Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

AutoML für Clustering zur Datenpartitionierung für einen Multi-Klassen-Anwendungsfall

Michael Schneider

Studiengang:	B.Sc. Data Science
Prüfer/in:	Prof. Dr. Holger Schwarz
Betreuer/in:	M.Sc. Dennis Tschechlov
Beginn am:	10. Mai 2023
Beendet am:	10. November 2023

Kurzfassung

In der heutigen datengetriebenen Welt ist die effiziente und effektive Verarbeitung und Analyse von vorhandenen Datenmengen von essenzieller Bedeutung. Bisherige Arbeiten von Hirsch et al. [HRM19; HRM20] zeigen, wie man analytischen Herausforderungen eines Datensatzes aus Produktionsdaten, nämlich eine kleine Datensatzgröße, Klassenungleichverteilung und ein heterogenes Produktportfolio (C1-C3) angehen kann. Mit Hilfe von Ensemble-Learning-Verfahren kann die Klassifikation verbessert werden [HRM19]. Es wird aufgezeigt, dass verfügbares Domänenwissen in Form einer Produkthierarchie dazu genutzt werden kann, die Daten zu partitionieren, sodass die Vorhersage einer anschließenden Klassifikation signifikant verbessert werden kann [HRM20].

In Der Arbeit von Braun [Bra21] wird untersucht, ob die Vorhersage auf einem synthetisch generierten Datensatz mit denselben analytischen Herausforderungen, auch ohne vorhandenes Domänenwissen, verbessert werden kann. Dabei sollen die Daten durch eine rein datengetriebene Partitionierung mittels Clustering aufgeteilt werden, um die nachfolgende Klassifikation zu vereinfachen und homogenere Gruppen mit ähnlichen Eigenschaften zu erzeugen. Dabei werden vergleichbare oder sogar leicht verbesserte Ergebnisse, im Vergleich zur Klassifikation ohne Datenpartitionierung, erzielt. Jedoch zeigt sich, dass es Schwierigkeiten bei der Parameterwahl der Clustering-Algorithmen und die Bewertung der resultierenden Cluster gibt. Daher wird AutoML4Clust [TFS+21] eingesetzt, um die Parameterauswahl mittels intrinsischer Clustering-Metriken zu optimieren.

In dieser Arbeit wird untersucht, ob die Erkenntnisse aus Brauns Vorarbeit [Bra21] auf realen Produktionsdaten eines Laserschneidemaschinenherstellers übertragen werden können. Hierbei sollen die Produktionsdaten dazu genutzt werden, den voll-automatisierten Prozess der Laserschneidemaschinen zu optimieren, indem Fehlzustände vor dem eigentlichen Prozess erkannt werden können. Die Produktionsdaten weisen dieselben analytischen Herausforderungen C2 und C3 aus [HRM19] auf, jedoch stellt im Vergleich zu C1, durch eigener Analyse die große Datenmenge, bestehend aus 6,5 Millionen Datenpunkten und 155 Features, eine neue Herausforderung dar. Die Umsetzung einer datengetriebenen Partitionierung eines Datensatzes dieser Größenordnung erfordert zusätzliche Methoden wie Undersampling und Feature-Selection, um der Laufzeit- und Speicherkomplexität der verwendeten Algorithmen entgegenzuwirken. Außerdem liegen keine vorher bekannten Cluster-Labels vor, wodurch eine Optimierung, basierend auf Metriken mit diesen bekannten Cluster-Labels, unmöglich ist. Das Hauptziel dieser Arbeit besteht darin, ein automatisch optimiertes Vorhersagemodell speziell für diesen Anwendungsfall zu erstellen, welches die Parameter der Clustering und Klassifikations-Algorithmen kombiniert optimiert. Dazu wird das Clustering als Vorverarbeitung der Daten genutzt, um homogenere Gruppen zu bilden.

Für die Analyse und um die Herausforderungen der Daten zu adressieren, wird eine vollständig modulare Pipeline in Python erstellt. Jeder Schritt, einschließlich Datenaufbereitung, Undersampling, Clustering, Klassifikation, automatisiertes maschinelles lernen (AutoML), Modell-Retraining und Evaluation, kann erweitert oder ausgetauscht werden. Zusätzlich werden Python-Module verwendet, um das Tracking und die Durchführung von Experimenten zu vereinfachen und die Algorithmen mit Hilfe von GPUs zu beschleunigen.

Bei der Evaluation zeigt sich eine geringfügig schlechtere Vorhersagegenauigkeit bei der Optimierung des kombinierten Clusterings und Klassifikation im Vergleich zum optimierten Baseline Random-Forest-Klassifikator ohne Datenpartitionierung. Zudem resultiert dies, trotz erheblichem zeitlichem Mehraufwand, der durch die benötigten extra Schritte für die Optimierung verursacht wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

In einer zunehmend digitalisierten und vernetzten Welt ist die Fähigkeit, Daten effizient und effektiv zu analysieren und daraus Wissen zu extrahieren, von entscheidender Bedeutung. Die Analyse von Daten ermöglicht es, Muster und Zusammenhänge zu erkennen, die für die Optimierung von Prozessen, die Entscheidungsfindung und die Vorhersage von Ereignissen genutzt werden können. Allerdings sind die in der Praxis anfallenden Daten oft komplex und stellen eine Reihe von Herausforderungen für die Datenanalyse dar. Diese Herausforderungen können eine sehr kleine oder sehr große Datensatzgröße, eine hohe Dimensionalität der Daten, eine ungleiche Verteilung der Klassen, Label Noise und viele andere sein.

In der Arbeit von Hirsch et al. [HRM20] wird der Nutzen der Partitionierung von Daten und anschließender Klassifikation im Bereich des Qualitätsmanagements gezeigt. Durch die Nutzung von Domänenwissen zur Partitionierung der Daten kann die Genauigkeit der Klassifikation trotz der Komplexität der Daten verbessert werden. Eine ähnliche Studie wird von Braun [Bra21] durchgeführt, der eine datengetriebene Partitionierung mittels Clustering vornimmt. Obwohl die Ergebnisse dieser Studie auf synthetisch generierten Datensätzen basieren, zeigen sie, dass eine datengetriebene Partitionierung zu vergleichbaren oder sogar leicht verbesserten Ergebnissen führen kann.

In der vorliegenden Arbeit wird ein spezifischer Anwendungsfall untersucht, nämlich der eines automatische Laserschneideprozesses. Dieser Prozess umfasst das Beladen von Blechteilen auf die Arbeitsfläche, das Ausschneiden verschiedener Teile aus dem Blech, die Entnahme der Teile aus dem Blech und das Abladen des übrigen Blechteils. Während dieser Prozess insgesamt zu einer effizienteren Produktion führt, können Schwierigkeiten beim automatischen Schneiden oder der automatischen Entnahme von Teilen auftreten, die den Maschinenablauf verzögern oder sogar zum Stillstand bringen können. Dabei können die zu schneidenden Produktionsteile verschieden komplexe Geometrien annehmen.

Der Datensatz, der für diese Arbeit verwendet wird, stellt eine Reihe von Herausforderungen dar. Er enthält 6,5 Millionen Datenpunkte und 155 Features, was sowohl hinsichtlich der Größe als auch der Komplexität des Datensatzes eine Herausforderung darstellt. Nach eigener Analyse der Daten, weist dieser eine ungleiche Verteilung der Klassen auf und besteht aus heterogenen Gruppen, was die Analyse und Modellierung zusätzlich erschwert. Diese beiden Herausforderungen werden, unter Anderem, in einer Vorarbeit von Hirsch et al. [HRM19], als analytische Herausforderung festgestellt und die Vorhersagegenauigkeit mittels geeigneter Klassifikatoren-Wahl verbessert. Jedoch bringt der Datensatz aus diesem Anwendungsfall auch seine eigenen Herausforderungen mit sich, wie die Größe des Datensatzes und die hohe Dimensionalität der Daten.

Das Hauptziel dieser Arbeit ist es, ein automatisch optimiertes Vorhersagemodell, speziell für diesen Anwendungsfall zu kreieren. Hierfür wird das Clustering als Vorverabeitung der Daten genutzt, um homogenere Gruppen zu erstellen. Dieser Ansatz soll die Notwendigkeit von Clustering-Metriken obsolet machen und es ermöglichen, die Partitionierung und Klassifikation der Daten gleichzeitig zu

optimieren. Für das automatische Optimieren von Hyperparameter oder die Wahl der Algorithmen, existieren bereits Tools [FKE+15; TFS+21]. Jedoch sind diese in ihrem Einsatz zu restriktiv für die Umsetzung in dieser Arbeit, weswegen auf ein automatisch maschinellem Lernen (AutoML) Framwork gesetzt wird, welche komplexe Optimierungsaufgaben ermöglicht.

In dieser Arbeit wird zunächst eine detaillierte Analyse des verwendeten Datensatzes und der damit verbundenen Herausforderungen durchgeführt. Um den genannten Herausforderungen des Datensatzes entgegenzuwirken und eine Implementierung der Umsetzung gewährleisten zu können, werden Methoden aus folgenden Bereichen angewandt:

- Datenaufbereitung: Um die Dimensionalität der Daten zu reduzieren
- Undersampling: Um die Anzahl der Datenpunkte zu reduzieren
- Clustering: Um eine datengetriebene Partitionierung zu erstellen
- Klassifikation: Die Vorhersage der tatsächlichen Klasse
- AutoML: Für das automatisierte und kombinierte Optimieren von Clustering und Klassifikation
- Evaluation: Direkter Vergleich der Umsetzung auf Testdaten, gegen eine alleinige Klassifikation

Für die tatsächliche Implementation wird eine vollständig modulare Pipeline in Python erstellt, in der jeder Schritt des Prozesses, erweitern werden kann. Diese erlaubt eine beschleunigte Ausführung der Algorithmen, die durch GPU-Unterstützung erheblich beschleunigt wird.

Die Arbeit ist wie folgt strukturiert: Zunächst werden in Kapitel 2 die Grundlagen und Konzepte erläutert. Anschließend wird in Kapitel 3 eine detaillierte Beschreibung aller Schritte und verwendeten Methoden gegeben. Kapitel 4 zeigt die Details der Implementierung in Python auf. Die Ergebnisse der Evaluation werden im vorletzten Kapitel - 5 der Arbeit präsentiert und analysiert. Anschließend werden sie in Kapitel 6 zusammengefasst und es wird ein Ausblick auf mögliche Erweiterungen gegeben.

2 Grundlagen und verwandte Arbeiten

Für die folgende Ausarbeitung wird in diesem Kapitel auf alle benötigten Grundlagen eingegangen, die sowohl für das Verständnis der einzelnen Komponenten in der Umsetzung notwendig sind, als auch verwandte Arbeiten, die als Basis für die eingeführten Konzepte dienen. Der Schwerpunkt dieser Arbeit liegt hierbei auf der Bewältigung der Herausforderungen, die beim Umgang mit dem Anwendungsfall-Datensatz auftreten. Dabei werden die folgenden vier Hauptthemen behandelt: Clustering, Klassifikation, Undersampling und AutoML.

2.1 Clustering

Clustering ist ein zentrales Konzept in der Datenanalyse und spielt eine wichtige Rolle in wissenschaftlichen und industriellen Bereichen. Es ermöglicht die Identifikation von Strukturen und Mustern in großen Datenmengen und kann zur Vorverarbeitung von Daten für andere maschinelle Lernverfahren oder zur Erkenntnisgewinnung in verschiedenen Anwendungsbereichen eingesetzt werden. Die Hauptaufgabe des Clustering besteht darin, Datenpunkte in verschiedene Gruppen oder Cluster zu unterteilen, so dass Datenpunkte innerhalb desselben Clusters eine höhere Ähnlichkeit aufweisen als Datenpunkte aus unterschiedlichen Clustern. Die Ähnlichkeit wird in der Regel durch eine Distanzmetrik, wie z.B. der euklidischen Distanz, gemessen. Die Anzahl der Cluster kann, je nach Algorithmus, entweder vorab festgelegt oder durch das Verfahren selbst bestimmt werden. Im Gegensatz zur Klassifikation, die zur Gruppe des überwachten Lernens gehört, ist das Clustering ein unüberwachtes Lernverfahren. Es basiert auf der Gruppierung von Datenpunkten auf der Grundlage ihrer Ähnlichkeit, ohne dass eine vordefinierte Zielvariable oder Klasse vorliegt. Es existieren verschiedene Arten von Clustering-Methoden, darunter Zentrum-, Dichte- und Hierarchisch-basiertes Clustering, die sich in der Art und Weise unterscheiden, wie Ähnlichkeit definiert und Cluster gebildet werden. Zur Bewertung des Clustering-Algorithmus können entweder extrinsische Metriken verwendet werden, die wahre Cluster-Labels verwenden, oder intrinsische Metriken, die die Kompaktheit einzelner Cluster und die Trennung von anderen Clustern bewerten. In dieser Arbeit wird Clustering verwendet, um die Daten zu partitionieren und homogenere Gruppen mit ähnlichen Eigenschaften zu erstellen. Dies ist ein entscheidender Schritt, um die Komplexität der nachfolgenden Klassifikation zu reduzieren und die Vorhersagegenauigkeit dieser zu verbessern.

2.2 Klassifikation

Die Klassifikation dient zur Vorhersage der Klassenzugehörigkeit eines Datenpunktes. Diese kann in binäre und Multi-Klassen Klassifikation unterteilt werden. Bei der binären Klassifikation existieren lediglich zwei Ausgabeklassen. Ein typisches Beispiel hierfür ist die E-Mail-Spam-Erkennung,

bei der E-Mails entweder als „Spam“ oder „kein Spam“ klassifiziert werden. Im Gegensatz dazu existieren bei der Multi-Klassen Klassifikation mehr als zwei Ausgabeklassen. Ein Beispiel hierfür ist die Erkennung von Handschriften, bei der jeder Buchstabe eine eigene Klasse repräsentiert.

Die Klassifikation ist ein überwachtes Lernverfahren, dies bedeutet, dass ein Modell auf Basis von Trainingsdaten konstruiert wird, die sowohl die Eingabevariablen (Features) als auch die Ausgabevariable (Klasse) beinhalten. Das Modell erlernt die Beziehungen zwischen Eingabe- und Ausgabevariablen und nutzt dieses Wissen, um die Klassen neuer, unbekannter Daten vorherzusagen. Die Klassifikation durchläuft typischerweise zwei Phasen: die Trainingsphase und die Vorhersagephase. Während der Trainingsphase wird das Modell auf der Grundlage der gelabelten Trainingsdaten trainiert, indem es die Beziehungen zwischen Feature und Klassenlabel lernt. In der Vorhersagephase wird das trainierte Modell dann auf neue, ungesehene Daten angewendet, um die Ausgabeklassen dieser Daten vorherzusagen.

Eine bedeutende Methode in der Klassifikation sind die sogenannten Ensemble-Methoden. Diese Techniken kombinieren die Vorhersagen mehrerer Basis-Modelle, um eine finale Vorhersage zu generieren. Die Prämisse dahinter ist, dass ein Ensemble von Modellen in der Regel eine bessere Leistung erbringt als ein einzelnes Modell, da es in der Lage ist, kleinere Unterschiede in der Variation von unterschiedlichen Dateneigenschaften abzudecken. Bekannte Beispiele für Ensemble-Methoden sind Bagging und Boosting. [Mehr auf Bagging und Boosting eingehen]

2.2.1 Klassenungleichverteilung

Eine vorhandene Klassenungleichverteilung ist eine verbreitete Herausforderung in vielen Klassifizierungsproblemen und tritt auf, wenn die Anzahl der Datenpunkte in den verschiedenen Klassen eines Datensatzes stark variiert. Dies kann dazu führen, dass Klassifikationsmodelle eine Tendenz gegenüber der überrepräsentierten Klasse entwickeln und nur für dies gute Vorhersagewerte liefert. Dabei kann die Vorhersage für Minderheitsklassen schlecht ausfallen,

In vielen realen Anwendungsfällen sind die Klassen in den Datensätzen ungleich verteilt. Dies kann beispielsweise in medizinischen Diagnosedaten der Fall sein, in denen die Mehrheit der Patienten nicht an einer bestimmten Krankheit leidet (negative Klasse), während nur eine kleine Minderheit der Patienten positiv ist (positive Klasse). In solchen Fällen kann ein Klassifikationsmodell, das auf einem solchen ungleichmäßig verteilten Datensatz trainiert wurde, dazu neigen, die überrepräsentierte Klasse zu bevorzugen, was zu einer hohen Fehlerrate bei der Klassifikation der unterrepräsentierten Klasse führt.

Die ungleiche Verteilung der Klassen in einem Datensatz kann auch dazu führen, dass die Klassifikationsmodelle überangepasst werden (overfitting), d.h., diese lernen die spezifischen Muster der überrepräsentierten Klasse zu genau und können dann nicht gut auf neue, unbekannte Daten generalisieren. Dies kann insbesondere dann problematisch sein, wenn die überrepräsentierte Klasse in den Trainingsdaten Muster aufweist, die in den Testdaten nicht vorhanden sind. Somit kann es bei der Anwendung auf neuen Daten zu einer schlechter Vorhersageleistung kommen.

