Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Analyse von datengetriebenen Verfahren zur Erkennung falsch dokumentierter Klassenlabel

Gabriel Bonnet

Studiengang:	Softwaretechnik
Prüfer/in:	Dr. Holger Schwarz
Betreuer/in:	M. Sc. Dennis Tschechlov
Beginn am:	1. April 2021
Beendet am:	1. Oktober 2021

Kurzfassung

Über die letzten Jahrzente hinweg lässt sich eine stetige Zunahme gesammelter Daten verschiedenster Herkunft beobachten. Diese sind beispielsweise durch technologische Entwicklungen im Zusammenhang mit dem IoT entstanden. Damit diese Daten genutzt werden können, werden sie mithilfe datengetriebener Verfahren des Maschinellen Lernens ausgewertet. Solche Daten sind für eine große Anzahl industrieller Wertschöpfungen relevant, denn mithilfe von Datenanalysen lassen sich beispielsweise im Bereich des Qualitätsmanagements Ursachen für fehlerhafte Vorgänge und die daraus resultierenden fehlerhaften Produkte bestimmen. Die gesammelten Daten bringen jedoch zahlreiche Herausforderungen mit sich. Zu diesen zählen zum Beispiel eine Klassenungleichverteilung, Label Noise und wenige Datenpunkte je Klasse, welche zu einer mäßigen Vorhersagegenauigkeit herkömmlicher Klassifikatoren führen. Wie schon das Ergebnis der Arbeit von Hirsch et. al. [HRM19] zeigt, schneidet der Random Forest Klassifikator unter den gegebenen Herausforderungen und somit auch im Hinblick auf Label Noise am besten ab.

Ziel dieser Arbeit ist es demnach unterschiedliche datengetriebene Verfahren zur Erkennung von Label Noise in Kombination mit den genannten Herausforderungen zu untersuchen. Dabei wird speziell die Datenvorbereitung betrachtet und es wird untersucht, ob diese Verfahren gut mit den Herausforderungen umgehen können. Methoden zur Erkennung von Label Noise, auch Detektoren genannt, berechnen für jeden Datenpunkt aus einem Datensatz einen Konfidenzwert oder eine Wahrscheinlichkeit, ob dieser Datenpunkt richtig gelabelt wurde. Dafür nutzen die Detektoren unterschiedliche Algorithmen, wobei der Fokus dieser Arbeit auf fünf Detektoren, KDN, Instance-Hardness, Partitioning-Detektor, Random Forest Detektor und MCS, liegt. Um die Detektoren zu vergleichen, werden unterschiedliche Szenarien genutzt, da es nach dem Anwenden der Detektoren zwei Möglichkeiten gibt: Entweder können Datenpunkte gelöscht oder nach dem Konfidenzwert gewichtet werden. Zur Evaluation der Ansätze werden eine Reihe an Versuchen durchgeführt, die verschiedene Datensätze mit einer Vielzahl an unterschiedlichen Eigenschaften berücksichtigen. Dabei müssen fehlende Attributwerte abgeschätzt werden, damit der Random Forest angewandt werden kann. In der ersten Versuchsreihe werden iterativ Datensätze erstellt, mit denen die Herausforderungen isoliert betrachtet werden können. In dieser wird deutlich, dass der Random Forest mithilfe des Instance-Hardness Detektors durchschnittlich besser abschneidet als der Random Forest ohne Detektoren. In der zweiten Versuchsreihe dagegen wird der ImbalanceDataGenerator von Dennis Tschechlov verwendet, um Datensätze zu generieren, die alle Herausforderungen abdecken. Dabei stellt sich heraus, dass die Szenarien sehr ähnlich abschneiden. Generell erzielen der Instance-Hardness Detektor und der KDN unter den Detektoren die beste Genauigkeit. Dagegen schneidet der Partitioning-Detektor am schlechtesten ab. Der MCS-Detektor ist nicht für Mehrklassenprobleme geeignet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Heutzutage werden sehr viele Daten generiert und gesammelt, denn sie sind Grundlage vieler Prozesse in der Industrie und Forschung. Gerade durch technologische Entwicklungen, wie zum Beispiel die des IoT und der Industrie 4.0, wächst die Anzahl der Daten stetig und es wird von Big Data gesprochen. Dadurch wird es immer wichtiger, Muster in den Daten zu erkennen, um aus diesen und dementsprechend aus den Daten einen Nutzen zu ziehen. Mithilfe dieser Daten können zum Beispiel im Bereich des Qualitätsmanagements effizient Ursachen für fehlerhafte Produkte bestimmt werden, um diese anschließend zu beheben. Dadurch können erheblich Kosten eingespart werden, was das Arbeiten mit Big Data attraktiv macht. Die Sensordaten des Qualitätsmanagements werden dann beispielsweise in einem Datensatz zusammengefasst, der verschieden Herausforderungen in sich hat.

Die hierfür benötigten Daten müssen meistens in mehrere Klassen gruppiert werden. Dabei wird auch von einem Mehrklassenproblem gesprochen, welches durch Klassifikatoren gelöst werden soll. Die Klassifikatoren werden dafür auf einem Trainingsdatensatz trainiert, um einen Testdatensatz vorherzusagen. Dazu muss allerdings der Trainingsdatensatz bereits von einer Person in Klassen unterteilt worden sein, da die Klassifikatoren sonst die Muster und zugehörigen Klassenlabel nicht erlernen können. Diese Person ordnet jedem Datenpunkt ein Klassenlabel zu und labelt somit die Daten, damit die Klassifikatoren die Daten gruppieren können. Dies fällt den Klassifikatoren allerdings aufgrund verschiedener Charakteristika schwer, die zu folgenden Herausforderungen führen [HRM19]:

- **Wenige Datenpunkte je Klasse:** Die verwendeten Datensätze haben generell wenige Datenpunkte und beinhalten die Herausforderung mit nur durchschnittlich 10 Datenpunkten je Klasse. Die Klassifikatoren können hierfür Muster bei wenigen Datenpunkten nicht verallgemeinern und können diese daraus resultierend schlecht auf andere Daten anwenden.
- **Klassenungleichverteilung:** Diese Herausforderung tritt auf, wenn wenige Klassen einen Großteil der Datenpunkte besitzen. In den verwendeten Datensätzen ist dies bei etwa 10 von 84 Klassen der Fall, die 44% der Datenpunkte besitzen. Hierbei ist die Schwierigkeit die Klassen gleich zu gewichten, da Klassen, die häufig vorkommen, bevorzugt werden.
- **Vielfältiges Produktportfolio:** Hierzu gehören zum Beispiel Subkonzepte innerhalb einer Klasse oder fehlende Attributwerte, mit denen separat umgegangen werden muss. Im Anwendungsfall ist dies zum Beispiel durch Motoren mit unterschiedlicher Zylinder- und somit Sensorenanzahl gegeben.
- **Falsch dokumentierte Klassenlabel und sich überschneidende Klassen:** Diese Herausforderung tritt auf, wenn Datenpunkte einer falschen Klasse zugeordnet wurden und daher ein falsches Label besitzen. Bei sich überschneidenden Klassen handelt es sich um Klassen, die ähnliche Konzepte haben und somit schwierig zu differenzieren sind. Auf diese Herausforderung wird in dieser Arbeit gezielt eingegangen.

Diese Herausforderungen sorgen dafür, dass die meisten Klassifikatoren eine schlechte Vorhersagegenauigkeit erzielen [HRM19]. Es existieren bereits Ansätze, welche die Genauigkeit der Klassifikatoren für die einzelnen Herausforderungen verbessern, jedoch haben diese Ansätze häufig einen negativen Einfluss auf die anderen Herausforderungen. Dazu gehören zum Beispiel die Sampling Methoden Under- und Oversampling, die eine Möglichkeit sind, mit der Klassenungleichverteilung umzugehen. Dabei werden jedoch zum Beispiel die Problematiken der falsch dokumentierten Klassenlabel verstärkt, da diese ebenfalls beim Sampling dupliziert werden [HRM19][YL09].

Insbesondere stellt das Erkennen falsch dokumentierter Klassenlabel in einem Datensatz eine schwierige Aufgabe dar, wenn in diesem die weiteren Herausforderungen, wie wenige Datenpunkte je Klasse, vorliegen. Falsch dokumentierte Klassenlabel, auch Label Noise genannt, sind Datenpunkte, die manuell falsch gelabelt wurden. Beim Qualitätsmanagement müssen zum Beispiel gesammelte Daten zuerst gelabelt werden, sodass diese im Anschluss zum Trainieren verwendet werden können. Dafür ist häufig Domänenexpertise notwendig, da sonst gesammelte Sensordaten falsch interpretiert werden. Doch selbst wenn Experten die Daten labeln, kommt es häufig zu Label Noise, da die Datenpunkte nicht immer eindeutig zu einer Klasse gehören. Einer der Gründe dafür sind sich überschneidende Klassen, die dafür sorgen, dass ähnliche Konzepte in mehreren Klassen auftreten. In Isolation wurden dafür bereits Verfahren entwickelt [KWK20][GCL15]. Es ist jedoch unklar, wie gut diese Verfahren auch in Kombination mit den anderen Herausforderungen, vor allem der Klassenungleichverteilung, funktionieren. Denn dadurch existieren Klassen mit nur wenigen Datenpunkten. Insbesondere diese Klassen sind in Kombination mit falsch dokumentierten Klassenlabeln problematisch, da Algorithmen die kaum vertretenen Klassen als falsch dokumentierte Klassenlabel detektieren. Dadurch werden manche Klassen im Trainingsdatensatz gar nicht in die Mustererkennung eingebunden oder vom Klassifikator erlernt, wodurch dieser die Klassen im Testdatensatz nicht erkennt.

Falsch dokumentierte Klassenlabel sind keine seltene Herausforderung, da oftmals sehr spezifisches Domänenwissen benötigt wird, um komplexe Daten manuell zu labeln. Dabei treten häufig Fehler auf, da sich Klassen überschneiden oder kaum vertreten sind. Dadurch entstehen falsch dokumentierte Klassenlabel.

Ziel dieser Arbeit ist es, Datenvorbereitungsverfahren zu untersuchen, die mit falsch dokumentierten Klassenlabeln umgehen. Dabei werden diese in Hinblick auf die erwähnten Herausforderungen betrachtet und es wird untersucht, inwiefern datengetriebene Methoden dazu in der Lage sind, falsch dokumentierte Klassenlabel zu erkennen. Dafür sollen bereits existierende Verfahren untersucht und evaluiert werden, um insbesondere zu bestimmen, inwiefern diese Verfahren wenig vertretene Klassen von falsch dokumentierten Klassenlabeln unterscheiden können. Außerdem soll untersucht werden, ob sich diese Verfahren auch dafür eignen, falsch dokumentierte Klassenlabel direkt zu korrigieren oder ob nur die Erkennung möglich ist. Hierzu wird ein Datengenerator bereitgestellt, mit welchem Datensätze generiert werden können, in denen alle Herausforderungen enthalten sind.

Die Arbeit ist auf folgende Weise gegliedert: Zuerst werden in Kapitel 2 die Grundlagen und bereits existierende Arbeiten näher beschrieben. Dann wird in Kapitel 3 beschrieben, wie verschiedene Algorithmen untersucht werden, um im Anschluss in Kapitel 4 die Ergebnisse der Untersuchung zu evaluieren. Schließlich wird in Kapitel 5 die Arbeit zusammengefasst und ein Ausblick gegeben.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel wird beschrieben, welche Arbeiten zum Thema Label Noise bereits existieren und als Grundlage verwendet werden, um darauf aufbauend zu forschen. Dafür werden Grundlagen benötigt, die in Abschnitt 2.1 erläutert werden. Diese sind gegliedert in Maschinelles Lernen, Klassifizieren und den Random Forest Klassifikator. Im Anschluss wird in Abschnitt 2.2 auf die Herausforderungen genauer eingegangen. In Abschnitt 2.3 werden drei Ausreißermethoden vorgestellt, während in Abschnitt 2.4 auf drei unterschiedliche Label Noise Verfahren eingegangen wird.

2.1 Grundlagen

In diesem Abschnitt werden die Grundlagen beschrieben. Dazu gehört das Prinzip des Maschinellen Lernens und wie der Computer Muster erkennen kann. Außerdem wird beschrieben, wie ein bestimmter Klassifikationsalgorithmus, der Random Forest, funktioniert und was unter Label Noise verstanden wird.

2.1.1 Maschinelles Lernen

Unter Maschinellem Lernen wird das Erlernen von Mustern aus Daten verstanden. Es ist ein Teilgebiet der künstlichen Intelligenz und macht das eigenständige Lösen von Problemen möglich. Dazu werden die Daten verallgemeinert, um sie im nächsten Schritt auf ähnliche Daten anzuwenden. Anschließend ist das System in der Lage, unbekannte Daten zu beurteilen.

Es gibt zwei Arten des Maschinellen Lernens: Überwachtes und unüberwachtes Lernen. Beim überwachten Lernen handelt es sich um Daten, die bereits mit Labeln versehen sind und aus denen somit direkt gelernt werden kann. Beim unüberwachten Lernen haben die Daten keinerlei weitere Informationen außer ihren Attributwerten. Algorithmen müssen daher anhand der Struktur und der Attributwerte der Daten Muster erkennen, um diese im Anschluss auf andere Daten anzuwenden. Darüber hinaus gibt es Arten, die eine Verbindung beider Ansätze sind, und dementsprechend aus Daten lernen, die teilweise Label beinhalten.

2.1.2 Klassifikation

Die Klassifikation ist eine Anwendung des unüberwachten Maschinellen Lernens. Bei der Klassifikation wird mit gelabelten Datensätzen gearbeitet, das heißt, dass jeder Datenpunkt genau einer Klasse zugeordnet ist. Die Datenpunkte haben hierfür unterschiedliche Attributwerte und werden anhand dieser in Gruppen sortiert. Werden die Attributwerte der Datenpunkte als Dimensionen

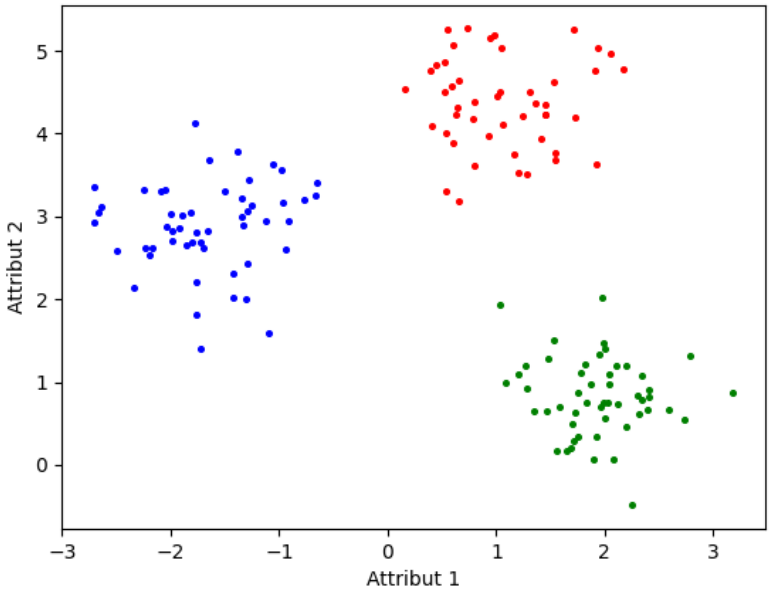

Abbildung 2.1: Graphische Darstellung dreier Klassen mit zwei Attributen

betrachtet, wird bei einer Menge von zwei Attributen in Abbildung 2.1 Folgendes deutlich: Wenn die Attributwerte grafisch dargestellt werden, mit Attribut 1 auf der x-Achse und Attribut 2 auf der y-Achse, versuchen die Klassifikatoren einen örtlichen Zusammenhang zu finden, um die Datenpunkte in Klassen einzuteilen.

Dabei gibt es zuerst eine Trainingsphase, in welcher der Algorithmus versucht, Muster zu erkennen und im Anschluss eine Testphase, in der die gelernten Muster angewandt werden. Dafür existieren unterschiedliche Datensätze: Ein Trainingsdatensatz, bei dem die Klassen im überwachten Lernen bekannt sind, und ein Testdatensatz, bei dem die Klassen unbekannt sind.

Der Klassifikator erlernt Muster des Trainingsdatensatzes durch bestimmte Algorithmen und versucht diese auf den Testdatensatz anzuwenden, um die Datenpunkte des Testdatensatzes zu klassifizieren. Dafür benötigt der Klassifikator ein Array für jeden Datenpunkt. Dieses Array beinhaltet für jede Klasse einen Wert, der besagt, mit welcher Wahrscheinlichkeit ein Datenpunkt zu dieser Klasse gehört.

Gegeben ist ein Beispiel mit drei Klassen: A, B und C, und ein Datenpunkt X mit zugehörigem Array. Nachdem ein Klassifikator angewendet wird, befinden sich in dem Array des Datenpunkts die Werte, die besagen, dass dieser zu 60% zur Klasse A, zu 30% zur Klasse B und zu 10% zur Klasse C gehört. Anhand dieses Arrays können die wahrscheinlichsten Label für einen Datenpunkt ausgelesen werden, die beim manuellen Klassifizieren helfen können.

Werden zur Klassifikation eines Datensatzes mehrere Lernalgorithmen, zum Beispiel Klassifikatoren, verwendet, wird von Ensemble Methoden gesprochen. Die Ausgabe dieser Lernalgorithmen wird zu einem Gesamtergebnis verrechnet. Da nicht nur ein einziger Lernalgorithmus das Ergebnis beeinflusst, wird mehr Rechenleistung benötigt, was zu einer höheren Fehlertoleranz und Genauigkeit führt.

2.1.3 Random Forest Klassifikator

Der Random Forest Algorithmus [Bre01] ist eine Ensemble Methode und funktioniert wie folgt:

Der Random Forest Algorithmus trainiert für eine zufällige Teilmenge an Datenpunkten und Features einen Entscheidungsbaum. Dabei werden verschiedene Entscheidungsbäume zufällig erstellt, indem die Datenpunkte und Attributwerte zufällig gewählt werden. Es entsteht somit ein Random Forest mit unterschiedlichen Entscheidungsbäumen. Beim Klassifizieren eines Datenpunkts trifft jeder Entscheidungsbaum eine Entscheidung und gibt seine Stimme ab, um bei der Gesamtentscheidung mitzuwirken. Der Random Forest Algorithmus basiert daher auf Bagging, da mehrere Verfahren oder in diesem Fall Bäume kombiniert werden, um auf eine Entscheidung zu kommen. Aus diesem Grund ist dieser robuster gegenüber Label Noise, da anstelle eines Entscheidungsbaums mehrere Entscheidungsbäume bei der Gesamtentscheidung mitwirken [HRM19].

2.1.4 Label Noise

Ein wichtiger Bestandteil des Klassifizierens ist das Labeln des Trainingsdatensatzes. Im Zusammenhang mit den in dieser Arbeit auftretenden, überwachten Lernmethoden, werden die Datenpunkte des Trainingsdatensatzes verschiedenen Klassen durch ein Label zugeordnet. Die Trainingsdaten durchlaufen einen Prozess, in welchem Datenpunkte erstellt und gelabelt werden. Dabei kann es vorkommen, dass den Datenpunkten in diesem Prozess ein falsches Label zugeordnet wird. Ein Grund dafür ist, dass durch sich überschneidende Klassen das Labeln sehr anspruchsvoll wird. Bei falsch dokumentierten Klassenlabeln wird auch von Label Noise gesprochen.

Wenn nicht beim manuellen Labeln Label falsch zugeordnet werden, sondern beim Erstellen der Daten fehlerhafte Datenpunkte entstehen, so wird von Ausreißern gesprochen. Diese entstehen zum Beispiel durch fehlerhafte Sensorwerte und sorgen dafür, dass Datenpunkte Attributwerte haben, die untypisch für Datenpunkte ihrer Klasse sind. Somit können diese leicht zu Label Noise führen [HRM19].

Meistens ist es sinnvoller, mehr Datenpunkte zu löschen als weniger, damit möglichst wenige falsche Datenpunkte den Lernerfolg verhindern [FV13]. Wenn es allerdings bereits wenige Datenpunkte gibt, muss ein Mittelweg gewählt werden.

2.2 Herausforderungen im End Of Line Testing

In der Arbeit von Vitali Hirsch, Peter Reimann und Bernhard Mitschang wurden datengetriebene Ansätze in Hinsicht auf Herausforderungen, wie zum Beispiel Label Noise, im End Of Line Testing untersucht [HRM19]. Dafür wurde ein industrienaher Anwendungsfall gewählt, in dem Motoren produziert und getestet werden. Im ersten Abschnitt wird beschrieben was unter End Of Line Testing verstanden wird. Im Anschluss wird auf die Herausforderungen der Datensätze genauer eingegangen und erläutert, wie in der Arbeit von Vitali Hirsch mit diesen umgegangen wird.

2.2.1 End Of Line Testing

Es wird speziell ein Szenario betrachtet, in welchem ein Mitarbeiter des Qualitätsmanagements im End-of-Line Testing, am Ende der Produktion, Motoren auf ihre Funktionalität testet. Dafür bekommt der Mitarbeiter die Sensordaten aus der Produktion und führt daraufhin eine Qualitätsprüfung durch. Im Anschluss werden die Sensordaten zu einem Datesatz zusammengefasst, in dem er nach Fehlern und deren Ursachen sucht, um die fehlerhaften Bauteile zu reparieren. Dies ist meistens sehr zeitaufwändig und benötigt fachspezifisches Wissen. Deshalb kann ein entscheidungsunterstützender Algorithmus des Maschinellen Lernens genutzt werden.

In der Arbeit von Vitali Hirsch wurde ein Datensatz aus dem End-Of-Line Testing Szenario verwendet, der verschiedene Herausforderungen hat, die im nächsten Abschnitt genauer erläutert werden. Dabei wurden verschiedene Verfahren untersucht und es wurde analysiert, ob und wie Verfahren oder Klassifikatoren mit den Herausforderungen umgehen. Bei diesen Verfahren handelt es sich um Ensemble-, Feature Selection- und Sampling-Techniken.

2.2.2 Herausforderungen

Datensätze aus industriellen Anwendungsfällen haben häufig bestimmte Herausforderungen, die es Klassifikatoren erschweren, den Datensatz richtig zu klassifizieren. Vor allem in der Industrie wird mit Herausforderungen gearbeitet, auf die im Folgenden genauer eingegangen wird. Dabei werden die gleichen Herausforderungen wie in Hirsch et. al. [HRM19] betrachtet.

Wenige Datenpunkte je Klasse Die Menge an gesammelten Daten ist häufig zu gering und sorgt dafür, dass Klassifikatoren nicht genug Daten zum Trainieren und Lernen haben. In den gegebenen Datensätzen ist dies mit 1050 Datenpunkten und 84 Klassen der Fall. Im Zusammenhang mit der hohen Anzahl an Attributen oder Dimensionen schaffen es Klassifikatoren im Großteil der Fälle, den Trainingsdatensatz zu klassifizieren. Die gelernten Konzepte sind allerdings zu genau und werden somit nicht im Testdatensatz erkannt.

