Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Analyse von Clustering-Algorithmen zur Partitionierung von Trainingsdaten für komplexe Mehrklassenprobleme

Kai Braun

Studiengang:	Bsc. Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dennis Tschechlov, M.Sc.
Beginn am:	1. April 2021
Beendet am:	1. Oktober 2021

Kurzfassung

In den letzten Jahrzenten nahm die Anzahl der Datenmengen im Bereich der Informationstechnologie immer weiter zu. Mittlerweile können sie mithilfe algorithmischer Verfahren automatisch ausgewertet werden, um neue Informationen und Erkenntnisse zu erlangen. Diese algorithmischen Verfahren werden im Gebiet des Machine-Learnings fortlaufend weiterentwickelt. Daten aus der Industrie weisen meist komplexe Charakteristika auf, welche die Vorhersagegenauigkeit herkömmlicher Machine-Learning Klassifikatoren verringert. Bekannte Beispiele dafür sind etwa die Multi-Class Imbalance oder ein heterogenes Produktportfolio. Aus einer Vorarbeit [HRM20] geht hervor, dass eine geeignete Partitionierung der Daten die Vorhersagegenauigkeit der Klassifikatoren signifikant verbessert. Jedoch wird bei diesem Verfahren spezifisches Domänenwissen verlangt, mit dessen Hilfe die Daten partitioniert werden. Eine Methode, welche Daten unabhängig von Domänenwissen partitioniert, ist das Clustering. Im Rahmen dieser Arbeit soll geprüft werden, inwiefern eine rein datengetriebene Partitionierung mittels Clustering ähnliche Verbesserungen der Vorhersagegenauigkeit hervorbringen kann wie das domänengetriebene Verfahren. Hierfür wird ein Konzept zur datengetriebenen Partitionierung und anschließenden Klassifikation entwickelt. Ziel der datengetriebenen Partitionierung ist es herauszufinden, durch welche Clustering-Algorithmen eine verbesserte Vorhersagegenauigkeit möglich ist.

Im Rahmen der Arbeit wird für die datengetriebene Partitionierung und Klassifikation folgendes Konzept entwickelt: Der erste Schritt der datengetriebenen Partitionierung sieht vor, den gesamten Datensatz mittels Clustering in einzelne Partitionen zu unterteilen. Im zweiten Schritt werden Klassifikatoren auf den einzelnen Partitionen angewendet. Dabei soll geprüft werden, durch welche Partitionierung der Daten die besten Klassifikationsergebnisse erreicht werden können. Mithilfe verschiedener Clustering-Metriken werden die Clustering-Ergebnisse der einzelnen Algorithmen evaluiert. Außerdem wird untersucht, welche Einflussfaktoren sich am stärksten auf die Klassifikationsergebnisse auswirken. Des Weiteren wird auf spezifische Herausforderungen, wie die Klassenungleichverteilung oder die fehlenden Features, eingegangen. Die Messergebnisse ergeben, dass eine Partitionierung mittels K-Means oder X-Means die besten Werte liefert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1. Einleitung

Im heutigen Zeitalter der Informationstechnologie nimmt die Anzahl der zur Verfügung stehenden Daten immer weiter zu. Neue Entwicklungen, wie das IoT oder Industrie 4.0 bieten immer mehr Möglichkeiten, Daten zu erfassen und auszutauschen. Deshalb ist es von enormer Wichtigkeit, diese Vielzahl an Daten effizient und automatisch verarbeiten zu können, um sie zur Gewinnung neuer Informationen und Erkenntnisse zu nutzen. Mithilfe von Machine Learning Algorithmen können die gesammelten Daten verwendet werden, um automatisiert neue Muster und Zusammenhänge zu erkennen und zu nutzen.

Voraussetzung eines erfolgreich und effektiv arbeitenden Machine Learning Algorithmus ist die Reinheit der zugrundeliegenden Daten. Häufig ist es jedoch der Fall, dass Daten aus der realen Welt komplexe Strukturen aufweisen und diverse Herausforderungen mit sich bringen, welche die Genauigkeit der Algorithmen beeinträchtigen. Typische Merkmale sind dabei eine hohe Dimensionalität der Daten, eine starke Klassenungleichverteilung oder Label Noise.

Ein Bereich, in dem diese Merkmale der Daten auftreten, ist das Qualitätsmanagement in der Produktion. Hier werden Sensordaten genutzt, um die Ursachen für fehlerhafte Produkte zu ermitteln und diese zu reparieren. Diese Sensordaten können als Datensatz zusammengefasst werden, auf welcher Machine Learning Klassifikatoren trainiert werden können, um fehlerhafte Produkte automatisch zu erkennen. Die Produkte weisen jedoch häufig komplexe Strukturen und Merkmale auf, welche die Vorhersagegenauigkeit der Klassifikatoren beeinträchtigen. In Vorarbeit wurden folgende Herausforderungen bezüglich des Sensor-Datensatzes ermittelt [HRM19]:

- Eine relativ kleine Größe des Datensatzes: Kleine Datenmengen erhöhen die Gefahr der Überanpassung (engl. *Overfitting*). Dieses Phänomen beschreibt, dass der Machine Learning Algorithmus nicht korrekte Muster in den Daten erkennt, sondern die Merkmale der einzelnen Instanzen „auswendig“ lernt.
- Multi-Class Imbalance - Klassenungleichverteilung bei Mehrklassenproblemen: Bei einer starken Klassenungleichverteilung existieren einzelne Klassenlabel, welche häufig bei den Instanzen auftreten, während die restlichen Label nur selten vorkommen. Übersetzt auf den Anwendungsbereich bedeutet dies, dass einige wenige Bauteile den Großteil der Defekte in den Produkten verursachen.
- Heterogenes Produktportfolio: Für jedes Produkt existieren eine Reihe verschiedener Ausprägungen und Varianten. Dies führt dazu, dass die jeweiligen gesammelten Sensordaten für jedes Produkt variieren und entsprechende unterschiedliche Merkmale aufweisen. Typisch auftretende Merkmale sind dabei fehlende Features, Subkonzepte oder die Klassenzugehörigkeit.

Anhand der Merkmale des Datensatzes wurde festgestellt, dass herkömmliche Klassifikatoren eher mäßige Vorhersageergebnisse erzielen [HRM19]. Zwar existieren bereits Lösungen, welche die genannten Herausforderungen im Einzelnen adressieren können, jedoch werden dadurch die

Probleme der anderen Herausforderungen verstärkt. Beispielsweise bewährte sich Sampling [YL09] als effektive Strategie, die Klassenungleichverteilung zu reduzieren, jedoch erhöht sie das Risiko des Overfittings, welches durch die Vielzahl an Klassen und das heterogene Produktportfolio entsteht [HRM19].

Aufgrund der eher durchwachsenen Vorhersagegenauigkeit der Klassifikatoren wurde eine Partitionierung der Trainingsdaten auf Basis von Domänenwissen vorgeschlagen [HRM20]. Diese soll die Herausforderungen gleichermaßen adressieren, indem die heterogenen Daten mittels einer vordefinierten Produkthierarchie aufgeteilt werden. Auf den einzelnen Gruppen wird die Klassenungleichverteilung nochmals separat adressiert. Sind die Partitionen gebildet, können einzelne Klassifikatoren auf ihnen separat trainiert werden. Durch diese geeignete Partitionierung der Daten wird die Vorhersagegenauigkeit der Klassifikatoren deutlich verbessert. Jedoch wird bei dem Verfahren spezielles Domänenwissen aus dem Anwendungsbereich verlangt, um die Daten gemäß der Produkthierarchie geeignet aufzuteilen. Steht dieses Domänenwissen nicht zur Verfügung müssen andere Methoden gefunden werden, um die Daten geeignet zu partitionieren.

Eine weitere Möglichkeit der Partitionierung ist das Clustering. Beim Clustering werden anhand der Merkmale der Daten Ähnlichkeiten zwischen den einzelnen Instanzen ermittelt. Ähnliche Instanzen, welche die gleichen Merkmale aufweisen, werden als ein Cluster zusammengefasst. Anders als bei der Aufteilung gemäß Produkthierarchie wird in diesem Fall kein spezielles Domänenwissen verlangt, da Clustering eine rein datengetriebene Partitionierung der Daten vollzieht. Es existieren bereits diverse Ansätze, welche Clustering für eine geeignete Partitionierung der Daten nutzen [LTHJ17; ORSS17; YL09], jedoch sind sie nicht ggeignet, um alle der genannten Herausforderungen zu adressieren. Vielmehr finden sie ausschließlich Verwendung zur Verbesserung existierender Sampling Techniken bei Zweiklassenproblemen, welche ausschließlich die Klassenungleichverteilung adressiert.

Ziel dieser Arbeit ist es zu identifizieren, ob eine rein datengetriebene Partitionierung mittels Clustering ähnliche Verbesserungen der Vorhersagegenauigkeit erzielen kann, wie das domänengetriebene Pendant. Dabei sollen verschiedene Clustering-Algorithmen im Hinblick auf die Qualität ihrer Partitionen analysiert werden, sodass die genannten Herausforderungen gezielt angegangen werden können. Im Rahmen der Arbeit werden eine Vielzahl unterschiedlicher Clustering-Algorithmen (partitionierend, dichte-basierend, hierarchisch) betrachtet.

Der grobe Ablauf der datengetriebenen Partitionierung sieht folgendermaßen aus: Der zugrundeliegende Datensatz wird zunächst in einen Trainings-Datensatz und einen Test-Datensatz aufgeteilt. Der Trainingssatz wird mithilfe eines Clustering-Algorithmus in Partitionen aufgeteilt. Auf diesen einzelnen Partitionen können Klassifikatoren trainiert werden. Mithilfe des Test-Datensatzes kann die Vorhersagegenauigkeit der einzelnen Klassifikatoren geprüft werden. Jede Instanz des Testsatzes wird dem nächstgelegenen Cluster zugeteilt und der entsprechende Klassifikator angewendet. Zur Messung und Evaluation der Vorhersagegenauigkeit werden herkömmliche Metriken aus dem Machine Learning Bereich, wie Accuracy, Precision, etc., herangenommen.

Die vorliegende Arbeit ist auf folgende Weise gegliedert: In Kapitel 2 werden die Grundlagen der Partitionierung und Klassifikation behandelt und verwandte Arbeiten aufgeführt. Im Speziellen wird hier die Vorarbeit der Partitionierung auf Basis von Domänenwissen im näheren betrachtet. Kapitel 3 beschäftigt sich detailliert mit den Konzepten und Methoden der datengetriebenen Partitionierung und anschließenden Klassifikation. In Kapitel 4 werden die Ergebnisse der Partitionierung und Klassifikation vorgestellt und die Resultate evaluiert. Schließlich fasst Kapitel 5 die Erkenntnisse der Arbeit zusammen und stellt Anknüpfungspunkte für weiterführende Arbeiten vor.

2. Grundlagen und verwandte Arbeiten

Dieses Kapitel beschreibt die Grundlagen der Klassifikation und des Clusterings. Sowohl das Clustering, als auch die Klassifikation sind Verfahren, welche typischerweise im Bereich des maschinellen Lernens angewendet werden. Das Grundprinzip des maschinellen Lernens ist es, mithilfe algorithmischer Verfahren Muster aus gegebenen Daten automatisch zu erkennen und zu erlernen. Das maschinelle Lernen lässt sich grundlegend in zwei Arten unterteilen: Dem **überwachten Lernen** und dem **unüberwachten Lernen**.

Beim überwachten Lernen existieren sogenannte Trainingsdaten, mit denen ein Algorithmus Muster erkennen und erlernen soll. Bei diesen Trainingsdaten ist die korrekte Ausgabe bereits bekannt (etwa Sensordaten von Produkten, bei denen bekannt ist, welches Bauteil den Defekt verursacht). Mithilfe dieses Datensatzes werden die Muster erlernt, um so für neue Daten die gewünschte Ausgabe vorherzusagen. Die Klassifikation ist dabei ein Beispiel für das überwachte Lernen.

Beim unüberwachten Lernen existiert grundsätzlich keine gewünschte Ausgabe, die es vorherzusagen gilt. Hier geht es vielmehr darum, anhand der Merkmale der gegebenen Daten Strukturen und gleichartige Ausprägungen zu erkennen. Beim Clustering werden diese Daten gleicher Ausprägung in ein Cluster zusammengefasst.

2.1. Klassifikation

Die Klassifiaktion ist ein Teilbereich des überwachten Lernens. Hier werden anhand eines gegebenen Datensatzes Muster erlernt, um für weitere Daten die gewünschte Ausgabe vorherzusagen.

Der Datensatz ist eine Menge aus Instanzen. Jede Instanz besitzt dabei verschiedene Merkmale, sogenannte Features. Zudem erhält jede Instanz ein Klassenlabel, welches die gewünschte Ausgabe (Klasse) repräsentiert. Bei Zweiklassenproblemen kann dieses Label lediglich zwei Werte (in der Regel „Wahr“ und „Falsch“) annehmen. Ein typisches Beispiel für ein Zweiklassenproblem ist die Frage, ob eine Person anhand gegebener Symptome (Features) an einer Krankheit (Klassenlabel) erkrankt ist. Bei Mehrklassenproblemen hingegen kann das Klassenlabel mehrere diskrete Werte annehmen. Ein Beispiel für ein Mehrklassenproblem ist die Aufgabe, anhand gegebener Sensordaten herauszufinden, welches von vielen Bauteilen einen Defekt verursacht.

Bei Klassifikationsproblemen wird der gegebene Datensatz typischerschweise in einen Trainings- und einen Test-Datensatz aufgeteilt. Mithilfe des Trainings-Datensatzes wird ein Klassifikator trainiert, welcher die bereits bekannten Muster innerhalb des Datensatzes erlernen soll. Danach wird der Klassifikator auf dem Test-Datensatz angewendet, das heißt, er versucht das Klassenlabel der Instanzen des Test-Datensatzes vorherzusagen. Dieser Vorgang wird auch labeln genannt. Da das Klassenlabel dieser Instanzen bereits bekannt ist, kann das tatsächliche Label mit der Vorhersage

des Klassifikators verglichen werden, um somit die Genauigkeit des Klassifikators zu ermitteln. Außerdem kann der Klassifikator verwendet werden, um das Klassenlabel neuer Instanzen, bei welchen das Label nicht bekannt ist, vorherzusagen.

2.1.1. Multi-Class Imbalance

Eine Klassenungleichverteilung liegt vor, wenn die im Datensatz vorkommenden Klassenlabel nicht gleichverteilt sind. Dementsprechend existieren zu einigen wenigen Klassen sehr viele Instanzen, während die restlichen Klassen nur wenige Instanzen aufweisen. Häufig auftretende Klassen werden Majority-Klassen (und deren Instanzen Majority-Instanzen) genannt, während selten auftretende Klassen als Minority-Klassen (Minority-Instanzen) bezeichnet werden. Die Multi-Class Imbalance stellt dabei eine Erweiterung der Klassenungleichverteilung für Zweiklassenprobleme dar, bei der die Ungleichverteilung auf Daten mit mehr als zwei Klassen betrachtet wird. Die Klassenungleichverteilung ist ein präsentes Thema und ein aktives Forschungsgebiet im Bereich des maschinellen Lernens [HYS+17; Kra16].

Herkömmliche Klassifikatoren schneiden vergleichsweise schlecht auf Daten mit ungleicher Klassenverteilung ab, da sie die Instanzen der Majority-Klasse tendenziell bevorzugen. Instanzen der Minority-Klasse werden dann häufig vom Klassifikator als Majority-Klasse gelabelt [HYS+17]. Dies liegt in der Regel daran, dass die Klassifikatoren bei falscher Vorhersage einer Minority-Instanz weiterhin vermeintlich gute Ergebnisse erzielen, da Minority-Instanzen nur selten im Datensatz enthalten sind. Liegt beispielsweise ein Datensatz vor, bei dem 99% der Daten einer Klasse A und nur 1% der Klasse B angehören, kann ein Klassifikator, welcher alle Instanzen der Klasse A zuordnet, eine Genauigkeit von exakt 99% erzielen. Diese vermeintlich gute Vorhersagegenauigkeit spiegelt jedoch kein gutes Klassifikationsergebnis wider. In vielen Anwendungsbereichen, wie etwa der Krebsforschung oder der Intrusion-Detection, kann eine Missklassifizierung einer Minority-Instanz fatale Folgen mit sich bringen. Aus diesem Grund ist es in vielen Bereichen von enormer Wichtigkeit, genau diese Minority-Instanzen korrekt zu erkennen [KGJH16].

Grundsätzlich werden zwei verschiedene Methoden unterschieden, um das Problem der Klassenungleichverteilung anzugehen:

- **Methoden auf Datenebene:** Hier wird gezielt versucht, die zugrundeliegenden Daten so zu modifizieren, sodass herkömmliche Klassifikatoren bessere Ergebnisse erzielen. Die populärste Methode hierbei ist das Sampling, bei dem entweder Instanzen der Majority-Klasse(n) entfernt werden (undersampling) oder künstliche Instanzen der Minority-Klasse(n) hinzugefügt werden (oversampling). Daraus entsteht ein neuer Datensatz, bei welchem die Klassenlabel gleichmäßiger verteilt sind.
  Sampling gilt als effektive Methode, um die Klassenungleichverteilung zu adressieren [YL09], jedoch wird die Gefahr des Overfitting bei einem kleinen Datensatz oder hoher Anzahl an Label Noise erhöht.
- **Methoden auf Algorithmenebene:** Bei diesen Methoden werden die Algorithmen verändert, um Ungleichverteilungen gezielt zu adressieren. Eine beliebte Methode ist dabei das *Cost-Sensitive Learning*, bei dem eine Missklassifizierung von Instanzen mit Kosten verbunden werden. Gehört etwa eine Instanz der Klasse A an, wird vom Klassifikator jedoch der Klasse

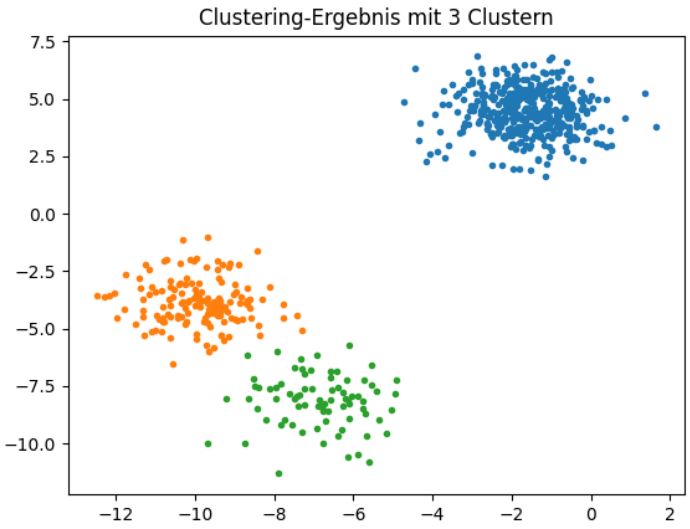


Abbildung 2.1.: Beispiel einer Einteilung des Datensatzes in drei Cluster mit K-Means

B zugeschrieben, so werden Kosten für diese Missklassifizierung aufgeschrieben. Durch höhere Kosten bei falscher Klassifizierung einer Minority-Instanz wird vom Algorithmus ein höherer Wert darauf gelegt, diese richtig zu klassifizieren.