Es gibt verschiedene Methoden, um das Problem der Klassenungleichverteilung zu bewältigen, darunter das Over- und das Undersampling. Beide Methoden zielen darauf ab, die Klassenverteilung in den Trainingsdaten auszugleichen, um eine bessere und ausgewogenere Vorhersage der Klassifikation zu erzielen. Im Folgenden werden diese beiden Methoden genauer erläutert.

Undersampling ist eine weit verbreitete Methode zur Bewältigung von Klassifizierungsproblemen, die durch eine ungleichmäßige Verteilung von Klassen gekennzeichnet sind. Besonders in Situationen, in denen der Datensatz groß ist und eine ungleiche Verteilung von Klassen aufweist, kann das Undersampling effektiv angewendet werden. Zum einen um die Datensatzgröße zu verringern, welches die Trainigsdauer des Modells reduziert, und zum anderen, um die Klassenungleichverteilung zu verbessern.

Das Prinzip des Undersampling liegt darin, die Anzahl der Datenpunkte in den überrepräsentierten Klassen zu reduzieren. Dies kann durch verschiedene Verfahren erreicht werden. Eine gängige Methode ist das zufällige Undersampling, bei dem Beispiele aus der überrepräsentierten Klasse zufällig entfernt werden. Dieser Ansatz ist jedoch mit Risiken verbunden, da durch das willkürliche Entfernen von Beispielen potenziell wichtige Informationen verloren gehen können, die für die Klassifizierung von Bedeutung sind.

Eine alternative Methode ist das gezielte Undersampling, bei dem Beispiele auf der Grundlage spezifischer Kriterien entfernt werden. So könnten beispielsweise solche Beispiele ausgewählt werden, die nahe an der Entscheidungsgrenze liegen oder als „Rauschen“ oder „Ausreißer“ identifiziert werden. Diese Methode erfordert jedoch eine sorgfältige Auswahl der zu entfernenden Beispiele, um sicherzustellen, dass keine relevanten Informationen verloren gehen.

Das Hauptziel des Undersampling ist es, die Klassifikationsleistung zu verbessern, indem ein ausgewogeneres Verhältnis zwischen den Klassen hergestellt wird. Es ist jedoch wichtig zu beachten, dass das Undersampling auch seine Nachteile hat. Ein wesentliches Risiko besteht darin, dass durch das Entfernen von Beispielen wichtige Informationen verloren gehen können, die für die Klassifizierung relevant sind. Daher besteht die Gefahr, dass das Modell bestimmte Muster übersehen kann, die in den entfernten Daten vorhanden waren. Darüber hinaus kann das Undersampling auch dazu führen, dass die Klassifikationsergebnisse verzerrt werden, da die ursprüngliche Verteilung der Klassen im Datensatz nicht mehr vorhanden ist. Daher muss bei der Anwendung von Undersampling geprüft werden, welche Methode die besten Ergebnisse, an dem konkreten Datensatz, erzielt.

Insgesamt ist das Undersampling eine nützliche Methode zur Behandlung von Klassifizierungsproblemen mit ungleichmäßiger Klassenverteilung, insbesondere wenn der Datensatz groß ist. Es erfordert jedoch eine sorgfältige Wahl der Algorithmen, um sicherzustellen, dass keine oder nur wenige Informationen verloren gehen.

Oversampling ist eine weitere Technik zur Behandlung von Klassifizierungsproblemen mit ungleichmäßiger Klassenverteilung. Im Gegensatz zum Undersampling, bei dem die Anzahl der Beispiele in den überrepräsentierten Klassen reduziert wird, zielt das Oversampling darauf ab, die Anzahl der Beispiele in den unterrepräsentierten Klassen zu erhöhen. Dies geschieht typischerweise durch das Hinzufügen von Kopien bestehender Beispiele oder durch das Erzeugen neuer, synthetischer Beispiele. Ziel des Oversamplings ist es, die Vorhersage der darauffolgenden Klassifikation zu verbessern, indem der Bias zu den Mehrheitsklassen, durch das Vergrößern der Minderheitsklassen, verringert wird.

Eine häufig verwendete Methode für Oversampling ist die Synthetic Minority Over-Sampling Technique (SMOTE) [CBHK02]. Bei dieser Methode werden neue Datenpunkte nicht einfach durch Kopieren von Bestehenden erzeugt, sondern durch eine Kombination der Features von einem oder mehreren ähnlichen Datenpunkten. Es ist jedoch wichtig zu beachten, dass Oversampling auch seine Nachteile hat. Einer der Hauptnachteile ist, dass es das Risiko des Overfittings erhöhen kann. Da die erzeugten Datenpunkte auf den Eigenschaften der bestehenden Datenpunkte basieren, kann

das Modell dazu neigen, sehr spezifische Muster zu lernen, die nur in den Trainingsdaten vorhanden sind und nicht unbedingt auf neue, unbekannte Daten verallgemeinert werden können. Da die Größe des vorhandenen Datensatzes bereits eine Herausforderung darstellt, wird Oversampling in dieser Arbeit nicht untersucht und angewendet,

2.2.2 Heterogene Gruppen

Heterogene Gruppen sind in vielen Branchen und Geschäftsbereichen üblich und stellen eine besondere Herausforderung für die Datenanalyse und die Modellierung dar. Heterogene Produktgruppen sind durch eine Vielzahl von Produkten gekennzeichnet, die sich in mehreren Features oder Eigenschaften deutlich voneinander unterscheiden. Diese Unterschiede können sich auf eine Vielzahl von Faktoren beziehen, wie zum Beispiel einer breiten Produktpalette oder Unterschiede in der Komplexität von Produktionsteilen.

Die Heterogenität der Produktgruppen kann zu vielen einzelnen Mustern in den Daten führen, was die Modellierung und Vorhersage erschwert. Beispielsweise können Klassifikationsmodelle, die auf heterogenen Produktgruppendaten trainiert werden, Schwierigkeiten haben, genaue und zuverlässige Vorhersagen zu treffen. Die Vielfalt der Produkte kann dazu führen, dass die Modelle überangepasst werden und nicht gut auf neue, unbekannte Produkte generalisieren können (Overfitting). Darüber hinaus kann die Heterogenität der Produktgruppen auch zu Problemen bei der Interpretation der Modellergebnisse führen. Da die Produkte so unterschiedlich sind, kann es schwierig sein, allgemeine Schlussfolgerungen über die Produktgruppe als Ganzes zu ziehen. Stattdessen können die Modellvorhersagen stark von den spezifischen Merkmalen und Eigenschaften einzelner Produkte abhängen.

Es existieren verschiedene Ansätze zur Bewältigung der Herausforderungen, die durch heterogene Produktgruppen entstehen. Eine Möglichkeit besteht darin, separate Modelle für jede Untergruppe von Produkten zu erstellen (anhand von vorhandenem Domänenwissen), die sich in bestimmten Merkmalen ähneln. Eine andere Möglichkeit besteht darin, fortschrittliche Modellierungstechniken, wie künstliche neuronale Netzwerke, zu verwenden, die in der Lage sind, die hohe Variabilität und Komplexität der Daten zu berücksichtigen. Darüber hinaus können Techniken zur Datenbereinigung und -transformation verwendet werden, um die Heterogenität der Daten zu reduzieren und die Modellierung zu erleichtern.

2.3 Automatisiertes Maschinelles Lernen

Automatisiertes maschinelles Lernen (AutoML) ist ein sich schnell entwickelndes Gebiet, das darauf abzielt, Teile oder den gesamten Prozess des Maschinellen Lernens zu automatisieren. Es umfasst Techniken wie automatische Hyperparameteroptimierung, Modellauswahl und Feature-Engineering.

AutoML hat das Potenzial, die Effizienz und Produktivität in der Datenwissenschaft erheblich zu steigern, indem es zeitaufwändige und komplexe Aufgaben automatisiert und gleichzeitig die Qualität der Modellergebnisse verbessert. Eine wichtige Funktion von AutoML besteht darin, einen geeigneten Algorithmus für ein bestimmtes Problem zu identifizieren. Dies wird oft als Modellauswahl bezeichnet und kann eine erhebliche Verbesserung der Modellgenauigkeit und

-leistung bewirken. AutoML kann auch die Hyperparameter-Optimierung automatisieren, die ein besonders undurchsichtiger und zeitaufwändiger Teil des maschinellen Lernens ist. Hyperparameter sind Modellparameter, die vor dem Training festgelegt werden müssen und einen erheblichen Einfluss auf die Leistung des Modells haben können. Die manuelle Optimierung dieser Parameter ist zeitaufwändig, doch AutoML kann diesen Prozess erheblich beschleunigen und verbessern.

Ein weiterer wichtiger Aspekt von AutoML ist die Automatisierung des Feature-Engineerings. Feature-Engineering beinhaltet die Erstellung, Auswahl und Transformation von Features, die in einem Modell verwendet werden. AutoML kann diesen Prozess automatisieren, indem es die aussagekräftigsten Feature des Datensatzes eines Modells identifiziert und automatisch erstellt.

Darüber hinaus kann AutoML dazu beitragen, die Datenanalyse und Modellierung zugänglicher und benutzerfreundlicher zu gestalten, indem komplexe und technische Aspekte des maschinellen Lernens automatisiert werden. Dadurch wird es auch Nicht-Experten möglich, leistungsstarke und präzise Modelle zu erstellen.

2.3.1 Auto-Sklearn

Auto-sklearn [FKE+15] ist ein AutoML-Tool, das auf der weit verbreiteten Scikit-Learn-Bibliothek [PVG+11] basiert. Es wurde entwickelt, um eine direkte Schnittstelle für Entwickler und Modellerstellung sowie Modelloptimierung zu ermöglichen und zu vereinfachen.

Auto-sklearn verwendet mehrere fortschrittliche Techniken zur Verbesserung der Vorhersagegenauigkeit. Dazu gehören Bayesianische Optimierung, Meta-Lernen und Ensemble-Konstruktion. Bayesianische Optimierung wird verwendet, um den Prozess der Hyperparameterwahl zu automatisieren, indem ein probabilistisches Modell der Ziel-Funktion erstellt und dieses Modell dann verwendet wird, um die nächste Reihe von Hyperparametern zu wählen, die getestet werden sollen. Meta-Lernen wird verwendet, um die anfängliche Auswahl von Algorithmen und Hyperparametern geeignet zu wählen, indem es auf früheren Ergebnissen aufbaut. Schließlich verwendet Auto-sklearn Ensemble-Konstruktion, um eine Reihe von Modellen zu erstellen, die gemeinsam Vorhersagen treffen, wodurch die Vorhersagegenauigkeit verbessert werden kann.

2.3.2 AutoML4Clust

In einer neueren Publikation „AutoML4Clust: Efficient AutoML for Clustering Analyses“ [TFS+21] wird ein neuartiger AutoML-Ansatz für Clusteranalysen vorgestellt. AutoML4Clust unterstützt Analysten beim CASH-Problem (combined algorithm selection and hyperparameter optimization) für Clusteranalysen.

Bisherige AutoML-Systeme konzentrieren sich auf überwachte Lernalgorithmen. AutoML4Clust überträgt die Konzepte auf unüberwachte Clusteralgorithmen. Der Kernunterschied ist, dass für Clustering keine externen Labels zur Bewertung der Ergebnisse vorliegen. Daher wird bei AutoML4Clust eine intrinsische Metrik zur Bewertung der Clusterstruktur verwendet (extrinische Metriken werden auch unterstützt).

Das Vorgehen bei AutoML4Clust ähnelt dem bei AutoML für überwachtes Lernen: Basierend auf einem Konfigurations-Suchraum werden in Optimierungsschleifen Konfigurationen (Algorithmus + Hyperparameter) ausgewählt, bis zu einem festgelegten Budget ausgeführt und anhand einer Metrik

bewertet. Verschiedene Optimierer wie Random Search, Bayes, Hyperband oder BOHB können eingesetzt werden. Als interne Metriken dienen z.B. Calinski-Harabasz, Davies-Bouldin oder der Silhouetten Koeffizient.

Die Evaluierung zeigt, dass AutoML4Clust andere Verfahren bzgl. Genauigkeit und Laufzeit übertrifft. Es werden ähnlich gute Ergebnisse wie eine Brute-Force-Suche erzielt, aber mit einer vielfachen Beschleunigung.

2.3.3 Optuna

Optuna [ASY+19] ist ein Open-Source-Framework im Bereich des AutoMLs zur Optimierung von Hyperparametern. Es zielt darauf ab den Prozess der Hyperparameter-Optimierung zu vereinfachen und zu beschleunigen, indem es eine intuitive und flexible API für die Definition von Optimierungsaufgaben bereitstellt und auch Möglichkeiten zum frühzeitigen Beenden nicht-Zielführender Versuche (Pruning) anbietet. Der Anwender definiert hierbei einen geeigneten Hyperparameter-Suchraum, als auch eine Ziel-Funktion, welche zumeist die Vorhersagegenauigkeit maximieren soll (Aber auch das Zeit-Budget oder die Modellkomplexität minimieren kann).

Optuna führt automatisch die angegeben Anzahl an Versuchen mit unterschiedlichen Hyperparametern durch und lernt aus den Ergebnissen, um die Ziel-Funktion effizient zu optimieren. Dabei verwendet es eine Reihe von fortschrittlichen Techniken zur Hyperparameteroptimierung. Dazu gehören Bayesianische Optimierung und evolutionäre Algorithmen.

Die Flexibilität von Optuna zeigt sich in seiner API, die es Benutzern ermöglicht, komplexe Optimierungsaufgaben mit minimalem Code-Aufwand zu definieren. Diese Flexibilität ist besonders in Anwendungsfällen nützlich, die eine hohe Anpassungsfähigkeit erfordern, wie z.B. das kombinierte Optimieren von Clustering und Klassifikation. In solchen Fällen können benutzerdefinierte Optimierungsaufgaben definiert und dabei verschiedene Aspekte des Optimierungsprozesses gesteuert werden, wie z.B. die Auswahl der Optimierungsmethode, die Definition des Suchraums und die Festlegung der Abbruchkriterien. Diese Flexibilität ermöglicht es, die Optimierungsaufgabe genau auf die spezifischen Anforderungen des Anwendungsfalls anzupassen, um so die bestmöglichen Ergebnisse zu erzielen.

2.4 Datenpartitionierung für die Klassifikation

Es existieren bereits Vorarbeiten von Hirsch et al. [HRM19; HRM20] mit der Anwendung von Klassifikationsverfahren und Datenpartitionierung zur Unterstützung der Qualitätssicherung und Fehlerdiagnose in der Fertigungsindustrie. Dabei werden insbesondere die End-of-Line Tests komplexer Produkte wie Motoren untersucht, um fehlerhafte Bauteile automatisiert zu identifizieren. Hierbei treten verschiedene analytische Herausforderungen auf, welche die Vorhersagegenauigkeit der Klassifikationsalgorithmen einschränken.

In der ersten Arbeit [HRM19] werden zunächst die relevanten Datenquellen sowie die resultierenden Datencharakteristika analysiert und daraus analytische Herausforderungen abgeleitet. Hierzu zählen beispielsweise eine geringe Datenmenge (C1), eine starke Klassenungleichverteilung (C2) sowie

eine heterogene Produktpalette (C3). Anschließend werden verschiedene etablierte Methoden identifiziert, die prinzipiell geeignet erscheinen, um diese Herausforderungen zu adressieren. Dies umfasst Verfahren basierend auf Ensemble Learning, Sampling und Feature-Selection.

Umfassende theoretische und experimentelle Evaluation wird angewendet, um die Eignung der Methoden für die konkreten Gegebenheiten zu überprüfen. Als Ergebnis stellt sich das Ensemble-Verfahren Random-Forest als beste Lösung heraus (Kombination von Feature-Selection und einem Random-Forest lieferten nur leicht schlechtere Ergebnisse).

In der zweiten Publikation [HRM20] bauen Hirsch et al. auf diesen Erkenntnissen auf und stellen ein Klassifikationssystem vor, welches explizit verfügbares Domänenwissen nutzt. Konkret wird eine Produkt-Hierarchie verwendet, um den Datensatz entsprechend ähnlicher Produkt-Varianten in Teilmengen aufzuteilen. Dies reduziert die Heterogenität und zusätzlich kommen Techniken zum Einsatz, um die Klassenverteilung auszubalancieren. Die Ergebnisse zeigen, dass dieser, auf Domänenwissen basierende Ansatz, bestehende Verfahren hinsichtlich Genauigkeit übertrifft und die Anzahl der Nacharbeiten im Prozess reduziert.