Klassenungleichverteilung Eine Klassenungleichverteilung liegt vor, wenn ein Großteil der Datenpunkte zu einem kleinen Teil der Klassen gehört. Im Falle der gegebenen Datensätzen haben die 10 größten Klassen 44% der Datenpunkte. Klassen mit vielen Datenpunkten werden auch als Majority Klassen bezeichnet, während kaum vertretene Klassen als Minority Klassen bezeichnet werden. Es existieren demnach einige Minority Klassen, die wenig vertreten sind. Dies sorgt dafür, dass diese überwiegend von Klassifikatoren in der Trainingsphase ignoriert beziehungsweise kaum betrachtet werden. Zu den Majority Klassen gehört allerdings ein Großteil der Datenpunkte, weshalb die Genauigkeit des Klassifikators nicht erkennen lässt, dass dieser kaum Minority Klassen erkennt. Dadurch ist die Genauigkeit des Klassifikators hoch, obwohl nur eine von vielen Klassen richtig erkannt wird. Die Klassenungleichverteilung ist ein aktuelles Problem des Qualitätsmanagements [HYS+17], denn in den meisten Fällen, gibt es deutlich mehr korrekte als fehlerhafte Produkte. Dementsprechend werden auch sehr viel mehr Sensordaten der korrekten Produkte aufgezeichnet. Es gibt somit wesentlich weniger Sensordaten der fehlerhaften Produkte (der Minority Klassen), obwohl gerade das Erkennen dieser Produkte einen hohen Stellenwert hat [KGJH16].

Vielfältiges Produkt-Portfolio Der Datensatz umfasst viele Sensordaten von Motoren. Dabei haben die Motoren jedoch unterschiedliche Bauteile und Sensoren. Zum Beispiel gibt es in manchen Motoren weniger Zylinder als in anderen, weshalb sie weniger Sensordaten haben. Ein hierbei auftretendes Problem wird im folgenden Unterpunkt beschrieben. Das vielfältige Produkt-Portfolio wurde in Hirsch et. al. [HRM20] noch weiter unterteilt in fehlende Attributwerte, Subkonzepte und Klassenzugehörigkeit.

- Bei manchen Datenpunkten der Niedrig-Zylinder-Motoren werden manchen Attributen keine Werte zugewiesen. Dies kann problematisch sein, da Klassifikatoren für jedes Attribut Werte benötigen. Dabei handelt es sich um **fehlende Attributwerte**. Davon sind im vorhandenen Datensatz circa 17% der Datenpunkte betroffen. Diese fehlenden Attributwerte können entweder ganz entfernt oder mit einem Imputer abgeschätzt werden. Dafür werden die benachbarten Datenpunkte verglichen und anhand derer Label entschieden, mit welchem Wert die fehlenden Attributwerte belegt werden sollen. Es gibt unterschiedliche Möglichkeiten, die Werte abzuschätzen, die in Abschnitt 3.2 genauer erklärt werden.
- Eine weitere Eigenschaft sind **Subkonzepte**, die innerhalb einer Klasse auftreten können. Klassen haben zwar vorwiegend eine bestimmte Ausprägung von Werten mehrerer Attribute, jedoch treten bei heterogenen Klassen meistens unterschiedliche Konzepte innerhalb einer Klasse auf. Durch die unterschiedlichen Wertausprägungen versuchen die Klassifikatoren die Subkonzepte separat zu erlernen. Die Herausforderung, wenige Datenpunkte je Klasse, erschwert dies jedoch, da die Subkonzepte dadurch noch weniger vertreten sind. Folglich wird auch das Erkennen der gesamten Klasse erschwert, da die Attributwerte stärker variieren. Beispielsweise existieren zwei Muster in einer Klasse, die separat erlernt werden.
- Bei der **Klassenzugehörigkeit** geht es darum, dass Klassifikatoren manche Informationen, obwohl sie vorhanden sind, nicht in ihre Klassifikation miteinbeziehen können. Dazu gehört zum Beispiel, dass manchen Datenpunkten Attributwerte fehlen und diese somit nicht mit anderen Datenpunkten in Verbindung gebracht werden können. Zum Beispiel haben Motoren mit sechs Zylindern mehr Sensordaten als Motoren mit vier Zylindern. Es gibt dementsprechend Klassen oder Bauteile, die in Motoren mit sechs und vier Zylindern vorkommen. Das Problem ist allerdings, dass es auch Klassen gibt, die nur in Kombination mit einer speziellen Zylinderart vorkommen und daher fest zusammenhängen. Diese Information hat der Algorithmus allerdings nicht, dementsprechend klassifiziert er weniger gut, da er auch Muster zwischen Klassen sucht, die gar nicht zusammenhängen können.

Label Noise und sich überschneidende Klassen Label Noise sind Datenpunkte, die falsch gelabelt wurden. Diese gehören eigentlich zu anderen Klassen und haben demnach Attributwerte, die nicht zur zugewiesenen Klasse passen. In Datensätzen der Industrie kann dies zum Beispiel zustande kommen, wenn Datenpunkte manuell gelabelt werden. Das liegt daran, dass manuelles Labeln zeitintensiv ist und häufig spezifisches Domänenwissen benötigt. Im Zusammenhang mit sich überschneidenden Klassen ist das Klassifizieren noch schwieriger, da die Klassen ähnliche Konzepte haben.

In Abbildung 2.2 sind zwei sich überschneidende Klassen dargestellt. Die Datenpunkte haben zwei Attribute, die auf den x- und y-Achsen abgebildet sind. Die Datenpunkte sind nach ihrem Label gefärbt und überschneiden sich im Bereich um (0,-2,5). Es ist deutlich zu sehen, dass manche Attributwerte in beiden Klassen auftreten, daher überschneiden sie sich auch in der Grafik. Daraus

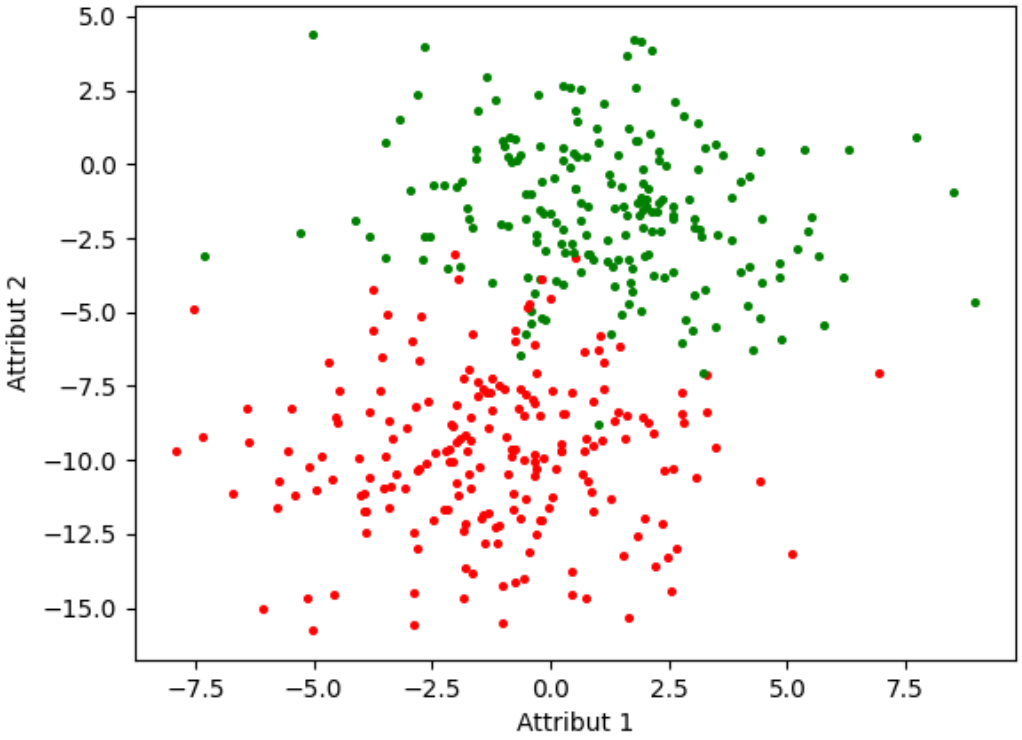

Abbildung 2.2: Visualisierung sich überschneidender Klassen

folgt, dass manche Datenpunkte gerade in der Schnittmenge schwierig zuzuordnen sind. Selbst für geschulte Fachkräfte ist das richtige Klassifizieren schwierig und es kommt häufiger zu Label Noise.

2.2.3 Ansätze zur Adressierung der Herausforderungen

Im Hinblick auf die Herausforderungen werden verschiedene Verfahren analysiert. Dazu werden diverse Klassifikatoren angewendet und auf ihre Genauigkeit untersucht. Unter anderem wurden Ensemble Algorithmen wie AdaBoost und Random Forest und Sampling Methoden wie Random Oversampling und Random Undersampling verwendet. Die Genauigkeit der Algorithmen wurde mithilfe der Genauigkeiten A@1 bis A@10 gemessen.

Um der Klassenungleichverteilung entgegenzuwirken wurden Sampling Verfahren genutzt, die versuchen, die wenigen Datenpunkte einer Klasse auszugleichen. Dafür werden entweder durch Oversampling Datenpunkte der Minority Klassen künstlich erstellt oder durch Undersampling Datenpunkte der Majority Klasse entfernt. Beim Erstellen neuer Datenpunkte wird sich jedoch an der ursprünglichen Datenmenge orientiert. Das heißt, dass die vierte Herausforderung „Label Noise und sich überschneidende Klassen“ verstärkt wird, denn auch an den Label Noise aus der ursprünglichen Datenmenge wird sich orientiert und daher werden auch falsche Datenpunkte generiert. Beim Löschen neuer Datenpunkte wird hingegen die erste Herausforderung „Wenige Datenpunkte je

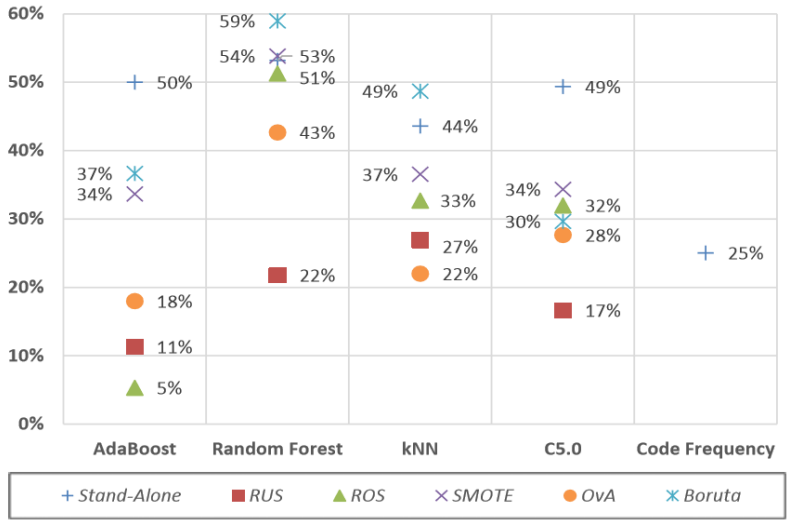

Abbildung 2.3: Evaluationsergebnisse aus [HRM19]: A@4 Genauigkeiten unterschiedlicher Klassifikatoren im Zusammenhang mit Sampling Methoden

Klasse“ verstärkt, da bereits wenige Datenpunkte existieren und von diesen wenigen Datenpunkten noch weitere gelöscht werden. Auch die Ergebnisse belegen diese Theorien und zeigen, dass Random Over- und Undersampling die Genauigkeit der Klassifikatoren sogar verschlechtern [HRM19].

In Abbildung 2.3 sind auf der x-Achse die unterschiedlichen Klassifikatoren abgebildet, wobei der Fokus auf dem Random Forest liegt. Auf der y-Achse ist die A@4 Genauigkeit aufgezeigt. Wie in Abschnitt 2.1 beschrieben, haben Klassifikatoren für jeden Datenpunkt ein Array, welches beschreibt, wie wahrscheinlich eine Klasse zu diesem Datenpunkt gehört. Bei der A@4 Genauigkeit werden diese Arrays verwendet, um nicht nur die wahrscheinlichste Klasse, sondern die vier wahrscheinlichsten Klassen zu verwenden. Des Weiteren werden unterschiedliche Sampling Methoden wie Random Oversampling (ROS) und Random Undersampling (RUS) in Kombination mit den Klassifikatoren verwendet. Im Vergleich zum Random Forest ohne vorangehendes Sampling (Stand Alone) sind diese mit Abstand schlechter. Es zeigt sich daher, dass Sampling Methoden nicht gut geeignet sind.

Der Random Forest Algorithmus ist bei den gegebenen Herausforderungen nach [HRM19] der robusteste Klassifikator. Robust bedeutet, dass er im Vergleich weniger negativ auf die Herausforderungen reagiert. In Abbildung 2.3 ist zu sehen, dass er die höchsten Genauigkeiten erzielt. Der Random Forest Algorithmus wurde in Abschnitt 2.1 genauer beschrieben.

2.3 Ausreißer

Als Ausreißer werden Datenpunkte bezeichnet, die weit entfernt von ihrem Zentrum liegen. Diese Datenpunkte haben eine Kombination aus Attributen, die zu keiner Klasse passt. Zwar gibt es meistens nur wenige Ausreißer, die durch Fehler oder Ausnahmezustände entstanden sind, doch spielen sie in Praxisnahen Anwendungen eine wichtige Rolle. Dies liegt daran, dass auch nur wenige Ausreißer die Genauigkeit von Klassifikatoren verschlechtern können [CBK09]. Die

Ausreißererkennung ist unüberwacht und versucht somit, ohne Betrachtung der Label, Datenpunkte als Ausreißer zu erkennen. Es existiert eine Vielzahl an Methoden, um die Ausreißer zu erkennen. Die beliebtesten sind [MCS21][RD99]:

- Die erste Methode arbeitet mit der Berechnung des **Local Outlier Factors** [BKNS00]. Dazu wird die Dichte jedes Datenpunkts mit der seiner k-Nachbarn verglichen. Wenn sich diese zu sehr unterscheiden, handelt es sich wahrscheinlich um einen Ausreißer.
- Der **Isolation Forest** ist die zweite Methode, welche versucht jeden Datenpunkt durch vertikales Aufbauen eines Gitters in einer Zelle zu isolieren [LTZ08]. Je mehr Gitterlinien dafür benötigt werden, desto näher liegt der Datenpunkt vermutlich im dicht besiedelten Zentrum und ist daher mit hoher Wahrscheinlichkeit kein Ausreißer.
- Als dritte Methode gibt es die **robuste Kovarianz** [RD99]. Bei dieser Methode geht der Algorithmus von einer Gauß-Verteilung aus und versucht eine Ellipse um die Daten zu legen. Je weiter die Datenpunkte außerhalb dieser Ellipse liegen, desto höher ist die Wahrscheinlichkeit, dass es sich um Ausreißer handelt.

Diese Verfahren nutzen jedoch die Label nicht aus, die im gegebenen Datensatz vorliegen. Deshalb werden im folgenden Kapitel Label Noise Verfahren untersucht.

2.4 Label Noise Verfahren

Wenn es sich um Datenpunkte handelt, die nicht wie Ausreißer durch einen Fehler entstanden sind, denen jedoch falsche Label zugeordnet wurden, wird von Label Noise gesprochen. Wie bereits erwähnt kann dies häufig auftreten, da tiefgründiges Domänenwissen benötigt wird und es daher selbst für Experten schwierig ist, die korrekten Label zu wählen. Nach [FV13] gibt es folgende drei Methoden mit Label Noise umzugehen.

2.4.1 Robuste Methoden

Die erste Methode sind die robusten Klassifikatoren. Diese betrachten nicht jedes Detail und sorgen dafür, dass einzelne falsche Label kaum ins Gewicht fallen und nicht als Muster erlernt werden. Ein Beispiel dafür wäre der Random Forest, der im Hinblick auf die Herausforderungen am besten abgeschnitten hat [HRM19]. Diese robusten Methoden werden jedoch trotzdem von Label Noise beeinflusst. Dies liegt daran, dass die Label Noise nicht aus dem Datensatz entfernt und dementsprechend auch zum Lernen verwendet werden. Die Genauigkeit der Klassifikatoren wird somit zwar verringert, jedoch nicht im gleichen Ausmaß wie ohne robuste Klassifikatoren.

2.4.2 Tolerante Methoden

Eine weitere Methode ist es, sowohl die normalen Daten als auch Label Noise zu modellieren. Da Label Noise nur dann modelliert werden können, wenn auch bekannt ist, welche der Datenpunkte ein falsches Label haben, ist es wichtig, dass diese Informationen vorhanden sind. Anhand der gegebenen Label Noise wird demnach ein Modell entwickelt, welches Label Noise vorhersagen kann.

Wenn allerdings davon ausgegangen wird, dass Label Noise zufällig und nicht, wie richtig gelabelte Datenpunkte, nach einer gewissen Verteilung entstehen, ist dies kein sinnvoller Ansatz. Der Grund dafür ist, dass es keine direkte Verbindung zwischen verschiedenen Label Noise gibt. Allerdings könnte es vorkommen, dass ähnliche Datenpunkte falsch gelabelt werden, da diese für Experten sehr schwer zu bestimmen sind. Dies ist bei sich überschneidenden Klassen zum Beispiel der Fall, da hier die Attributwerte nicht eindeutig zu einzelnen Klassen zugeordnet werden können. Somit kommt es auf das Anwendungsgebiet an, ob diese Methode sinnvoll ist.

2.4.3 Detektoren

Die dritte Methode ist das Säubern der Daten, indem versucht wird Label Noise einzeln herauszufiltern. Label Noise besitzt andere Ausprägungen in den Attributen im Vergleich zu den anderen Instanzen der Klassen und kann somit erkannt werden Es wird daher überwiegend nach Datenpunkten geschaut, die inmitten von Datenpunkten mit anderem Label liegen oder weit vom Zentrum ihrer Klassen entfernt sind. Diese Algorithmen heißen Detektoren, da sie versuchen die falschen Label zu detektieren. Nachdem ein Label Noise erkannt wurde, ist es möglich, entweder den Datenpunkt komplett zu entfernen oder nur das Label.

3 Methodik zur Untersuchung der Detektoren

In diesem Kapitel wird beschrieben, wie mit den Herausforderungen und speziell in diesem Zusammenhang mit den falsch dokumentierten Klassenlabeln umgegangen wird. Dafür werden die Detektoren, die in Abschnitt 2.4.3 eingeführt wurden, weiter angeschaut. Die anderen Label Noise Verfahren, die in Abschnitt 2.4 beschrieben wurden, werden nicht weiter betrachtet, da wie in [FV13] beschrieben, die toleranten Methoden Informationen über Label Noise benötigen. Außerdem wurde bereits ein robuster Klassifikator, der Random Forest in Hirsch et. al. [HRM19] untersucht, der in dieser Arbeit verwendet wird.

In Abbildung 3.1 ist der Ablauf der Untersuchung dargestellt. Um die Detektoren zu untersuchen wird ein Datensatz benötigt. Dieser wird generiert, damit die Eigenschaften isoliert betrachtet werden können. In Abschnitt 3.1 wird beschrieben, wie der Datensatz generiert wird und welche Parameter für die Generierung gewählt werden. Im Anschluss müssen fehlende Attributwerte mit Imputern abgeschätzt werden, sollten diese im Datensatz vorkommen. Wie Imputer funktionieren wird in Abschnitt 3.2 beschrieben. Im Anschluss werden die Detektoren auf den Datensatz angewendet, damit diese Label Noise herausfiltern. In Abschnitt 3.3 wird die allgemeine Vorgehensweise der Detektoren beschrieben. Damit die Auswirkung der Detektoren festgestellt werden kann, wird im Anschluss ein Klassifikator angewandt, dessen Funktionsweise in Abschnitt 3.4.1 beschrieben wird. Abschließend werden Metriken angewendet, die zum Beispiel die Genauigkeit des Klassifikators bestimmen. Welche Metriken verwendet werden, wird in Abschnitt 3.5 erläutert.

3.1 Datensatz

In diesem Abschnitt wird beschrieben welche Datensätze verwendet und generiert werden und welche Parameter dafür gewählt werden, damit die Herausforderungen bestmöglich untersucht werden können.

Abbildung 3.1: Graphische Darstellung der Vorgehensweise der Untersuchung

3.1.1 Datensatz generieren

Die Datensätze müssen generiert werden, da es kaum Echtweltdaten gibt, die öffentlich verfügbar sind und die beschriebenen Herausforderungen beinhalten. Außerdem können die Eigenschaften so kontrolliert untersucht werden. Deswegen wird eine Bibliothek verwendet, mit der Datensätze generiert werden können. Dabei gibt es zwei Hauptmethoden aus Scikit-Learn[1], die „make_classification"- und die „make_blobs"-Methode. Beide werden verwendet, um Datensätze zu generieren.

Die „make_blobs"-Methode generiert für jede Klasse ein Zentrum. Um dieses Zentrum werden die Datenpunkte isotropisch, das heißt unabhängig von einer Richtung, mit einer Gauß-Verteilung gestreut. Es entstehen somit mehrere Cluster aus Datenpunkten. Dabei gibt es Parameter die unabhängig voneinander gewählt werden können.

In der zweiten Versuchsreihe wird der ImbalanceDataGenerator[2] von Dennis Tschechlov genutzt, der die „make_classification"-Methode verwendet. Bei der „make_classification"-Methode hängen die Parameter enger zusammen. Das heißt, dass zum Beispiel die Attributanzahl nicht unabhängig von der Klassenanzahl verändert werden kann. Diese Methode erstellt komplexer zusammengesetzte Daten, bei der lineare Transformation und mehrere Gauß-Verteilungen pro Klasse verwendet werden [Guy03].

Nachdem mit einer der beiden Methoden ein Datensatz erstellt wurde, werden Label vertauscht, um Label Noise zu simulieren. Anschließend wird der Datensatz in einen Trainings- und Testdatensatz aufgeteilt.

3.1.2 Eigenschaften, die Auswirkung auf Detektoren haben

Die Anzahl der Attributwerte für jeden Datenpunkt wird mit dem „n-features"-Parameter festgelegt. Mit diesem kann auch die Dimension der Daten festgelegt werden.

Um die Anzahl an Datenpunkten festzulegen, wird der „n-samples"-Parameter angepasst. Bei diesem Parameter handelt es sich um ein Array, das je Eintrag die Anzahl der Datenpunkte einer Klasse umfasst. Dieser beschreibt, wie viele Datenpunkte im Datensatz in jeder Klasse vorhanden sind. Die hier festgelegte Größe des Arrays beschreibt, wie viele Klassen es gibt.

Bei dem „cluster-std"-Parameter handelt es sich um die Standardabweichung der Gauß-Verteilung, welche die Datenpunkte innerhalb einer Klasse streut.