2.2. Clustering

Das Clustering ist eine Methode des unüberwachten Lernens. Anhand der gegebenen Features der Instanzen wird hier versucht, Ähnlichkeiten bezüglich der Ausprägungen innerhalb der Daten zu finden und diese Daten in gemeinsame Gruppen einzuteilen. Hierbei sollen Daten innerhalb der Gruppen möglichst kompakt und zwischen den Gruppen möglichst separiert sein. Anders als bei der Klassifikation existiert beim Clustering kein Klassenlabel, es wird demnach nicht von außen bestimmt, nach welchem Kriterium die Daten gruppiert werden sollen. Die Einteilung in die einzelnen Cluster geschieht allein durch die Betrachtung der jeweiligen Features der Instanzen. Eine der populärtsten Clustering-Algorithmen ist der **K-means** [Mac67]. Dieser ist ein partitionierender Algorithmus und dient als zeiteffiziente Methode, den Datensatz in k Cluster einzuteilen. In Abbildung 2.1 ist die Einteilung eines Datensatzes in drei Cluster mittels K-Means dargestellt.

2.2.1. Clustering auf ungleichverteilten Daten

Auch bei nicht gelabelten Daten kann eine Ungleichverteilung im Hinblick auf die Ausprägungen vorliegen. Ein bekannter Effekt, welcher zu beobachten ist, wenn Clustering-Algorithmen auf ungleichverteilten Datensätzen angewendet werden, ist der sogenannte **Uniform Effekt**. Dieser tritt auf, weil herkömmliche Clustering-Algorithmen, wie etwa K-Means, versuchen, die Daten in gleich große Cluster einzuteilen [XWC09]. Dadurch werden Instanzen aus kleinen Clustern mit

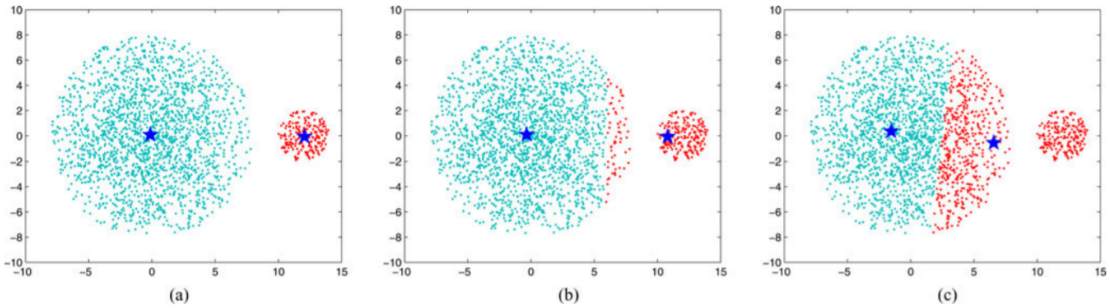


Abbildung 2.2.: Uniform effekt bei Anwendung des Clustering-Algorithmus K-Means - Aus: [LBDC12]

Instanzen aus anderen, größeren Clustern vermischt und tatsächliche Clusterzentren werden von den Algorithmen verschoben. Abbildung 2.2 liefert eine graphische Visualisierung des Uniform Effektes bei der Anwendung eines Clustering-Algorithmus.

2.2.2. Multi-Centroid Ansätze

Um den Uniform Effekt zu verhindern wurde von mehreren Arbeiten das Konzept der Multi-Centroid Ansätze vorgeschlagen [LBDC12; LCT21]. Die grundsätzliche Idee hinter diesem Ansatz ist es, den Datensatz anstelle der gewünschten Anzahl an Clustern in mehrere Subcluster zu unterteilen. Dadurch werden große Cluster in mehrere kleinere Cluster unterteilt, während kleinere Cluster gleich bleiben. Daraufhin werden Subcluster, welche eng beieinander liegen, wieder zu großen Clustern zusammengefügt. Dieses Verfahren verhindert die Zuteilung von Instanzen aus größeren Clustern in ein kleines Cluster. Eine mögliche Einteilung in Subcluster ist in Abbildung 2.3 zu sehen.

2.2.3. X-Means

Ein weiterer Clustering-Algorithmus, welcher ebenfalls effektiv gegenüber ungleichverteilten Daten ist, ist der X-Means Algorithmus [PM02]. Der X-Means Algorithmus kann Daten ebenfalls in Cluster unterschiedlicher Größe partitionieren und somit den Uniform Effekt vermeiden.

Grundsätzlich basiert der X-Means Algorithmus auf dem K-Means Algorithmus, jedoch ist es nicht nötig, eine feste Clusteranzahl anzugeben. Stattdessen wird eine gewisse Reichweite $[u, o]$, in der sich die Anzahl der Cluster bewegen soll, verlangt. Zunächst nimmt der Algorithmus die untere Grenze u des Cluster-Intervalles und teilt den Datensatz mittels K-Means in u Cluster ein. Anschließend wird jedes Cluster nochmals in zwei Subcluster aufgeteilt. Dabei wird der BIC Score zu Hilfe genommen, um zu messen, ob die Unterteilung in zwei Subcluster ein besseres Clustering-Ergebnis hervorbringt als ohne. Die Aufteilung in Subcluster wird so lange fortgesetzt, bis die optimale Clusteranzahl innerhalb der vorgegebenen Reichweite gefunden wurde.

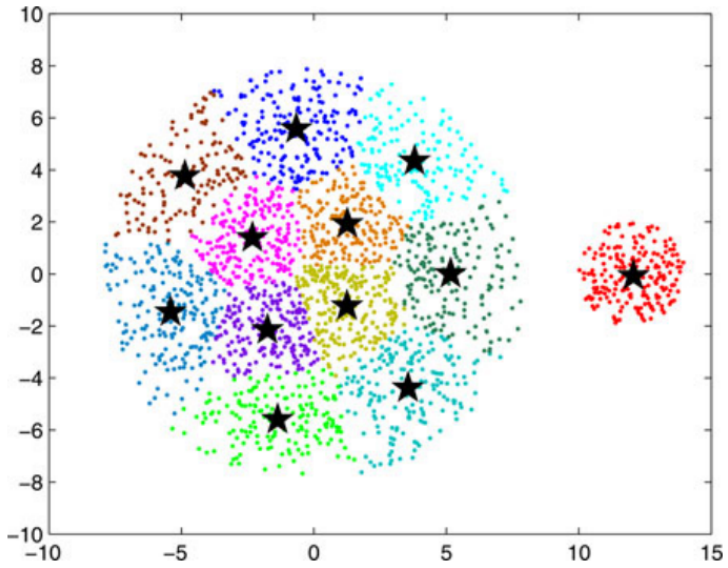


Abbildung 2.3.: Multi-Centroid Ansatz zur Adressierung des Uniform Effekt - Aus: [LBDC12]

2.2.4. AutoML4Lcust

AutoML4Clust [TFS21] ist ein Framework, welches eine Unterstützung zur Auswahl geeigneter Clustering-Algorithmen und deren Parameter bietet. Die Auswahl geeigneter Parameter für ein gutes Clustering-Ergebnis stellt selbst für erfahrene Analysten eine Herausforderung dar. Unerfahrene Analysten haben bei der Auswahl passender Clustering-Algorithmen und deren Konfigurationen noch größere Schwierigkeiten, da sie oft die Auswirkungen einzelner Parameter auf das Ergebnis nicht kennen oder richtig einschätzen können. Eine umfangreiche Untersuchung auf geeignete Algorithmen und Parameter kann sich dann oft als ein sehr zeitaufwändiges Unterfangen herausstellen. Eine Reihe verschiedener AutoML Systeme zur automatischen Identifizierung passender Algorithmen und Hyperparameter existieren bereits im Bereich des überwachten Lernens [FKE+19; THHL13].

Das AutoML4Clust Framework verläuft nach folgendem Schema (siehe Abbildung 2.4):

- **Definieren der Inputs:** Als Input erhält das Framework den Datensatz (D), eine interne Metrik zur Evaluation des Clustering-Ergebnis (M), sowie ein Budget (l).
- **Optimierungsschleife:** Zunächst wird eine Konfiguration (Kombination aus Algorithmus und Hyperparameter) aus dem Konfigurationsraum (CS) ausgewählt. Diese Konfiguration wird auf dem Datensatz ausgeführt und das Ergebnis mithilfe der internen Metrik (M) evaluiert. Anhand der bisher ausgewerteten Konfigurationen wird eine neue Konfiguration aus CS ausgewählt und der erste Schritt wiederholt. Diese Schleife setzt sich so lange fort, bis das Budget (l) aufgebraucht ist.
- Die anhand von (M) berechnete **beste Konfiguration** wird zurückgegeben und final auf dem Datensatz ausgeführt.

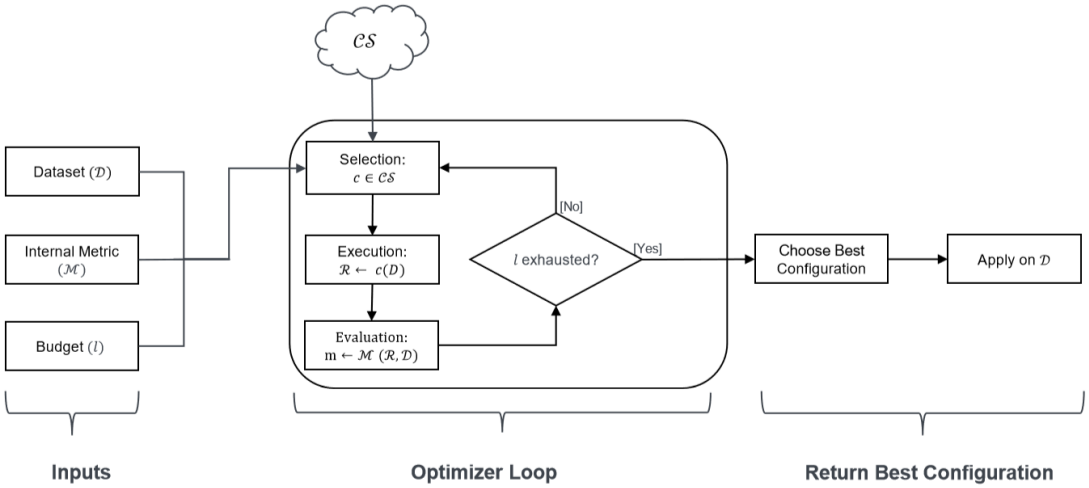


Abbildung 2.4.: Funktionsweise des AutoML4Clust Frameworks - Aus: [TFS21]

2.3. Partitionierung gelabelter Daten

Auch bei gelabelten Daten existieren bereits Ansätze, die den Datensatz mittels Clustering in einzelne Partitionen aufteilen [LTHJ17; ORSS17; YL09]. Die Partitionen werden genutzt, um die Klassenungleichverteilung zu adressieren und bessere Klassifikationsergebnisse zu erzielen. Eine geeignete Partitionierung der Daten wird jedoch ausschließlich als Vorbereitung für effizientere Sampling Strategien durchgeführt. Algorithmen wie RUS oder SMOTE erweisen sich bereits als bewährte Techniken gegen die Klassenungleichverteilung. Dementsprechend wird versucht diese Techniken weiter zu optimieren. Andererseits beschränken sich viele der Techniken auf Zweiklassenprobleme bzw. wurden nicht auf Daten mit mehreren Klassenlabel getestet.

Die Idee der Partitionierung für Sampling Strategien besteht darin, geeignete Datenpunkte für das Sampling zu identifizieren. RUS beispielsweise wählt zufällige Instanzen der Majority-Klasse aus dem Datensatz aus, um mit den Instanzen der Minority-Klasse einen neuen, balancierten Datensatz zu erstellen. Dadurch können wichtige Instanzen der Majority-Klassen und somit Informationen zu den tatsächlichen Mustern der Klasse verloren gehen. Damit diese Informationen erhalten bleiben, können beispielsweise alle Instanzen der Majority-Klasse geclustert werden, um damit alle relevanten Instanzen aus einem Cluster als Basis der Sampling Strategie auszuwählen [LTHJ17]. Andere Strategien teilen entweder den gesamten Datensatz in Cluster ein, um auf den einzelnen Clustern das Sampling durchzuführen [YL09] oder führen das Clustering auf den Minority-Instanzen aus [ORSS17], um Majority-Instanzen dem nächstgelegenen Cluster aus Minority-Instanzen zuzuteilen.

2.4. Hirsch et. al.

Diese Arbeit basiert auf den Vorarbeiten von Hirsch et. al. [HRM19; HRM20]. In seinen Arbeiten wird der zugrundeliegende Datensatz aus dem Qualitätsmanagement der Automobilindustrie auf seine Herausforderungen hin analysiert und Lösungsansätze zur Bewältigung der Herausforderungen ermittelt.

2.4.1. Herausforderungen

Anhand des gegebenen Datensatzes wurden in der Arbeit von Hirsch et. al. zunächst die konkreten Herausforderungen bezüglich des Datensatzes formal beschrieben. Folgende Herausforderungen wurden dabei ermittelt:

- C1 - Kleines Sample Set X: Die Größe des Datensatzes von 1050 Instanzen ist eine relativ kleine Anzahl bei einer großen Menge von 84 Klassen. Daraus folgt, dass viele dieser Klassen nur sehr wenige Instanzen besitzen, was den Lernprozess der Klassifikatoren erschwert. Zudem besitzt jede Instanz bis zu 115 Features. Diese hohe Dimensionalität der Daten, in Kombination mit den wenigen Instanzen pro Klasse, führt zu einem hohen Risiko von *overfitting*.
- C2 - Klassenungleichverteilung: Die zehn größten Klassen (Majority-Klassen) erscheinen in etwa 44% der gesamten Instanzen, während die restlichen 56% der Instanzen über die 74 weiteren Klassen (Minority-Klassen) verteilt sind. Dies verstärkt das Problem des Overfitting aus C1, da Minority-Klassen tendenziell noch weniger Instanzen zur Verfügung stehen. Außerdem tritt durch die Klassenungleichverteilung der Effekt auf, dass Klassifikatoren nicht in der Lage sind, Minority-Instanzen korrekt zu identifizieren. Im Detail wird das Problem der Multi-Class Imbalance in Abschnitt 2.1.1 beschrieben.
- C3 - Heterogenes Produktportfolio: Im Domänenbereich der Automobilindustrie lassen sich viele verschiedene Produktvarianten mit unterschiedlichen Ausprägungen finden. Diese Heterogenität spiegelt sich in den Daten wider. So kann es beispielsweise sein, dass ein Produkt die Motorvariante mit sechs Zylindern verwendet, während andere Produkte eine Variante mit vier Zylindern nutzen. Dementsprechend lassen sich in manchen Instanzen sechs Sensorsignale finden, während andere lediglich vier Signale aufweisen. Die Heterogenität der Daten wurde in drei Unterpunkten näher erläutert:
  - C3.1 - Fehlende Features: Das oben beschriebende Szenario ist ein typisches Beispiel des Problems der fehlenden Features. Diese treten aufgrund der Existenz unterschiedlicher Produktvarianten auf, welche jeweils verschiedene Sensordaten sammeln. Aufgrund der heterogenen Daten ist es häufig der Fall, dass mehrere der insgesamt 100 Features pro Dateninstanz nicht mit einem Wert behaftet sind (etwa 17% aller im Datensatz vohandenen Features). Da Klassifikatoren für gewöhnlich die fehlenden Features nich behandeln können, müssen diese entweder ganz entfernt oder mit abgeschätzten Werten befüllt werden.

  - C3.2 - Subkonzepte: Für gewöhnlich besitzen Instanzen, welche derselben Klasse angehören, vergleichbare Ausprägungen bezüglich der Werte ihrer Features. Bei heterogenen Daten können jedoch häufig mehrere Ausprägungen für eine Klasse existieren. Dies führt dazu, dass ein Klassifikator diese Subkonzepte separat lernen muss, was im Zusammenhang mit C1 die Vorhersagegenauigkeit erschwert.
  - C3.3 - Klassenzugehörigkeit: Die unterschiedlichen Produktvarianten führen dazu, dass nicht jedes der insgesamt 84 Klassenlabel eine theoretisch mögliche Zuweisung für eine Instanz sein kann. Bei einer Produktvariante mit Vierzylinder-Motor etwa kann nicht der sechste Zylinder das defekte Bauteil sein. Diese Informationen schränken die möglichen Klassenzuweisungen ein und würden die Klassifikation vereinfachen. Klassifikatoren kennen diese falschen Klassenzuweisungen jedoch nicht und können ungültige Komponenten als Klassenlabel vorhersagen.

- C4 - Label Noise und überlappende Klassen: Als Label Noise werden die Trainingsdaten bezeichnet, welche fälschlicherweise ein inkorrektes Klassenlabel zugewiesen bekommen. Ausreißer können ebenfalls als Label Noise angenommen werden. Label Noise erschwert die Vorhersage der Klassifikatoren, da durch diese Instanzen verfälschte Muster erlernt werden. Ein ähnliches Problem zum Label Noise ist die Klassenüberlappung. Hierbei kann es vorkommen, dass Ausprägungen zweier unterschiedlicher Klassen sich im selben Bereich befinden und sich überschneiden. Beide Phänomene schränken die Vorhersagegenauigkeit der Klassifikatoren ein. Wie sich das Problem der Label Noise bei komplexen Mehrklassenproblemen gezielt angehen lässt ist jedoch Bestandteil einer anderen Arbeit und wird hier nicht im Näheren betrachtet.