Insgesamt leisten beide Publikationen einen Beitrag für ein besseres Verständnis bei der Anwendung von Klassifikationsverfahren auf Daten mit analytischen Herausforderungen, im Kontext der Qualitätssicherung und Fehlerdiagnose. Durch die detaillierte Analyse und den Methodenvergleich wird aufgezeigt, wie sich die Leistungsfähigkeit in realen Szenarien steigern lässt.

2.4.1 Braun

Die Bachelorarbeit von Braun [Bra21] behandelt das Thema der Analyse von Clustering-Algorithmen zur Partitionierung von Trainingsdaten für komplexe Mehrklassenprobleme, basierend auf synthetisch generierten Daten. Die vorgestellte Methode aus [HRM20] setzt spezielles Domänen-Wissen voraus. Ziel der Arbeit von Braun ist es daher zu prüfen, ob eine rein datengetriebene Partitionierung mittels Clustering-Algorithmen ohne Domänenwissen ähnliche Verbesserungen erzielen kann.

Hierfür wird ein Konzept zur datengetriebenen Partitionierung mittels Clustering und anschließender Klassifikation entwickelt. Der Datensatz wird mittels Clustering in Partitionen unterteilt, anschließend werden Klassifikatoren auf diesen Partitionen trainiert und mittels eines Testdatensatzes evaluiert. Zur Partitionierung werden verschiedene Clustering-Algorithmen wie K-Means, X-Means, DBSCAN oder Affinity-Propagation betrachtet. Die Clustering Ergebnisse werden mittels intrinsischer Metriken optimiert und extrinsischer Metriken evaluiert. Dabei werden die separierten Cluster auf Kompaktheit und Separation bewertet. Als Klassifikatoren kommen die Ensemble-Methoden Random Forest (Bagging-Ensemble) und AdaBoost (Boosting-Ensemble) zum Einsatz.

Die Ergebnisse zeigen, dass durch geeignete Partitionierung mit K-Means oder X-Means ähnliche Vorhersagegenauigkeiten wie ohne Partitionierung erreicht werden können. Allerdings ist die hohe Variation der Ergebnisse abhängig von den Daten und Parametern. Trotzdem wird aufgezeigt, dass Clustering prinzipiell eine mögliche Alternative zur Partitionierung der Daten ohne Domänenwissen darstellt.

3 Konzepte und Methodik

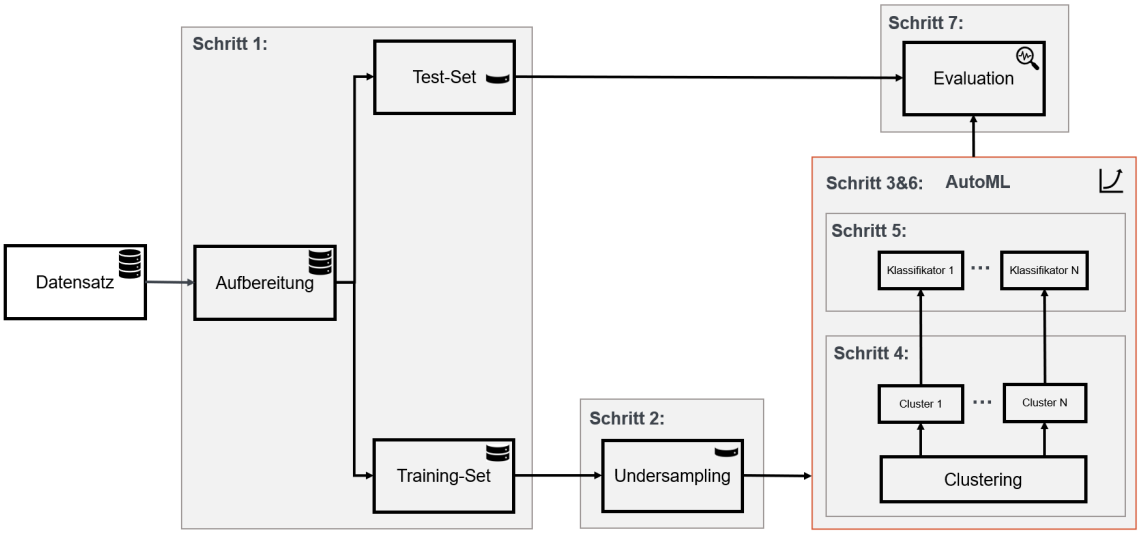

Abbildung 3.1: Abbildung der einzelnen Schritte

Das 3. Kapitel dieser Arbeit befasst sich mit den Konzepten und Methoden, die zur Optimierung der Kombination von Clustering und Klassifikation eingesetzt werden. Es folgt eine Übersicht der einzelnen Schritte (Abbildung 3.1):

1. Schritt: Der Datensatz wird aufbereitet und in einen Trainings- und Testdatensatz geteilt
2. Schritt: Die Datenpunkte im Trainigsdatensatz werden mittels Undersampling verringert
3. Schritt: Das AutoML-Framework wählt Parameter für die kommenden Algorithmen
4. Schritt: Der verringerte Trainingsdatensatz wird mittels Clustering in N Cluster paritioniert
5. Schritt: Für jeden Cluster wird auf diesen Teildaten ein Klassifikator trainiert
6. Schritt: Das AutoML-Framwork optimiert diesen Prozess K Mal anhand der Vorhersagegenauigkeit der einzelnen Klassifikatoren
7. Schritt: Die optimierten Algorithmen werden auf dem Testdatensatz evaluiert

In den folgenden Abschnitten werden die einzelnen Schritte im Detail ausgearbeitet.

3.1 Anwendungsfall-Datensatz

Der erste Schritt in der Umsetzung des Konzepts ist das Aufbereiten der Daten aus dem Anwendungsfall. Der hier zur Untersuchung stehende Datensatz stellt die Sammlung von Informationen dar, die von einem Hersteller von Laserschneidemaschinen, durch einen voll-automatisierten Maschinenprozess, generiert werden. Speziell von vollautomatisierte Maschinen, die in der Lage sind, Bleche autonom einzulegen, mehrere und unterschiedliche Formen auf demselben Blech präzise auszuschneiden und die fertigen Komponenten aus dem Blech zu entnehmen sowie abzutransportieren. Der Datensatz enthält alle Informationen, die vor Beginn dieses automatisierten Prozesses zur Verfügung stehen. Das Ziel ist es, mit Hilfe dieser Informationen Fehler noch vor dem tatsächlichen Prozessdurchlauf zu erkennen. Damit sollen präventiv Stillstände verhindert werden oder auch eine optimierte Reihenfolge von Wiederholungsstufen generiert werden. Wiederholungsstufen sind dabei, automatisierte Versuche der Maschine, das verhakten ausgeschnittenen Teile aus dem Blech zu lösen.

Bei diesem Datensatz stellen sich folgende Herausforderungen (H1-H4):

Die Datensatzgröße, sowohl die Anzahl der Datenpunkte (H1), als auch die Anzahl der Features (H2), verhindern die Ausführung von komplexen Clustering-Algorithmen, welche für die Datenpartitionierung benötigt werden. Die Schnitt-Teile variieren in ihrer Komplexität, verursacht durch unterschiedlichste Material und Geometrie Eigenschaften, was zu einer breiten Streuung der Daten-Charakteristica resultiert. Dies führt zu heterogenen Gruppen innerhalb der Daten (H3). Dieser Aspekt ist von besonderer Bedeutung, da die Mustererkennung der Klassifikaiton erschweren [HRM19] und das Risiko von Overfitting erhöht wird. Zusätzlich gibt es elf verschiedene Klassen in den Daten, welche in Abschnitt 3.1.3 modelliert und erklärt werden. Die Anzahl der Datenpunkte pro Klasse variiert stark (H4), wodurch eine Klassifikation auf diesen Daten zu einer Tendenz der Mehrheitsklassen führt. Die Herausforderungen H1-4 werden in den folgenden Abschnitten aufgegriffen und durch verschiedene Methoden adressiert.

3.1.1 Datenbereinigung

Der Datensatz enthält insgesamt 6,5 Millionen Datenpunkte und 154 Features. Jedoch zeigt sich bei genauerer Betrachtung, dass 32 dieser Features entweder einen konstanten Wert enthalten oder eine hohe Anzahl an fehlenden Werten aufweisen (über 80%). Um die Qualität und Verlässlichkeit der Analyse zu gewährleisten, werden diese Features aus der weiteren Betrachtung ausgeschlossen. Die übrigen Daten sind vollständig und bestehen hauptsächlich aus numerischen Werten, wie zum Beispiel Material- oder Geometrie-Daten, oder aus kategorischen Werten über allgemeinere Informationen wie Modelltyp oder die Art des verwendeten Gasgemsischs des Lasers. Neben den Material- oder Geometrie-Daten pro Schnitt-Teil, sind noch Daten über die relative Position von Pins zu dem jeweiligen Schnitt-Teil. Die Pins sind Teil der Wiederholungsstufen, welche versuchen, dass Schnitt-Teil aus dem Blech zu lösen.

Herausforderung H1 und H2 erschließen sich durch die absolute Größe des Datensatzes, welches ein anschließendes Clustering, aufgrund hoher Speicher- und Laufzeitkomplexität dieser Algorithmen, ohne spezielle Hardware, nicht umsetzen lässt. Hierbei kann durch gezieltes Feature-Engineering die Dimensionalität des Datensatzes werden und somit H2 adressiert werden.

3.1.2 Feature Engineering

Im Rahmen des Feature Engineerings erfolgt eine sorgfältige Auswahl der wichtigsten Features pro Klasse mittels SHAP (SHapley Additive exPlanations) [LL17]. SHAP ist eine Methode zur Erklärung der Ausgabe von Machine-Learning-Modellen. Sie basiert auf dem Konzept der Shapley-Werte [Sha51] aus der kooperativen Spieltheorie und ermöglicht es, den Beitrag jedes Features zur Vorhersage für jeden einzelnen Datenpunkt zu quantifizieren. Die SHAP-Methode ist modellagnostisch und kann für jedes Modell verwendet werden, das eine Ausgabe für eine bestimmte Eingabe liefert. Sie hat jedoch den Vorteil, dass sie konsistent und lokal genau ist. Konsistenz bedeutet, dass der SHAP-Wert eines Merkmals bei einer Änderung des Modells, bei der dieses Merkmal stärker gewichtet wird, nicht abnimmt. Unter lokaler Genauigkeit versteht man, dass die Summe der SHAP-Werte eines Datenpunkts plus dem Basiswert der Vorhersage des Modells für diesen Datenpunkt entspricht.

In diesem speziellen Anwendungsfall wird SHAP für die Feature-Auswahl an dem unausgeglichenen Multiklassen Anwendungsfall-Datensatz angewendet. Hierbei werden die SHAP-Werte für jede Vorhersage-Klasse separat berechnet und verwendet, um die wichtigsten Feature für die Klassifizierung zu identifizieren. Dies ist insbesondere von Nutzen in Situationen, in denen die einzelnen Klassen unausgeglichen sind, da es ermöglicht, Feature zu identifizieren, die für die Vorhersage von Minderheits-Klassen von Bedeutung sind.

Die SHAP-Werte werden in mehreren Schritten berechnet [LL17]. Zunächst wird der Klassifikator (hierbei ein Random-Forest aus Abschnitt 3.4.1) mit den Trainingsdaten trainiert. Anschließend wird für jeden Datenpunkt in den Testdaten ein sogenanntes „Erklärungsspiel“ durchgeführt, bei dem die Feature der Datenpunkte als „Spieler“ fungieren. Die „Auszahlung“ dieses Spiels wird als Differenz zwischen der Vorhersage für den jeweiligen Datenpunkt und einem Basis-Wert definiert. Dieser Basiswert entspricht entweder der Vorhersage ohne Features (Durchschnitt der Klasse in den Daten) oder der durchschnittlichen Vorhersage über alle Beobachtungen hinweg. Die SHAP-Werte für jedes Feature werden bestimmt, indem man den durchschnittlichen zusätzlichen Beitrag, den dieses Feature zum Gesamtnutzen leistet, über alle möglichen Kombinationen und Anordnungen der Features hinweg berücksichtigt. Auf diese Weise erhält man für jeden Datenpunkt und jedes Feature einen SHAP-Wert, der den Erklärungsbeitrag dieses Features für die konkrete Vorhersage angibt. Pro Klasse wird eine Liste von Werten pro Feature erstellt, die sich aufsummiert auf eins belaufen. Auf Basis dieser Liste werden die wichtigsten Feature für jede Klasse ausgewertet und zu einer kombinierten Endauswahl von Features zusammengefasst.

In der Studie von [ME20] lässt sich erkennen, dass die Anwendung von SHAP zur Feature-Auswahl im Vergleich zu anderen gängigen Auswahlmethoden die besten Vorhersageergebnisse für eine anschließende Klassifikation liefert. Nach der Anwendung des SHAP-Frameworks [LL17] auf den gegebenen Datensatz reduziert sich die Anzahl der Features von 122 auf 60, ohne die Vorhersagegenauigkeit der nachfolgenden Klassifikation signifikant zu beeinflussen. Dies ergibt sich daraus, dass entfernte nicht-aussagekräftige Features das Ergebnis der Klassifikation kaum bis gar nicht beeinflussen.

Zusammenfassend kann gesagt werden, dass SHAP eine effektive Methode zur Feature-Selection und Modellinterpretation ist, die es ermöglicht, den Beitrag jedes Features zur Klassifikation auf einer relativen Skala zu quantifizieren. Im Kontext eines unausgeglichenen Multiklassen-Datensatzes kann SHAP die wichtigsten Features für jede Klasse identifizieren und somit eine hohe Vorhersagegenauigkeit bei reduzierten Features beibehalten.

3.1.3 Klassenmodellierung

Die Modellierung der Klassen erfolgt in drei Hauptgruppen: (G1) „Erfolgreich beim ersten Versuch“ (Klasse 0), (G2) „Erfolgreich nach einer von acht Wiederholungsstufen“ (Klasse 1-8) und (G3) „Fehlgeschlagen oder abgebrochen“ (Klasse 9-10). Insgesamt ergeben sich hieraus 11 Klassen, die jedoch logisch in die genannten drei Gruppen zusammengefasst werden.

Die Klassifizierung mit Hilfe dieser Klassen-Modellierung ermöglicht es, im Vorfeld Vorhersagen darüber zu treffen, ob ein Teil im Prozess erfolgreich sein wird, welche Wiederholungsstufe zuerst ausprobiert werden sollte oder ob ein Prozess scheitert oder abbricht. Darüber hinaus ermöglicht eine probabilistische Vorhersage eines passenden Klassifikation-Modells Erkentnis darüber, in welcher Reihenfolge die Wiederholungsstufen durchgeführt werden sollen. Es lassen sich dadurch auch Analysen durchführen, welche die Klassifikations-Ergebnisse granularer miteinander vergleichen können, um auch Aussagen über die Position der korrekten Klasse relativ zu der Wahrscheinlichkeitsverteilung der Vorhersage zu treffen.

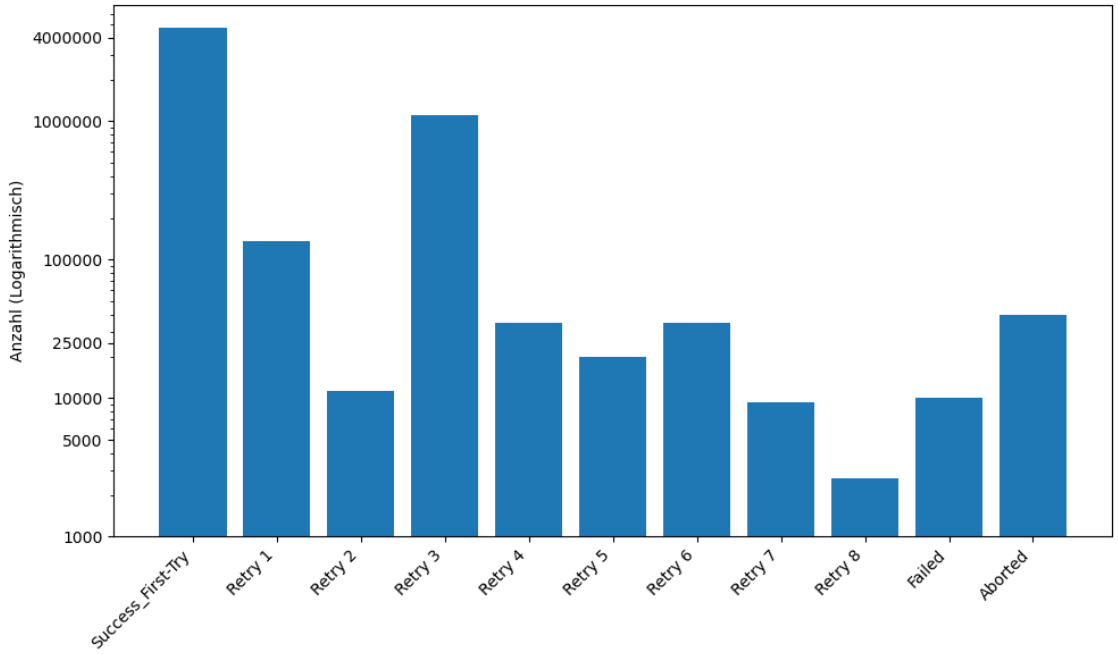

Abbildung 3.2: Klassenverteilung des Anwendungsfall-Datensatzes

Es ist jedoch zu beachten, dass die Klassengruppen (G1) ca. 4 Millionen Mal vorkommt, während die meisten Klassen aus (G2) und (G3) nur einige zehntausend Mal auftreten (Abbildung 3.2). Dies führt zu einem ungleichmäßig verteilten Multi-Klassen-Problem, bei dem die Mehrheitsklasse von dem Klassifikator bevorzugt wird, obwohl die Klassen (G2) und (G3) von größerem Interesse für den erfolgreichen Prozessablauf sind.

