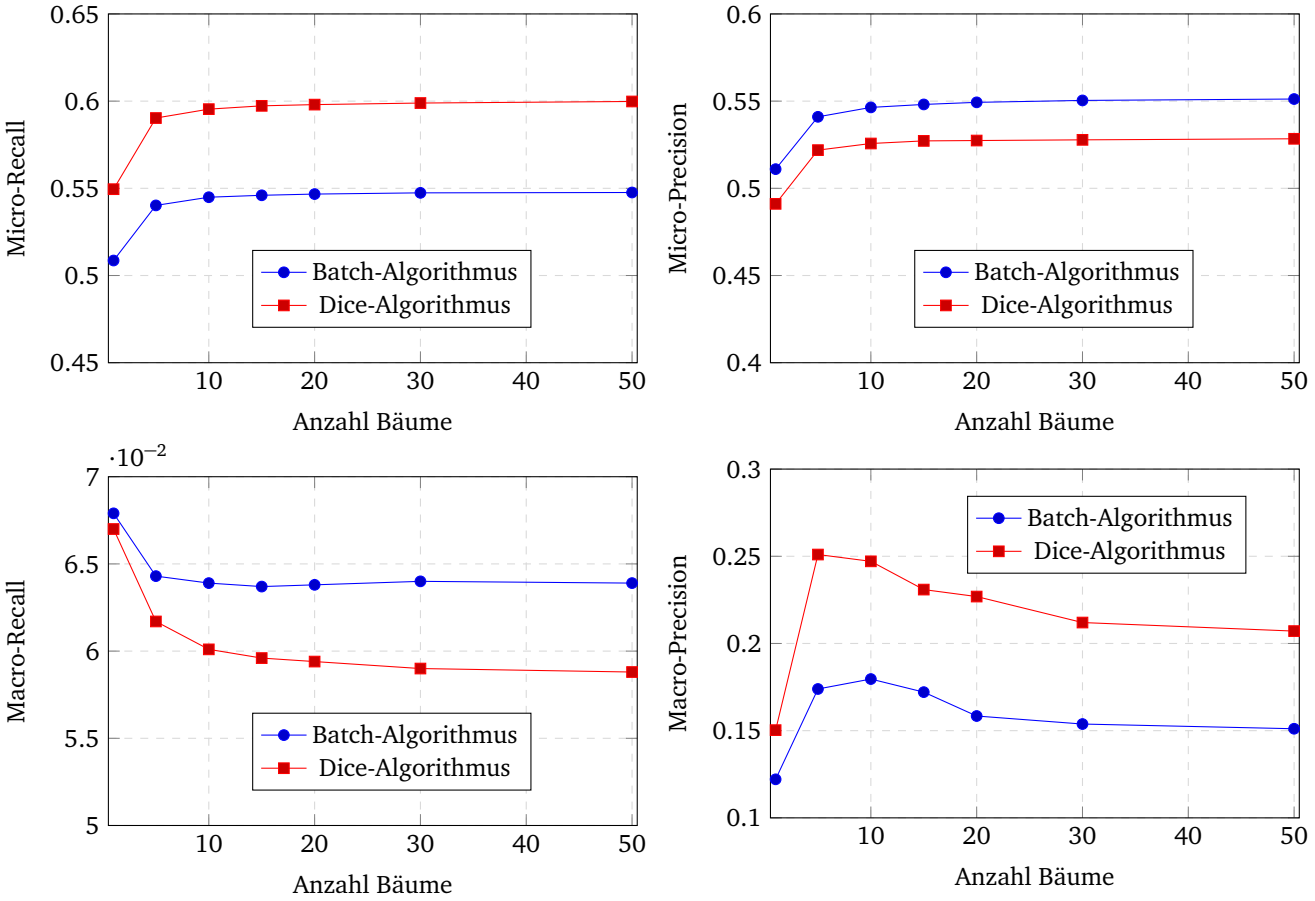


Abbildung 6.1: Vergleich der Metriken *Micro-Precision*, *Micro-Recall*, *Macro-Precision* und *Macro-Recall* für den Datensatz *Mediamill* bei einer unterschiedlichen Anzahl an Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests

Bei dem Vergleich des Batch-Algorithmus mit dem Dice-Algorithmus ist außerdem auf dem Datensatz *Flags* aufgefallen, dass der Batch-Algorithmus für die Evaluationsmetriken *Micro-Recall* und *Macro-Recall* bessere Ergebnisse erzielt. Für die Evaluationsmetriken *Micro-Precision* und *Macro-Precision* erzielt jedoch der Dice-Algorithmus die besseren Ergebnisse. Die Evaluationsergebnisse dieser vier Metriken für eine unterschiedliche Anzahl an Entscheidungsbäumen sind in Abbildung 6.2 abgebildet. Hierbei wurden ebenfalls Entscheidungsbäume mit einer maximalen Tiefe von 15 und einer minimale Anzahl von 10 Instanzen für die Erstellung eines Tests verwendet.

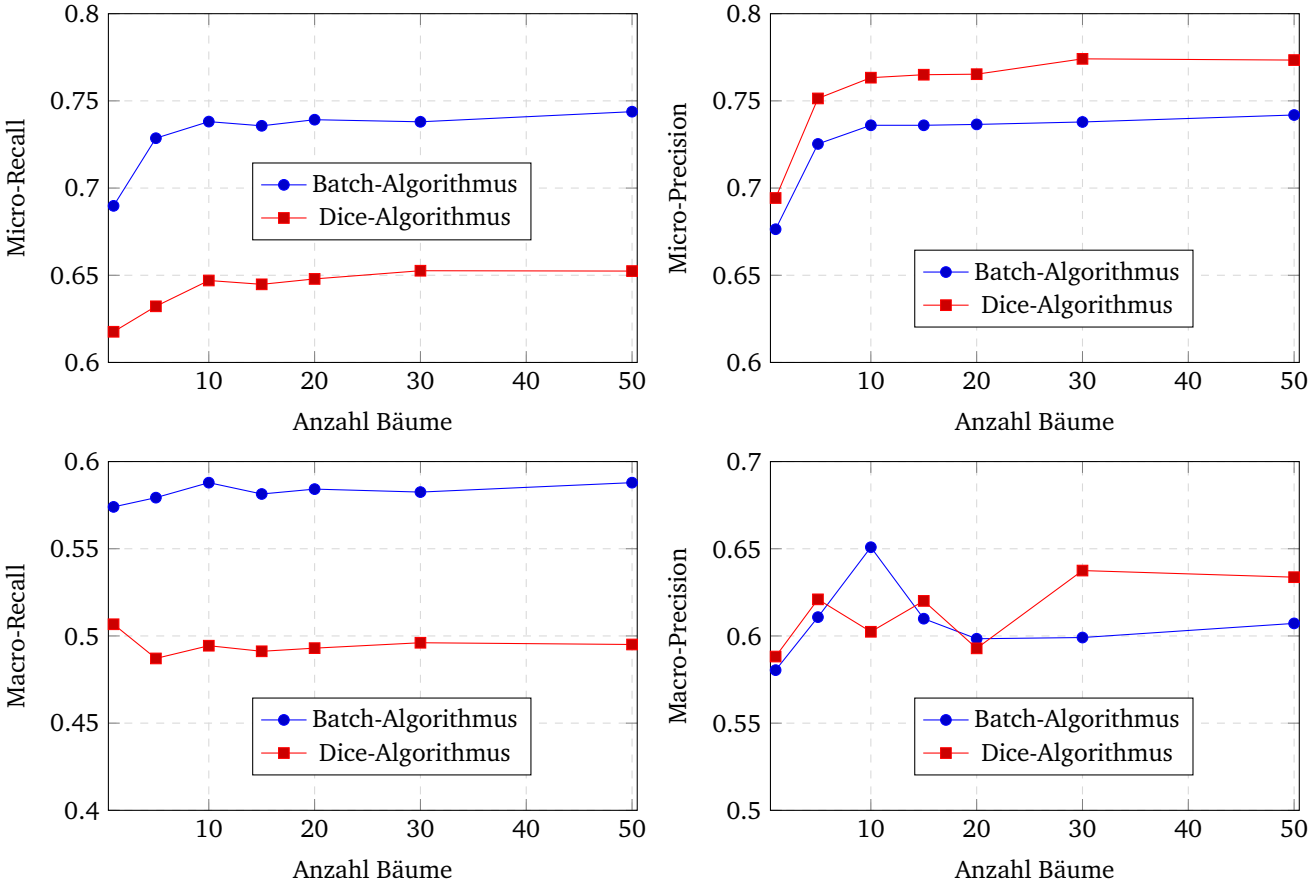


Abbildung 6.2: Vergleich der Metriken *Micro-Precision*, *Micro-Recall*, *Macro-Precision* und *Macro-Recall* für den Datensatz *Flags* bei einer unterschiedlichen Anzahl an Entscheidungsbäumen mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von 10 Instanzen für die Erstellung eines Tests