In Abbildung 3.2 ist eine Klasse dargestellt, die in der linken Abbildung mit Streuung = 1 und in der rechten Abbildung mit Streuung = 5 erstellt wurde. Je größer die Standardabweichung, desto größer ist die Fläche, auf der die Datenpunkte einer Klasse verteilt werden. Bei mehreren Klassen wächst dementsprechend der Grad der Überschneidung mit der Größe des Parameters.

Damit die Datenpunkte in Trainings- und Testdatensatz aufgeteilt werden können, wird noch ein „TestSize"-Parameter benötigt, der beschreibt wie viel Prozent der Datenpunkte im Testdatensatz sind, während der Rest der Datenpunkte im Trainingsdatensatz ist.

[1] https://scikit-learn.org

[2] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/ImbalanceDataGenerator.git

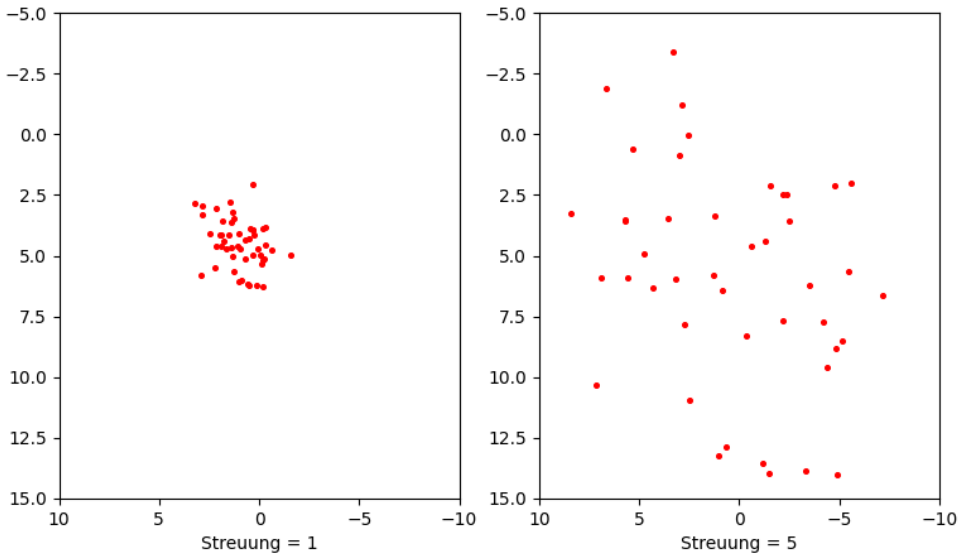

Abbildung 3.2: Visualisierung der Streuung einer Klasse mit zwei Attributen

Es wird noch ein „Noise“-Parameter benötigt um Label zu vertauschen. Hierfür werden je zwei Datenpunkte betrachtet, um dem ersten Datenpunkt das Label des zweiten Datenpunkts zuzuweisen und umgekehrt. Hiermit werden falsch dokumentierte Klassenlabel simuliert. Je höher der Parameter ist, desto mehr Prozent der Label werden getauscht und daher falschen Klassen zugeordnet.

Damit die Datensätze durch die Gaußverteilung nicht zufällig erstellt werden wird der „Random-State“-Parameter angegeben. Dieser macht es möglich, dass bei jedem Durchlauf mit den gleichen Parametern der gleiche Datensatz generiert werden kann. Dadurch hat diese Varianz keinen Einfluss mehr auf die Untersuchung.

3.2 Imputer

Aufgrund der Möglichkeit des Vorkommens fehlender Attributwerte in den Daten müssen diese entsprechend behandelt werden. Dafür werden Imputer benötigt. Diese versuchen anhand gegebener Informationen die fehlenden Attributwerte abzuschätzen. Dafür werden Informationen ähnlicher Datenpunkte in Betracht gezogen.

In Abbildung 3.2 sind Datenpunkte mit zwei Attributen dargestellt. Wenn zum Beispiel Datenpunkte mit fehlenden Attributwerten existieren, können diese nicht visualisiert werden, da sie nur einen x- oder y-Wert haben. Hat ein Datenpunkt einen x-Wert von 0 aber keinen y-Wert, kann ein Imputer beispielsweise alle Datenpunkte mit einem x-Wert von 0 betrachten und einen Mittelwert der zugehörigen y-Werte berechnen. Anschließend kann der Datenpunkt bei (0, 5) dargestellt werden.

Simple Imputer Bei dem Simple Imputer können verschiedene Standardwerte gewählt werden. Anstatt die fehlenden Attributwerte abzuschätzen, wird hierbei Rechenleistung gespart, indem die fehlenden Attributwerte mit einem gewählten Wert, in diesem Fall mit 0, ersetzt werden. Dabei werden weder die Nachbarn noch das eigene Label betrachtet. Wenn im Datensatz bereits viele Attribute mit dem Wert 0 existieren, kann es zu Problemen führen, da die Klassifikatoren Muster erkennen, die nicht erkannt werden sollen. In diesem Fall kann beispielsweise ein negativer Wert gewählt werden.

KNN Imputer Der K Nearest Neighbor Imputer nutzt, im Gegensatz zum Simple Imputer, mehr Informationen. Er nutzt die Attributwerte der direkten Nachbarn und schaut, ob diese für das Attribut einen Wert besitzen, das dem betrachteten Datenpunkt fehlt. Dazu werden genau k Nachbarn verglichen und ein Durchschnittswert gebildet. Die k Nachbarn haben dabei alle einen Wert für das Attribut, welches bei dem betrachteten Datenpunkt fehlt. k ist allerdings ein weiterer Parameter, der gesetzt und angepasst werden muss, um die bestmöglichen Ergebnisse zu erzielen [TCS+01].

Miss-Forest Der Miss-Forest Imputer nutzt, wie der Name schon sagt einen Random Forest, der die fehlenden Attributwerte abschätzt. Auf den Daten wird iterativ gelernt, um pro Iteration mehr fehlende Werte vorherzusagen, damit als Ausgabe alle fehlenden Attributwerte durch den Random Forest vorhergesagt werden können. Der Algorithmus hat viele Vorteile gegenüber dem KNN-Imputer, wie zum Beispiel die Robustheit gegenüber Noise, jedoch benötigt er mehr Rechenleistung [SB12].

3.3 Detektoren

Nach der Generierung und Verbesserung des Datensatzes werden Detektoren wie in Abbildung 3.1 angewandt. Im Rahmen der Label Noise Verfahren liegt der Fokus dieser Arbeit auf den Detektoren. Dafür werden die Detektoren aus Scikit-Clean[3] genutzt und die Parameter angepasst. In diesem Abschnitt wird beschrieben, wie die Detektoren zuerst ausgewählt und im Anschluss angewandt werden.

3.3.1 Detektorenauswahl

Auf Scikit-Clean gibt es zehn verschiedene Detektoren, die zuerst gruppiert werden. Die zehn Detektoren sind in Tabelle 3.1 mit ihren Gruppen dargestellt. Da es mehrere Versionen des Random Forest Detektors und des KDNs gibt, werden diese aussortiert, um sich auf die Standardvariante zu fokussieren. Zur Untersuchung wird aus jeder Gruppe ein Detektor ausgewählt, die im Folgenden näher betrachtet werden.

Die meisten Detektoren nutzen Klassifikatoren, um mit deren Ausgaben für jeden Datenpunkt eine Wahrscheinlichkeit zu berechnen, wie wahrscheinlich dieses Label richtig gelabelt wurde. Diese Wahrscheinlichkeit wird als Konfidenzwert bezeichnet.

[3] https://scikit-clean.readthedocs.io

Detektor	genutzte(r) Klassifikator(en)
KDN	K-Disagreeing-Neighbor
RKDN	K-Disagreeing-Neighbor
WKDN	K-Disagreeing-Neighbor
Random Forest Detector	Random Forest Klassifikator
ForestKDN	Random Forest Klassifikator
Dummy Detector	-
MCS	Ensemble auf zufälligen Teilmengen
Partitioning-Detector	Ensemble auf Partitionen
Instance-Hardness Detektor	Ensemble auf dem Datensatz

Tabelle 3.1: Detektoren mit zugewiesenen Gruppen

KDN Der K-Disagreeing-Neighbour (KDN) [WCO+18] Detektor baut darauf auf, dass er jedes Label den jeweilig k nächstgelegenen Nachbarn vergleicht und daraus eine Wahrscheinlichkeit ermittelt. Intuitiv beruht dies auf der Annahme, dass Datenpunkte, die zur gleichen Klasse gehören, nahe beieinander liegen. Je mehr Nachbarn das gleiche Label wie der zu vergleichende Datenpunkt haben, desto höher ist die Wahrscheinlichkeit, dass es sich um einen richtig gelabelten Datenpunkt handelt. Denn Datenpunkte die ähnliche Attributwerte haben, gehören normalerweise der gleichen Klasse an. Wenn ein Datenpunkt viele Nachbarn hat, die ein anderes Label haben, liegt er vermutlich in einer falschen Klasse und hat somit ein falsches Label. Problematisch sind allerdings Ränder von Klassen und sich überschneidende Klassen. In diesen Fällen ist es jedoch selbst für qualifizierte Experten schwierig die Label Noise zu erkennen.

Instance-Hardness Der Instance-Hardness Detektor [SMG14] berechnet wie hart oder schwierig ein Datenpunkt im Vergleich zu den anderen Datenpunkten des Datensatzes richtig klassifiziert werden kann. Je härter ein Datenpunkt ist, desto höher ist die Wahrscheinlichkeit, dass es sich um ein richtiges Label handelt. Der Instance-Hardness Detektor berechnet diese Härte, indem er ein Ensemble aus unterschiedlichen Klassifikatoren auf dem Datensatz trainieren lässt. Dabei vergleicht der Algorithmus die Label, welche die unterschiedlichen Klassifikatoren gesetzt haben. Je Datenpunkt wird berechnet, wie viele der Klassifikatoren das Label richtig gesetzt haben. Diese Anzahl wird dann im Zusammenhang mit der Gesamtzahl der Klassifikatoren mit Cross-Validation zu einem Konfidenzwert berechnet, der besagt, ob ein Datenpunkt richtig gelabelt wurde.

Partitioning-Detektor Der Partitioning-Detektor [KR07] basiert wie auch der Instance-Hardness Detektor auf der Annahme, dass ein Label vermutlich falsch gesetzt wurde, wenn es von vielen Modellen falsch klassifiziert wurde. Der Partitioning-Detektor teilt den Datensatz zuerst in mehrere Datensätze auf. Dann werden mehrere Klassifikatoren auf diesen Teil-Datensätzen trainiert. Wenn mehrere Klassifikatoren ein Label falsch klassifiziert haben, wird es als Label Noise detektiert und mit den anderen Datenpunkten zu einem Konfidenzwert berechnet. Dabei kann entweder ein oder sogar mehrere Klassifikatoren gewählt werden.

Random Forest Detektor Der Random Forest Detektor [SMS18] nutzt den Random Forest Klassifikator. Dafür wird zuerst ein Random Forest auf dem gesamten Datensatz trainiert. Dieser erstellt mehrere Entscheidungsbäume, die auf einer Teilmenge des Datensatzes trainieren. Dabei ist zu beachten, dass der Random Forest Klassifikator für einen Entscheidungsbaum nicht den gesamten Datensatz verwendet, sondern nur eine Teilmenge, in der nicht alle Datenpunkte vorhanden sind. Nachdem alle Entscheidungsbäume trainiert wurden, wird für jeden Datenpunkt die Menge an Entscheidungsbäumen genutzt, die nicht auf diesem Datenpunkt trainiert wurden. Der Prozentanteil an Bäumen, die das Label richtig vorhergesagt haben, wird dann als Konfidenzwert ausgegeben. Allgemein wendet der Random Forest somit die Muster des Trainingsdatensatzes auf den Testdatensatz an und ordnet anschließend jedem Datenpunkt des Testdatensatzes ein Label zu. Diese Label sind dann die Ausgabe des Random Forest.

MCS MCS [ZCTP19] nimmt an, dass wahre Daten nach einer Verteilung entstanden sind, während Label Noise keinen Zusammenhang zu anderen Label Noise hat und somit nicht modellierbar ist. Dafür wird ein Markov Chain Sampling Framework verwendet, um mehrere Klassifikatoren auf zufällig erstellten Teilmengen zu trainieren. MCS kann im Anschluss mit jeder Ausgabe eine Wahrscheinlichkeit ausrechnen, ob ein Datenpunkt richtig gelabelt wurde. Die Problematik des MCS ist wie in [ZCTP19] beschrieben, dass er vor allem für binäre Klassifikationsprobleme gemacht wurde. Zwar wird er in der Evaluation mit betrachtet, jedoch werden keine guten Ergebnisse erwartet.

3.3.2 Detektoren anwenden

Um die Detektoren auch anzuwenden, müssen diese implementiert werden. Die Detektoren bekommen als Eingabe den Trainingsdatensatz inklusive Label und Label Noise. Die Detektoren berechnen für jeden Datenpunkt einen Konfidenzwert. Dieser Konfidenzwert besagt, wie wahrscheinlich ein Datenpunkt richtig gelabelt wurde.

Um die Menge an Label Noise zu bestimmen, wird zusätzlich ein Grenzwert benötigt, der festlegt ab welcher Größe des Konfidenzwerts ein Datenpunkt als Label Noise zählt. Dies ist ein weiterer Parameter, der, wenn nicht der Standardwert 0.5 angenommen wird, gegebenenfalls gesetzt werden muss.

3.4 Szenarien

Um die Genauigkeit der Detektoren zu messen, werden verschiedene Szenarien verwendet. Dabei bauen diese auf den verwendeten Datensätzen und Detektoren auf. Insgesamt gibt es drei Szenarien, die alle parallel auf den gleichen Datensätzen angewendet werden, damit sie miteinander verglichen werden können.

S-Baseline Das erste Szenario nutzt den gegebenen Datensatz und führt darauf den Random Forest Klassifikator aus. Dabei wird vorher kein Detektor ausgeführt. Die Genauigkeit des Random Forest dient somit als Vergleich, wie der Random Forest ohne Detektoren auf dem Datensatz inklusive Label Noise abschneiden würde.

S-Löschen Im zweiten Szenario dagegen werden die von Detektoren erkannten Label Noise gelöscht und haben dementsprechend keinen Einfluss mehr auf das Ergebnis. Dafür wird ein Detektor auf den Daten mit Label Noise ausgeführt. Dieser gibt für jeden Datenpunkt einen Konfidenzwert aus. Anhand dieser Konfidenzwerte werden die Datenpunkte mit einem Grenzwert verglichen und entfernt, wenn der Konfidenzwert niedriger als der Grenzwert ist. Als Grenzwert wird 0.1 gewählt, da sonst zu viele Datenpunkte gelöscht werden. Dabei kann es auch vorkommen, dass richtige Datenpunkte (kein Label Noise) gelöscht werden, was zu einer Verringerung der Genauigkeit führt.

S-Gewichten Im dritten Szenario werden die Konfidenzwerte der Detektorausgabe dem Random Forest Klassifikator mitgegeben. Dieser nutzt die Konfidenzwerte, um die Datenpunkte dementsprechend zu gewichten. Ein Datenpunkt, der wahrscheinlicher Label Noise ist, wird weniger gewichtet und hat demnach weniger Auswirkung auf das Klassifizieren. In diesem Szenario haben die Label Noise daher weniger Einfluss auf das Ergebnis.

Probleme der Szenarien Wenn in S-Löschen die Datenpunkte gelöscht werden, muss aufgepasst werden, dass nicht zu viele und dadurch auch richtige Datenpunkte gelöscht werden und es zum „Overcleansing“ kommt. Beim „Overcleansing“ werden zu viele Datenpunkte entfernt und die Klassifikatoren können im Anschluss den Trainingsdatensatz nicht verallgemeinern.

Hierbei ist eine mögliche Lösung (die in keiner der Szenarien auftaucht) nur die Label und nicht die Datenpunkte zu löschen, da sonst eventuell auch wichtige Muster gelöscht werden. Wenn nur die Label der Datenpunkte gelöscht werden, gehen weniger Daten verloren, es entstehen allerdings Datenpunkte ohne Label. Die Datenpunkte ohne Label müssen im nächsten Schritt mit einem halb-überwachten Klassifikator klassifiziert werden, der auch mit nicht gelabelten Datenpunkten umgehen kann. Hier kann zum Beispiel auf den Daten mit Labeln trainiert werden, um anschließend die Datenpunkte ohne Label zu klassifizieren. Beim halb-überwachten Klassifizieren kann allerdings wieder Label Noise entstehen.

Ein Problem, das in beiden Fällen existiert ist, dass es schnell dazu kommen kann, dass Minority-Klassen, da sie im Datensatz selten vorkommen, als Label Noise erkannt und entfernt werden. Manche Minority-Klassen existieren somit im Trainingsdatensatz nicht mehr und können demnach nicht mehr erkannt werden.

3.4.1 Klassifikator anwenden

Als Klassifikator wird in dieser Arbeit, der Random Forest gewählt, der in Abschnitt 2.1 beschrieben wurde. Der Random Forest wird, je nach Szenario, entweder mit oder ohne den Konfidenzwerten eines Detektors ausgeführt. Dafür wird er zuerst auf den Trainingsdaten trainiert. Dazu bekommt der Random Forest den Trainingsdatensatz inklusive Label Noise und gegebenenfalls noch den Konfidenzwert eines Detektors übergeben. Im Trainingsdatensatz versucht der Random Forest Muster zu erkennen, die er im Testdatensatz anwenden kann. Für den Testdatensatz sind dagegen nur Datenpunkte ohne Label vorhanden.

Wenn der Random Forest in S-Löschen angewendet wird, tritt dabei ein Problem auf: In S-Löschen werden ab einem bestimmten Grenzwert Datenpunkte mit einem niedrigen Konfidenzwert gelöscht. Dabei kann es vorkommen, dass alle Datenpunkte einer Klasse gelöscht werden. Da Scikit-Learn immer Klassen von 0 bis N ausgibt und diese keine Lücken haben können, muss folgendes beachtet werden. Wenn der Random Forest dann auf diesen Daten trainiert, kennt er manche Klassen gar nicht. Wenn zum Beispiel die Klasse 5 fehlt, wird der Random Forest die Klasse 5 nicht überspringen, sondern er wird der Klasse 5 zum Beispiel die Klasse 6 zuordnen und der Klasse 6 dann die Klasse 7 und so weiter. Wenn dann der Random Forest auf den Testdatensatz angewendet wird, wird dieser eine Klasse 6 erkennen, die im echten Datensatz aber eigentlich Klasse 7 ist. Wenn die tatsächlichen Label mit dem vorhergesagten Label verglichen werden, entsteht eine viel schlechtere Genauigkeit, da zwar die gleichen Klassen gemeint sind, diese allerdings falsch zugeordnet wurden.

Durch das Aufteilen in Test- und Trainingsdatensatz kann es vorkommen, dass einzelne Klassen im Trainingsdatensatz nicht vertreten sind. Im Trainingsdatensatz fehlen dementsprechend einzelne Klassen, was zu Fehlern bei der Ausführung der Detektoren führt, da diese damit nicht umgehen können. Damit dieses Problem nicht auftritt, wird vor dem Anwenden der Detektoren überprüft, ob jede Klasse im Trainingsdatensatz vertreten ist. Wenn eine Klasse im Trainingsdatensatz nicht vertreten ist, wird ein Datenpunkt dieser Klasse aus dem Testdatensatz genommen und dem Trainingsdatensatz hinzugefügt. Das kann dazu führen, dass es bei einer hohen Gewichtung der Majority Klassen kaum Datenpunkte der Minority Klassen im Testdatensatz gibt.

3.5 Metriken

Damit die Genauigkeit der Verfahren bestimmt und verglichen werden kann, werden bestimmte Metriken eingeführt, die entweder die Genauigkeit des Random Forest oder eines Detektors messen. Außerdem wird der Gini-Index in Abschnitt 3.5.2 vorgestellt, der den Grad der Ungleichverteilung eines Datensatzes bestimmen kann.

3.5.1 Log-Loss

Die Metrik, welche die Genauigkeit der Detektoren bestimmt, untersucht die ausgegebenen Konfidenzwerte der Detektoren. Dabei sind die Label Noise bekannt und können mit den Konfidenzwerten verglichen werden.

Dazu wird zuerst ein Array gebildet, das markiert welche Datenpunkte Label Noise sind. Hierfür werden Label Noise mit einer 0 und richtige Label mit einer 1 markiert. Im Anschluss werden die Konfidenzwerte des Detektors mit diesem Array verglichen und die Differenzen berechnet. Je größer diese Differenzen sind, desto schlechter sind die Konfidenzwerte. Die Log-Loss-Metrik [Bis06] ist nur definiert für zwei oder mehr Label und findet auch Verwendung bei der logistischen Regression. Dabei haben die Label in diesem Zusammenhang die Bedeutung: Label Noise und kein Label Noise. Diese werden wie erwähnt in einem Array gespeichert. Für einen Datenpunkt mit einem Label $y \in \{0, 1\}$ und für die Wahrscheinlichkeit $p = P(y = 1)$, dass es sich bei dem Datenpunkt um ein richtiges Label handelt, gilt:

Definition 3.5.1 (Log-Loss)
$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$

Wenn der Detektor alle Label Noise perfekt vorhergesagt hat, beträgt der Log-Loss $0,0$. Bei dieser Funktion werden allerdings Minority Klassen nicht gleich gewichtet wie Majority Klassen. Die Genauigkeit der Detektoren ist daher auf die Gesamtzahl der Datenpunkte und nicht auf die Datenpunkte je Klasse bezogen.

3.5.2 Gini-Index

Der Gini-Index kann als Maß der Ungleichverteilung von Klassenlabeln eines Datensatzes genutzt werden. Er wird im Folgenden verwendet um den Grad der Ungleichverteilung eines Datensatzes festzustellen und dessen Auswirkung auf die Messergebnisse. Dafür kann er Werte zwischen 0 und 1 annehmen, wobei der Wert 0 bedeutet, dass alle Datenpunkte auf alle Klassen gleich verteilt sind und der Wert 1 bedeutet, dass eine Klasse alle Datenpunkte besitzt. Für die Berechnung des Gini-Indexes wird die Lorenz-Kurve verwendet [Gas72].

3.5.3 Genauigkeits-Funktionen

Die Genauigkeits-Funktion wird verwendet, um die Genauigkeit des Random Forest zu berechnen. Dafür werden die korrekten Label mit den durch den Random Forest vorhergesagten Label verglichen. Als Ergebnis gibt der Random Forest ein Array pro Datenpunkt aus. In diesen Arrays steht für jede Klasse die Wahrscheinlichkeit für die Klassenzugehörigkeit eines Datenpunkts. Zusammen mit diesem Array wird die Genauigkeit berechnet. Dazu wird mit einem Parameter k bestimmt, wie viele der höchsten Wahrscheinlichkeiten verwendet werden. Der Parameter bestimmt außerdem den Namen der Genauigkeit A@K, wobei A für Accuracy steht, dessen Definition in Definition 3.5.2 steht.