2.4.2. Datengetriebene Fehlerbehandlung

In einer Vorarbeit wurden verschiedene, bereits existierende, datengetriebene Verfahren angewendet und deren Wirksamkeit evaluiert [HRM19]. Dabei wurden die Algorithmen *KNN* und *C5.0*, sowie die Ensembles AdaBoost und Random Forest verwendet. Auf allen Klassifikatoren wurden verschiedene Sampling Methoden (*RUS,ROS,SMOTE*), sowie die *OvA* Technik und *Boruta* als Feature Selection Algorithmus angewendet, mit welchen versucht wurde, die Ergebnisse noch weiter zu optimieren. Beim Klassifizieren wird für jede Instanz eine geordnete Liste mit Wahrscheinlichkeiten zurückgegeben, welche jeweils angeben, mit welcher Sicherheit die Instanz zur jeweiligen Klasse gehört. Die Wirksamkeit aller Algorithmen wurde mithilfe der Accuracy-Werte der Ergebnisse geprüft. Herfür wurden Accuracy-Werte A@1 bis A@10 betrachtet, welche angeben, ob das richtige Label in den Top 1 bis 10 der geordneten Listen der Klassifikatoren vorzufinden ist.

Die Ergebnisse zeigen, dass der Random Forest in Kombination mit Boruta am besten unter den Klassifikatoren abschneidet. Dies lässt sich darauf zurückführen, dass Random Forest durch die zufällige Auwahl von Features besser mit dem heterogenen Produktportfolio umgehen kann. Außerdem wird das Risiko des Overfittings des kleinen Datensatzes minimiert.

Ein weiterer Klassifikator, der ebenfalls gute Ergebnisse liefert, ist AdaBoost. Dieser erzielt die besten Ergebnisse ohne zusätzliche Methoden.

Des Weiteren lieferten die Ergebnisse die Erkenntnis, dass Sampling-Strategien vergleichsweise schlecht auf dem Datensatz abschneiden. Die Accuracy-Werte zeigen, dass keine der Sampling-Strategien eine klare Verbesserung der Ergebnisse hervorbringt. Im Gegenteil verschlechtern sich in die Messergebnisse in den meisten Fällen, teilweise deutlich. Sampling gilt normalerweise als eine der effektivsten Methoden, um die Klassenungleichverteilung anzugehen. In diesem Fall wird jedoch davon ausgegangen, dass Sampling nicht mit der Herausforderung des kleinen Datensatzes, sowie des heterogenen Produktportfolios, zurechtkommt und zu Overfitting tendiert.

2.4.3. Partitionierung auf Basis von Domänenwissen

Da die Klassifikatoren auf dem gesamten Datensatz keine besonders erfolgreichen Ergebnisse liefern, wird von Hirsch et. al. eine neue Methode vorgestellt, welche spezifisches Domänenwissen ausnutzt, um eine geeignete Partitionierung der Daten durchzuführen. Dabei wird der Datensatz in zwei Schritten aufgeteilt:

Im ersten Schritt, der *Segmentierung nach Produkthierarchie* (SPH), wird das spezielle Domänenwissen aus dem Fachbereich verwendet, um den gesamten Datensatz in Bauteilgruppen aufzuteilen. In diesen einzelnen Partitionen sind nun lediglich Bauteile vorzufinden, welche gemäß der Produkthierarchie ähnliche Strukturen aufweisen. Diese Aufteilung soll speziell die Herausforderungen des heterogenen Produktportfolios angehen. Sowohl das Problem der fehlenden Features als auch die Klassenzugehörigkeit können durch die Gruppierung der Daten ähnlicher Bereiche weitestgehend vermieden werden.

Nachdem im ersten Schritt verschiedene Partitionen der Daten gemäß der Produkthierarchie erstellt wurden, wird im zweiten Schritt in der *Klassenpartitionierung gemäß Ungleichverteilung* gezielt die Multi-Class Imbalance angegangen. In diesem Schritt wird mittels Gini-Index auf jeder bereits im ersten Schritt erstellten Partition die Klassenungleichverteilung gemessen. Falls der Gini-Index einen Schwellenwert überschreitet, wird die Partition nochmals in zwei Subgruppen (Major- und Minor-Gruppen) unterteilt. Welche Klassen dabei der Major- und welche der Minor-Gruppe zugeteilt werden, wird durch das p-Quantil $G(p)$ (mit $p = 0.8$) ermittelt.

Nachdem die Partitionierung vollständig auf dem Datensatz durchgeführt wurde, wird auf jeder einzelnen Partition ein Klassifikator trainiert. Da sich der Random Forest bereits aus der Vorarbeit bewährt hat, wird dieser Klassifikator auf den partitionierten Daten ebenfalls verwendet. Neue Instanzen, die nun klassifiziert werden sollen, werden zunächst unter Zuhilfenahme der Produkthierarchie der entsprechenden Partition zugeordnet. Liegt auf dieser Partition eine Aufteilung in Major- und Minor-Gruppen aufgrund der Ungleichverteilung vor, werden die Klassifikatoren beider Gruppen angewendet und die geordneten Listen beider Klassifikatoren zusammengefügt. Liegt diese Aufteilung nicht vor, wird der einzige vorliegende Klassifikator für diese Partition verwendet.

Durch die Partitionierung der Daten lässt sich, verglichen mit der Klassifikation durch Random Forest und Boruta auf dem gesamten Datensatz, eine Verbesserung der Accuracy-Werte feststellen.

3. Konzepte zur Partitionierung der Trainingsdaten und Klassifikation

Dieses Kapitel beschreibt Konzepte und Methoden zur datengetriebenen Partitionierung für komplexe Mehrklassenprobleme. Der Lösungsansatz ist grundsätzlich in zwei Teilschritte unterteilt: Dem **Clustering** zur geeigneten Einteilung in Partitionen und der anschließenden **Klassifikation** auf den einzelnen Partitionen. Für beide Schritte werden im Rahmen dieser Arbeit geeignete Konzepte zur Umsetzung des jeweiligen Teilschrittes entwickelt und Metriken zur Evaluation untersucht. Der Fokus der Arbeit liegt hierbei auf dem ersten Teilschritt, dem Clustering. Abschnitt 3.1 beschreibt zunächst das Vorgehen zur datengetriebenen Partitionierung und anschließenden Klassifikation. In Abschnitt 3.2 und Abschnitt 3.3 werden die jeweiligen Konzepte zum Clustering und zur Klassifikation im Detail erläutert.

3.1. Vorgehen

Abbildung 3.1 liefert eine Übersicht zum allgemeinen Konzept, eine datengetriebene Partitionierung mit anschließender Klassifikation mittels Clustering zu erreichen.

Zunächst wird der Datensatz, wie gewöhnlich bei der Klassifikation, in einen Trainings- und einen Test-Satz aufgeteilt.

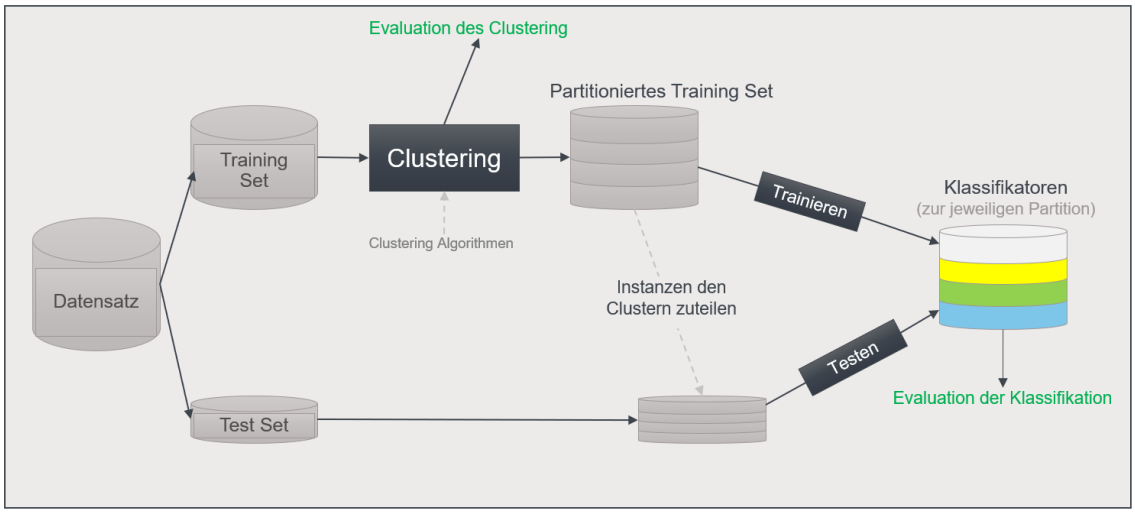


Abbildung 3.1.: Übersicht zur Vorgehensweise bei der datengetriebenen Partitionierung

Der Trainingsdatensatz soll im ersten Schritt mittels Clustering partitioniert werden. Hierzu werden die in Abschnitt 3.2 beschriebenen Algorithmen jeweils betrachtet und angewendet. Die Clustering-Ergebnisse werden mit den in Abschnitt 3.2.3 beschriebenen Metriken evaluiert und verfeinert.

Nachdem die Daten durch den entsprechenden Clustering-Algorithmus aufgeteilt wurden, können die Klassifikatoren auf den einzelnen Partitionen trainiert werden. Die trainierten Klassifikationsmodelle werden pro Partition abgespeichert.

In der Test-Phase können nun neue Instanzen herangenommen und mithilfe der Klassifikatoren das Klassenlabel vorhergesagt werden. Hierfür werden die Instanzen aus dem Test-Datensatz verwendet. Da nun eine Reihe von Klassifikatoren vorliegt, muss zunächst für jede Instanz des Test-Datensatzes ein entsprechender Klassifikator zugeteilt werden. Um diesen zu bestimmen wird das Clustering-Ergebnis des Trainings-Datensatzes zu Hilfe genommen: Je nach verwendetem Clustering-Algorithmus liegen die Clusterzentren der einzelnen Partitionen noch nicht vor. In dem Fall müssen die Clusterzentren zunächst manuell berechnet werden. Jede Instanz des Test-Datensatzes berechnet den Abstand zu allen Clusterzentren, um daraus das nächstgelegene Cluster zu bestimmen. Sobald das nächstgelegene Cluster ermittelt wurde, kann der dem Cluster zugehörige Klassifikator verwendet werden, um das Klassenlabel der Instanz vorherzusagen.

Nachdem alle Daten klassifiziert wurden, wird zuletzt noch eine Evaluation des Klassifikationsergebnisses durchgeführt, um die Vorhersagegenauigkeit der Klassifikatoren zu überprüfen.

3.2. Clustering-Phase

Mittels Clustering wird der Trainings-Datensatz in die einzelnen Partitionen eingeteilt. Damit eine geeignete Partitionierung durchgeführt werden kann, müssen entsprechende Clustering-Algorithmen gefunden und deren Parameter optimiert werden. Außerdem muss definiert werden, wann ein Clustering-Algorithmus gute Ergebnisse liefert. Diese drei Aspekte werden in den nächsten Abschnitten im Detail erläutert.

3.2.1. Clustering-Algorithmen

Der erste große Schritt zur geeigneten Partitionierung ist die Wahl der Clustering-Algorithmen. Eine Reihe bekannter und verbreiteter Clustering-Algorithmen wurde ausgewählt, um deren Wirksamkeit auf einem gegebenen Datensatz zu überprüfen:

K-Means

Erstmals von MacQueen (1967) formuliert ist der K-Means Algorithmus [Mac67] einer der am weitest verbreiteten Clustering-Algorithmen. K-Means ist ein partitionierender Algorithmus, was bedeutet, dass er alle vorhandenen Instanzen einem Cluster zuteilt. Der K-Means Algorithmus initialisiert zunächst k zufällig gewählte Punkte als Clusterzentren, sogenannte Zentroide. Alle Instanzen werden dem nächstgelegenen Zentroid zugeordnet und mithilfe der zugeteilten Instanzen eine neue Position der Zentroide berechnet. Dieser Vorgang wird so lange wiederholt, bis ein

lokales Optimum gefunden wurde, das heißt die Zuteilung der Instanzen zu den Zentroiden sich nicht mehr verändert. Da die Initialisierung der Zentroide im ersten Schritt zufällig ist, kann der K-Means Algorithmus bei mehrmaliger Ausführung jeweils ein neues lokales Optimum finden und ein unterschiedliches Clustering-Ergebnis herausgeben. Daher ist es sinnvoll, den K-Means Algorithmus mehrmals auf denselben Daten auszuführen, um die Wahrscheinlichkeit eines globalen Optimums als Ergebnis zu erhöhen.

Der K-Means Algorithmus bringt in seiner Grundform einige Probleme bezüglich komplexer Datenstrukturen mit sich. Am relevantesten für die in dieser Arbeit betrachteten Mehrklassenprobleme sind:

1. Die Anzahl der Cluster muss als Eingabe vom Nutzer mitgegeben werden. Es muss also im Voraus klar sein, in wie viele Cluster der Datensatz eingeteilt werden soll.
2. K-Means teilt den Datensatz in möglichst gleich große Cluster ein. Dieses Problem wird auch „Uniform Effekt“ genannt und wird im Genauen in Abschnitt 2.2.1 erklärt.

Aus diesem Grund existieren diverse Erweiterungen des K-Means Algorithmus, welche das Clustering-Ergebnis optimieren: K-Median ist eine solche Erweiterung [Sar90]. Hier wird zur Berechnung der Zentroide der Median, anstatt des Mittelwerts aller zugehörigen Instanzen, genutzt. Außerdem erfolgt die Distanzberechnung nicht mithilfe der euklidischen Distanz, sondern der Manhattan Distanz. Bei K-Means++ [AV06] ist die initiale Wahl der Zentroide nicht zufällig, sondern gemäß einer Vorschrift. Diese Vorschrift ist so gewählt, dass die Zentroide anfangs möglichst weit auseinanderliegen, damit der K-Means nicht bereits nach kurzer Zeit bei einem schlechten lokalen Optimum terminiert.

X-Means

Eine Erweiterung des K-Means Algorithmus ist X-Means [PM02]. X-Means macht sich die grundsätzliche Funktionsweise von K-Means zu Nutze und führt ihn mehrmals auf verschiedenen Teilen des Datensatzes aus. Nach einer Einteilung des Datensatzes in k Cluster mittels K-Means wird jedes Cluster nochmals in zwei Subcluster unterteilt. Anschließend wird geprüft, ob dies das Clustering-Ergebnis verbessert. Die Einteilung in Subcluster wird rekursiv so lange fortgesetzt, bis ein Optimum oder ein festgelegtes Maximum an Clustern erreicht wird. Die genaue Funktionsweise des Algorithmus wird in Abschnitt 2.2.3 erklärt. Grundsätzlich können durch den X-Means Algorithmus sowohl das Problem der festen Clusteranzahl, als auch der Uniform Effekt behandelt werden.

Affinity Propagation

Affinity Propagation [FD07] ist eine Clustering-Methode, bei der Nachrichten zwischen allen Paaren von Datenpunkten ausgetauscht werden. Mithilfe der ausgetauschten Daten werden bestimmte Instanzen aus dem Datensatz als Clusterzentren gewählt und alle anderen Instanzen den jeweiligen Zentren zugewiesen. Ein großer Vorteil von Affinity Propagation ist, dass der Algorithmus die Anzahl der Cluster selbständig ermittelt. Demnach ist es nicht nötig, die mögliche Clusteranzahl

im Vorhinein abzuschätzen. Da jedoch paarweise zwischen allen Instanzen Informationen zu allen Features ausgetauscht werden, kann es bei großen Datenmengen zu einer vergleichsweise hohen Laufzeit kommen.

Mean-Shift

Mean-Shift [FH75] ist ein weiterer zentroid-basierter Clustering-Algorithmus. Anders als K-Means benutzt Mean-Shift die lokalen Maxima einer Dichtefunktion zum Bestimmen der Zentroide. Jeder Datenpunkt wird nun dem nächsten Zentroid zugeteilt.

DBSCAN und OPTICS

Der „Density-Based Spatial Clustering of Applications with Noise“ (DBSCAN) [EKSX96] ist ein dichtebasierter Algorithmus. Regionen, welche eine hohe Dichte an Datenpunkten aufweisen, werden als ein Cluster zusammengefasst. Dabei wird der Schwellwert der Dichte vom Nutzer festgelegt. DBSCAN ist kein partitionierender Algorithmus. Dies bedeutet, dass er Datenpunkte in Regionen mit geringer Dichte als Ausreißer erkennt und keinem Cluster zuweist. DBSCAN ermittelt die Clusteranzahl ebenfalls automatisch. Zusätzlich kann er, anders als zentroid-basierte Clustering-Algorithmen wie K-Means, Cluster unterschiedlicher Formen erkennen.

„Ordering Points To Identify the Clustering Structure“ (OPTICS) [ABKS99] ist eine Erweiterung des DBSCAN Algorithmus. Er adressiert dabei das größte Problem, welches der DBSCAN Algorithmus mit sich bringt: DBSCAN setzt voraus, dass alle Cluster des Datensatzes eine etwa gleich große Dichte aufweisen. Variiert die Dichte hingegen für jedes Cluster, so hat DBSCAN Probleme, diese korrekt zu erkennen. OPTICS schafft es hingegen, auch Cluster mit unterschiedlicher Dichte korrekt zu identifizieren.

Sowohl DBSCAN als auch OPTICS hängen von zwei Parametern ab: ϵ beschreibt den maximalen Abstand zweier Datenpunkte, welche bis zu diesem Abstand als „nah beieinander“ gelten, sowie *MinPts*, was die minimale Anzahl an „nah beieinanderliegenden“ Instanzen angibt, bis sie als Cluster gelten. Auch wenn die kleine Anzahl an zu konfigurierenden Parametern von Vorteil ist, kann sich die Wahl eines geeigneten ϵ-Parameters bei komplexen Daten als schwieriger Prozess herausstellen.

Hierarchisches Clustering

Hierarchisches Clustering ist eine Methode, bei welcher in Teilschritten eine Cluster-Hierarchie aus dem Datensatz aufgebaut wird. Grundsätzlich wird beim hierarchischen Clustering zwischen zwei Verfahren unterschieden: Beim **agglomerativen** Clusterverfahren gilt zunächst jeder Datenpunkt als eigenes Cluster. Schrittweise werden dann Cluster mit geringer Distanz zueinander zusammengefügt ("Bottom-Up"). Das **divisive** Verfahren hingegen funktioniert genau umgekehrt. Hier wird zunächst der gesamte Datensatz als ein Cluster betrachtet, welches schrittweise in kleinere Cluster aufgeteilt wird.