3.2 Undersampling

Der zweite Schritt bei der Implementierung der Pipeline ist das Undersampling des Datensatzes. Undersampling wird in dieser Arbeit aufgrund von zwei genannten Herausforderungen aus 3.1 eingesetzt. Primär, um die Größe des Datensatzes zu reduzieren (H1) und sekundär, um das Verhältnis von Mehrheits- zu Minderheitsklassen zu verbessern (H3). Dabei werden die Datenpunkte der Mehrheitsklassen auf eine vorgegebene Anzahl reduziert (abhängig von der Anzahl der Datenpunkte der Minderheitsklassen), welches gleichzeitig die Datensatzgröße verringert und der Ungleichverteilung entgegenwirkt. Damit soll die Vorhersagegenauigkeit der Klassifikation für die Minderheitsklassen verbessert werden.

Hierfür werden verschieden Undersampling-Methoden untersucht und implementiert. Mit Hilfe des Python Moduls „imbalanced-learn“ [LNA17], welches verschiedene Undersampling Verfahren implementiert, können diese in der Pipeline aus Kapitel 4 integriert werden.

3.2.1 RandomUndersampler

Der RandomUndersampler entfernt zufällig ausgewählte Datenpunkte aus den Mehrheits-Klassen. Die Wahrscheinlichkeit eines Datenpunkts für die Entfernung hängt dabei von der Anzahl an Datenpunkten in der jeweiligen Klasse ab. Klassen mit vielen Datenpunkten haben somit eine höhere Wahrscheinlichkeit, Datenpunkte zu verlieren. Der Vorteil dieses simplen Ansatzes ist, die schnelle Laufzeit sowie die einfache Implementierung. Nachteile sind der Einfluss des Zufalls ohne Berücksichtigung des Informationsgehalts der einzelnen Klassen, sowie möglicherweise ungünstige Entfernungen von wichtigen Datenpunkten nahe der Entscheidungsgrenzen. Datenpunkte nahe an diese Grenzen haben einen direkten Einfluss auf die Genauigkeit eines Klassifikators, da diese Datenpunkte relativ ähnliche Features aufweisen, und sich ihre Klasse lediglich durch kleinere Unterschiede differenzieren lassen.

3.2.2 NearMiss

NearMiss [MZ03] ist eine Methode des Undersamplings, die darauf abzielt, informative Datenpunkte zu erhalten. Es gibt drei verschiedene Versionen von NearMiss, in denen die zu entfernenden Datenpunkte auf unterschiedliche Weise ausgewählt werden. NearMiss Version 1 identifiziert dabei für jeden Datenpunkt der Mehrheitsklasse die „k“ nächsten Nachbarn der Minderheitsklasse. Hierbei werden bevorzugt Mehrheitsklassen-Datenpunkte (MD) mit geringeren durchschnittlichen Abständen behalten. Dies führt dazu, dass MD nahe der Entscheidungsgrenzen erhalten bleiben, da diese hier eine höhere Dichte von Minderheitsklassen-Datenpunkte in der Nachbarschaft aufweisen. Weiter entfernte MD von Clustern werden eher entfernt, da diese bereits signifikantere Feature-Unterschiede aufweisen, welche für Klassifikationsmodelle einfacher vorhersagbar sind. Nachteile dieser Methode sind die höhere Zeit- und Speicher-Komplexität bei der Nachbarschaftsberechnungen sowie die Wahl geeigneter „k“ -Werte dieser. Da die Speicherkomplexität von NearMiss Version 2 und 3 zu hoch ist, wird in dieser Arbeit nur NearMiss Version 1 implementiert und untersucht.

3.2.3 OneSidedSelection

One-Sided Selection (OSS) [KM+97] ist ein weiteres komplexes Undersampling-Verfahren, das darauf abzielt, informative Datenpunkte zu erhalten und gleichzeitig Rauschen in den Daten zu reduzieren. OSS kombiniert das Tomek-Links Verfahren [Tom76] zur Rauschreduktion mit dem Condensed Nearest Neighbour (CNN) [Har68] Algorithmus zur Auswahl von Datenpunkten.

Tomek-Links [Tom76] sind Paare von gegenseitig nächsten Nachbarn aus unterschiedlichen Klassen. Wenn ein solches Paar gefunden wird, wird der Datenpunkt der Mehrheitsklasse entfernt. Dies hat zur Folge, dass die Entscheidungsgrenzen klarer definiert sind und Rauschen reduziert wird. Der CNN-Algorithmus [Har68] ist ein „Instance-Selection-Verfahren“, das darauf abzielt, eine Untermenge der ursprünglichen Daten zu finden, die eine ähnliche Klassifikationsleistung dazu erzielt. CNN beginnt mit den Datenpunkten der Minderheitsklasse und fügt dann diejenigen Datenpunkte der Mehrheitsklasse hinzu, die dazu beitragen, die Klassifikationsleistung zu verbessern.

Die Stärke des OSS-Verfahrens liegt in seiner Fähigkeit, Rauschen effektiv zu reduzieren und gleichzeitig eine hohe Klassifikationsleistung zu erzielen. Sein Hauptnachteil ist seine hohe Rechenkomplexität, da sowohl Tomek-Links als auch CNN rechenintensive Verfahren sind. Wie bei NearMiss ist auch bei OSS die Wahl geeigneter Parameter entscheidend für die Effektivität des Verfahrens. In dieser Arbeit wird OSS implementiert, da es trotz seiner höheren Rechenkomplexität eine effektive Methode zur Rauschreduktion und Informationsbeibehaltung bietet.

Insgesamt erlauben die vorgestellten Undersampling-Ansätze eine gezielte Anpassung der Klassenverteilung sowie Reduktion der Datensatzgröße, um so die Voraussetzungen für eine erfolgreiche anschließende Modellierung mittels Klassifikation und Clustering zu schaffen. Insbesondere NearMiss Version 1 und OneSidedSelection versprechen dabei durch Berücksichtigung des Informationsgehalts Vorteile gegenüber einer rein zufälligen Vorgehensweise. Die tatsächliche Effektivität dieser verschiedenen Undersampling Techniken werden in 5.2 analysiert.

3.2.4 Verhältnis der resultierenden Klassen-Anzahl

Das Verhältnis der resultierenden Klassenanzahl nach dem Undersampling ist ein zentraler Aspekt, der sorgfältig berücksichtigt werden muss. Hierfür werden zwei Ansätze untersucht: das relative Undersampling und das absolute Undersampling. Bei dem relativen Undersampling wird die Anzahl der Datenpunkte relativ pro Klasse reduziert, wodurch die originale Klassenverteilung beibehalten wird. Dieser Ansatz hat jedoch nach eigenen Analysen ein schlechteres Ergebnis, da hierbei die Klassenungleichverteilung beibehalten wird und lediglich die Datensatzgröße verringert wird.

Im Gegensatz dazu erlaubt das absolute Undersampling, die Anzahl der Datenpunkte pro Klasse selbst zu definieren. Dies ermöglicht es, gezielt die Mehrheitsklassen zu verringern, ohne die Minderheitsklassen zu reduzieren, was in der Regel zu besseren Ergebnissen führt. Hierbei kann eine Zahl festgelegt werden, ab welcher die Datenpunkte einer Klasse, bis zu dieser Zahl, reduziert werden. Dies hilft nicht nur bei der Reduzierung der Datensatzgröße, sondern auch bei dem Ausgleichen der Klassenungleichverteilung.

3.3 Clustering

Clustering ist eine unüberwachte Lernmethode, die zur Gruppierung ähnlicher Datenpunkte eingesetzt wird. In diesem speziellen Kontext nutzt man Clustering, um die Daten zu partitionieren und homogene Gruppen zu bilden. Diese Gruppierung reduziert die Komplexität der nachfolgenden Klassifizierung und verbessert somit deren Vorhersagegenauigkeit [Bra21]. Da die Feature der Daten verschiedene Wertebereiche aufweisen, ist es notwendig, eine Skalierungsmethode wie den MinMax-Scaler oder Standard-Scaler anzuwenden, um die Daten in einen geeigneten Wertebereich zu transformieren. Dies ist erforderlich, um sicherzustellen, dass die verwendeten Clustering-Algorithmen nicht negativ beeinflusst werden.

Verschiedene Clustering-Methoden werden in dieser Arbeit untersucht und implementiert. Darunter befinden sich KMeans [Mac+67] (Zentrum-basiert), DBSCAN [EKSX+96] (dichtebasiert), HDBSCAN [CMS13] (dichtebasiert und hierarchisch) sowie Agglomerative Clustering (hierarchisch), die jeweils ihre Vor- und Nachteile aufweisen.

Die genannten Clustering-Algorithmen werden in dieser Arbeit verwendet, da sie zum einen eine praktikable Laufzeit- und Speicherkomplexität aufweisen und zum anderen eine differenzierte Analyse der Partitionierung des vorliegenden Datensatzes durch ihre verschiedenen Ansätze ermöglichen.

3.3.1 KMeans

KMeans ist eine weit verbreitete Clustering-Methode, die auf dem Prinzip basiert, Datenpunkte um einen zentralen Punkt, den sogenannten Centroid, zu gruppieren. Die Anzahl der Cluster wird im Voraus festgelegt und die Datenpunkte werden so zugeordnet, dass die Summe der quadrierten Abweichungen von den Centroids minimiert wird. Dieser Algorithmus ist aufgrund seiner Einfachheit und Effizienz weit verbreitet. Allerdings hat dieser auch seine Grenzen, insbesondere bei der Behandlung von Clustern, die nicht kugelförmig sind oder differenzierte und komplexere Strukturen aufweisen.

3.3.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) ist eine dichtebasierte Clustering-Methode, die im Gegensatz zu KMeans keine vordefinierte Anzahl von Clustern benötigt. Stattdessen werden Cluster auf Basis der Dichte der Datenpunkte identifiziert. Datenpunkte in dicht besiedelten Regionen und solche in weniger dicht besiedelten Gebieten werden von DBSCAN unterschiedlich behandelt. Erstere werden zu Clustern zusammengefasst, während letztere als Rauschen klassifiziert werden. DBSCAN ist in der Lage, Cluster unterschiedlicher Formen und Größen zu erkennen, kann jedoch Schwierigkeiten haben, wenn die Parameter des Dichte-Schwellenwerts nicht angemessen gewählt werden. Der "Rausch-Cluster"kann bei heterogenen Datensätzen im Verhältnis zu den anderen Clustern unverhältnismäßig groß sein.

3.3.3 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) ist eine Erweiterung von DBSCAN, die eine hierarchische Clustering-Struktur erzeugt. Ähnlich wie DBSCAN identifiziert auch HDBSCAN Cluster basierend auf der Dichte der Datenpunkte, aber es berücksichtigt auch die Hierarchie der Cluster. Dies ermöglicht es HDBSCAN, eine variable Anzahl von Clustern zu erzeugen und Cluster unterschiedlicher Dichte zu erkennen. HDBSCAN ist eine leistungsfähige Clustering-Methode, die in der Lage ist, komplexe Clusterstrukturen zu erkennen und handzuhaben. Jedoch können ähnliche Probleme wie bei DBSCAN auftreten, wie beispielsweise die geeignete Wahl der Parameter und die Erstellung von „Rausch-Clustern“ , welche übermäßig groß werden können.

3.3.4 Agglomerative Clustering

Agglomerative Clustering ist eine hierarchische Clustering-Methode, bei der jeder Datenpunkt zunächst als eigenständiger Cluster betrachtet wird. Im weiteren Verlauf werden die ähnlichsten Cluster schrittweise zusammengeführt, bis nur noch ein einziger Cluster übrig bleibt oder eine bestimmte Anzahl von Clustern erreicht ist. Die Ähnlichkeit zwischen den Clustern wird durch verschiedene Distanzmetriken bestimmt, wie z.B. die euklidische- oder Manhattan-Distanz. Agglomerative Clustering ist besonders nützlich, wenn eine hierarchische Unterteilung der Daten von Interesse ist, welche in diesem Anwendungsfall von Vorteil sein kann.

3.4 Klassifikation

Klassifikation ist ein überwachtes Lernverfahren, das dazu dient, die tatsächliche Klasse eines neuen Datenpunktes vorherzusagen. Dafür wird der Klassifikator mit den vorhandenen Features und Klassen der Trainingsdaten trainiert. Nachdem die Daten durch die Undersampling-Methode reduziert und anschließend durch das Clustering partitioniert werden, wird die Klassifikation auf die einzelnen Cluster angewendet, um die endgültigen Klassen der Datenpunkte zu bestimmen. Für jedes Cluster wird ein eigener Klassifikator auf den jeweiligen Teil-Daten trainiert. Die Vorarbeiten von Hirsch et al. [HRM19; HRM20] zeigen, dass Ensemble-Learning-Klassifikatoren die besten Ergebnisse liefern, da diese sich besser den analytischen Herausforderungen der Daten anpassen können. Der Klassifikator wird verwendet, um nach der Partitionierung mittels Clustering die Klasse der Datenpunkte vorherzusagen.

3.4.1 Klassifikator-Wahl

Die Wahl eines geeigneten Klassifikators hängt von den Eigenschaften der zugrundeliegenden Daten ab. Die Vorarbeiten von [HRM19; HRM20] zeigen, dass Ensemble-Methoden im Vergleich zu weniger komplexen Klassifikationsverfahren die höchste Vorhersagegenauigkeit für Daten mit den Herausforderungen C1-C3 aufweisen. Hierbei erzielt insbesondere der Random-Forest (RF) Klassifikator sehr gute Ergebnisse. Der behandelte Datensatz in dieser Arbeit umfasst die analytischen Herausforderungen C2 (Klassenungleichverteilung) und C3 (heterogenes Produktportfolio).

Angewandte Wahrscheinlichkeitsvorhersagen

Durch die Modellierung der Klassen aus Abschnitt 3.1.3 wird deutlich, dass die Erstellung einer Wahrscheinlichkeitsverteilung, der vorhergesagten Klassen, eine detailliertere Analyse der Vorhersageleistung des Klassifikationsmodells ermöglicht. Dadurch kann in Kapitel 5 ein präziser Unterschied zwischen einem Baseline-Klassifikator und der in Kapitel 3 vorgestellten Methoden untersucht werden. Für den Hersteller der Laserschneidemaschinen ist es von Nutzen, eine Liste mit Vorhersagen der Klassen für neue Produktionsteile zu erhalten, die auf ihrer Vorhersagewahrscheinlichkeit basieren. Auf diese Weise können effektivere Reihenfolgen für Wiederholungen von Schneidvorgängen gezielt angewendet werden, anstatt bei jedem Produktbauteil dieselbe Standardreihenfolge zu verwenden. Jedoch ist zu beachten, dass das Durchführen der wahrscheinlichsten Wiederholungsstufe nicht automatisch zum erfolgreichen Durchlauf führt. Eine optimierte Reihenfolge der Wiederholungsstufen kann jedoch die Durchlaufzeit eines Teils verringern. Zusätzlich ist für jedes Produkt-Teil die Wahrscheinlichkeit eines fehlgeschlagenen oder abgebrochenen Durchlaufs erkennbar.

Random-Forest-Klassifikator

Der Random-Forest (RF) Klassifikator [Bre01] ist eine häufig verwendete Methode im Bereich des maschinellen Lernens und gehört zur Gruppe der Ensemble-Methoden. Bei einem RF bedeutet Ensemble, dass die Vorhersage auf der Kombination mehrerer Entscheidungsbäume basiert, weshalb sie als Ensemble bezeichnet wird. Die Idee dieser Methodik ist, dass die Kombination von mehreren einfachen Modellen ein komplexes und leistungsstärkeres Modell erzeugt, das die Vorhersagegenauigkeit verbessern und das Risiko von Overfitting reduzieren kann.

Seine Funktionsweise basiert auf dem Prinzip des „Bootstrap Aggregating“ (Baggings). Dabei wird das Modell auf verschiedenen Teilmengen der Trainingsdaten trainiert und die Vorhersagen der einzelnen Entscheidungsbäume anschließend gemittelt oder durch Mehrheitsentscheidung aggregiert. Jeder Entscheidungsbaum im RF wird unabhängig von den anderen Bäumen erstellt. Durch die zufällige Auswahl der Datenpunkte und Feature für jeden Baum werden die Bäume diversifiziert, was zu einer Reduzierung der Varianz des Gesamtmodells führt.