Definition 3.5.2 (Genauigkeit A@K)

$$A@K = \frac{|Label \in k - WahrscheinlichstenLabel|}{|Label|}$$

Bei der Genauigkeit A@1 wird je Datenpunkt die höchste der Wahrscheinlichkeiten betrachtet und mit dem richtigen Label verglichen. Um den Wert für die Genauigkeits-Funktion zu bekommen, wird die Anzahl der richtig vorhergesagten Datenpunkte durch die Gesamtzahl der Datenpunkten geteilt. Die Ausgabe besagt somit, wie viele der Label prozentual richtig vorhergesagt wurden.

Wenn k größer als Eins ist, wird geschaut, ob das richtige Label in den k-wahrscheinlichen Label enthalten ist. Dieses Vorkommen wird über alle Datenpunkte gezählt und durch die Gesamtzahl der Datenpunkte geteilt.

3.5.4 Precision und Recall

Das Problem der Genauigkeitsfunktionen ist, dass sie die einzelnen Klassen nach der Anzahl der Datenpunkte gewichten. Zum Beispiel kann dies auftreten, wenn es eine Majority Klasse gibt, die 80% der Datenpunkte hat und weitere 20 Minority Klassen, auf die die restlichen 20% aufgeteilt sind. Ordnet ein Random Forest im Anschluss jedem Datenpunkt das Label der Majority Klasse zu, hat

dieser 80% der Label korrekt vorhergesagt und besitzt somit eine Genauigkeit von 80%, obwohl nur eine von 21 Klassen erkannt werden. Bei Szenarien mit Klassenungleichverteilung ist das korrekte Erkennen der Minority Klassen allerdings häufig genauso wichtig wie das korrekte Erkennen der Majority Klasse. Damit eine Metrik eine aussagekräftige Genauigkeit von Klassifikatoren auch bei einem hohen Grad an Klassenungleichverteilung bestimmt können Precision und Recall genutzt werden.

Um Precision und Recall zu definieren, wird das Ergebnis der Klassifikation in vier Kategorien aufgeteilt. Es wird jede Klasse x separat betrachtet. Dafür werden die durch die Klassifikation vorhergesagten Label v und tatsächlichen Label t mit der Klasse x verglichen. Wobei sich „Wahr“ darauf bezieht, dass das vorhergesagte Label dem tatsächlichen Label entspricht (richtige Vorhersage). „Positiv“ bezieht sich dagegen darauf, dass das vorhergesagte Label zur betrachteten Klasse gehört. Es entstehen die folgenden vier Kategorien:

- **Wahr-Positiv**$_x$: $v == x \;\&\; t == x$
- **Wahr-Negativ**$_x$: $v \mathrel{!}= x \;\&\; t \mathrel{!}= x$
- **Falsch-Positiv**$_x$: $v == x \;\&\; t \mathrel{!}= x$
- **Falsch-Negativ**$_x$: $v \mathrel{!}= x \;\&\; t == x$

Bezogen auf eine Klasse werden somit alle ihr zugeordneten Datenpunkte angeschaut. Alle Datenpunkte, die tatsächlich zur Klasse gehören und richtig zu dieser Klasse zugeordnet wurden, sind **Wahr-Positive** Datenpunkte. Datenpunkte, die einer betrachteten Klasse nicht angehören und dieser auch nicht zugeordnet wurden, sind **Wahr-Negative** Datenpunkte. Die Datenpunkte, die nicht zu einer Klasse gehören, dieser Klasse jedoch fälschlicherweise zugeordnet wurden, sind **Falsch-Positive** Datenpunkte, da das vorhergesagte Label der Klasse entspricht, allerdings nicht das tatsächliche Label. Solche Datenpunkte, die einer Klasse eigentlich zugehörig sind, dieser jedoch nicht zugeordnet wurden, sind **Falsch-Negative** Datenpunkte. In diesem Fall entspricht das tatsächliche Label der Klasse, jedoch nicht das vorhergesagte.

Mithilfe dieser Kategorien kann Precision und Recall definiert werden:

Definition 3.5.3 (Precision)

$$Prec_x = \frac{Wahr\text{-}Positiv_x}{Wahr\text{-}Positiv_x + Falsch\text{-}Positiv_x}$$

Definition 3.5.4 (Recall)

$$Rec_x = \frac{Wahr\text{-}Positiv_x}{Wahr\text{-}Positiv_x + Falsch\text{-}Negativ_x}$$

Wenn der Random Forest einen Datenpunkt der Klasse y mit dem Label z klassifiziert hat, ist dies ein Falsch-Positiver Datenpunkt bezüglich der Klasse y und ein Falsch-Negativer Datenpunkt bezüglich der Klasse z. Wenn der gesamte Datensatz betrachtet wird, treten Falsch-Negativ und Falsch-Positiv Datenpunkte symmetrisch auf. Deswegen müssen die Klassen einzeln betrachtet werden und für diese Precision und Recall berechnet werden, um im Anschluss die Werte zusammen zu addieren und durch die Anzahl Klassen zu teilen. Precision und Recall können Werte zwischen 0 und 1 annehmen, wobei 0 eine geringe und 1 eine hohe Genauigkeit darstellt.

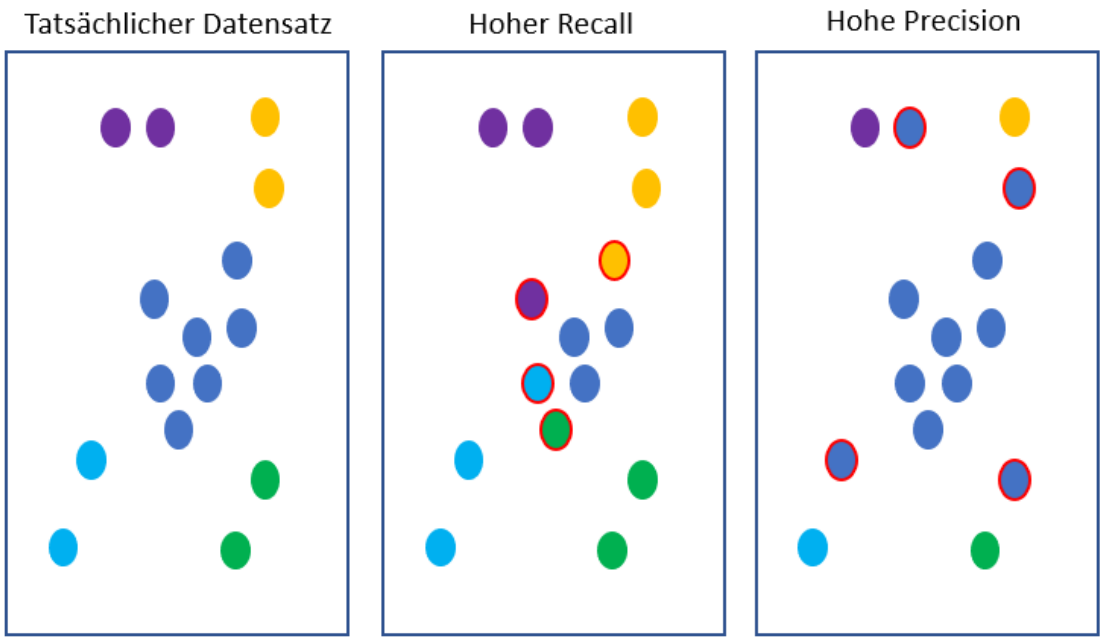

Abbildung 3.3: Visualisierung von Precision und Recall an einem Beispieldatensatz

In Abbildung 3.3 ist die Visualisierung eines Beispieldatensatzes gezeigt. Auf der linken Seite sind die tatsächlichen Label abgebildet, während die mittlere und rechte Abbildung eine mögliche Klassifikation darstellen.

In „Hohe Precision“ in Abbildung 3.3 ist ein hoher **Precision**-Wert dargestellt. Es ist zu sehen, dass eine dunkelblaue Majority Klasse erkannt wird und viele Minority Klassen (Lila, Gelb, Grün und Hellblau) schlecht erkannt werden. In Definition 3.5.3 ist zu sehen, dass der Precision-Wert hoch ist, wenn wenige Klassen mit wenigen Falsch-Positiven Datenpunkten aus der Klassifikation entstehen. Dies bedeutet wie in der Abbildung rechts zu sehen, dass es viele Klassen gibt, denen zu wenige Datenpunkte zugeordnet werden und es eine Klasse gibt, der zu viele Datenpunkte zugeordnet werden. Daher gibt es viele Minority Klassen mit einem Falsch-Negativen Datenpunkt und eine Majority Klasse mit vier Falsch-Positiven Datenpunkten. Es gibt somit eine Klasse mit niedrigem Precision-Wert und viele Klassen mit hohem Precision-Wert. Daraus folgt im Durchschnitt ein hoher Precision-Wert für das Klassifikationsergebnis, obwohl es absolut betrachtet gleich viele Falsch-Positive wie Falsch-Negative Datenpunkte gibt.

In „Hoher Recall“ in Abbildung 3.3 ist dagegen ein hoher **Recall**-Wert dargestellt. Es ist eine Majority Klasse (die dunkel blaue Klasse) zu sehen, die kaum erkannt wird, wohingegen viele Minority Klassen (Lila, Gelb, Hellblau und Grün) erkannt werden. In Definition 3.5.4 ist zu sehen, dass der Recall-Wert hoch ist, wenn wenige Klassen mit wenigen Falsch-Negativen Datenpunkten aus der Klassifikation entstehen. Dies bedeutet, dass es wie in der Abbildung viele Klassen gibt, denen zu viele Datenpunkte zugeordnet werden, aber es auch eine Klassen gibt, der zu wenige Datenpunkte zugeordnet werden. Insgesamt existieren somit viele Klassen mit mehr Falsch-Positiven Datenpunkten und somit einem hohen Recall-Wert.

4 Evaluation

In der Evaluation werden die vorgestellten Detektoren in den beiden Szenarien untersucht und mit S-Baseline verglichen. Zuerst wird in Abschnitt 4.1 beschrieben mit welchen Hilfsmitteln die Versuche durchgeführt werden. Im Anschluss werden die Ergebnisse zweier Versuchsreihen vorgestellt. Die erste Versuchsreihe wird in Abschnitt 4.2 evaluiert. Diese nutzt die „make_blobs“-Methode, um die Parameter der Datensätze gezielt verändern zu können und die beeinflussten Herausforderungen isoliert zu untersuchen. Dazu wird die erste Versuchsreihe in drei Versuche aufgeteilt. Die beiden ersten Versuche unterscheiden sich in der Gewichtung der Majority Klasse, während im dritten Versuch die Klassenanzahl untersucht wird.

Die Ergebnisse der zweiten Versuchsreihe werden in Abschnitt 4.3 vorgestellt. Dazu wird der ImbalanceDataGenerator verwendet, der Datensätze generiert, die alle Herausforderungen besitzen. Im Anschluss wird in Abschnitt 4.4 eine Schlussfolgerung gegeben.

4.1 Versuchsaufbau

In diesem Abschnitt wird beschrieben, wie die Experimente durchgeführt werden. Damit die Detektoren und der Random Forest getestet werden können, wird mit einem Computer mit einem 16GB RAM und einem „i5-7400“ Prozessor von Intel gearbeitet. Als Betriebssystem wird Windows 10 verwendet, um mit Visual Studio Code mit Python 3.9.4 zu programmieren. Hierbei werden Funktionen der SciKit-Learn Bibliotheken importiert.

An den verschiedenen Datensätzen die in Abschnitt 3.1 genauer beschrieben wurden, werden die unterschiedlichen Metriken aus Abschnitt 3.5 angewendet. Die Messergebnisse werden mit Pandas[1] in eine Excel-Datei exportiert. Die Excel-Datei besitzt sowohl alle wichtigen Informationen, die den Datensatz beschreiben, als auch die Ergebnisse der Metriken, die die Genauigkeit des Random Forest oder der Detektoren angeben.

Im ersten Schritt werden Datensätze generiert und mit einem Imputer ausgebessert, woraufhin die fünf ausgewählten Detektoren in den in Abschnitt 3.4 vorgestellten Szenarien getestet werden. Zu den Szenarien gehören:

- **S-Baseline:** Der Random Forest wird auf den Datensatz angewendet ohne, dass davor Detektoren auf diesen angewendet wurden.
- **S-Löschen:** Bevor der Random Forest angewendet wird, werden die durch den Konfidenzwert des Detektors wahrscheinlichen Label Noise aus dem Datensatz gelöscht.

[1] https://pandas.pydata.org/

- **S-Gewichten:** Die Ausgabe des Detektors wird dem Random Forest mitgegeben, damit dieser die Datenpunkte dementsprechend gewichten kann.

Anschließend wird die Genauigkeit des Random Forest in jedem Szenario gemessen. Dabei hat der Random Forest als Parameter „n_estimators“ den Wert 50.

Um die Detektoren zu testen, gibt es zwei unterschiedliche Versuchsreihen. Diese unterscheiden sich in den generierten Datensätzen. In der ersten Versuchsreihe werden mit der „make_blobs“-Methode iterativ Datensätze erstellt, damit einzelne Datencharakteristika gezielt in Isolation untersucht werden können. Während in der zweiten Versuchsreihe der ImbalanceDataGenerator verwendet wird, um einen Datensatz zu generieren, der alle Herausforderungen abdeckt.

Als Metriken werden die in Abschnitt 3.5 vorgestellten Metriken verwendet. Um die Genauigkeit der Detektoren zu messen, wird der Log-Loss berechnet und zu jedem Detektor jedes Datensatzes gespeichert. Je schlechter die Detektoren die Datenpunkte einordnen, desto höher wird der Log-Loss Wert. Die Genauigkeit des am Ende verwendeten Random Forest Klassifikator wird mit der in Abschnitt 3.5.3 beschriebenen Genauigkeitsfunktion bestimmt. Dies ist die Genauigkeit A@1 in der ersten Versuchsreihe und Genauigkeit A@1 bis A@10 in der zweiten Versuchsreihe.

4.2 Versuchsreihe mit „make_blobs“

In der ersten Versuchsreihe wird mit der „make_blobs“-Methode ein Datensatz erstellt. Dabei werden die folgenden Eigenschaften festgesetzt:

- Attribute: 100
- Testdatensatz: 30% der Datenpunkte Durchschnittlich 10 Datenpunkte pro Klasse

Die Datenpunkte sind durchschnittlich festgelegt, dass heißt die Datenpunktanzahl variiert zwar mit der Klassenanzahl, es sind allerdings in jedem Datensatz durchschnittlich 10 Datenpunkte pro Klasse. Die restlichen Eigenschaften variieren und dienen als Vergleich um die Einwirkung der einzelnen Eigenschaften festzustellen. Die in Tabelle 4.1 dargestellten Eigenschaften sind variabel. Die erste genannte Eigenschaft ist die Anzahl an Label Noise, die durch Vertauschen von Labeln erhöht wird und somit die Herausforderung Label Noise beeinflusst. Die zweite Eigenschaft ist die Streuung, die beschreibt wie sehr die Datenpunkte um die Zentren der Klasse gestreut werden. Bei einem hohen Streuungsgrad, in diesem Fall 10, überschneiden sich mehr Klassen als bei einem niedrigen Streuungsgrad, da mehr Datenpunkte weiter von ihrem Zentrum entfernt liegen können.

Die Abbildung 4.1 visualisiert die Gewichtung von Beispielklassen. Es sind sechs Klassen dargestellt, wobei auf der linken Abbildung die Majority Klasse mit 50% gewichtet werden und auf der rechten mit 80%. Die Gewichtung der Majority Klasse zeigt, wie viel Prozent der Datenpunkte einer einzigen Klassen zugeordnet sind. Die restlichen Datenpunkte werden gleich auf die anderen Klassen verteilt. Dabei wird die erste Klasse als Majority Klasse gewählt. Dies beeinträchtigt die Klassenungleichverteilung, da bei höherer Gewichtung weniger Datenpunkte den Minority Klassen zugeordnet werden. Sobald weniger Datenpunkte den Minority Klassen zugeordnet werden, wird auch die Herausforderung wenige Datenpunkte je Klasse beeinflusst, da alle Klassen außer der Majority Klasse bei höherer Gewichtung weniger Datenpunkte besitzen.

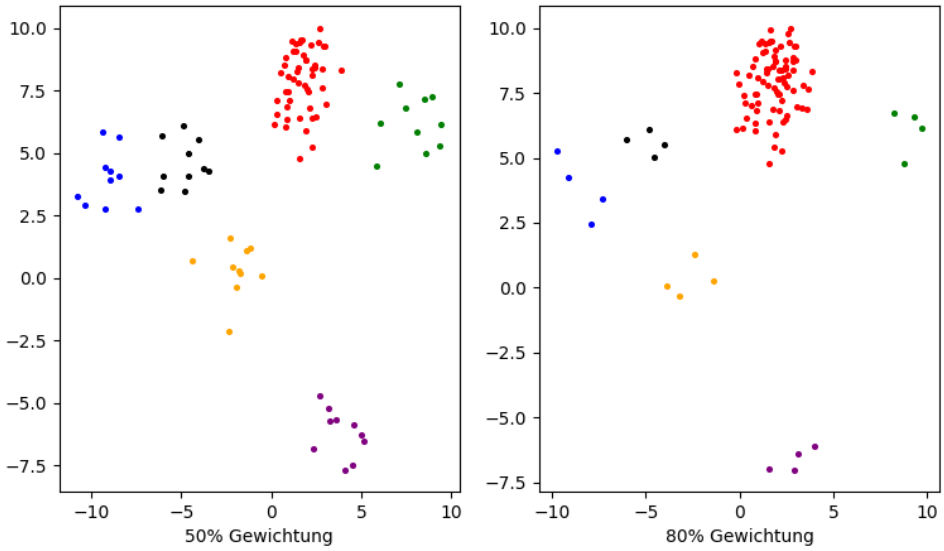

Abbildung 4.1: Visualisierung der Gewichtung von fünf Minority und einer Majority Klasse mit zwei Attributen

Eigenschaften	**Werte**	**Beeinflusste Herausforderung**
Anzahl an Label Noise	10%, 30%	Label Noise
Streuung	1, 5, 10	Sich überschneidende Klassen
Gewichtung der Majority Klasse	50%, 70%, 90%	Klassenungleichverteilung & Wenige Datenpunkte je Klasse
Klassenanzahl	50, 100, 150	Klassenungleichverteilung

Tabelle 4.1: Variable Eigenschaften und die beeinflussten Herausforderungen

Des Weiteren beeinträchtigt die Klassenanzahl die Klassenungleichverteilung. Dabei ist zu beachten, dass die Datenpunktanzahl mit der Klassenanzahl steigt und es in jedem Fall durchschnittlich 10 Datenpunkte je Klasse gibt. Allerdings wird die Anzahl der Minority Klassen erhöht, welche die Klassenungleichverteilung erhöhen.

Wenn alle dargestellten variablen Eigenschaften miteinander verknüpft werden entstehen 54 unterschiedliche Datensätze, die in dieser ersten Versuchsreihe untersucht werden. Um die einzelnen Eigenschaften zu untersuchen, werden alle Datensätze mit dieser einen gleichen Eigenschaft zusammengefasst, indem für diese ein Durchschnittswert der Genauigkeiten berechnet wird. Dabei wird die Genauigkeit des Random Forest mit der A@1 Genauigkeit gemessen. Die vielfältiges Produkt-Portfolio Herausforderung wird in dieser Versuchsreihe noch nicht betrachtet.

Des Weiteren ist bei allen Versuchen zu beachten, dass bei dem Random Forest ein Random-State von 6 gewählt wird. Bei einem anderen Random-State können die Ergebnisse um einige Prozentpunkte abweichen.

Wie erwartet erzielt der MCS-Detektor keine Ergebnisse für die vorliegenden Mehrklassenprobleme und wird somit nicht in der Evaluation betrachtet. Die Ausgabe des MCS-Detektors ist ein Array, dass jeden Datenpunkt zu 100% gewichtet. Es gibt dementsprechend keine Veränderung zu S-Baseline.

4.2.1 Versuch mit 44% Gewichtung

In diesem Versuch wird die Gewichtung der Majority Klassen auf 44% gesetzt. Dabei werden die 44% Datenpunkte, wie im Datensatz von [HRM19], auf 10 Majority Klassen verteilt und die anderen 56% auf die restlichen 70 Klassen gleichmäßig verteilt. Die generierten Datensätze haben dementsprechend folgende Eigenschaften:

- Datenpunkte: 800
- Anzahl Klassen: 80
- Gewichtung: 44% auf 10 Klassen
- Anzahl an Label Noise: 10%, 30%, 50%
- Streuung: 1, 5, 10

Indem Label Noise und Streuung kombiniert werden, entstehen neun Datensätze. Auf diese Datensätze werden die Detektoren und anschließend der Random Forest angewendet. Die variablen Eigenschaften werden im Anschluss mit Durchschnittswerten der Genauigkeit einzeln betrachtet:

Streuung Die Streuung hat einen großen Einfluss auf die Genauigkeit. Wie in Abbildung 4.2 dargestellt, sinkt die Genauigkeit von 90% auf unter 40%. Dies liegt daran, dass die Klassen bei hohem Streuungsgrad kaum zu unterscheiden sind und sich die Attributwerte sehr stark überschneiden. Die Anzahl sich überschneidender Klassen wird somit sehr groß und sorgt dafür, dass es weniger Muster gibt, die erkannt werden können. Außerdem sorgt dies dafür, dass weniger Minority Klassen erkannt werden. Im Hinblick auf die beeinflusste Herausforderung sich überschneidender Klassen ist zu sehen, dass diese die Genauigkeit erheblich verschlechtert. Die einzige Ausnahme ist der Partitioning Detektor, denn seine Genauigkeit ist bereits bei niedrigem Streuungsgrad schlecht und stagniert auch auf diesem niedrigen Level. **Der Partitioning-Detektor hat die geringste Genauigkeit, während der Instance-Hardness Detektor in S-Löschen S-Baseline bei hohem Streuungsgrad mit einer Differenz von 4%-Punkten überholt.**

Label Noise Im Hinblick auf die Menge an Label Noise wird Abbildung 4.3 erstellt. Dafür wird, wie in Abschnitt 4.2.2, die Genauigkeit A@1 verwendet. **Auffällig ist, dass der Instance-Hardness Detektor in allen Werten, in S-Löschen, mit bis zu 73% am Besten abschneidet.** Dabei ist er bis zu 0,5%-Punkte besser als S-Baseline. Bei hoher Anzahl an Label Noise gibt es mehrere Detektoren, KDN in S-Löschen und Instance-Hardness, die besser als S-Baseline abschneiden.

Insgesamt verringert sich die Genauigkeit um bis zu 20%-Punkte bei einer Erhöhung der Label Noise Anzahl von 10% auf 50%. Dabei sinkt die Genauigkeit auf fast 44% und zeigt, dass alle Detektoren Schwierigkeiten haben, eine hohe Anzahl an Label Noise richtig zu erkennen. Die davon betroffene Herausforderung Label Noise sorgt somit dafür, dass sich die Genauigkeit des Random Forest verschlechtert.