Zu den beiden Clusterverfahren gibt es beim hierarchischen Clustering verschiedene Distanzmaße, um die Abstände der einzelnen Cluster zu ermitteln. Diese ermöglichen es, die Kriterien zum Zusammenfassen beziehungsweise Aufteilen der Cluster je nach Verfahren zu variieren.

Wie beim K-Means Algorithmus muss auch bei hierarchischem Clustering die Clusteranzahl vor der Ausführung bekannt sein.

3.2.2. Parameter

Eine große Herausforderung bei der Konfiguration der Clustering-Algorithmen ist die geeignete Wahl der Parameter der jeweiligen Clustering-Algorithmen. Viele der Algorithmen liefern bei falschem oder ungenauem Setzen ihrer Parameter keine sinnvollen Clustering-Ergebnisse. Deshalb gilt es durch diverse Versuche die geeigneten Parameter für jeden Clustering-Algorithmus zu finden. Um diesen Prozess zu automatisieren und zu optimieren wurde das AutoML4Clust Framework [TFS21] zu Hilfe genommen.

Der vermutlich wichtigste Parameter bei einigen Clustering-Algorithmen ist die Anzahl der Cluster. Diese muss bei einigen Algorithmen, wie etwa K-Means oder hierarchischem Clustering, manuell eingestellt werden und wird nicht automatisch ermittelt. Die Clusteranzahl hat einen großen Einfluss auf die Anzahl der Instanzen pro Cluster, als auch auf die Positionen der einzelnen Cluster und Clusterzentren.

3.2.3. Clustering-Metriken

Um geeignete Parameter für die Clustering-Algorithmen zu finden, muss definiert werden, welche Clustering-Ergebnisse als geeignet oder effektiv eingestuft werden können. Unterschiedliche Metriken zur Interpretation von Clustering-Ergebnissen wurden in den letzten Jahrzenten entwickelt. Einige davon können verwendet werden, um die Genauigkeit der Clustering-Ergebnisse auf den genutzten Datensätzen zu validieren.

Silhouetten Koeffizient

Der Silhouetten Koeffizient [Rou87] ist ein Maß für die Separation zwischen den Clustern. Bei der Berechnung des Koeffizienten wird für jede Instanz eines Datensatzes zunächst die Silhouette errechnet. Diese ergibt sich aus dem durchschnittlichen Abstand zu allen Punkten innerhalb des Clusters, sowie dem durchschnittliche Abstand zu allen anderen Clustern, welche miteinander verrechnet werden. Der Silhouetten Koeffizient erschließt sich dann als Mittelwert der Silhouetten aller Instanzen. Der Wert des Koeffizienten befindet sich im Intervall $[-1, 1]$. Je höher der Koeffizient, desto besser sind die einzelnen Cluster separiert und desto kompakter ist das Clustering-Ergebnis.

Davies-Bouldin Index

Der Davies-Bouldin Index [DB79] misst Ähnlichkeiten R_{ij} zwischen zwei Clustern C_i und C_j. Die Ähnlichkeit setzt sich zusammen aus dem Streuungsmaß, welches die Streuung der Instanzen eines Clusters zu seinem Zentrum angibt, sowie dem Unähnlichkeitsmaß, das den Abstand der zwei Clusterzentren angibt. Je ähnlicher zwei Cluster C_i und C_j sind, desto höher ist der Ähnlichkeitswert R_{ij}. Dementsprechend nimmt der Davies-Bouldin Index zu jedem Cluster C_i das Maximum aller R_{ij} und errechnet daraus einen Durchschnittswert. Besitzt ein Clustering-Ergebnis einen hohen Davies-Bouldin Index, so bedeutet dies, dass sich die Cluster nicht deutlich voneinander unterscheiden. Somit spiegelt ein niedriger Davies-Bouldin Index ein besseres Clustering-Ergebnis wider.

Calinski-Harabasz Index

Eine weitere Validierungsmetrik ist der Calinski-Harabasz Index [CH74]. Dieser Index setzt sich als Quotient aus Separierung und Kompaktheit zusammen. Ein hoher Calinski-Harabasz Index bedeutet dichte Cluster und eine gute Trennung zwischen den Clustern.

Gini Koeffizient

Der Gini Koeffizient (Gini-Index) kann genutzt werden, um die Ungleichverteilung von Klassenlabel innerhalb eines Datensatzes zu messen. Er nimmt einen Wert im Bereich $[0, 1]$ an. Der Wert 0 steht dabei für die völlige Gleichverteilung aller vorhandenen Klassenlabel, während der Wert 1 bedeutet, dass alle Instanzen dasselbe Klassenlabel besitzen.

Der Gini Koeffizient ist eigentlich keine Metrik zur Validierung eines Clustering-Ergebnis. Dies liegt daran, dass für den Gini Koeffizient das Klassenlabel jeder Instanz bekannt sein muss und es demnach für unüberwachtes Lernen, wie Clustering, ungeeignet ist. Im Rahmen dieser Arbeit ist er dennoch ein sehr nützlicher Wert, das Clustering-Ergebnis des Trainings-Datensatzes zu überprüfen, da hier die Klassenlabel der Instanzen vorliegen. Um die Mulit-Class Imbalance im Konkreten zu adressieren, sollte die Ungleichverteilung der Daten in den einzelnen Clustern möglichst gering sein. Der Gini Koeffizient kann auf jedem Cluster gemessen werden, um daraus einen Mittelwert zu bilden. Für ein gutes Clustering-Ergebnis soll der Gini Koeffizient möglichst gering ausfallen.

3.3. Klassifikations-Phase

Mit den ausgewählten Clustering-Algorithmen wird der Trainings-Datensatz in einzelne Partitionen unterteilt. Liegen die Instanzen aufgeteilt in ihren Partitionen vor, kann auf ihnen im nächsten Schritt jeweils ein Klassifikator trainiert werden. Auch beim Klassifizieren gibt es eine Vielzahl von möglichen Klassifikatoren, welche auf den einzelnen Partitionen trainert werden können. Die Auswahl geeigneter Klassifikatoren erfolgt unmittelbar aus den Ergebnissen von Hirsch et. al. [HRM19].

3.3.1. Ensemble-Learning Klassifikatoren

Klassifikatoren benutzen in der Regel einen Algorithmus, um das Klassenlabel einer Instanz vorherzusagen. Ensemble Methoden hingegen basieren auf der Idee, eine Menge unterschiedlicher Lernalgorithmen auf dem Datensatz zu trainieren. Die Ergebnisse der einzelnen Lerner werden im Kollektiv zusammengefasst und anhand der einzelnen Ergebnisse eine Gesamtentscheidung getroffen. So können falsche Vorhersagen einzelner Lerner durch die Entscheidung des Kollektivs überstimmt werden und es besteht eine höhere Wahrscheinlichkeit einer korrekten Klassifikation. Außerdem verringert es das Risiko des Overfittings. Diese Technik stellt sich als besonders gut bei kleineren Datensätzen mit vielen Klassenlabel und Label Noise heraus [Die00].

Ein Klassifikator, welcher auf Ensemble Learning basiert, ist der **Random Forest** Algorithmus [Bre01]. Dabei benutzt der Random Forest das Prinzip des *baggings*. Bei dieser Methode wird der Datensatz zufällig nach einzelnen Instanzen und Features aufgeteilt. Jede zufällige Aufteilung gilt dabei als Entscheidungsbaum, welcher mit den zugehörigen Instanzen und Features trainiert wird. Bei der Vorhersage neuer Instanzen wird die Klassifikation jedes einzelnen Entscheidungsbaumes betrachtet. Die Entscheidung des finalen Klassenlabel wird nach Mehreitsentscheid aller Entscheidungsbäume gefällt.

Eine weitere Ensemble Methode ist **AdaBoost** [FS97]. Anders als Random Forest benutzt AdaBoost die *boosting* Methode, bei welcher die Aufteilung des Datensatzes nicht zufällig abläuft. Vielmehr erfolgt die Aufteilung des Datensatzes iterativ. In jeder Iteration wird außerdem genau ein Lerner trainiert. In der ersten Iteration wird zunächst ein Teil des Datensatzes ausgewählt und ein Lerner darauf trainiert. Die nicht verwendeten Instanzen werden anschließend als Test-Datensatz verwendet und mithilfe des Lerners klassifiziert. Das Klassifikationsergebnis des Lerners wird in der nächsten Iteration genutzt, um auf einem neu aufgeteilten Datensatz einen neuen Lerner zu trainieren. Falsch klassifizierte Instanzen erhalten nun ein höheres Gewicht und werden in der nächsten Iteration stärker betrachtet, während korrekt klassifizierte Instanzen beiseite gelegt werden.

Sowohl Random Forest als auch AdaBoost erzielten in der Vorarbeit von Hirsch et. al. gute Ergebnisse bei der Klassifikation [HRM19]. Aus diesem Grund wurden die beiden Methoden bei der Klassifikation der einzelnen Partitionen ebenfalls betrachtet.

3.3.2. Klassifikationsmetriken

Auch für die Klassifikation existieren Metriken, mit denen das Ergebnis nach einer Vorhersage evaluiert werden kann. Für jede Instanz des Test-Datensatzes wird hier geprüft, ob das tatsächliche Klassenlabel mit dem vom Klassifikator vorhergesagten übereinstimmt. Das Ergebnis einer Klassifikation kann bei Zweiklassenproblemen (mit zwei Label „positiv“ und „negativ“) grundsätzlich in vier Kategorien eingeteilt werden.

1. **Wahr-Positiv:** Vorhersage des Klassifikators und tatsächliches Klassenlabel sind beide „positiv“.
2. **Wahr-Negativ:** Vorhersage des Klassifikators und tatsächliches Klassenlabel sind beide „negativ“.
3. **Falsch-Positiv:** Vorhersage des Klassifikators ist „positiv", tatsächliches Klassenlabel ist „negativ“.

4. **Falsch-Negativ:** Vorhersage des Klassifikators ist „negativ", tatsächliches Klassenlabel ist „positiv“.

Da in dieser Arbeit jedoch Mehrklassenprobleme betrachtet werden, wird diese Definition erweitert. Sie wird nun über alle Instanzen separat für jedes Klassenlabel i definiert.

1. **Wahr-Positiv$_i$:** Vorhersage des Klassifikators und tatsächliches Klassenlabel sind beide i.
2. **Wahr-Negativ$_i$:** Vorhersage des Klassifikators und tatsächliches Klassenlabel sind beide ungleich i.
3. **Falsch-Positiv$_i$:** Vorhersage des Klassifikators ist i, tatsächliches Klassenlabel ist ungleich i.
4. **Falsch-Negativ$_i$:** Vorhersage des Klassifikators ist ungleich i, tatsächliches Klassenlabel ist i.

Eine Instanz, welche vom Klassifikator das Label n erhält, jedoch eigentlich zur Klasse k gehört, ist dementsprechend eine Falsch-Positive bezüglich Klasse n und gleichzeitig eine Falsch-Negative bezüglich Klase k.

Mithilfe dieser Definition lassen sich nun die folgenden Metriken formulieren:

Accuracy

Der Accuracy-Wert gibt an, welcher Anteil der vorhergesagten Klassenlabel mit den tatsächlichen Label übereinstimmt. Er gibt also das Verhältnis aus Wahr-Positiv plus Wahr-Negativ zu allen Instanzen an:

Definition 3.3.1

$$Acc = \frac{Wahr\text{-}Positiv + Wahr\text{-}Negativ}{Alle\ Instanzen}$$

Aus der Formel lässt sich schließen, dass der Accuracy-Wert einen Wert zwischen 0 und 1 annehmen kann. Je höher der Accuracy-Wert, desto besser ist das Ergebnis der Klassifikation.

Precision, Recall und F1

Bei Datensätzen mit starker Klassenungleichverteilung weist der Accuracy-Wert seine Grenzen auf. Als Beispiel stelle man sich einen Datensatz vor, bei dem 99% der Daten einer Klasse A angehören und die restlichen 1% der Daten auf zwei Klassen B und C verteilt sind. Ein Klassifikator, welcher jede Instanz das Klassenlabel A vorhersagt, würde demnach in 99% der Fälle richtig liegen und einen Accuracy-Wert von 0.99 erhalten. In Wahrheit hat der Klassifikator jedoch alle Instanzen der Minority Klassen B und C nicht korrekt identifizieren können. In vielen Fällen ist eine korrekte Identifizierung dieser Instanzen genauso wichtig oder sogar viel bedeutungsvoller. Aus diesem Grund existieren die beiden Metriken *Precision* und *Recall*, welche ein aussagekräftigeres Bild bei Klassifikation mit ungleichverteilten Daten geben sollen. Precision und Recall sind folgendermaßen definiert:

Definition 3.3.2

$$Prec_i = \frac{Wahr\text{-}Positiv_i}{Wahr\text{-}Positiv_i + Falsch\text{-}Positiv_i}$$

Definition 3.3.3

$$Rec_i = \frac{Wahr\text{-}Positiv_i}{Wahr\text{-}Positiv_i + Falsch\text{-}Negativ_i}$$

Wie zu sehen, muss bei Mehrklassenproblemen Precision und Recall ebenfalls für jede Klasse separat definiert werden. Es existiert auch die Möglichkeit, die Instanzen aller Klassen zusammenzuzählen und den Anteil zu berechnen. Da jedoch, wie oben bereits erwähnt, zu jeder Falsch-Positiv Instanz eine zugehörige Falsch-Negativ Instanz existiert, würden in diesem Fall Precision und Recall den selben Wert zurückgeben. Aus diesem Grund wird für gewöhnlich für die Berechnung der Precision und Recall Metrik für jede Klasse einzeln ein Wert berechnet und ein Mittelwert über alle Ergebnisse ermittelt. Analog zur Accuracy können auch Precision und Recall einen Wert im Bereich $[0, 1]$ annehmen.

Um eine passende Balance zwischen Precision und Recall zu finden, kann die *F1* Metrik genutzt werden. Diese nutzt beide Werte, um daraus einen neuen Wert zwischen 0 und 1 zu berechnen.

Definition 3.3.4

$$F1_i = 2 \cdot \frac{Prec_i \cdot Rec_i}{Prec_i + Rec_i}$$

Intuititv lässt sich F1 als eine neue Accuracy-Metrik sehen, welche nun die Klassenungleichverteilung in Betracht zieht.

3.4. Fehlende Features

Das Problem der fehlenden Features ist eine der zentralen Herausforderungen, welche aus der Analyse des Datensatzes hervorgeht. Sowohl Clustering-Algorithmen als auch die Klassifikatoren können nicht mit fehlenden Features umgehen, das heißt sie gehen prinzipell davon aus, dass alle eingehenden Features mit einem sinnvollen Wert belegt sind. Aus diesem Grund müssen die fehlenden Features jeweils vor dem Clustering beziehungsweise der Klassifizierung behandelt werden. Dabei ist zu empfehlen, beim Clustering und der Klassifizierung unterschiedliche Verfahren anzuwenden und den Datensatz an das Clustering oder die Klassifikation angzupassen. Dadurch lassen sich die Ergebnisse des Clusterings beziehungsweise der Klassifikation verbessern und auftretende Probleme jeweils gezielt vermeiden. Im Folgenden werden drei Möglichkeiten vorgestellt, die fehlenden Features im Datensatz zu behandeln.

3.4.1. Auffüllen fehlender Features durch einen festen Wert

Die einfachste Methode, die fehlenden Features zu behandeln, ist, nicht vorhandene Werte durch den Wert ’0’ aufzufüllen. Trotz ihrer Einfachheit ist diese Methode dennoch äußerst effektiv. Der Wert ’0’ wirkt als Platzhalter für den nicht vorhandenen Wert, wodurch der Klassifikator weiterhin die Muster korrekt erkennen kann. Zwei Instanzen, die ursprünglich am selben Feature keinen Wert aufweisen, haben nun beide den Wert ’0’. Dadurch trägt das Feature nicht zur Abstandsberechnung bei, wodurch beispielsweise ein Clustering-Algorithmus sie mit höherer Wahrscheinlichkeit in dasselbe Cluster einteilt. Eine wichtige Voraussetzung für eine korrekte Funktionsweise der Methode ist jedoch, dass die ursprünglichen Daten nicht bereits viele Features mit dem Wert ’0’ aufweisen. Hier würden sich die eingefügten ’0’ Werte mit den bereits vorhandenen vermischen und es würden sich neue Muster ergeben, welche das Ergebnis verfälschen. In diesem Fall sollte ein anderer Wert, beispielsweise ein negativer Wert nahe ’0’, genutzt werden.

3.4.2. Imputation

Eine weitere Methode, die fehlenden Features zu adressieren, ist, anhand der existierenden Daten die fehlenden Werte abzuschätzen. Dabei gibt es verschiedene Imputations-Methoden, welche die Abschätzung fehlender Features durchführen. Die *KNN-Imputation* [TCS+01] betrachtet die Features der k-nächsten Nachbarn, um die fehlenden Features einer Instanz abzuschätzen. *Miss Forest* [SB12] ist eine weitere Methode, die die Abschätzung vornehmen kann. Dieser Algorithmus basiert auf dem Random Forest Ansatz. Anders als die KNN-Impuation besitzt Miss Forest eine höhere Laufzeitkomplexität, weshalb der Algorithmus schlechter mit größeren Datenmengen und Features skaliert.

Im Vergleich zum Ersetzen mit dem Wert ’0’ macht es bei der Imputation einen Unterschied, ob diese auf dem gesamten Datensatz oder auf den einzelnen Partitionen ausgeführt wird. Dies liegt daran, dass in den einzelnen Partitionen nur ein Teil der Instanzen vorliegen und demnach andere Werte für die fehlenden Features abgeschätzt werden. Dementsprechend gibt es mehrere Möglichkeiten, die Imputation anzuwenden. Die einfachste Umsetzung ist es natürlich, die Imputation auf dem gesamten Datensatz auszuführen. Gerade bei Miss Forest kommt es hier jedoch zu erhöhten Laufzeiten.

Eine weitere Variante sieht vor, das Clustering auf dem mit ’0’ aufgefüllten Daten durchzuführen und auf den einzelnen Partitionen die Imputation vorzunehmen. Die aufgefüllten Nullen müssten in diesem Fall natürlich nach dem Clustering wieder entfernt werden. Die Imputation der fehlenden Features auf den einzelnen Partitionen hat zum Vorteil, dass die Instanzen nach dem Clustering nach ihrer Ähnlichkeit gruppiert sind und die Imputation nicht durch irrelevante Instanzen aus anderen Clustern verfälscht wird. Ist das zugrundeliegende Clustering-Ergebnis jedoch ungeeignet, so werden die „Fehler“ des Clusterings durch die Imputation verstärkt, da in diesem Fall die fehlenden Features nur über die Instanzen aus dem ungeeigneten Cluster abgeschätzt werden.

