Predictive Maintenance: Vorhersage von Stromrichter-ausfällen bei Güterzügen

Masterarbeit

Dennis Schmidt | 1946134
Wirtschaftsinformatik

TECHNISCHE UNIVERSITÄT DARMSTADT

WIRTSCHAFTS INFORMATIK

Dennis Schmidt
Matrikelnummer: 1946134
Studiengang: Master of Science Wirtschaftsinformatik

Masterarbeit
Thema: "Predictive Maintenance: Vorhersage von Stromrichterausfällen bei Güterzügen"

Eingereicht: 31. März 2017

Betreuer:
Dipl.-Inform. Sebastian Kauschke (Fachgebiet Knowledge Engineering)
M.Sc. Adrian Engelbrecht (Fachgebiet Wirtschaftsinformatik)

Prof. Dr. Johannes Fürnkranz
Fachgebiet Knowledge Engineering
Fachbereich Informatik
Technische Universität Darmstadt
Hochschulstraße 10
64289 Darmstadt

Prof. Dr. Peter Buxmann
Fachgebiet Wirtschaftsinformatik | Software Business & Information Management
Fachbereich Rechts- und Wirtschaftswissenschaften
Technische Universität Darmstadt
Hochschulstraße 1
64289 Darmstadt

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Sämtliche aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

Darmstadt, den 31. März 2017

Abstract

Die DB Cargo verfolgt im Rahmen des „Techlok“-Projektes das Ziel, ein Predictive Maintenance System für ihre Flotte zu entwickeln und kooperiert hierzu mit der Technischen Universität Darmstadt. Als Teil dieser Kooperation untersucht die vorliegende Masterarbeit die Vorhersage von Stromrichterausfällen bei Lokomotiven der Baureihe 185. Hierzu wurde von Kauschke et al. (2016) bereits ein erster Prototyp entwickelt. Die Masterarbeit baut auf diesem auf und untersucht im Rahmen der ersten Forschungsfrage, ob es möglich ist, die Vorhersagequalität durch alternative Ansätze zu steigern. Hierzu werden in drei der vier grundlegenden Bestandteile des bisherigen Systems Änderungen vorgenommen bzw. neue Algorithmen entwickelt. Die Ergebnisse zeigen, dass die Vorhersageleistung durch die Neuerungen gesteigert werden kann. Des Weiteren wird im Rahmen der zweiten Forschungsfrage untersucht, ob es neben einer Warnung des Lokführers vor einem drohenden Stromrichterausfall auch möglich ist, eine Vorhersage über die Zeit bis zum Ausfall zu geben. Hierzu wird ein Multiklassen-Ansatz eingeführt und das zuvor entwickelte System angepasst. Die Ergebnisse zeigen, dass die Genauigkeit der Zeitvorhersage mit Hilfe dieses Ansatzes gering ausfällt und die Zeitvorhersage somit nicht für eine praktische Anwendung geeignet ist. Zuletzt wird das entwickelte System im Rahmen der dritten Forschungsfrage aus wirtschaftlicher Sicht untersucht. Als zentrales Ergebnis liefert die Arbeit hierbei ein Kostenmodell, welches zur Bewertung des Systems genutzt werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ARFF	Attribute-Relation File Format
ARMA	Autoregressive-Moving Average
CBM	Condition Based Maintenance
CV	Cross Validation (dt. Kreuzvalidierung)
DIN	Deutsches Institut für Normung
DL	Description Length
EN	Europäische Norm
FN	False Negatives
FP	False Positives
I-REP	Incremental Reduced Error Pruning
IT	Informationstechnik
KDD	Knowledge Discovery in Database
PdM	Predictive Maintenance
PMS	Predictive Maintenance System
REMAIN	Modular System for Reliability and Maintenance Management in European Rail Transport
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
SINTEF	Stiftelsen for industriell og teknisk forskning (norwegisch)
TN	True Negatives
TP	True Positives
TU	Technische Universität
Weka	Waikato Environment for Knowledge Analysis

1 Einleitung

Das Internet der Dinge und Maschinelles Lernen (bzw. Künstliche Intelligenz im Allgemeinen) zählen laut diverser Studien zu den wichtigsten aktuellen Technologietrends.[1] So ermittelte die International Data Group im Jahre 2016, dass 72% der in einer Studie befragten Unternehmen davon ausgehen, dass das Internet der Dinge zukünftig eine wichtige oder sogar sehr wichtige Bedeutung bei ihnen einnehmen wird.[2] Gleichzeitig zeigen Studien von General Electric und Accenture (2015), Accenture (2014) und McKinsey Global Institute (2016) die besondere Relevanz einer automatisierten Datenanalyse (insbesondere durch Methoden des Maschinellen Lernens).[3] Beide Technologien sind dabei eng miteinander verbunden. So werden Methoden des Maschinellen Lernens verwendet, um Sensordaten aus dem Internet der Dinge zu analysieren.[4] Ein populäres Anwendungsszenario in diesem Zusammenhang ist Predictive Maintenance.[5] Kurz gefasst besteht das Ziel von Predictive Maintenance darin, Zustandsdaten von technischen Anlagen zur Vorhersage von Fehlern zu nutzen.[6] Aus diesem Grund findet Predictive Maintenance in der Industrie[7], bei Automobilen[8] aber auch bei Zügen[9] Anwendung.

Auch der größte Güterbahnbetreiber Europas, die DB Cargo, welche 2.869 Lokomotiven und 87.264 Güterwagen betreibt,[10] hat sich im Rahmen des Digitalisierungsprojektes „Techlok“ dazu entschlossen, Predictive Maintenance für ihre Flotte einzuführen. Neben der Ausstattung der Lokomotiven und Waggons mit neuer Sensorik liegt der Fokus dabei auf der Entwicklung eines Vorhersagesystems.[11] Insbesondere für die Umsetzung des letzten Punktes wurde im Jahr 2014 eine Kooperation mit dem Fachgebiet Knowledge Engineering der Technischen Universität Darmstadt eingegangen[12], welches unter anderem im Umfeld von Maschinellem

1 Vgl. Gartner (2016) , IDG Research Services (2016), S. 16 ff, McKinsey Global Institute (2016), S. vi ff, General Electric;Accenture (2015), S. 5 ff, Accenture (2014), S. 2 ff.

2 Vgl. IDG Research Services (2016), S. 19.

3 Vgl. General Electric;Accenture (2015), S. 5 f, Accenture (2014), S. 3 ff, McKinsey Global Institute (2016), S. vi ff.

4 Vgl. McKinsey Global Institute (2016), S. 32, 75 f. und 107 ff, Hewlett Packard Enterprise (2016).

5 Vgl. IDG Research Services (2016), S. 21, McKinsey Global Institute (2016), S. 84.

6 Vgl. Krishnamurthy et al. (2005), S. 64.

7 Vgl. Susto et al. (2015), S. 812 ff.

8 Vgl. Prytz et al. (2013), S. 118 ff.

9 Vgl. Yang;Létourneau (2005), S. 516 ff.

10 Vgl. DB Cargo (2016).

11 Vgl. DB Cargo (2015).

12 Vgl. Knowledge Engineering Group (o. J.-a).

Lernen und Data Mining forscht[13]. Ziel der Kooperation ist es, Methoden zur Analyse der Daten zu entwickeln, um so ein möglichst zuverlässiges Vorhersagesystem für verschiedene Fehler zu erhalten.[14]

Die vorliegende Masterarbeit ist Teil dieser Kooperation. Sie verfolgt das allgemeine Ziel, auf Basis von Logdaten Stromrichterausfälle der Lokomotiven Baureihe 185 im Voraus zu erkennen. Ein Stromrichter ist dabei ein Bauteil der Lok, welcher für die elektrische Versorgung der Antriebsmotoren verantwortlich ist. Die Masterarbeit baut auf dem bisherigen System von Kauschke et al. (2016) auf, welche bereits einen ersten Prototyp zur Vorhersage dieser Ausfälle entwickelt haben.[15] Konkretes Ziel der Masterarbeit bildet die Untersuchung der folgenden Forschungsfragen:

1. Ist es möglich, die Vorhersageleistung des Predictive Maintenance Systems von Kauschke et al. (2016) durch alternative Ansätze bzw. Algorithmen zu verbessern?
2. Kann zusätzlich zur Warnung vor einem Stromrichterausfall auch eine zeitliche Vorhersage getroffen werden?
3. Wie ist das Predictive Maintenance System aus wirtschaftlicher Sicht zu bewerten?

Zur Beantwortung dieser Forschungsfragen unterteilt sich die Masterarbeit grundlegend in sieben Kapitel. Nach dem einleitenden, aktuellen Kapitel werden im *zweiten Kapitel* die theoretischen Grundlagen beschrieben, welche für das Verständnis der weiteren Arbeit notwendig sind. Dabei wird unter anderem auf Predictive Maintenance, Maschinelles Lernen, die verwendeten Lernalgorithmen und auf die Evaluierung eingegangen. Anschließend werden im *dritten Kapitel* der zur Verfügung stehende Datensatz und das bisherige System von Kauschke et al. (2016) beschrieben.

Nachdem die theoretischen Grundlagen gelegt und das bisherige System präsentiert wurde, wird im *vierten Kapitel* die erste Forschungsfrage untersucht. Hierzu wird zunächst erläutert, welche Änderungen im Vergleich zum System von Kauschke et al. (2016) vorgenommen bzw. welche neue Verfahren im Rahmen der Masterarbeit entwickelt wurden. Anschließend wird beschrieben, wie bei der Evaluierung des alten und des neuen Systems vorgegangen wurde. Zuletzt werden die Evaluierungsergebnisse präsentiert und dabei beide Systeme verglichen. Das

13 Vgl. Knowledge Engineering Group (o. J.-b).

14 Vgl. Knowledge Engineering Group (o. J.-a).

15 Vgl. Kauschke et al. (2016), S. 151 ff.

fünfte Kapitel untersucht daraufhin die zweite Forschungsfrage. Hierzu wird wiederum beschrieben, wie das System angepasst wurde, um auch eine Vorhersage über die Zeit bis zum Ausfall geben zu können. Anschließend folgt, wie auch im vierten Kapitel, die Beschreibung der Evaluierung und anschließend die Präsentation bzw. Diskussion der Ergebnisse. Im *sechsten Kapitel* wird letztlich die dritte Forschungsfrage bearbeitet. Dabei wird das Predictive Maintenance System aus finanzieller und nicht-finanzieller Sichtweise untersucht. Im Fokus steht dabei die finanzielle Analyse, in welcher ein Kostenmodell zur Bewertung des Systems entwickelt und beispielhaft angewandt wird. In der nicht-finanziellen Analyse wird anschließend untersucht, welche nicht-monetären Gründe für die Einführung des Systems sprechen.

Zuletzt folgt im *siebten Kapitel* eine Zusammenfassung der Ergebnisse und ein Ausblick auf mögliche Erweiterungen des Predictive Maintenance Systems.

2 Grundlagen

Nachdem die Forschungsfragen definiert bzw. die Problemstellung erläutert wurde, werden im folgendem Kapitel die theoretischen Grundlagen beschrieben, welche als Fundament für die Beantwortung der Forschungsfragen und somit für das Verständnis der weiteren Vorgehensweise notwendig sind. Das Kapitel unterteilt sich hierzu in sechs Abschnitte. Im ersten Abschnitt wird zunächst Predictive Maintenance definiert und die Eigenschaften dieses Wartungskonzepts näher erläutert. Im zweiten Abschnitt folgt eine grundlegende Beschreibung von Maschinellem Lernen, welche die in dieser Arbeit verwendete Technik zur Umsetzung von Predictive Maintenance darstellt. Im dritten Abschnitt wird anschließend die Funktionsweise der konkret verwendeten Lernalgorithmen erklärt. Nach der Beschreibung der Algorithmen folgen im vierten Abschnitt die Verfahren und Maße zur deren Evaluierung. In den letzten beiden Abschnitten werden die verwendete Softwaresuite Weka und verwandte Forschungsarbeiten vorgestellt.

2.1 Predictive Maintenance

Im folgenden Abschnitt wird nun auf Predictive Maintenance eingegangen. Im ersten Unterabschnitt wird hierzu Predictive Maintenance definiert und abgegrenzt, im zweiten Unterabschnitt hingegen werden die Arten und Verfahren von Predictive Maintenance präsentiert.

2.1.1 Definition und Abgrenzung

Preditive Maintenance, manchmal auch als Condition Based Maintenance (CBM), Online Monitoring oder Risk-Based Maintenance bezeichnet[16], besitzt in der Literatur keine eindeutige Definition[17]. Um den Begriff in dieser Arbeit festzulegen, soll deshalb auf zwei ausgewählte Definitionen zurückgegriffen werden. Die erste Definition geben Jardine et al. (2006):

> "CBM is a maintenance program that recommends maintenance actions based on the information collected through condition monitoring. CBM attempts to avoid unnecessary maintenance tasks by taking maintenance actions only when there is evidence of abnormal behaviours of a physical asset."[18]

16 Vgl. Hashemian;Bean (2011), S. 3480.

17 Vgl. Mobley (2002), S. 4.

18 Jardine et al. (2006), S. 1484.

Die zweite Definition stammt von Krishnamurthy et al. (2005), welche Predictive Maintenance definieren als:

> "Predictive Maintenance (PdM) is the general term applied to a family of technologies used to monitor and assess the health status of a piece of equipment (e.g., a motor, chiller, or cooler) that is in service. PdM technologies allow the user to detect most impending failures well in advance, as long as analysis is performed with sufficient frequency."[19]

Zusammengefasst kann festgehalten werden, dass Predictive Maintenance einen Wartungsansatz darstellt, welcher die Zustände von technischen Anlagen beobachtet, auf deren Basis Abweichungen vom Normzustand erkennt und hierdurch Fehler im Voraus antizipiert, sodass unnötige Wartungsarbeiten vermieden werden.[20]

Predictive Maintenance ordnet sich dabei in eine Reihe von Wartungsansätzen ein. Grundlegend unterscheidet die Literatur drei Ansätze[21], welche jeweils unterschiedlich komplex und effizient sind[22]:

- Run-To-Failure Maintenance (Unplanned Maintenance, Breakdown Maintenance)
- Preventive Maintenance (Planned Maintenance, Time-Based Maintenance)
- Predictive Maintenance (Condition Based Maintenance, Online Monitoring, Risk-Based Maintenance)[23]

Den ältesten Ansatz bildet die *Run-To-Failure Maintenance*.[24] Unter diesem Begriff wird eine Wartungsstrategie verstanden, bei welcher Reparaturen und Wartungen erst durchgeführt werden, wenn ein Fehler bereits aufgetreten ist.[25] Run-To-Failure Maintenance ist somit ein reaktiver Wartungsansatz, da nicht versucht wird, Ausfälle im Voraus zu verhindern.[26] Damit ist Run-To-Failure Maintenance die einfachste, jedoch gleichzeitig die ineffizienteste der drei

19 Krishnamurthy et al. (2005), S. 64.

20 In Anlehnung an Jardine et al. (2006), S. 1484, Krishnamurthy et al. (2005), S. 64.

21 Vgl. Susto et al. (2012), S. 638, Mobley (2002), S. 2 ff., Jardine et al. (2006), S. 1484, Susto et al. (2015), S. 812, Hashemian;Bean (2011), S. 3480.

22 Vgl. Susto et al. (2012), S. 638.

23 ebd. unterteilen diese Kategorie nochmals in Condition Based Maintenance und Predictive Maintenance. Der Unterschied zwischen beiden Kategorien besteht darin, dass bei Predictive Maintenance ein Vorhersagesystem analysiert, ob eine Aktion durchgeführt werden soll. Bei Condition Based Maintenance hingegen liegt die Entscheidung beim Menschen.

24 Vgl. Jardine et al. (2006), S. 1484.

25 Vgl. Susto et al. (2012), S. 638.

26 Vgl. Mobley (2002), S. 2.

Wartungsstrategien.[27] Die Ineffizienz und damit verbunden höheren Kosten sind darin begründet, dass viele Ersatzteile auf Lager gehalten werden müssen, Überstunden für die Behebung des Defektes anfallen und Maschinen eine gewisse Zeit nicht einsatzfähig sind.[28] Unabhängig davon ist der Ansatz aufgrund seiner Einfachheit weit verbreitet.[29]

Einen anderen Ansatz verfolgt *Preventive Maintenance*. Wie der Name bereits sagt, wird bei dieser Wartungsstrategie versucht, den Fehler im Voraus zu vermeiden.[30] Charakteristisch ist, dass ein Wartungsintervall für jede Maschine definiert wird, in welchem die Maschine überprüft und Reparaturen durchgeführt werden. Wartungsaktivitäten werden in zeitlichen Abständen (bspw. alle 6 Monate) oder nach einer bestimmten Anzahl an Prozessiterationen (bspw. gefahrene Kilometer bei Autos) durchgeführt[31] und sind unabhängig vom aktuellen Zustand der Anlage.[32] Vorteil dieser Wartungsstrategie im Vergleich zu Run-To-Failure Maintenance ist, dass Teile der Fehler vermieden werden und die damit verbundenen Kosten somit nicht anfallen. Gleichzeitig kann es jedoch dazu kommen, dass unnötige Wartungen durchgeführt werden, die für den Zustand der Anlage nicht notwendig wären.[33]

Einen weiteren Nachteil bildet der Fakt, dass dieser Ansatz ein Fehlermodell annimmt, bei dem die Fehlerwahrscheinlichkeit von der Betriebszeit abhängt.[34] Nowlan und Heap (1978) konnten hierzu bei einer Studie im Auftrag des U.S. Department of Defense feststellen, dass es grundsätzlich sechs Fehlermodelle gibt, die bei der Betrachtung des Zusammenhangs von Fehlerwahrscheinlichkeit zu Betriebszeit unterschieden werden können. Die Modelle und ihre Häufigkeit in der genannten Studie sind in Abbildung 1 (S. 7) dargestellt. Festzuhalten bleibt, dass lediglich 11% der in der Studie untersuchten Fehler einem Fehlermodell folgen, dessen Fehlerwahrscheinlichkeit mit zunehmendem Alter zunimmt. Bei einem großen Anteil von 89% hingegen ist die Wahrscheinlichkeit eines Fehlers mit zunehmendem Alter konstant.[35] In diesen Fällen wäre die Durchführung einer intervallbasierten Wartung also nicht zielführend.[36]

27 Vgl. Susto et al. (2012), S. 638, Susto et al. (2015), S. 812.

28 Vgl. Mobley (2002), S. 3.

29 Vgl. Susto et al. (2015), S. 812.

30 Vgl. Susto et al. (2012), S. 638.

31 Vgl. Susto et al. (2015), S. 812.

32 Vgl. Jardine et al. (2006), S. 1484.

33 Vgl. Susto et al. (2012), S. 638.

34 Vgl. Hashemian;Bean (2011), S. 3480 f.

35 Vgl. Nowlan;Heap (1978), S. 45 ff.

36 Vgl. Hashemian;Bean (2011), S. 3480 f.

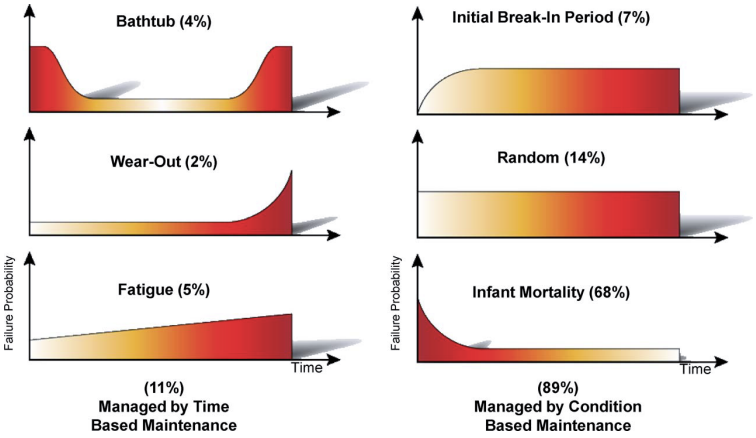

Abbildung 1: Fehlermodelle (Fehlerwahrscheinlichkeit ~ Betriebszeit)[37]

Anstatt also in bestimmten Intervallen Wartungen durchzuführen, verfolgt *Predictive Maintenance* einen anderen Ansatz. Die grundsätzliche Idee dabei ist, Wartungen abhängig vom Anlagenzustand durchzuführen. Es werden also die Zustände der Anlagen beobachtet und Abweichungen, die zu Fehler führen können, im Voraus erkannt.[38] Dies bietet im Vergleich zu den anderen beiden Ansätzen einige zusätzliche Vorteile:

Ein Vorteil von Predictive Maintenance ist, dass die Intervalle zwischen zwei Reparaturen durch die zustandsabhängige Wartung maximiert werden und dementsprechend das Anlagenteil so lang wie möglich genutzt wird. Zusätzlich können Ausfälle der Anlagen durch die Vorhersage von Fehlern vermieden werden.[39] Dies kann wiederum auf Prozessseite zu einer höheren Verfügbarkeit, Effizienz und Sicherheit der Anlagen führen.[40] Zusammenfassend kann gesagt werden, dass Predictive Maintenance die gesamten Wartungskosten reduzieren[41] und damit die Produktivität von technischen Anlagen erhöhen kann[42].

2.1.2 Arten und Verfahren

Predictive Maintenance selbst kann wiederum auf unterschiedliche Art und Weise umgesetzt werden. Die verschiedenen Ansätze können anhand der Datenquelle, der Datenarten, dem Vorhandensein von Fehlerinformationen und der Auswertungstechnik voneinander abgegrenzt

37 ebd., S. 3481. (in Anlehnung an Nowlan;Heap (1978), S. 46).

38 Vgl. Krishnamurthy et al. (2005), S. 64, Jardine et al. (2006), S. 1484.

39 Vgl. Mobley (2002), S. 60.

40 Vgl. Hashemian;Bean (2011), S. 3491.

41 Vgl. Ebd.

42 Vgl. Mobley (2002), S. 4 f.

werden.[43] Die Dimensionen und ihre jeweiligen Ausprägungen sind hierzu in Abbildung 2 dargestellt.

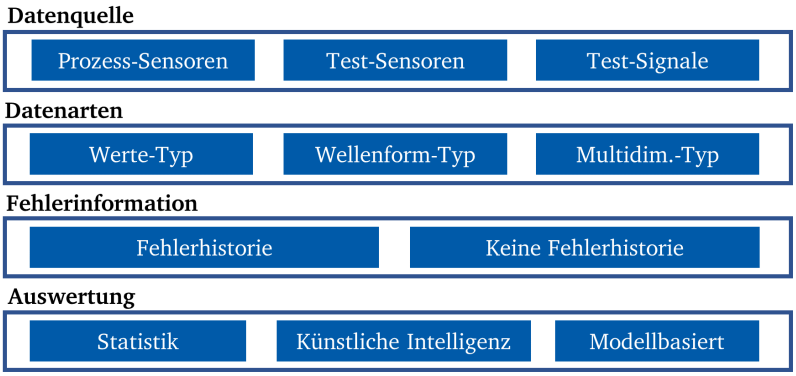

Abbildung 2: Predictive Maintenance Arten[44]

Die ersten Dimensionen unterscheidet Predictive Maintenance Ansätze anhand der *Datenquelle*, die zum Aufnehmen des Systemzustands verwendet wird. Hierbei lassen sich drei Ausprägungen unterscheiden: Prozess-Sensoren, Test-Sensoren und Test-Signale. Bei der Ausprägung „Prozess-Sensoren“ werden als Datengrundlage Messwerte von bereits vorhandenen Sensoren verwendet. Bei der Ausprägung „Test-Sensoren“ hingegen werden zur Umsetzung von Predictive Maintenance zusätzliche Sensoren an der Anlage installiert, um die notwendigen Daten zu erhalten. Während die beiden ersten Ausprägungen passiv den Zustand der Anlage messen, setzt die letzte Ausprägung „Test-Signale“ auf eine direkte, aktive Messung mit Hilfe von Signalen, die direkt in das Anlagenteil eingespeist werden. Um bspw. einen Defekt an einem Kabel zu erkennen, können elektrische Signale durch das Kabel geleitet, die Reflektion des Signals gemessen und hierdurch Defekte erkannt werden. Eine passive Vorgehensweise würde im Gegensatz hierzu bspw. die Temperatur des Kabels messen.[45]

Eine weitere Dimension zur Unterscheidung der Predictive Maintenance Verfahren bildet die *Datenart,* die von den Datenquellen zur Verfügung gestellt wird. Auch hierbei lassen sich wiederum drei Ausprägungen unterscheiden: Werte-Typ, Wellenform-Typ und Multidimensionaler-Typ. Unter Werte-Typ werden Daten verstanden, bei welchen für einen bestimmten Zeitraum ein einzelner Wert vorliegt (bspw. Temperatur). Mit Wellenform-Typ hingegen sind

43 Vgl. Hashemian;Bean (2011), S. 3482 ff., Jardine et al. (2006), S. 1486 ff., Susto et al. (2015), S. 813.

44 Eigene Abbildung auf Grundlage von Hashemian;Bean (2011), S. 3482 ff., Jardine et al. (2006), S. 1486 ff., Susto et al. (2015), S. 813.