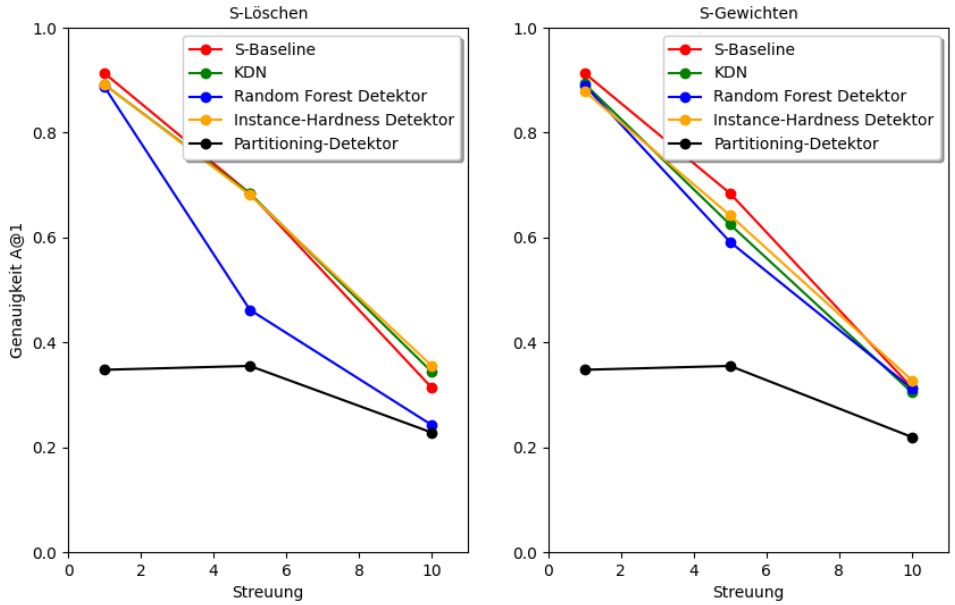

Abbildung 4.2: Untersuchung der Streuung in der ersten Versuchsreihe mit den Datensätzen mit 44% Gewichtung

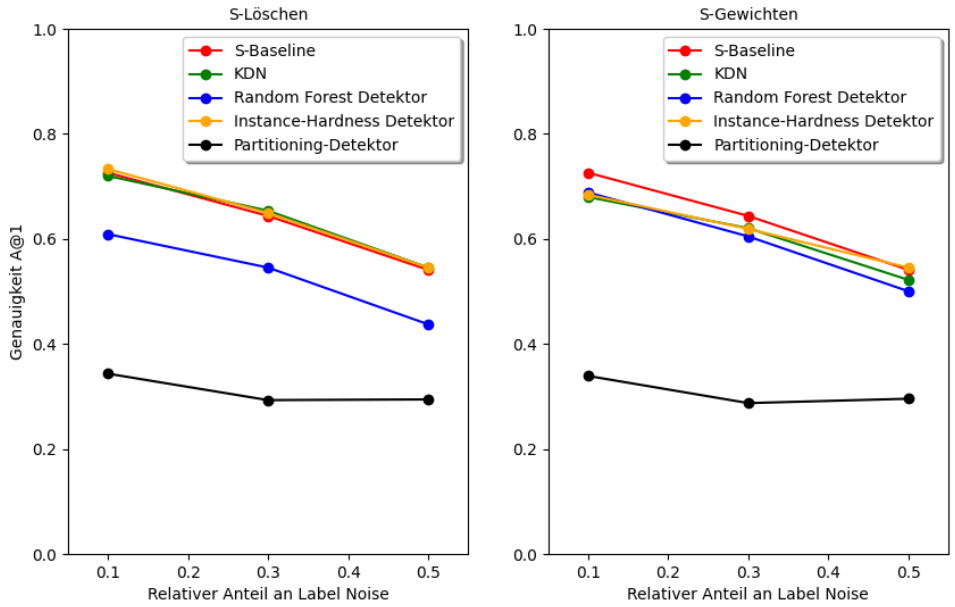

Abbildung 4.3: Untersuchung der Label Noise Anzahl in der ersten Versuchsreihe mit 44% Gewichtung

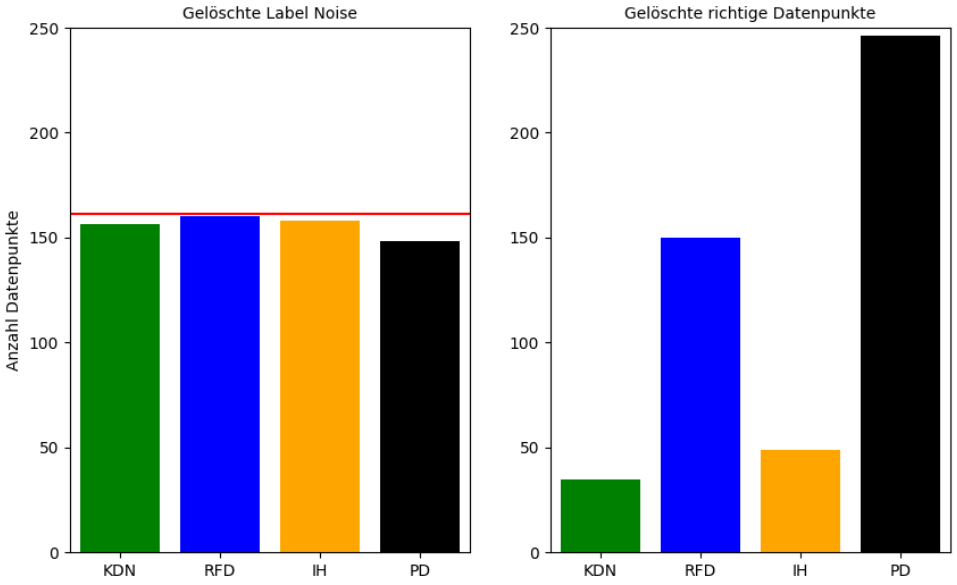

Abbildung 4.4: Untersuchung Anzahl gelöschter Datenpunkte in S-Löschen

Beim Vergleich der Szenarien schneidet nur der Random-Forest-Detektor in S-Löschen um bis zu 8%-Punkte schlechter ab als in S-Gewichten. Die anderen Algorithmen haben sehr ähnliche Genauigkeiten über die Szenarien hinweg, obwohl sie durchschnittlich in S-Gewichten 3%-Punkte schlechter sind als in S-Löschen.

Der Partitioning-Detektor schneidet auffällig schlecht ab. In diesem Versuch ist die Genauigkeit mit 30% bis 35% geringer als die Gewichtung der Majority Klassen mit 44%. Daraus lässt sich schließen, dass sogar die Majority Klasse nicht erkannt wird. Mit Abbildung 4.4 ist dies erklärbar. Dabei wird nur das Szenario S-Löschen betrachtet, da nur in diesem Datenpunkte gelöscht werden. Auf dieser ist die Anzahl gelöschter Label Noise und die Anzahl gelöschter richtiger Datenpunkte (keine Label Noise) im Durchschnitt über alle Datensätze aufgezeigt. Daraus kann geschlossen werden, dass der Random Forest nach dem Anwenden des Partitioning-Detektors nicht nur mit dem Erkennen von Minority Klassen Schwierigkeiten hat, sondern auch mit dem Erkennen von Majority Klassen, da 246 Datenpunkte zu viel als Label Noise detektiert werden. Beim Partitioning Detektor sind fast doppelt so viele richtig gelabelte Datenpunkte, wie Label Noise gelöscht. Da die Datenpunkte wegen des geringen Konfidenzwerts gelöscht werden, lässt sich darauf schließen, dass auch die Menge gelöschter Datenpunkte in S-Gewichten kaum gewichtet werden.

Insgesamt werden mithilfe der Detektoren mit bis zu 160 von 161 Datenpunkten nahezu alle Label Noise aus dem Trainingsdatensatz entfernt. Dies ist anhand der Abbildung 4.4 zu sehen, da die Anzahl an Label Noise mit der roten Linie visualisiert ist. Dabei löschen der Instance-Hardness Detektor und KDN am wenigsten richtige Label, obwohl sie, wie die anderen Detektoren, fast alle Label Noise löschen.

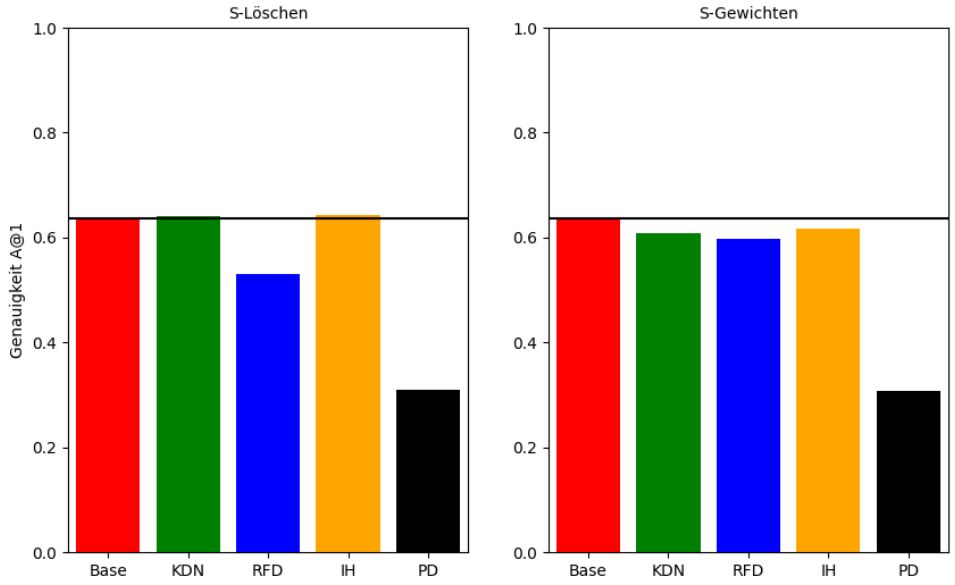

Abbildung 4.5: Untersuchung der Detektoren in der ersten Versuchsreihe mit den Datensätzen mit 44% Gewichtung

Detektoren In Abbildung 4.5 sind die durchschnittlichen Genauigkeiten der Detektoren im S-Löschen Szenario (rechts) und im S-Gewichten Szenario (links) dargestellt. Die Genauigkeit des Random Forest in S-Baseline wird nur sehr knapp von der Genauigkeit des Instance-Hardness Detektors in S-Löschen überholt. S-Baseline hat eine Genauigkeit von 64% und ist deswegen nur einem Prozentpunkt schlechter als der KDN und Instance-Hardness Detektor aus S-Löschen oder S-Gewichten.

Wenn die Ergebnisse der Abbildung 4.4 mit den Genauigkeiten aus Abbildung 4.5 der Detektoren verglichen werden, fällt auf, dass der Instance-Hardness Detektor und der KDN mit 64% und 63% am besten abschneiden und auch die geringste Menge an richtigen Datenpunkten mit 35 und 48 gelöscht haben. Jedoch hat der KDN am wenigsten richtige Datenpunkte gelöscht, hat aber dennoch nicht die beste Genauigkeit. Der KDN ist um 0,2%-Punkte schlechter als der Instance-Hardness Detektor. Dies liegt daran, dass der Instance-Hardness Detektor mehr Label Noise gelöscht hat, und diese somit weniger Einfluss auf das Ergebnis des Random Forest haben.

4.2.2 Versuch mit bis zu 90% Gewichtung

In diesem Versuch werden alle variablen Eigenschaften der ersten Versuchsreihe untersucht und es gibt durchschnittlich 10 Datenpunkt pro Klasse. Die folgenden Eigenschaften werden variiert:

- Anzahl Klassen: 50, 100, 150
- Gewichtung: 50%, 70%, 90%
- Anzahl an Label Noise: 10%, 30%

- Streuung: 1, 5, 10

Wenn alle Eigenschaften in allen möglichen Kombinationen kombiniert werden entstehen 54 Datensätze. Die Bedeutung der einzelnen Eigenschaften wurde in Abschnitt 3.1 genauer beschrieben.

Bei diesen Datensätzen ist zu beachten, dass es durch die hohe Gewichtung von bis zu 90% dazu kommen kann, dass im Testdatensatz nur Datenpunkte der Majority Klasse vorhanden sind. Das liegt daran, dass es im Durchschnitt 10 Datenpunkte je Klasse gibt und die Minority Klassen deswegen bei einer 90% Gewichtung der Majority Klasse meistens nur einen Datenpunkt haben. Dieser Datenpunkt muss somit in den Trainingsdatensatz, da Klassen, die im Testdatensatz sind, aber nicht im Trainingsdatensatz vorkommen, nicht erkannt werden können. Dementsprechend werden viele der Datenpunkte der Minority Klassen aus dem Testdatensatz genommen und dem Trainingsdatensatz hinzugefügt. Bei einer Gewichtung von 90% sind daher keine Datenpunkte der Minority Klassen mehr im Testdatensatz.

Um die Auswirkung der Eigenschaften auf die Genauigkeit des Random Forest oder der Detektoren zu messen, werden die Datensätze je nach Eigenschaft mit Durchschnittswerten zusammengefasst. Die folgenden Eigenschaften wurden untersucht:

Streuung In Abbildung 4.6 sind die Genauigkeiten A@1 des Random Forest in Verbindung mit den Detektoren aufgezeigt. Auf der x-Achse ist der Grad der Streuung abgebildet, während auf der y-Achse die A@1 Genauigkeit abgebildet ist. Das erste der drei Schaubilder zeigt die Genauigkeiten des Random Forest, wie er in S-Baseline ohne Einwirkung der Detektoren abschneidet. Das zweite und dritte Schaubild zeigen die Genauigkeit des Random Forest, wenn die Ausgaben der Detektoren je nach Szenario mit einfließen. Dabei wurden die Genauigkeiten über die 54 Datensätze gemittelt.

Bei der Streuung ist klar erkennbar, dass die Genauigkeit des Random Forest in Verbindung mit allen Detektoren von circa 90% auf 70% sinkt. Wie in Abschnitt 4.2.1 zu sehen, steht dies in Verbindung mit der Herausforderung die sich überschneidender Klassen. Die 75% Genauigkeit spiegelt die Anzahl an Datenpunkten wider, die Majority Klasse im Durchschnitt besitzt. Die durchschnittliche Anzahl von 70% wird knapp übertroffen und es werden dementsprechend auch Minority Klassen erkannt, jedoch hauptsächlich Datenpunkte der Majority Klasse.

Die einzige Ausnahme ist der Random Forest in Verbindung mit dem Partitioning-Detektor. Denn die Genauigkeit ist bereits bei niedrigem Streuungsgrad schlecht und stagniert auf diesem niedrigen Level. Dieser schafft es, wie in Abschnitt 4.2.1, fast nur Datenpunkte der Majority Klasse zu erkennen.

Die Szenarien untereinander unterscheiden dagegen sich kaum. Die Genauigkeit aller Szenarien weist eine abfallende Tendenz auf, während in S-Baseline die Genauigkeit bei geringem Streuungsgrad im Vergleich zu den anderen Szenarien am höchsten ist. Bei hohem Grad an Streuung überholen dagegen die Genauigkeiten der Szenarien S-Löschen und S-Gewichten die Genauigkeit des S-Baseline Szenarios um 4%-Punkte. **Alle Detektoren außer der Random Forest Detektor schneiden bei hohem Streuungsgrad mit 75,2% sehr gut ab, während S-Baseline nur 71% hat.** Der beste Detektor, KDN, aus S-Gewichten schneidet um 0,1%-Punkte besser ab als die besten Detektoren aus S-Löschen.

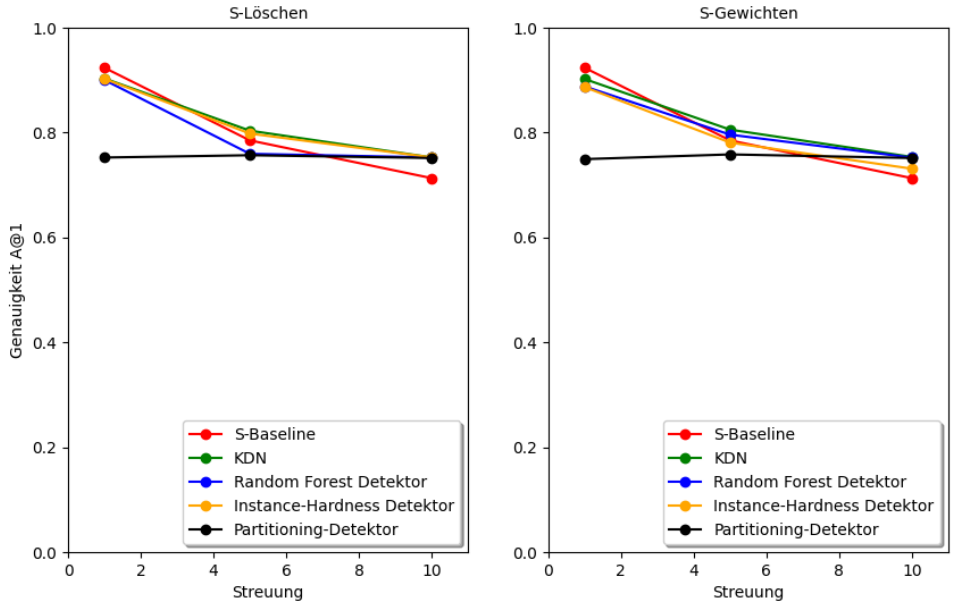

Abbildung 4.6: Untersuchung der Streuung in der ersten Versuchsreihe mit allen variablen Eigenschaften

Gewichtung In Abbildung 4.7, ist die Genauigkeit A@1 des Random Forest in Verbindung mit den unterschiedlichen Detektoren und Szenarien dargestellt. Dabei wurden in diesem Versuch die 54 Datensätze im Hinblick auf die Gewichtung gemittelt.

Bei einer Variation der Gewichtung ist ein Anstieg der Genauigkeit von 70% auf bis zu 100% zu beobachten. Da vor allem Datenpunkte der Majority Klasse im Testdatensatz sind, werden diese fast alle erkannt. Am besten schneiden der KDN bei hoher Gewichtung mit 99% in beiden Szenarien und der Instance-Hardness und Random Forest Detektor mit 100% bei hoher Gewichtung in S-Löschen ab.

Hierbei ist zu beachten, dass bei diesen Detektoren 95% bis 100% der Label Noise gelöscht werden. In Abbildung 4.8 ist die Anzahl gelöschter Datenpunkte zu sehen. Auf der linken Seite sind die gelöschten Label Noise abgebildet, wobei die rote Linie die Gesamtzahl an Label Noise darstellt. Auf der rechten Seite ist zu sehen, wie viele richtige Datenpunkte (keine Label Noise) gelöscht werden. Es ist somit zu sehen, dass mithilfe der Detektoren mit 150 von 151 Label Noise fast alle Label Noise gelöscht werden. Im Testdatensatz befinden sich daher fast nur Datenpunkte der Majority Klasse und es ist deswegen für den Random Forest möglich, eine Genauigkeit von 100% zu erreichen, da Label Noise entfernt wird.

Allerdings werden, wie in der rechten Abbildung zu sehen, auch bis zu 151 richtige Datenpunkte gelöscht. Vermutlich werden demnach 50% der Minority Klassen gelöscht, da diese nur einfach vertreten sind. Es können aber auch vereinzelt Datenpunkte der Majority Klasse gelöscht worden sein.

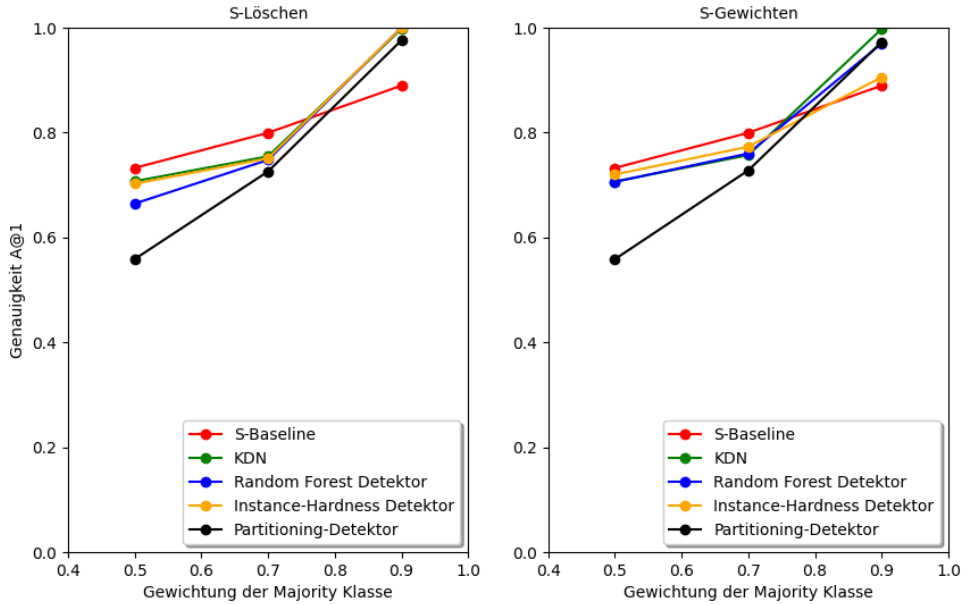

Abbildung 4.7: Untersuchung der Gewichtung in der ersten Versuchsreihe mit allen variablen Eigenschaften

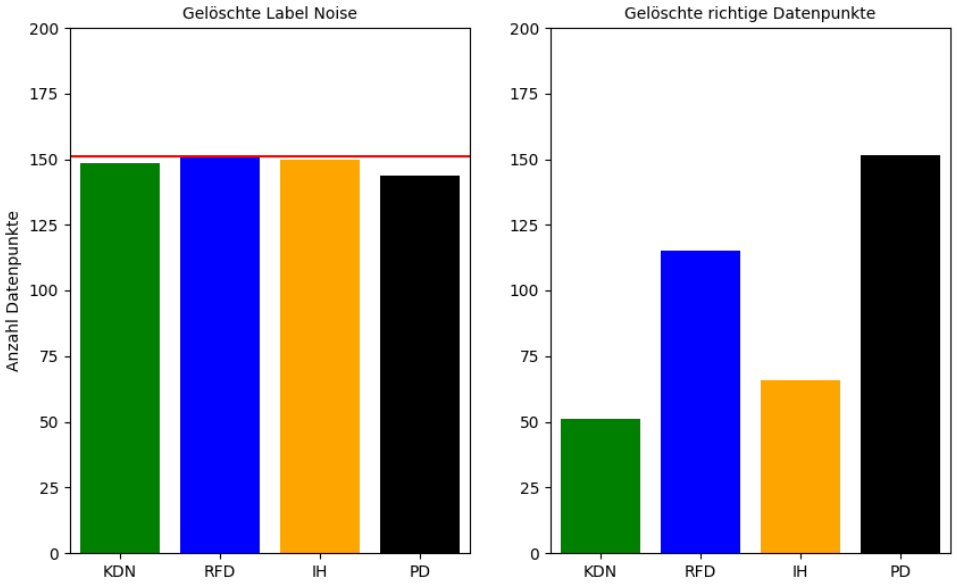

Abbildung 4.8: Untersuchung der gelöschten Datenpunkte in S-Löschen

Der Partitioning-Detektor ist mit einer Genauigkeit von ungefähr 55%, bei geringer Gewichtung, am wenigsten genau. Dies liegt erneut daran, dass er nur die Majority Klasse erkennt. Im Anschluss lässt sich auch der Zusammenhang zwischen Gewichtung und Genauigkeit des Partitioning-Detektors erklären. Auch bei 70% beziehungsweise 90% Gewichtung der Majority Klasse hat er eine Genauigkeit von circa 70% beziehungsweise 90%.