3.4.3. Feature Selection

Eine dritte Möglichkeit zur Behandlung des Problems der fehlenden Features ist die Feature Selection. Im Gegensatz zu den anderen beiden vorgestellten Methoden werden hier die Features nicht durch Werte aufgefüllt, sondern gewisse Features ganz verworfen. Der Algorithmus sucht

dabei heraus, welche der Features am „unwichtigsten“ sind und sortiert diese aus den Daten heraus. Um das fehlende Feature Problem vollständig beheben zu können, müssten alle Features, die bei mindestens einer Instanz keinen Wert aufweisen, entfernt werden. Da dies jedoch in der Regel nicht nicht realisierbar/umsetzbar ist, kann Feature Selection mit einer der zuvor genannten Methoden kombiniert werden. Die KNN-Imputation beispielsweise besitzt die Funktion, Features ganz zu entfernen.

Auch bei der Feature Selection ist es von Vorteil, die Methode auf den einzelnen Partitionen durchzuführen, da in diesem Fall besser erkannt werden kann, welche der Features verworfen werden können. Dies führt für gewöhnlich dazu, dass auf jeder Partition unterschiedliche Features entfernt werden. Dieser Ansatz führt beim Klassifizieren neuer Instanzen zu einem Problem. Nun muss für jede Partition separat geprüft werden, welche der Features beim Trainieren entfernt wurden und die Features der neuen Instanz müssen dementsprechend angepasst werden. Folglich muss sich jede Partition merken, welche der Features entfernt wurden, was zu einem hohen Aufwand führen kann.

4. Evaluation

In diesem Kapitel werden die Ergebnisse der datengetriebenen Partitionierung und Klassifikation aus Kapitel 3 vorgestellt und evaluiert. Mehrere Versuchsreihen mit unterschiedlichen Clustering-Algorithmen und Parametern wurden durchgeführt und deren Ergebnisse mittels verschiedener Metriken ausgewertet und verglichen. Abschnitt 4.1 beschreibt den technischen Aufbau zur Durchführung der Messungen. In Abschnitt 4.2 finden sich die Ergebnisse und Auswertungen, welche direkt aus der Clustering-Phase hervorgehen. Ferner werden in Abschnitt 4.3 die Ergebnisse nach der Klassifikations-Phase aufgeführt und evaluiert.

4.1. Aufbau

Das in Abschnitt 3.1 beschriebene Vorgehen ist in Python 3.9 umgesetzt. Die verwendeten Clustering-Algorithmen und Klassifikatoren sowie die jeweiligen Metriken werden größtenteils mithilfe der Python Bibliothek Scikit-Learn[1] implementiert. Eine Implementierung des X-Means Algorithmus wurde von Kazuhisa Fujita[2] realisiert und wird im Rahmen der Arbeit genutzt.

Ausgeführt wurden die Implementierungen auf einer Linux Virtual Machine mit Ubuntu 16.04, 32GB RAM und 6 VCPUs. Für Messungen auf einfachen Datensätzen mit weniger Durchläufen wurde außerdem ein Laptop verwendet. Dieser läuft auf einem Windows 10 Betriebssystem, einer Intel(R) Core(TM)i7-6700HQ CPU @2.6GHz und einem 8GB Arbeitsspeicher.

4.1.1. Datensätze

Eine Reihe verschiedener Datensätze werden verwendet, um die Vorhersagegenauigkeit der Klassifikatoren zu testen. Diese decken eine Vielzahl an Charakteristika ab und reichen von einfachen Datensätzen mit wenigen Klassen (9) und Features (10) zum Testen der groben Funktionsweise, bis hin zu den in Abschnitt 2.4.1 beschriebenen komplexen Strukturen aus dem Qualitätsmanagement.

Einfacher Datensatz

Um die Korrektheit des experimentellen Aufbaus und generelle Funktionen der Algorithmen zu testen wird zunächst ein synthetisch generierter Datensatz aus Scikit-Learn verwendet. Dieser weist weniger komplexe Strukturen auf. Die Klassifikatoren erzielen dementsprechend bessere Resultate hinsichtlich der Evaluationsmetriken.

[1] https://scikit-learn.org/stable/index.html
[2] https://github.com/KazuhisaFujita/X-means

- Scikit-Learn make_classification() Methode
- 9 Klassen
- 10 Featues pro Klasse
- 1000 Instanzen
- Gini-Index: 0.634
- 2 Cluster pro Klasse

Imbalance Generator

Neben dem einfachen Datensatz wird der Imbalance Generator (entwickelt von Dennis Tschechlov[3]) verwendet. Dieser ist an den Datensatz aus den Arbeiten von Hirsch et. al. angelehnt [HRM20] und modelliert die Produkthierarchie des Domänenwissens, sowie die Charakteristika der verwendeten Daten. Dementsprechend weist er die in Abschnitt 2.4.1 beschriebenen komplexen Strukturen auf.

Da der Imbalance Generator nicht in jedem Durchlauf die exakt selben Daten generiert, können die Ergebnisse der datengetriebenen Partitionierung je nach generiertem Datensatz unterschiedlich ausfallen. Außerdem lässt sich der Grad der Klassenungleichverteilung in fünf verschiedenen Stufen einstellen. Um all diese Eigenschaften zu berücksichtigen, werden insgesamt sechs verschiedene, vom Imbalance Generator erstellte Datensätze, verwendet.

Alle genutzten Datensätze aus dem Imbalance Generator besitzen grundsätzlich folgende Gemeinsamkeiten:

- 1050 Instanzen (750 für den Trainingssatz und 300 für den Testsatz)
- 84 Klassen
- 100 Features pro Klasse

Der Imbalance Generator besitzt insgesamt fünf Stufen für die Grade der Ungleichverteilung. Der Standardwert ist dabei „normal“. Zwei Datensätze mit dieser Einstellung werden in den Versuchen verwendet. Die Ergebnisse der zwei Datensätze werden direkt gegenübergestellt, um Unterschiede der Messergebnisse innerhalb derselben Konfiguration zu verdeutlichen. Des Weiteren wird für jede weitere Stufe der Klassenungleichverteilung jeweils ein Datensatz generiert:

- „very low“: Gini Index - 0.325
- „low“: Gini Index - 0.42
- „normal“: Gini Index - 0.51
- „high“: Gini Index - 0.54
- „very high“: Gini Index - 0.57

[3] https://gitlab-as.informatik.uni-stuttgart.de/tschecds/imbalancedatagenerator

Clustering-Algorithmus	Konfiguration
K-Means	2-10 Cluster; 20,30, ..., 100 Cluster
X-Means	10-30 Cluster
Hierarchisch	Linkage="complete"; 2-10 Cluster; 20,30,..., 100 Cluster
Affinity Propagation	maxIteration=200; convergenceIteration=15
DBSCAN	Epsilon= 0.1, 0.2 ...., 1.0 ; minSamples= 2-10
OPTICS	Epsilon= 0.1, 0.2, ..., 1.0; minSamples= 2-10
Mean-Shift	Bandbreite= 6.85

Tabelle 4.1.: Genutzte Clustering-Algorithmen und ihre Konfigurationen

4.2. Ergebnisse der Clustering-Phase

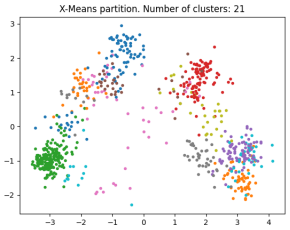
Für die Clustering-Phase werden die in Abschnitt 3.2.1 beschriebenen sieben Clustering-Algorithmen in Betracht gezogen (siehe Tabelle 4.1). Mithilfe der Clustering-Metriken aus Abschnitt 3.2.3 soll identifiziert werden, welche der Algorithmen gute Clustering-Ergebnisse liefert, damit sich im nächsten Schritt auf eine engere Auswahl beschränkt werden kann.

Nach erstem Testen aller Algorithmen auf den Datensätzen des Imbalance Generators stellt sich heraus, dass Mean-Shift, DBSCAN und OPTICS keine sinnvollen Clustering-Ergebnisse liefern. Dabei bedeutet „nicht sinvoll“, dass diese Algorithmen es mit den gewählten Konfigurationen nicht schaffen, den Datensatz in mehrere Cluster zu unterteilen. Mean-Shift teilt den gesamten Datensatz in genau ein Cluster ein, während der DBSCAN jeden Datenpunkt als Ausreißer erkennt und keinem Cluster zuteilt. Ein ähnliches Phänomen lässt sich auch beim OPTICS Algorithmus beobachten. Es ist davon auszugehen, dass bei diesen Algorithmen nicht die richtigen Parameter gefunden wurden, mit welchen die Algorithmen ein sinnvolles Ergebnis liefern.

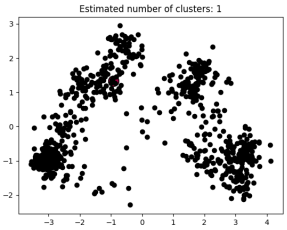
Das AutoML4Clust Framework [TFS21] wird zur Unterstützung bei der Suche geeigneter Parameter für die drei Algorithmen herangezogen. Bei der Ausführung wird der SMAC-Optimierer mit einem Budget (l) von 100 verwendet. Als interne Metrik zur Evaluation werden für den Optimierer sowohl der Silhouetten Koeffizient, als auch der Calinski-Harabasz Index ausprobiert. Jedoch lassen sich bei allen Durchläufen keine geeigneten Parameter für die drei genannten Algorithmen finden. Aus diesem Grund werden für weitere Untersuchungen ein Fokus auf die Algorithmen K-Means, X-Means, hierarchisches Clustering und Affinity Propagation gelegt. K-Means und hierarchisches Clustering schaffen es, den Datensatz korrekt in die mitgegebene Anzahl von zehn Clustern einzuteilen. X-Means teilt den Datensatz in 20 bis 22 Cluster ein und Affinity Propagation je nach Datensatz in 27 bis 32.

4.2.1. Evaluation der Clustering-Metriken

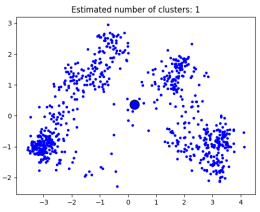
Die Evaluation der Clustering-Metriken wurde auf einem Datensatz mit Klassenungleichverteilung „normal “ durchgeführt. Fehlende Werte wurden mit ’0’ aufgefüllt.



(a) X-Means

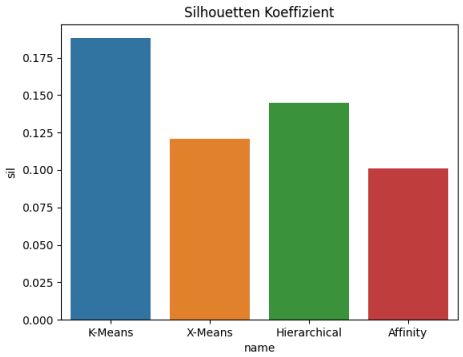


(b) DBSCAN

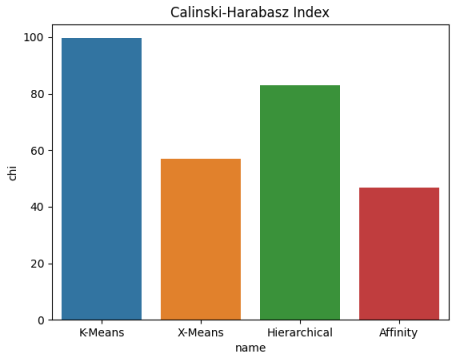


(c) Mean-Shift

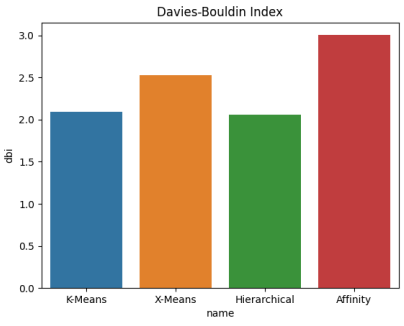
Abbildung 4.1.: Visualisierung der Clustering-Ergebnisse von drei Algorithmen. DBSCAN und Mean-Shift liefern keine sinnvollen Ergebnisse.



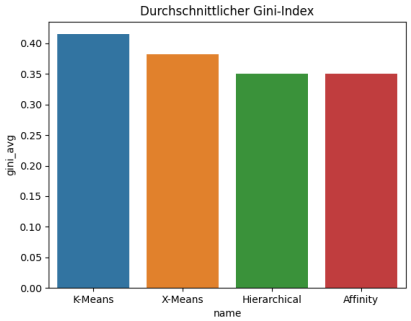
(a) Silhouetten Koeffizient



(b) Calinski-Harabasz Index



(c) Davies-Bouldin Index (niedriger Wert bedeutet gutes Clustering-Ergebnis)



(d) Gini Index

Abbildung 4.2.: Clustering-Metriken der einzelnen Algorithmen.

	K-Means	Zufällig
Accuracy	0.343	0.186

Tabelle 4.2.: Accuracy-Werte der Klassifikation nach K-Means und zufälliger Partitionierung.

Der K-Means Algorithmus erzielt generell die besten Ergebnisse bei den Clustering-Metriken. Lediglich beim Gini-Index schneidet er vergleichsweise schlecht ab mit einer hohen Ungleichverteilung von mehr als 0.4.

Genau umgekehrt verhält es sich beim Affinity-Propagation Algorithmus. Dieser liefert bei allen Clustering-Metriken das schlechteste Ergebnis im Vergleich zu den anderen Clustering-Algorithmen. Dies deutet darauf hin, dass er keine guten Partitionierung für die anschließende Klassifikation liefert. Als Außnahme gilt bei Affinity Propagation der Gini-Index, wo mit 0.35 der beste Wert erzielt wird. Dies lässt sich jedoch damit begründen, dass der Affinity Propagation den Datensatz in **bis zu dreimal so viele Cluster** unterteilt wie etwa K-Means oder hierarchisches Clustering. Eine Einteilung in viele kleine Cluster adressiert den in Abschnitt 2.2.1 genanten „Uniform Effekt“. Diese Einteilung behandelt indirekt auch die Klassenungleichverteilung, da Instanzen gleicher Ausprägung, wie sie in einem Cluster vorzufinden sind, generell die selben Klassenlabel besitzen. Jedoch ist zu erkennen, dass eine Einteilung in viele kleine Cluster nicht zwingend ein gutes Clustering-Ergebnis hervorbringt. In Betrachtung der anderen Clustering-Metriken ist bei Affinity Propagation das Gegenteil der Fall. Außerdem sorgt eine Einteilung in viele kleine Cluster mit wenigen Instanzen zu einem erhöhten Risiko des Overfittings in der Klassifikation.

X-Means und hierarchisches Clustering schneiden im Mittelfeld ab. X-Means erreicht jedoch meist schlechtere Werte, welche im Bereich derer von Affinity Propagation liegen.

4.3. Ergebnisse der Klassifikations-Phase

In diesem Abschnit werden die Ergebnisse der Klassifikation auf den einzelnen Partitionen beschrieben. Hierfür werden die in Abschnitt 3.3.2 Klassifikationsmetriken zur Evaluation verwendet.

Die Ergebnisse der Klassifikation werden primär genutzt, um die Partitionierung der Clustering-Algorithmen zu evaluieren. Um zu verdeutlichen, dass das Clustering-Ergebnis einen direkten Einfluss auf die Klassifikation hat, zeigt Tabelle 4.2 einen Vergleich der Accuracy-Werte nach der Klassifikation bei Partitionierung durch K-Means und zufälliger Partitionierung. Bei der zufälligen Partitionierung wird jede Instanz per Zufall in eine von zehn Clustern eingeteilt. Das Ergebnis zeigt, dass der Accuracy-Wert nach der Partitionierung durch K-Means deutlich höher ausfällt als bei der zufälligen Partitionierung. Eine geeignete Einteilung in die Partitionen hat demnach einen signifikanten Einfluss auf das Ergebnis der Klassifikatoren.

Zunächst wird der einfache Datensatz aus Scikit-Learn sowie Datensätze des Imbalance Generators mit Klassenungleichverteilung „normal“ betrachtet. Die vollständigen Messergebnisse des einfachen Datensatzes befinden sich in Tabelle A.1. Als Baseline dienen die Klassifikationsergebnisse ohne Partitionierung. Diese werden mit den Ergebnissen nach der Partitionierung durch die Clustering-Algorithmen verglichen. Zur Klassifikation werden die beiden Algorithmen AdaBoost und Random Forest verwendet.