45 Vgl. Hashemian;Bean (2011), S. 3482 ff.

Zeitserien in Form von Wellen gemeint (bspw. akustische Daten). Beim multidimensionalen Typ hingegen werden Daten mit mehreren Dimensionen aufgenommen (bspw. Bilddaten).[46]

Neben der Datenquelle und den Datenarten kann Predictive Maintenance in Verfahren aufgeteilt werden, die *Fehlerinformationen* verwenden oder nicht verwenden. Im ersten Fall ist eine Fehlerhistorie vorhanden und wird für Erkennung und Vorhersage von Fehlern genutzt. Beim zweiten Fall sind die Fehlerdaten nicht verfügbar, sodass diese Art von Verfahren rein auf den aufgenommenen Zuständen/Messwerten aufsetzt.[47]

Diese letzte Dimension unterscheidet die Technologien, die zur *Auswertung* der Daten verwendet werden. Hierbei gibt es nach Jardine et al. (2006) grundsätzlich drei Kategorien: Statistik, Künstliche Intelligenz und modellbasierte Verfahren. In der ersten Kategorie werden statistische Methoden verwendet, um Fehler im Voraus zu prognostizieren.[48] Ein Beispiel hierfür zeigen Yan et al. (2004), welche eine logistische Regression und ein ARMA[49]-Modell zur Schätzung der Restlebenszeit verwenden.[50] Unter die zweite Kategorie fallen Verfahren, die Methoden der Künstlichen Intelligenz bzw. des Maschinellen Lernens zur Datenanalyse nutzen.[51] Ein Beispiel hierfür ist die Verwendung von künstlichen neuronalen Netzen.[52] Bei der letzten Kategorie „modellbasierte Verfahren“ wird im Gegensatz zu den vorherigen, rein datengetriebenen Verfahren, ein physikalisches Modell der Anlage erstellt und mit Hilfe dieses Modells Prognosen gegeben. Die Vorhersage basiert somit nicht nur auf Vergangenheitsdaten, sondern umfasst auch eine explizite Modellierung der physikalischen Funktionsweise der Anlage.[53]

2.2 Maschinelles Lernen

Da im Rahmen dieser Masterarbeit Maschinelles Lernen zur Auswertung der Daten verwendet wird, folgt im kommenden Abschnitt ein Überblick über die Grundkonzepte des Maschinellen Lernens.

46 Vgl. Jardine et al. (2006), S. 1486 ff.

47 Vgl. Susto et al. (2015), S. 813.

48 Vgl. Jardine et al. (2006), S. 1491 ff.

49 ARMA = Autoregressive-Moving Average.

50 Vgl. Yan et al. (2004), S. 796 ff.

51 Vgl. Jardine et al. (2006), S. 1491 ff.

52 Vgl. Ebd, Yam et al. (2001), S. 383 ff.

53 Vgl. Roemer et al. (2006), S. 707 ff., Jardine et al. (2006), S. 1494 ff.

2.2.1 Definition, Problemstellung und Ziel

Seit der Erfindung des Computers haben Forscher versucht, die Fähigkeit des Lernens auch auf den Computer zu übertragen.[54] Es bildete sich im Lauf der Jahre ein neues Forschungsfeld namens Maschinelles Lernen als Zweig der Künstlichen Intelligenz.[55] Eine Definition des Begriffes und damit des Forschungsfeldes gibt Samuel (1959), der Maschinelles Lernen definiert als:

> "field of study that gives computers the ability to learn without being explicitly programmed"[56]

Besonders der letzte Teil der Definition ist hierbei entscheidend. Die Fähigkeit eines Computers zu lernen, ist also nur dann gegeben, wenn der Computer Dinge eigenständig, ohne explizite Programmierung von außen, erlernen kann. Die Definition lässt jedoch offen, was genau unter Lernen zu verstehen ist.

Eine Definition, die eine Beschreibung von Lernen beinhaltet und dabei gleichzeitig das Lernproblem formuliert, gibt Mitchell (1997):

> "A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."[57]

Als Beispiel hierfür führt Mitchell (1997) das Erlernen von Schach an, welches die Aufgabe *T* darstellt, die gelernt werden soll. Hierfür ist eine Menge an Trainingserfahrungen *E* gegeben, welche die bereits gespielten Schachpartien umfasst. Die Leistungsmessung *P* ist definiert durch den Prozentsatz der gewonnenen Spiele. Wenn ein Computerprogramm nun mit Hilfe der bereits gespielten Spiele (also seiner Erfahrung *E*) den Prozentsatz der gewonnenen Spiele (also die Leistungsmessung *P*) erhöhen kann, so hat das Programm in Bezug auf das Spielen von Schach (also der Aufgabe *T*) etwas gelernt.[58]

Das Resultat des Maschinellen Lernens ist ein Programm bzw. eine Funktion (auch Modell genannt[59]), welche das gelernte Wissen repräsentiert. Dabei muss bei jedem Lernproblem

---

54 Vgl. Carbonell et al. (1983), S. 3.

55 Vgl. Asthana;Khorana (2013), S. 267.

56 Obwohl dieses Zitat in mehreren Quellen und von verschiedenen Autoren zur Definition/Erklärung von Maschinellem Lernen zitiert wird, ist es sehr schwierig die ursprüngliche Quelle zu finden. Die Autoren verweisen entweder auf Sekundärquellen oder zitieren fälschlicherweise Samuel (1959) . Siehe dazu auch Schuld et al. (2015).

57 Mitchell (1997), S. 2.

58 Vgl. Ebd., S. 2 ff.

59 Vgl. Flach (2012), S. 20 f.

festgelegt werden, welche Art von Funktion gelernt, wie die Funktion repräsentiert und wie die Funktion gelernt werden soll.[60]

Maschinelles Lernen ist eng verknüpft mit dem Begriff des Data Mining.[61] Data Mining kann definiert werden als:

> "The field of data mining addresses the question of how best to use this historical data to discover general regularities and improve the process of making decisions."[62]

Ziel des Data Mining ist es demnach, in bereits existierenden, historischen Daten Muster zu finden, die zukünftige Entscheidungen verbessern können.[63] Die Verknüpfung zwischen Data Mining und Maschinellem Lernen ist aufgrund dessen gegeben, da Data Mining Methoden des Maschinellen Lernens zur Analyse der Daten verwendet.[64]

2.2.2 Überwachtes Lernen

Grundlegend lassen sich nach Russell und Norvig (2012) vier Problemtypen des Machinellen Lernens unterscheiden: Überwachtes Lernen (engl. Supervised Learning), nicht überwachtes Lernen (engl. Unsupervised Learning), halb überwachtes Lernen (engl. Semisupervised Learing) und verstärkendes Lernen (engl. Reinforcement Learning).[65] Im Fokus der Arbeit steht aufgrund der Verfügbarkeit der Fehlerhistorie das überwachte Lernen, welches nun näher erläutert werden soll.

Beim überwachten Lernen besteht die Aufgabe darin, aus Trainingsbeispielen, bestehend aus Eingaben-Ausgaben-Paaren, eine Funktion zu lernen, die eine Zuweisung von Eingaben zu Ausgaben vornimmt.[66] Die Beispiele, auch Instanzen genannt, werden dabei durch eine Menge von Attributen bzw. Features spezifiziert, welche bestimmte Eigenschaften der Instanzen darstellen. Im allgemeinen Fall ist dabei eines der Attribute das Zielattribut (die Ausgabe) und der Rest definiert die Eingabe.[67] Dabei wird das Zielattribut in der Literatur auch als Label bezeichnet.[68] Im Sinne der klassischen Statistik kann sich vorgestellt werden, dass eine bestimmte Anzahl

60 Vgl. Mitchell (1997), S. 2 ff.
61 Vgl. Witten;Frank (2005), S. 4 ff.
62 Mitchell (1999), S. 30.
63 Vgl. Ebd.
64 Vgl. Witten;Frank (2005), S. 4 ff, Fayyad et al. (1996), S. 39.
65 Vgl. Russell;Norvig (2012), S. 811.
66 Vgl. Ebd.
67 Vgl. Witten;Frank (2005), S. 41 ff.
68 Vgl. Chapelle et al. (2006), S. 1.

unabhängiger Variablen einen Einfluss auf eine abhängige Variable besitzt und dieser Einfluss/Zusammenhang gelernt werden soll.[69]

Als Beispiel kann in diesem Zusammenhang das Wetter-Problem verwendet werden. Bei diesem Beispiel ist eine Menge von Instanzen gegeben, die jeweils mit den Eingabe-Attributen Aussicht, Temperatur, Luftfeuchtigkeit und Windstärke versehen sind. Als Zielattribut ist die Information gegeben, ob ein (nicht weiter spezifiziertes) Spiel gespielt wurde oder nicht. Es soll nun als eine Zuweisung gelernt werden, welche gegeben der Aussicht, Temperatur, Luftfeuchtigkeit und Windstärke sagt, ob der Spieler spielen gehen sollte oder nicht.[70]

Das Supervised-Learning Problem kann letztlich wie folgt formalisiert werden: [71]

- **Gegeben:** Trainingsmenge $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ mit n Beispielen, jeweils bestehend aus Eingabe x und Ausgabe y, bei der jedes y durch eine unbekannte Funktion $y = f(x)$ generiert wurde.
- **Gesucht:** Eine Funktion/Hypothese $h(x)$, die $f(x)$ annähert

Dabei wird das Lernproblem je nach dem Wertebereich von y unterschiedlich bezeichnet. Nimmt y Werte einer endlichen Wertemenge (also diskrete Label) an, so liegt ein Klassifizierungsproblem vor. Nimmt y hingegen Werte aus einem kontinuierlichem Wertebereich (bspw. $Y = \mathbb{R}$ oder $Y = \mathbb{R}^d$) an, so wird dies Regressionsproblem genannt.[72]

2.2.3 Prozess und Vorgehensweise

Für die Anwendung von Maschinellem Lernen bzw. Data Mining gibt es verschiedene Vorgehensweisen.[73] Eine dieser Vorgehensweisen ist dabei der Knowledge Discovery in Databases Prozess (KDD-Prozess) von Fayyad et al. (1996), welcher in Abbildung 3 (S. 13) dargestellt ist.[74]

[69] Vgl. Hastie et al. (2009), S. 9.

[70] Vgl. Witten;Frank (2005), S. 10 ff.

[71] Vgl. Russell;Norvig (2012), S. 811 f.

[72] Vgl. Chapelle et al. (2006), S. 1.

[73] Vgl. KDnuggets (2014).

[74] Vgl. Fayyad et al. (1996), S. 37 ff.

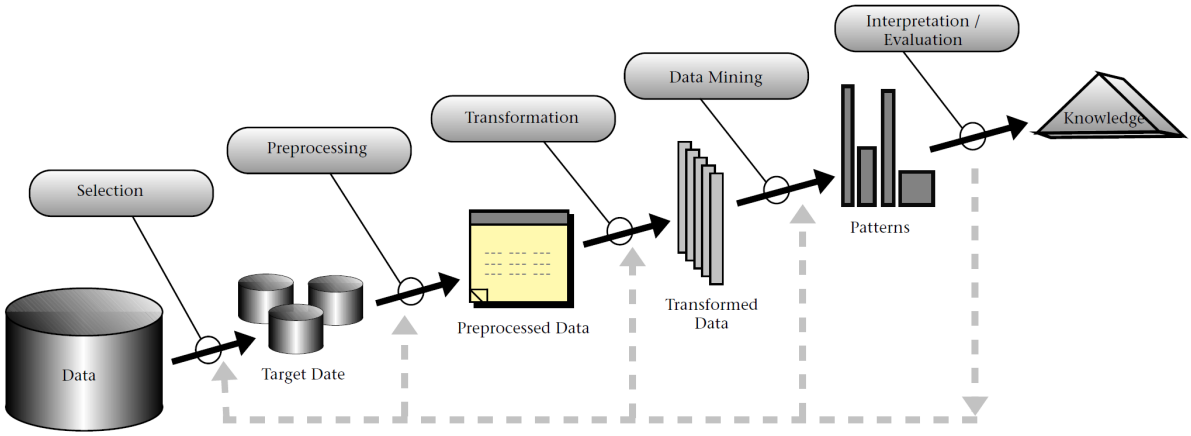

Abbildung 3: KDD-Prozess[75]

Nachdem sich ein grundlegendes Verständnis in der Anwendungsdomäne erarbeitet und das Ziel der Problemstellung definiert wurde, folgt im *ersten Schritt* das Selektieren der Zieldaten. Ziel dieses Schrittes ist es, aus dem existierenden Datenbestand den Teil auszuwählen, der für die Problemstellung relevant ist. Unter Umständen stammen diese Daten dabei aus unterschiedlichen Datenquellen. Dabei findet die Selektion sowohl in Bezug auf die Beispiele als auch auf die möglicherweise relevanten Attribute statt.[76]

Im *zweiten Schritt* folgt die Vorverarbeitung und Datenbereinigung. So werden Messfehler entfernt und fehlende Attributwerte behandelt. Anschließend folgt im *dritten Schritt* die Transformation der Daten, welche vorwiegend aus der Datenreduktion und -projektion besteht. In diesem Schritt können also Dimensionsreduktionverfahren angewandt werden, sodass die Anzahl an Attributen verringert oder Attribute zusammengefasst werden.[77]

Im *vierten Schritt* folgt die Auswahl der Methode, die zur Analyse der Daten verwendet werden soll. So wird beispielsweise bestimmt, ob ein Clustering, eine Klassifizierung oder eine Regression durchgeführt werden soll. Anschließend wird entschieden welcher konkreter Algorithmus hierfür verwendet wird. Dabei muss auch festgelegt werden, welche Parameter des Algorithmus für die konkrete Fragestellung bzw. die konkreten Daten in Frage kommen. Zuletzt wird in diesem Schritt die ausgewählte Methode in Form des ausgewählten Algorithmus ausgeführt. Im letzten, *fünften Schritt* folgt schließlich eine Evaluation und Interpretation der Lernergebnisse.[78]

[75] ebd., S. 41.

[76] Vgl. Ebd., S. 41 f.

[77] Vgl. Ebd.

[78] Vgl. Ebd.

2.3 Lernalgorithmen

Nachdem im vorherigen Abschnitt die Grundkonzepte des Maschinellen Lernens präsentiert wurden, folgt in diesem Abschnitt die Beschreibung der konkret verwendeten Lernalgorithmen. Im Rahmen der Arbeit wurden dabei ausschließlich Algorithmen zur Klassifikation von Beispielen verwendet. Klassifikationsalgorithmen versuchen, wie bereits in Abschnitt 2.2.2 beschrieben, aus den Trainingsbeispielen eine Funktion $y = h(x)$ zu finden, wobei y aus einem diskreten, endlichen Wertebereich stammt.[79] Im Folgenden wird nun auf die in der Arbeit genutzten Klassifikationsalgorithmen J48, JRip, Random Forest und AdaBoost eingegangen.

2.3.1 J48

Der Name J48 bezeichnet eine konkrete, frei verfügbare Implementierung des C4.5 Algorithmus von Quinlan (1993), welcher ein Verfahren darstellt, um Entscheidungsbäume zu lernen.[80] Dabei stellt der C4.5 Algorithmus eine Weiterentwicklung seines Vorgängers, dem ID3 Algorithmus, dar.[81] Um die Vorgehensweise des J48 zu erklären, geht der Abschnitt wie folgt vor: Zunächst wird grundlegend erläutert, wie Entscheidungsbäume definiert sind. Im Anschluss folgt eine Beschreibung, auf welche Weise der ID3 Algorithmus diese erstellt. Zuletzt werden die Verbesserungen des C4.5 Algorithmus im Vergleich zu ID3 dargelegt.

Entscheidungsbäume stellen eine Baumstruktur dar, in denen Instanzen von der Wurzel bis zu einem Blatt eingeordnet werden. Dabei definieren die Knoten mit Ausnahme der Blätter Tests auf bestimmte Attribute. Die nachfolgenden Kanten hingegen korrespondieren zu bestimmten Werten des Attributes, auf die der jeweilige Knoten testet. Die Blätter des Baumes definieren letztlich die Klassen, die in dem dazugehörigen Baumstrang vorhergesagt wird.[82] Ein beispielhafter Entscheidungsbaum ist in Abbildung 4 (S. 15) für das in Abschnitt 2.2.2 eingeführte Wetter-Problem dargestellt.

[79] Siehe Abschnitt 2.2.2.

[80] Vgl. Witten;Frank (2005), S. 373, Quinlan (1993), S. 5.

[81] Vgl. Quinlan (1993), S. 2.

[82] Vgl. Quinlan (1986), S. 86 f., Mitchell (1997), S. 52 f.

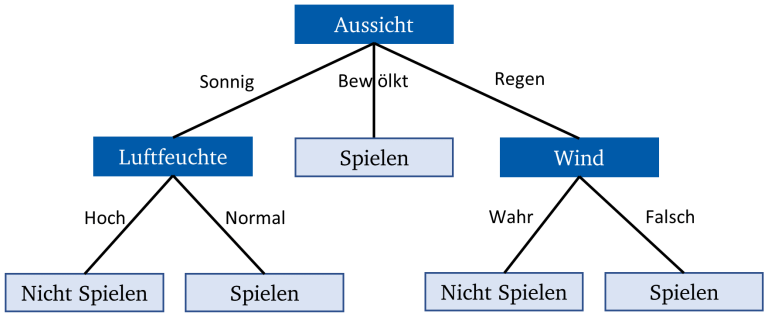

Abbildung 4: Entscheidungsbaum Wetter-Problem[83]

Der ID3 Algorithmus von Quinlan (1986) erstellt solche Entscheidungsbäume mit Hilfe einer Divide-And-Conquer Strategie, wobei er den Baum von oben nach unten aufbaut. Der Algorithmus startet folglich mit dem Erstellen der Wurzel. Er sucht nach einem Attribut bzw. einem Test, der die Menge an Trainingsbeispielen auf Basis eines bestimmten Maßes am besten trennt. Hat er dieses Attribut gefunden, so unterteilt er die Trainingsmenge S in Teilmengen $\{S_1, S_2, \dots, S_v\}$[84], wobei eine Teilmenge S_i alle Trainingsbeispiele enthält, die bei dem ausgewählten Attribut A die Ausprägung A_i besitzen. Enthält eine Teilmenge S_i nur Instanzen einer Klasse, so bildet der ID3-Algorithmus aus diesem Knoten ein Blatt, welcher die besagte Klasse voraussagt. Auf allen anderen Teilmengen S_i wird die beschriebene Vorgehensweise wiederholt. Der Algorithmus endet letztlich, wenn alle Teilmengen ein Blatt darstellen oder wenn keine Attribute mehr zur Verfügung stehen.[85]

Zentral für die Erstellung des Baumes ist dabei die Auswahl des besten Attributes. Hierzu greift Quinlan (1986) auf das Entropie-Maß von Shannon (1948) aus der Informationstheorie zurück. Die Entropie charakterisiert die Unordnung/Ordnung bezüglich der Klassenverteilung in einer Menge von Beispielen, indem sie berechnet, wie viel Information zur Beschreibung der Menge notwendig ist.[86] Die Entropie (einer Menge S) ist dabei wie folgt definiert:[87]

$$E(S) = -\sum_i p_i * \log_2 p_i \qquad (1)$$

[83] In Anlehnung an Quinlan (1986), S. 87.

[84] v = Anzahl der Werte von Attribut A.

[85] Vgl. Quinlan (1986), S. 87 ff., Mitchell (1997), S. 55 ff.

[86] Vgl. Quinlan (1986), S. 89 ff, Shannon (1948), S. 392 ff., Mitchell (1997), S. 55 ff.

[87] Vgl. Shannon (1948), S. 393.

Dabei stellt p_i den Anteil der Klasse i in der Menge S dar. Im Falle einer binären Klassifizierung mit einer positiven Klasse P und einer negativen Klasse N, ist die Entropie somit wie folgt zu berechnen:[88]

$$E(S) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2}$$

Wie zu erkennen ist, stellt $\frac{p}{p+n}$ den Anteil der positiven Klasse in der Menge dar, $\frac{n}{p+n}$ hingegen den Anteil der negativen Klasse. Die Entropie nimmt dabei bei einem positiven Anteil von 0 oder 1 (also im Falle der größtmöglichen Ordnung) den Wert 0 an. Ist hingegen die Verteilung der Klassen 50 % positiv und 50 % negativ (also die Unordnung sehr hoch), so nimmt diese ihren maximalen Wert von 0,5 an. Allgemein ist also die Ordnung in Bezug auf die Klassenverteilung einer Menge umso höher, je kleiner die Entropie ausfällt.[89]

Um nun die Ordnung/Unordnung in Bezug auf die Klassenverteilung bei einem Test auf ein Attribut A zu bestimmen, wird der gewichtete durchschnittliche Entropiewert aller dadurch entstehenden Teilmengen $\{S_1, S_2, \dots, S_v\}$ verwendet. Die Qualität eines Attributs A wird schließlich durch den Information Gain gemessen, der die Reduktion der Entropie nach einer Aufteilung der Menge S durch A darstellt. Der Information Gain ist wie folgt definiert:[90]

$$I(S,A) = E(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} E(S_v) \tag{3}$$

Der linke Teil der Formel stellt die Entropie der ursprünglichen Menge S dar, der rechte Teil hingegen den erwarteten Entropiewert nach einer Aufteilung der Menge durch einen Test auf Attribut A.[91] Zusammengefasst wird also das Attribut ausgewählt, welches den höchsten Information Gain realisiert.[92]

Der ursprüngliche ID3 Algorithmus bildet die Basis für den C4.5 Algorithmus, welche einige neue Features beinhaltet. Die größten Neuerungen bilden dabei die folgenden Punkte: GainRatio als Auswahlkriterium, Zulassung von numerische Attributen, Verarbeitung fehlender Attributwerte und Pruning.[93]

88 Vgl. Quinlan (1986), S. 89 ff, Shannon (1948), S. 392 ff., Mitchell (1997), S. 55 ff.

89 Vgl. Shannon (1948), S. 394, Mitchell (1997), S. 55 ff.

90 Vgl. Quinlan (1986), S. 90, Mitchell (1997), S. 57 ff.

91 Vgl. Mitchell (1997), S. 58.

92 Vgl. Quinlan (1986), S. 90.

93 Vgl. Quinlan (1993), S. 2 ff.

Die Verwendung des Information Gain beim ID3 Algorithmus hat den Nachteil, dass dieser Attribute bevorzugt, welche viele Werte annehmen. Aus diesem Grund wird beim C4.5 Algorithmus der GainRatio zur Auswahl der Attribute verwendet:[94]

$$GainRatio(S,A) = \frac{I(S,A)}{SplitInfo(S,A)} \quad \text{mit} \quad SplitInfo(S,A) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} * \log_2 \frac{|T_i|}{|T|} \qquad (4)$$

Wie zu erkennen ist, wird der Information Gain weiterhin verwendet, jedoch durch den Informationsgehalt der Aufteilung dividiert. Dabei steigt der Wert von SplitInfo je mehr Werte ein Attribut annimmt, sodass Attribute mit vielen Werten tendenziell niedriger bewertet werden.[95]

Um ein numerisches Attribut A verarbeiten zu können, verwendet der C4.5 Algorithmus Tests, die nicht auf Gleichheit prüfen, sondern den Wertebereich anhand eines Schwellenwerts Z in zwei Intervalle unterteilen. Es wird also getestet, ob $A \leq Z$ oder $A > Z$, wobei Z den Schwellenwert darstellt. Dabei werden nur die Werte als Schwellenwerte in Betracht gezogen, die in den Beispieldaten vorhanden sind.[96]

Für die Verwendung des Algorithmus im Falle von fehlenden Attributwerten wurde sowohl die Baumerstellung als auch die Klassifikation angepasst. So werden bei der Baumerstellung zur Berechnung des Information Gain Instanzen ignoriert, bei denen das jeweilige Attribut fehlt. Zur Berechnung des SplitInfo werden die fehlenden Fälle als eigene Gruppe gesehen, wodurch SplitInfo berechnet wird, also ob $n + 1$ Äste erstellt werden würden. Weiterhin werden sogenannte „fractional cases“[97] verwendet. Ist eine zu klassifizierende Instanz bei einem Test angelangt, der auf ein Attribut testet, welches bei der Instanz nicht bekannt ist, so wird die Instanz entsprechend dem Mengenanteil der Attributwerte (bei Baumerstellung) aufgeteilt und alle Äste mit diesem Gewicht weiterverfolgt. Anstelle einer eindeutigen Zuordnung zu einem Ast und somit zu einer Klasse, erhält man hierdurch eine Wahrscheinlichkeitsverteilung über die Klassen. Es wird schließlich die Klasse vorhergesagt, die die höchste Wahrscheinlichkeit besitzt.[98]

Um Overfitting[99], also die Überanpassung an die Trainingsbeispiele zu vermeiden, besitzt der C4.5 Algorithmus die Möglichkeit des Prunings. Dabei setzt der C4.5 Algorithmus auf die

94 Vgl. Ebd., S. 23.

95 Vgl. Ebd.

96 Vgl. Ebd., S. 24 ff.

97 ebd., S. 29.