Die Szenarien unterscheiden sich auch im Hinblick auf die Gewichtung kaum. Auffällig ist, dass **bei geringer Gewichtung S-Baseline am besten abschneidet, während bei hoher Gewichtung S-Löschen und S-Gewichten mit Instance-Hardness-, Partitioning- und Random Forest Detektor besser abschneiden.** Dies liegt daran, dass die Detektoren die Datenpunkte der Minority Klasse als Label Noise erkennen und diese dementsprechend kaum eine Rolle beim Erlernen der Muster spielen. Da bei hoher Gewichtung nur noch Datenpunkte der Majority Klasse im Testdatensatz liegen, werden diese auch besser klassifiziert. Das Szenario S-Löschen schneidet mithilfe des Random Forest oder des Instance Hardness Detektors mit 100% am besten ab, S-Gewichten kommt auch auf 99%, S-Baseline aber nur auf 88%.

Die Herausforderung der Klassenungleichverteilung, die beim Erhöhen der Gewichtung betroffen ist, verbessert somit zwar die Genauigkeit, verschlechtert allerdings das Erkennen der Minority Klassen.

Label Noise Wenn die relative Label Noise Anzahl von 10% auf 30% erhöht wird, verschlechtert sich die Genauigkeit kaum, und zwar nur von 82% auf 80%. Das liegt zum Teil daran, dass noch nicht alle Herausforderungen in den Datensatz eingebunden sind. Es kann jedoch auch daran liegen, dass im Testdatensatz bei 90% Gewichtung nur Datenpunkte der Majority Klasse vorhanden sind und diese trotz hoher Anzahl von Label Noise erkannt werden. Zwar sind auch Datenpunkte der Majority Klassen Label Noise, allerdings werden diese von den Detektoren gut erkannt.

Klassen Das Gleiche gilt für das Erhöhen der Klassenanzahl, welches auch nur eine Verschlechterung der Genauigkeit von 82,5% auf 80,0% zeigt. Da bereits wenige Datenpunkte je Klasse vorhanden sind hat auch das Erhöhen der Klassenanzahl kaum eine Auswirkung auf die Genauigkeit. **Die Herausforderungen Label Noise und wenige Datenpunkte je Klasse haben daher bei bereits vorliegender Klassenungleichverteilung und geringer Anzahl an Datenpunkten kaum einen Einfluss auf die Genauigkeit.**

Detektoren Im Anschluss werden die Ergebnisse aller Datensätze pro Detektor in Abbildung 4.9 gemittelt, damit diese besser verglichen werden können.

Auf der y-Achse ist die Genauigkeit A@1 aufgeführt, während auf der x-Achse die Detektoren abgebildet sind, die den Random Forest in den unterschiedlichen Szenarien beeinflussen. Die schwarze Linie zeigt die Genauigkeit des S-Baseline Szenarios, damit leichter erkennbar ist, welche der Detektoren aus S-Löschen und S-Gewichten besser abschneiden.

Auch bei dieser Grafik fällt wieder der Partitioning-Detektor auf, der aus den bereits genannten Gründen mit 75% die niedrigste Genauigkeit hat. Dagegen bewegen sich die anderen Detektoren, sowie die unterschiedlichen Szenarien in einem ähnlichen Wertebereich, von 80%. Die Algorithmen schneiden über die Szenarien hinweg gleich gut ab. Jedoch kann nicht festgestellt werden, ob das Entfernen der Label besser oder schlechter ist als das Gewichten der Datenpunkte, da der Random

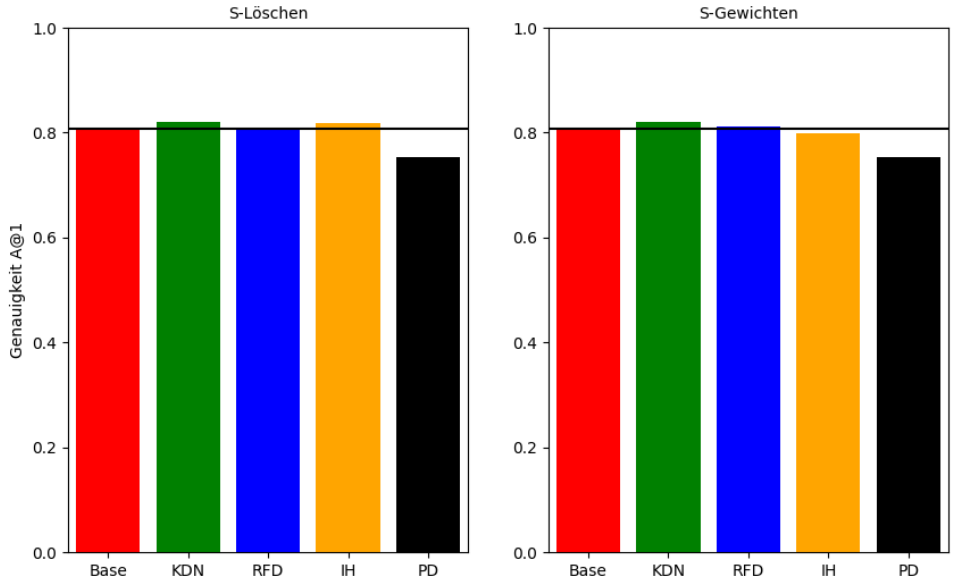

Abbildung 4.9: Untersuchung der Detektoren in der ersten Versuchsreihe mit allen variablen Eigenschaften

Forest Detektor in S-Löschen um einen Prozentpunkt schlechter ist, während der Instance-Hardness Detektor in S-Löschen um 2%-Punkte besser ist. **Am besten schneiden der KDN in S-Löschen und S-Gewichten mit 82,0% und der Instance-Hardness Detektor in S-Löschen mit 81,8% ab.** Dabei ist S-Baseline knapp 2%-Punkte schlechter als diese Detektoren.

Um den Zusammenhang zwischen der Ausgabe der Detektoren und der Genauigkeit des Random Forest zu überprüfen, wird der Log-Loss Wert gemessen. Der Log-Loss Wert misst die Ausgabe der Detektoren und beschreibt, wie nahe die Ausgabe der Detektoren an den tatsächlichen Labeln liegt. Genauer beschrieben wird dieser in Abschnitt 3.5.1.

In Abbildung 4.10 ist der Log-Loss Wert der Detektoren sowie die des Datensatzes ohne Detektoren (Base) dargestellt. Eine direkte Verbindung zwischen Genauigkeit und Log-Loss Wert ist wie erwartet zu sehen, da die Detektoren mit einem hohen Log-Loss Werte auch die niedrigste Genauigkeit in Abbildung 4.9 verursacht haben. **Allerdings hat der Instance-Hardness Detektor zwar den besten Log-Loss Wert von ungefähr 1, jedoch ist der KDN Detektor mit einer Genauigkeit von 81,9% besser als der Instance-Hardness Detektor mit einer Genauigkeit von 81,7%, obwohl dieser einen geringeren Log-Loss Wert hat.** Somit ist zwar eine Tendenz durch den Log-Loss Wert gegeben, jedoch keine eindeutige Korrelation. Der Grund dafür ist, dass der Log-Loss Wert nur eine Differenz berechnet. Es gibt allerdings Datenpunkte, die für einen Random Forest zur Mustererkennung wichtiger sind als andere Datenpunkte. Deswegen gibt es kein Maß, das beschreibt, wie wichtig der Datenpunkt für die Mustererkennung ist. Wenn ein Detektor viele unwichtige Datenpunkte, ein anderer Detektor jedoch wenige wichtige Datenpunkte richtig eingeschätzt hat, wird die Genauigkeit mithilfe des zweiten Detektors besser sein.

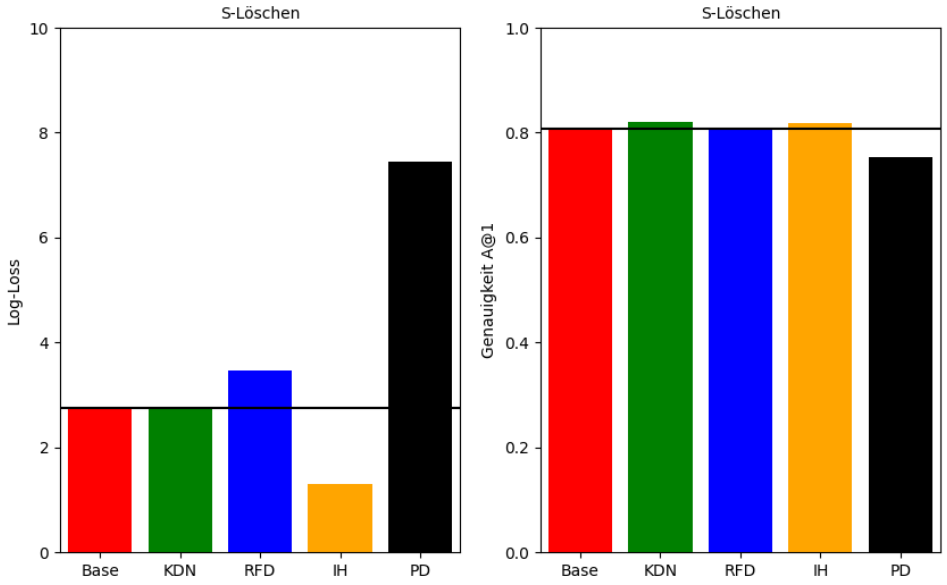

Abbildung 4.10: Gegenüberstellung des Log-Loss Werts mit der Genauigkeit A@1 in der ersten Versuchsreihe mit allen variablen Eigenschaften

4.2.3 Untersuchung der Klassenanzahl

Dadurch, dass die Gewichtung in Abschnitt 4.2.2 hoch ist, konnte der Einfluss der Klassenanzahl nicht richtig untersucht werden. Aufgrund dessen wird hierfür die Gewichtung wie bereits in Abschnitt 4.2.1 auf 44% reduziert, Die aktuelle Datensatzreihe hat folgende Parameter:

- Datenpunkte: Durchschnittlich 10 pro Klasse
- Anzahl Klassen: 50, 100, 150
- Anzahl an Label Noise: 10%, 30%, 50%
- Gewichtung: 44% auf 10 Klassen
- Streuung: 1, 5, 10

In Abbildung 4.11 wird die Genauigkeit der 27 Datensätze auf die Klassenanzahl gemittelt.

Bei einer Erhöhung der Klassenanzahl von 50 auf 150 Klassen sinkt die Genauigkeit des am besten abschneidenden Detektors KDN von 71% auf 63%. Dies liegt daran, dass es eine feste Anzahl an Majority Klassen gibt. Bei einer Erhöhung der Klassenanzahl wird daher die Anzahl Minority Klassen erhöht und diese werden schlechter erkannt als die Majority Klasse. Im Gegensatz zur Streuung aus Abschnitt 4.2.2, welche eine Verschlechterung der Genauigkeit von bis zu 50%-Punkten verursacht, ist die Verschlechterung von 10%-Punkten sehr gering. **Somit hat auch die Herausforderung wenige Datenpunkte je Klasse eine negative Auswirkung auf die Genauigkeit.**

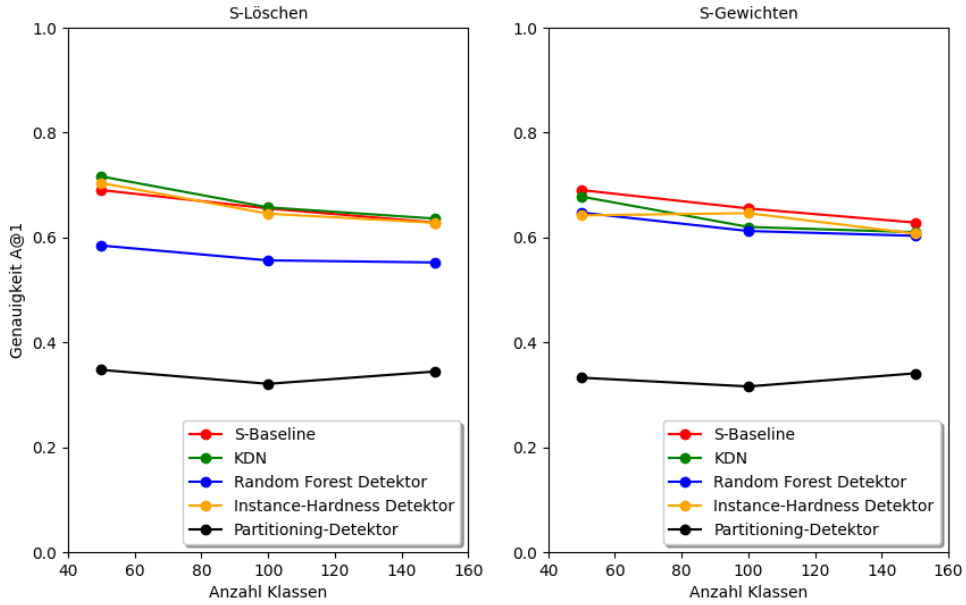

Abbildung 4.11: Untersuchung der Klassen in der ersten Versuchsreihe mit allen variablen Eigenschaften

4.2.4 Zusammenfassung

In diesem Abschnitt werden die Ergebnisse der ersten Versuchsreihe zusammengefasst. Die Besonderheiten der verwendeten Datensätze sind die große Variation der Parameter und die Möglichkeit, die Herausforderungen isoliert zu betrachten. Außerdem wird die Herausforderung des vielfältigen Produktportfolios nicht betrachtet.

Im **Versuch mit bis 44% Gewichtung** hat der Instance-Hardness Detektor mit 35%, bei hohem Streuungsgrad eine um 4%-Punkte bessere Genauigkeit als S-Baseline. Bei einer hohen Anzahl sich überschneidender Klassen überholt der Instance-Hardness Detektor somit S-Baseline.

Im **Versuch mit 90% Gewichtung** schneiden der KDN in S-Löschen und S-Gewichten und der Instance-Hardness Detektor in S-Löschen insgesamt am besten und somit mit durchschnittlich 82% besser als S-Baseline ab. Der Partitioning-Detektor erkennt nur die Datenpunkte der Majority Klassen und schneidet deswegen mit durchschnittlich 75% am schlechtesten ab. Bei Erhöhung der Label Noise Anzahl und dementsprechend einer Verstärkung der Herausforderung Label Noise sinkt die Genauigkeit aller Detektoren um bis zu 20%-Punkte. Unter Betrachtung der Label Noise Anzahl schneiden KDN und Instance-Hardness Detektor mit bis zu 73% in S-Löschen für jede Anzahl an Label Noise um 0.5%-Punkte besser als S-Baseline ab.

Im Anschluss wird noch der **Einfluss der unterschiedlichen Herausforderungen auf die Genauigkeit** der Szenarien in der gesamten Versuchsreihe geschildert. Bei Vergrößerung des Grades der Streuung und einer damit einhergehenden Verstärkung der Herausforderung sich überschneidender Klassen sinkt die Genauigkeit aller Detektoren ausgenommen der des Partitioning-Detektors um circa 50%-Punkte. Dagegen schneiden S-Löschen und S-Gewichten bei hohem Streuungsgrad mit

bis zu 75% um bis zu 5%-Punkte besser als S-Baseline ab. Das Erhöhen der Gewichtung und Verstärken der Herausforderung Klassenungleichverteilung führt dazu, dass nur Datenpunkte der Majority Klasse im Testdatensatz sind. Das führt im Versuch mit 90% Gewichtung dazu, dass der Instance-Hardness und Random Forest Detektor in S-Löschen eine Genauigkeit von 100% erreichen, während der Random Forest aus S-Baseline nur eine Genauigkeit von 90% erreicht. Aus der Untersuchung der Klassenanzahl kam heraus, dass auch die Herausforderung wenige Datenpunkte je Klasse eine negative Auswirkung auf die Genauigkeit um bis zu 10%-Punkte hat.

Im Hinblick auf die Szenarien schneidet S-Löschen besser als S-Gewichten ab. Wenn die vielfältiges Produktportfolio Herausforderung nicht im Datensatz vorhanden ist, lohnt es sich demnach, Detektoren im Zusammenhang mit S-Löschen zu nutzen.

4.3 Versuchsreihe mit dem ImbalanceDataGenerator

In dieser Versuchsreihe wird der Datensatz mit dem ImbalanceDataGenerator generiert. Die Datensätze haben folgende Parameter:

- Attribute: 100
- Testdatensatz: 33% der Datenpunkte
- Klassenanzahl: 84
- Datenpunkte: 1050

Dabei folgende Eigenschaften variiert:

- Anzahl an Label Noise: 10%, 20%, 30%
- Grad der Ungleichverteilung: sehr niedrig, niedrig, normal, hoch, sehr hoch

Insgesamt können die drei Label Noise Stufen und die fünf Grade an Ungleichverteilung kombiniert werden, um 15 Datensätze zu erhalten. Die Datensätze haben bei einigen Datenpunkten fehlende Attributwerte, die mit Imputern abgeschätzt werden müssen. Der Grad der Ungleichverteilung variiert zwischen folgenden Werten:

- „sehr niedrig “ : Gini Index - 0.325
- „niedrig “ : Gini Index - 0.42
- „normal “ : Gini Index - 0.51
- „hoch “ : Gini Index - 0.54
- „sehr hoch “ : Gini Index - 0.57

Dabei ist der Gini-Index ein Maß für die Ungleichverteilung und in Abschnitt 3.5.2 genauer beschrieben. Es ist zu beachten, dass es nicht möglich ist bei dem ImbalanceDataGenerator einen Random-State festzulegen. Dies wird umgangen indem mehrere Ausführungen zu einem Durchschnitt verrechnet werden.

Vor dem Anwenden der Detektoren und des Random Forest werden Imputer angewendet, um fehlende Attributwerte zu behandeln. Anhand der vorhandenen Attributwerte werden zum Beispiel die k nächstgelegenen Nachbarn betrachtet, aus denen die fehlenden Attributwerte abgeschätzt werden. Wie in Abschnitt 3.2 beschrieben, werden hierfür drei unterschiedliche Imputer verwendet. Im Folgenden werden als Erstes die unterschiedlichen Imputer verglichen. Diese werden einzeln auf einen Beispieldatensatz angewendet und die Genauigkeit des im Anschluss angewendeten Random Forest gemessen.

Imputer In dieser Arbeit werden der Simple-Imputer, der Miss-Forest und der KNN untersucht, die in Abschnitt 3.2 genauer beschrieben sind. Der KNN wird mit $k = 5$ ausgeführt und betrachtet somit die 5 nächsten Nachbarn, während der Simple Imputer den Wert 0 für fehlende Attributwerte einsetzt.

Um die Imputer zu vergleichen, wird ein Beispieldatensatz verwendet, der mit dem ImbalanceDataGenerator generiert wurde. Dazu sind die folgenden zwei Werte für die Eigenschaften festgelegt:

- Grad der Ungleichverteilung: normal
- Anzahl an Label Noise: 20%

Auffällig ist, dass die Imputer KNN und Simple-Imputer, im Gegensatz zum Miss-Forest Imputer, sehr schnell sind. Wie in Abschnitt 3.2 erwähnt, benötigt dieser mehr Rechenleistung und dementsprechend mehr Zeit um die fehlenden Attributwerte abzuschätzen.

In Tabelle 4.2 sind die Genauigkeitswerte des Random Forest abgebildet, nachdem der jeweilige Imputer und Detektor angewendet wurde.

Der Miss-Forest Imputer schneidet im Zusammenhang mit allen Detektoren um bis zu 6%-Punkte schlechter als der Simple-Imputer ab. Trotz der längeren Laufzeit ist der Miss-Forest Imputer nicht besser als die beiden anderen Imputer und wird daher in den weiteren Untersuchungen nicht betrachtet.

Anschließend werden KNN und Simple Imputer verglichen. Obwohl der Simple Imputer nur Nullen einfügt und keine Attributwerte der Umgebung nutzt hat er um bis zu 7%-Punkte bessere Ergebnisse als der KNN. Der Grund dafür ist, dass die fehlenden Attributwerte bei manchen Datenpunkten nicht vorhanden sein sollen. Werden beispielsweise die Zylindermotoren im Genauen betrachtet, haben die Motoren mit vier Zylindern weniger Sensordaten oder Attributwerte als Motoren mit sechs Zylindern. Die Algorithmen können deswegen anhand der Nullen erlernen, dass die Motoren mit vier nicht zu den Motoren mit sechs Zylindern gehören. Dies verhindert in gewissem Grad das Erlernen falscher Muster. **Um den Grad an Ungleichverteilung und Label Noise zu untersuchen, wird im Folgenden der Simple Imputer gewählt, da er die besten Genauigkeiten erzielt.**

Tabelle 4.2: Imputervergleich in einem Beispieldatensatz des ImbalanceDataGenerators

	Simple Imputer	MissForest	KNN
S-Baseline	**0,301587**	0,266667	0,269841
S-Löschen: KDN	**0,266667**	0,212698	0,225397
S-Löschen: Random Forest Detektor	**0,269841**	0,196825	0,196825
S-Löschen: Instance-Hardness Detektor	**0,269841**	0,215873	0,222222
S-Löschen: Partitioning-Detektor	**0,196825**	0,155556	0,215873
S-Gewichten: KDN	**0,285714**	0,24127	0,219048
S-Gewichten: Random Forest Detektor	**0,288889**	0,253968	0,24127
S-Gewichten: Instance-Hardness Detektor	**0,311111**	0,244444	0,253968
S-Gewichten: Partitioning-Detektor	**0,203175**	0,171429	0,196825

Grad der Ungleichverteilung Der Grad der Ungleichverteilung variiert zwischen „sehr niedrig“ mit einem Gini Index von 0,325 und „sehr hoch“ mit einem Gini Index von 0,57.

In Abbildung 4.12 werden die Ergebnisse für die unterschiedlichen Grade der Ungleichverteilung gezeigt. Die Genauigkeit erhöht sich um bis zu 50%, wenn der Grad der Ungleichverteilung von sehr niedrig auf sehr hoch erhöht wird. Dies liegt daran, dass bei einem sehr hohen Grad an Ungleichverteilung mehr Datenpunkte den Majority Klassen zugeordnet werden, welche vom Random Forest einfacher erkannt werden.