6.5 Vergleich der speziellen Leave-One-Out-Kreuzvalidierung des Batch-Algorithmus

In diesem Versuch wird das vorgestellte Verfahren für eine schnelle LOO-Kreuzvalidierung aus Abschnitt 5.1.4 mit der echten LOO-Kreuzvalidierung verglichen. Bei der Analyse der Versuche ist aufgefallen, dass die Evaluationsergebnisse der normalen LOO-Kreuzvalidiuerng sich kaum von den Ergebnissen der speziellen LOO-Kreuzvalidierung unterscheiden. Auch durch die Variation der Parametereinstellungen konnten keine auffälligen Abweichungen der Ergebnisse der beiden Evaluationsverfahren festgestellt werden. Alleinig in der benötigten Zeit für die Evaluierung unterscheiden sich die beiden Verfahren deutlich. In Tabelle 6.6 sind die Zeiten für die Evaluierung der Multiklassen-Klassifikations-Datensätze dargestellt. Der Datensatz *Airline* wurde für diesen Versuch nicht verwendet, da eine normale LOO-Kreuzvalidierung auf diesen Datensatz zu lange dauern würde. In Tabelle 6.7 sind die Zeiten für die Evaluierung der Multilabel-Klassifikations Datensätze dargestellt. Für die, in der Tabelle, abgebildeten Versuche wurden jeweils ein Entscheidungsbaum mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests verwendet.

Tabelle 6.6: Vergleich der Laufzeit der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multiklassen-Klassifikations-Datensätzen unter der Verwendung eines Entscheidungsbaumes mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests

Evaluierungs -datensätze	Laufzeit normale Leave-One-Out (Sekunden)	Laufzeit spezielle Leave-One-Out (Sekunden)
Eye State	677,695	0,240
Letter Recognition	1334,088	0,567
Mushroom	367,034	0,161
Thyroid Disease	498,457	0,320
Electricity	5461,143	0,618

Tabelle 6.7: Vergleich der Laufzeit der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multilabel-Klassifikations Datensätzen

Evaluierungs -datensätze	Laufzeit normale Leave-One-Out (Sekunden)	Laufzeit spezielle Leave-One-Out (Sekunden)
Corel5k	372,213	3,610
Flags	0,084	0,009
Scene	42,183	0,244
Yeast	36,661	0,170
Mediamill	23332,277	10,065
TMC2007	28412,112	1810,462

Aus diesen Tabellen lässt sich deutlich erkennen, dass die spezielle LOO-Kreuzvalidierung in jeder Hinsicht schneller terminiert als die normale LOO-Kreuzvalidierung. Hierbei lässt sich auch erkennen, dass die Laufzeit der normalen LOO-Kreuzvalidierung stark von der Anzahl der Instanzen abhängt, hingegen die spezielle LOO-Kreuzvalidierung nicht so stark von der Anzahl der Instanzen abhängt. Sehen lässt sich dies beispielsweise an dem Datensatz *Mediamill* und *Flags*. Auf dem Datensatz *Mediamill*, welcher 43907 Instanzen enthält, ist die spezielle LOO-Kreuzvalidierung ungefähr 2333 mal schneller als die normale Kreuzvalidierung, hingegen auf dem Datensatz *Flags*, welcher nur 197 Instanzen enthält, die spezielle LOO-Kreuzvalidierung nur ungefähr 9 mal schneller ist. Hieraus lässt sich schließen, dass besonders auf sehr großen Datensätzen die spezielle LOO-Kreuzvalidierung zeitlich gesehen einen großen Vorteil hat. Nun stellt sich die Frage, ob auch die Evaluationsergebnisse der beiden LOO-Kreuzvalidierungen dieselben sind.

Die zugehörigen Evaluationsergebnisse der Versuche aus den Tabellen 6.6 und 6.7 sind in Tabellen 6.8, 6.9, 6.10 und 6.11 dargestellt. Für die Multilabel-Klassifikations Datensätze werden aus Platzgründen nur die Metriken *Subset-Accuracy*, *Micro F1-Measure* und *Macro F1-Measure* abgebildet. Beim Vergleich der anderen Evaluationsmetriken, welche nicht in den Tabellen dargestellt sind, unterscheiden sich die Ergebnisse der speziellen LOO-Kreuzvalidierung ebenfalls kaum von den Ergebnissen der normalen LOO-Kreuzvalidierung. Bei der Analyse der Evaluationsergebnisse fällt auf, dass die Werte der Metriken der speziellen LOO-Kreuzvalidierung eine deutlich größere Standardabweichung haben, als die Werte der Metriken der normalen LOO-Kreuzvalidierung.

Tabelle 6.8: Vergleich der Evaluationsergebnisse der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multiklassen-Klassifikations-Datensätzen unter der Verwendung eines Entscheidungsbaumes mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests

Datensatz	*Accuracy* normale LOO-Kreuzvalidierung	*Accuracy* spezielle LOO-Kreuzvalidierung	t-Wert	Sicherheit, dass Gleichheit nicht zutrifft
Eye State	0,6633±0,0025	0,6690±0,0206	0,9439	50%
Letter Recognition	0,3184±0,0026	0,3257±0,0284	0,8065	50%
Mushroom	0,9922±0,0007	0,9926±0,0072	0,1668	<50%
Thyroid Disease	0,6277±0,0020	0,6240±0,0125	-0,8330	50%
Electricity	0,7242±0,0019	0,7191±0,0268	-0,6090	<50%

Um die Ergebnisse der beiden Verfahren auf Gleichheit testen zu können, wird der Zweistichproben t-Tests aus [Falk et al., 2014, S. 146ff.] verwendet. Die Berechnung des t-Wertes ist durch die Formel 6.9 dargestellt. Hierbei wird durch $\bar{x}$ und $\bar{y}$ das arithmetische Mittel und durch s_x und s_y die Standardabweichung der jeweiligen Stichprobe dargestellt. Durch die Variable n wird der Freiheitsgrad angegeben. Mit diesem Test können zwei Stichproben miteinander verglichen und abgeschätzt werden, zu welcher Wahrscheinlichkeit die Abweichung beider Stichproben durch den Zufall bestimmt ist. Hierbei handelt es sich um einen Hypothesentest, bei dem wir als Nullhypothese die Gleichheit beider Stichproben annehmen. Anhand der Tabelle der t-Quantile[7] kann überprüft werden, zu welchem Signifikanzniveau sich die Nullhypothese ablehnen lässt. Wird beispielsweise ein t-Wert von 0,8 bei einem Freiheitsgrad von 10 berechnet, so lässt sich die Nullhypothese nur mit einem Signifikanzniveau von 50% ablehnen. Je niedriger das Signifikanzniveau ist, desto stärker kann angenommen werden, dass die Nullhypothese gilt.

$$t\text{-}Wert = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{s_x^2 + s_y^2}{n}}} \tag{6.9}$$

[7] http://de.wikipedia.org/wiki/Studentsche_t-Verteilung Abgerufen am: 10.05.2015

Aus den Evaluationsergebnissen der Tabellen lässt sich ablesen, dass das Signifikanzniveau für viele Versuche weniger oder genau 50% beträgt, welches ein sehr gutes Ergebnis ist. Für diese Versuche lässt sich aussagen, dass die Abweichung der Evaluationsergebnisse der speziellen LOO-Kreuzvalidierung und der normalen Kreuzvalidierung mit einer sehr hohen Wahrscheinlichkeit durch den Zufall bestimmt ist. In den anderen Versuchen wird ein größerer t-Wert berechnet, wodurch ein höheres Signifikanzniveau ermittelt wird. Der größte Wert für das Signifikanzniveau, welches sich in den Tabellen ablesen lässt, beträgt 90%. Ein solch hoher Wert ist jedoch nicht schlimm, weil selbst mit einem Signifikanzniveau von 90% die Nullhypothese gut gestützt werden kann.