Ein RF kann als Klassifikator die erforderlichen Wahrscheinlichkeitsvorhersagen aller Klassen liefern. Dabei werden für jeden Teilbaum die Wahrscheinlichkeiten berechnet, dass ein Datenpunkt einer bestimmten Klasse angehört. Diese Wahrscheinlichkeit basiert auf der Anzahl der Datenpunkte pro Klasse im letzten Entscheidungsschritt des Random Forests, welcher auch als „Blatt“ (engl. Leaf) auf der tiefsten Ebene des Baums bezeichnet wird.

Ensemble-Methoden, insbesondere ein RF, bieten zahlreiche Vorteile. Diese sind robust gegenüber Ausreißern und Fehlern und sind in der Lage, mit großen Datenmengen und vielen Features umzugehen. Der RF-Klassifikator zeichnet sich durch eine hohe Vorhersagegenauigkeit und die Fähigkeit aus, komplexe Muster in den Daten zu erkennen.

K-Nearest-Neighbor-Klassifikator

Ein weiterer Klassifikator, der in dieser Arbeit verwendet wird, ist der K-Nearest-Neighbor (KNN) Klassifikator [CH67]. Dieser wird lediglich als Baseline-Vergleich zum Random Forest (RF) genutzt und ist nicht Teil der Optimierung. Im Vergleich zu Ensemble-Methoden wie dem RF, ist der KNN-Klassifikator ein einfacheres, aber dennoch effektives Modell im Bereich des maschinellen Lernens. Die Einfachheit des KNN zeigt sich in seiner intuitiven Herangehensweise und direkten Implementierung ohne Annahmen über die zugrundeliegenden Daten.

Die KNN-Methode ist ein Beispiel für instanzbasiertes Lernen, bei dem die Klassifikation eines neuen Datenpunkts auf Basis der K-nächsten Nachbarn in der Trainingsdatenmenge erfolgt. Dabei ist der Wert von K eine vom Anwender festgelegte Konstante (häufig K< 10). Die Klassifikation erfolgt durch eine Mehrheitsabstimmung der K nächstgelegenen Nachbarn des Punktes. Jeder Nachbar trägt dabei mit seiner Klassenzugehörigkeit zum Ergebnis bei. Die euklidische Distanz wird üblicherweise als Maß für die Nähe zu anderen Datenpunkten verwendet.

Obwohl der KNN-Klassifikator aufgrund seiner Einfachheit und Direktheit beliebt ist, besitzt er einige Nachteile. Einer der größten Nachteile des KNN-Algorithmus ist seine Anfälligkeit für hohe Dimensionalität in den Daten [Pes13]. Mit zunehmender Anzahl von Dimensionen (Features) wird der Abstand zwischen den Datenpunkten immer größer und die Datenpunktdichte nimmt ab, was sich negativ auf die Effektivität des KNNs auswirken kann.

Ähnlich wie der RF-Klassifikator kann auch der KNN probabilistische Wahrscheinlichkeitsvorhersagen liefern. Die Wahrscheinlichkeit, dass ein Datenpunkt einer bestimmten Klasse zugeordnet werden kann, wird als der Anteil der Nachbarn dieser Klasse, unter den K nächsten Nachbarn, berechnet. Obwohl der KNN-Klassifikator einfacher und direkter als der RF ist, bietet er dennoch eine effektive Methode, um die Leistung komplexerer Methoden wie dem RF zu bewerten.

Dummy-Classifier

Neben dem KNN wird in dieser Arbeit auch ein Dummy-Classifier als Baseline-Modell implementiert und analysiert. Ein Dummy-Classifier ist ein einfacher Algorithmus, der die Struktur der Input-Daten ignoriert und lediglich die Klassenverteilung der Zielvariablen betrachtet. Dieser wird oft als Vergleichsmaßstab verwendet, um die Leistungsfähigkeit komplexerer Modelle zu bewerten. Es ist wichtig zu beachten, dass der Dummy-Classifier, obwohl er als Baseline-Modell dient, nicht dazu verwendet werden sollte, tatsächliche Vorhersagen in einer realen Anwendung zu treffen. Sein Hauptzweck besteht darin, eine untere Leistungsgrenze zu definieren, anhand derer die anderen Modelle, die untersucht werden sollen, bewertet werden können.

Die Implementierung des Dummy-Classifiers in dieser Arbeit erfolgt mithilfe der Bibliothek sklearn [PVG+11]. Der Dummy-Classifier in sklearn bietet verschiedene Strategien zur Vorhersage von Klassen. Diese Strategien legen fest, wie der Dummy-Classifier Vorhersagen trifft, und können entsprechend der Anwendungsgebiete ausgewählt werden.

Die „stratified“ -Strategie trifft zufällige Vorhersagen auf der Grundlage der Klassenverteilung in den Trainingsdaten. Die „most_frequent“ -Strategie hingegen sagt immer die am häufigsten auftretende Klasse in den Trainingsdaten voraus. Diese Strategie könnte in Situationen mit einem Ungleichgewicht in der Verteilung der Klassen nützlich sein, in denen die am häufigsten

vorkommende Klasse eine hohe Vorhersagegenauigkeit aufweist. Die „uniforme“ -Strategie generiert zufällige Vorhersagen, unabhängig von der Klassenverteilung in den Trainingsdaten. Dabei hat jede Klasse die gleiche Wahrscheinlichkeit, vorhergesagt zu werden.

3.4.2 Metriken

Um die Leistung der Klassifikatoren zu bewerten, werden verschiedene Metriken untersucht. Diese umfassen Accuracy, Balanced-Accuracy und den Macro-F1-Score. Jede dieser Metriken bietet unterschiedliche Informationen über die Leistung des Modells und sollte in Abhängigkeit von den spezifischen Anforderungen des Anwendungsfalls ausgewählt werden. Für die kommenden Metriken werden die Definitionen von Precision und Recall benötigt:

Precision ist das Verhältnis von korrekt positiv klassifizierten Beispielen zu der Gesamtzahl der als positiv klassifizierten Datenpunkte. Es gilt:

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

wobei:

- TP die Anzahl der True Positives ist - die Anzahl der korrekt als positiv klassifizierten Datenpunkte.
- FP die Anzahl der False Positives ist - die Anzahl der fälschlicherweise als positiv klassifizierten Datenpunkte.

Recall ist das Verhältnis von korrekt positiv klassifizierten Beispielen zu der Gesamtzahl der tatsächlichen positiven Datenpunkte:

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

wobei:

- FN die Anzahl der False Negatives ist - die Anzahl der fälschlicherweise als negativ klassifizierten Datenpunkte.

Accuracy

Accuracy ist eine Metrik, die das Verhältnis der korrekten Vorhersagen zur Gesamtzahl der Vorhersagen misst. Sie ist intuitiv verständlich und in vielen Anwendungsfällen nützlich. Allerdings ist sie für ungleichverteilte Klassen unpassend, da diese die Minderheitsklassen weniger stark gewichtet und überwiegend die Vorhersagegenauigkeit der Mehrheitsklassen widerspiegelt. In diesem Kontext kann eine hohe Accuracy irreführend sein, da sie eine hohe Vorhersageleistung suggeriert, während die Leistung auf den Minderheitsklassen möglicherweise sehr schlecht ist. Trotz dieser Einschränkungen wird die Accuracy in dieser Arbeit angegeben, um eine zusätzliche Perspektive auf die Leistung der Klassifikatoren zu bieten.

Balanced-Accuracy

Die Balanced-Accuracy ist eine Metrik, die das arithmetische Mittel der Klassenspezifischen Accuracy berechnet. Sie ist besonders nützlich, wenn die Klassen ungleich verteilt sind, da sie im Gegensatz zur normalen Accuracy die Vorhersagegenauigkeit auf jeder Klasse gleich gewichtet. Die Balanced-Accuracy legt mehr Fokus auf den Recall, da sie die Sensitivität für jede Klasse berücksichtigt, d.h. die Fähigkeit des Klassifikators, positive Fälle korrekt zu identifizieren.

F1-Score

Der F1-Score ist das harmonische Mittel von Precision und Recall und gibt somit eine ausgewogene Bewertung der Leistung des Klassifikators. Sie ist insbesondere nützlich, wenn sowohl False Positives als auch False Negatives wichtig sind. In dieser Arbeit wird der Macro-F1-Score zur Optimierung verwendet, da dieser den F1-Score, unabhängig der Anzahl der Datenpunkte pro Klasse, sondern gemittelt über diese, wertet.

3.5 AutoML zur Hyperparameteroptimierung

AutoML ist von entscheidender Bedeutung für eine geeignete Parameterwahl der Algorithmen in dieser Arbeit. Daher wird AutoML konkret dafür eingesetzt, um einen Suchraum zu definieren, der die Parameter beider Methoden, Clustering und Klassifikation zusammen, umfasst. Der zu optimierenden Wert wird als das Gesamtergebnis der einzelnen Klassifikatoren festlegt. Das bedeutet, dass alle Vorhersagen für die Datenpunkte zusammengetragen werden und mit den wahren Klassen-Label evaluiert werden. AutoML ermöglicht eine Optimierung der Kombination von Clustering und Klassifikation, ohne die Verwendung von Clustering-Metriken, sondern um direkt die Vorhersagegenauigkeit zu optimieren. Sowohl intrinsische als auch extrinsische Clustering-Metriken haben ihre Vor- und Nachteile und können nur begrenzt dazu genutzt werden, um die optimale Kombination von Clustering und Klassifikation zu ermitteln [Bra21]. Jedoch erlauben die Frameworks aus Abschnitt 2.3.1 - Auto-Sklearn und Abschnitt 2.3.2 - AutoML4Clust keine flexiblen Problemstellungen, bei denen ein komplexerer und flexiblerer Optimierungssuchraum erstellt werden muss. Für diese Aufgabe wird das AutoML-Framework Optuna [ASY+19] (Abschnitt 2.3.3 - Optuna) verwendet, da es eine flexible Definition des Suchraums ermöglicht. Optuna ermöglicht eine effiziente Optimierung mittels moderner Algorithmen und Techniken, wie der Bayes'schen Optimierung. Die Wahl geeigneter Optimierungsalgorithmen und Parametersuchräume muss jedoch vom Nutzer analysiert und an den Anwendungsfall angepasst werden.

3.5.1 Optimierungs-Algorithmen

Optuna bietet eine Auswahl an Optimierungs-Samplern, die auf verschiedenen Optimierungsalgorithmen basieren, wie dem Grid-Sampler, dem Random-Sampler und dem CmaEs-Sampler. In dieser Arbeit wird zur automatischen Hyperparameter-Optimierung der Tree-structured Parzen Estimator (TPE) Sampler [BBBK11] eingesetzt. Die Wahl des TPE-Samplers für diese Arbeit hat mehrere Gründe. Im Vergleich zu anderen Bayes'schen Optimierungsalgorithmen, wie Gauß-Prozessen, ist TPE effizienter und schneller [BBBK11], insbesondere bei Problemen mit vielen Parametern

und bedingten Suchräumen. Bedingte Suchräume sind diejenigen, bei denen manche Parameter nur unter bestimmten Bedingungen zusammen auftreten. Diese bedingten Parameter kommen in der Hyperparameter-Optimierung häufig vor und werden für die Optimierung ignoriert, wenn die Bedingung nicht erfüllt ist. Der TPE-Sampler unterstützt auch die multivariate Optimierung [FKH18], was bedeutet, dass er in der Lage ist, nicht-lineare Abhängigkeiten zwischen Parametern zu erkennen.

TPE ist ein bayesischer Optimierungsalgorithmus, der auf der Kerndichteabschätzung beruht. Im Gegensatz zu bayesischen Optimierungsmethoden auf Basis von Gauß-Prozessen, modelliert TPE die Zielfunktionsverteilung p(y|x) bei gegebener Konfiguration x nicht direkt, sondern zerlegt sie in die bedingte Verteilung p(x|y) und p(y), um die nächsten Parameter-Vorschläge zu generieren. Diese iterative Methode erlaubt es dem TPE-Sampler, den Suchraum effizient zu durchsuchen und sich auf Bereiche zu konzentrieren, die wahrscheinlich bessere Ergebnisse liefern.

3.5.2 Optimierungs-Ablauf

Optuna optimiert die gewählte Zielfunktion mit Hilfe einer „Studie“, in der die Zielfunktion spezifiziert wird. Für jede Studie wird eine vorgegebene Anzahl von „Versuchen“ durchgeführt, für die Parameter aus dem Suchraum vorgeschlagen und mit der Zielfunktion bewertet werden. Der Benutzer wählt zunächst einen implementierten Clustering- und Klassifikationsalgorithmus aus. Anschließend werden die Anzahl der Versuche und die „Startversuche“ festgelegt. „Startversuche“ sind die ersten Versuche, die mit einer zufälligen Auswahl von Parametern durchgeführt werden, um eine erste Exploration des Suchraums zu ermöglichen. Die weiteren Versuche werden dann gezielt mit dem TPE durchgeführt, wobei die Optimierungszielfunktion darin besteht, den Macro-F1-Wert der Kombinierten Klassifikatoren zu maximieren.

3.5.3 Parameter und Suchraum

Die Parameter und ihr Wertebereich werden in dieser Arbeit vorab definiert. Parameter, wie die Batch-Größe, stehen in direkter Relation zur Größe des Datensatzes und der Hardware-Ressourcen. Der resultierende Suchraum für den TPE ist eine Kombination aus den Parametern der Clustering- und Klassifikationsalgorithmen. Wenn durch vorgeschlagene Clustering-Parameter „n“ Cluster auf den Daten erstellt werden, werden für jeden dieser Cluster Parameter für die Klassifikatoren erstellt. Damit diese Klassifikationsparameter innerhalb des TPEs differenziert werden können, wird folgendes Namensschema eingeführt: „param_{x}_{n_clusters}_{cluster_index}“. Hierbei steht param_{x} für die jeweiligen Parameter der Algorithmen. {n_clusters} steht für die Gesamtanzahl der durch das Clustering erstellten Cluster und {cluster_index} bezieht sich auf den entsprechenden Cluster. Dieses Schema (explizit {n_clusters}) stellt einerseits sicher, dass die Beziehung zwischen Klassifikations- und Clustering-Parameter für die Anzahl der Cluster erhalten bleibt. Dies basiert auf der Prämisse, dass ähnliche Parameter für das Clustering zu ähnlichen Clustern führen. Dadurch können die bereits verwendeten Klassifikations-Parameter aus vorherigen Optimierungsdurchläufen effektiv verwendet werden. Zudem wird sichergestellt, dass die Parameter auch nach der Optimierung die Zugehörigkeit zum Cluster beibehalten, wenn sie erneut für das Training zur Auswertung verwendet werden (explizit {cluster_index}).

3.5.4 Verwendung von Kreuzvalidierung

Der resultierende F1-Score, der für die Optimierungsfunktion benötigt wird, wird durch Kreuzvalidierung (KV) auf dem partitionierten Datensatz berechnet. Für die KV werden die Datenpartition in verschiedene Trainings- und Testteilmengen unterteilt. Nun werden „k“ Durchläufe gestartet, die das Modell mit den gewählten Parametern auf unterschiedlichen Trainingsteilmengen trainieren und auf der Testteilmenge evaluieren. Nach den „k“ Durchläufen wird der gemittelte Makro-F1_Score pro Teilmenge als Endergebnis gewählt. In dieser Arbeit wird die stratifizierte Kreuzvalidierung (SKV) verwendet, da diese sicherstellt, dass jede Klasse gleichmäßig in den Trainings- und Testdaten vertreten ist, und somit verhindert, dass das Modell eine Klasse bevorzugt, die in den Trainingsdaten überrepräsentiert ist.

Es gibt jedoch auch Herausforderungen bei der Verwendung von SKV für Cluster mit einer geringen Anzahl von Datenpunkten. Wenn die Anzahl der Datenpunkte pro Klasse kleiner als die gegebene Zahl „k“ ist, kann SKV nicht durchgeführt werden. Diese Fälle müssen in der Implementierung erkannt und wie folgt behandelt werden. Zunächst wird versucht, das „k“ so weit zu reduzieren, bis eine Durchführung möglich ist (k=2 ist möglich). Oder es wird auf die normale Kreuzvalidierung zurückgegriffen, wenn nur ein Datenpunkt in einer der Klassen vorliegt. Falls Datenpunkte von nur einer einzigen Klasse in einem Cluster vorliegen, kann und wird hierfür kein Klassifikator trainiert. Hierfür wird die zugehörige Klasse für diesen Cluster gespeichert und ein Macro-F1_Score von 1.0 zurückgegeben.