98 Vgl. Ebd., S. 28 ff.

99 Siehe hierzu auch Abschnitt 2.4.1.

Vereinfachung nach der Erstellung des Baumes (Postpruning). Das Postpruning erfolgt grundlegend über zwei Vereinfachungsoperatoren: Teilbaum Ersetzung und Teilbaum Anhebung. Bei der ersten Methode werden ganze Teile des Entscheidungsbaumes durch ein Blatt ersetzt, welcher die Mehrheitsklasse als Label erhält. Bei der zweiten Methode hingegen werden Teilbäume vollständig durch einen ihrer Äste ersetzt, wodurch Teilbäume in eine andere Baumebene gehoben werden. Dabei werden die Operatoren nur dann angewendet, wenn die Vereinfachung zu einer Minderung der geschätzten Fehlerrate bezogen auf die Gesamtheit aller Instanzen führt. Zur Schätzung der Fehlerrate verwendet der C4.5 Algorithmus eine pessimistische Schätzung, die rein auf den Trainingsbeispielen basiert. Um dies zu erläutern, sei ein Blatt gegeben, bei dem E Instanzen aus N Instanzen fehlklassifiziert wurden. Dies kann als eine Stichprobe mit N Proben angesehen werden, bei dem E mal die Ausprägung „Falsch“ beobachtet wurde. Nun wird, gegeben eines Konfidenzniveaus c, das Konfidenzintervall der Fehlerrate bei einer Binominalverteilung bestimmt. Der C4.5 Algorithmus setzt anschließend die geschätzte Fehlerrate mit dem oberen Limit dieses Konfidenzintervalls gleich (aus diesem Grund ist die Schätzung pessimistisch). Diese Schätzung kann dabei auf Teilbäume/Bäume übertragen werden, indem die geschätzten Fehlerraten der Äste aufsummiert werden. Letztlich kann so ohne Verwendung einer Pruning-Menge entschieden werden, ob ein Teilbaum vereinfacht werden soll oder nicht.[100] Bei der J48-Implementierung gibt es letztlich noch die Möglichkeit einen Schwellenwert m festzulegen, welcher die Anzahl an Instanzen bei einer Aufteilung des Baumes beschränkt. Es werden dabei nur Attributtests erlaubt, aus welchem mindestens zwei Teilmengen entstehen, die mindestens m Instanzen enthalten.[101]

2.3.2 JRip

Der JRip Algorithmus ist eine Implementierung des RIPPER (Repeated Incremental Pruning to Produce Error Reduction) Algorithmus von Cohen (1995), welcher eine Menge von Klassifikationsregeln erzeugt und somit den Regellernern zugeordnet werden kann.[102] Eine Klassifikationsregel besteht dabei aus zwei Teilen: dem Bedingungsteil und dem Folgeteil. Der Bedingungsteil definiert dabei eine Reihe von Tests auf verschiedene Attribute. Der Folgeteil hingegen bestimmt welche Klasse beim Eintreffen der Tests vorhergesagt werden soll. Dabei

100 Vgl. Quinlan (1993), S. 35 ff.

101 Vgl. Witten;Frank (2005), S. 199 u. 406 f.

102 Vgl. Ebd., S. 408 f., Cohen (1995), S. 115 ff.

werden die Tests im Allgemeinen durch logische Konjunktionen verknüpft.[103] Eine Regelmenge für das Wetter-Problem könnte dabei wie folgt aussehen:[104]

- $Aussicht = Sonnig \ \wedge \ Luftfeuchte = Hoch \rightarrow Spielen = Nein$
- $Aussicht = Bewölkt \rightarrow Spielen = Ja$

Der RIPPER Algorithmus erzeugt aus den Trainingsbeispielen eine solche Regelmenge mit dem Ziel, möglichst gut auf andere Beispiele zu generalisieren.[105] RIPPER ist dabei eine Weiterentwicklung des Incremental Reduced Error Pruning (I-REP) Algorithmus von Fürnkranz und Widmer (1994).[106] Die Erstellung der finalen Regelmenge unterteilt sich bei RIPPER bzw. dessen Implementierung JRip grundlegend in eine Aufbau- und Optimierungsphase auf. Dabei wird in der Aufbauphase zunächst eine initiale Regelmenge erzeugt, welche in der Optimierungsphase verbessert wird.[107] In binären Problemstellungen wird dabei wie folgt vorgegangen:

In der *Aufbauphase* werden die einzelnen Regeln nacheinander gelernt. Für jede Regel werden die Beispiele zunächst im Verhältnis 2:1 in eine Wachstums- und eine Pruningmenge aufgeteilt. Anschließend wird mit Hilfe der Wachstumsmenge eine Regel erzeugt. Hierzu werden Bedingungen solange zur Regel hinzugefügt, bis diese keine Beispiele der negativen Klasse mehr abdeckt (sie also zu 100% korrekt ist). Dabei wird jedes Mal die Bedingung hinzugefügt, die den Information Gain auf der Wachstumsmenge maximiert. Im nächsten Schritt wird die erzeugte Regel sofort vereinfacht. Hierzu werden sukzessive Bedingungen am Ende der Regel entfernt, solange sich (bei JRip) folgendes Maß auf der Pruningmenge erhöht:

$$v = \frac{p + 1}{p + n + 2} \tag{5}$$

Dabei steht p für die korrekt abgedeckten Beispiele der Regel und n für die fälschlicherweise abgedeckten Beispiele. Nachdem nun eine Regel erstellt und vereinfacht wurde, werden alle Beispiele aus der Trainingsmenge entfernt, die von dieser abgedeckt sind. Daraufhin folgt die Erzeugung der nächsten Regel. Dies wird solange wiederholt, bis (a) alle positiven Beispiele von der Regelmenge abgedeckt sind, (b) die Description Length (DL) der Regelmenge und Beispiele um mehr als 64 bits größer als die bisher kleinste DL ist oder (c) die Fehlerrate einer

103 Vgl. Witten;Frank (2005), S. 65.

104 Vgl. Ebd., S. 11.

105 Vgl. Cohen (1995), S. 115 ff.

106 Vgl. Fürnkranz;Widmer (1994), S. 70 ff., Cohen (1995), S. 115 ff.

107 Vgl. Cohen (1995), S. 116 ff., Xu;Frank (2016) , Witten;Frank (2005), S. 206 f.

Regel auf der Pruningmenge 50% überschreitet. Das Endresultat dieser Phase ist schließlich eine initiale Regelmenge.[108]

In der *Optimierungsphase* wird diese Regelmenge nun weiter verbessert. Dabei werden für jede Regel zwei Alternativen erzeugt, wobei die Regel in ihrer Erstellungsreihenfolge abgearbeitet werden. Die Trainingsbeispiele werden hierzu erneut in eine Wachstums- und eine Pruningmenge aufgeteilt. Anschließend werden alle Beispiele aus der Pruningmenge entfernt, die durch eine andere als die gerade untersuchte Regel *R* abgedeckt werden. Daraufhin werden mit Hilfe der Wachstums- und Pruningmenge zwei Alternativen für *R* erzeugt. Die eine Alternative ist dabei eine Regel, die komplett neu erstellt wird. Die andere Alternative ist eine Regel, die durch Hinzufügen weiterer Bedingungen zur bisherigen Regel *R* generiert wird. Beide Alternativen werden dabei auf der Pruningmenge vereinfacht. Zuletzt wird die DL der ursprünglichen Regel *R* und der beiden Alternativen berechnet und letztlich die Regel ausgewählt, die die geringste DL besitzt. Sind nun noch positive Beispiele vorhanden, die von keiner Regel abgedeckt werden, so wird zur Aufbauphase gesprungen, um Regeln für diese Beispiele zu erzeugen.[109] Letztlich werden noch die Regeln aus der Regelmenge entfernt, die die DL der gesamten Regelmenge erhöhen. Die resultierenden Regeln bilden die finale Regelmenge.[110]

2.3.3 Random Forest

Random Forest ist eine Ensemble Methode von Breimann (2001), die eine große Anzahl von Entscheidungsbäumen generiert, welche anschließend über die vorherzusagende Klasse per Mehrheitsentscheid abstimmen.[111] Ziel ist es, die Varianz bei gleichbleibendem Bias zu reduzieren.[112]

Dabei ist unter Bias der Fehler zu verstehen, der durch die Vorgehensweise des Lernalgorithmus zustande kommt, der also im Modell des Lernalgorithmus begründet ist. Der Bias hängt also davon ab, wie gut der Lernalgorithmus auf das Lernproblem passt. Dieser Fehler kann somit nicht durch eine größere Anzahl an Trainingsbeispielen reduziert werden. Die Varianz hingegen stellt den Fehler dar, der in einer endlichen Auswahl an Trainingsbeispielen begründet ist, die

[108] Vgl. Cohen (1995), S. 116 ff., Xu;Frank (2016) , Witten;Frank (2005), S. 206 f.

[109] Vgl. Cohen (1995), S. 116 ff., Xu;Frank (2016) , Witten;Frank (2005), S. 206 f.

[110] Vgl. Cohen (1995), S. 116 ff., Xu;Frank (2016) , Witten;Frank (2005), S. 206 f.

[111] Vgl. Breiman (2001), S. 5 f.

[112] Vgl. Hastie et al. (2009), S. 587 ff.

nicht die Gesamtheit aller möglichen Instanzen vollständig repräsentieren.[113] Die Varianz hängt also davon ab, wie sensitiv ein Lernalgorithmus auf kleine Veränderungen der Trainingsmenge reagiert.[114]

Grundlegend basiert Random Forest auf dem Konzept des Baggings.[115] Die Idee von Bagging ist es, die ursprüngliche Trainingsmenge durch randomisiertes Ziehen in B Trainingsmengen aufzuteilen und darauf Klassifizier zu lernen, welche über die Vorhersage abstimmen. Gegeben der Anzahl an Beispielen pro Trainingsmenge N, werden für jede der B Trainingsmengen N Beispiele aus der ursprünglichen Trainingsmenge mit Zurücklegen gezogen. Die resultierenden Mengen sind also Stichproben der ursprünglichen Trainingsmenge und basieren somit auf der gleichen Verteilung. Auf diesen B Trainingsmengen werden anschließend bspw. Entscheidungsbäume gelernt.[116] Der große Vorteil dieses Verfahrens ist, dass der Bias im Vergleich zu einzelnen Bäumen konstant bleibt, während die Varianz durch die Verwendung verschiedener Datensätze reduziert werden kann.[117]

Random Forest erweitert diese Idee durch eine randomisierte Vorauswahl der Variablen bei Erzeugung der Attributtests.[118] In der Lernphase geht der Algorithmus wie in Abbildung 5 vor.

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

Abbildung 5: Pseudocode Random Forest[119]

Wie zu erkennen ist, findet in Schritt 1. b. i. – ii. die randomisierte Vorauswahl von m Variablen aus der Menge aller p Variablen statt. Dies wird durchgeführt um die Korrelation der Trainingsdaten der einzelnen Random Forest Bäume zu minimieren, sodass die Varianz weiter sinken

113 Vgl. Witten;Frank (2005), S. 317 f.

114 Vgl. Sammut;Webb (2010), S. 100 f.

115 Vgl. Hastie et al. (2009), S. 587, Breiman (2001), S. 5 ff.

116 Vgl. Breiman (1996), S. 123 f.

117 Vgl. Hastie et al. (2009), S. 587 f.

118 Vgl. Breiman (2001), S. 5 ff.

119 Hastie et al. (2009), S. 588 (Ausschnitt).

und dadurch die Genauigkeit steigen kann. Random Forest klassifiziert ein Beispiel letztlich, indem er es zunächst von jedem der Bäume klassifizieren lässt. Anschließend werden alle Vorhersagen gesammelt und die Klasse vorgesagt, die von der Mehrheit der Bäume ausgegeben wurde.[120]

2.3.4 AdaBoost

Boosting stellt, wie auch Random Forest, einen Ensemble Ansatz dar, um die Leistung von Algorithmen zu erhöhen. Die Grundidee von Boosting ist es, wiederholt einen schwachen Klassifizierer auf unterschiedlichen Verteilungen der Trainingsmenge laufen zu lassen und die schwachen Klassifizierer anschließend zu einem starken Klassifizierer zu verbinden. Die schwachen Klassifizierer müssen dabei nur etwas besser als zufälliges Raten zu sein.[121]

In der Masterarbeit wird für die Umsetzung von Boosting der AdaBoost Algorithmus von Freund und Schapire (1997) verwendet. Von diesem Algorithmus gibt es verschiedene Versionen, wobei im Folgendem die Variante des AdaBoost.M1 Algorithmus fokussiert wird, der im Gegensatz zum ursprünglichen AdaBoost Algorithmus auch mit Multiklassen Problem umgehen kann.[122] Die Vorgehensweise des Algorithmus ist dabei in Pseudocode-Form in Abbildung 6 abgebildet.

Algorithm AdaBoost.M1
Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$
with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations
Initialize $D_1(i) = 1/m$ for all i.
Do for $t = 1, 2, \ldots, T$:
1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \rightarrow Y$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.
If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t:
$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$
where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:
$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}.$$

Abbildung 6: Pseudocode AdaBoost.M1[123]

120 Vgl. Ebd., S. 587 ff., Breiman (2001), S. 7 f.

121 Vgl. Freund;Schapire (1996), S. 148.

122 Vgl. Freund;Schapire (1997), S. 126 ff.

123 Freund;Schapire (1996), S. 149.

Der Algorithmus erhält eine Menge von m Beispielen $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$, wobei $x_i \in X$ die nicht Klassenattribute und $y_i \in Y$ die Klasse des Beispiels darstellt, welche beim AdaBoost.M1 Algorithmus nicht binär sein muss. Zusätzlich ist ein schwacher Lernalgorithmus gegeben, der im Pseudcode *WeakLearn* genannt wird und eine Anzahl an Iterationen T, die durchlaufen werden sollen. Bevor jedoch die erste Iteration beginnt, wird die Verteilung der Beispiele D mit einer Gleichverteilung initialisiert, indem jedes Beispiel i das Gewicht $\frac{1}{m}$ bekommt.[124]

Anschließend beginnen die Iterationen von AdaBoost, in denen der WeakLearner mit unterschiedlichen Verteilungen D aufgerufen wird. Dabei wird in Iteration t WeakLearn mit der Verteilung D_t aufgerufen. Dieser lernt auf Grundlage dieser Verteilung und der gegebenen Trainingsmenge eine Hypothese $h_t: X \to Y$. Diese Hypothese wird jedoch wahrscheinlich einen Teil der Beispiele falsch klassifizieren. Der Fehler der Hypothese wird dabei durch $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$ bestimmt. Es werden also Gewichte der Beispiele aufsummiert, die die Hypothese h_t falsch klassifiziert hat. Ist der Fehler $\epsilon_t > \frac{1}{2}$, so wird die Schleife abgebrochen, andernfalls folgt nun die Bestimmung der Verteilung D_{t+1} für die nächste Iteration. Hierzu wird zunächst β_t auf Grundlage der Fehlerrate ϵ_t bestimmt. Anschließend wird das Gewicht $D_t(i)$ für jedes korrekt klassifizierte Beispiel i mit $\beta_t \in [0,1)$ multipliziert, während die Gewichte der falsch klassifizierten Beispiele unverändert bleiben. Die neue Verteilung D_{t+1} ergibt sich anschließend, indem die Gewichte mit Hilfe von Z_t normalisiert werden. Hierdurch wird also das Gewicht von falsch klassifizierten Beispielen in der nächsten Iteration angehoben, während das Gewicht von korrekt klassifizierten Beispielen gesenkt wird. Schwer zu klassifizierende Beispiele erhalten also im Laufe der Iterationen immer höhere Gewichte, sodass sich in den späteren Iterationen immer mehr auf diese Fälle konzentriert wird. Dieser Vorgang wird letztlich solange wiederholt, bis eine festgelegte Anzahl an Iteration T erreicht ist.[125]

Die finale, kombinierte Hypothese h_{fin}, welche für die Klassifikation von Beispielen verwendet wird, ist ein gewichtetes Abstimmen der schwachen Hypothesen h_t. Das Gewicht für die Hypothese h_t ist dabei durch $log\frac{1}{\beta_t}$ gegeben, sodass Hypothesen mit höheren Fehlerraten ein geringeres Gewicht erhalten. Die Hypothese h_{fin} wählt letztlich das Label, bei dem die Summe der Gewichte der Hypothesen, die dieses Label vorhersagen, maximiert wird.[126]

[124] Vgl. Ebd., S. 149 f.

[125] Vgl. Ebd.

[126] Vgl. Ebd.

Zusammengefasst lernt AdaBoost.M1 eine kombinierte Hypothese, deren Fehler klein ist, obwohl er auf Basisklassifizieren basiert, deren Fehler möglicherweise hoch ist. Dabei kann gezeigt werden, dass der Fehler auf der Trainingsmenge der finalen Hypothese h_{fin} exponentiell gegen 0 konvergiert, solange der Basisklassifizierer ein bisschen besser als Zufall ist.[127]

Im Rahmen der Masterarbeit wird AdaBoost in Kombination mit dem REPTree verwendet. REPTree ist ein Lernalgorithmus aus der Softwaresuite Weka, der sowohl Entscheidungsbäume als auch Regressionsbäume erstellen kann. Zur Erstellung eines Klassifikationsbaumes verwendet der REPTree, wie der C4.5 Algorithmus, den Information Gain. Beim Pruning hingegen kann das sogenannte Reduced-Error Pruning verwendet werden.[128] Hierbei wird der Baum solange vereinfacht, bis der Fehler auf einer separaten Testmenge nicht ansteigt.[129] Weiterhin ermöglicht der REPTree wie auch der C4.5 Algorithmus die Verwendung von numerischen Attributen und unterstützt fehlende Attributwerte. Letztlich wurde der REPTree als WeakLearner gewählt, da es bei diesem Algorithmus möglich ist, die maximale Baumtiefe über einen Parameter zu beschränken.[130]

2.4 Evaluierung

Nachdem nun die in der Masterarbeit verwendeten Algorithmen vorgestellt wurden, wird in diesem Abschnitt die Evaluierung von Algorithmen erläutert. Hierzu wird im ersten Unterabschnitt zunächst die allgemeine Vorgehensweise beschrieben. Im zweiten Unterabschnitt folgt anschließend eine Darstellung der verwendeten Evaluierungsmaße.

2.4.1 Vorgehensweise und Datenaufteilung

Zur Evaluierung der Algorithmen werden die verfügbaren Daten im Allgemeinen in zwei Teile aufgeteilt: eine Trainingsmenge und eine Testmenge. Die Trainingsmenge ist dabei die Menge, die der Algorithmus zum Lernen erhält. Auf dieser Grundlage erstellt der Lernalgorithmus das Modell. Die Testmenge hingegen wird dazu verwendet, um die Vorhersagequalität des gelernten Modells auf Daten zu testen, die der Algorithmus zuvor noch nicht gesehen hat. Diese Aufteilung ist notwendig, da das Messen der Fehlerrate auf den Trainingsdaten im Allgemeinen

127 Vgl. Ebd.

128 Vgl. Witten;Frank (2005), S. 407 f.

129 Vgl. Quinlan (1999), S. 501 f.

130 Vgl. Witten;Frank (2005), S. 407 f.

keine gute Schätzung für die zukünftige Vorhersagequalität des Algorithmus ist.[131] Dies wird klar, wenn man beispielweise Entscheidungsbäume betrachtet. Es ist relativ einfach, den Entscheidungsbaum exakt an die Trainingsdaten anzupassen, indem für jedes Beispiel ein Strang im Baum gebildet wird. Der gelernte Entscheidungsbaum würde in diesem Fall keines der Beispiele falsch klassifizieren. Würde dieser Entscheidungsbaum jedoch mit komplett neuen Daten konfrontiert werden, so wäre der Fehler aufgrund des geringen Generalisierungsgrades relativ hoch. Es ist folglich zentral, dass die Testdaten in keiner Weise zum Trainieren des Algorithmus bzw. des Modells verwendet werden.[132]

Bei manchen Lernverfahren müssen einige Parameter an das Problem angepasst werden. Beim C4.5 Algorithmus ist dies beispielsweise das Konfidenzlevel c, welches beim Postpruning des Baumes verwendet wird.[133] In solchen Fällen wird eine dritte Menge verwendet, die Validierungsmenge genannt wird. Die Validierungsmenge wird dazu benutzt, die optimale Parameterkonfiguration oder den besten Klassifizierer (bei mehreren Algorithmen) auszuwählen. Die Validierungsmenge muss dabei, wie die anderen Mengen auch, unabhängig von den anderen Mengen sein. Dabei wird wie folgt vorgegangen: Zunächst wird auf den Trainingsdaten mit verschiedenen Parameterkonstellationen bzw. Lernalgorithmen gelernt. Anschließend wird die Leistung der verschiedenen Parameterkonstellationen bzw. Lernalgorithmen auf der Validierungsmenge gemessen und davon die beste Konfiguration ausgewählt. Um letztlich eine verlässliche Abschätzung der Fehlerrate zu erhalten, wird die finale, optimierte Methode auf der Testmenge angewandt.[134]

Eine Möglichkeit die verfügbaren Daten möglichst gut zu nutzen, ist die Durchführung einer Kreuzvalidierung. Hierbei werden die Daten in k verschiedene Teilmengen aufgeteilt, wobei k häufig auf 10 gesetzt wird. Nun wird auf $k - 1$ der Teilmengen gelernt, während die Fehlerrate auf der übrig gebliebenen Teilmenge bestimmt wird. Anschließend wird eine andere Teilmenge als Testmenge gewählt. Dies wird solange wiederholt, bis auf allen Teilmengen einmal getestet und $(k - 1)$-mal gelernt wurde. Bei einer stratifizierten Kreuzvalidierung erfolgt die Aufteilung dabei so, dass die Klassenverteilung der einzelnen Teilmengen der Klassenverteilung der gesamten Menge folgt und somit in jeder Teilmenge ungefähr gleich ist.[135] Die Vorgehensweise einer Kreuzvalidierung mit $k = 5$ ist in Abbildung 7 (S. 26) dargestellt.