S-Baseline schneidet im Durchschnitt am besten ab, wenn die Herausforderung Klassenungleichverteilung verstärkt wird. **Bei einem sehr hohen Grad an Ungleichverteilung schneiden die Detektoren, Instance-Hardness Detektor und KDN mit 52%, besser als S-Baseline ab.** Zusätzlich schneidet S-Gewichten besser als S-Löschen ab. Dies ist am Random Forest Detektor bei sehr hoher Ungleichverteilung zu sehen, der nur 0,4%-Punkte unter S-Baseline liegt. Dieser ist dicht gefolgt vom Instance-Hardness Detektor und vom KDN. Dagegen ist, wie in den Versuchen zuvor zu sehen, dass der Partitioning-Detektor in beiden Szenarien um bis zu 11%-Punkte schlechter als die anderen Detektoren abschneidet. Die anderen Detektoren schneiden ungefähr gleich gut ab.

Label Noise In diesem Abschnitt wird die Auswirkung des Grades an Label Noise betrachtet. Dazu wird zu jedem Grad an Label Noise der Durchschnitt der Genauigkeitswerte berechnet.

In Abbildung 4.13 sind die über die Datensätze gemittelten Genauigkeiten dargestellt, wobei auf der linken Abbildung S-Löschen und auf der rechten Abbildung S-Gewichten abgebildet ist. Das Variieren der Anzahl an Label Noise von 10% auf 30% hat, wie in Abbildung 4.13 dargestellt, kaum eine Auswirkung auf die Genauigkeit des Random Forest. Die Genauigkeit des Random Forest in Verbindung mit einem beliebigen Detektor verschlechtert sich nur um bis zu 3%-Punkte. **Jedoch ist zu sehen, dass S-Baseline mit einer Differenz von minimal 2%-Punkten, in allen Graden an Label Noise mit circa 30%, besser abschneidet als die untersuchten Detektoren.**

Der im Durchschnitt am besten abschneidende Detektor ist der Instance-Hardness Detektor. Dieser schneidet bei allen Graden an Label Noise mindestens 1% besser als die anderen Detektoren ab. Dagegen schneidet, wie in den Versuchen davor zu sehen, der Partitioning-Detektor in beiden Szenarien um bis zu 11%-Punkte schlechter als die anderen Detektoren ab. Die anderen Detektoren

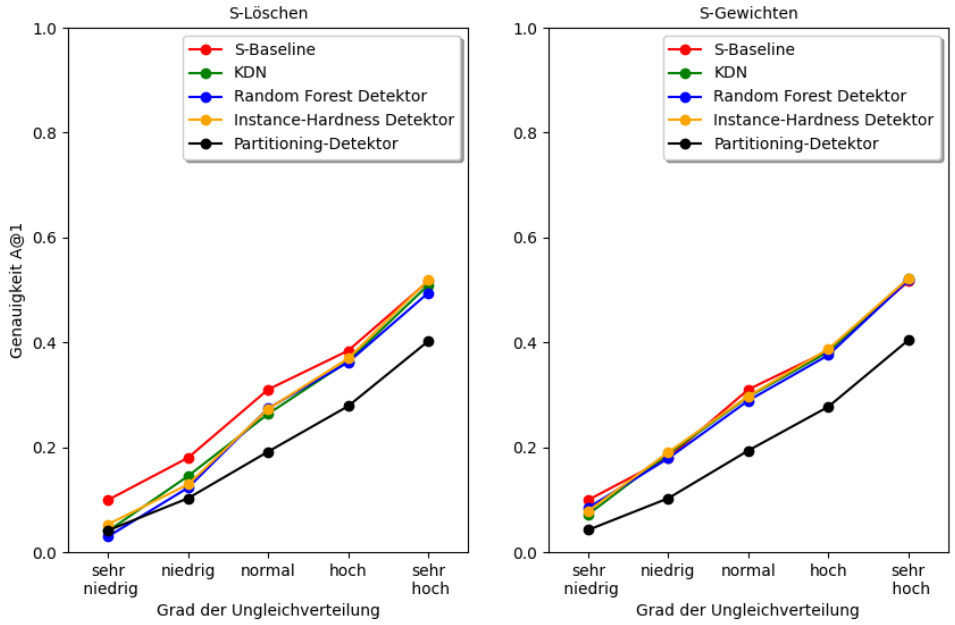

Abbildung 4.12: Untersuchung des Grads der Ungleichverteilung in der zweiten Versuchsreihe

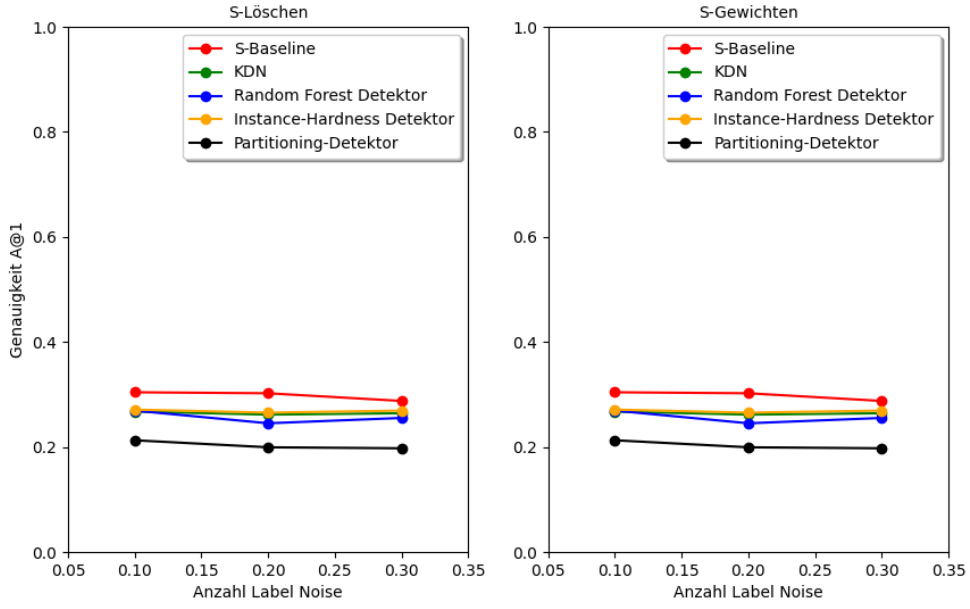

Abbildung 4.13: Untersuchung der Label Noise in der zweiten Versuchsreihe

	Log-Loss	A@1	Prec	Rec	Klassen
S-Baseline: Ohne Detektor	**9**	**29,8%**	**11,6%**	**14,6%**	**39**
S-Löschen: KDN	9	26,2%	7,2%	11,1%	19
S-Löschen: RandomForestDetector	9	26,0%	6,3%	10,2%	19
S-Löschen: InstanceHardness	**3**	**27,1%**	**8%**	**12,5%**	**23**
S-Löschen: PartitioningDetector	21	20,1%	2,8%	6,5%	9
S-Gewichten: KDN	9	**28,6%**	**10,3%**	**14,2%**	33
S-Gewichten: RandomForestDetector	9	27,7%	9,4%	13,2%	32
S-Gewichten: InstanceHardness	**3**	**28,6%**	**10,3%**	**14,2%**	**38**
S-Gewichten: PartitioningDetector	21	20,4%	2,7%	6,6%	9

Tabelle 4.3: Vergleich von Log-Loss, Genauigkeit A@1, Precision, Recall und Anzahl vorhergesagter Klassen

dagegen schneiden in S-Gewichten und S-Löschen, mit einer Varianz von 3%, fast gleich gut ab. Das Verstärken der Label Noise Herausforderung verringert die durchschnittliche Genauigkeit demnnach nur um wenige Prozentpunkte.

Log-Loss und A@K Da bisher nur die Genauigkeit A@1 betrachtet wurde, werden im Folgenden noch die Log-Loss Werte und die Genauigkeiten A@1 bis A@10 betrachtet. In Tabelle 4.3 sind die Detektoren mit den jeweiligen Genauigkeits-, Precision- und Recall-Werten dargestellt. Sie repräsentieren Durchschnittswerte über alle Datensätze hinweg. Auf die Precision- und Recall-Werte wird allerdings erst im nächsten Abschnitt eingegangen. Außerdem steht in der letzten Spalte die Anzahl der Klassen, die der Random Forest von durchschnittlich 76 Klassen vorhergesagt hat. Das heißt 76 abzüglich der $Klassen$ ergibt die Anzahl der Klassen, denen der Random Forest kein einziges Label zugewiesen hat.

Die Log-Loss Werte in Tabelle 4.3 zeigen, dass der Random Forest im Allgemeinen eine bessere Genauigkeit A@1 vorzuweisen hat, wenn der Log-Loss Wert gering ist. Auffällig ist jedoch, dass der ursprüngliche Datensatz mit Label Noise einen Log-Loss Wert von 9 und der Instance-Hardness Detektor in S-Löschen einen Log-Loss Wert von 3 hat. Obwohl der Instance-Hardness Detektor für einen besseren Log-Loss Wert sorgt, ist die Genauigkeit von S-Baseline mit 29,8% besser. Dies ist bereits in Abschnitt 4.2.1 aufgetreten und erklärt worden.

Dagegen ist eine direkte Korrelation zwischen vorhergesagten Klassen und Log-Loss Wert gegeben. Wenn der Log-Loss Wert niedrig ist, betrachtet der Random Forest viele Klassen, wohingegen bei einem hohen Log-Loss von 21 nur 9 Klassen betrachtet werden, da die anderen Klassen zu wenig gewichtet sind oder gelöscht werden. Wenn die Anzahl der vorhergesagten Klassen der beiden Szenarien verglichen werden, stellt sich heraus, dass in S-Gewichten wesentlich mehr Klassen betrachtet werden als in S-Löschen, da diese nach dem Löschen nicht mehr im Datensatz vorhanden sind.

In Abbildung 4.14 wird die über alle Datensätze separat für die Klassenungleichverteilung gemittelte Genauigkeit abgebildet. Die Genauigkeiten sind, wie erwartet, aufsteigend von Genauigkeit A@1 bis Genauigkeit A@10 angeordnet. Dies lässt sich dadurch erklären, dass es nicht möglich ist,

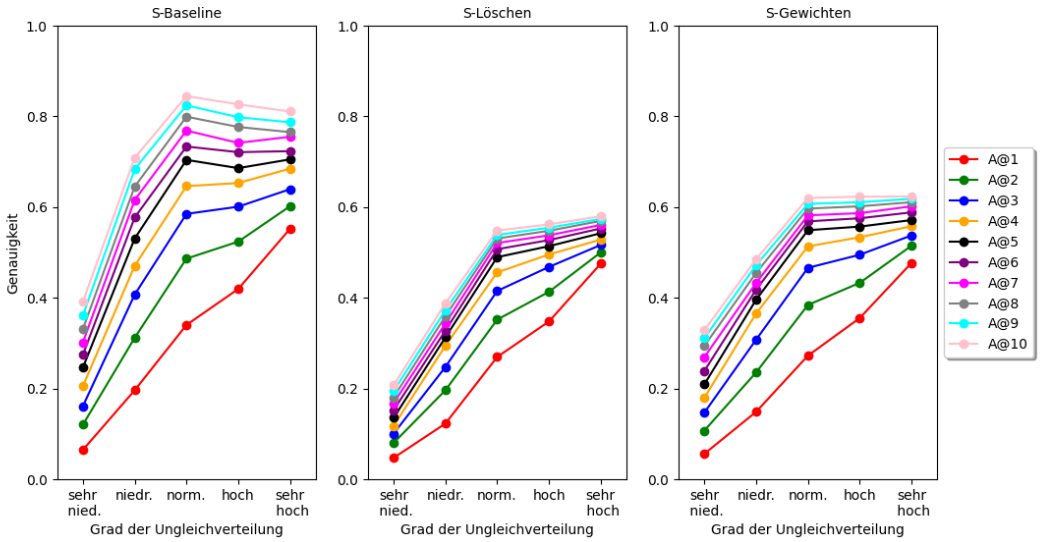

Abbildung 4.14: Untersuchung der Genauigkeiten in der zweiten Versuchsreihe

dass die Genauigkeit von A@1 nach A@10 sinkt, denn wenn es bereits an einer Stelle korrekt ist, wirkt sich das auch auf die folgende Stelle aus. Des Weiteren steigen alle Genauigkeiten wie in Abbildung 4.12 mit dem Grad der Ungleichverteilung.

Auffällig ist, dass die Genauigkeitswerte bis zu A@10 bei normaler und niedriger Ungleichverteilung schneller steigen als bei sehr niedriger, hoher und sehr hoher Ungleichverteilung. Die Genauigkeiten sind mit einem höheren k in S-Baseline bei normalem Grad an Ungleichverteilung im Vergleich zu den anderen Graden am höchsten. Dagegen sind die Genauigkeiten bei normalem Grad in S-Löschen und S-Gewichten nie höher als bei einem anderen Grad. Dies liegt daran, dass der Konfidenzwert der Datenpunkte der Minority Klassen geringer ist und diese in S-Baseline somit stärker gewichtet werden als in S-Löschen und S-Gewichten, Die Genauigkeit A@1 ist bei hohem Grad an Ungleichverteilung am höchsten, da vor allem viele Majority Klassen sicher erkannt werden. Ab A@5 wird die Genauigkeit A@K bei normalem Grad an Ungleichverteilung allerdings höher, da mehr Datenpunkte in den Minority Klassen liegen und dementsprechend mehr Muster der Minority Klassen erkannt werden können. Bei normalem Grad an Ungleichverteilung ist die Genauigkeit am höchsten, da genug Datenpunkte in den Majority Klassen und Minority Klassen sind, sodass bei hohem k die meisten Klassen erkannt werden können. **Zudem ist die Genauigkeit A@10 in S-Baseline mit bis zu 85% sehr viel höher als die in S-Löschen mit 55% und in S-Gewichten mit 63%.** Die Differenzen der Genauigkeiten A@K lassen sich ebenso mit der in S-Baseline stärkeren Gewichtung der Minority Klassen erklären. Allerdings ist die Differenz zwischen S-Gewichten und S-Löschen von bis zu 5% dadurch zu erklären, dass bei S-Löschen Datenpunkte der Minority Klassen gelöscht werden und deshalb nicht gewichtet werden können. Somit haben sie keinen Einfluss auf das Ergebnis. Wohingegen die Datenpunkte bei S-Gewichten nur weniger gewichtet werden und demnach trotzdem zum Erkennen von Mustern beitragen.

Die Abbildung 4.15 zeigt die durchschnittliche Anzahl gelöschter Datenpunkte in S-Löschen, während die rote Linie die vorhandene Anzahl an Label Noise widerspiegelt. Die linke Abbildung zeigt die gelöschten Label Noise, während auf der rechten Abbildung die Anzahl gelöschter richtiger

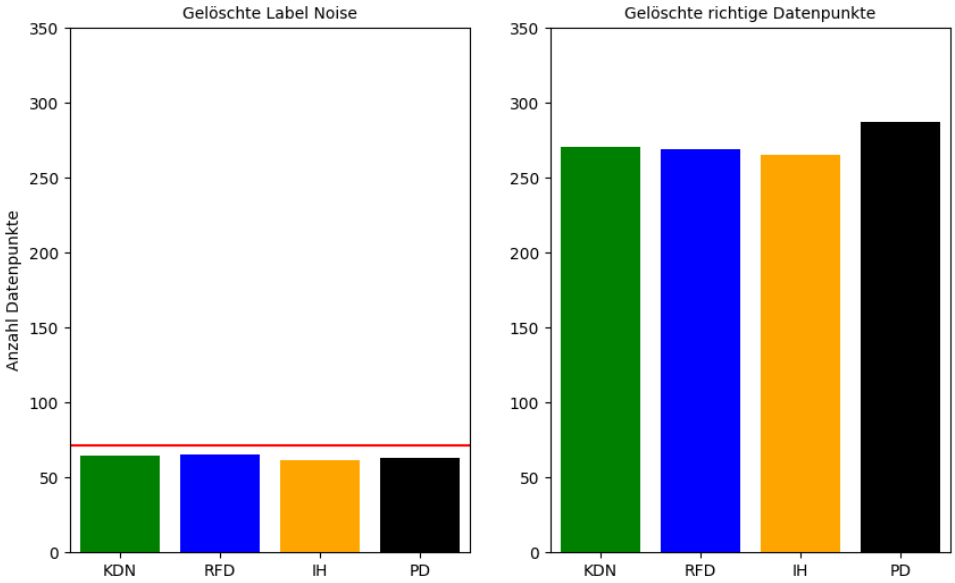

Abbildung 4.15: Untersuchung der durchschnittlich durch S-Löschen gelöschten Datenpunkte

Datenpunkte (kein Label Noise) dargestellt sind. Wie in der linken Abbildung zu sehen, werden bis zu 65 von 71 Label Noise erkannt, jedoch werden auch sehr viele (bis zu 287) weitere Datenpunkte, wie rechts zu sehen, gelöscht. Ein Datenpunkt wird nur dann gelöscht, wenn sein Konfidenzwert unter dem Grenzwert von 0,1 liegt. Dies trifft auf alle in der rechten Abbildung gelöschten Datenpunkte zu, die dadurch auch in S-Gewichten kaum gewichtet werden. Dadurch wird deutlich, dass es viele Datenpunkte mit geringem Konfidenzwert gibt, die vermutlich zu Minority Klassen gehören und somit kaum Einfluss auf das Trainieren in S-Gewichten oder S-Löschen haben. Es ist zu sehen, dass durch die Detektoren weniger Minority Klassen Einfluss auf das Ergebnis haben.

Precision & Recall In Tabelle 4.3 sind die Recall- und Precision-Werte dargestellt. Dabei fällt auf, dass der Recall-Wert 3%-Punkte schlechter ist als der Precision-Wert. Dies bedeutet, dass es viele Klassen gibt denen zu viele Datenpunkte zugeordnet werden, wie in Abschnitt 3.5.4 genauer beschrieben wird. Der Precision-Wert sollte somit größer sein als der Recall-Wert, da viele Minority Klassen vorliegen und diese schlechter erkannt werden.

In Abbildung 4.16 ist an einem Beispieldatensatz visualisiert, wieso der Precision-Wert niedriger ist. Auf der linken Seite ist der tatsächliche Datensatz dargestellt und auf der mittleren und rechten Abbildung sind zwei mögliche Klassifikationsergebnisse dargestellt. In der Abbildung „Hoher Recall“ ist der Normalfall dargestellt, der bereits erläutert wurde. In der Abbildung „Niedrige Precision“ ist der Fall dagegen dargestellt, der den Precision-Wert negativ beeinflusst, da die Minority Klassen nicht erkannt werden.

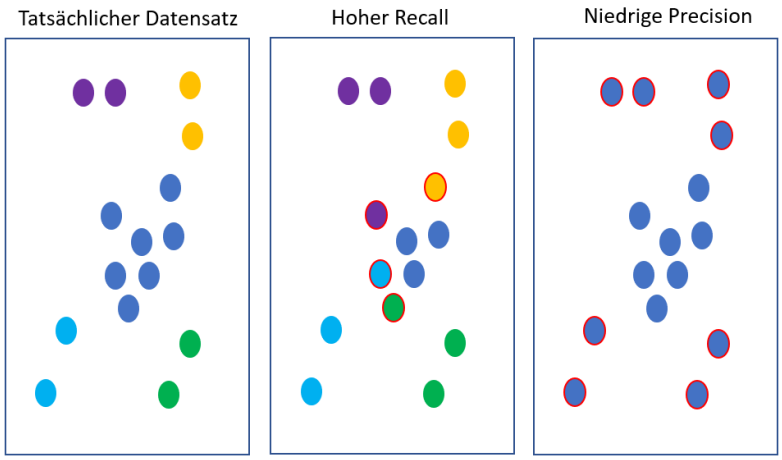

Abbildung 4.16: Visualisierung des niedrigen Precision-Werts durch einen Beispieldatensatz

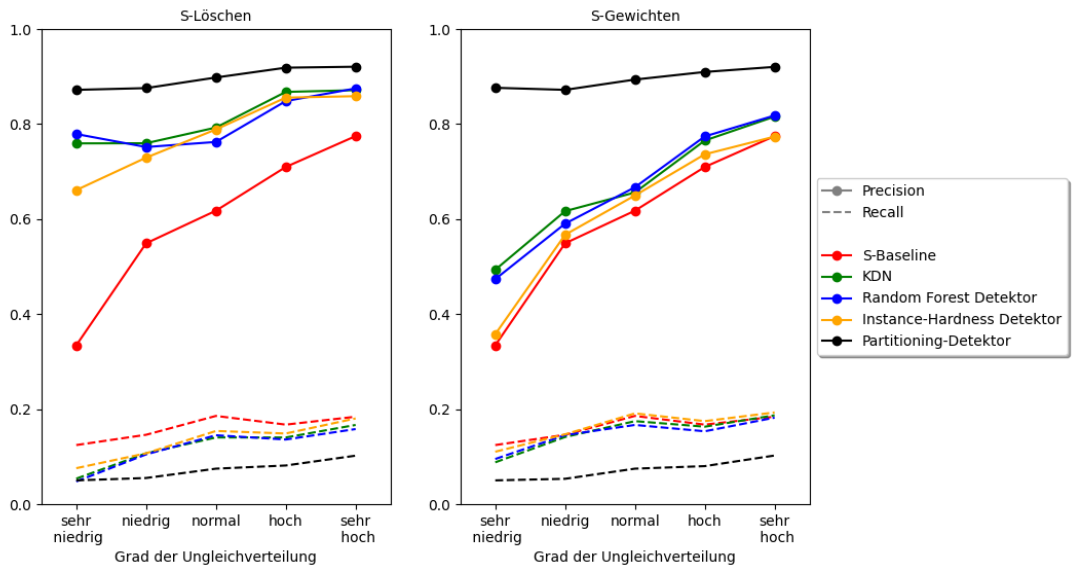

Abbildung 4.17: Untersuchung der Precision- & Recall-Werte in der zweiten Versuchsreihe

Wie in „Niedrige Precision“ in Abbildung 4.16 dargestellt ist, erkennt der Random Forest viele Minority Klassen gar nicht. Das heißt, dass diese einen Precision- und Recall-Wert von 0 haben, da keine Wahr-Positiven Datenpunkte der Minority Klassen existieren. Es existieren somit viele Minority Klassen mit einem Precision-Wert von 0 und eine Majority Klasse mit einem hohen Precision-Wert. Im Durchschnitt entsteht somit ein niedriger Precision-Wert für den Datensatz.