Tabelle 6.9: Vergleich der Metrik *Subset-Accuracy* der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multilabel-Klassifikations Datensätzen unter der Verwendung eines Entscheidungsbaumes mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests

Datensatz	*Subset-Accuracy* normale LOO-Kreuzvalidierung	*Subset-Accuracy* spezielle LOO-Kreuzvalidierung	t-Wert	Sicherheit, dass Gleichheit nicht zutrifft
Corel5k	0,0016±0,0003	0,0017±0,0012	0,3177	<50%
Flags	0,1072±0,0217	0,1175±0,0182	1,1521	50%
Mediamill	0,0132±0,0006	0,0125±0,0037	-0,6018	<50%
Scene	0,3757±0,0085	0,3703±0,0153	-0,9805	50%
Yeast	0,0905±0,0054	0,0964±0,0114	1,4866	80%
TMC2007	0,0488±0,0001	0,0489±0,0001	1,8791	90%

Tabelle 6.10: Vergleich der Metrik *Micro F1-Measure* der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multilabel-Klassifikations Datensätzen unter der Verwendung eines Entscheidungsbaumes mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests

Datensatz	*Micro F1-Measure* normale LOO-Kreuzvalidierung	*Micro F1-Measure* spezielle LOO-Kreuzvalidierung	t-Wert	Sicherheit, dass Gleichheit nicht zutrifft
Corel5k	0,2046±0,0009	0,2045±0,0023	-0,1312	<50%
Flags	0,7023±0,0110	0,7026±0,0122	0,0627	<50%
Mediamill	0,5245±0,0004	0,5227±0,0030	-1,8434	90%
Scene	0,4051±0,0098	0,3979±0,0176	-1,1315	75%
Yeast	0,5779±,00360	0,5805±0,0081	0,9401	50%
TMC2007	0,4163±0,0003	0,4164±0,0010	0,1911	<50%

Tabelle 6.11: Vergleich der Metrik *Macro F1-Measure* der normalen LOO-Kreuzvalidiuerng und der speziellen LOO-Kreuzvalidierung auf den Multilabel-Klassifikations Datensätzen unter der Verwendung eines Entscheidungsbaumes mit einer maximalen Tiefe von 10 und einer minimalen Anzahl von 15 Instanzen für die Erstellung eines Tests

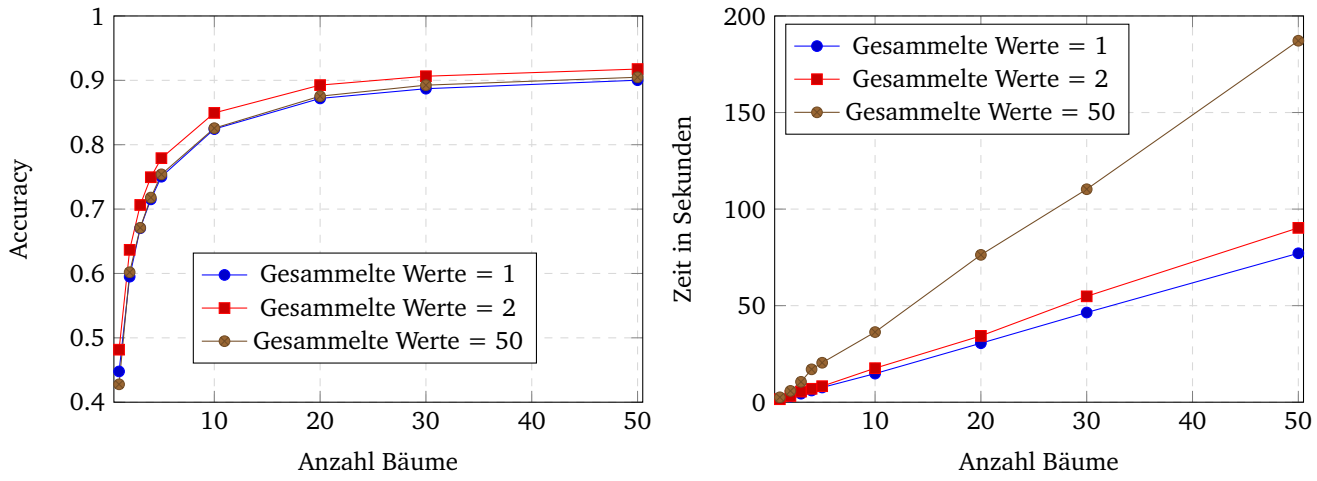
Datensatz	*Macro F1-Measure* normale LOO-Kreuzvalidierung	*Macro F1-Measure* spezielle LOO-Kreuzvalidierung	t-Wert	Sicherheit, dass Gleichheit nicht zutrifft
Corel5k	0,0081±0,0003	0,0076±0,0011	-1,3485	75%
Flags	0,5721±0,0157	0,5794±0,0292	0,7059	<50%
Mediamill	0,0542±0,0006	0,0531±0,0031	-1,0941	50%
Scene	0,4092±0,0098	0,3993±0,0205	-1,3637	75%
Yeast	0,3234±0,0027	0,3242±0,0126	0,1921	<50%
TMC2007	0,0561±0,0004	0,0561±0,0033	0,0869	<50%

6.6 Unterschiedliche Parametereinstellungen für den Online-Algorithmus

In diesem Versuch werden unterschiedliche Parametereinstellungen für den Algorithmus für die Erstellung der Entscheidungsbäume analysiert, der unter der Annahme des Online-Lernens arbeitet. Allgemein lässt sich in den Versuchen für alle Datensätze die Tendenz erkennen, dass mit zunehmender Anzahl an Entscheidungsbäume im Ensemble sich die

Werte der Evaluationsmetriken verbessern. Auch bei einer zunehmenden Tiefe der Entscheidungsbäume werden die Ergebnisse fast aller Datensätze besser. Die einzige Ausnahme bildet hier der Datensatz *Eye State*. Bei diesem Datensatz werden die besten Ergbenisse bei einer maximalen Höhe von sechs erreicht. Die Anzahl der gesammelten Werte in den Quantil-Tests haben nur eine sehr geringe Auswirkung auf die Güte der Entscheidungsbäume. In vielen Fällen werden die besten Ergebnisse bei einer Anzahl von gesammelten Werten von eins oder zwei erreicht. Bei Datensätzen, die nur nominale Attribute besitzen, hat die Anzahl der gesammelten Werte keine Auswirkung auf die Güte, weil keine numerischen Tests verwendet werden. Diese Erkenntnisse werden anhand von ausgewählten Versuchen in den nächsten Absätzen noch einmal genauer unterlegt.

Für die genauere Analyse verwenden wir zuerst den Datensatz *Letter Recognition*. In Abbildung 6.3a ist die *Accuracy* in Abhängigkeit von der Anzahl der Entscheidungsbäume im Ensemble dargestellt. In dieser Grafik sind drei Kurven eingezeichnet. Jede Kurve entspricht hierbei einer definierten Anzahl an gesammelten Werten in den numerischen Tests. Durch diese Abbildung ist zu erkennen, dass die *Accuracy* für den Datensatz *Letter Recognition* mit zunehmender Anzahl an Entscheidungsbäumen im Ensemble steigt. Hierbei ist diese Tendenzen für jede Anzahl an gesammelten Werten gleich. Auch bei den Versuchen mit einer zunehmenden maximalen Tiefe der Entscheidungsbäume ist aufgefallen, dass die Güte der Entscheidungsbäume, unabhängig von der Anzahl der gesammelten Werte, mit einer zunehmenden Tiefe steigt. Der einzige Unterschied entsteht nur, wenn wir uns die benötigte Zeit für die Erstellung und der Evaluierung der Modelle vergleichen. In Abbildung 6.3b ist die für den Aufbau und die Evaluierung verwendete Zeit dargestellt. In dieser Grafik lässt sich erkennen, dass deutlich mehr Zeit benötigt wird, wenn mehr Werte in den numerischen Tests gesammelt werden müssen. Dieses Verhalten entsteht, weil mit dem Hinzufügen eines Wertes zu einem numerischen Test, immer der Trennwert neu berechnet werden muss. Hierbei müssen die gesammelten Werte sortiert und anschließend der entsprechende Trennwert aus der Liste ausgewählt werden. Zusätzlich verbraucht das Sammeln der Werte Speicherplatz, was einen weiteren Nachteil darstellt. Hieraus folgt, dass das Sammeln von mehreren Werten deutlich mehr Zeit kostet, aber bei diesen Parametereinstellungen keinen Vorteil bringt. Es ist sogar das Gegenteil der Fall. In Abbildung 6.3a lässt sich deutlich erkennen, dass die besten Ergebnisse für die verwendeten Parametereinstellungen erzielt werden, wenn nur zwei Werte gesammelt werden.