131 Vgl. Ebd., S. 145 ff.

132 Vgl. Ebd.

133 Siehe Abschnitt 2.3.1.

134 Vgl. Witten;Frank (2005), S. 146.

135 Vgl. Ebd., S. 149 ff.

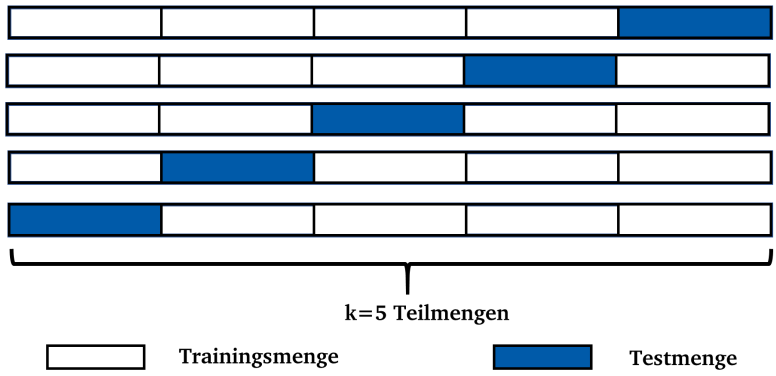

Abbildung 7: Kreuzvalidierung

Die Kreuzvalidierung wird unter anderem dazu verwendet, die optimale Parameterkonstellation bzw. den besten Algorithmus zu finden, wobei der Fehler auf der Kreuzvalidierung in diesem Fall (im Allgemeinen) keine gute Abschätzung für den echten Fehler auf neuen Daten darstellt. In diesem Fall wird die Menge aller Beispiele also zunächst in eine Trainingsmenge und Testmenge aufgeteilt und anschließend auf der Trainingsmenge eine Kreuzvalidierung durchgeführt, sodass eine separate Validierungsmenge nicht benötigt wird.[136]

2.4.2 Evaluierungsmaße

Nachdem im vorherigen Abschnitt erläutert wurde, wie die Daten bei der Evaluierung aufgeteilt werden und welche Aufgaben die jeweiligen Mengen erfüllen, folgt nun die Beschreibung der in der Arbeit verwendet Evaluierungsmaße.

Bei Klassifikationsproblemen kann die Klassifikationsleistung in einer Konfusionsmatrix abgebildet werden, welche die Grundlage für einige der Evaluierungsmaße bildet.[137] Die grundlegende Form der Konfusionsmatrix im binären Fall {positiv, negativ} ist dabei in Abbildung 8 (S. 27) dargestellt.

[136] Vgl. Varma;Simon (2006), S. 91:1 ff.

[137] Vgl. He;Garcia (2009), S. 1276.

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	P'	N'	$P + N$

Abbildung 8: Konfusionsmatrix[138]

Die Matrix wird erstellt, indem für jedes Beispiel die vorhergesagte Klasse mit der realen Klasse verglichen wird. Die Anzahl der Beispiele in den verschiedenen Kombinationen von Vorhersage und realer Klasse werden letztlich in die 2x2 Matrix abgetragen und bilden damit verschiedene Basismaße:[139]

- True Positives (TP): Anzahl der positiven Beispiele, die richtigerweise als positiv klassifiziert wurden
- True Negatives (TN): Anzahl der negativen Beispiele, die richtigerweise als negativ klassifiziert wurden
- False Positives (FP): Anzahl der negativen Beispiele, die fälschlicherweise als positiv klassifiziert wurden
- False Negatives (FN): Anzahl der positiven Beispiele, die fälschlicherweise als negativ klassifiziert wurden

Auf Grundlage dieser Konfusionsmatrix können nun verschiedene Evaluationsmaße definiert werden.[140] Eines der bekanntesten Maße ist dabei die Accuracy (dt. Korrektklassifizierungsrate). Die Accuracy misst den Anteil der Beispiele, die korrekt klassifiziert wurden. Formal ist sie also wie folgt definiert:[141]

$$Accuracy = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

Dabei ist die Accuracy am effektivsten, wenn die Klassen relativ gleichverteilt sind.[142] Sind die Klassen hingegen sehr ungleich verteilt, so kann ein hoher Wert erreicht werden, indem immer die Mehrheitsklasse vorhergesagt wird. Wird nun den Anteil der Beispiele der Mehrheitsklasse

138 Han et al. (2012), S. 366.

139 Vgl. Ebd., S. 365 f.

140 Vgl. Fawcett (2006), S. 862.

141 Vgl. Han et al. (2012), S. 366 f.

142 Vgl. Ebd., S. 367.

erhöht, so steigt die Accuracy an, obwohl sich an die Leistung des Klassifizierers nichts verändert hat.[143]

Gerade für Probleme mit ungleicher Klassenverteilung werden von der Forschungsgemeinschaft deshalb alternative Evaluierungsmaße verwendet: Precision, Recall und F-Measure. Diese werden wie folgt berechnet:[144]

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN} \quad F_\beta = \frac{(1+\beta)^2 * Recall * Precision}{\beta^2 * Recall + Precision} \tag{7}$$

Dabei kann die Precision als ein Maß der Exaktheit verstanden werden. Sie definiert den Anteil der wirklich positiven Beispiele unter allen Beispielen, die als positiv klassifiziert wurden. Der Recall hingegen kann als Maß der Vollständigkeit interpretiert werden. Er misst, wie viele der in Wirklichkeit positiven Beispiele auch als positiv klassifiziert wurden. Dabei ist nur die Precision abhängig von der Klassenverteilung, während der Recall unabhängig von dieser ist. Eine Bewertung nur auf Grundlage des Recall zu machen ist jedoch nicht zielführend, da diese die FP nicht mit einbezieht. Die alleinige Betrachtung der Precision besitzt umgekehrt das Problem, dass die FN nicht mit einbezieht. Aus diesem Grund kombiniert die F-Measure die beiden Maße. Dabei kann über den Parameter β geregelt werden, welcher der beiden Maße mehr Gewichtung bekommt. Im Normalfall wird $\beta = 1$ gesetzt.[145]

Für Multiklassen-Probleme (mit l Klassen) existieren Generalisierungen der binären Maße. Hierbei steht TP_i für die True Positives, FP_i für die False Positives, FN_i die False Negatives und TN_i für die True Negatives der i-ten Klassen. Es wird also für jede Klasse eine binäre Konfusionsmatrix erstellt, wobei die i-te Klasse als positiv und alle anderen als negativ interpretiert werden. Auf Grundlage dieser Konfusionsmatrizen sind folgende Multiklassen-Maße definiert:[146]

$$Accuracy = \frac{\sum_{i=1}^{l} \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}}{l} \tag{8}$$

$$Precision_\mu = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FP_i)} \qquad Recall_\mu = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FN_i)}$$

143 Vgl. He;Garcia (2009), S. 1276 f.

144 Vgl. Ebd., S. 1277.

145 Vgl. Ebd.

146 Vgl. Sokolova;Lapalme (2009), S. 430.

$$Precision_M = \frac{\sum_{i=1}^{l} \frac{TP_i}{(TP_i + FP_i)}}{l} \qquad Recall_M = \frac{\sum_{i=1}^{l} \frac{TP_i}{(TP_i + FN_i)}}{l}$$

$$F_\mu = \frac{(1+\beta)^2 * Recall_\mu * Precision_\mu}{\beta^2 * Recall_\mu + Precision_\mu} \qquad F_M = \frac{(1+\beta)^2 * Recall_M * Precision_M}{\beta^2 * Recall_M + Precision_M}$$

Die Accuracy ist, wie zu erkennen, definiert als die durchschnittliche Accuracy über alle Klassen. Bei Precision und Recall hingegen gibt es nun zwei verschiedene Arten: Mikro μ und Makro M. Die Mikro Maße summieren die Basismaße über alle Klassen zunächst auf und berechnen darauf die Precision bzw. den Recall. Die Makro Maße hingegen berechnen zuerst für jede Klasse die Precision bzw. den Recall und bilden dann den Mittelwert davon. Auf diesen beiden Arten wird letztlich auch die Mikro und Makro F-Measure definiert.[147]

2.5 Softwaresuite Weka

Das Waikato Environment for Knowledge Analysis, kurz Weka, ist eine frei verfügbare Sammlung von Algorithmen des Maschinellen Lernens und der Datenvorverarbeitung. Weka beinhaltet viele Algorithmen aus den Bereichen Regression, Klassifikation, Clustering, Assoziationslernen und Attributselektion und besitzt deshalb sowohl bei Forschern als auch Praktikern große Akzeptanz als Software zur Durchführung von Data Mining.[148] Neben den Algorithmen und Vorverarbeitungsverfahren besitzt Weka die Funktionalität die Algorithmen zu evaluieren und Ergebnisse zu visualisieren.[149] Dabei ist Weka komplett in Java geschrieben und somit unter Windows, Linux und Macintosh lauffähig.[150]

Auch bringt Weka verschiedene graphische Oberflächen mit. Dies ermöglicht es auch Personen ohne Programmierkenntnisse Datenanalysen durchzuführen.[151] Im Fokus der Masterarbeit steht jedoch der Kern in Form einer Java Bibliothek, welche es erlaubt, die Algorithmen und Funktionalitäten von Weka in einer eigenen Anwendung zu nutzen.[152]

147 Vgl. Ebd.

148 Vgl. Hall et al. (2009), S. 10.

149 Vgl. Ebd.

150 Vgl. Witten;Frank (2005), S. 366 ff.

151 Vgl. Hall et al. (2009), S. 10 ff.

152 Vgl. Frank et al. (2016), S. 107 ff.

2.6 Verwandte Forschungsarbeiten

Die Masterarbeit reiht sich in eine Vielzahl von Forschungsarbeiten ein, welche Maschinelles Lernen für die Umsetzung von Predictive Maintenance verwenden. Eine Auswahl dieser Forschungsarbeiten soll im Folgendem vorgestellt werden.

Verwandt in Bezug auf die Verwendung von Predictive Maintenance bei Zügen ist Yang und Létourneau (2005). Diese versuchten mit Hilfe historischer Sensor- und Wartungsdaten der kanadischen Eisenbahn ein Modell zu lernen, das Defekte von Zugrädern vorhersagen kann. Die Zustandsdaten wurden dabei von Sensoren an der Strecke aufgenommen, welche beim Überfahren eines Zuges Informationen über den Zustand der Räder ermittelten und diese an eine zentrale Datenbank schickten. Für das Lernen des Vorhersagemodells wurden verschiedene Schritte durchlaufen. Im ersten Schritt wurden die Daten mit einer Klasse versehen, indem alle Messungen in einem bestimmten Zeitintervall vor dem Fehler die Klasse positiv erhielten. Im nächsten Schritt wurden aus den bestehenden Daten einige neue Attribute, die zeitliche Trends abbilden, generiert. Anschließend wurde das Vorhersagemodell mit Hilfe der Algorithmen C4.5 und Naive Bayes gelernt. Die erzeugten Modelle wurden daraufhin mit Hilfe einer Belohnungsfunktion, welche die Vorhersage der positiven Klasse zu verschiedenen Zeitpunkten vor dem Fehler unterschiedlich bewertet, evaluiert. Die Ergebnisse zeigen, dass die meisten relevanten Fehler bei einer geringen Rate von falschen Alarmen vorhergesagt werden konnten.[153] Dabei basierte die Forschungsarbeit auf einer zuvor veröffentlichen Arbeit von Létourneau et al. (1999), welche eine ähnliche Vorgehensweise zur Fehlervorhersage bei Flugzeugen des Typs Airbus A 320 verwendeten.[154]

Neben dieser Studie gibt es weitere Arbeiten mit Bezug zum Zugverkehr. So versuchte Yilboga et al. (2010) mit Hilfe von zeitverzögerten neuronalen Netzen die restliche Lebenszeit von Eisenbahnweichen vorherzusagen.[155] Li et al. (2014) hingegen wendete Support Vektor Maschinen und Entscheidungsbaum Algorithmen an, um gleich mehrere Fehler im Bahnverkehr der Vereinigten Staaten in Voraus zu erkennen. Als Datenbasis nutzen Li et al. (2014) dabei eine Kombination verschiedener Datenquellen. So wurden Messungen von Streckendetektoren, Fehler- und Inspektionshistorie, Zug- und Wetterdaten bei der Erstellung der Vorhersagemodelle einbezogen.[156]

153 Vgl. Yang;Létourneau (2005), S. 516 ff.

154 Vgl. Létourneau et al. (1999), S. 59 ff.

155 Vgl. Yilboga et al. (2010), S. 134 ff.

156 Vgl. Li et al. (2014), S. 17 ff.

Eng verwandt mit der vorliegenden Masterarbeit sind auch die Studien von Prytz et al. (2013) und Prytz et al. (2015). In diesen Studien wurde in Kooperation mit dem Automobilhersteller Volvo versucht, Defekte des Druckluftkompressors von Lastwagen vorherzusagen. Dabei dienten als Datenquellen vorwiegend die Logdaten der Fahrzeuge, die aggregierte Informationen über die Fahrzeugnutzung beinhalten und bei jedem Werkstattbesuch von den Servicekräften heruntergeladen wurden. In Kombination mit den Service- und Fahrzeugbasisdaten wurde schließlich ein Modell zur Vorhersage von Kompressor-Defekten gelernt. Dabei wurden die Lernalgorithmen k-Nearest-Neighbour, der Entscheidungsbaumlerner C5.0 und Random Forest verwendet. Evaluiert wurden die Modelle letztlich durch ein Kostenmodell, welche den Nutzen bzw. die Kosten des Modells mit Hilfe der Konfusionsmatrix berechnete.[157]

Weitere verwandte Arbeiten bilden Yann-Chang (2003), Salfner und Malek (2007) und Moura et al. (2011). Yann-Chang (2003) versuchte mit Hilfe von evolutionären neuronalen Netzwerken Fehler von Stromtransformatoren auf Grundlage der Gaskonzentration im Voraus zu erkennen.[158] Salfner und Malek (2007) hingegen verwendeten Eventdaten eines Telekommunikationssystems zur Erstellung eines Hidden Semi-Markov Models mit dem Ziel, Systemfehler vorherzusagen.[159] Moura et al. (2011) nutze letztlich Support Vektor Maschinen, um die Zeit/Strecke zum Fehler von Turboladern in Dieselmotoren abzuschätzen.[160]

[157] Vgl. Prytz et al. (2013), S. 118 ff., Prytz et al. (2015), S. 139 ff.

[158] Vgl. Yann-Chang (2003), S. 843 ff.

[159] Vgl. Salfner;Malek (2007), S. 161 ff.

[160] Vgl. Moura et al. (2011), S. 1527 ff.

3 Datensatz und Framework

Nachdem im vorherigen Kapitel die theoretischen Grundlagen gelegt wurden, folgt nun die Beschreibung des Datensatzes und der bisherigen Vorgehensweise zur Erstellung eines Vorhersagemodells von Kauschke et al. (2016). Im ersten Abschnitt werden dabei zunächst die Rohdaten beschrieben. Es wird erläutert, woher die Daten stammen und in welchem Format diese vorliegen. Im zweiten Abschnitt folgt die Beschreibung der bisherigen Vorgehensweise von Kauschke et al. (2016) für die Vorhersage von Stromrichterschäden. Dies bildet letztlich das grundlegende Framework, auf dem die Arbeit für die Beantwortung der Forschungsfragen aufbaut.

3.1 Datensatz

Zur Erstellung eines Vorhersagemodells wurde von der DB Cargo ein historischer Datensatz der Baureihe 185 in Form von Log-Dateien bereitgestellt. Die Baureihe 185 ist ein in den 90er Jahren hergestellter Lokomotiven-Typ vom Hersteller Bombardier, welcher bei der DB Cargo im Güterverkehr eingesetzt wird.[161] Dieser Lokomotiven-Typ besitzt ein System, welches Events der verschiedenen Teilsysteme der Lokomotive in Form von Diagnosemeldungen in einer Logdatei abspeichert.[162] Dabei wird die Aufzeichnung nur beim Auftreten der Events durchgeführt, da zur damaligen Zeit die Speichergröße eine entscheidende Limitation darstellte. Wichtig dabei ist zu beachten, dass das System im ursprünglichen Sinne nicht für die Verwendung im Rahmen eines Predictive Maintenance Systems gedacht war. So werden die Logdateien manuell bei jedem Werkstattbesuch von einem Mitarbeiter ausgelesen und anschließend in das System der DB Cargo übertragen.[163]

Die Diagnosemeldungen können Zustandsinformationen, Warnungen oder Fehler darstellen. Insgesamt gibt es 6909 verschiedene Diagnosemeldungen, wobei manche sehr häufig, andere jedoch so gut wie nie auftreten. Jede Diagnosemeldung spezifiziert das aufgetretene Event durch einen spezifischen Code.[164] So steht der Code 1412 beispielsweise für „Batteriespannung unter 77V“ oder der Code 1516 für „Fahrmotorlüfter 1 aus“. Zusätzlich zur Information, welches Event gerade aufgetreten ist, enthalten die Diagnosemeldungen einen Zeitstempel für Start-

[161] Vgl. Kauschke et al. (2015), S. 122 f, Kauschke (2014), S. 9.

[162] Vgl. Kauschke et al. (2016), S. 152.

[163] Vgl. Kauschke et al. (2015), S. 123.

[164] Vgl. Kauschke et al. (2016), S. 152 f.

und Endzeit. So wird die Zeitspanne vom Auftreten bis zum Verschwinden des jeweiligen Events definiert.[165]

Neben dem Diagnosecode und den Zeitstempel werden an jede Diagnosemeldung zwei Zeichenketten angehängt, die diverse Umfelddaten in einer codierten Form umfassen.[166] Unter den Umfelddaten werden Zusatzinformationen zum konkreten Diagnosecode wie beispielsweise die aktuelle Motortemperatur oder der Kühlmitteldruck gesammelt. Insgesamt gibt es 2291 verschiedene Umfelddaten, wobei für jeden Diagnosecode nur eine spezifische Auswahl der Umfelddaten abgespeichert wird. Dabei wird sich auf die Umfelddaten beschränkt, die für den jeweiligen Diagnosecode von Relevanz sind.[167] Eine der Zeichenketten umfasst dabei die Messwerte beim Auftreten, die anderen hingegen die Messwerte beim Verschwinden des jeweiligen Diagnosecodes.[168] Letztlich sind somit nicht alle Umfelddaten zur jeder Zeit vorhanden. Beispielweise kann es so vorkommen, dass keinerlei Messwerte der Motortemperatur über einen Zeitraum von mehreren Stunden vorliegen.[169]

Zusammenfassend ist in Tabelle 1 eine beispielhafte Diagnosemeldung abgebildet. Insgesamt existieren ca. 32 Millionen solcher Einträge. Diese stammen von 401 Lokomotiven und umfassen einen Zeitraum von 3 Jahren.

Tabelle 1: Beispiel Diagnosemeldung[170]

Code	Startdatum	Enddatum	Umfeld Start	Umfeld Ende
3984	1355556098	1355556368	96F80E00500085010D000 0CE0000000045122411	A5F80E00500087020D000 0CE140000004F153611

3.2 Grundlegendes Framework

Auf Grundlage dieser Rohdaten wurde nun von Kauschke et al. (2016) ein Predictive Maintenance System erstellt, welches Stromrichterausfälle vorhersagen soll.[171] Die Grundstruktur dieses Predictive Maintenance System soll im Folgenden erläutert werden.

165 Vgl. Ebd., S. 153.

166 Vgl. Kauschke (2014), S. 10.

167 Vgl. Kauschke et al. (2016), S. 153.

168 Vgl. Kauschke (2014), S. 10.

169 Vgl. Kauschke et al. (2016), S. 153.

170 In Anlehnung an ebd.

171 ebd., S. 151 ff.

3.2.1 Datentransformation und Attributauswahl

Kauschke et al. (2016) erstellen auf Grundlage der Rohdaten eine Menge von Instanzen. Dabei wird immer dann eine Instanz erzeugt, wenn sich am Zustand der Lok etwas verändert. Eine Instanz charakterisiert dabei den Zustand der Lok zu diesem Zeitpunkt. Sie besitzt so für jeden betrachteten Diagnosecode ein boolesches Attribut, welches kennzeichnet, ob der Code zum jeweiligen Zeitpunkt angelegen hat oder nicht. Zudem besteht sie aus numerischen Attributen, welche die Umfelddaten zum jeweiligen Zeitpunkt darstellen.[172]

Ein Beispiel ist dabei in Tabelle 2 abgebildet. Wie zu erkennen ist, existieren für die Zeitpunkte 12:01:10 Uhr und 12:01:20 Uhr zwei Instanzen. Die ersten Attribute codeX enthalten die Informationen über die Diagnosecodes. Dabei bedeutet 1, dass Code angelegen, und 0, dass der Code zu diesem Zeitpunkt nicht angelegen hat. Nach den Diagnosecodes folgen die Attribute umfeldX, die die Werte der Umfelddaten enthalten. Da wie in Abschnitt 3.1 nicht zu jedem Zeitpunkt alle Umfeldmessungen zur Verfügung stehen, fehlen einige Werte, welche in der Tabelle mit „?“ markiert wurden.

Tabelle 2: Beispiel Instanz[173]

Instanz/Attribut	code1	code2	…	codeN	umfeld1	…	umfeldM
Instanz 12.01.14 12:01:10	1	0	…	1	12.5 °C	…	?
Instanz 12.01.14 12:01:20	1	0	…	0	?	…	10,1 °C

Aufgrund der großen Anzahl unterschiedlicher Diagnosecodes (insgesamt 6909) und Umfelddaten (insgesamt 2291) wurde eine Attributauswahl durchgeführt, um die initiale Anzahl an Dimensionen zu reduzieren. Hierzu wurden Experten befragt, die eine finale Auswahl an relevanten Attributen für das Problem trafen. Das zentrale Entscheidungskriterium für die Aufnahme war dabei, ob das Attribut ein Indikator für einen Stromrichterausfall darstellen kann oder nicht.[174]

Jede Instanz lässt sich letztlich einer Tour zuordnen, wobei die Anzahl bzw. zeitliche Dichte der Instanzen pro Tour variiert. Neben der Auswahl an Attributen wurde auch hier eine Auswahl getroffen. Dabei wurden nur Instanzen ausgewählt, die innerhalb einer Tour liegen, welche mindestens 100km umfasst und zwischen 2 und 48 Stunden angedauert hat. Somit werden nur

172 Vgl. Ebd., S. 153 f.

173 In Anlehnung an ebd., S. 154.

174 Vgl. Ebd., S. 153 f.

Touren mit normaler Länge betrachtet, sodass bspw. Rangierfahrten in Bahnhöfen nicht einbezogen werden.[175]

3.2.2 Labeln der Instanzen

Für das Labeln der Instanzen wird von Kauschke et al. (2016) auf Létourneau et al. (1999) zurückgegriffen. Die Idee von Létourneau et al. (1999) besteht darin, Instanzen in einem bestimmten Zeitfenster *w* (Warnungszeitraum) vor Auftreten des Fehlers mit der Klasse „positiv“ und alle anderen Fehler mit der Klasse „negativ“ zu versehen. Die Annahme dabei ist, dass sich ein Muster vor Auftreten des Fehlers innerhalb dieses Zeitfensters *w* zeigt, sodass dieser vor dessen eigentlichen Auftreten erkannt werden kann.[176]

Um diesen Ansatz jedoch durchführen zu können, ist es notwendig, die genauen Zeitpunkte zu identifizieren, zu denen der Stromrichterausfall bei der jeweiligen Lokomotive aufgetreten ist. Hierzu wurden Experten befragt, die auf Grundlage der Werkstatthistorie und weiterer Daten, 40 Touren (im folgenden Fehlertouren genannt) identifizieren konnten, bei denen ein Stromrichterausfall aufgetreten war.[177] Zudem konnten die Experten eine Menge von Diagnosecodes nennen, die auftreten, wenn der Stromrichter bereits ausgefallen ist. Auf Grundlage dieser Informationen ist es somit möglich, den genauen Zeitpunkt des Defektes innerhalb der Fehlertouren zu identifizieren.