	Random Forest	AdaBoost
Baseline	0.327	0.047
K-Means Mittelw.	0.315	0.234
K-Means Max.	0.340	0.260
K-Means Min.	0.297	0.207
X-Means Mittelw.	0.315	0.250
X-Means Max.	0.343	0.307
X-Means Min.	0.283	0.203
Hierarchisch	0.307	0.233
Affinity	0.323	0.240

Tabelle 4.3.: Partitionierung mit Random Forest und AdaBoost als Klassifikationen

Zwei Erkenntnisse lassen sich aus den ersten Durchläufen bereits herauskristallisieren:

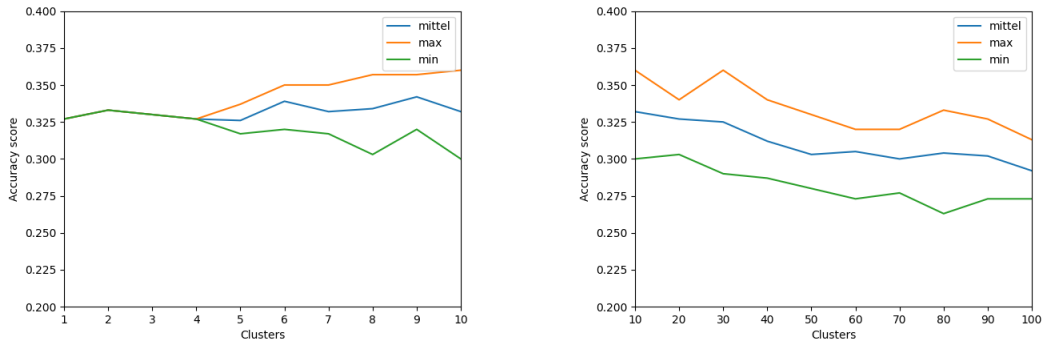
1. Der Random Forest Algorithmus liefert deutlich bessere Werte nach der Klassifikation als AdaBoost (Tabelle 4.3). Während die Accuracy-Werte bei Random Forest in der Regel bei über 30% liegen, kommt AdaBoost in den meisten Fällen nicht an die 25% Grenze heran. Da der Vergleich der Clustering-Algorithmen für diese Arbeit relevant ist, und nicht der Vergleich der unterschiedlichen Klassifikatoren, wurden weitere Untersuchungen mit AdaBoost verworfen. Für alle weiteren Ergebnisse gelten die Ergebnisse des Random Forest Algorithmus auf dem gesamten Datensatz als Baseline.
2. Die Metriken Accuracy, Precision, Recall und F1 verhalten sich bei gleicher Ungleichverteilung analog zueinander. Liefert beispielsweise der K-Means Algorithmus den besten Accuracy-Wert, liegt K-Means auch bezüglich der anderen Metriken meist an erster Stelle. Als einzige Ausnahme gilt hier der X-Means Algorithmus. Dieser liefert auch dann gute Werte bezüglich Precision, Recall und F1 wenn er in der Accuracy nicht am besten abschneidet. Da dieser Effekt jedoch bei keinem weiteren Algorithmus auftritt, werden im weiteren Verlauf auf den Datensätzen mit Ungleichverteilung „normal“ ausschließlich die Accuracy-Werte verglichen. Erst bei variierender Klassenungleichverteilung werden auch die anderen Klassifikationsmetriken wieder in Betracht gezogen, da gerade für hohe Klassenungleichverteilung die Accuracy nicht zwingend ein aussagekräftigeres Ergebnis zur Qualität der Klassifikation liefert.

4.3.1. Wahl der Clusteranzahl

Für die Algorithmen K-Means und hierarchisches Clustering muss vor Ausführung die Clusteranzahl mitgegeben werden. Um die geeignete Anzahl an Clustern zu erhalten, wurde hierfür eine Testreihe mit variierender Clusteranzahl durchgeführt. Hierbei lässt sich beobachten, wie sich die Algorithmen bei einer Partitionierung in wenige Cluster verhalten (2-10 Cluster), sowie die Auswirkung auf die Messergebnisse bei einer höheren Anzahl an Clustern (10-100 Cluster). Für jede Clusteranzahl wird die Partitionierung zehn Mal durchgeführt und ein Mittelwert gebildet. Die Ergebnisse der Testreihe werden in Abbildung 4.3 für den K-Means Algorithmus aufgeführt.

	acc	prec	rec	f1
RF Baseline	0.327	**0.188**	0.171	0.151
K-Means Mittelw.	0.332	0.145	0.170	0.135
K-Means Max.	**0.360**	0.167	**0.192**	**0.155**
K-Means Min.	0.300	0.120	0.139	0.110
X-Means Mittelw.	0.331	0.143	0.170	0.132
X-Means Max.	**0.357**	**0.168**	**0.187**	**0.154**
X-Means Min.	0.297	0.100	0.142	0.102
Hierarchisch	0.323	0.131	0.161	0.128
Affinity	0.297	0.112	0.145	0.110

Tabelle 4.4.: Accuracy, Precision, Recall und F1 Werte nach Klassifikation mit Random Forest. Die zwei besten Werte jeweils hervorgehoben.



(a) K-Means Clustering 1-10 Cluster

(b) K-Means Clustering 10-100 Cluster

Abbildung 4.3.: K-Means: Verhalten der Accuracy-Werte bei Änderung der Clusteranzahl.

Bei einer kleinen Clusteranzahl von 2-10 (Abbildung 4.3a) erreicht K-Means im Mittelwert in allen Fällen eine ähnliche Accuracy im Bereich von 33%. Aus diesem Grund werden zusätzlich noch das erreichte Minimum und Maximum ermittelt, um die Variationen genauer zu betrachten. Zu beobachten ist, dass die Min- und Max-Werte bei einer sehr kleinen Anzahl an Clustern identisch sind. Eine mögliche Erklärung dafür ist, dass K-Means bei einer kleinen Clusteranzahl immer zum selben Optimum konvergiert und das Clustering-Ergebnis sich somit nicht verändert. Bei steigender Clusteranzahl werden größere Variationen der Accuracy-Werte erreicht, somit im Maximum auch höhere Werte bis zu 36%.

Bei Betrachtung der Clusteranzahl von 10-100 (Abbildung 4.3b) ist zu erkennen, dass die Accuracy mit steigender Clusteranzahl beinahe stetig abnimmt. Im Maximum können vereinzelt noch höhere Werte erzielt werden, tendenziell verhalten sich hier jedoch alle drei Kurven analog. Der optimale Bereich für eine geeignete Partitionierung liegt demnach bei etwa 10 Clustern. Aus diesem Grund wurde für K-Means die Clusteranzahl 10 ausgewählt.

Bei hierachischem Clustering werden ähnliche Tendenzen wie bei K-means festgestellt. Da es sich bei Hierachischem Clustering um ein deterministisches Verfahren handelt wird hier jedoch kein Mittelwert aus 10 Ausführungen gebildet. Somit wird auch für hierarchisches Clustering die Clusteranzahl 10 bestimmt.

4.3.2. Einfluss des Zufalls auf die Ergebnisse

Sowohl der Random Forest Klassifikator als auch die beiden Clustering-Algorithmen K-Means und X-Means sind randomisierte Algorithmen und liefern unterschiedliche Ergebnisse in jeder Ausführung. Um trotzdem reproduzierbare Ergebnisse zu liefern, besitzen sie einen Random State Parameter. Beim Setzen dieses Parameters auf einen festen Wert, bleibt die Ausgabe des jeweiligen Algorithmus immer gleich. Je nach Wahl des Random State Parameters variieren die Accuracy-Werte nach der Klassifikation um bis zu 5%-Punkte. Die Auswahl des Random State Parameters hat demnach einen signifikanten Einfluss auf das Messergebnis. Wird er weggelassen, so ist die Initialisierung der Algorithmen zufällig. Dies kommt der zufälligen Wahl des Random State Parameters gleich. Um die stark variierenden Accuracy-Werte anzugehen wurden zwei Strategien entwickelt, nach denen die Ergebnisse evaluiert sollen:

- Random Forest erhält als Random State Parameter einen **festen Wert**: Dadurch liefern sowohl die Baseline (Random Forest), als auch Affinity Propagation und hierarchisches Clustering deterministische Werte (Affinity Propagation und Hierachisches Clustering sind deterministische Clustering-Verfahren). Da jeweils das beste Clustering-Ergebnis für eine geeignete Partitionierung gesucht wird, erhalten K-Means und X-Means keinen Random State. Bei diesen Algorithmen wird aus 100 Durchläufen jeweils das Maximum, Minimum und der Mittelwert hinsichtlich der Accuracy-Metrik betrachtet. Der Wert des Random State Parameters für den Random Forest Klassifikator wurde aus den Ergebnissen von Hirsch et. al. übernommen.

  Diese Einstellung ist nützlich, sofern lediglich die Auswirkung des Clusterings auf die Accuracy betrachtet werden soll.

- Der Random Forest erhält **keinen Random State Parameter**. In diesem Fall werden für alle Algorithmen aus 100 Durchläufen jeweils Maximum, Minimum und Mittelwert in Betracht gezogen.

  Durch diese Einstellung lässt sich das bestmögliche Ergebnis sowie der erwartete Mittelwert besser abschätzen.

Tabelle 4.5 zeigt die Accuracy-Werte nach der Klassifikation mit und ohne den Random State Parameter. Bereits bei der Baseline ist zu erkennen, dass der Random Forest ohne Random State Parameter im Maximum eine deutlich höhere Accuracy erreichen kann. Ohne Random State schaffen K-Means, X-Means und hierarchisches Clustering eine Accuracy von zwischen 35% und 36%. Jedoch erreicht diesen Wert auch der Random Forest ohne Partitionierung. Eine deutliche Verbesserung der Accuracy durch die Partitionierung ist somit nicht zu erkennen. Bemerkenswert hingegen ist, dass die Accuracy von 36% von K-Means und X-Means auch dann erreicht werden kann, wenn der Random State auf einen festen Wert gesetzt ist. Daraus lässt sich schließen, dass eine vergleichbare Vorhersagegenauigkeit erreichbar ist, selbst wenn der zugrundeliegende Random Forest nicht durch randomisierte Einstellung das beste Klassifikationsergebnis herausgibt.

Kein Random State		**Random State = 1234**	
RF Baseline Mittelw.	0.332		
RF Baseline Max.	**0.357**	RF Baseline	0.327
RF Baseline Min.	0.303		
K-Means Mittelw.	0.337	K-Means Mittelw.	0.332
K-Means Max.	**0.360**	K-Means Max.	**0.360**
K-Means Min.	0.307	K-Means Min.	0.300
X-Means Mittelw.	0.324	X-Means Mittelw.	0.331
X-Means Max.	**0.353**	X-Means Max.	**0.357**
X-Means Min.	0.297	X-Means Min.	0.297
Hierarchisch Mittelw.	0.321		
Hierarchisch Max.	**0.347**	Hierarchisch	0.323
Hierarchisch Min.	0.297		
Affinity Mittelw.	0.299		
Affinity Max.	**0.320**	Affinity	0.297
Affinity Min.	0.277		

Tabelle 4.5.: Klassifikation mit Random Forest: Ohne Random State und mit festem Random State Parameter

Aus den Ergebnissen erschließt sich folgende Erkenntnis: Erhält der Random State einen festen Wert, ist es durch geeignetes Clustering mittels K-Means oder X-Means möglich, die Accuracy zu erhöhen. Wird der Random State weggelassen, ist eine Verbesserung bei jeder der Partionierungen gleichermaßen zu erkennen. In dem Fall ist die Initialisierung jedoch zufällig und das optimale Ergebnis muss für jede Partitionierung separat gefunden werden.

4.3.3. Einfluss der Daten auf die Ergebnisse

Einen weitereren großen Einfluss auf die Messergebnisse hat die Wahl des Datensatzes. Dies mag auf den ersten Blick offensichtlich klingen, in diesem Fall soll jedoch verdeutlicht werden, dass kleine Veränderungen innerhalb der Daten einen großen Einfluss auf die Ergebnisse mit sich bringen.

Wie bereits erwähnt ist die Datengenerierung durch den Imbalance Generator nichtdeterministisch. Dementsprechend ist klar, dass aus manchen Datensätzen des Generators bessere Accuracy-Ergebnisse hervorgehen als auf anderen. Zu erwarten ist dabei, dass sich die Änderung der Datensätze gleichermaßen auf die Performance aller Clustering-Algorithmen auswirkt. Interessanterweise verändern sich je nach Datensatz jedoch die Tendenzen innerhalb der Clustering-Algorithmen bezüglich ihrer Performance. Um dieses Phänomen zu verdeutlichen werden zwei Datensätze mit gleicher Konfiguration des Generators erstellt und die datengetriebene Partitionierung auf ihnen ausgeführt. Die Ergebnisse werden in Tabelle 4.6 aufgeführt:

	Datensatz 1	**Datensatz 2**
RF Baseline Mittelw.	0.332	0.348
RF Baseline Max.	0.357	**0.383**
RF Baseline Min.	0.303	0.317
K-Means Mittelw.	0.337	0.327
K-Means Max.	**0.360**	0.350
K-Means Min.	0.307	0.297
X-Means Mittelw.	0.324	0.320
X-Means Max.	**0.353**	0.350
X-Means Min.	0.297	0.283
Hierarchisch Mittelw.	0.321	0.323
Hierarchisch Max.	0.347	**0.363**
Hierarchisch Min.	0.297	0.290
Affinity Mittelw.	0.299	0.326
Affinity Max.	0.320	**0.350**
Affinity Min.	0.277	0.303

Tabelle 4.6.: Zwei unterschiedliche Datensätze des Imbalance Generators mit gleicher Konfiguration

Bereits der Baseline Random Forest erzielt bei Datensatz 2 eine um etwa 3%-Punkte höhere Accuracy als in Datensatz 1. Während K-Means und X-means in Datensatz 1 noch etwa so gute Ergebnisse liefern wie die Baseline, schneiden sie in Datensatz 2 schlechter ab. Dagegen können sowohl hierarchisches Clustering als auch Affinity Propagation in Datensatz 2 deutlich höhere Accuracy-Werte erzielen als in Datensatz 1.

Die zwei Datensätze aus Tabelle 4.6 zeigen repräsentativ, wie sich kleine Änderungen der Daten auf die Messergebnisse auswirken. Die zwei gewählten Datensätze wurden vom selben Generator erzeugt und weisen grundsätzlich ähnliche Merkmale auf. Trotzdem können bereits kleine Änderungen signifikante Auswirkungen auf die Performance der Clustering-Algorithmen mit sich ziehen. Die geeignete Partitionierung hängt dementsprechend stark von den zugrundeliegenden Daten ab.

4.3.4. KNN-/ und Miss Forest-Imputation

In Abschnitt 3.4 wurden mehrere Konzepte zur Bewältigung des Problems der fehlenden Features behandelt. Bei den bisherigen Ergebnissen wurden als Vergleichsbasis alle fehlenden Features durch den Wert ’0’ ersetzt. Nun wollen wird betrachtet, wie sich die Imputations-Methoden KNN-Imputation und Miss Forest auf die Ergebnisse auswirken. Für die KNN-Imputation werden die fünf nächsten Nachbarn zur Abschätzung fehlender Werte betrachtet. Wie in Abschnitt 3.4.2 beschrieben, wird die Imputation einmal auf dem gesamten Datensatz oder jeweils auf den einzelnen Partitionen durchgeführt. Sofern die Imputation auf den einzelnen Parititionen angewendet wird, wird für das Clustering der mit ’0’ aufgefüllte Datensatz verwendet.

	’0’ aufgefüllt	MissingForest	KNN-Imputation
RF Baseline	**0.327**	0.287	0.283
K-Means Mittelw.	0.332	0.300	0.290
K-Means Max.	**0.360**	0.330	0.333
K-Means Min.	0.300	0.280	0.260
X-Means Mittelw.	0.331	0.304	0.303
X-Means Max.	**0.357**	**0.340**	**0.340**
X-Means Min.	0.297	0.270	0.263
Hierarchisch	**0.323**	0.280	0.267
Affinity	**0.297**	0.297	0.220

Tabelle 4.7.: Ergebnisse nach Imputation auf gesamtem Datensatz: Mit ’0’ aufgefüllt, Miss Forest und KNN-Imputation

	’0’ aufgefüllt	MissingForest	KNN-Imputation
RF Baseline	0.327	0.287	0.283
K-Means Mittelw.	0.332	0.333	0.332
K-Means Max.	**0.360**	**0.357**	**0.360**
K-Means Min.	0.300	0.303	0.300
X-Means Mittelw.	0.331	0.329	0.331
X-Means Max.	**0.357**	**0.353**	**0.357**
X-Means Min.	0.297	0.300	0.297
Hierarchisch	0.323	0.323	0.323
Affinity	0.297	0.297	0.297

Tabelle 4.8.: Ergebnisse nach Imputation auf einzelnen Partitionen: Mit ’0’ aufgefüllt, Miss Forest und KNN-Imputation

Beim Betrachten der Ergebnisse der Imputation auf dem gesamten Datensatz in Tabelle 4.7 ist zu erkennen, dass sowohl der Miss Forest als auch die KNN-Imputation schlechtere Accuracy-Werte erzielen als das Auffüllen durch ’0’. Der Random Forest ohne Partitionierung verliert durch Imputation etwa vier Prozentpunkte an Accuracy. Mithilfe der Partitionierung der Clustering-Algorithmen verschlechtern sich die Accuracy-Werte durch die Imputation lediglich um etwa einen bis zwei Prozentpunkte. Jedoch kommen sie trotzdem nicht an die Accuracy-Werte des durch ’0’ aufgefüllten Datensatzes heran. Miss Forest liefert minimal bessere Ergebnisse als die KNN-Imputation. Eine mögliche Erklärung für das schlechtere Abschneiden der Impuation ist die Distanzberechnung zwischen Instanzen, wie sie innerhalb des Clusterings und der Klassifikation durchgeführt wird. Während der Wert ’0’ nicht weiter zur Distanzberechnung beiträgt, ergeben sich durch abgeschätzte Werte neue Abstände zwischen Datenpunkten. Dadurch erschließen sich neue Muster innerhalb des Datensatzes, welche in der ursprünglichen Form nicht existieren. Diese Muster werden von den Klassifikatoren erlernt und verzerren das Klassifikationsergebnis.

Die Imputation auf den einzelnen Partitionen liefert hingegen vergleichbare Accuracy-Werte zum mit ’0’ aufgefülltem Datensatz. Da bei der Baseline keine Partitionen vorliegen wird hier die jeweilige Impuation auf dem gesamten Datensatz ausgeführt. Nennenswert ist, dass der Miss Forest, KNN-Imputation und Auffüllen durch ’0’ meist die exakt selben Ergebnisse liefern. Daraus lässt sich folgende Vermutung erschließen: Alle drei Methoden besitzen die Gemeinsamkeit, dass die Clustering-Phase mithilfe des durch ’0’ aufgefüllten Datensatzes ausgeführt wird. Lediglich die Klassifikation auf den Partitionen unterscheidet sich. Da in den Messergebnissen jedoch kaum Unterschiede nach der Klassifikation zu erkennen sind, lässt sich mutmaßen, dass die Imputation auf den Partitionen keinen erkennbaren Einfluss mit sich bringt.

Zusammenfassend lässt sich Folgendes sagen: Auf dem gesamten Datensatz ausgeführt, bringt das Auffüllen fehlender Werte durch ’0’ bessere Ergebnisse hervor als der Miss Forest oder die KNN-Imputation. Wird die Impuation auf den einzelnen Partitionen ausgeführt, wiegt das Clustering-Ergebnis zu stark, als dass nennenswerte Änderungen festzustellen sind. Das Auffüllen fehlender Features durch den Wert ’0’ trägt folglich zu den besten Ergebnissen bei der datengetriebenen Partitionierung bei.