(a) Vergleich der Metrik *Accuracy* des Online-Algorithmus bei einer unterschiedlichen Anzahl von Entscheidungsbäumen mit einer maximalen Tiefe von 15 für den Datensatz *Letter Recognition*

(b) Vergleich der benötigten Zeit des Online-Algorithmus bei einer unterschiedlichen Anzahl von Entscheidungsbäumen mit einer maximalen Tiefe von 15 für den Datensatz *Letter Recognition*

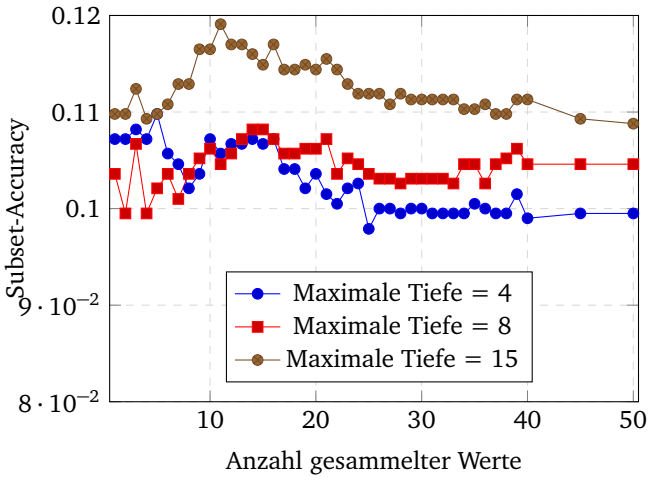
Abbildung 6.3

Die Tendenz, dass mit zunehmender Tiefe und mit zunehmender Anzahl an Entscheidungsbäumen die Güte der Entscheidungsbäume steigt, ist bei allen Datensätzen zu erkennen. Bei der Variation der Anzahl von gesammelten Werten gibt es jedoch zwischen manchen Datensätzen Unterschiede. Aus diesem Grund wird auf die Variation der Anzahl von gesammelten Werten noch einmal genauer eingegangen.

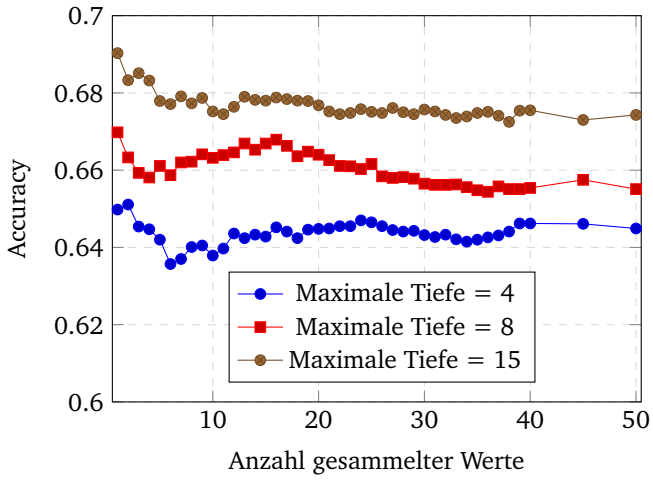
Bei der Analyse der Datensätze ist aufgefallen, dass die Variation der gesammelten Werte nur einen sehr geringen Einfluss auf die Evaluationsergebnisse der Datensätze *Yeast*, *Mediamill*, *Eye State*, *Airline* und *Thyroid Disease* hat. Für die Datensätze *Flags*, *Electricity*, *Letter Recognition* und *Scene* lässt sich jedoch eine deutlichere Abweichung feststellen. Hierbei ist aufgefallen, dass die Anzahl der gesammelten Werten in Zusammenhang mit der maximalen Tiefe des Entscheidungsbaumes steht. In Abbildung 6.4 sind deswegen die Evaluationsergebnisse für diese Datensätze in Abhängigkeit der Anzahl der gesammelten Werte abgebildet. Hierbei wurde für jeden Datensatz drei Kurven in die Grafik eingezeich-

net. Durch jede Kurve wird hierbei die maximale Tiefe des Entscheidungsbaumes definiert. Alle abgebildeten Versuche wurden mit einem Entscheidungsbaum durchgeführt. Bis auf die Evaluationsergebnisse des Datensatzes *Flags* haben alle Datensätze dieselben Tendenzen. So fällt zum Beispiel auf, dass bei den Versuchen mit einer maximalen Tiefe von acht oder 15 die besten Ergebnisse erzielt werden, wenn nur ein oder zwei Werte in den numerischen Tests gesammelt werden. Auch bei einer maximalen Tiefe von vier lässt sich bei dem Datensatz *Electricity* dieses Verhalten erkennen. Anders sieht das bei den Datensätzen *Letter Recognition* und *Scene* aus. Für die Versuche auf dem Datensatz *Letter Recognition* mit einer maximalen Tiefe von vier lässt sich feststellen, dass die *Accuracy* mit zunehmender Anzahl an gesammelten Werten steigt. Auch bei dem Datensatz *Scene* lässt sich feststellen, dass für diese maximale Tiefe bessere Ergebnisse erzielt werden können, wenn mehrere Werte gesammelt werden.

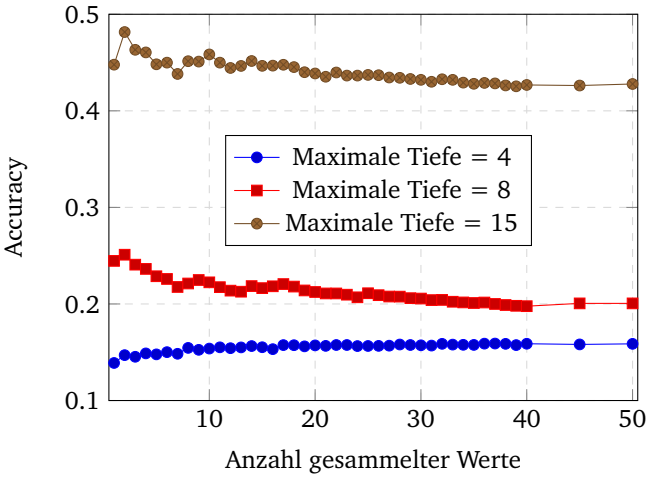
Der Datensatz *Flags* stellt genau das Gegenteil dieser Tendenzen dar. Bei diesem Datensatz hat die Variation der gesammelten Werte einen viel größeren Einfluss auf den Entscheidungsbaum, wenn dieser eine maximale Tiefe von 15 hat. Hierbei lässt sich deutlich erkennen, das die besten Ergebnisse mit 11 gesammelten Werten erreicht wird. Auch bei einer maximalen Tiefe von acht lässt sich ein Zuwachs der *Subset-Accuracy* mit zunehmender Anzahl an gesammelten Werten erkennen. Hier werden die besten Ergebnisse bei 14 gesammelten Werten erreichen. Bei einer maximalen Tiefe müssen nur fünf Werte gesammelt werden, um die besten Ergebnisse zu erreichen. An dieser Stelle ist jedoch zu Hinterfragen, welche Aussagekräftigkeit die Versuche auf dem Datensatz *Flags* haben, weil dieser Datensatz nur aus 194 Instanzen besteht.