Letztlich werden also alle Instanzen, die in einem bestimmten Zeitraum *w* vor dem Fehler liegen mit der positiven Klasse, alle anderen mit der negativen Klasse versehen. Beispielhaft sind in Abbildung 9 alle Instanzen einer Fehlertour mit ihren jeweiligen Klassen auf einer Zeitachse abgebildet.

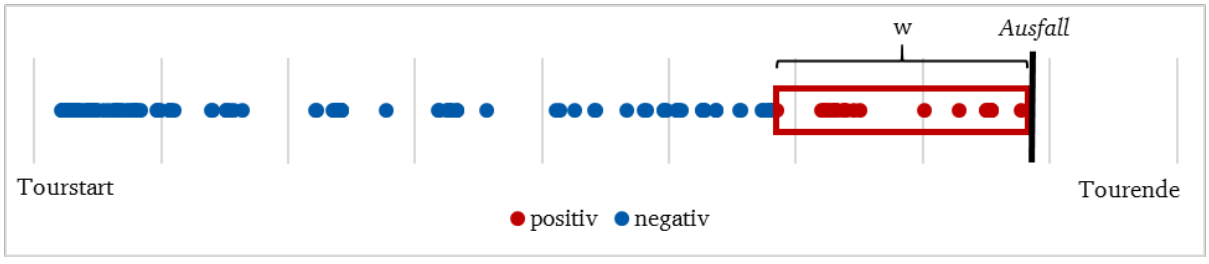

Abbildung 9: Klassen der Instanzen (innerhalb einer Fehlertour)[178]

175 Vgl. Ebd., S. 153 ff.

176 Vgl. Létourneau et al. (1999), S. 62, Kauschke et al. (2016), S. 156 f.

177 Vgl. Kauschke et al. (2016), S. 154 f.

178 In Anlehnung an ebd., S. 156.

Das optimale Zeitfenster muss dabei experimentell bestimmt werden, d.h. ähnlich wie die Parameter der Algorithmen (siehe Abschnitt 2.4.1) muss dieses optimiert werden.[179] Dabei gilt: Je größer das Zeitfenster *w*, desto mehr Instanzen erhalten ein positives Label, aber desto mehr Instanzen, die eigentlich normale Zustände wiederspiegeln, können fälschlicherweise mit der Klasse „positiv“ versehen werden.

3.2.3 Klassifizierung

Auf Grundlage der mit Klassen versehenen Instanzen und der Fehler-/Normaltouren wird nun von Kauschke et al. (2016) ein Klassifikationsmodell gelernt, welche sich in Instanz-Ebene und Tour-Ebene aufteilt. Die Ebenen unterscheiden sich dabei wie folgt: Auf der Instanz-Ebene wird ein Modell gelernt, welche die positiven von den negativen Instanzen möglichst gut trennt. Ziel ist es also, eine neue Instanz, welche die Zustandsinformationen der Lok zu einem bestimmten Zeitpunkt erhält, als positiv oder negativ zu klassifizieren. Die Instanz-Ebene agiert somit auf Ebene der einzelnen Zeitstempel. Diese Vorhersagen dienen anschließend als Eingabe für die Tour-Ebene. Die Tour-Ebene besitzt dabei die Aufgabe, gegeben der bisherigen Serie von Vorhersagen der Instanz-Ebene, zu entscheiden, ob im Laufe der Tour ein Fehler aufritt oder nicht. Sie klassifiziert also auf Grundlage des bisherigen Tour-Verlaufes die Tour als Fehlertour oder als Normaltour. Ziel der Meta-Ebene ist es somit, während des Fahrens einer Tour auf Grundlage der Instanz-Vorhersagen entscheiden zu können, ob der Zugführer gewarnt werden soll oder nicht, und dabei Fehler der Instanz-Ebene abzufangen. Wichtig ist festzuhalten, dass die Meta-Ebene im Laufe einer Tour immer mehr Vorhersagen der Instanz-Ebene erhält.[180] Eine schematische Darstellung der Verknüpfung der beiden Ebenen ist in Abbildung 10 (S. 37) dargestellt. Die beiden Ebenen werden nun im Weiteren näher betrachtet.

[179] Vgl. Ebd., S. 156 f.

[180] Vgl. Ebd., S. 151 ff.

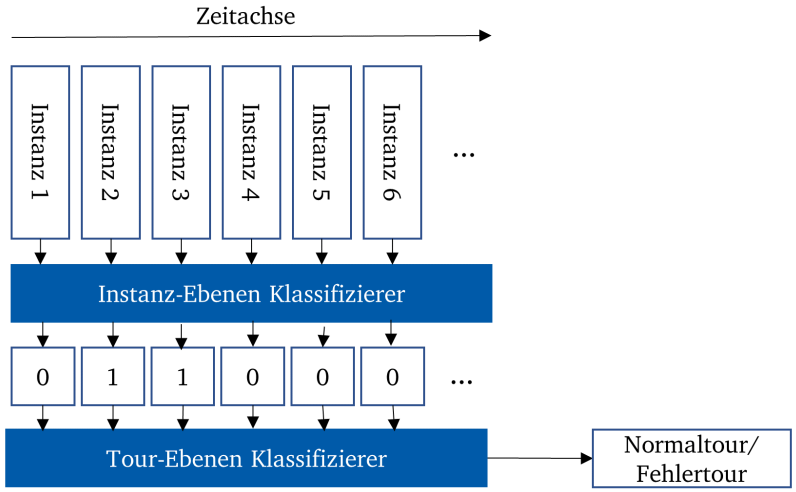

Abbildung 10: Klassifizierungsebenen

3.2.3.1 Instanz-Ebenen Klassifizierung

Für die Erstellung eines Klassifikationsmodells auf Instanz-Ebene werden von Kauschke et al. (2016) zunächst Methoden eingeführt, um die Klassenverteilung der Instanzen anzupassen. So ist der Anteil der negativen Instanzen wesentlich höher als der Anteil der positiven Klassen, was daran liegt, dass der Stromrichter in den drei aufgezeichneten Jahren nur 40 mal ausgefallen ist.[181] Dieses Ungleichgewicht das Klassen kann beim Erstellen eines Klassifikationsmodells zu Problemen führen, indem die Mehrheitsklasse das Modell dominiert.[182] Aus diesem Grund wird von Kauschke et al. (2016) u.a. mit Hilfe des SpreadSubsample Filters von Weka die Anzahl der Instanzen der negativen Klassen reduziert. Dabei wird, gegeben eines Verhältnisses von Mehrheits- zu Minderheitsklasse, eine zufällige Stichprobe aus der Mehrheitsklasse gezogen, sodass das Verhältnis nach Anwendung des Filters mit dem angegebenen übereinstimmt.[183] Das optimale Verhältnis wird dabei experimentell bestimmt.[184]

Für das Erstellen eines Klassifikationsmodells auf Instanz-Ebene werden von Kauschke et al. (2016) zwei Lernalgorithmen angewandt, welche in Abschnitt 2.3 vorgestellt wurden. Zum einen wird der Entscheidungsbaum-Lerner J48 verwendet, zum anderen der Regel-Lerner JRip.

181 Vgl. Ebd., S. 154 ff.

182 Vgl. He;Garcia (2009), S. 1263 ff.

183 Vgl. Kauschke et al. (2016), S. 160, Inglis (2016).

184 Vgl. Kauschke et al. (2016), S. 162 f.

Beide Algorithmen werden jedoch keinem Parametertuning unterzogen, sodass beide die Standardparameter zur Erstellung des Modells nutzen.[185]

3.2.3.2 Tour-Ebenen Klassifizierung

Auch auf der Tour-Ebene werden von Kauschke et al. (2016) zwei Methoden angewandt. Die erste Methode klassifiziert die laufende Tour als Fehlertour, sobald eine Vorhersage auf Instanz-Ebene positiv ausfällt. Dies bedeutet, dass ein einzelner Fehler auf Instanz-Ebene dazu führt, dass eine Warnung an den Lokführer über einen drohenden Stromrichterausfall angezeigt wird.[186]

Um dies zu vermeiden wurde ein schwellenwertbasierter Klassizierer entwickelt. Dieser betrachtet das Verhältnis $r = \frac{Anzahl\ positiver\ Vorhersagen}{Anzahl\ aller\ Vorhersagen}$ der Instanz-Ebene und wirft einen Fehler, sobald r größer als ein bestimmter Schwellenwert t ist. Dabei wird dieser Schwellenwert erst angewandt, wenn eine bestimmte Anzahl an Vorhersagen vorhanden ist. Dies verhindert, dass beispielsweise eine positive Vorhersage bei der ersten Instanz (somit $r = 1$) nicht direkt zum Ausgeben einer Warnmeldung führt. Der optimale Schwellenwert t wurde wiederum experimentell bestimmt.[187]

Die Ergebnisse von Kauschke et al. (2016) zeigen, dass der schwellenwertbasierte Klassifizierer im Vergleich zum ersten Tour-Ebenen Klassifizierer eine höhere Vorhersageleistungen erzielt.[188] Aus diesem Grund fokussiert sich die Masterthesis auf Tour-Ebene auf den schwellenwert-basierten Klassifizierer, welcher im Folgendem als „nicht lernender Verhältnisklassifizierer“ bezeichnet wird.

185 Vgl. Ebd., S. 160 f.

186 Vgl. Ebd., S. 159.

187 Vgl. Ebd., S. 159 ff.

188 Vgl. Ebd., S. 163 f.

4 Versuchsreihe 1: Binäre Vorhersage

Nachdem nun das Predictive Maintenance System von Kauschke et al. (2016) beschrieben wurde, wird im folgendem Kapitel die erste Forschungsfrage untersucht. Ziel der ersten Forschungsfrage ist es, zu klären, ob es möglich ist, die Vorhersagequalität des bisherigen Predictive Maintenance System durch Verwendung alternativer Ansätze bzw. Algorithmen zu verbessern. Zur Beantwortung der Forschungsfrage unterteilt sich das Kapitel grundlegend in drei Abschnitte. Im ersten Abschnitt werden zunächst die neuen Verfahren und Algorithmen beschrieben, die zur Verbesserung der Vorhersagequalität entwickelt bzw. ausgewählt wurden. Anschließend wird im zweiten Abschnitt erläutert, wie bei der Evaluierung dieser Neuerungen vorgegangen wurde. Es wird also beschrieben, wie die Neuerungen in Weka implementiert, wie die Daten aufgeteilt, wie die Versuche aufgebaut und welche Parameterkonfigurationen ausprobiert wurden. Im letzten Abschnitt folgt anschließend die Präsentation und Diskussion der Versuchsergebnisse.

4.1 Konzeption

Im folgenden Abschnitt sollen die neu entwickelten Verfahren bzw. Anpassungen zum ursprünglichen System erläutert werden. Wie in Abbildung 11 dargestellt, wurde im Rahmen der ersten Forschungsfrage in drei der vier grundlegenden Bestandteile des Systems von Kauschke et al. (2016) Änderungen vorgenommen. So wurden die Datentransformation, die Instanz-Ebenen Klassifizierung und die Tour-Ebenen Klassifizierung angepasst. Die Änderungen sollen nun im Folgendem erklärt werden.

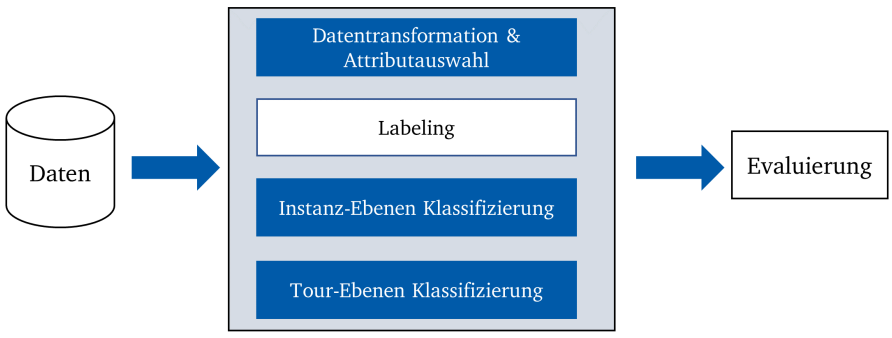

Abbildung 11: Vorhersagesystem mit Anpassungen (Versuchsreihe 1)

4.1.1 Datentransformation: Gleitendes Zeitfenster

Die erste Änderung betrifft die Erstellung der Instanzen bzw. deren Attribute. Bisher wird, wie in Abschnitt 3.2.1 beschrieben, für jeden Zeitpunkt eine Instanz erzeugt, die als Attribute den aktuellen Zustand der Diagnosecodes bzw. die aktuellen Umfelddaten erhält. Jede Instanz stellt also den Zustand der Lok zu einem konkreten Zeitpunkt dar, bildet also einen Schnitt entlang der Zeitachse.[189] Das Prinzip dieser Vorgehensweise ist an einem Beispiel in Abbildung 12 dargestellt.

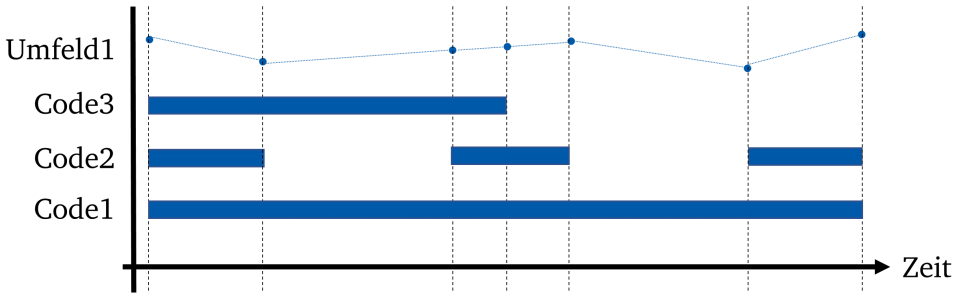

Abbildung 12: Beispiel Datentransformation

In der Abbildung ist der zeitliche Verlauf von drei Diagnosecodes und einem Umfeldwert visualisiert. Die Y-Achse unterteilt dabei die Attribute, die X-Achse hingegen stellt die Zeitachse dar. Für jeden Diagnosecode ist nun durch einen blauen Balken symbolisiert, wann dieser aktiv war. Für den Umfeldwert hingegen sind die einzelnen numerischen Werte als Punkte dargestellt. Die erzeugten Instanzen hingegen sind durch eine gepunktete Linie symbolisiert. Wie zu erkennen ist, wird bei jeder Änderung von Code1, Code2 und Code3 eine neue Instanz erzeugt. Im konkreten Beispiel werden also sieben Instanzen aus den Rohdaten generiert. Der Klassifizierer erhält also zum Lernen vier Instanzen mit $code3 = 1$ und drei Instanzen mit $code3 = 0$.

Durch diese Vorgehensweise beim Erstellen der Instanzen geht jedoch die Information über den zeitlichen Verlauf verloren. So erhält der Klassifizierer zum Lernen die besagten vier Instanzen mit $code3 = 1$ und die drei Instanzen mit $code3 = 0$, besitzt jedoch keine Information in welchen zeitlichen Zusammenhang diese zueinander stehen. Letztlich können so die zeitlichen Muster in Bezug auf Code3, welche in Abbildung 13 (S. 41) dargestellt sind, nicht unterschieden werden.

[189] Vgl. Ebd., S. 154 f.

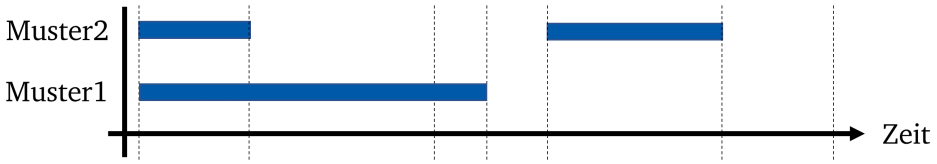

Abbildung 13: Zeitliche Muster des Beispielcodes Code3

Wie zu erkennen ist, sind in Abbildung 13 zwei unterschiedliche zeitliche Muster von $Code3$ abgebildet. Beim ersten Muster, welche identisch zum Beispiel in Abbildung 12 ist, liegt der Diagnosecode einmal und relativ lange an. Beim zweiten Muster hingegen taucht der Diagnosecode zweimal, jedoch jeweils kürzer auf. Das Problem ist nun, dass bei beiden Mustern die identische Instanzmenge generiert wird. So erstellt die bisherige Vorgehensweise in beiden Fällen vier Instanzen mit $code3 = 1$ und drei Instanzen mit $code3 = 0$. An diesem Beispiel wird klar, dass der bisherige Transformations-Ansatz es nicht erlaubt, zeitliche Muster zu lernen und es somit nicht ermöglicht, drohende Stromrichterausfälle anhand dieses Musters zu erkennen.

Um auch zeitliche Muster lernen zu können, die einen bevorstehenden Stromrichterausfall kennzeichnen, war nun die Idee die Instanzen durch Attribute zu erweitern, die den zeitlichen Verlauf der Diagnosecodes abbilden. Um nicht die gesamte Vergangenheit abbilden zu müssen, wird hierzu ein gleitendes Fenster einer festen Zeitlänge z (in Sekunden) eingeführt. Das Prinzip ist dabei in Abbildung 14 abgebildet.

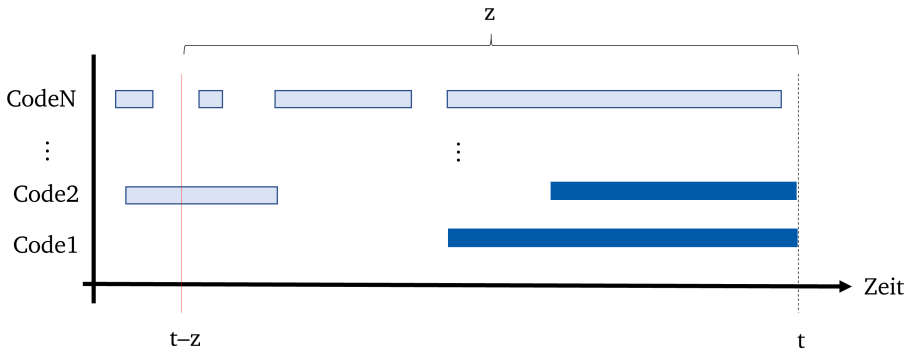

Abbildung 14: Gleitendes Zeitfenster

In Abbildung 14 ist eine Situation dargestellt, in der gerade eine Instanz für den Zeitpunkt t erstellt werden soll. Dabei stellen die hellblauen Balken die vergangenen Zeiträume dar, an denen der jeweilige Code angelegen hat. Die dunkelblauen Balken hingegen symbolisieren den Zeitraum von Codes, die aktuell noch andauern, also zum Zeitpunkt t noch nicht beendet sind.

Das gleitende Zeitfenster umfasst dabei den Zeitraum von $t - z$ bis t. Die Idee war nun diesen Zeitraum durch zusätzliche Attribute zu beschreiben.

Ein möglicher Ansatz wäre gewesen, den Zeitraum z in einem bestimmten Intervall l für jeden Diagnosecode abzutasten und alle abgetasteten Werte als Attribute der Instanz zu Zeitpunkt t abzuspeichern. In diesem Fall wäre eine komplette Zeitreihe für den Zeitraum z für jeden Diagnosecode abgespeichert worden, sodass pro Diagnosecode $\frac{z}{l}$ neue Attribute erzeugt werden müssen. Um jedoch die Anzahl der Dimensionen möglich gering zu halten, galt es, den Verlauf des Zeitraums $t - z$ bis t möglich gut durch wenige Aggregationsmaße zu beschreiben. Letztlich wurden zwei Maße identifiziert, die den zeitlichen Verlauf der vergangenen z Sekunden für jeden Code charakterisieren sollen:

- $codeX\_num$: Kennzeichnet die Anzahl, wie oft CodeX in den vergangenen z Sekunden aufgetreten ist. Dabei werden alle Zeiträume mit in die Berechnung aufgenommen, die im Zeitraum $(t - z, t)$ enden.
- $codeX\_avg$: Kennzeichnet die durchschnittlich Zeitspanne, die CodeX in den vergangenen z Sekunden angedauert hat. Dabei werden auch hier alle Zeiträume mit in die Berechnung aufgenommen, deren Ende im Zeitraum $(t - z, t)$ liegt.

Zusätzlich hierzu wird noch ein drittes Attribut berechnet, das jedoch unabhängig von gleitenden Zeitfenster ist:

- $codeX\_dur$: Kennzeichnet, wie lange der aktuell anliegende CodeX bereits andauert.

Für die Instanz zum Zeitpunkt t aus Abbildung 14 (S. 41) wird dabei wie folgt vorgegangen. Code1 und Code2 liegt aktuell an, wohingegen CodeN aktuell nicht anliegt. Aus diesem Grund wird wie bisher $code1 = 1$, $code2 = 1$ und $codeN = 0$ gesetzt. Zusätzlich wird der Zeitraum seit dem Code1 und Code2 andauern, in die Attribute $code1\_dur$ und $code2\_dur$ eingefügt, wohingegen $codeN\_dur = 0$ gesetzt wird. Nun kommen wir zu den Attributen, bei denen das gleitende Zeitfenster eine Rolle spielt. Code1 war im Zeitraum $t - z$ bis t vor dem aktuellen Auftreten nie aktiv, weshalb $code1\_num = 0$ und $code1\_avg = 0$ gesetzt wird. Bei Code2 hingegen ist der Code innerhalb des gleitenden Zeitfensters bereits einmal aktiv gewesen. Aus diesem Grund wird $code2\_num = 1$ gesetzt und $code2\_avg$ mit der Dauer dieses vergangenen Zeitfensters gesetzt. Zuletzt war CodeN in den vergangenen z Sekunden drei Mal aktiv. Somit wird $codeN\_num = 3$ gesetzt und der arithmetische Mittelwert dieser drei Zeiträume in $codeN\_avg$ eingetragen. Zusammengefasst werden nun vier Attribute pro Diagnosecode erzeugt, wohingegen beim bisherigen Ansatz ein Attribut pro Diagnosecode erstellt wurde.

Somit erhöht sich die Anzahl der binären Attribute von 888 auf 3552, welche jetzt jedoch auch Informationen über den zeitlichen Verlauf enthalten. So ist es nun möglich, die beiden zeitlichen Muster aus Abbildung 13 (S. 41) zu unterscheiden, da diese nun durch die Attribute *codeX_num*, *codeX_avg* und *codeX_dur* unterscheidbar sind.

Dabei sei an dieser Stelle angemerkt, dass sich bewusst gegen die Erzeugung von Attributen entschieden wurde, die den zeitlichen Verlauf der Umfelddaten charakterisieren. Grund hierfür ist, dass bei diesen Attributen relativ häufig Werte fehlen, sodass teilweise über größere Zeiträume keine Werte vorliegen.

4.1.2 Instanz-Ebenen Klassifizierung

Neben der Möglichkeit, auch zeitliche Muster mit Hilfe der zusätzlichen Attribute erkennen zu können, wurde auch bei der Instanz-Ebenen Klassifizierung Anpassungen vorgenommen. So wurden im bisherigen Ansatz auf Instanz-Ebenen die beiden Lernalgorithmen J48 und JRip angewendet.[190] Die Idee war nun die Vorhersagequalität des Predictive Maintenance Systems durch andere Lernalgorithmen zu verbessern. Da jedoch aufgrund der begrenzten Zeit eine Auswahl der Lernalgorithmen stattfinden musste, wurde zunächst im Rahmen einer Literaturrecherche geklärt, welche Algorithmen im Allgemeinen eine gute Klassifikationsleistung versprechen.