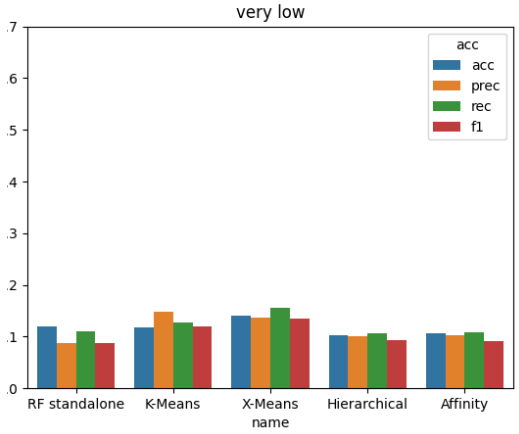
4.3.5. Einfluss der Klassenungleichverteilung

In diesem Abschnitt werden fünf Datensätze mit variierender Klassenungleichverteilung betrachtet. Alle fehlenden Features werden mit ’0’ aufgefüllt. Für die Klassifikation wird der Random Forest Algorithmus mit festem Random State verwendet.

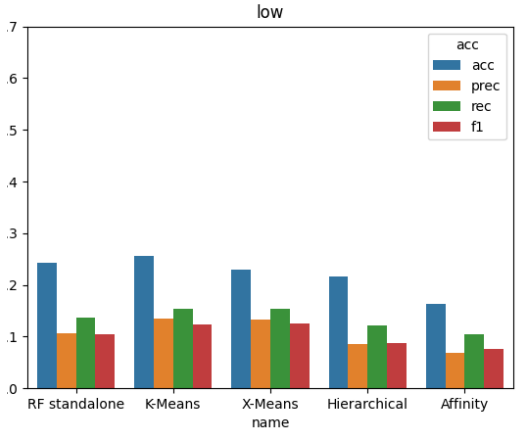
Steigt die Klassenungleichverteilung, so verbessern sich die Accuracy-Werte bei der Klassifikation. Dies liegt daran, dass bei starker Ungleichverteilung im Verhältnis mehr Majority Instanzen vorliegen und es dem Klassifiaktor leichter fällt, diese korrekt vorherzusagen. Da in diesem Fall jedoch eine hohe Accuracy nicht zwingend für jeden Anwendungsfall ein gutes Klassifikationsergebnis widerspiegelt, werden die Metriken Precision, Recall und F1 ebenfall betrachtet.

In Abbildung 4.4 ist der Unterschied von Accuracy und den anderen Metriken zur Klassifikation verdeutlicht Die exakten Zahlenwerte sind in Tabellen A.5 bis A.9 aufgetragen). Bei Klassenungleichverteilung „very low“ nehmen alle Metriken einen vergleichbaren Wert an. Bei höherer Klassenungleichverteilung steigen die Accuracy-Werte auf bis zu 60% an, während die anderen Metriken im Schnitt bei etwa 20% liegen. Dieses Phänomen ist sowohl bei der Baseline, als auch bei allen Clustering-Verfahren zu beobachten.

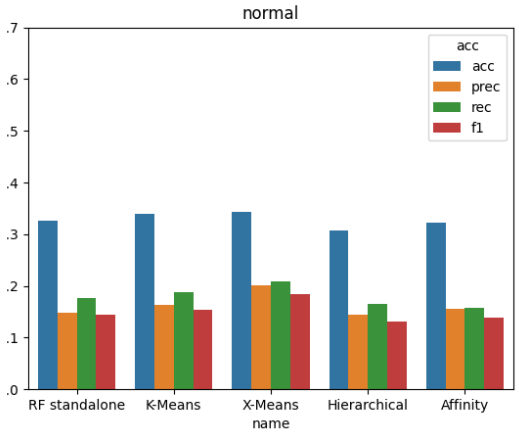
Sonst ergeben sich bei den unterschiedlichen Klassenungleichverteilungen ähnliche Bilder: Im Vergleich liefern K-Means und X-Means meist die besten Accuracy-Werte, jedoch sind diese meist nicht signifikant höher als durch Random Forest ohne Partitionierung. In manchen Fällen liegt auch das hierarchische Clustering in einem ähnlichen Bereich. Affinity Propagation hingegen erreicht keine hohen Werte. Was die Werte bezüglich Precision, Recall und F1 angeht, liefert X-Means die besten Ergebnisse. Diese liegen im Schnitt ein paar Prozentpunkte über dem Schnitt der anderen Algorithmen sowie der Baseline.



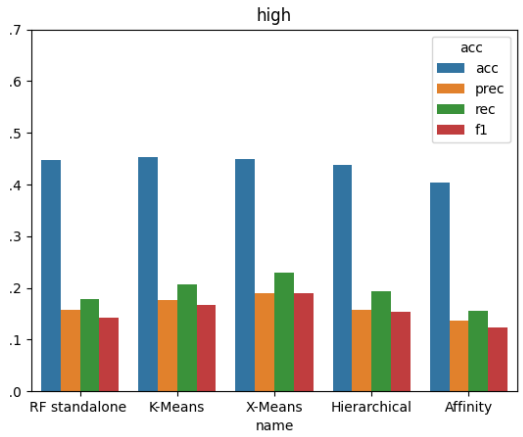
(a) Very Low



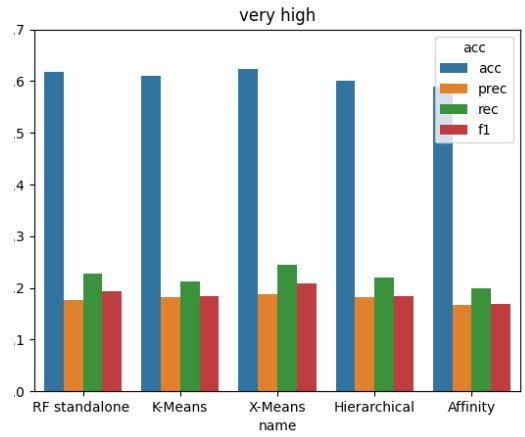
(b) Low



(c) Normal



(d) High



(e) Very High

Abbildung 4.4.: Accuracy, Precision, Recall und F1-Werte bei unterschiedlicher Klassenungleichverteilung.

4.3.6. Precision und Recall

Bezüglich Precision und Recall lässt sich noch eine weitere interessante Beobachtung feststellen: Der Recall-Wert liegt in fast allen Fällen über dem Precision-Wert, unabhängig vom verwendeten Clustering-Verfahren.

Dieses Phänomen scheint aus folgendem Grund gerade bei einer hohen Klassenungleichverteilung auf den ersten Blick ungewöhnlich: Bei hoher Klassenungleichverteilung werden die meisten Minority-Instanzen vom Klassifikator den Majority-Klassen zugeteilt. Dementsprechend existieren in den Majority-Klassen viele Falsch-Positive Instanzen, während die Minority-Klassen fast ausschließlich Falsch-Negative Instanzen aufweisen. Gemäß Definition 3.3.2 und Definition 3.3.3 schließt sich daraus, dass Majority-Klassen einen schlechten Precision-Wert aufweisen, während Minority-Klassen tendenziell einen schlechten Recall-Wert besitzen. Da jedoch viel mehr Minority-Klassen existieren, müsste der Recall insgesamt schlechter ausfallen als die Precision. Bei genauerer Betrachtung lässt sich der bessere Recall-Wert jedoch folgendermaßen erklären: Durch den bereits genannten Effekt, dass Klassifikatoren die Minority-Instanzen den Majority-Klassen zuteilen, werden vielen Minority-Klassen überhaupt keine Instanzen vom Klassifikator zugewiesen. Laut Berechnungen von Precision und Recall erhalten diese Klassen dann für beide Metriken den Wert 0, da sie keine Wahr-Positiven Instanzen aufweisen können. Für die Berechnung des Precision-Werte kann es für Minority-Klassen oft vorkommen, dass der Nenner ebenfalls den Wert null besitzt, da keine Falsch-Positiven Instanzen vorliegen. In diesem Fall wird automatisch der Wert 0 zugewiesen, da das Teilen durch den Wert 0 nicht möglich ist.

Folglich existieren nach der Klassifikation viele Minority-Klassen, welche für beide Metriken den Wert 0 aufweisen. Dadurch werden die Werte der Majority-Klassen, bei welchen der Recall meist besser ausfällt als die Precision, stärker gewichtet.

4.4. Schlussfolgerungen

Die Evaluation der Messergebnisse ergibt, dass durch eine geeignete Partitionierung der Trainingsdaten ähnlich gute, in Spezialfällen auch leicht verbesserte Vorhersagegenauigkeiten möglich sind. In den meisten Fällen reichen sie jedoch nicht signifikant über die Vorhersagegenauigkeit ohne Partitionierung hinaus. Jedoch können vergleichbare Ergebnisse auch dann erzielt werden, selbst wenn der zugrundeliegende Random Forest Klassifikator einen festen Random State erhält.

Innerhalb der Algorithmen zur Partitionierung ergibt sich sowohl auf Clustering als auch auf Klassifikationsseite der K-Means Algorithmus in den meisten Fällen als beste Option. Dieser liefert bei den Clustering-Metriken die besten Werte, was sich positiv auf die Klassifikationsergebnisse auswirkt. Ähnlich gute Werte nach der Klassifikation liefert der X-Means Algorithmus, auch wenn dieser den Clustering-Metriken zufolge eher schlecht abschneidet. Außerdem liefert X-Means die besten Ergebnisse bezüglich Precision, Recall und F1. In wenigen Fällen stellt sich auch hierarchisches Clustering als geeignete Partitionierungsmethode dar, während Affinity Propagation in den meisten Fällen keine guten Ergebnisse liefern kann.

Die vergleichsweise kleine Größe und hohe Komplexität des Datensatzes hat zur Folge, dass kleine Unterschiede der Einflussfaktoren einen direkten Effekt auf die Messergebnisse haben. Als größte Faktoren wurden dabei die zugrundeliegenden Daten und de Random State Parameter identifiziert.

Die hohe Variation der Ergebnisse hinsichtlich der Einflussfaktoren hat zur Folge, dass keine einheitliche Methode und Konfiguration für eine generelle und geeignete Partitionierung gewählt werden kann. Selbst wenn K-Means und X-Means in der Tendenz die besten Ergebnisse liefern, muss für jede neue Situation der geeignete Algorithmus und Parameter zunächst ermittelt werden, bevor die Partitionierung effektiv durchgeführt werden kann. Eine Verbesserung der Vorhersagegenauigkeit durch datengetriebene Partitionierung ist demnach von theoretischer Natur.

5. Zusammenfassung und Ausblick

In dieser Arbeit wurden die datengetriebene Partitionierung mittels Clustering-Algorithmen für komplexe Mehrklassenprobleme untersucht. Als Grundlage dienten dafür synthetisch generierte Daten, welche an realen Daten des Qualitätsmanagements des Fahrzeugbereiches angelehnt sind. Diese Daten weisen komplexe Eigenschaften wie Muli-Class Imbalance oder ein heterogenes Produktportfolio auf, welche zu einer eher mäßigen Vorhersagegenauigkeit herkömmlicher Klassifikatoren führt. Aus der Vorarbeit von Hirsch et. al. [HRM20] geht hervor, dass eine geeignete Partitionierung der Daten mittels spezifischen Domänenwissens die Vorhersagegenauigkeit der Klassifikatoren verbessert. Im Rahmen dieser Arbeit sollte geprüft werden, ob ein rein datengetriebenes Vorgehen ohne Domänenwissen ähnliche Verbesserungen hervorbringt.

Das Konzept zur datengetriebenen Partitionierung und Klassifikation verläuft in zwei Teilschritten: Zur Evaluation der Klassifikationsergebnisse muss zunächst der Datensatz in einem Trainingssatz und einen Testsatz aufgeteilt vorliegen. Der erste große Schritt sieht vor, den Trainingssatz mittels Clustering in einzelne Partitionen zu unterteilen. Hierzu muss ein geeigneter Clustering-Algorithmus mit einer passenden Konfiguration gefunden werden. Im zweiten Schritt werden Klassifikatoren auf den einzelnen Partitionen trainiert. Mithilfe des Testsatzes kann das Klassifikationsergebnis evaluiert werden. Jede Instanz des Testsatzes wird hier dem nächstgelegenen Cluster zugeteilt und der entsprechende Klassifikator zur Vorhersage angewendet. Sowohl für das Clustering als auch für die Klassifiaktion musste außerdem geklärt werden, wie das Problem der fehlenden Features behandelt wird.

Für die datengetriebene Partitionierung wurden die Clustering-Algorithmen K-Means, X-Means, hierachisches Clustering und Affinity Propagation im Näheren betrachtet. Diese Algorithmen sollen den Datensatz jeweils in geeignete Partitionen unterteilen. Um die Qualität der Clustering-Ergebnisse zu analysieren, wurden verschiedene Clustering-Metriken zu Hilfe genommen. K-Means erzielte hinsichtlich der Clustering-Metriken die besten Werte, gefolgt von hierarchischem Clustering. X-Means lieferte keine sonderlich guten Ergebnisse hinsichtlich der Clustering-Metriken, auch wenn der Algorithmus in der anschließenden Klassifikation der Daten besser abschnitt. Ebenfalls mäßige Ergebnisse erzielte der Affinity Propagation Algorithmus.

Bei der anschließenden Klassifikation auf den einzelnen Partitionen wurde zur Evaluation die Accuracy-Metrik genutzt. Zusätzlich wurden die Metriken Precision, Recall und F1 zu Hilfe genommen, um die Qualität des Klassifiaktionsergebnisses genauer zu evaluieren. Als Baseline zum Vergleich der Klassifikationsergebnisse wurden die Ergebnisse des Random Forest Algorithmus ohne Partitionierung verwendet. Durch die Partitionierung der Daten ergaben sich vergleichbare Ergebnisse zur Baseline. Die Algorithmen K-Means und X-Means erzielten meist ähnliche Accuracy-Werte zur Baseline und in wenigen Fällen ein besseres Ergebnis. X-Means schafft es außerdem, meist einen besseren Precision, Recall und F1 Wert zu erreichen. Affinity Propagation erwies sich in der Regel nicht geeignet zur Partitionierung der Trainingsdaten.

Da für K-means und hierarchisches Clustering die Anzahl der Cluster vom Nutzer bestimmt werden muss, wurde eine Testreihe mit unterschiedlicher Clusteranzahl durchgeführt. Sowohl für K-Means als auch für hierarchisches Clustering ergaben sich die besten Klassifikationsergebnisse bei einer Einteilung in vergleichsweise wenige Cluster. Als Optimum wurde dabei eine Clusteranzahl von 10 Clustern ermittelt.

Des Weiteren wurde identifiziert, dass die Wahl des Random State Parameters und des Datensatzes einen hohen Einfluss darauf haben, welcher der Clustering-Algorithmen eine geeignete Partitionierung liefert. Je nach Datensatz kann es sich unterscheiden, welcher der genutzten Clustering-Algorithmen das beste Ergebnis liefert. Außerdem wurde betrachtet, welche Methode sich am besten zur Adressierung der fehlenden Features eignet: Dabei stellte sich heraus, dass das Auffüllen fehlender Werte durch den Wert '0' bessere Ergebnisse liefert als eine Imputation fehlender Werte mittels KNN-Imputation oder Miss Forest.

Die Betrachtung verschiedener Grade der Klassenungleichverteilung lieferte ebenfalls keine neuen Tendenzen bezüglich der Vorhersagegenauigkeit. Sowohl bei sehr niedriger, als auch bei sehr hoher Klassenungleichverteilung konnten die Clustering-Algorithmen K-Means und X-Means mit den Accuracy-Ergebnissen der Baseline mithalten, erreichten jedoch selten nennenswerte Besserungen.

Zusammenfassend lässt sich sagen, dass eine datengetriebene Partitionierung der Trainingsdaten vergleichbare Klassifiaktionsergebnisse erzielen kann wie ein Ansatz ohne Partitionierung. Für eine Verbesserung der Vorhersagegenauigkeit, wie sie durch eine Partitionierung mittels spezifischen Domänenwissens hervorgegangen ist, müssen die Ansätze jedoch noch weitreichender untersucht werden. Dabei sollten die einzelnen Clustering-Verfahren gezielter auf die Herausforderungen angepasst werden, etwa durch adaptieren der Konfigurationen oder der Funktionsweise des Algorithmus. Außerdem könnte, durch eine Kombination der in der Arbeit genutzten Clustering- und Klassifikationsmetriken, eine gezielte Metrik formuliert werden, welche eine gute Qualität der Partitionierung widerspiegelt.

Ausblick

Zur Partitionierung der Trainingsdaten wurden insgesamt vier Clustering-Algorithmen im Näheren betrachtet. Weitere vielversprechende Algorithmen standen in Erwägung, jedoch schafften es diese nicht, den vorliegenden Datensatz in mehrere Partitionen zu unterteilen. Dabei ist unklar, ob eine Partitionierung mithilfe dieser Algorithmen auf dem Datensatz grundsätzlich nicht möglich ist oder ob die geeigneten Parameter für diese Algorithmen nicht gefunden werden konnten. Auch die Verwendung des AutoML4Clust [TFS21] Frameworks ergab keine aufschlussreichen Parameter, mit welchen die Clustering Algorithmen den Datensatz partitionieren könnten. Demnach ist durch genauere Betrachtung weiterer Clustering-Algorithmen zu klären, ob diese die Vorhersagegenauigkeit der Klassifikatoren verbessern können.

Weitere Analyse ist ebenfalls bei Betrachtung der unterschiedlichen Klassenungleichverteilungen möglich. Die verwendeten Clustering-Algorithmen nutzen für alle Stufen der Ungleichverteilung dieselben Parameter. Dabei ist es durchaus möglich, dass etwa eine Einteilung in mehrere Cluster

bei besonders hoher oder niedriger Klassenungleichverteilung eine geeignetere Partitionierung hervorbringt. Eine genauere Untersuchung, welche Parameter bei welcher Ungleichverteilung die besten Ergebnisse erbringen, kann neue Erkenntnisse zur geeigneten Partitionierung liefern.