Anhand der Anzahl vorhergesagter Klassen in Tabelle 4.3, wird klar, dass genau dies auftritt und die Recall- und Precision-Werte verfälscht. Denn die Anzahl vorhergesagter Klassen ist mit zum Beispiel 39 Klassen in S-Baseline deutlich kleiner als die durchschnittlichen 76 vorhandenen Klassen. **Die Precision-Werte sind somit niedriger als die Recall-Werte, da durch Nichterkennung ganzer Minority Klassen häufig durch 0 geteilt werden muss.**

Die Erkenntnis, dass S-Gewichten und S-Löschen dafür sorgen, dass weniger Minority Klassen erkannt werden, zeigt sich auch in Abbildung 4.17. Dort sind links die Precision- und Recall-Werte von S-Löschen und rechts von S-Gewichten aufgezeigt. Dabei wurden die Werte des Precision- und Recall-Werts bereits mit dem einem Parameter angepasst. Das heißt, wenn eine Minority Klasse gar nicht erkannt wurde, müsste, wie bereits erwähnt, durch 0 geteilt werden. Anstatt des Standard-Werts 0, wird durch den Parameter eine 1 zurückgegeben. Wenn viele Minority Klassen nicht erkannt werden, wird in häufigen Fällen durch 0 geteilt, und es wird der Precision-Wert 1 zurückgegeben. Dadurch kommt generell ein höherer Precision-Wert zustande. Wenn es allerdings kaum Klassen gibt, die nicht erkannt werden, wird auch der Precision-Wert nicht erhöht.

Der Recall-Wert mit bis zu 20% ist nach dieser Korrektur deutliche geringer als der Precision-Wert mit bis zu 90%. Dies spiegelt die aktuelle Situation besser wider als die nicht korrigierten Werte aus Tabelle 4.3. Denn es gibt wegen der vielen Minority Klassen viele Klassen denen zu wenige Datenpunkte zugeordnet werden. Dies sorgt dafür, dass es viele Klassen mit Falsch-Negativen Datenpunkten gibt, wie es bereits in Abschnitt 3.5.4 beschrieben wurde. Durch die wenigen Majority Klassen gibt es auch wenige Klassen, die Falsch-Positive Datenpunkte haben, was zu einem niedrigen Recall-Wert führt. **Der Precision-Wert ist somit hoch, weil die Minority Klassen schlecht erkannt werden.**

Wird der Precision-Wert der Detektoren in Abbildung 4.17 verglichen, ist ersichtlich, dass durch das Löschen vieler Datenpunkte in S-Löschen der Precision-Wert der Detektoren um bis zu 40%-Punkte höher ist als der von S-Baseline. Dies liegt wieder daran, dass auch Datenpunkte der Minority Klasse gelöscht werden und diese deshalb schlechter erkannt werden. Im Vergleich zu S-Löschen, ist in S-Gewichten der Precision-Wert nur maximal 10%-Punkte höher als in S-Baseline. Dementsprechend wird auch durch das geringe Gewichten der Minority Klassen der Precision-Wert erhöht. Beide Szenarien mit Detektoren haben somit größere Schwierigkeiten als S-Baseline, die Minority Klassen zu erkennen. Außerdem ist in Abbildung 4.17 gezeigt, dass der Partitioning Detektor mit einem Precision-Wert von fast 90% kaum Minority Klassen erkennt. Anhand des steigenden Precision-Werts von S-Baseline von 33% auf 77%, ist zu erkennen, das bei sehr niedriger Ungleichverteilung wesentlich mehr Minority Klassen erkannt werden als bei sehr hoher Ungleichverteilung.

Zusammenfassung Die Besonderheit dieser Datensätze ist es, dass alle Herausforderungen vorhanden sind und die Datensätze mit dem ImbalanceDataGenerator generiert werden.

Bei Erhöhung der Label Noise Anzahl verringert sich die Genauigkeit nur um bis zu 3%-Punkte. Dahingegen wird, wenn der Grad der Ungleichverteilung erhöht wird, die Genauigkeit um bis zu 40%-Punkte erhöht. Zudem steht der Log-Loss-Wert tendenziell in Verbindung mit der Genauigkeit. Dabei ist aufgefallen, dass der Instance-Hardness Detektor immer die besten Log-Loss Werte hat, allerdings wird dessen Genauigkeit A@1 in den meisten Fällen von S-Baseline überholt. Die daraus resultierende vorhergesagte Anzahl an Klassen ist bei Instance-Hardness Detektor unter den Detektoren am höchsten, er betrachtet somit nach S-Baseline die meisten Minority Klassen. Zudem wurden die Genauigkeiten A@1 bis A@10 untersucht und es wurde festgestellt, dass S-Baseline sowohl bei niedrigem als auch bei hohem Parameter k in A@K am besten abschneidet. Dies liegt daran, dass S-Gewichten und S-Löschen die Minority Klassen weniger gewichten oder sogar löschen.

Im Anschluss wurde der Precision- und Recall-Wert untersucht. Dabei ist aufgefallen, dass das Nichterkennen einiger Minority Klassen den Precision-Wert deutlich verschlechtert. Der Precision-Wert war deswegen gering, obwohl er aufgrund der schlechten Erkennung der Minority Klassen eigentlich hoch sein sollte. Demnach wurden die Precision- und Recall-Werte mit einem Parameter angepasst, der bei Nichterkennung einer Minority Klasse den Wert 1 zurück gibt und somit den Precision-Wert nach oben korrigiert. Bei den nicht korrigierten Werten ist aufgefallen, dass der Recall-Wert höher als der Precision-Wert ist, da einige Minority Klassen gar nicht erkannt werden und somit durch 0 geteilt werden musste. Dagegen entsprechen die mit dem Parameter korrigierten Werte den Erwartungen. Der Precision-Wert ist höher als der Recall-Wert, da vielen Minority Klassen kaum vorhergesagt werden.

Unter den Detektoren schneiden die Detektoren aus S-Gewichten generell ein wenig besser ab. S-Baseline schneidet allerdings durchschnittlich am besten ab. Der Instance-Hardness Detektor schneidet zusammen mit dem KDN in S-Gewichten unter den Detektoren am besten ab. Dabei liegt die Genauigkeit durchschnittlich 1%-Punkt unter der Genauigkeit von S-Baseline.

Daraus folgt, dass der Random Forest in den meisten Fällen ohne Detektoren besser abschneidet.

4.4 Schlussfolgerung

Aus den Ergebnissen lässt sich schlussfolgern, dass vor allem, wenn nicht alle Herausforderungen im Datensatz vorkommen, Detektoren zu einer erhöhten Vorhersagegenauigkeit führen können. Dagegen führen Detektoren, wenn alle Herausforderungen gemeinsam auftreten, kaum zu einer besseren Vorhersagegenauigkeit.

Generell kann gesagt werden, dass der Instance-Hardness Detektor unter den Detektoren am besten abschneidet. Der Partitioning Detektor dagegen erkennt nur die Datenpunkte der Majority Klassen und schneidet deshalb am schlechtesten ab. Es gibt unterschiedliche Kombinationen der Datensatz-Parameter, S-Löschen oder S-Gewichten besser abschneiden als S-Baseline

Diese Kombinationen traten in der ersten Versuchsreihe auf, in der die Label Noise Anzahl, die Klassenanzahl, die Gewichtung der Majority Klasse und der Streuungsgrad variiert wird. In der ersten Versuchsreihe kam heraus, dass vor allem bei einem hohen Streuungsgrad und einem hohen Grad an Ungleichverteilung der Instance-Hardness Detektor und KDN besser abschneiden als S-Baseline.

In der zweiten Versuchsreihe wurden die Datensätze mit dem ImbalanceDataGenerator generiert. Dabei kam heraus, dass **Detektoren zuverlässig Label Noise erkennen, allerdings auch Minority Klassen weniger gewichten.** Bei vielen Minority Klassen im Datensatz werden diese selbst mit der A@10 Genauigkeit nicht erkannt, wenn zuvor Detektoren angewendet werden. Es kommt somit auf den Anwendungsfall an, ob Detektoren hilfreich sein können. Im gegebenen Szenario war S-Baseline in fast allen Datensätzen am besten. In einem Datensatz mit vielen Minority Klassen ist es dementsprechend nicht ratsam, mit Detektoren zu arbeiten. Werden die beiden Szenarien verglichen fällt auf, dass S-Gewichten in der zweiten Versuchsreihe meistens besser war. Dies liegt vermutlich daran, dass die Datenpunkte zwar weniger gewichtet, aber immerhin nicht gelöscht werden. Dabei

schneidet der Instance-Hardness Detektor auch in der zweiten Versuchsreihe am besten ab und ist demnach insgesamt der beste der untersuchten Detektoren. In der ersten Versuchsreihe dagegen war der Instance-Hardness Detektor in S-Löschen besser.

Wenn alle Herausforderungen in einem Datensatz auftreten, ist es nur im Falle von wenigen Minority Klassen ratsam, den Instance-Hardness Detektor anzuwenden. Wenn die „vielfältiges Produktportfolio“-Herausforderung nicht vorhanden ist, ist es **bei einem hohen Streuungsgrad und einem hohen Grad an Ungleichverteilung möglich, den Instance-Hardness Detektor und KDN anzuwenden.** Diese bringen eine Erhöhung der Genauigkeit in S-Löschen und S-Gewichten. In allen anderen Fällen, schneidet der Random Forest ohne Anwendung von Detektoren am besten ab.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurden unterschiedliche datengetriebene Verfahren zur Erkennung von Label Noise in komplexen Mehrklassenproblemen untersucht. Dazu wurden synthetisch generierte Daten verwendet, die an realen Daten des Qualitätsmanangements der industriellen Wertschöpfungskette angelehnt sind. Das Problem gesammelter Daten, besteht jedoch darin, dass diese diverse Herausforderungen mit sich bringen, wie zum Beispiel eine Klassenungleichverteilung, Label Noise, welche die Vorhersagegenauigkeit herkömmlicher Klassifikatoren verschlechtern. Wie aus der Arbeit von Hirsch et. al. [HRM19] hervorgeht, schneidet der Random Forest Klassifikator unter den gegebenen Herausforderungen am besten ab. Um dessen Genauigkeit noch weiter zu verbessern, wurde im Rahmen dieser Arbeit untersucht, wie Filtermethoden mit Label Noise in Kombination mit den genannten Herausforderungen umgehen. Dazu wurde speziell die Datenvorbereitung betrachtet.

Die Ausreißererkennung erschien im ersten Schritt vielversprechend, wurde allerdings aufgrund des unüberwachten Lernens der Ausreißererkennung ausgeschlossen. Im nächsten Schritt wurden Detektoren untersucht, auf denen anschließend eine Klassifikation ausgeführt wird. Da es eine Vielzahl unterschiedlicher Detektoren gibt, wurden in dieser Arbeit fünf davon fokussiert: KDN, Random Forest Detektor, MCS, Instance-Hardness Detektor und Partitioning-Detektor. Da durch die Herausforderungen fehlende Attributwerte auftreten, mussten diese mit Imputern aufgearbeitet werden.

Um die Auswirkungen der Detektoren zu messen wurden Szenarien entwickelt Das erste Szenario S-Baseline lässt den Random Forest den Datensatz klassifizieren, ohne vorher Detektoren anzuwenden. Im zweiten und dritten Szenario, S-Löschen und S-Gewichten, wurden dagegen Detektoren verwendet. Während in S-Löschen Datenpunkte, die von den Detektoren als wahrscheinliche Label Noise erkannt wurden, gelöscht werden, wird in S-Gewichten die Ausgabe der Detektoren dem Random Forest mitgegeben, damit dieser die Datenpunkte dementsprechend gewichten kann.

Anschließend wurden die Detektoren im Zusammenhang mit den Szenarien in zwei Versuchsreihen untersucht. In der ersten Versuchsreihe werden Datensätze mit unterschiedlichen Charakteristika erstellt, um diese in Isolation zu betrachten. In der zweiten Versuchsreihe dagegen werden die Datensätze mit dem ImbalanceDataGenerator erstellt, damit diese die zuvor genannten Herausforderungen beinhalten. Um die Performance der Detektoren in den jeweiligen Szenarien zu messen wurde die Genauigkeit des Random Forest evaluiert. Zudem wurden die Ausgaben der Detektoren mithilfe des Log-Loss Werts gemessen. In der zweiten Versuchsreihe wurden unterschiedlichen Imputer untersucht. Dabei ist zu beachten, dass die Datenpunkte mit fehlenden Attributwerten meistens nicht mit Datenpunkten ohne fehlende Attributwerte zusammenhängen. Zum Beispiel sollen vier Zylindermotoren, denen einige Attributwerte fehlen, nicht mit sechs Zylindermotoren in Verbindung gebracht werden. Demnach hat der Simple Imputer, der nur Nullen einfügt, am besten abgeschnitten.

Beim Evaluieren der Datencharakteristika stellte sich heraus, dass das Verstärken der Herausforderung Klassenungleichverteilung die Genauigkeit der Szenarien verbessert. Dies liegt daran, dass mehr Datenpunkte in den Majority Klassen liegen. Das Vergrößern der Anzahl an Label Noise und des Streuungsgrades, welcher den Grad der Klassenüberschneidung angibt, verschlechtert die Genauigkeit der Szenarien um bis zu 20%. Das Erhöhen der Klassenanzahl hat dabei kaum negative Auswirkungen auf die Genauigkeit der Szenarien. Die Genauigkeit A@K liefert bei höherem Parameter k auch einen höheren Wert und zeigt, dass S-Löschen und S-Gewichten dazu führen, dass kaum Minority Klassen unter den k höchsten Labeln liegen. Je besser die Ausgabe und der Log-Loss der Detektoren ist, desto besser werden die Minority Klassen vorhergesagt. Jedoch betrachtet S-Baseline die meisten Klassen und erkennt somit auch am meisten Minority Klassen, was zu der durchschnittlich höchsten Genauigkeit in der zweiten Versuchsreihe führt.

Beim Vergleichen der Detektoren hat sich ergeben, dass der Partitioning-Detektor kaum Minority Klassen erkennt und dementsprechend nicht gut geeignet ist. Der Instance-Hardness Detektor und KDN haben in der ersten Versuchsreihe besser als S-Baseline abgeschnitten. In der zweiten Versuchsreihe dagegen, waren sie knapp schlechter als S-Baseline. Der Instance-Hardness Detektor in S-Gewichten hat im Durchschnitt über beide Versuchsreihen sogar eine durchschnittlich höhere Genauigkeit A@1 als S-Baseline.

Bei der Betrachtung des Precision-Werts ist aufgefallen, dass das Nichterkennen einiger Minority Klassen den Precision-Wert deutlich verringert. Demnach wurden die Precision- und Recall-Werte mit einem Parameter angepasst, der den Precision-Wert nach oben korrigiert. Dadurch zeigt sich, dass der Precision-Wert durch die Korrektur deutlich größer war im Vergleich zum Recall-Wert.

Im Hinblick auf die Ziele der Arbeit etablierte sich die Erkenntnis, dass wenn alle Herausforderungen vorliegen die Detektoren gut in der Lage sind, Label Noise zu erkennen und herauszufiltern. Dabei werden allerdings häufig auch Minority Klassen als Label Noise erkannt, was zu einer Verschlechterung der Genauigkeit des Random Forest Klassifkators führt. Mithilfe der Detektoren können Label Noise zwar nicht korrigiert, jedoch kann Label Noise weniger gewichtet werden, sodass der Klassifikator weniger falsche Muster erlernt.

Ausblick

In zukünftigen Arbeiten kann versucht werden, die Ausgaben der Detektoren im Zusammenhang mit halb-überwachtem Lernen zu nutzen. Dazu könnten die Label der Datenpunkte gelöscht werden, die von den Detektoren als Label Noise erkannt wurden, Daraus entsteht ein Datensatz, der aus Datenpunkten mit und ohne Label besteht. Dies hat den Vorteil, dass die Datenpunkte nicht komplett entfernt werden.

Ein weiterer Anknüpfungspunkt wäre das Untersuchen der fehlenden Attributwerte im Zusammenhang mit den Detektoren. Bei den fehlenden Attributwerten handelt es sich zum Beispiel um fehlende Sensordaten, wie sie bei vier im Gegensatz zu sechs Zylindermotoren auftreten. Dabei ist es mit spezifischem Domänenwissen möglich verschiedene Zusammenhänge der Klassen auszuschließen. Zum Beispiel sollen vier nicht zu sechs Zylindermotoren zugeordnet werden. Das Problem dabei ist, dass der Klassifikator die zusätzlichen Informationen, also das Domänenwissen nicht nutzen kann. Das Ausschließen mancher Kombinationen, wie vier und ‚sechs Zylinder, ist somit nur für den Menschen möglich und verbessert demnach die Genauigkeit der Klassifikatoren

nicht, da diese datengetriebenen Verfahren diese Informationen nicht nutzen können. In zukünftigen Arbeiten könnte nach einer Möglichkeit gesucht werden, wie die Korrelationen zwischen den Datenpunkten den Detektoren hilft, damit diese Label Noise effizienter erkennen können. Dazu könnten die Datensätze zum Beispiel nach ihren fehlenden Attributen aufgeteilt werden. Eine weitere Möglichkeit ist das Ersetzen der fehlenden Attributwerte mit einem Wert, der nicht in Verbindung mit bereits existierenden Attributwerten gebracht werden kann.

Des Weiteren kann in zukünftigen Arbeiten untersucht werden, wie eine Partitionierung der Daten im Zusammenhang mit dem Anwenden von Detektoren die Genauigkeit der Klassifikatoren beeinflusst. Hierbei wird die Partitionierung in einem separaten Schritt durchgeführt und nicht beim Detektieren, wie es der Partitioning-Detektor macht. Eine Möglichkeit wäre es, die Daten zuerst zu partitionieren und im Anschluss die Detektoren auf die einzelnen Partitionen anzuwenden. Dies hätte einige Vorteile, wie eine geringere Komplexität, weniger Überschneidungen und eine geringere Klassenanzahl.

Literaturverzeichnis

[Bis06] C. M. Bishop. „Pattern recognition“. In: *Machine learning* 128.9 (2006) (zitiert auf S. 30).

[BKNS00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander. „LOF: identifying density-based local outliers“. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, S. 93–104 (zitiert auf S. 20).

[Bre01] L. Breiman. „Random forests“. In: *Machine learning* 45.1 (2001), S. 5–32 (zitiert auf S. 15).

[CBK09] V. Chandola, A. Banerjee, V. Kumar. „Anomaly detection: A survey“. In: *ACM computing surveys (CSUR)* 41.3 (2009), S. 1–58 (zitiert auf S. 19).

[FV13] B. Frénay, M. Verleysen. „Classification in the presence of label noise: a survey“. In: *IEEE transactions on neural networks and learning systems* 25.5 (2013), S. 845–869 (zitiert auf S. 15, 20, 23).

[Gas72] J. L. Gastwirth. „The estimation of the Lorenz curve and Gini index“. In: *The review of economics and statistics* (1972), S. 306–316 (zitiert auf S. 31).

[GCL15] L. P. Garcia, A. C. de Carvalho, A. C. Lorena. „Effect of label noise in the complexity of classification problems“. In: *Neurocomputing* 160 (2015), S. 108–119 (zitiert auf S. 12).

[Guy03] I. Guyon. „Design of experiments for the NIPS 2003 variable selection benchmark“. In: 2003 (zitiert auf S. 24).

[HRM19] V. Hirsch, P. Reimann, B. Mitschang. „Data-driven fault diagnosis in end-of-line testing of complex products“. In: *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2019, S. 492–503 (zitiert auf S. 3, 11, 12, 15, 16, 19, 20, 23, 38, 61).

[HRM20] V. Hirsch, P. Reimann, B. Mitschang. „Exploiting domain knowledge to address multi-class imbalance and a heterogeneous feature space in classification tasks for manufacturing data“. In: *Proceedings of the VLDB Endowment* 13.12 (2020), S. 3258–3271 (zitiert auf S. 17).

[HYS+17] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing. „Learning from class-imbalanced data: Review of methods and applications“. In: *Expert Systems with Applications* 73 (2017), S. 220–239 (zitiert auf S. 16).

[KGJH16] B. Krawczyk, M. Galar, Ł. Jeleń, F. Herrera. „Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy“. In: *Applied Soft Computing* 38 (2016), S. 714–726 (zitiert auf S. 16).

[KR07] T. M. Khoshgoftaar, P. Rebours. „Improving software quality prediction by noise filtering techniques“. In: *Journal of Computer Science and Technology* 22.3 (2007), S. 387–396 (zitiert auf S. 27).

[KWK20] M. Koziarski, M. Woźniak, B. Krawczyk. „Combined cleaning and resampling algorithm for multi-class imbalanced data with label noise“. In: *Knowledge-Based Systems* 204 (2020), S. 106223 (zitiert auf S. 12).

[LTZ08] F. T. Liu, K. M. Ting, Z.-H. Zhou. „Isolation forest“. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, S. 413–422 (zitiert auf S. 20).

[MCS21] N. Myrtakis, V. Christophides, E. Simon. „A Comparative Evaluation of Anomaly Explanation Algorithms“. In: *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. Hrsg. von Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, F. Guerra. OpenProceedings.org, 2021, S. 97–108. DOI: 10.5441/002/edbt.2021.10. URL: https://doi.org/10.5441/002/edbt.2021.10 (zitiert auf S. 20).

[RD99] P. J. Rousseeuw, K. V. Driessen. „A fast algorithm for the minimum covariance determinant estimator“. In: *Technometrics* 41.3 (1999), S. 212–223 (zitiert auf S. 20).

[SB12] D. J. Stekhoven, P. Bühlmann. „MissForest—non-parametric missing value imputation for mixed-type data“. In: *Bioinformatics* 28.1 (2012), S. 112–118 (zitiert auf S. 26).

[SMG14] M. R. Smith, T. Martinez, C. Giraud-Carrier. „An instance level analysis of data complexity“. In: *Machine learning* 95.2 (2014), S. 225–256 (zitiert auf S. 27).

[SMS18] M. Sabzevari, G. Martínez-Muñoz, A. Suárez. „A two-stage ensemble method for the detection of class-label noise“. In: *Neurocomputing* 275 (2018), S. 2374–2383. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.11.012. URL: https://www.sciencedirect.com/science/article/pii/S0925231217317265 (zitiert auf S. 28).

[TCS+01] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, R. B. Altman. „Missing value estimation methods for DNA microarrays“. In: *Bioinformatics* 17.6 (2001), S. 520–525 (zitiert auf S. 26).

[WCO+18] F. N. Walmsley, G. D. Cavalcanti, D. V. Oliveira, R. M. Cruz, R. Sabourin. „An ensemble generation method based on instance hardness“. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, S. 1–8 (zitiert auf S. 27).

[YL09] S.-J. Yen, Y.-S. Lee. „Cluster-based under-sampling approaches for imbalanced data distributions“. In: *Expert Systems with Applications* 36.3 (2009), S. 5718–5727 (zitiert auf S. 12).

[ZCTP19] Z. Zhao, L. Chu, D. Tao, J. Pei. „Classification with label noise: a Markov chain sampling framework“. In: *Data Mining and Knowledge Discovery* 33.5 (2019), S. 1468–1504 (zitiert auf S. 28).

Alle URLs wurden zuletzt am 30. 09. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, 01.10.2021

Ort, Datum, Unterschrift