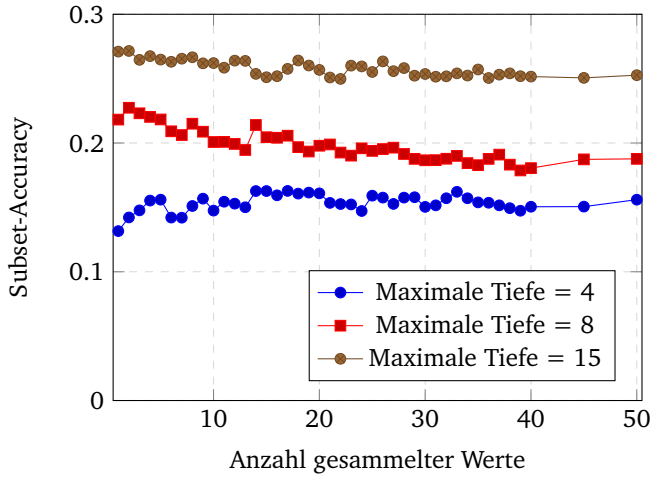
(a) Metrik *Subset-Accuracy* für unterschiedliche maximale Tiefen eines Entscheidungsbaumes bei einer unterschiedlichen Anzahl von gesammelten Werten für den Datensatz *Flags*



(b) Metrik *Accuracy* für unterschiedliche maximale Tiefen eines Entscheidungsbaumes bei einer unterschiedlichen Anzahl von gesammelten Werten für den Datensatz *Electricity*



(c) Metrik *Accuracy* für unterschiedliche maximale Tiefen eines Entscheidungsbaumes bei einer unterschiedlichen Anzahl von gesammelten Werten für den Datensatz *Letter Recognition*



(d) Metrik *Subset-Accuracy* für unterschiedliche maximale Tiefen eines Entscheidungsbaumes bei einer unterschiedlichen Anzahl von gesammelten Werten für den Datensatz *Scene*

Abbildung 6.4

6.7 Vergleich des Online-Algorithmus im Online-Modus mit dem Online-Algorithmus im Batch-Modus

In diesem Versuch wird evaluiert, welche Auswirkungen das Sammeln der Werte mit dem Online-Algorithmus hat. Um ein solches Szenario testen zu können, muss das Batch-Lernen des Online-Algorithmus imitiert werden. Hierfür iterieren wir zweimal über den Datensatz. Zuerst werden die Instanzen genutzt, um die Entscheidungsbäume aufzubauen. In diesem Fall werden die Instanzen den Blättern nicht hinzugefügt, wodurch es möglich ist, dass nur die Tests in den inneren Knoten die Werte sammeln können. Anschließend wird in der zweiten Iteration die Instanzen allen Entscheidungsbäumen übergeben. Hierbei werden die Instanzen über den Wurzelknoten einem Blatt zugeordnet, an dessen Kollektoren diese übergeben werden. Die Entscheidungsbäume werden also erst nach der Erstellung und dem Sammeln der Werte mit den Daten trainiert. Durch diese Imitation wird vermieden, dass die Instanzen falschen Blattknoten zugewiesen werden, weil sich die Trennwerte der Tests bei der zweiten Iteration nicht mehr ändern. Für diesen Versuch werden die Ergebnisse des Online-Algorithmus im Batch-Modus mit den Ergebnissen des Online-Algorithmus im Online-Modus verglichen. Hierfür werden in jedem Versuch dieselben Entscheidungsbäume aufgebaut, jedoch mit dem Unterschied, dass im Batch-Modus zweimal über die Daten iteriert wird. Somit kann mit diesem Versuch getestet werden, welche Auswirkung die falsche Zuordnung von Instanzen durch sich ändernde Trennwerte während des Online-Lernens hat.

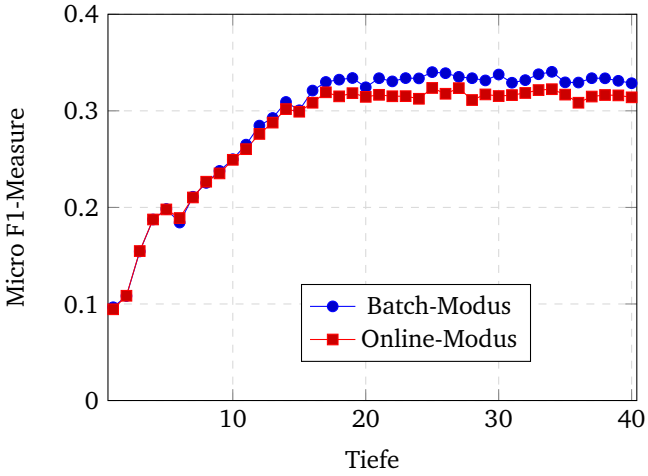
Tabelle 6.12: Evaluationsergebnisse für den Datensatz *Thyroid Disease*

Datensatz	Anzahl Bäume	Maximale Tiefe	Maximale Anzahl gesammelter Werte	Batch-Modus	Online-Modus
Thyroid Disease	1	1	20	0,6009±0,0026	0,6009±0,0026
Thyroid Disease	1	5	20	0,6073±0,0068	0,6080±0,0073
Thyroid Disease	1	10	20	0,6197±0,0033	0,6100±0,0103
Thyroid Disease	1	20	20	0,6406±0,0067	0,6187±0,0168
Thyroid Disease	1	25	20	0,6484±0,0081	0,6250±0,0155
Thyroid Disease	1	35	20	0,6420±0,0062	0,6063±0,0090
Thyroid Disease	1	8	1	0,6147±0,0052	0,6126±0,0049
Thyroid Disease	1	8	10	0,6155±0,0051	0,6108±0,0052
Thyroid Disease	1	8	20	0,6144±0,0062	0,6087±0,0057
Thyroid Disease	1	8	50	0,6152±0,0063	0,6089±0,0064
Thyroid Disease	1	8	75	0,6151±0,0066	0,6095±0,0058
Thyroid Disease	1	8	100	0,6146±0,0065	0,6101±0,0069
Thyroid Disease	1	8	150	0,6140±0,0073	0,6094±0,0095
Thyroid Disease	1	8	20	0,6144±0,0062	0,6087±0,0057
Thyroid Disease	10	8	20	0,6235±0,0055	0,6231±0,0051
Thyroid Disease	25	8	20	0,6185±0,0053	0,6191±0,0042
Thyroid Disease	50	8	20	0,6155±0,0034	0,6168±0,0037
Thyroid Disease	100	8	20	0,6133±0,0023	0,6169±0,0027

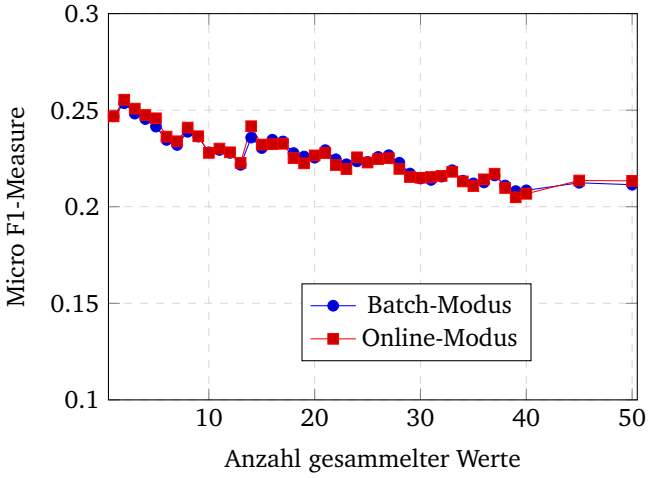
In den Versuche ist aufgefallen, dass es zwischen dem Batch-Lernen und dem Online-Lernen des Online-Algorithmus kaum Unterschiede gibt. Bei den Datensätzen, die nur nominale Attribute verwenden, werden bei beiden Lernvarianten dieselben Ergebnisse erzielt, weil keine numerischen Tests verwendet werden. Folglich kann sich kein Trennwert über die Zeit verändern, wodurch keine Instanz im Online-Modus einem falschen Blattknoten zugewiesen werden kann. Aus diesem Grund werden hier nur die Datensätze mit numerischen Attributen genauer analysiert. In Tabelle 6.12 sind die Evaluationsergebnisse des Datensatzes *Thyroid Disease* abgebildet. Obwohl dieser Datensatz nur sehr wenige numerische Attribute verwendet, ist deutlich zu erkennen, dass mit zunehmender Tiefe eines Entscheidungsbaum die *Accuracy* im Online-Modus nicht so stark zunimmt, wie die *Accuracy* im Batch-Modus. Ausschlaggebend hierfür ist, dass den numerischen Tests, die sehr Tief in dem Entscheidungsbaum liegen, nur selten Instanzen zugewiesen werden. Dadurch ändert sich der Trennwert dieser Tests auch noch, wenn sehr viele Instanzen mit dem Baum verarbeitet wurden. Daraus folgt, dass mehr Instanzen beim Lernen falschen Blattknoten zugewiesen werden, wodurch die Güte sinkt. Interessant ist, dass die Anzahl der gesammelten Werte in den Quantil-Tests nur minimale Auswirkungen auf die Güte der Entscheidungsbäume im Online-Modus hat. Auch bei der Variation der Anzahl der Bäume unterscheiden sich die Ergebnisse für diesen Datensatz im Batch-Modus kaum von den Ergebnissen im Online-Modus.