Der Gini-Index wurde im Rahmen dieser Arbeit lediglich genutzt, um das Clustering-Ergebnis der Algorithmen zu evaluieren. Jedoch existieren Möglichkeiten, die Metrik gezielt zu verwenden, um das Ergebnis der Partitionierung zu verbessern. In der Partitionierung mittels Domänenwissen von Hirsch et. al. [HRM20] etwa wurde mithilfe des Gini-Index die Klassenungleichverteilung auf den einzelnen Partitionen gemessen. Überschritt der Index dabei einen Schwellwert, wurden die Partitionen nochmals in zwei Untergruppen nach Majority und Minority Instanzen getrennt. Ein analoges Vorgehen ist auch bei der datengetriebenen Partitionierung möglich und trägt möglicherweise zu einer Verbesserung der Vorhersagegenauigkeit bei. Dabei besteht die Möglichkeit ein solches Vorgehen gezielt in einen existierenden Clustering-Algorithmus, wie etwa X-Means oder dem Bisecting K-Means [SKK00], zu implementieren. Zusätzlich sollte geprüft werden, welche weiteren Metriken sich als Kriterien eignen, um den Datensatz entsprechend zu partitionieren. Mithilfe dieser Untersuchungen wäre es möglich, einen angepassten partitionierenden Algorithmus zu formulieren, welcher die Herausforderungen gezielt adressiert.

Literaturverzeichnis

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander. „OPTICS: Ordering Points to Identify the Clustering Structure“. In: *SIGMOD Rec.* 28.2 (Juni 1999), S. 49–60. ISSN: 0163-5808. DOI: 10.1145/304181.304187. URL: https://doi.org/10.1145/304181.304187 (zitiert auf S. 30).

[AV06] D. Arthur, S. Vassilvitskii. *k-means++: The advantages of careful seeding*. Techn. Ber. Stanford, 2006 (zitiert auf S. 29).

[Bre01] L. Breiman. „Machine Learning, Volume 45, Number 1 - SpringerLink“. In: *Machine Learning* 45 (Okt. 2001), S. 5–32. DOI: 10.1023/A:1010933404324 (zitiert auf S. 33).

[CH74] T. Caliński, J. Harabasz. „A dendrite method for cluster analysis“. In: *Communications in Statistics* 3.1 (1974), S. 1–27. DOI: 10.1080/03610927408827101. eprint: https://www.tandfonline.com/doi/pdf/10.1080/03610927408827101. URL: https://www.tandfonline.com/doi/abs/10.1080/03610927408827101 (zitiert auf S. 32).

[DB79] D. L. Davies, D. W. Bouldin. „A Cluster Separation Measure“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), S. 224–227. DOI: 10.1109/TPAMI.1979.4766909 (zitiert auf S. 32).

[Die00] T. G. Dietterich. „Ensemble methods in machine learning“. In: *International workshop on multiple classifier systems*. Springer. 2000, S. 1–15 (zitiert auf S. 33).

[EKSX96] M. Ester, H. P. Kriegel, J. Sander, X. Xiaowei. „A density-based algorithm for discovering clusters in large spatial databases with noise“. In: (Dez. 1996). URL: https://www.osti.gov/biblio/421283 (zitiert auf S. 30).

[FD07] B. Frey, D. Dueck. „Clustering by Passing Messages Between Data Points“. In: *Science* 315 (2007), S. 972–976 (zitiert auf S. 29).

[FH75] K. Fukunaga, L. Hostetler. „The estimation of the gradient of a density function, with applications in pattern recognition“. In: *IEEE Transactions on Information Theory* 21.1 (1975), S. 32–40. DOI: 10.1109/TIT.1975.1055330 (zitiert auf S. 30).

[FKE+19] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter. „Auto-sklearn: efficient and robust automated machine learning“. In: *Automated Machine Learning*. Springer, Cham, 2019, S. 113–134 (zitiert auf S. 21).

[FS97] Y. Freund, R. E. Schapire. „A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting“. In: *Journal of Computer and System Sciences* 55.1 (1997), S. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X (zitiert auf S. 33).

[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. 15, 16, 23, 24, 32, 33).

[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, 16, 23, 40, 55, 57).

[HYS+17] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing. „Learning from class-imbalanced data: Review of methods and applications“. In: *Expert Systems with Applications* 73 (2017), S. 220–239. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.12.035. URL: https://www.sciencedirect.com/science/article/pii/S0957417416307175 (zitiert auf S. 18).

[KGJH16] B. Krawczyk, M. Galar, Ł. Jeleń, F. Herrera. „Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy“. In: *Applied Soft Computing* 38 (2016), S. 714–726. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2015.08.060. URL: https://www.sciencedirect.com/science/article/pii/S1568494615005815 (zitiert auf S. 18).

[Kra16] Krawczyk. „Learning from imbalanced data: open challenges and future directions.“ In: *B. Prog Artif Intell 5, 221–232* (2016). URL: https://doi.org/10.1007/s13748-016-0094-0 (zitiert auf S. 18).

[LBDC12] J. Liang, L. Bai, C. Dang, F. Cao. „The *K*-Means-Type Algorithms Versus Imbalanced Data Distributions“. In: *IEEE Transactions on Fuzzy Systems* 20.4 (2012), S. 728–745. DOI: 10.1109/TFUZZ.2011.2182354 (zitiert auf S. 20, 21).

[LCT21] Y. Lu, Y.-M. Cheung, Y. Y. Tang. „Self-Adaptive Multiprototype-Based Competitive Learning Approach: A k-Means-Type Algorithm for Imbalanced Data Clustering“. In: *IEEE Transactions on Cybernetics* 51.3 (2021), S. 1598–1612. DOI: 10.1109/TCYB.2019.2916196 (zitiert auf S. 20).

[LTHJ17] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, J.-S. Jhang. „Clustering-based undersampling in class-imbalanced data“. In: *Information Sciences* 409-410 (2017), S. 17–26. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.05.008. URL: https://www.sciencedirect.com/science/article/pii/S0020025517307235 (zitiert auf S. 16, 22).

[Mac67] J. B. MacQueen. „Some Methods for Classification and Analysis of MultiVariate Observations“. In: *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. Hrsg. von L. M. L. Cam, J. Neyman. Bd. 1. University of California Press, 1967, S. 281–297 (zitiert auf S. 19, 28).

[ORSS17] N. Ofek, L. Rokach, R. Stern, A. Shabtai. „Fast-CBUS: A fast clustering-based undersampling method for addressing the class imbalance problem“. In: *Neurocomputing* 243 (2017), S. 88–102. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.03.011. URL: https://www.sciencedirect.com/science/article/pii/S0925231217304939 (zitiert auf S. 16, 22).

[PM02] D. Pelleg, A. Moore. „X-means: Extending K-means with Efficient Estimation of the Number of Clusters“. In: *Machine Learning, p* (Jan. 2002) (zitiert auf S. 20, 29).

[Rou87] P. J. Rousseeuw. „Silhouettes: A graphical aid to the interpretation and validation of cluster analysis“. In: *Journal of Computational and Applied Mathematics* 20 (1987), S. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257 (zitiert auf S. 31).

[Sar90] W. S. Sarle. *Algorithms for clustering data*. 1990 (zitiert auf S. 29).

[SB12] D. Stekhoven, P. Bühlmann. „MissForest?Non-parametric missing value imputation for mixed-type data“. In: *Bioinformatics (Oxford, England)* 28 (Jan. 2012), S. 112–8. DOI: 10.1093/bioinformatics/btr597 (zitiert auf S. 36).

[SKK00] M. Steinbach, G. Karypis, V. Kumar. „A comparison of document clustering techniques“. In: (2000) (zitiert auf S. 57).

[TCS+01] O. Troyanskaya, M. Cantor, G. Sherlock, T. Hastie, R. Tibshirani, D. Botstein, R. Altman. „Missing Value Estimation Methods for DNA Microarrays“. In: *Bioinformatics* 17 (Juli 2001), S. 520–525. DOI: 10.1093/bioinformatics/17.6.520 (zitiert auf S. 36).

[TFS21] D. Tschechlov, M. Fritz, H. Schwarz. „AutoML4Clust: Efficient AutoML for Clustering Analyses“. Englisch. In: *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*. Online, 2021, S. 1–6. DOI: 10.5441/002/EDBT.2021.32. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2021-03&engl=0 (zitiert auf S. 21, 22, 31, 41, 56).

[THHL13] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown. „Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms“. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, S. 847–855 (zitiert auf S. 21).

[XWC09] H. Xiong, J. Wu, J. Chen. „K-Means Clustering Versus Validation Measures: A Data-Distribution Perspective“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2 (2009), S. 318–331. DOI: 10.1109/TSMCB.2008.2004559 (zitiert auf S. 19).

[YL09] S.-J. Yen, Y.-S. Lee. „Cluster-based under-sampling approaches for imbalanced data distributions“. In: *Expert Systems with Applications* 36.3, Part 1 (2009), S. 5718–5727. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2008.06.108. URL: https://www.sciencedirect.com/science/article/pii/S0957417408003527 (zitiert auf S. 16, 18, 22).

Alle URLs wurden zuletzt am 27. 07. 2021 geprüft.

A. Liste der Clustering und Klassifikationsmetriken

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline Mittelw.	0.901	0.775	0.691	0.720				
RF Baseline Max.	0.925	0.812	0.730	0.760				
RF Baseline Min.	0.880	0.734	0.663	0.690				
K-Means Mittelw.	0.898	0.701	0.656	0.673	0.176	108.297	1.643	0.588
K-Means Max.	0.925	0.903	0.790	0.831	0.176	108.487	1.670	0.607
K-Means Min.	0.865	0.679	0.653	0.662	0.182	108.784	1.601	0.575
X-Means Mittelw.	0.888	0.726	0.672	0.684	0.137	69.800	1.780	0.434
X-Means Max.	0.930	0.898	0.810	0.838	0.146	71.514	1.744	0.403
X-Means Min.	0.845	0.554	0.509	0.513	0.135	67.392	1.778	0.394
Hierarchisch Mittelw.	0.892	0.696	0.641	0.660	0.079	70.297	2.030	0.430
Hierarchisch Max.	0.910	0.750	0.717	0.718	0.079	70.297	2.030	0.430
Hierarchisch Min.	0.875	0.676	0.613	0.638	0.079	70.297	2.030	0.430
Affinity Mittelw.	0.891	0.641	0.689	0.648	0.113	44.202	1.720	0.260
Affinity Max.	0.905	0.666	0.721	0.677	0.113	44.202	1.720	0.260
Affinity Min.	0.880	0.582	0.660	0.611	0.113	44.202	1.720	0.260

Tabelle A.1.: Klassifikations- und Clustering-Ergebnisse nach Partitionierung: Make_classification() Methode aus Scikit-Learn

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline Mittelw.	0.332	0.156	0.177	0.139				
RF Baseline Max.	0.357	0.178	0.189	0.157				
RF Baseline Min.	0.303	0.119	0.154	0.115				
K-Means Mittelw.	0.337	0.152	0.178	0.139	0.182	98.688	2.138	0.412
K-Means Max.	0.360	0.172	0.187	0.149	0.186	98.499	2.096	0.410
K-Means Min.	0.307	0.114	0.146	0.107	0.179	97.906	2.137	0.426
X-Means Mittelw.	0.324	0.142	0.168	0.131	0.137	54.988	2.412	0.355
X-Means Max.	0.353	0.143	0.193	0.148	0.125	50.393	2.346	0.336
X-Means Min.	0.297	0.134	0.151	0.117	0.139	53.987	2.275	0.354
Hierarchisch Mittelw.	0.321	0.136	0.165	0.126	0.145	82.926	2.059	0.350
Hierarchisch Max.	0.347	0.131	0.163	0.126	0.145	82.926	2.059	0.350
Hierarchisch Min.	0.297	0.130	0.151	0.113	0.145	82.926	2.059	0.350
Affinity Mittelw.	0.299	0.112	0.146	0.110	0.101	46.832	3.004	0.350
Affinity Max.	0.320	0.134	0.159	0.121	0.101	46.832	3.004	0.350
Affinity Min.	0.277	0.101	0.135	0.101	0.101	46.832	3.004	0.350

Tabelle A.2.: Klassifikations- und Clustering-Ergebnisse nach Partitionierung: Data Generator, kein Random State, Fehlende Werte durch ’0’ ersetzt

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.283	0.123	0.128	0.094				
K-Means Mittelw.	0.290	0.110	0.150	0.106	0.138	114.917	3.054	0.420
K-Means Max.	0.333	0.164	0.186	0.138	0.121	115.150	3.116	0.420
K-Means Min.	0.260	0.075	0.114	0.083	0.150	113.518	2.951	0.398
X-Means Mittelw.	0.303	0.130	0.158	0.119	0.090	76.348	3.380	0.388
X-Means Max.	0.340	0.133	0.166	0.130	0.037	68.911	3.882	0.404
X-Means Min.	0.263	0.078	0.118	0.084	0.139	78.122	3.080	0.385
Hierarchisch	0.267	0.111	0.126	0.094	0.196	98.431	2.273	0.261
Affinity	0.220	0.090	0.105	0.081	0.002	48.180	4.937	0.339

Tabelle A.3.: KNN-Imputation auf gessamtem Datensatz.

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.287	0.117	0.140	0.104				
K-Means Mittelw.	0.300	0.118	0.148	0.108	0.135	109.315	3.029	0.417
K-Means Max.	0.330	0.146	0.183	0.137	0.121	109.419	3.099	0.426
K-Means Min.	0.280	0.098	0.132	0.094	0.150	108.912	2.773	0.419
X-Means Mittelw.	0.304	0.123	0.153	0.113	0.075	74.173	3.401	0.393
X-Means Max.	0.340	0.167	0.184	0.151	0.038	62.878	3.522	0.377
X-Means Min.	0.270	0.106	0.122	0.093	0.036	62.321	3.727	0.381
Hierarchisch	0.280	0.106	0.128	0.097	0.173	96.446	2.637	0.307
Affinty	0.297	0.108	0.136	0.104	0.002	42.713	4.761	0.324

Tabelle A.4.: Miss Forest Impuation auf gessamtem Datensatz.

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.120	0.088	0.110	0.088				
K-Means Mittelw.	0.102	0.115	0.112	0.103	0.201	103.659	2.041	0.263
K-Means Max.	0.117	0.148	0.127	0.120	0.197	103.788	1.983	0.267
K-Means Min.	0.083	0.082	0.090	0.077	0.202	104.563	2.033	0.264
X-Means Mittelw.	0.110	0.107	0.110	0.099	0.148	58.105	2.339	0.216
X-Means Max.	0.140	0.137	0.155	0.134	0.124	62.026	2.447	0.234
X-Means Min.	0.077	0.061	0.077	0.058	0.133	56.964	2.286	0.220
Hierarchisch	0.103	0.100	0.106	0.093	0.178	87.463	2.043	0.254
Affinity	0.107	0.103	0.108	0.091	0.128	50.657	2.397	0.207

Tabelle A.5.: Klassenungleichverteilung - very low

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.243	0.107	0.136	0.104				
K-Means Mittelw.	0.225	0.127	0.138	0.114	0.188	80.593	2.133	0.324
K-Means Max.	0.257	0.135	0.154	0.124	0.176	80.150	2.266	0.341
K-Means Min.	0.200	0.116	0.115	0.093	0.173	79.174	2.054	0.321
X-Means Mittelw.	0.200	0.115	0.128	0.105	0.154	47.798	2.208	0.274
X-Means Max.	0.230	0.133	0.153	0.125	0.165	47.139	2.178	0.295
X-Means Min.	0.163	0.088	0.106	0.085	0.121	45.363	2.609	0.271
Hierarchisch	0.217	0.086	0.121	0.087	0.204	76.380	2.060	0.303
Affinity	0.163	0.068	0.105	0.076	0.100	39.243	2.711	0.278

Tabelle A.6.: Klassenungleichverteilung - low

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.327	0.148	0.177	0.144				
K-Means Mittelw.	0.315	0.141	0.170	0.133	0.171	89.798	2.233	0.423
K-Means Max.	0.340	0.164	0.187	0.153	0.175	86.379	2.415	0.411
K-Means Min.	0.297	0.106	0.155	0.111	0.160	87.824	2.186	0.429
X-Means Mittelw.	0.315	0.154	0.183	0.147	0.127	48.335	2.462	0.367
X-Means Max.	0.343	0.201	0.209	0.184	0.153	47.992	2.460	0.374
X-Means Min.	0.283	0.112	0.151	0.113	0.117	51.868	2.410	0.383
Hierarchisch	0.307	0.145	0.165	0.131	0.188	74.401	2.218	0.355
Affinity	0.323	0.155	0.157	0.138	0.069	37.497	3.227	0.344

Tabelle A.7.: Klassenungleichverteilung - normal

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF baseline	0.447	0.157	0.179	0.143				
K-Means Mittelw.	0.442	0.153	0.193	0.152	0.188	98.612	2.307	0.444
K-Means Max.	0.453	0.176	0.206	0.167	0.191	99.584	2.252	0.457
K-Means Min.	0.427	0.146	0.183	0.142	0.195	97.282	2.517	0.437
X-Means Mittelw.	0.427	0.161	0.195	0.157	0.114	53.573	2.569	0.381
X-Means Max.	0.450	0.189	0.230	0.190	0.098	54.215	2.831	0.398
X-Means Min.	0.407	0.148	0.176	0.140	0.113	57.172	2.639	0.387
Hierarchisch	0.437	0.157	0.194	0.154	0.216	91.555	2.091	0.408
Affintiy	0.403	0.136	0.156	0.124	0.069	46.427	3.425	0.384

Tabelle A.8.: Klassenungleichverteilung - high

Verfahren	acc	prec	rec	f1	sil	chi	dbi	gini_avg
RF Baseline	0.617	0.176	0.227	0.193				
K-Means Mittelw.	0.599	0.172	0.210	0.180	0.166	88.327	2.286	0.473
K-Means Max.	0.610	0.182	0.212	0.185	0.158	88.051	2.399	0.481
K-Means Min.	0.573	0.177	0.211	0.180	0.166	88.069	2.259	0.471
X-Means Mittelw.	0.604	0.185	0.233	0.198	0.125	46.379	2.366	0.401
X-Means Max.	0.623	0.188	0.245	0.208	0.143	43.997	2.157	0.376
X-Means Min.	0.583	0.153	0.208	0.167	0.105	45.846	2.531	0.401
Hierarchisch	0.600	0.182	0.220	0.184	0.180	82.245	2.240	0.445
Affinity	0.590	0.167	0.200	0.169	0.100	41.728	2.562	0.398

Tabelle A.9.: Klassenungleichverteilung - very high

Erklärung

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

Stuttgart, den 01.10.2021 

Ort, Datum, Unterschrift

Nachreichen der Druckexemplare

Ich versichere hiermit, dass ich die Druckexemplare meiner Abschlussarbeit innerhalb von maximal 4 Wochen nach der Abgabedeadline für die Arbeit per Post nachreiche. Die Druckexemplare stimmen mit dem eingereichten elektronischen Exemplar überein.

Datum und Unterschrift:

01.10.2021