Einen umfangreichen empirischen Vergleich von überwachten Lernalgorithmen bieten dabei Caruana und Niculescu-Mizil (2006). Diese verglichen unter anderem Support Vektor Maschinen, Neuronale Netzwerke, Naive Bayes, K-Nearest-Neighbour, Random Forest, Entscheidungsbäume und Boosting mit Entscheidungsbäumen auf unterschiedlichen Datensätzen mit unterschiedlichen Evaluierungsmaßen miteinander.[191] Auf Grundlage dieser empirischen Ergebnisse wurde entschieden, folgende neue Lernalgorithmen für die Vorhersage auf Instanz-Ebene zu verwenden:

- AdaBoost in Kombination mit Entscheidungsbäumen
- Random Forest
- J48

190 Vgl. Ebd., S. 160.

191 Vgl. Caruana;Niculescu-Mizil (2006), S. 161 ff.

Dabei werden bei allen Lernalgorithmen die Parameter mit Hilfe einer Kreuzvalidierung an das Problem angepasst, da dies die Vorhersageleistung der Algorithmen erhöhen kann.[192] Somit stellt auch der J48 insofern eine Neuerung dar, da er im bisherigen Ansatz nur mit Standardparametern ausgeführt wurde.[193] Die Algorithmen wurden dabei bereits in den Grundlagen vorgestellt, weshalb sich der folgende Abschnitt auf die Auswahl und Beschreibung der zu optimierenden Parameter konzentriert.

4.1.2.1 J48

Der J48 Algorithmus ist, wie in Abschnitt 2.3.1 erläutert, eine konkrete Implementierung des C4.5 Lernalgorithmus in Weka, der als Klassifikationsmodell Entscheidungsbäume erstellt.[194] Dabei besitzt der J48 Algorithmus als Lernparameter unter anderem das Konfidenzniveau, die minimale Anzahl Instanzen pro Blatt und verschiedene Parameter zum Einstellen des Prunings.[195] Im Rahmen der Arbeit wurden, angelehnt an die Studie von Molina et al. (2012), folgende Parameter zum Tuning ausgewählt[196]:

- *Konfidenzniveau*: Dieser Parameter beeinflusst das Post-Pruning. Wie bereits in Abschnitt 2.3.1 erläutert, wird dies benutzt um eine pessimistische Schätzung der Fehlerrate anhand der vorliegende Menge an Trainingsbeispielen vorzunehmen.[197] Je kleiner der Wert, desto mehr Pruning Operationen sind erlaubt.[198]
- *Minimale Anzahl Instanzen pro Blatt:* Dieser Parameter legt einen Schwellenwert m für die Anzahl an Instanzen bei einer Aufteilung des Baumes fest. Es werden nur Attributtests erlaubt, aus welchen mindestens zwei Teilmengen entstehen, die mindestens *m* Instanzen enthalten.[199]
- *Pruning aus/an:* Dieser Parameter beeinflusst, ob Pruning zum Stutzen des Baumes angewendet werden soll oder nicht.[200]

192 Vgl. Lavesson;Davidsson (2006), S. 395, Caruana;Niculescu-Mizil (2006), S. 161 ff.
193 Vgl. Kauschke et al. (2016), S. 160 f.
194 Vgl. Witten;Frank (2005), S. 373.
195 Vgl. Lavesson;Davidsson (2006), S. 397.
196 Vgl. Molina et al. (2012), S. 181.
197 Vgl. Quinlan (1993), S. 35 ff.
198 Vgl. Molina et al. (2012), S. 181.
199 Vgl. Witten;Frank (2005), S. 199 u. 406 f.
200 Vgl. Frank (2016a).

Bei der Auswahl der konkreten Werte für die jeweiligen Parameter wurde sich am Wertebereich von Lavesson und Davidsson (2006) orientiert.[201] Letztlich wurden so folgende Parameterkonfigurationen verwendet, wobei für alle anderen Parameter der Standardwert verwendet wurde:

- *Konfidenzniveau:* 0,25 [Standard]; 0,5
- *Minimale Anzahl Instanzen pro Blatt:* 2 [Standard]; 4
- *Pruning:* an [Standard]; aus

Anzumerken ist, dass bei Pruning = aus, die anderen Parameter nicht variiert werden, da sie bei abgeschaltetem Pruning keinen Einfluss auf die Berechnung haben. Letztlich existieren für J48 somit 5 Parameterkonstellationen.

4.1.2.2 Random Forest

Random Forest ist, wie in Abschnitt 2.3.3 beschrieben, ein Verfahren, dass eine große Anzahl von Entscheidungsbäumen generiert, welche anschließend per Mehrheitsentscheid über die vorherzusagende Klasse abstimmen.[202] Dabei können verschiedene Parameter zum Lernen festgelegt werden.[203] Angelehnt an die Studie von Huang und Boutros (2016), welche die Parametersensitivität von Random Forest untersuchten, wurden letztlich folgende Parameter zur Optimierung ausgewählt:[204]

- *Anzahl der Bäume/Iterationen:* Beschreibt die Anzahl an Iterationen, die durchgeführt werden sollen. Hierdurch wird die Anzahl der zu erzeugenden Bäume definiert.
- *Anzahl der Attribute pro Baum:* Dies beschreibt die Anzahl an Attributen, die zur Erzeugung eines Baumes verwendet werden soll.
- *Anzahl der Trainingsbeispiele pro Baum in Prozent:* Beschreibt die Größe der Stichproben in Relation zur Menge aller Trainingsbeispiele, welche bei jeder Iteration/bei jedem Baum erzeugt wird.

Da eine hohe Anzahl an Bäumen die Leistung von Random Forest nicht verschlechtert, wurde eine möglichst große Anzahl von 1000 Bäumen festgesetzt.[205] Für die Anzahl an Attributen

201 Vgl. Lavesson;Davidsson (2006), S. 397.

202 Vgl. Breiman (2001), S. 6.

203 Vgl. Kirkby (2016)

204 Vgl. Huang;Boutros (2016), S. 331:1 ff, Kirkby (2016).

205 Vgl. Breiman (2001), S. 7.

wurde sich hingegen an den Empfehlungen von Breiman (2001) und an der Dokumentation des Random Forest Klassifizier in scikit-learn (2016) orientiert.[206] Letztlich wurden folgende Parameterwerte ausgewählt:

- *Anzahl der Bäume/Iterationen:* 1000
- *Anzahl der Attribute pro Baum:* $\lfloor \log_2(\text{Anzahl Attribute}) + 1 \rfloor$ [Standard]; $\sqrt{\text{Anzahl Attribute}}$
- *Anzahl Trainingsbeispiele pro Baum [%]:* 50; 100 [Standard]

Insgesamt existieren folglich für Random Forest vier unterschiedliche Konfigurationen.

4.1.2.3 AdaBoost

Die Idee von AdaBoost ist es, wie bereits in Abschnitt 2.3.4 erläutert, eine Menge von schwachen Klassifizierern iterativ anzuwenden, die aus den Fehlern ihrer Vorgänger lernen, indem die Gewichte der Instanzen angepasst werden.[207] Dabei sollte AdaBoost im Rahmen der Arbeit mit Entscheidungsbäumen kombiniert werden. Als schwacher Klassifizierer wurde der REPTree von Weka verwendet, welcher sowohl für die Regression, als auch für die Klassifikation verwendet werden kann.[208] Um dabei einen schwachen Klassifizierer darzustellen, gibt es beim REPTree die Möglichkeit, die maximale Baumtiefe festlegen zu können, sodass auch Bäume der Höhe 1 (auch Decision Stumps genannt) erzeugt werden können.[209] Letztlich wurden, angelehnt an Caruana und Niculescu-Mizil (2006), folgende Parameter zum Tuning festgelegt:[210]

- *Anzahl der Iterationen:* Dieser Parameter bestimmt die Anzahl an Iterationen im AdaBoost Algorithmus[211]
- *Maximale Baumtiefe:* Dieser Parameter bestimmt die maximale Tiefe des REPTree, welcher den Basisklassifizierer darstellt.[212]

206 Vgl. Ebd., S. 12, scikit-learn (2016).
207 Vgl. Freund;Schapire (1997), S. 126 ff.
208 Vgl. Witten;Frank (2005), S. 407 f.
209 Vgl. Frank (2016b).
210 Vgl. Caruana;Niculescu-Mizil (2006), S. 162.
211 Vgl. Frank;Trigg (2016).
212 Vgl. Frank (2016b).

Als Wertemenge der Parameter wurde sich wiederum auf Grundlage von Caruana und Niculescu-Mizil (2006) für folgende Werte entschieden:[213]

- *Anzahl der Iterationen:* 100; 1000
- *Maximale Baumtiefe:* 1; 3

Es existieren also auch in diesem Fall vier verschiedene Parameterkonfigurationen.

4.1.3 Tour-Ebenen Klassifizierung

Zusätzlich zur Verwendung alternativer Algorithmen auf Instanz-Ebene wurden neue Verfahren auf der Tour-Ebene entwickelt. Das Ziel dabei war es, die Zeitreihe, die durch die Vorhersagen der Instanz-Ebenen Klassifizier im Laufe jeder Tour aufgespannt wird, zur Klassifikation der Tour in „Normaltour“ oder „Fehlertour“ zu nutzen. Anzumerken ist, dass mit zunehmender Dauer einer Tour die Anzahl der Vorhersagen auf Instanz-Ebene ansteigt, sodass die Informationsgrundlage für den Tour-Ebenen Klassifizierer stetig größer wird. Um zu vermeiden, dass der Tour-Ebenen Klassifizierer im Falle eine Fehlertour diese erst relativ spät als „Fehlertour“ klassifiziert (und somit der Lokführer relativ spät gewarnt wird), bestand ein weiteres Ziel darin, Fehlertouren möglichst früh zu erkennen. Auf der anderen Seite sollte bei der Klassifikation einer Tour als Fehlertour möglichst pessimistisch vorgegangen werden, welches ein weiteres Ziel definierte. Dies ist darin begründet, dass im Falle vieler falscher Alarme, das Vertrauen der Mitarbeiter in das System vermutlich sinkt. Ein weiterer Grund bildet der Fakt, dass zum Lernen (wie später in Abschnitt 4.2.2 beschrieben) nur eine Stichprobe aller Touren verwendet wird, wobei die Anzahl der Normaltouren um einen höheren Faktor reduziert wird. Zum Beispiel kann so eine Fehlerrate (in Bezug auf die Klassifikation von Normaltouren) in Höhe von 0,2% in der Stichprobe, absolut gesehen, einen einzigen falschen Alarm in der Stichprobe hervorrufen. Bezogen auf die gesamte Menge der Normaltouren kann diese Fehlerrate jedoch zu mehr als 140 falschen Alarmen führen.

Auf Grundlage dieser Anforderungen wurden zwei Klassifizierer auf der Tour-Ebene entwickelt. Dabei stellt einer der Klassifizierer eine Weiterentwicklung einer der bisherigen Tour-Ebenen Klassifizierer dar, während der andere komplett neu konzipiert wurde. Beide Klassifizierer sollen nun im Folgendem vorgestellt werden.

[213] Vgl. Caruana;Niculescu-Mizil (2006), S. 162.

4.1.3.1 Lernender Verhältnisklassifizierer

Einer der Tour-Ebenen Klassifizierer baut auf dem in Abschnitt 3.2.3.2 beschriebenen, nicht lernenden Verhältnisklassifizierer von Kauschke et al. (2016) auf. Dieser betrachtet das Verhältnis $r = \frac{Anzahl\ positiver\ Vorhersagen}{Anzahl\ aller\ Vorhersagen}$ der Instanz-Ebene und klassifiziert eine Tour als „Fehlertour“ sobald ein bestimmter Schwellenwert t überschritten wurde, also $r > t$.[214] Die wesentliche Annahme dabei ist, dass bei Fehlertouren die positive Klasse vom Instanz-Ebenen Klassifizierer häufiger (im relativen Sinne) vorhersagt wird als bei Normaltouren. Mit anderen Worten wird also angenommen, dass die Wahrscheinlichkeit einer Tour zur Klasse „Fehlertour“ zugehören, höher ist, je höher der Anteil der positiven Vorhersagen auf Instanz-Ebene ausfällt. Dabei ist es in der bisherigen Version notwendig, den Schwellenwert t experimentell zu bestimmen. Es muss folglich durch eine Parameteroptimierung ermittelt werden, welches der optimale t Wert ist. Da jedoch die Zeit- und Rechenressourcen begrenzt sind, ist es nur schwer möglich, beliebig viele Wert auszuprobieren, weshalb sich Kauschke et al. (2016) auf 9 Werte beschränkt.[215]

Die Idee war nun, den Schwellenwert t mit Hilfe der Trainingsdaten zu lernen. Die Motivation unterteilt sich dabei in zwei Punkte. Zum einen sollte die Anzahl an möglichen Parameterkonfigurationen reduziert werden, da nun auch auf Instanz-Ebene Parameter optimiert werden sollten. Zudem sollte der Schwellenwert t genauer bestimmt werden, um so möglicherweise die Klassifikationsleistung zu erhöhen.

Um den Schwellenwert t zu lernen, wird grundlegend wie in Abbildung 15 (S. 49) dargestellt vorgegangen. Gegeben ist also eine Trainingsmenge, welche eine bestimmte Anzahl Fehler- und Normaltouren enthält, die wiederum aus verschiedenen Instanzen bestehen. Dabei sind die Instanzen, wie bereits in Abschnitt 3.2.2 beschrieben, mit einer positiven und negativen Klasse versehen. Im ersten Schritt werden alle positiven Instanzen der Fehlertouren (rot) und alle negativen Instanzen der Normaltouren (hellblau) in einem Datensatz gesammelt. Dieser stellt die Trainingsmenge für den Klassifizierer auf Instanz-Ebene dar. Mit diesen Trainingsdaten wird im zweiten Schritt das Instanz-Ebenen Klassifikationsmodell erzeugt, welches gegeben einer Instanz vorhersagt, ob diese positiv oder negativ ist. Das gelernte Klassifikationsmodell wird im dritten Schritt auf alle Touren angewandt, sodass nun der Vorhersageverlauf auf Instanz-Ebene für jede Tour bekannt ist. Mit Hilfe dieser Vorhersagen kann nun das Verhältnis $r = \frac{Anzahl\ positiver\ Vorhersagen}{Anzahl\ aller\ Vorhersagen}$ zu jedem Zeitpunkt der Tour berechnet werden. Im vierten Schritt

[214] Vgl. Kauschke et al. (2016), S. 159 ff.

[215] Vgl. Ebd.

wird mit diesen Informationen anschließend eine Trainingsmenge auf Tour-Ebene erzeugt. Dabei wird für jede Tour ein Trainingsbeispiel generiert, wobei jedes Beispiel aus dem maximalen Verhältnis r der Tour und deren Klasse besteht. Im fünften Schritt wird mit dieser Trainingsmenge nun ein Entscheidungsbaum der Tiefe 1 gelernt, welches letztlich Touren nur anhand des maximalen Tour-Verhältnisses r unterteilten kann. Der resultierende Baum, in Form von $r \leq t$ und $r > t$, enthält als Attributtest letztlich den Schwellenwert t. Das Lernen des Tour-Ebenen Klassifizierers fokussiert sich also hauptsächlich auf die Schritte 4 und 5, welche im Folgenden näher untersucht werden sollen.

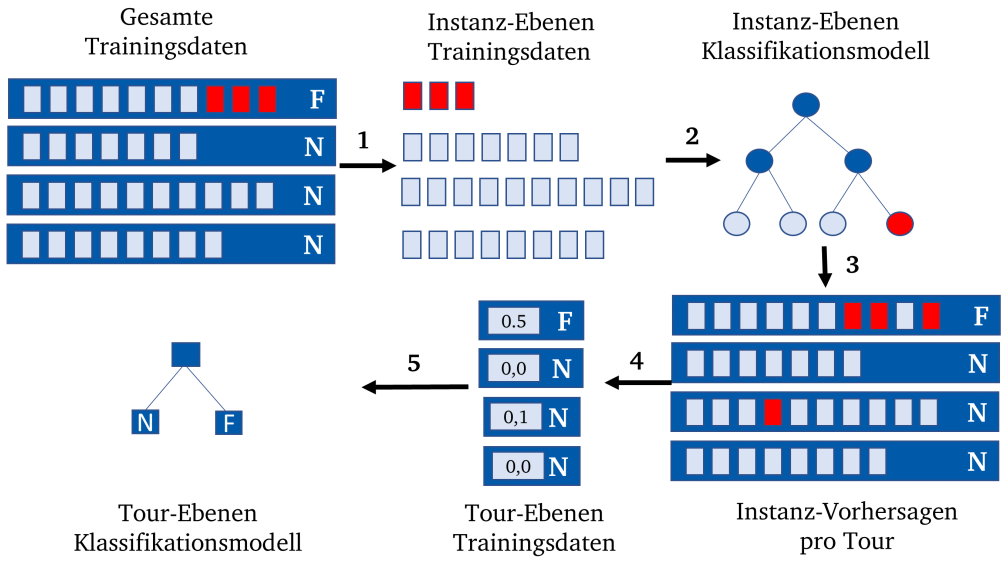

Abbildung 15: Schematische Vorgehensweise zum Lernen des Schwellenwertes t

Wie bereits im vorherigen Abschnitt erwähnt, ist es die Aufgabe von Schritt 4 eine Trainingsmenge für den Tour-Ebenen Klassifizierer zu erstellen, die auf den Instanz-Vorhersagen bzw. auf den Verhältniswerten r der Tour basiert. Betrachten wir also als Beispiel eine Fehlertour und eine Normaltour, welche in Abbildung 16 (S. 50) und Abbildung 17 (S. 50) dargestellt sind. Für beide Touren ist der Vorhersageverlauf auf Instanz-Ebene im oberen Graphen in grün dargestellt. Wie zu erkennen ist, wurden die beispielhaften Touren gemäß der grundlegenden Annahme erzeugt, dass bei Fehlertouren häufiger die Klasse positiv vorhergesagt wird. Mit Hilfe des Vorhersageverlaufs auf Instanz-Ebene kann nun das Verhältnis $r = \frac{Anzahl\ positiver\ Vorhersagen}{Anzahl\ aller\ Vorhersagen}$ für jeden Zeitpunkt bestimmt werden. Es entstehen für jede der Touren also 30 Verhältniswerte r.[216] Gesucht ist jetzt ein Schwellenwert t für das Verhältnis r,

[216] Dabei wird bei diesem Beispiel darauf verzichtet, dass eine minimale Anzahl an Instanz-Vorhersagen zur Berechnung des Verhältnisses r existieren muss.

wobei r im Laufe einer Fehlertour den Schwellenwert mindestens einmal und möglichst weit vor dem Fehler überschreiten soll. Für Normaltouren hingegen soll gelten, dass das Verhältnis r niemals den Schwellenwert t überschreitet. Zusammengefasst ist also ein Schwellenwert t gesucht, der die Verhältnisverläufe der Normal- und Fehlertouren möglichst gut trennt.

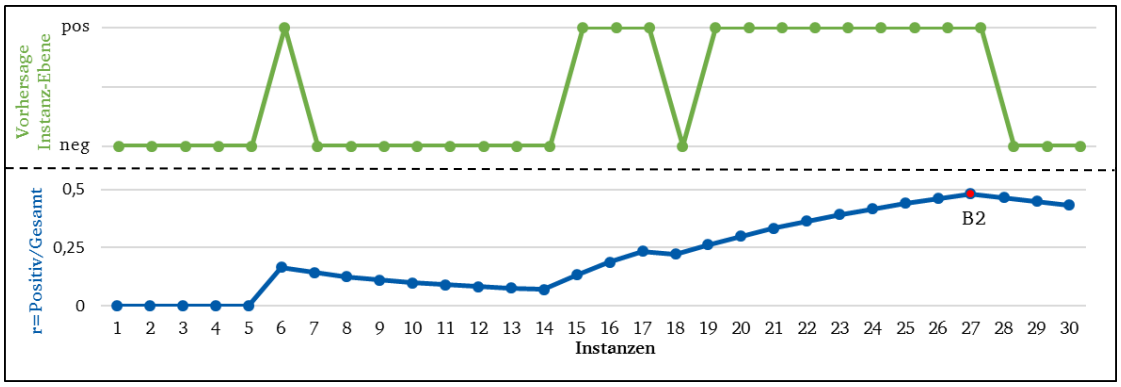

Abbildung 16: Beispiel Fehlertour (lernender Verhältnisklassifizierer)

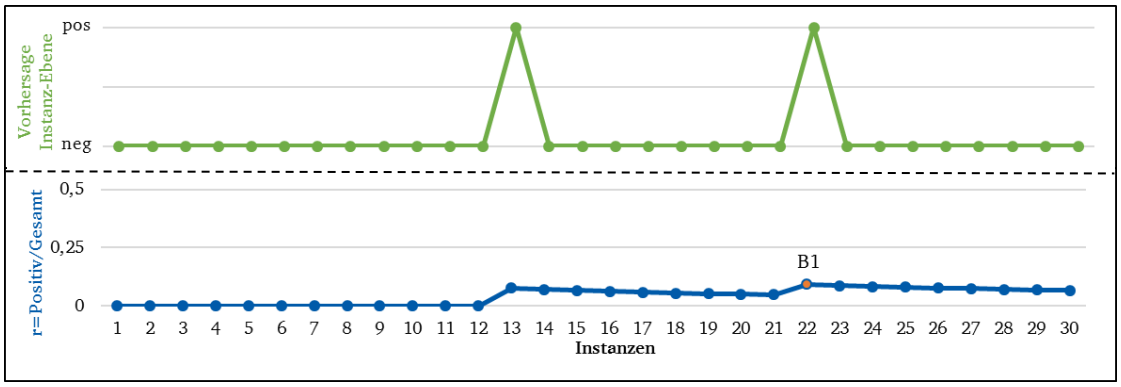

Abbildung 17: Beispiel Normaltour (lernender Verhältnisklassifizierer)

Die Idee ist nun, den maximalen Verhältniswert r jeder Tour zu verwenden, um einen Schwellenwert t zu lernen. Im Beispiel wäre dies für die Normaltour der Wert $r = 0{,}091$ und für die Fehlertour der Wert $r = 0{,}481$. Es würden also zwei Trainingsbeispiele für den Tour-Ebenen Klassifizierer erstellt. Zum einen das Beispiel $B_1 = (0{,}091; \mathrm{N})$ für die Normaltour und das Beispiel $B_2 = (0{,}481; F)$ für die Fehlertour. In Abbildung 18 (S. 51) ist dabei eine Trainingsmenge für den Tour-Ebenen Klassifizierer abgebildet, die auf realen Daten erzeugt wurde. Dabei ist auf den ersten Blick ersichtlich, dass kein Schwellenwert t die beiden Klassen perfekt trennen kann. So existiert beispielsweise eine Normaltour mit einem Wert von $r = 0{,}188$ und mehrere Fehlertouren mit $r = 0$. Es ist also ein Schwellenwert gesucht, der einen gewissen Grad an Fehlern zulässt. Dies ist die Aufgabe von Schritt 5.