Diese Tendenzen sind bei fast allen Versuchen auf den anderen Datensätze, die numerische Attributen enthalten, ebenfalls für jede Metriken zu erkennen. Die einzige Ausnahme bildet der Datensatz *Scene*. Für diesen Datensatz entsteht auch bei zunehmender Anzahl an Entscheidungsbäume im Ensemble ein Unterschied zwischen den Evaluationsergebnissen im Batch-Modus gegenüber den Evaluationsergebnissen im Online-Modus. In Abbildung 6.5 sind die Ergebnisse des Datensatzes *Scene* für diesen Versuch dargestellt. Zusätzlich sind in Abbildung 6.6 die Ergebnisse des Datensatzes *Letter*

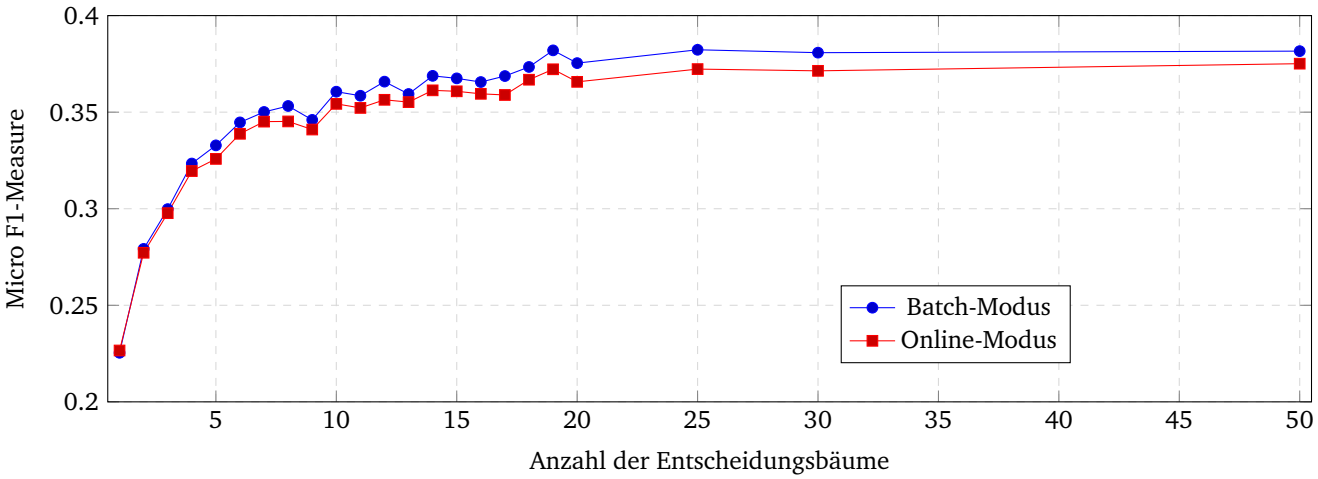
Recognition abgebildet, um noch einmal zu verdeutlichen, dass bei fast allen Datensätze nur bei der Variation der maximalen Tiefe ein Unterschied in den Evaluationsergebnissen entsteht. An dieser Stelle ist noch einmal hervorzuheben, dass bei der Variation der gesammelten Werten in den numerischen Tests, es nur vernachlässigbar kleine Unterschiede in Ergebnissen zwischen dem Batch-Modus und dem Online-Modus gibt.



(a) Parametereinstellung: 50 Entscheidungsbäume mit unterschiedlichen maximalen Tiefe und einer maximalen Anzahl von 20 gesammelten Werten

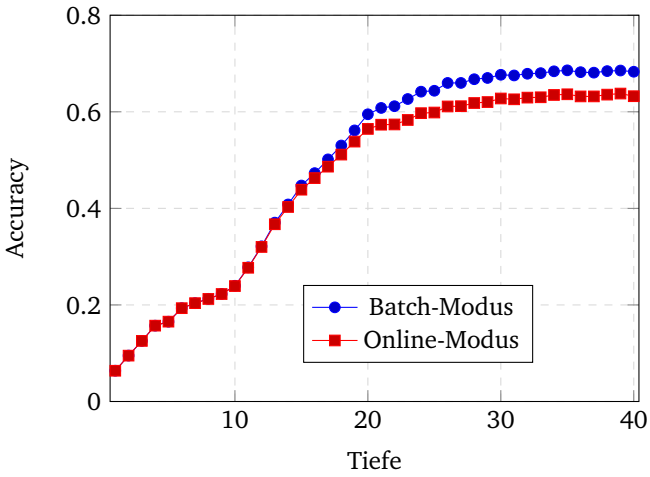


(b) Parametereinstellung: 50 Entscheidungsbäume mit einer maximalen Tiefe von 8 und einer unterschiedlichen maximalen Anzahl von gesammelten Werten

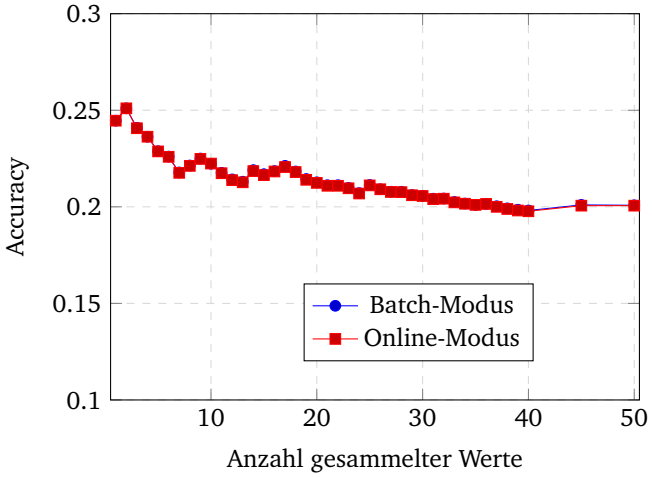


(c) Parametereinstellung: Unterschiedliche Anzahl an Entscheidungsbäume mit einer maximalen Tiefe von 8 und einer maximalen Anzahl von 20 gesammelten Werten

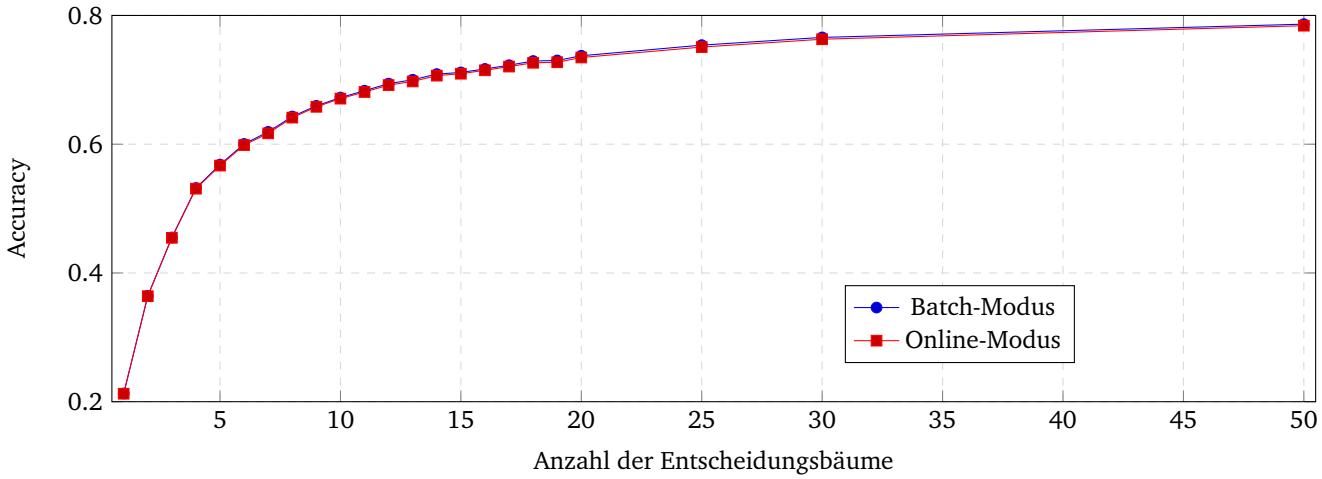
Abbildung 6.5: Vergleich der Metrik *Micro F1-Measure* des Online-Algorithmus im Batch-Modus mit dem Online-Algorithmus im Online-Modus für den Datensatz *Scene*



(a) Parametereinstellung: 50 Entscheidungsbäume mit unterschiedlichen maximalen Tiefen und einer maximalen Anzahl von 20 gesammelten Werten



(b) Parametereinstellung: 50 Entscheidungsbäume mit einer maximalen Tiefe von 8 und einer unterschiedlichen maximalen Anzahl von gesammelten Werten



(c) Parametereinstellung: Unterschiedliche Anzahl an Entscheidungsbäume mit einer maximalen Tiefe von 8 und einer maximalen Anzahl von 20 gesammelten Werten

Abbildung 6.6: Vergleich der Metrik *Accuracy* des Online-Algorithmus im Batch-Modus mit dem Online-Algorithmus im Online-Modus für den Datensatz *Letter Recognition*.