3.5.5 Vorhersage neuer Datenpunkte

Nach Ermitteln der optimierten Parameter für das Clustering und die Klassifikation durch den Optimierungsalgorithmus, muss die Effizienz der Kombination dieser anhand eines Testdatensatzes evaluiert werden. Um für jeden Datenpunkt im Testdatensatz den richtigen Klassifikator zu wählen, muss eine entsprechende Cluster-Zuordnung der neuen Datenpunkte vorgenommen werden. Dieser Schritt ist jedoch nicht trivial, weshalb in diesem Zusammenhang drei mögliche Ansätze diskutiert werden.

Nächstem Cluster zuweisen

Ein möglicher Ansatz zur Zuweisung eines neuen Datenpunkts zu einem Cluster besteht darin, das nächstgelegene Cluster-Zentrum zu verwenden. Allerdings kann dies bei dichtebasierten Clustering-Verfahren zu Problemen führen, insbesondere wenn die Struktur komplexer ist und das Cluster-Zentrum nicht repräsentativ für diesen Cluster ist.

Mit neuen Datenpunkten erneut Clustern

Eine Alternative besteht darin, Testdatenpunkte in den Trainingsdatensatz aufzunehmen und mit den optimierten Clustering-Parametern erneut zu trainieren. Hierbei besteht jedoch das Risiko, dass sich neue Clusterstrukturen oder eine andere Anzahl von Clustern bilden, die vom optimierten

Muster abweichen und somit zu schlechteren Vorhersagen durch die Klassifikatoren führen können. Zudem entsteht ein erhöhter Zeitaufwand, da das Clustering-Modell für neue Datenpunkte erneut trainiert werden muss.

Extra Klassifikator trainieren

Ein weiterer Lösungsansatz für das Problem der Clusterzuweisung besteht darin, einen zusätzlichen Klassifikator zur Vorhersage des Clusters für neue Datenpunkte zu verwenden. Durch die Verwendung eines Klassifikators kann die Zuordnung eines neuen Datenpunktes zu einem Cluster schnell und präzise vorhergesagt werden, ohne dass das Clustering-Modell erneut trainiert werden muss. Aus diesem Grund wird dieser Ansatz hierfür umgesetzt. Nach der Optimierung des Clusterings und der Klassifikation wird ein zusätzlicher Klassifikator trainiert, der die optimalen Cluster-Labels als Zielklasse beim Training verwendet. Hierbei wird mittels TPE eine eigene Optimierungsschleife durchlaufen, um eine möglichst genaue Zuordnung neuer Datenpunkte zu erreichen. Dieser Schritt wird in Abschnitt 5.3.3 evaluiert.

4 Implementierung

Die Implementierung des vorgeschlagenen Ansatzes erfolgt in einem vollmodularen Framework, das in Python geschrieben ist. Dieses Framework ermöglicht eine flexible und effiziente Umsetzung der verschiedenen Komponenten des Ansatzes, einschließlich Undersampling, Clustering, Klassifikation und Vorhersage neuer Datenpunkte. Durch die Verwendung von Python, und einer modularen Klassen-Schreibweise, kann das Framework einfach erweitert und angepasst werden, um verschiedene Datensätze und Anwendungsfälle zu unterstützen. Hierbei können Undersampling-, Clustering- und Klassifikation-Methoden, mit nur kleinen Änderungen am vorhandenen Code, ergänzt werden.

4.1 Verwendete Module

Für die Implementierung der vollmodularen Pipeline werden verschiedene Python Module verwendet. Diese Module erleichtern die Durchführung von Experimenten und ermöglichen eine effiziente und effektive Umsetzung des Ansatzes. Die Verwendung dieser Pakete bietet mehrere Vorteile, darunter verbesserte Leistung, höhere Flexibilität und eine einfachere Integration mit anderen Tools und Frameworks.

CUML

Die Integration von CUML [RPN20] bietet optionalen GPU-Support. CUML ist eine Kombination von Modulen für GPU-beschleunigte maschinellen-Lern-Algorithmen. Die Implementierung und Verwendung von CUML ist für die Durchführung des Ansatzes auf dem Datensatz unabdingbar. Die Verwendung der GPU-Beschleunigung kann mittels eines Parameters in der Implementation an- oder ausgeschalten werden. Die Integration von CUML benötigte die GPU-Beschleunigung von NumPy [HMW+20], namens CuPy [OUN+17]. Diese Implementierung der Algorithmen beschleunigt den Durchlauf um das 1.5 bis 10 Fache, im Gegensatz zu der Implementierung von Scikit-learn.

Scikit-learn

Scikit-Learn (sklearn) [PVG+11] ist eine effiziente Python-Bibliothek für maschinelles Lernen. Sie bietet eine Vielzahl von Algorithmen über eine direkte Schnittstelle. Sklearn ist für seine klare API und umfassende Dokumentation bekannt. Obwohl es nicht nativ auf GPUs läuft, ist es eine wichtige Bibliothek im Datenwissenschaften-Bereich. In dieser Arbeit wird sklearn für Daten-Vorverarbeitung und der CPU Implementierung von maschinellen-Lern-Modellen verwendet.

Hydra

Hydra [Yad19] wird für das Konfigurationsmanagement verwendet, wobei dieses eine flexible und effiziente Verwaltung von Konfigurationen ermöglicht. Mit Hydra können die verschiedenen Parameter und Einstellungen des Ansatzes zentral verwaltet und angepasst werden, was die Durchführung von Experimenten und die Anpassung des Ansatzes an verschiedene Datensätze und Anwendungsfälle erleichtert. Zusätzlich können Batch-Scripts ausgeführt werden, bei welchen man vor der Ausführung der Pipeline einzelne Parameter flexibel ändern kann.

MlFlow

MlFlow [ZCD+18] wird für das Experiment Tracking verwendet. MlFlow ist ein Open-Source-Plattform, die das Management von maschinellen-Lern-Experimenten erleichtert. Mit MlFlow können die Ergebnisse verschiedener Experimente einfach verfolgt, verglichen und visualisiert werden, was die Evaluierung und Verbesserung des Ansatzes erleichtert. Es erlaubt das Speichern von Parametern und Metriken über mehrere Python-Dateien hinweg, welche in dieser Implementierung unabdingbar wären.

Arrow

Das Arrow-Dateiformat wird für den Datensatz verwendet. Dieses Format bietet mehrere Vorteile, darunter hohe Lese- und Schreibgeschwindigkeiten, effiziente Speicherung von Daten und Unterstützung für eine Vielzahl von Datentypen. Durch die Verwendung des Arrow-Dateiformats kann der Datensatz schnell und effizient verarbeitet werden, was besonders wichtig ist bei großen Datensätzen wie dem in dieser Arbeit verwendeten.

4.2 Pipeline

Die Pipeline ähnelt dem Verlauf aus Abbildung 3.1. Die implementierten Klassen spiegeln die Funktion und den Ablauf dieser Schritte wieder. Hinzu kommen folgende Klassen:

1. Pipeline: Die Pipeline-Klasse dient als zentraler Punkt in der Implementierung, welcher die gewünschte Analyse schrittweise abarbeitet. Hierfür werden Instanzen aus den nötigen Python Klassen erstellt und in einem zentralen Punkt kontrolliert ausgeführt.
2. Optimizer: Der Optimizer ist die größte und komplexeste Klasse, da diese für alle Optimierungsaufgaben eingesetzt wird. In dieser wird das kombinierte Optimieren der Clustering und Klassifikations-Algorithmen durchgeführt, in welcher auch die zugehörige Datenpartitionierung und Zuweisung dieser, für die nötigen Klassifikatoren, umgesetzt wird.
3. Algorithm-Factory: Diese dient als zentraler Punkt, um überall im Code Instanzen der Algorithmen zu erstellen. Dabei werden die Parameter entweder aus der Konfigurations-Datei von Hydra geladen oder durch die Optimierungsfunktion von Optuna (Abschnitt 2.3.3) vorgeschlagen. Diese Klasse ermöglicht es unter Anderem, das modular weitere oder andere Algorithmen in die Pipeline integriert werden können.
4. Model-Retrainer: Dieser wird nach Vollendung der Optimierungsaufgabe verwendet, um als Schnittstelle zwischen den vorgeschlagenen Hyperparameter von Optuna und der Algorithm-Factory zu agieren. Dies ermöglicht erneut einen modularen Austausch vorhandener Algorithmen.

5 Evaluation

In dieser Arbeit werden verschiedene Methoden wie Undersampling, Hyperparameteroptimierung mittels AutoML-Tools, Clustering und Klassifikation angewandt, analysiert und ihre Ergebnisse mit Baseline-Klassifikatoren verglichen. Es werden keine subjektiven Bewertungen vorgenommen und die Informationen sind klar, prägnant und in einfachen Sätzen formuliert. Im Evaluationsabschnitt, der mit Abschnitt 5.1 beginnt, wird der technische Aufbau beschrieben, mit dem die Analyse durchgeführt wird. Im nächsten Abschnitt (5.2) werden die erwähnten Undersampling-Methoden aus Abschnitt (3.2) verglichen, um anschließend die effektivste Methode für die weiteren Schritte zu verwenden. Daraufhin folgt in Abschnitt (5.3) die Analyse der kombinierten Optimierung von Clustering und Klassifikation. Hier werden die Vorhersagen der kombinierten Optimierung aller Cluster-Algorithmen verglichen und es wird eine ausführliche Analyse der Wahrscheinlichkeitsvorhersagen der Klassifikation durchgeführt. Zudem wird in Abschnitt 5.3.3 überprüft, ob der zusätzliche Klassifikator, der die Cluster-Labels auf dem Test-Datensatz vorhersagt (aus Abschnitt 3.5.5), in der Lage ist, die Cluster für die Partitionierung vorherzusagen. In Abschnitt 5.4 wird der zusätzliche Aufwand der kombinierten Optimierung untersucht und in Abschnitt 5.5 wird ein Fazit gezogen.

5.1 Aufbau

Um die Konzepte aus Kapitel 3 unter Verwendung der Implementierung aus Kapitel 4 umzusetzen, werden die Experimente auf einer Virtual-Machine mit Linux (Ubuntu 22.04.3 LTS) durchgeführt. Die Hardware besteht aus 32GB RAM und verwendet 14 vCPUs eines AMD Ryzen 5700X Prozessors. Für die GPU-Beschleunigung wird eine Nvidia RTX 3090 mit 24GB VRAM eingesetzt. Die experimentelle Durchführung erfolgte hauptsächlich auf der GPU unter Verwendung von CUML (vgl. Abschnitt 4.1). Im kommenden werden die Spezifikationen des endgültigen Trainings- und Testdatensatzes sowie die verwendeten Parameter-Suchbereiche der Algorithmen aufgeführt.

5.1.1 Datensatz

Im ersten Schritt der Implementierung wird der verwendete Datensatz gereinigt und die Dimensionalität mittels Feature-Selection reduziert (Kapitel 3.1). Die Struktur des Datensatzes sieht nach diesem Schritt wie folgt aus: Circa 6,5 Millionen Datenpunkte mit 60 Features (ursprünglich 154 Features). Anschließend werden die Daten in einen Trainings- und Testdatensatz mit einer 90/10-Relation aufgeteilt. Diese Aufteilung gewährleistet, dass zur Evaluation ein Testdatensatz zur Verfügung steht, der dem Modell unbekannt ist und nicht durch das Undersampling beeinflusst wird.

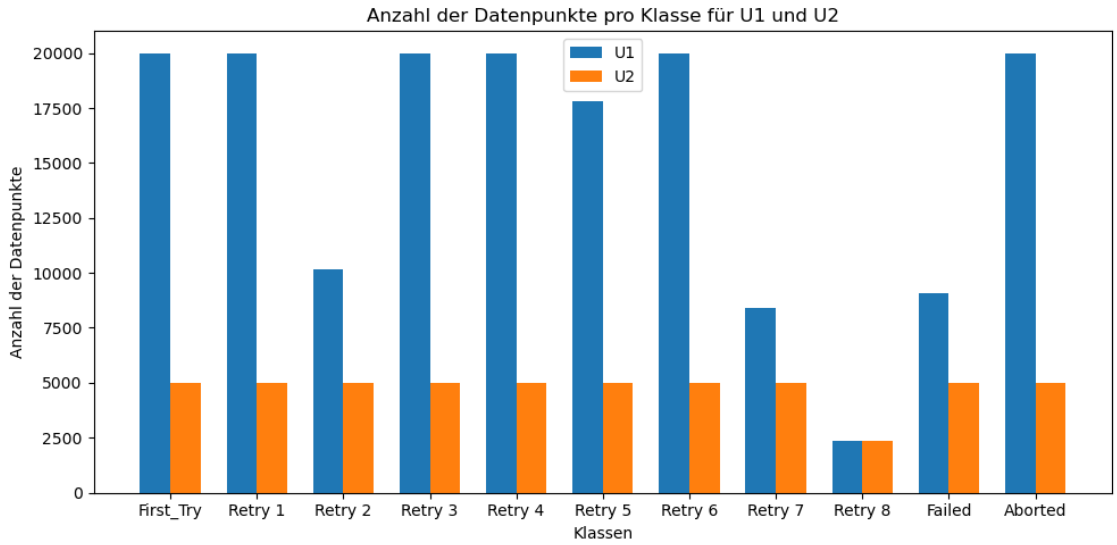

Abbildung 5.1: Klassenverteilung nach Undersampling

Im zweiten Schritt wird die Undersampling-Methode auf dem Trainingsdatensatz angewendet. Hierfür werden die Datenpunkte pro Klasse so weit reduziert, bis die angegebene Anzahl an Datenpunkten pro Klasse erreicht ist. Es werden dabei 2 verschiedene Stärken des Undersamplings verwendet. Einerseits werden pro Klasse bis zu 20000 Datenpunkte beibehalten (U1), um den Einfluss eines moderaten Undersamplings zu untersuchen (maximale Anzahl an Datenpunkten, die die Laufzeit und Speicherkomplexität der verwendeten Algorithmen auf der verwendeten Hardware erlauben). Andererseits werden die Datenpunkte pro Klasse auf 5000 Datenpunkte reduziert (U2), um den Einfluss eines stärkeren Undersamplings zu untersuchen. Es wird dargestellt, wie sich die Stärke des Undersamplings auf die Vorhersagegenauigkeit der Klassifikation auswirkt. Es wird sichergestellt, dass durch das Undersampling nicht zu viel Information entfernt wird und die Methodik dadurch ineffizient wird.

Durch die Anwendung von U2 wird eine ausgeglichene Klassenverteilung erreicht, mit jeweils exakt 5000 Datenpunkten pro Klasse (außer in Wiederholungs-Stufe acht mit 2362). Im Gegensatz dazu wird bei U1 primär die Anzahl an Datenpunkten in den Mehrheitsklassen reduziert, da nur diese über 20.000 Datenpunkte verfügen. Dies führt zur Klassenverteilung in Abbildung 5.1.

5.1.2 Optimierungs-Parameter

Für die Erstellung aller Messergebnisse werden drei verschiedene Optimierungsaufgaben benötigt:

1. Optimierung der Kombination aus Clustering und Klassifikation.
2. Optimierung des extra Klassifikators, für die Vorhersage neuer Datenpunkte
3. Optimierung des Base-Line Klassifikators

Clustering-Algorithmus	Konfiguration
K-Means	n_clusters=2-100; n_init=1-10, tol=(5e-5)-(5e-3)
DBSCAN	eps=0.1-0.5; min_samples:10-250
HDBSCAN	min_cluster_size=10-250; min_samples=10-250; cluster_selection_epsilon=0.1-0.5; alpha=0.75-1.25
AGGLOMERATIVE	linkage=['ward, 'complete', 'average', 'single']; n_clusters=2-100; n_neighbors=5-15

Tabelle 5.1: Optimierung - Clustering-Parametersuchraum

Klassifikator	Konfiguration
Random-Forest	n_estimators=2-150; max_depth=15-100; min_samples_split=2-10; min_samples_leaf=1-10; min_impurity_decrease=(1e-5)-(1e-3); bootstrap=[True, False]; max_features=0.2-1.0; ccp_alpha=(5e-9)-(1e-4)

Tabelle 5.2: Optimierung - Klassifikator-Parametersuchraum

Jede dieser Optimierungsaufgaben besteht aus 200 Durchläufen (20 davon mit zufälligen Startparametern), bei denen der Optimierungsalgorithmus Parameter vorschlägt und den Macro-F1_Score als Optimierungsziel festlegt.

Parameter-Suchraum

Der Suchraum für die Parameter der Clustering-Algorithmen (Tabelle 5.1) sowie des Klassifikators (Tabelle 5.2) wird so definiert, dass er einerseits einen ausreichend großen Optimierungsraum für Optuna bietet, andererseits jedoch nicht zu groß ist, um die Optimierungsdauer und eine Überanpassung an die Trainingsdaten zu vermeiden.