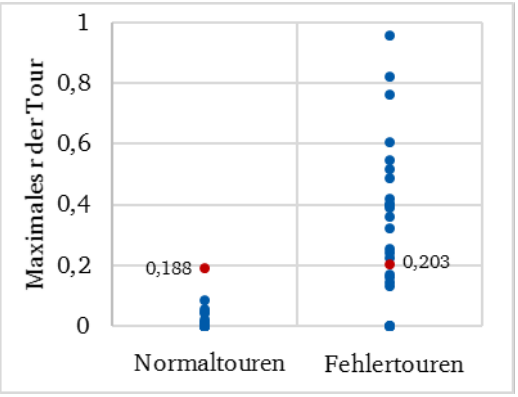

Abbildung 18: Trainingsbeispiele Tour-Ebene

In Schritt 5 wird auf den Tour-Ebenen Trainingsbeispielen ein Entscheidungsbaum der Tiefe 1 gelernt. Der Entscheidungsbaum ist dabei im Wesentlichen ein binärer Test $r \leq t$ bzw. $r > t$ auf das Attribut r. Wie in Abschnitt 2.3.1 erläutert, kann hierfür ein Entscheidungsbaum-Algorithmus verwendet werden, welcher den optimalen Wert t anhand des Information Gain bestimmt und dabei alle Beispiele gleich gewichtet. Es wurde aufgrund der letzten Anforderungen aus Kapitel 4.1.3 entschieden, einen kostensensitiven Entscheidungsbaum zu verwenden. Die Idee war, durch die Kosten abzubilden, dass das Verhältnis zwischen Fehler- und Normaltouren in der Trainingsmenge von dem Verhältnis der Realität abweicht. So führt eine fehlerhafte Klassifizierung einer Normaltour in der Trainingsmenge in der Realität vermutlich zu einer vielfachen Anzahl an Fehlern. Zum anderen sollte darin einfließen, dass das Vertrauen der Mitarbeiter durch häufige Fehlalarme vermutlich verloren geht. Dennoch wird davon ausgegangen, dass eine unnötige Wartung weniger kostet, als ein Ausfall auf der Strecke. Letztlich wurde folgende Kostenmatrix festgelegt:

$$C = \begin{pmatrix} 0 & 10 \\ 1 & 0 \end{pmatrix} \tag{9}$$

Die Kosten für die Fehlklassifikation einer Normaltour sind also 10-mal höher, als die einer Fehlertour angesetzt. Letztlich wird durch die Kostenmatrix definiert, dass eine Warnung eines drohenden Stromrichterausfalls pessimistisch herausgegeben wird.

Um die Kosten bei der Erstellung des Baumes zu berücksichtigen, existieren verschiedene Ansätze. Einige der Ansätze verwenden hierfür eine Erweiterung oder Anpassung der bisherigen Maße wie beispielsweise des Information Gains.[217] Im Rahmen der Arbeit wird jedoch der Algorithmus von Ling et al. (2004) verwendet, welcher bei der Erstellung des Baumes die

[217] Vgl. Lomax;Vadera (2013), S. 16:6.

Kosten direkt mit einfließen lässt. Grundlegend geht der Algorithmus dabei wie der C4.5 Algorithmus vor. Jedoch wird das optimale Attribut bzw. Attributwert nicht mehr durch den Information Gain bestimmt, sondern es wird der Attributtest gewählt, der die Gesamtkosten minimiert. Die Gesamtkosten setzen sich dabei aus den Kosten für jede zusätzliche Aufteilung des Baumes (Testkosten) und den Kosten für eine fehlerhafte Klassifikation (Misklassifikationskosten) zusammen.[218] Da in der konkreten Problemstellung auf jeden Fall eine Aufteilung vorgenommen werden soll, werden die Testkosten nicht berücksichtigt und deshalb auf 0 gesetzt.

Um die Vorgehensweise zu verdeutlichen, sei eine Trainingsmenge mit P positiven und N negativen Beispielen gegeben. Zusätzlich seien die Kosten für False Negatives C_{FN} und für False Positives C_{FP} geben. Ist nun $P * C_{FN} > N * C_{FP}$, so ist es, ohne eine Aufteilung vorzunehmen, günstiger die positive Klasse vorherzusagen. Die Misklassifikationskosten wären also in diesem Fall, da alle negativen Beispiele der Menge fehlerklassifiziert werden würden, gegeben durch:[219]

$$T_0 = N * C_{FP} \tag{10}$$

Gehen wir nun weiter davon aus, dass ein Attribut A existiert, welches die Menge in zwei Teilmengen unterteilt. Dabei würden in der ersten Menge $P1$ positive und $N1$ negative Beispiele und in der zweiten Menge $P2$ positive und $N2$ negative Beispiele resultieren. Auch hier wird nun durch die Vergleiche $P1 * C_{FN} > N1 * C_{FP}$ und $P2 * C_{FN} > N2 * C_{FP}$ jeweils die Klasse vorhergesagt, welche die geringsten Misklassifikationskosten mit sich zieht. Nehmen wir an, dass es in der ersten Menge günstiger wäre die positive Klasse, in der zweiten hingegen die negative Klasse vorherzusagen. Die Gesamtkosten nach einem Test von Attribut A wären nun:[220]

$$T_1 = N1 * C_{FP} + P2 * C_{FN} \tag{11}$$

Letztlich wird das Attribut gewählt, welches die größte Reduktion der Misklassifikationskosten $T_0 - T_1$ ermöglicht.[221] Da in der konkreten Problemstellung nur eine einzige Aufteilung des Baumes vorgenommen werden soll, wird also der Schwellenwert t verwendet, der die minimalen Gesamtkosten T_1 besitzt. Existieren hierbei mehrere Attributewerte t, wird ein zufälliger Wert gewählt. Dabei werden für t nur Werte untersucht, die zwischen zwei, nach der Größe sortierten, Attributwerten r liegen. In Bezug auf das Beispiel aus Abbildung 18 (S. 51) wird also folgender Baum gelernt:

$$\begin{aligned} r &\leq 0{,}1955 \rightarrow 0 \\ r &> 0{,}1955 \rightarrow 1 \end{aligned} \tag{12}$$

[218] Vgl. Ling et al. (2004), S. 69:1 ff.

[219] Vgl. Ebd., S. 69:2 ff.

[220] Vgl. Ebd.

[221] Vgl. Ebd.

Der Schwellenwert t ist also in diesem Fall durch den Wert 0,1955 gegeben. Wobei t durch den Mittelwert der beiden roten Punkte in Abbildung 18 berechnet wurde. Zusammengefasst wurde also jetzt ein Schwellenwert t für das Verhältnis r gelernt, bei dessen Überschreitung ein Alarm an den Lokführer herausgegeben bzw. die Tour als Fehlertour klassifiziert wird.

4.1.3.2 Aktivierungsklassifizierer

Neben dem Klassifizierer aus dem vorherigen Abschnitt wurde ein weiterer Klassifizierer für die Tour-Ebene entwickelt. Das Ziel hierbei war es, neben der relativen Häufigkeit auch die zeitliche Dichte der positiven Vorhersagen (Instanz-Ebene) zur Klassifikation der Touren zu nutzen. So könnten fünf positive Vorhersagen (Instanz-Ebene) in kurzen zeitlichen Abstände eher auf einen Fehler hindeuten, als fünf positive Vorhersagen (Instanz-Ebene), die in relativ großem Abstand zueinander liegen. Auch sollte vermieden werden, dass die Klassifikation auf Tour-Ebene erst nach einer gewissen Mindestanzahl an Vorhersagen durchgeführt werden kann. Die Grundidee für die umgesetzte Lösung besteht darin, für jede Tour ein Aktivierungslevel a einzuführen, welches bei einer positiven Vorhersage auf Instanz-Ebene ansteigt, jedoch mit der Zeit um einen festen Betrag abfällt.

Um die Funktionsweise zu erläutern, sei eine beispielhafte Fehlertour gegeben, welche in Abbildung 19 (S. 54) graphisch dargestellt ist. Das Aktivierungslevel a wird zu Beginn der Tour auf den Wert 0 gesetzt. Wird nun auf Instanz-Ebene die positive Klasse vorhergesagt, so wird das Aktivierungslevel a angehoben. Die Erhöhung richtet sich dabei nach dem Konfidenzlevel der Vorhersage, welches angibt, wie sicher sich der Klassifizierer bei seiner Vorhersage ist. Bei einem Entscheidungsbaum kann dies beispielsweise durch die Verteilung der Klassen im Blatt berechnet werden.[222] Dies bietet den Vorteil, dass eine Vorhersage mit einem Konfidenzlevel nahe der 0,5 das Aktivierungslevel a nicht so stark erhöht, wie eine Vorhersage mit einem Konfidenzlevel von 0,99. Im konkreten Beispiel wird also zum Zeitpunkt 6 eine positive Vorhersage mit einem Konfidenzlevel von 0,7 auf Instanz-Ebene getätigt. Somit erhöht sich a um 0,7. Wie bereits erwähnt, sinkt das Aktivierungslevel a nun mit der Zeit. Im Beispiel sinkt a pro Zeiteinheit um 0,2 (Sinkrate s). Da bis zum Zeitpunkt 15 auf Instanz-Ebene keine positive Vorhersage erscheint, sinkt das Aktivierungslevel a somit bis zum Zeitpunkt 10, bei dem es wieder den Wert 0 annimmt. Im weiteren Verlauf steigt das Aktivierungslevel a nach dem gleichen Prinzip noch mehrere Male an bzw. fällt ab.

[222] Vgl. Quinlan (1986), S. 98 f.

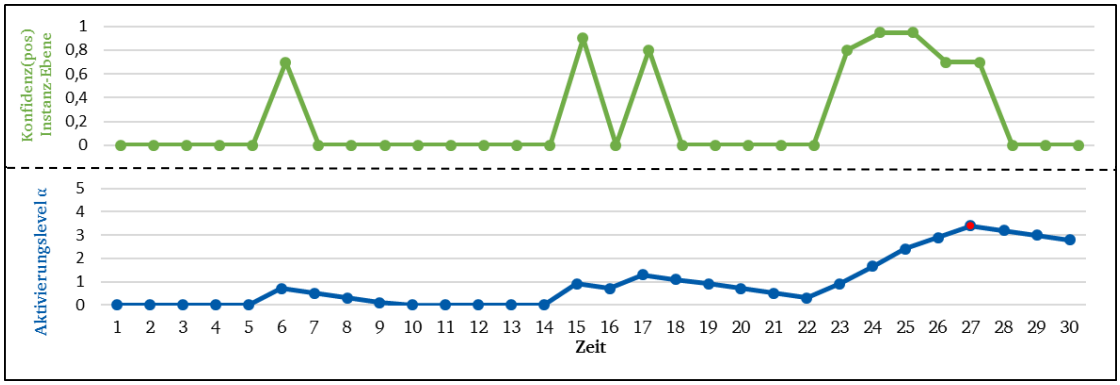

Abbildung 19: Beispiel Fehlertour (Aktivierungsklassifizier)

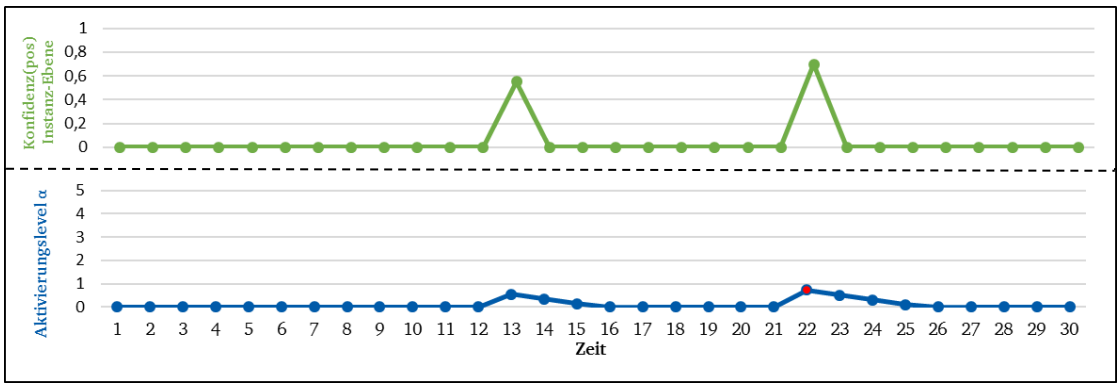

Abbildung 20: Beispiel Normaltour (Aktivierungsklassifizier)

Zum Vergleich ist in Abbildung 20 eine weitere Tour gegeben, welche nur relativ selten und zeitlich verteilt eine positive Vorhersage auf Instanz-Ebene vorweist. Es wird im Folgenden angenommen, dass diese Tour ein Beispiel für eine Normaltour ist. Gesucht ist nun, wie in Abschnitt 4.1.3.1 eine Schwellenwert t, welcher die Normaltouren von Fehlertouren anhand des Aktivierungslevel separiert. Der einzige Unterschied ist, dass die Datenpunkte nun das Aktivierungslevel a und nicht das Verhältnis r darstellen. Zum Lernen eines Schwellenwertes t wird also wie in Abschnitt 4.1.3.1 vorgegangen:

1. Extrahiere die relevanten Instanzen aus den Trainingstouren.
2. Lerne hieraus ein Klassifikationsmodell für die Instanz-Ebene.
3. Wende dieses Klassifikationsmodell auf die Trainingstouren an.
4. Berechne aus den Vorhersagen die Aktivierungslevel und erstelle für die Tour-Ebene einen Trainingsdatensatz, wobei für jede Tour ein Trainingsbeispiel, bestehend aus dem maximalen Aktivierungslevel und der Klasse, erzeugt wird.

5. Lerne auf Tour-Ebenen Trainingsdatensatz einen kostensensitiven Entscheidungsbaum der Tiefe 1 nach dem Algorithmus von Ling et al. (2004).

Zusammengefasst gibt dieser Klassifizierer also eine Warnung an den Lokführer über einen drohenden Stromrichterausfall heraus, sobald ein bestimmtes Aktivierungslevel überschritten wird. Touren werden bei der Evaluierung also in Normal- und Fehlertouren abhängig von ihrem Aktivierungslevel eingeteilt.

4.2 Durchführung

Nachdem die neuen Verfahren entwickelt wurden, war es zur Beantwortung der ersten Forschungsfrage notwendig, zu überprüfen, ob die neuen Verfahren zu einer Verbesserung der Klassifikationsleistung führen. Hierzu wurde das System inklusive der alten und neuen Verfahren zunächst implementiert. Anschließend folgte die Evaluierung des neuen und alten Systems mit Hilfe des Hochleistungsclusters der TU Darmstadt. Der folgende Abschnitt erläutert die beiden Punkte und liefert so eine Beschreibung der Durchführung der Evaluierung. Der erste Abschnitt beginnt dabei mit der Implementierung. Unter anderem wird beschrieben, welche Skripte und Klassen existieren, welche Aufgaben diese besitzen und wie diese interagieren. Der zweite Abschnitt hingegen beschäftigt sich mit dem Evaluierungsprozess. Es wird erläutert, wie die Trainings- und Testdaten ausgewählt, welche Parameterkonstellationen ausprobiert und wie die daraus resultierenden Versuche auf dem Cluster ausgeführt wurden.

4.2.1 Implementierung

Die Implementierung des Systems wurde, wie in Abbildung 21 (S: 56) schematisch dargestellt, in zwei Teile unterteilt. Der erste Teil besitzt die Aufgabe, aus den Rohdaten einmalig die Menge von Instanzen zu erzeugen, auf welchen später gelernt und getestet werden kann. Es sind also das bisherige und das neue Verfahren zur Datentransformation und Attributauswahl umgesetzt worden. Dabei wurde dieser Teil in der Skriptsprache PHP implementiert. Der zweite Teil hingegen übernimmt im Wesentlichen das Labeln der Instanzen, die Klassifizierung auf Tour- und Instanz-Ebene und die Evaluierung. Dieser Teil wurde in Java mit Hilfe der Weka-Bibliothek programmiert. Im Folgenden sollen die beiden Teile nun genauer untersucht werden.

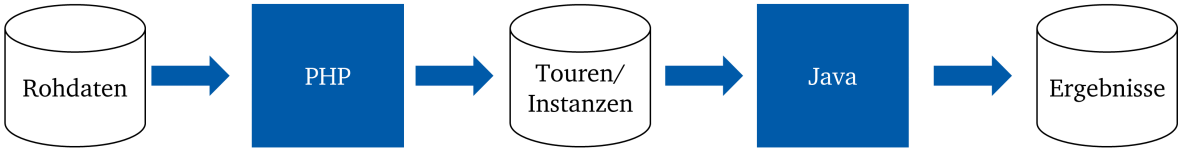

Abbildung 21: Schema Implementierung

Wie bereits im vorigen Absatz erwähnt, wurden im ersten Teil die beiden Verfahren zur Datentransformation und Attributauswahl aus Abschnitt 3.2.1 und 4.1.1 implementiert. Der Kern der Verfahren wurde dabei in den Skripten *dataset_binary.php* und *dataset_window.php* umgesetzt. Das Skript *dataset_binary.php* ist dabei für das bisherige Verfahren verantwortlich, in welchem die Instanzen den Zustand zu einem bestimmten Zeitpunkt darstellen. Das Skript *dataset_window.php* setzt das neu entwickelte Verfahren um, welches zusätzlich noch den zeitlichen Verlauf der Diagnosecodes in Form der drei Zeitverlaufsmaßen in die Instanzen integriert.

Grundlegend gehen die beiden Skripte nahezu identisch vor. Nachdem die relevanten Diagnosecodes und Umfelddaten in Form eines Arrays definiert wurden, findet zunächst eine Auswahl statt, für welche Touren die Instanzen erzeugt werden sollen. Dabei werden nur Touren selektiert, in welchen eine Strecke von mindestens 100 km zurückgelegt wurde. Daraufhin werden für jede dieser Touren alle Zeitpunkte ermittelt, bei dem sich am Zustand der Lok etwas verändert hat. Anschließend wird für alle dieser Zeitpunkte eine Instanz in Form der Diagnosecode-Attribute und Umfeld-Attribute erzeugt. Beim Skript *dataset_window.php* werden zusätzlich die drei Verlaufsattribute für jeden Diagnosecode berechnet. Dies wird dabei durch das Mitführen des zeitlichen Verlaufs der Diagnosecodes innerhalb eines gleitenden Zeitfensters w ermöglicht. Die erzeugten Instanzen werden anschließend verknüpft mit ihrer Tour in eine Datenbanktabelle geschrieben, welche letztlich die Instanzen für die relevanten Touren in zwei Konfigurationen enthält.

Für die Umsetzung des zweiten Teils wurden mehrere Java-Klassen implementiert, welche mit den Klassen aus der Weka-Bibliothek interagieren. Die grundlegende Architektur ist in Abbildung 22 (S. 57) abgebildet. Begonnen wird mit der *TourManager* Klasse. Diese ist dafür verantwortlich, eine Auswahl der Touren für das Training und das Testen zu erzeugen. Hierzu erhält die Klasse jeweils die Anzahl an Normal- und Fehlertouren. Anschließend wählt sie gemäß dieser Werte eine Menge an Touren für Training und Test aus, welches sie anschließend im Weka-internen ARFF[223]-Format abspeichert.

[223] ARFF = Attribute-Relation File Format.

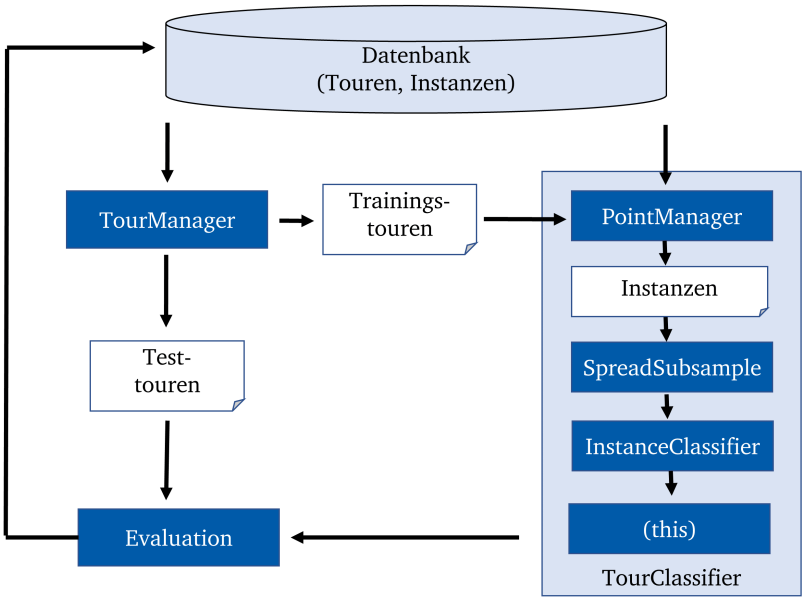

Abbildung 22: Architektur Java

Die Trainingstouren werden nun zum Lernen dem *TourClassifier* übergeben. Dieser besitzt die Aufgabe, ein Klassifikationsmodell auf Tour-Ebene mit Hilfe eines Klassifikationsmodells auf Instanz-Ebene bereitzustellen. Um das Tour-Ebenen Klassifikationsmodell zu erstellen, stößt dieser also zunächst die Erstellung des Instanz-Ebenen Klassifikationsmodells an. Hierzu werden die Trainingstouren (bestehend aus Normal- und Fehlertouren) zunächst einem *PointManager* übergeben. Dieser lädt die zu den Touren zugehörigen Instanzen aus der Datenbank und versieht diese (gegeben des Labeling-Zeitfensters *w*) mit einem positiven oder negativen Label. Um dabei die Datenbank nicht unnötig zu belasten, werden die heruntergeladenen Instanzen auf dem Dateisystem abgespeichert. So wird vermieden, dass bei einer parallelen Ausführung auf dem Cluster die Datenbank für die gleiche Trainingsdatenmenge mehrere Male abgefragt werden muss. Um die Verteilung der Klassen innerhalb der Instanzen anzupassen, wird anschließend der *SpreadSubsample* Filter aus der Weka Bibliothek verwendet. Dieser erzeugt eine Stichprobe der Mehrheitsklasse, sodass die Klassenverteilung einem gegebenen, festen Verhältnis folgt.[224] Nachdem nun die Instanzen der Trainingstouren eingelesen und die Klassenverteilung derer angepasst wurde, wird die Trainingsmenge nun dem jeweiligen InstanceClassifier übergeben. Hierzu werden die Klassen *JRip*, *J48*, *RandomForest*, *AdaBoostM1* und *REPTree* von Weka eingesetzt.

[224] Vgl. Inglis (2016).

Nachdem diese ihr Klassifikationsmodell erstellt haben, folgt nun bei den neuen Tour-Ebenen Klassifizierern die Erstellung ihres Modells. Hierzu wurden die beiden in Abschnitt 4.1.3 beschriebenen Verfahren in den Klassen *BinaryRatioLearnTourClassifier* und *BinaryActivation-TourClassifier* implementiert. Diese wenden nun den *InstanceClassifier* auf die Trainingstouren an und erstellen auf Grundlage deren Vorhersage das Tour-Ebenen Klassifikationsmodell. Das in Abschnitt 4.1.3 beschriebene kostensensitive Lernen eines Entscheidungsbaumes der Tiefe 1 wurde in der Klasse *DecisionStumpCostSensitiv* umgesetzt. Dies basiert auf der Weka-Klasse *DecisionStump*, ersetzt jedoch den Information Gain durch das Kostenkriterium von Ling et al. (2004), um so die Vorgehensweise aus Abschnitt 4.1.3.1 und 4.1.3.2 umzusetzen.[225] Der ursprüngliche Tour-Ebenen Verhältnisklassifizierer wurde hingegen in der Klasse *BinaryRatioClassifier* implementiert.

Durch die Kapselung der *TourClassifiers* ist es nun möglich die *Evaluation*-Klasse von Weka für die Berechnung der Evaluierungsmaße zu verwenden. Es werden also die vom *TourManager* erzeugten Testtouren an den jeweiligen *TourClassifier* übergeben. Anschließend wird für jede Tour eine Klasse vorhergesagt, indem der *PointManger* zunächst die Instanzen der Tour bereitstellt, anschließend der *InstanceClassifier* die Instanzen nacheinander klassifiziert und währenddessen das Klassifikationsmodell des *TourClassifier* auf den Vorhersageverlauf angewandt wird. Die *Evaluation*-Klasse vergleicht anschließend für jede Tour die reale Klasse mit der vorhergesagten Klasse und berechnet auf deren Grundlage die Evaluierungsmaße. Diese werden anschließend zusammen mit dem Vorhersageverlauf und den Klassifikationsmodellen auf Tour- und Instanz-Ebene in der Datenbank abgespeichert.

Eine Übersicht aller Skripte und Klassen ist letztlich im Anhang A angefügt. Zusätzlich ist der gesamte, dokumentierte Quelltext auf dem beigefügten Datenträger mitgeliefert.

4.2.2 Vorgehensweise

Nachdem die alten und neuen Verfahren implementiert wurden, galt es nun diese zu evaluieren. Das Ziel war zu überprüfen, ob die neuen Verfahren die Klassifikationsleistung verbessern können. Der folgende Abschnitt beschreibt deshalb, wie bei der Evaluierung vorgegangen wurde.