6.8 Vergleich des Online-Algorithmus mit dem Hoeffding-Tree

In diesem Versuch wird der erstellte Online-Algorithmus mit dem Hoeffding-Tree verglichen. Leider konnte der Datensatz *TMC2007* bei diesem Experiment nicht verwendet werden, weil bei der Verarbeitung dieses Datensatzes der Hoeffding-Tree Fehler geworfen hat. Für die Evaluierung des Hoeffding-Trees wurden die Standardparametereinstellungen verwendet. Um die beiden Algorithmen miteinander vergleichen zu können, wurde sich an der Laufzeit des Hoeffding-Trees orientiert. Für den Online-Algorithmus wurden dabei die Parametereinstellungen so gewählt, dass dieser die besten Ergebnisse bei ungefähr gleicher Laufzeit erzielt. Wegen einiger Probleme bei der Verwendung des Multilabel-Hoeffding-Tree wurde für die Evaluierung auf den Multilabel-Klassifikations-Datensätze für jede zu vorhersagende Klasse ein Hoeffding-Tree verwendet. Das bedeutet, dass das Problem der Multilabel-Klassifikation mit n Klassen in n binäre Klassifikationsprobleme aufgeteilt wird. Für jede binäre Klassifikation wird ein Hoeffding-Tree verwendet, der unabhängig von den Ergebnissen der anderen Klassen arbeitet.

Tabelle 6.13: Vergleich der Evaluationsergebnisse für die Multiklassen-Klassifikations-Datensätze

Datensatz	Laufzeit Online-Algorithmus (Sekunden)	Laufzeit Hoeffding Tree	Accuracy Online-Algorithmus (Sekunden)	Accuracy Hoeffding Tree
Airline	290,861	330,819	0,6364±0,0029	0,6380
Electricity	8,316	10,214	0,7009±0,0110	0,7245
EEG Eye State	1,536	6,110	0,5762±0,0096	0,5695
Letter Recognition	59,364	75,268	0,9175±0,0023	0,6426
Mushroom	2,130	2,393	0,9981±0,0024	0,9950
Thyroid Disease	0,784	0,863	0,6236±0,0063	0,5989

In Tabelle 6.13 sind die Evaluationsergebnisse der Versuche auf den Multiklassen-Klassifikations-Datensätzen abgebildet. Bei den Datensätzen *Airline* und *Electricity* konnte der Hoeffding-Tree in der gegebenen Zeit bessere Ergebnisse erzielen. Auf den anderen Multiklassen-Klassifikations-Datensätzen konnten mit dem Online-Algorithmus bessere Ergebnisse erreicht werden. Besonders auffällig ist der Versuch auf dem Datensatz *Letter Recognition*. Die Laufzeit des Hoeffding-Trees für diesen Datensatz ist bemerkenswert hoch. Dieses Verhalten lässt sich wahrscheinlich durch die hohe Anzahl an vorherzusagenden Klassen erklären. Aus diesem Grund konnten für die Parametereinstellungen des Online-Algorithmus außergewöhnliche Werte angenommen werden. Hierbei war es möglich, 50 Entscheidungsbäume mit einer maximalen Tiefe von 15 und einer minimalen Anzahl von zwei Instanzen für die Erstellung eines Tests zu verwenden. Trotz der umfangreichen Parametereinstellung arbeitet der Online-Algorithmus in diesem Fall sogar schneller und erzielt deutlich bessere Ergebnisse als der Hoeffding-Tree. Auch auf dem Datensatz *Eye State* lassen sich bei kürzerer Laufzeit bessere Ergebnisse mit dem Online-Algorithmus erzielen. Hieraus lässt sich wahrscheinlich schließen, dass der Online-Algorithmus für Datensätze, die viele numerische Attribute enthalten, besser abschneidet als der Hoeffding-Tree.

In Tabelle 6.14 werden die Laufzeiten der beiden Algorithmen für die Multilabel-Klassifikations-Datensätze dargestellt. Aufgrund der Umwandlung des Problems der Multilabel-Klassifikation in mehrere binäre Klassifikationsprobleme, braucht der Hoeffding-Tree deutlich mehr Zeit, um die Datensätze zu verarbeiten. Die Evaluationsergebnisse sind in den Tabellen 6.15, 6.16, 6.17 und 6.18 abgebildet. Daraus lässt sich erkennen, dass sich die Ergebnisse für fast jeden Datensatz deutlich unterscheiden. Auf den Datensätzen *Corel5k*, *Mediamill* und *Yeast* werden bessere Ergebnisse mit dem Hoeffding-Tree erzielt, während auf dem Datensatz *Scene* der Online-Algorithmus die besseren Ergebnisse liefert. Auf dem Datensatz *Flags* werden durch beide Algorithmen ungefähr dieselben Ergebnisse erzielt, bis auf den Wert der Metrik *Macro Precision*. Bei dieser Metrik wird durch den Online-Algorithmus ein besserer Wert erreicht als durch den Hoeffding-Tree. An dieser Stelle ist jedoch auch zu hinterfragen, welche Aussagekraft der Vergleich auf den Multilabel-Klassifikations-Datensätze hat, weil unterschiedliche Verfahren für die Multilabel-Klassifikation verwendet wurden. Bei dem Online-Algorithmus wurde der Kollektor für die Multilabel-Klassifikation verwendet, hingegen bei dem Hoeffding-Tree das Problem auf mehrere binäre Klassifikationen aufgeteilt wurde. Aufgrund dieser Fakten unterscheiden sich die Laufzeiten der beiden Algorithmen enorm und die Ergebnisse lassen sich deswegen nur schwer vergleichen. Generell lässt sich die Aussage treffen, dass der Online-Algorithmus zu bevorzugen ist, wenn die Laufzeit eine wichtige Rolle spielt. Ist man mit den Ergebnissen des Online-Algorithmus nicht zufrieden, so besteht die Möglichkeit, dass durch die Verwendung des Hoeffding-Tree bessere Ergebnisse erzielt werden können.

Tabelle 6.14: Vergleich der Laufzeit für die Multilabel-Klassifikations-Datensätze

Datensatz	Laufzeit Online-Algorithmus (Sekunden)	Laufzeit Hoeffding Tree (Sekunden)
Corel5k	37,308	18675,150
Flags	0,122	1,415
Mediamill	69,515	12276,878
Scene	9,729	104,754
Yeast	6,307	112,567

Tabelle 6.15: Vergleich der Evaluationsergebnisse der Metriken *Subset-Accuracy* und *Hamming Loss* für die Multilabel-Klassifikations-Datensätze

Datensatz	Subset-Accuracy Online-Algorithmus	Subset-Accuracy Hoeffding Tree	Hamming Loss Online-Algorithmus	Hamming Loss Hoeffding Tree
Corel5k	0,0004±0,0001	0,0054	0,0159±0,0000	0,0104
Flags	0,1495±0,0069	0,1340	0,2485±0,0037	0,2482
Mediamill	0,0147±0,0003	0,1433	0,0396±0,0001	0,0204
Scene	0,4373±0,0070	0,1446	0,1823±0,0025	0,2113
Yeast	0,1677±0,0029	0,4063	0,2159±0,0009	0,0716

Tabelle 6.16: Vergleich der Evaluationsergebnisse der Metriken *Micro Precision* und *Micro Recall* für die Multilabel-Klassifikations-Datensätze

Datensatz	Micro Precision Online-Algorithmus	Micro Precision Hoeffding Tree	Micro Recall Online-Algorithmus	Micro Recall Hoeffding Tree
Corel5k	0,1914±0,0004	0,3468	0,2138±0,0004	0,1132
Flags	0,7443±0,0045	0,7473	0,7421±0,0041	0,7371
Mediamill	0,5423±0,0010	0,8317	0,5443±0,0009	0,6624
Scene	0,4902±0,0074	0,3875	0,4581±0,0069	0,3106
Yeast	0,6502±0,0016	0,8867	0,6206±0,0017	0,8753