5.2 Undersampling-Verfahren

Um die passende Undersampling Methode für den Datensatz auszuwählen, wird ein RF Klassifikator auf den jeweils reduzierten Datensätzen trainiert und evaluieret (mittels der Klassifikations-Metriken aus 3.4.2). Der Macro-F1_Score, Accuracy und Balanced_Accuracy für die drei getesteten Methoden - RandomUndersampler, Nearmiss und Onesided - zeigen deutliche Unterschiede (Abbildung 5.2).

Die RandomUndersampler-Methode erzielt insgesamt die besten Ergebnisse mit einem Macro-F1_Score von 0,24, einer Accuracy von 0,56 und einer Balanced-Accuracy von 0,58. Im Gegensatz dazu zeigen die Ergebnisse der Nearmiss-Methode in allen drei Kategorien die schlechteste Leistung, was darauf hindeutet, dass sie für diese spezielle Aufgabe weniger geeignet ist. Die Onesided-Methode zeigt eine moderate Leistung mit Werten, die zwischen denen der RandomUndersampler-

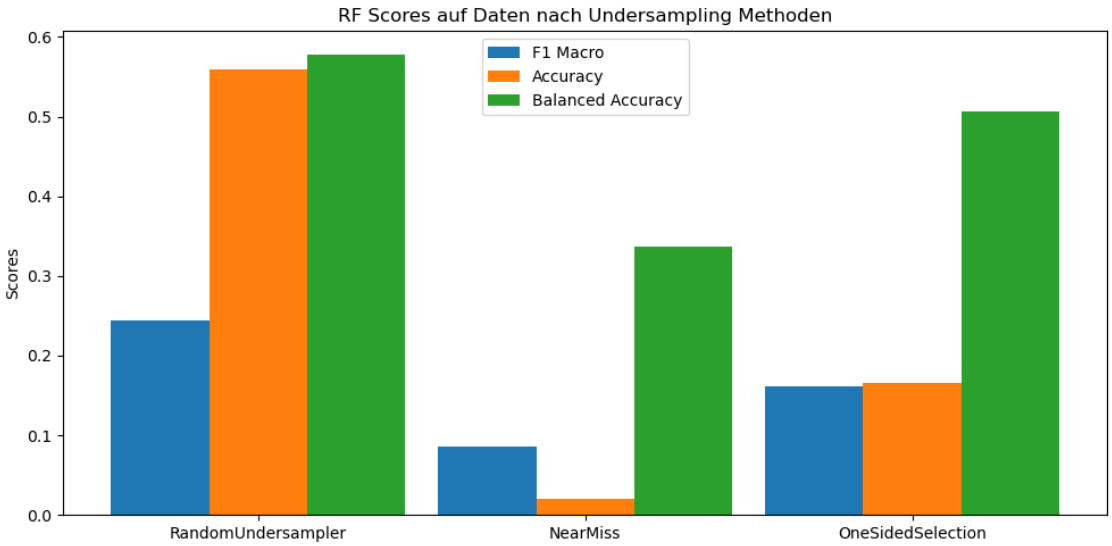

Abbildung 5.2: Scores eines RFs auf Daten nach Undersampling

und Nearmiss-Methoden liegen. Interessant ist, dass diese Methode trotz einer niedrigen Accuracy von 0,17 eine Balanced-Accuracy von 0,51 aufweist. Dies legt nahe, dass sie eine gute Leistung bei der Klassifizierung von Minderheitsklassen erbringt.

Aufgrund dieser Ergebnisse wird die RandomUndersampler-Methode als bevorzugte Methode für das Undersampling auf diesem Datensatz ausgewählt.

5.3 Kombination von Clustering und Klassifikation

Hier wird die Effektivität einer kombinierten Optimierung von Clustering und Klassifikation analysiert. Dabei werden die Clustering-Algorithmen verwendet, die in Abschnitt 3.3 eingeführt werden, und sie werden mit einem optimierten Random-Forest-Klassifikator kombiniert. Der direkte Unterschied in der Vorhersagegenauigkeit im Vergleich zu den Ergebnissen der Baseline-Klassifikatoren wird gegenübergestellt. Das Training der Algorithmen erfolgt auf den Trainingsdatensätzen U1 und U2, während die Vorhersage auf dem Testdatensatz durchgeführt wird.

5.3.1 Vergleich mit Trainingsdaten U2

In Tabelle - 5.3 sind die genauen Ergebnisse der jeweilig untersuchten Algorithmen, welche auf dem Datensatz Undersampling U2 trainiert werden.

Unter den vier betrachteten Clustering-Algorithmen (KMEANS, DBSCAN, HDBSCAN und Agglomerative Clustering) zeigt DBSCAN in Kombination mit einem RF die besten Ergebnisse mit einem F1-Macro-Wert von 0,2273, einer Accuracy von 0,5270 und einer Balanced-Accuracy von 0,5635. KMeans zeigt eine etwas schlechtere Leistung im Vergleich zu DBSCAN mit einem F1-Macro-Wert von 0,2143, einer Genauigkeit von 0,5037 und einer Balanced-Accuracy von 0,5492. HDBSCAN zeigt ähnliche Werte wie KMeans, mit einem F1-Macro-Wert von 0,2165, einer

Methode/Metrik	F1-Macro	Accuracy	Balanced-Accuracy
KMEANS	0.2143	0.5037	0.5492
DBSCAN	**0.2273**	**0.5270**	**0.5635**
HDBSCAN	0.2165	0.5003	0.5526
AGGLOMERATIVE	0.2273	0.5262	0.5626
RF (Optimiert)	**0.2297**	**0.5302**	**0.5637**
RF (Nicht-Optimiert)	0.2207	0.5107	0.5637
KNN (Nicht-Optimiert)	0.1330	0.3626	0.3621
Dummy stratified	0.0346	0.0952	**0.0943**
Dummy most_frequent	**0.0791**	**0.7706**	0.0909
Dummy uniform	0.0333	0.0911	0.0867

Tabelle 5.3: Klassifikations-Ergebnisse (U2 Undersampling)

Accuracy von 0,5003 und einer Balanced-Accuracy von 0,5526. Agglomerative Clustering zeigt eine vergleichbare Leistung wie DBSCAN, mit einem F1-Macro-Wert von 0,2273, einer Accuracy von 0,5262 und einer Balanced-Accuracy von 0,5626.

Unter den Baseline-Methoden erzielt das optimierte Random Forest die besten Ergebnisse mit einem F1-Macro-Wert von 0,2297, einer Accuracy von 0,5302 und einer Balanced-Accuracy von 0,5637. Das nicht optimierte Random Forest zeigt eine etwas geringere Leistung, mit einem F1-Macro-Wert von 0. 2207 erzielt eine Accuracy von 0,5107 und eine Balanced-Accuracy von 0,5637. Das nicht optimierte KNN zeigt die schlechtesten Ergebnisse unter den Baseline-Methoden.

Der F1-Macro-Wert beträgt 0,1330, die Accuracy 0,3626 und die Balanced-Accuracy 0,3621. Die Dummy-Klassifikatoren zeigen wie erwartet die schlechtesten Ergebnisse. Der Dummy-Stratified-Klassifikator hat einen F1-Macro-Wert von 0,0346, eine Accuracy von 0,0952 und eine Balanced-Accuracy von 0,0943 gezeigt. Der Dummy-Most_Frequent-Klassifikator hat die höchste Accuracy von 0,7706 im Vergleich zu 0,0911 für den Dummy-Uniform-Klassifikator.

Wenn die beste Kombination aus Clustering und Klassifikation (DBSCAN) mit dem besten Baseline-Ergebnis (optimiertem Random Forest) verglichen wird, zeigt sich, dass der optimierte RF in allen drei Metriken besser abschneidet. Der Unterschied ist jedoch minimal, was aufzeigt, dass die Kombination von Clustering und Klassifikation fast genauso effektiv sein kann wie die reine Klassifikation. Jedoch sollte man beachten, dass die Datengröße wegen der Speicher- und Laufzeitkomplexität durch Undersampling und Feature-Engineering reduziert werden muss. Diese Maßnahmen wären bei der alleinigen Klassifikation eines RF nicht notwendig gewesen. Auf diese Weise kann ein Random Forest auf dem vollen Trainingsdatensatz optimiert werden und möglicherweise bessere Ergebnisse erzielen.

5.3.2 Vergleich mit Trainingsdaten U1

Nun werden die Ergebnisse der Vorhersage untersucht, bei welchem die Algorithmen auf dem Trainings-Datensatz U1 trainiert werden.

Die Ergebnisse aus Tabelle 5.4 zeigen ähnliche Trends wie jene aus Tabelle 5.3.

Methode/Metrik	F1-Macro	Accuracy	Balanced_Accuracy
KMEANS	0.2764	0.6270	0.5934
DBSCAN	0.2749	0.6262	0.5936
HDBSCAN	0.2692	0.6145	0.5867
AGGLOMERATIVE	**0.2780**	**0.6325**	**0.5937**
RF (Optimiert)	**0.2785**	**0.6338**	**0.5952**
RF (Nicht-Optimiert)	0.2700	0.6223	0.5931
KNN (Nicht-Optimiert)	0.1709	0.4792	0.3912
Dummy (stratified)	0.0396	0.1179	**0.0930**
Dummy (most_frequent)	**0.0791**	**0.7706**	0.0909
Dummy (uniform)	0.0333	0.0911	0.0866

Tabelle 5.4: Klassifikations-Ergebnisse (U1 Undersampling)

Bei den vier Clustering-Algorithmen erzielt dieses Mal der Agglomerative Clustering Algorithmus die besten Ergebnisse, mit einem Macro-F1_Score von 0.2780, einer Accuracy von 0.6325 und einer Balanced-Accuracy von 0.5936. Im Vergleich zu den Resultaten der Analyse des Datensatzes U2 ergeben sich für kombiniertes Clustering und Klassifizieren mit DBSCAN nicht mehr die besten Ergebnisse, obwohl diese weiterhin vergleichsweise nahe beieinander liegen. HDBSCAN erzielt erneut die schlechtesten Ergebnisse mit einem Macro-F1_Score von 0,2692.

Der optimierte RF erzielt hingegen die besten Resultate bei den Basisalgorithmen. Sein F1-Makro-Wert beträgt 0,2785, seine Accuracy 0,6338 und seine Balanced-Accuracy 0,5952. Im Vergleich dazu schneidet der nicht optimierte RF erneut etwas schlechter ab. Außerdem zeigen die Ergebnisse des KNN erneut, dass es Schwierigkeiten hat, die Herausforderungen des Datensatzes zu bewältigen. Unter den Dummy-Baseline-Ergebnissen erreicht der Algorithmus "Dummy most_frequentëine Accuracy von 0,7706, indem er für jeden Testdatenpunkt erneut die Klasse mit den meisten Datenpunkten vorhersagt. Beim Vergleich zwischen dem besten Clustering-Ergebnis (Agglomerative) und dem besten Baseline-Ergebnis (optimierter RF) zeigt sich, dass die Vorhersagegenauigkeit beider Verfahren sehr ähnlich ist. Allerdings erzielt der optimierte RF-Algorithmus leicht bessere Ergebnisse in allen drei Metriken.

Zusammenfassend lässt sich feststellen, dass die Kombination von Clustering und Klassifikation nicht signifikant bessere Ergebnisse liefert als die alleinige Klassifikation. Der Unterschied ist minimal. Des Weiteren zeigt sich, dass die Stärke des Einflusses von Undersampling keine wesentliche Variation in der Vorhersagegenauigkeit der verwendeten Klassifikatoren zur Folge hat. Dadurch wird sichergestellt, dass das Undersampling die Daten nicht zu stark verändert und somit eine geeignete Partitionierung mittels Clustering nicht beeinträchtigt wird.

5.3.3 Klassifikator für die Vorhersage von Cluster-Label neuer Datenpunkte

Da für die Vorhersage der Cluster-Label neuer Datenpunkte ein zusätzlicher Klassifikator benötigt wird, ist es notwendig, den Einfluss dessen zu untersuchen. Eine angemessene Vorhersagegenauigkeit ist erforderlich, um die Datenpartitionierung auf Testdaten zu übertragen. Aus diesem Grund werden die Metriken dieser Klassifikatoren während der Trainingsphase (auf Datensatz U1) erfasst und verglichen.

Algorithmus/Metrik	F1-Macro	Accuracy	Balanced-Accuracy
KMEANS	0.9869	0.9883	0.9866
DBSCAN	0.9970	0.9994	0.9988
HDBSCAN	0.9639	0.9884	0.9498
AGGLOMERATIVE	0.3416	0.9998	0.3429

Tabelle 5.5: Klassifikation der Cluster-Label

Die zusätzlichen Klassifikatoren zur Vorhersage der Cluster-Labels im Testdatensatz erzielen eine ausgezeichnete Genauigkeit von über 0,98 (5.5). Abgesehen vom Agglomerative-Clustering-Algorithmus sind auch die Ergebnisse für den Macro-F1-Score und die Balanced-Accuracy sehr hoch. Dies lässt sich durch den optimierten KMEANS-Algorithmus erklären, der genau 2 Cluster auf den Trainingsdaten mit der Verteilung [0: 111772, 1:56082] erstellt. Die optimierten Algorithmen von DBSCAN und HDBSCAN erzeugen einen großen Cluster, der knapp 80% der Daten enthält, sowie mehrere ähnlich große, kleinere Cluster (insgesamt 21 Cluster bei DBSCAN und 42 Cluster bei HDBSCAN). Im Gegensatz dazu erstellt der Agglomerativ-Clustering-Algorithmus 35 Cluster, von denen 24 jeweils nur einen oder zwei Datenpunkte enthalten. Aufgrund der geringen Anzahl an Datenpunkten ist es dem RF nicht möglich, das Muster für diese Klassen zu erkennen. Für die übrigen Klassen werden jedoch nahezu alle Datenpunkte korrekt klassifiziert, was aufgrund der hohen Genauigkeit bestätigt werden kann. Zusammenfassend kann festgestellt werden, dass der zusätzliche Klassifikator die Muster der Clustering-Labels mit hoher Vorhersagewahrscheinlichkeit identifizieren kann. Daher kann ausgeschlossen werden, dass die Datenpartitionierung der Testdaten aufgrund dieses zusätzlichen Klassifikators scheitert.

5.3.4 Analyse Wahrscheinlichkeits-Vorhersage

Dieser Abschnitt zielt darauf ab, detailliertere Unterschiede in der Vorhersagegenauigkeit der verschiedenen Methoden genauer zu untersuchen. Hierbei werden ausschließlich Clustering-Algorithmen mit den besten Ergebnissen berücksichtigt. Agglomerative Clustering und HDBSCAN zur Datenpartitionierung haben nach den Ergebnissen aus Abschnitt 5.3.2 und 5.3.1 die besten oder fast besten Ergebnisse erzielt. Aus diesem Grund werden diese nun für die erweiterte Analyse verwendet. Für die Bewertung in diesem Abschnitt werden für jeden der verwendeten Algorithmen auf den Testdaten (U1) folgende Werte berechnet:

- Die Top-k-Accuracy ist ein Maß, das angibt, wie oft das korrekte Ergebnis unter den k höchsten Wahrscheinlichkeitsvorhersagen eines Modells zu finden ist. Um diese zu berechnen, ordnet man zunächst die vorhergesagten Wahrscheinlichkeiten in absteigender Reihenfolge. Anschließend vergibt man Rangplätze, wobei der höchste Wert den Rang 1 erhält. Sollten mehrere Vorhersagen denselben Wert haben, wird ihnen der Durchschnitt ihrer Rangplätze zugewiesen. Um die Top-k-Accuracy zu bestimmen, prüft man, ob der wahre Wert innerhalb der ersten k Ränge liegt. Die Genauigkeit ergibt sich aus dem Prozentsatz der Fälle, in denen dies zutrifft.

- Average Rank: Diese Metrik wird ebenfalls mit den durchschnittlichen Rängen der vorhergesagten Wahrscheinlichkeiten berechnet. Im Gegensatz zur Top-k-Accuracy wird hier jedoch der durchschnittliche Rang der korrekten Vorhersagen über alle Testdaten hinweg berechnet.

Diese Metriken ermöglichen eine genauere Analyse der Vorhersagegenauigkeit der verschiedenen Methoden und geben Aufschluss darüber, wie gut die Algorithmen nicht nur die korrekte Klasse vorhersagen, sondern auch die Unsicherheit ihrer Vorhersagen abschätzen können.