Tabelle 6.17: Vergleich der Evaluationsergebnisse der Metriken *Macro Precision* und *Macro Recall* für die Multilabel-Klassifikations-Datensätze

Datensatz	Macro Precision Online-Algorithmus	Macro Precision Hoeffding Tree	Macro Recall Online-Algorithmus	Macro Recall Hoeffding Tree
Corel5k	0,0053±0,0002	0,0729	0,0108±0,0000	0,0366
Flags	0,6206±0,0345	0,5334	0,5798±0,0033	0,5785
Mediamill	0,1911±0,0162	0,3520	0,0646±0,0006	0,2552
Scene	0,5132±0,0096	0,2569	0,4721±0,0070	0,341
Yeast	0,4449±0,0704	0,7996	0,3570±0,0014	0,7667

Tabelle 6.18: Vergleich der Evaluationsergebnisse der Metriken *Micro F1-Measure* und *Macro F1-Measure* für die Multilabel-Klassifikations-Datensätze

Datensatz	Micro F1-Measure Online-Algorithmus	Micro F1-Measure Hoeffding Tree	Macro F1-Measure Online-Algorithmus	Macro F1-Measure Hoeffding Tree
Corel5k	0,2020±0,0003	0,1707	0,0041±0,0000	0,0370
Flags	0,7432±0,0037	0,7422	0,5568±0,0038	0,5474
Mediamill	0,5433±0,0009	0,7374	0,0606±0,0011	0,2757
Scene	0,4736±0,0071	0,3449	0,4837±0,0078	0,2913
Yeast	0,6350±0,0016	0,8810	0,3285±0,0016	0,7794

7 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein modulares Programm entwickelt, mit dem Ensembles von Entscheidungsbäumen erstellt werden können. Die Schnittstelle für die Erstellung der Entscheidungsbäume wurde so entworfen, dass der Algorithmus für die Erstellung der Entscheidungsbäume beliebig ausgetauscht werden kann. Um die Kompatibilität mit dem Programm gewährleisten zu können, wurde eine spezielle Struktur der Entscheidungsbäume festgelegt. Mit der Verwendung von Kollektoren wird die Möglichkeit geboten, mehrere Lernaufgaben durch einen einzigen Entscheidungsbaum übernehmen zu können.

Darüber hinaus wurden zwei Algorithmen für die Erstellung der Entscheidungsbäume in dieser Arbeit entwickelt. Der erste Algorithmus wurde für das Batch-Lernen entworfen. Für die erstellten Entscheidungsbäume dieses Algorithmus wurde ein Verfahren entwickelt, mit dem eine schnelle Kreuzvalidierung durchgeführt werden kann. Hierbei hat sich herausgestellt, dass sich die Leave-One-Out-Kreuzvalidierung am besten für dieses Verfahren eignet. In den Versuchen stellte sich heraus, dass die Evaluierung mit dem entwickelten Verfahren deutlich schneller durchgeführt werden kann und die Evaluationsergebnisse annähernd den Evaluationsergebnissen der echten Leave-One-Out-Kreuzvalidierung entsprechen. Des Weiteren wurde der Batch-Algorithmus mit dem Dice-Algorithmus verglichen. Hierbei konnte beobachtet werden, dass der Batch-Algorithmus Datensätze mit vielen numerischen Attributen schneller verarbeiten kann als der Dice-Algorithmus. Einen weiteren Vorteil gegenüber dem Dice-Algorithmus hat der Batch-Algorithmus auf den Multilabel-Klassifikations-Datensätzen, bei denen viele Klassen vorhergesagt werden sollen.

Der zweite entworfene Algorithmus für die Erstellung der Entscheidungsbäume wurde für das Online-Lernen entworfen. Die Besonderheit dieses Algorithmus liegt in der Verwendung von Quantilen für die Bestimmung der Trennwerte in den numerischen Tests des Entscheidungsbaumes. Hierbei werden in den Tests Mengen von Werten verwaltet, anhand derer der Trennwert bestimmt werden kann. Diese Mengen werden erst beim inkrementellen Aufbau der Entscheidungsbäume befüllt, wodurch sich der Trennwert mit jeder Aktualisierung ändern kann. Zusätzlich wurden die Entscheidungsbäume, die durch diesen Algorithmus erstellt werden, so konzipiert, dass sie unter bestimmten Voraussetzungen mit wenig Aufwand aggregiert werden können. In Bezug auf diese Eigenschaft wurden unterschiedliche Verfahren diskutiert, mit denen die Aggregation in Kombination mit einer Kreuzvalidierung durchgeführt werden kann. In den Versuchen stellte sich heraus, dass die Änderung der Trennwerte nur sehr minimale Auswirkungen auf die Güte der Entscheidungsbäume hat. Des Weiteren wurde in den Versuchen erkannt, dass für viele Datensätze die besten Evaluationsergebnisse erzielt werden, wenn nur eine geringe Anzahl an Werten in den Mengen der Tests gesammelt werden. Beim Vergleich mit dem Hoeffding-Tree ließ sich erkennen, dass der entworfene Online-Algorithmus deutlich besser auf Mutiklassen-Klassifikations-Datensätzen abschneidet, die viele numerische Attribute beinhalten und bei denen viele Klassen vorhergesagt werden müssen. Aufgrund des eingegrenzten Zeitrahmens der Bachelorarbeit konnten keine Versuche für die unterschiedlichen Verfahren der Aggregation gemacht werden.

Das in dieser Arbeit entworfene Programm kann in zukünftigen Projekten verwendet werden, um neue Ideen für die Erstellung von Entscheidungsbäumen auszuprobieren und deren Tauglichkeit testen zu können. So lässt sich beispielsweise ein neuer Algorithmus entwickeln, der sich besonders gut für die Verarbeitung von Datensätzen mit spärlichen Daten eignet. Des Weiteren wird die Möglichkeit geboten, eigene Tests und eigene Kollektoren zu entwickeln, die sich für die gegebene Problemstellung besonders gut eignen. Als offene Frage dieser Arbeit bleibt bestehen, welche Auswirkungen die Aggregation der Ensembles des Online-Algorithmus auf die Güte der Entscheidungsbäume hat. Auch die unterschiedlichen Verfahren, bei denen eine Aggregation in Kombination mit einer Kreuzvalidierung eingesetzt werden kann, konnten in dieser Arbeit nicht evaluiert werden. Des Weiteren existieren unterschiedliche Varianten des Hoeffding-Trees (z.B. Random-Hoeffding-Tree), die mit dem Online-Algorithmus verglichen werden können. Diese Punkte können in zukünftigen Arbeiten genauer untersucht und bearbeitet werden.

Literaturverzeichnis

A. Andrzejak, F. Langner, and S. Zabala. Interpretable Models from Distributed Data via Merging of Decision Trees. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 1–9, April 2013.

Michael Falk, Johannes Hain, Frank Marohn, Hans Fischer, and René Michel. *Statistik in Theorie und Praxis*. Springer Berlin Heidelberg, 2014.

Geoffrey Holmes und Bernhard Pfahringer Jesse Read, Albert Bifet. Efficient Multi-label Classification for Evolving Data Streams. *Computer Science Working Papers*, April 2010.

Bassem Khouzam. Incremental Decision Trees. Master's thesis, Orange Labs, September 2009.

Xiangnan Kong and Philip S. Yu. An Ensemble-based Approach to Fast Classification of Multi-label Data Streams. In Dimitrios Georgakopoulos and James B. D. Joshi, editors, *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 95–104, 2011.

A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line Random Forests. In *IEEE 12th International Conference on Computer Vision Workshops*, pages 1393–1400, September 2009.

Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences*, pages 1947–1958, 2003.

Grigorios Tsoumakas and Ioannis Vlahavas. Random k-Labelsets: An Ensemble Method for Multilabel Classification. In JoostN. Kok, Jacek Koronacki, RaomonLopezde Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML*, volume 4701 of *Lecture Notes in Computer Science*, pages 406–417. 2007.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining Multi-label Data. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 667–685. Springer US, 2010.

G.I. Webb. Contrary to Popular Belief Incremental Discretization can be Sound, Computationally Efficient and Extremely Useful for Streaming Data. In *IEEE International Conference on Data Mining*, pages 1031–1036, Dezember 2014.

Xiatian Zhang, Quan Yuan, Shiwan Zhao, Wei Fan, Wentao Zheng, and Zhong Wang. Multi-label Classification without the Multi-label Cost. In *Proceedings of the Tenth SIAM International Conference on Data Mining*, 2010.