Methode	Avg. Rank	Top-1 Accuracy	Top-2 Accuracy	Top-3 Accuracy
DBSCAN	1.8192	0.6262	0.8190	0.8974
AGGLOMERATIVE	1.7891	0.6326	0.8243	0.9015
RF (Optimiert)	**1.7694**	**0.6339**	**0.8271**	**0.9047**
RF (Nicht-Optimiert)	1.8441	0.6224	0.8156	0.8951
KNN (Nicht-Optimiert)	2.8583	0.4716	0.6272	0.7604

Tabelle 5.6: Vergleich der Wahrscheinlichkeitsvorhersage

Beim Vergleich der Methoden (Tabelle 5.6) fällt auf, dass das Agglomerative Clustering und der optimierte RF sehr ähnliche Ergebnisse liefern. Die Top-1 Accuracy, liegt bei beiden Methoden bei etwas über 63%, wobei das optimierte RF mit einer Genauigkeit von 63,4% geringfügig besser abschneidet. Bei der Top-2 Accuracy, übertrifft der optimierte RF das Agglomerative Clustering erneut leicht mit einer Genauigkeit von 82,7% gegenüber 82,4%. Ähnlich verhält es sich mit der Top-3 Accuracy, bei welchem er optimierte RF mit 90,5% gegenüber 90,1% des Agglomerative Clustering leicht vorn liegt. DBSCAN hingegen, schneidet im Schnitt erneut schlechter als das Agglomerative Clustering ab.

Interessant ist auch der Vergleich des Durchschnittlichen Rangs, Hier zeigt sich, dass der optimierte RF mit einem Durchschnittsrang von 1,769 die beste Leistung erbringt, dicht gefolgt vom Agglomerative Clustering mit einem Durchschnittsrang von 1,789.

Insgesamt lässt sich feststellen, dass das Agglomerative Clustering sehr gute Ergebnisse liefert und nur knapp hinter dem optimierten RF liegt. Dennoch ist der zeitliche Mehraufwand des Clustering Umsetzung nicht zu verbachlässigen, wie im folgenden Abschnitt aufgezeigt wird.

5.4 Overhead-Tracking

Um festzustellen, wie sich der zeitliche Aufwand bei der kombinierten Optimierung von Clustering und Klassifikation im Vergleich zur reinen Klassifikationsoptimierung (ohne Clustering) unterscheidet, müssen die Stellen identifiziert werden, an denen zeitlicher Mehraufwand entsteht.

1. Vor der Optimierung:
    - Aufgrund des Clusterings muss die Datensatzgröße mittels Undersampling und Feature-Engineering reduziert werden
    - Die Dauer hängt von dem angewandten Undersampling-Verfahren ab.
        - Der Random-Undersampler verursacht keinen zeitlichen Mehraufwand
        - Wohingegen NearMiss einige Minuten und OneSidedSelection mehrere Stunden benötigt
2. Während der Optimierung:

- Der Clustering-Algorithmus muss auf den Trainingsdaten während der Optimierung trainiert werden
  - Die benötigte Zeit variiert je nach verwendeten Clustering-Algorithmus und kann im Vergleich zur normalen Klassifikation mit einem RF mehrfach höher ausfallen.
- Anschließend müssen die Daten anhand der Cluster-Zuweisung partitioniert werden
- Pro Cluster muss jeweils ein Klassifikator trainiert werden
  - Der Zeitaufwand ist ähnlich wie ohne Partitionierung, da die Datengröße hierbei summiert dieselbe bleibt.

3. Nach der Optimierung:
   - Ein extra Klassifikator muss auf den Cluster-Labels trainiert werden
4. Bei Vorhersage neuer Datenpunkte:
   - Es ist notwendig, einen zusätzlichen Klassifikator zum Vorhersagen der Cluster-Labels zu erstellen.

Daraus lässt sich schließen, dass bei der kombinierten Optimierung von Clustering und Klassifikation ein erheblicher zeitlicher Mehraufwand entsteht. Im Gegensatz zu einer reinen Klassifikation verbessert die Partitionierung die Vorhersagegenauigkeit nicht.

5.5 Schlussfolgerung

Die Analyse der Testergebnisse zeigt, dass eine datengetriebene Partitionierung mittels kombinierter Optimierung von Clustering und Klassifikation leicht schlechtere Ergebnisse als eine normale Klassifikation mit einem RF liefert. Die untersuchten Clustering-Algorithmen sind nicht in der Lage, geeignete Partitionen spezifisch für diesen Datensatz zu bilden, die eine Verbesserung der Klassenvorhersage ermöglichen. Obwohl die Ergebnisse nur geringfügig schlechter sind, zeigt der in Abschnitt 5.4 beschriebene zeitliche Mehraufwand, dass diese Ähnlichkeiten nicht gerechtfertigt sind. Der wahrscheinlichste Grund für die mangelhafte Clustering-Leistung liegt in der Heterogenität des Datensatzes, da dieser aus einer Vielzahl von unterschiedlichen Gruppen besteht, die jeweils ihre eigenen Charakteristika und Muster aufweisen, die von den Clustering-Algorithmen nicht effektiv erfasst werden können. Daher stellt sich die Standard-Klassifikation mithilfe eines RFs als die zuverlässigere Methode zur Vorhersage von Klassen in diesem Anwendungsbereich dar.

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde die effiziente und effektive Verarbeitung und Analyse des Vorhandenen Anwendungsfall-Datensatz untersucht. Insbesondere wurde der Fokus auf die Anwendung von Clustering- und Klassifikationsalgorithmen zur Datenpartitionierung und Vorhersage von Klassen gelegt.

Die Arbeit startete mit einer Literaturübersicht, in welcher die bereits diskutierten Ansätze zur Bewältigung der analytischen Herausforderungen von Produktionsdaten (z.B. kleine Datensatzgröße, Klassenungleichverteilung und ein heterogenes Produktportfolio (C1-C3)) [HRM19] dargestellt wurden. Die Verwendung von Ensemble-Learning-Verfahren wurde als Möglichkeit aufgeführt, um die Klassifikation von Daten mit diesen Herausforderungen zu verbessern. Außerdem wurde gezeigt, dass vorhandenes Domänenwissen in Form einer Produkthierarchie genutzt werden kann, um die Daten effektiv zu partitionieren und die Vorhersagegenauigkeit einer nachfolgenden Klassifikation signifikant zu verbessern [HRM20]. Des Weiteren konnte durch datengetriebene Partitionierung mittels Clustering eine leichte Steigerung der Vorhersagegenauigkeit auf synthetisch generierten Datensätzen erreicht werden [Bra21].

Um die Erkenntnisse dieser datengetriebenen Partitionierung auf reale Produktionsdaten eines Laserschneidemaschinenherstellers zu übertragen, wurde ein umfangreicher Evaluationsprozess durchgeführt. Dabei wurde ein vollständig modularer Pipeline-Ansatz in Python entwickelt, der die Datenaufbereitung, das Undersampling, das Clustering, die Klassifikation, das automatisierte maschinelle Lernen (AutoML), das Modell-Retraining und die Evaluation umfasste. Zusätzlich wurden Python-Module verwendet, um das Tracking und die Durchführung von Experimenten zu vereinfachen und die Algorithmen mit Hilfe von GPUs zu beschleunigen. Der primäre Fokus lag dabei, das Clustering und die anschließende Klassifikation auf diesen Partitionen kombiniert zu optimieren. Hierfür wurde ein AutoML-Framework integriert, welches die kombinierten Clustering- und Klassifikations-Parameter als Optimierungs-Suchraum erhält.

Der Anwendungsfall-Datensatz enthält sowohl die Herausforderungen C2 und C3, hinzu kommt jedoch, dass der Datensatz mit circa 6.5 Millionen Datenpunkten und 155 Features eine neue Herausforderung mit sich bringt. Die Evaluierung der vorgestellten Methoden zeigte, dass das Undersampling essentiell ist, um die Laufzeit- und Speicherkomplexität der verwendeten Cluster-Algorithmen zu reduzieren. Es wurden zwei verschiedene Undersampling-Stärken untersucht, wobei gezeigt wurde, dass eine moderate Reduzierung der Datenpunkte pro Klasse zu besseren Ergebnissen führt als eine stärkere Reduzierung. Zudem wurde festgestellt, dass die Kombination von Clustering und Klassifikation nicht zu signifikant besseren Ergebnissen führt als die alleinige Klassifikation mit einem Random-Forest-Klassifikator.

Die Analyse der Vorhersagegenauigkeit der verschiedenen Methoden zeigte, dass der optimierte Random-Forest-Klassifikator die besten Ergebnisse erzielte. Dasselbe galt für eine detaillierte Analyse der Wahrscheinlichkeitsvorhersagen der verschiedenen Methoden. Es wurde festgestellt,

dass sowohl das Agglomerative Clustering als auch der optimierte Random-Forest-Klassifikator ähnliche Ergebnisse lieferten und nur geringfügige Unterschiede in der Vorhersagegenauigkeit aufwiesen.

Abschließend wurde der zeitliche Mehraufwand der kombinierten Optimierung von Clustering und Klassifikation im Vergleich zur reinen Optimierung der Klassifikation untersucht. Es wurde festgestellt, dass die kombinierte Optimierung einen erheblichen zeitlichen Mehraufwand verursacht, der jedoch keine signifikanten Verbesserungen in der Vorhersagegenauigkeit der verwendeten Algorithmen zur Folge hat.

Insgesamt lässt sich festhalten, dass die vorgestellten Methoden zur Datenpartitionierung und Klassifikation in diesem speziellen Anwendungsfall-Datensatz nicht zu besseren Ergebnissen führen als die alleinige Klassifikation mit einem Random-Forest-Klassifikator. Die Heterogenität des Datensatzes und die begrenzte Fähigkeit der Clustering-Algorithmen, spezifische Muster zu erkennen, wurden als mögliche Gründe für diese Ergebnisse identifiziert.

Ausblick

Für zukünftige Arbeiten gibt es mehrere Ansatzpunkte zur Verbesserung. Eine Möglichkeit besteht darin, weitere Undersampling-Algorithmen zu analysieren, um eine effektivere Reduzierung der Datensatzgröße und eine bessere Anpassung an das Klassenungleichgewicht zu erreichen. Darüber hinaus kann das Undersampling in die Optimierung einbezogen werden, um eine noch effizientere Kombination von Clustering und Klassifikation zu erreichen, welches jedoch den Suchraum der Optimierung vergrößert. Eine weitere Möglichkeit zur Verbesserung der Ergebnisse besteht in der Untersuchung und Implementierung weiterer Algorithmen für Clustering und Klassifikation. Diese können in der modularen Pipeline, ohne großen Aufwand, ergänzt werden. Schließlich könnte die Implementierung von Oversampling-Strategien in Betracht gezogen werden, um das Klassenungleichgewicht zu verbessern. Dies könnte dazu beitragen, die Leistung der Klassifikation zu verbessern, insbesondere in Bezug auf die Minderheitsklassen.

Literaturverzeichnis

[ASY+19] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama. „Optuna: A Next-generation Hyperparameter Optimization Framework“. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019 (zitiert auf S. 20, 34).

[BBBK11] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl. „Algorithms for Hyper-Parameter Optimization“. In: *Advances in Neural Information Processing Systems*. Hrsg. von J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger. Bd. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf (zitiert auf S. 34).

[Bra21] K. Braun. „Analyse von Clustering-Algorithmen zur Partitionierung von Trainingsdaten für komplexe Mehrklassenprobleme“. In: (Apr. 2021) (zitiert auf S. 3, 13, 21, 29, 34, 51).

[Bre01] L. Breiman. „Random forests“. In: *Machine learning* 45 (2001), S. 5–32 (zitiert auf S. 31).

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer. „SMOTE: Synthetic Minority Over-sampling Technique“. In: *Journal of Artificial Intelligence Research* 16 (Juni 2002), S. 321–357. ISSN: http://id.crossref.org/issn/1076-9757. DOI: 10.1613/jair.953. URL: https://cir.nii.ac.jp/crid/1363107370392803456 (zitiert auf S. 17).

[CH67] T. Cover, P. Hart. „Nearest neighbor pattern classification“. In: *IEEE Transactions on Information Theory* 13.1 (1967), S. 21–27. DOI: 10.1109/TIT.1967.1053964 (zitiert auf S. 32).

[CMS13] R. J. G. B. Campello, D. Moulavi, J. Sander. „Density-Based Clustering Based on Hierarchical Density Estimates“. In: *Advances in Knowledge Discovery and Data Mining*. Hrsg. von J. Pei, V. S. Tseng, L. Cao, H. Motoda, G. Xu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 160–172 (zitiert auf S. 29).

[EKSX+96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu et al. „A density-based algorithm for discovering clusters in large spatial databases with noise“. In: *kdd*. Bd. 96. 34. 1996, S. 226–231 (zitiert auf S. 29).

[FKE+15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter. „Efficient and Robust Automated Machine Learning“. In: *Advances in Neural Information Processing Systems 28 (2015)*. 2015, S. 2962–2970 (zitiert auf S. 14, 19).

[FKH18] S. Falkner, A. Klein, F. Hutter. „BOHB: Robust and Efficient Hyperparameter Optimization at Scale“. In: *Proceedings of the 35th International Conference on Machine Learning*. Hrsg. von J. Dy, A. Krause. Bd. 80. Proceedings of Machine Learning Research. PMLR, Juli 2018, S. 1437–1446. URL: https://proceedings.mlr.press/v80/falkner18a.html (zitiert auf S. 35).

[Har68] P. Hart. „The condensed nearest neighbor rule (corresp.)“ In: *IEEE transactions on information theory* 14.3 (1968), S. 515–516 (zitiert auf S. 28).

[HMW+20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant. „Array programming with NumPy“. In: *Nature* 585.7825 (Sep. 2020), S. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (zitiert auf S. 39).

[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)*. 2019, S. 492–503. DOI: 10.1109/DSAA.2019.00064 (zitiert auf S. 3, 13, 20, 24, 30, 51).

[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: *Proc. VLDB Endow.* 13.12 (Aug. 2020), S. 3258–3271. ISSN: 2150-8097. DOI: 10.14778/3415478.3415549. URL: https://doi.org/10.14778/3415478.3415549 (zitiert auf S. 3, 13, 20, 21, 30, 51).

[KM+97] M. Kubat, S. Matwin et al. „Addressing the curse of imbalanced training sets: one-sided selection“. In: *Icml*. Bd. 97. 1. Citeseer. 1997, S. 179 (zitiert auf S. 28).

[LL17] S. M. Lundberg, S.-I. Lee. „A Unified Approach to Interpreting Model Predictions“. In: *Advances in Neural Information Processing Systems 30*. Hrsg. von I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett. Curran Associates, Inc., 2017, S. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf (zitiert auf S. 25).

[LNA17] G. Lemaître, F. Nogueira, C. K. Aridas. „Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning“. In: *Journal of Machine Learning Research* 18.17 (2017), S. 1–5. URL: http://jmlr.org/papers/v18/16-365 (zitiert auf S. 27).

[Mac+67] J. MacQueen et al. „Some methods for classification and analysis of multivariate observations“. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Bd. 1. 14. Oakland, CA, USA. 1967, S. 281–297 (zitiert auf S. 29).

[ME20] W. E. Marcílio, D. M. Eler. „From explanations to feature selection: assessing SHAP values as feature selection mechanism“. In: *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2020, S. 340–347. DOI: 10.1109/SIBGRAPI51738.2020.00053 (zitiert auf S. 25).

[MZ03] I. Mani, I. Zhang. „kNN approach to unbalanced data distributions: a case study involving information extraction“. In: *Proceedings of workshop on learning from imbalanced datasets*. Bd. 126. 1. ICML. 2003, S. 1–7 (zitiert auf S. 27).

[OUN+17] R. Okuta, Y. Unno, D. Nishino, S. Hido, C. Loomis. „CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations“. In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*. 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf (zitiert auf S. 39).

[Pes13] V. Pestov. „Is the k-NN classifier in high dimensions affected by the curse of dimensionality?“ In: *Computers Mathematics with Applications* 65.10 (2013). Grasping Complexity, S. 1427–1437. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2012.09.011. URL: https://www.sciencedirect.com/science/article/pii/S0898122112006426 (zitiert auf S. 32).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830 (zitiert auf S. 19, 32, 39).

[RPN20] S. Raschka, J. Patterson, C. Nolet. „Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence“. In: *arXiv preprint arXiv:2002.04803* (2020) (zitiert auf S. 39).

[Sha51] L. S. Shapley. „Notes on the n-person game—ii: The value of an n-person game“. In: (1951) (zitiert auf S. 25).

[TFS+21] D. Tschechlov, M. Fritz, H. Schwarz, Y. Velegrakis, D. Zeinalipour-Yazti, P. Chrysanthis, F. Guerra. „AutoML4Clust: Efficient AutoML for Clustering Analyses.“ In: *EDBT*. 2021, S. 343–348. DOI: 10.5441/002/edbt.2021.32 (zitiert auf S. 3, 14, 19).

[Tom76] I. Tomek. „Two modifications of CNN“. In: *IEEE Trans. Syst. Man Cybern.* 6 (1976), S. 769–772 (zitiert auf S. 28).

[Yad19] O. Yadan. *Hydra - A framework for elegantly configuring complex applications*. Github. 2019. URL: https://github.com/facebookresearch/hydra (zitiert auf S. 40).

[ZCD+18] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe et al. „Accelerating the machine learning lifecycle with MLflow.“ In: *IEEE Data Eng. Bull.* 41.4 (2018), S. 39–45 (zitiert auf S. 40).

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.

Ort, Datum, Unterschrift