Grundlegend wurde zur Evaluierung die in Abbildung 23 (S. 59) dargestellte Architektur verwendet. Zunächst wurde aus der Menge aller Touren (mit min. 100km Länge) eine

[225] Siehe Abschnitt 4.1.3.1.

Trainings- und Testmenge erstellt. Hierzu wurden in die Trainingsmenge 32 Fehlertouren und 418 Normaltouren aufgenommen, in die Testmenge hingegen 8 Fehlertouren und 18.262 Normaltouren. Die Anzahl der Fehler- und Normaltouren in beiden Mengen hat dabei unterschiedliche Gründe. Die Fehlertouren wurden so aufgeteilt, dass sie einem 80:20 Verhältnis folgen. Die Anzahl an Normaltouren für die Testmenge hingegen wurde so gewählt, dass sie das Verhältnis von Normal- und Fehlertouren aus der Realität widerspiegeln. Für die Trainingstouren musste das Verhältnis jedoch angepasst werden, da ansonsten $\frac{91309}{40} * 32 \approx$ 73.047 Normaltouren hätten verwendet werden müssen. Dies hätte bedeutet, dass der Tour-Ebenen Klassifizierer mit einer hohen Ungleichverteilung der Klassen konfrontiert gewesen wäre, was gerade beim Trainieren des Modells zu Problemen führen kann.[226] Es wurde deshalb entschieden, 418 Normaltouren zu verwenden, sodass die Trainingsmenge insgesamt aus 450 Touren besteht.

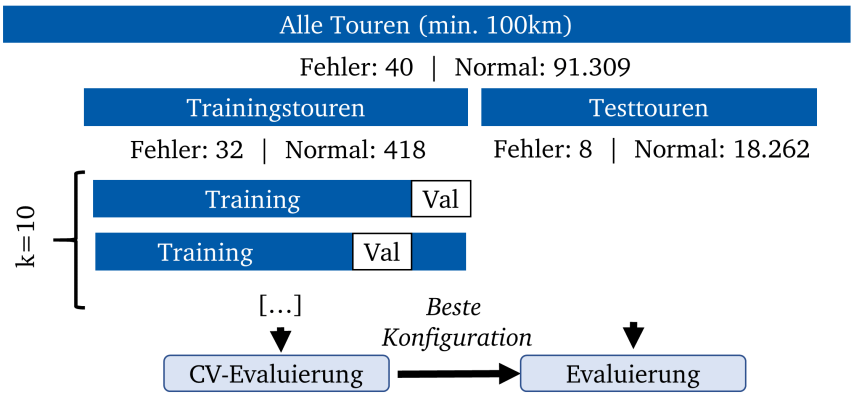

Abbildung 23: Architektur der Evaluierung

Auf Grundlage der beiden Mengen wurde nun wie in Kapitel 2.4.1 erläutert vorgegangen. Zunächst wurde die optimale Konfiguration sowohl für das bisherige System von Kauschke et al. (2016) als auch für das weiterentwickelte System mit Hilfe eine Gittersuche ermittelt. Hierzu wurde für jede Konfiguration des alten und neuen Systems eine stratifizierte Kreuzvalidierung (CV) mit 10 Teilmengen durchgeführt und anschließend mit Hilfe der Evaluationsmaße aus Kapitel 2.4.2 bestimmt, wie hoch deren Klassifikationsleistung ist. Mit Hilfe dieser Maße war es nun möglich, die Klassifikationsleistung auf der Kreuzvalidierung zwischen alten und neuem System zu vergleichen. Wichtig hierbei ist jedoch anzumerken, dass die Klassifikationsleistung der besten Konfigurationen auf der Kreuzvalidierung mit einem Bias versehen ist und somit häufig nicht die reale Klassifikationsleistung auf einem zuvor unbekannten Datensatz

[226] Vgl. He;Garcia (2009), S. 1263 ff.

widerspiegeln. Dies hat den Grund, dass gerade die Konfiguration gewählt wird, die am besten auf der Kreuzvalidierung funktioniert. Somit werden Parameter speziell an die gegebene Trainingsmenge angepasst.[227] Um also zu ermitteln, wie gut die besten Konfigurationen der beiden Systeme die Touren auf zuvor ungesehenen Daten klassifizieren können, wurden die besten Konfigurationen der beiden Systeme (bei der Kreuzvalidierung) auf der gesamten Trainingsmenge trainiert und deren Klassifikationsleistung auf der Testmenge verglichen.

Als Konfiguration des bisherigen Systems wurden die in Abbildung 24 dargestellten Parameter verwendet. Insgesamt mussten hierfür 150 verschiedene Konstellationen im Rahmen der Kreuzvalidierung ausgeführt werden.

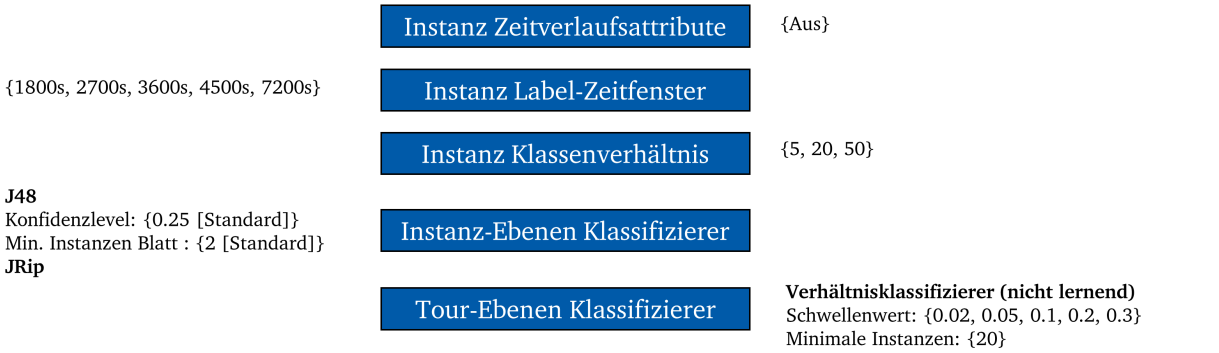

Abbildung 24: Konfigurationen des bisherigen Systems

Für die Konfiguration der neuen Verfahren wurden hingegen die in Abbildung 25 dargestellten Parameter verwendet. Hieraus resultieren insgesamt 2730 unterschiedliche Konfigurationen für die Kreuzvalidierung.

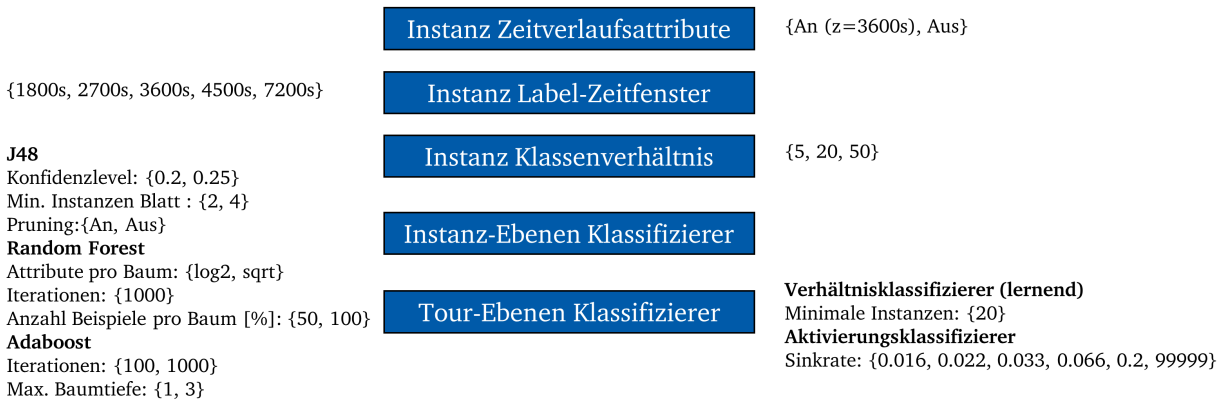

Abbildung 25: Konfigurationen des neuen Systems

[227] Vgl. Varma;Simon (2006), S. 91:1 ff.

Die insgesamt 2730+150=2880 Konfigurationen wurden anschließend auf dem Lichtenberg-Hochleistungscluster der TU Darmstadt ausgeführt. Hierfür wurde für jede Konfiguration ein Slurm-Skript[228] generiert, welches anschließend an das Cluster übermittelt wurde. Insgesamt waren ca. 3400 Rechenstunden für die Durchführung der Kreuzvalidierung notwendig, wobei mehrere hundert Skripte parallel ausgeführt werden konnten.

4.3 Ergebnisse

Im folgenden Abschnitt sollen nun die Ergebnisse der Evaluierung beschrieben und diskutiert werden. Hierzu unterteilt sich der Abschnitt grundlegend in zwei Teile. Im ersten Teil werden die Ergebnisse der Kreuzvalidierung bzw. die Ergebnisse der Parameteroptimierung präsentiert. Im zweiten Teil werden anschließend die Ergebnisse auf der Testmenge vorgestellt.

4.3.1 Kreuzvalidierung

Wie in Kapitel 4.3.1 beschrieben, wurden für die Kreuzvalidierung 2880 unterschiedliche Konfigurationen auf dem Cluster ausgeführt. Von diesen konnten 2851 erfolgreich berechnet werden, während 29 aufgrund von Speicher- oder Zeitrestriktionen frühzeitig abgebrochen wurden. Durchschnittlich dauerte eine Berechnung pro Konfigurationen ca. 1 Stunde und 11 Minuten.

Die Ergebnisse der Kreuzvalidierung sind in Tabelle 3 (S. 62) dargestellt. Wie zu erkennen, sind die Konfigurationen dabei absteigend nach der **Accuracy** sortiert.[229] Auch ist nur eine Auswahl aller Konfigurationen dargestellt. So stellen die ersten 10 Zeilen der Tabelle die besten aller Konfigurationen dar (in weiß markiert). Die restlichen Zeilen hingegen zeigen die besten Konfigurationen in Bezug auf alle Kombinationen der Parameter Zeitverlaufsattribute, Instanz-Ebenen Klassifizierer und Tour-Ebenen Klassifizierer (also jene Ebenen des Systems, für welche neue Verfahren entwickelt wurden). Beispielsweise erreicht so die beste Konfiguration, bei dem das Zeitverlaufsattribut = an, der Instanz-Ebenen Klassifizierer = Random Forest und der Tour-Ebenen Klassifizierer = Verhältnisklassifizierer (Lernend) eine Accuracy von 93,111%. Die besten Kombinationen dieser drei Ebenen sind dabei für das neue System in hellgrau und für das alte System in dunkelgrau unterlegt.

[228] Für weitere Detail zu Slurm siehe Slurm (2016).

[229] Bei Gleichheit der Accuracy wurde die Konfiguration nach F1-Measure, aTTF und mTTF sortiert (in dieser Reihenfolge). Waren alle diese Maße gleich, so entschied der Zufall. Der Rang richtet sich jedoch ausschließlich nach der Accuracy.

Tabelle 3: Ergebnisse Versuchsreihe 1 sortiert nach Accuracy (Kreuzvalidierung)

	Instanz-Ebene							Tour-Ebene		Maße								
Rang	Zeitverlaufs-attribut	Label-Zeitfenster	Klassen-Verhältnis	Klassifizierer	J48 Pruning RF Anzahl/Baum Adaboost Iteration	J48 Konf.-Level RF Iterationen Adaboost Baumtiefe	J48 Anzahl Blatt RF Anzahl Attr.	Klassifizierer	Verh. Schwellenwert Akt. Sinkrate	TP	TN	FP	FN	F1	Precision	Recall	aTTF (min)	mTTF (min)	Accuracy (%)
1	An	3600	5	Adaboost	1000	1		Aktivierung	0,0333	5	416	2	27	0,256	0,714	0,156	470	27	93,556
1	An	4500	20	Adaboost	1000	1		Aktivierung	0,2	4	417	1	28	0,216	0,8	0,125	878	318	93,556
1	An	4500	20	Adaboost	1000	1		Aktivierung	0,0667	4	417	1	28	0,216	0,8	0,125	870	318	93,556
1	An	4500	20	Adaboost	1000	1		Aktivierung	0,0333	4	417	1	28	0,216	0,8	0,125	870	318	93,556
1	An	3600	50	Adaboost	1000	1		Aktivierung	0,2	3	418	0	29	0,171	1	0,094	526	34	93,556
1	An	3600	50	Adaboost	100	1		Aktivierung	0,2	3	418	0	29	0,171	1	0,094	526	34	93,556
7	An	3600	5	Adaboost	100	1		Aktivierung	0,0333	5	415	3	27	0,25	0,625	0,156	467	10	93,333
7	An	3600	5	Adaboost	1000	1		Aktivierung	0,0667	4	416	2	28	0,211	0,667	0,125	535	34	93,333
7	An	4500	20	Adaboost	100	1		Aktivierung	0,2	3	417	1	29	0,167	0,75	0,094	742	318	93,333
7	An	4500	20	Adaboost	1000	1		Aktivierung	0,0222	3	417	1	29	0,167	0,75	0,094	742	318	93,333
19	An	4500	5	RF	50	1000	sqrt	Verhältnis (L)		4	415	3	28	0,205	0,571	0,125	207	68	93,111
19	An	2700	20	Adaboost	1000	1		Verhältnis (L)		3	416	2	29	0,162	0,6	0,094	521	35	93,111
19	An	2700	5	RF	100	1000	sqrt	Aktivierung	0,0333	3	416	2	29	0,162	0,6	0,094	210	66	93,111
19	Aus	1800	5	J48	Aus			Aktivierung	0,0167	3	416	2	29	0,162	0,6	0,094	196	40	93,111
19	Aus	3600	5	RF	50	1000	sqrt	Aktivierung	0,0333	2	417	1	30	0,114	0,667	0,062	105	51	93,111
19	Aus	1800	20	Adaboost	100	1		Aktivierung	0,0333	1	418	0	31	0,061	1	0,031	172	172	93,111
102	Aus	1800	5	J48	Aus			Verhältnis (L)		3	415	3	29	0,158	0,5	0,094	579	166	92,889
102	Aus	3600	20	Adaboost	100	1		Verhältnis (L)		1	417	1	31	0,059	0,5	0,031	1138	1138	92,889
102	An	1800	20	J48	An	0,25	4	Aktivierung	99999	0	418	0	32	0	0	0	0	0	92,889
543	Aus	4500	5	RF	50	1000	sqrt	Verhältnis (L)		2	415	3	30	0,108	0,4	0,062	511	104	92,667
1029	An	1800	5	J48	An	0,25	2	Verhältnis (L)		0	416	2	32	0	0	0	0	0	92,444
1266	Aus	4500	50	J48	An	0,25	2	Verhältnis (NL)	0,3	4	411	7	28	0,186	0,364	0,125	464	112	92,222
1469	Aus	1800	50	JRip				Verhältnis (NL)	0,3	2	412	6	30	0,1	0,25	0,062	377	178	92

RF = Random Forest,
Verhältnis (L) = Lernender Verhältnisklassifizierer, Verhältnis (NL) = Nicht Lernender Verhältnisklassifizierer
aTTF = Durchschnittliche Warnzeit vor dem Fehler, mTTF = Minimale Warnzeit vor dem Fehler

Zu jeder Konfiguration sind neben der Accuracy die True Positives, die True Negatives, die False Positives, die False Negatives, der Recall, die Precision und die F-Measure angegeben. Dabei stellen die Fehlertouren die positive und die Normaltouren die negative Klasse dar. Zusätzlich hierzu wurde jede Konfiguration mit den Maßen aTTF und mTTF versehen. Das Maß aTTF gibt an, wie weit vor dem Fehler die Warnung über einen drohenden Ausfall im Durschnitt herausgegeben wurde. Das Maß mTTF hingegen gibt die minimale Warnzeit vor einem Stromrichterausfall an.

Zusätzlich zu Tabelle 3 (S. 62) ist in Abbildung 26 ein Diagramm gegeben, welche die Accuracy der besten Konfigurationen gruppiert nach den zuvor beschriebenen drei Parametern darstellt. Auch hier sind die Verfahren des alten Systems in dunkelgrau und die Verfahren des neuen Systems in hellgrau gefärbt. Zusätzlich ist die Accuracy der Baseline in blau eingezeichnet. Dabei wird als Baseline ein Klassifizierer angenommen, der nur die Mehrheitsklasse vorhersagt. Im konkreten Fall würde er also nur Normaltouren vorhersagen und so eine Accuracy von $\frac{418}{450} = 92{,}889\%$ erreichen.

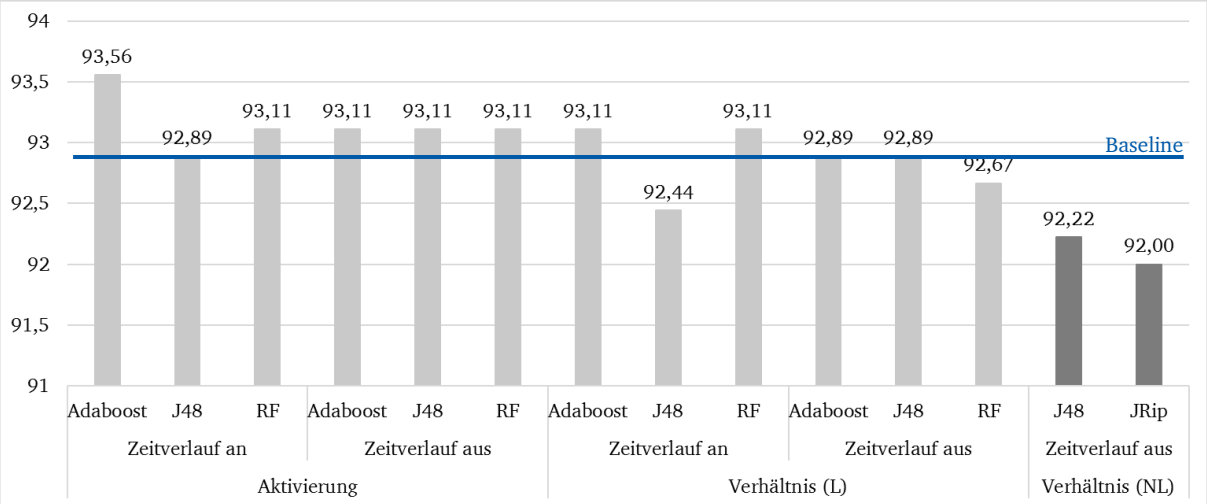

Abbildung 26: Vergleich der Konfigurationen nach Accuracy [%] (Kreuzvalidierung)

Wie in der Tabelle 3 (S. 62) und in Abbildung 26 zu erkennen, schneiden alle Kombinationen der neuen Verfahren in Bezug auf die Accuracy besser ab als die der alten Verfahren. Auch ist zu erkennen, dass bis auf zwei der neuen Kombinationen alle mindestens so gut wie die Baseline abschneiden, wohingegen die alten Verfahren stets eine schlechtere Accuracy als die Baseline aufweisen. Die beste Konfiguration der neuen Verfahren besitzt dabei eine Accuracy von 93,56%, wohingegen die beste Konfiguration der alten Verfahren eine Accuracy in Höhe von

92,22% erreicht. Durch Verwendung der neuen Verfahren konnte die Accuracy bei der Kreuzvalidierung also um 1,34% gesteigert werden.

Die beste Konfiguration der neuen Verfahren wird dabei durch die Kombination von Zeitverlaufsattributen und AdaBoost auf Instanz-Ebene und dem Aktivierungsklassifizierer auf Tour-Ebene erreicht. Hierbei konnten 5 der 32 Touren als Fehlertouren detektiert werden, wohingegen nur 2 der 418 Normaltouren fehlklassifiziert wurde. Relativ ausgedrückt konnten also 15,6% der Stromrichterausfälle erkannt werden, wohingegen in 0,48% der Fälle eine Normaltour als Fehlertour klassifiziert wurde. Wurde bei dieser Konfiguration also eine Tour als Fehlertour klassifiziert, so war dies zu 71,4% auch eine Tour, in welcher ein Stromrichter ausgefallen war. Dabei wäre der Lokführer im Schnitt 7,83 Stunden im Voraus gewarnt worden.

Die beste Konfiguration der alten Verfahren wird hingegen durch die Verwendung des J48 Klassifizierers und einem Schwellenwert von 0,3 auf Tour-Ebene erreicht. Diese erkannte 4 der 32 Fehlertouren, machte jedoch bei 7 der Normaltouren einen Fehler. Somit wurden 12,5% der Fehlertouren erkannt, jedoch wird in 1,67% der Fälle ein falscher Alarm herausgegeben. Wurde also eine Warnung herausgegeben, so war dieser in 36,4% der Fälle korrekt. Zusätzlich wäre der Lokführer im Schnitt 7,73 Stunden im Voraus gewarnt worden.

Neben dem Vergleich der Konfigurationen anhand der Accuracy, können die verschiedenen Konfigurationen auch anhand der **F1-Measure** verglichen und sortiert werden. Wie in Abschnitt 2.4.2 beschrieben, ermöglicht diese Messung eine Bewertung der Klassifikationsleistung auch bei ungleichen Klassenverteilung, wie dies in der konkreten Problemstellung der Fall ist. Aus diesem Grund sind in Tabelle 4 (S. 65) eine Auswahl der Konfigurationen, sortiert nach ihrem F1-Wert, abgebildet.[230] Wie auch in Tabelle 3 (S. 62) sind dabei zunächst die insgesamt 10 besten Konfigurationen angegeben (in weiß markiert). Die restlichen Zeilen zeigen wiederum die besten Konfigurationen in Bezug auf alle Kombinationen der Parameter Zeitverlaufsattribut, Instanz-Ebenen Klassifizierer und Tour-Ebenen Klassifizierer (also jener Ebenen, für welche neue Verfahren entwickelt wurden). Die besten Konfigurationen der alten Verfahren sind dabei in dunkelgrau und die der neuen Verfahren in hellgrau hervorgehoben. Zur Visualisierung sind die besten F1-Werte oben genannten Kombination in Abbildung 27 (S. 66), wie auch zuvor, als Diagramm dargestellt.

[230] Bei Gleichheit des F1-Wertes wurde die Konfiguration nach Accuracy, aTTF und mTTF sortiert (in dieser Reihenfolge). Waren alle diese Maße gleich, so entschied der Zufall. Der Rang richtet sich jedoch ausschließlich nach dem F1-Wert.

Tabelle 4: Ergebnisse Versuchsreihe 1 sortiert nach F1-Measure (Kreuzvalidierung)

	Instanz-Ebene							Tour-Ebene		Maße								
Rang	Zeitverlaufs-attribut	Label-Zeitfenster	Klassen-Verhältnis	Klassifizierer	J48 Pruning RF Anzahl/Baum Adaboost Iteration	J48 Konf.-Level RF Iterationen Adaboost Baumtiefe	J48 Anzahl Blatt RF Anzahl Attr.	Klassifizierer	Verh. Schwellenwert Akt. Sinkrate	TP	TN	FP	FN	Accuracy (%)	Precision	Recall	aTTF (min)	mTTF (min)	F1
1	Aus	7200	20	J48	An	0,5	4	Aktivierung	0,0167	10	398	20	22	90,667	0,333	0,312	673	61	0,323
1	Aus	7200	20	J48	An	0,5	4	Aktivierung	0,0222	10	398	20	22	90,667	0,333	0,312	672	61	0,323
1	Aus	7200	20	J48	An	0,5	4	Aktivierung	0,0333	10	398	20	22	90,667	0,333	0,312	665	61	0,323
4	Aus	3600	20	J48	Aus			Aktivierung	0,0167	9	401	17	23	91,111	0,346	0,281	519	103	0,31
5	Aus	7200	20	J48	An	0,25	4	Aktivierung	0,0167	9	399	19	23	90,667	0,321	0,281	737	61	0,3
5	Aus	7200	20	J48	An	0,25	4	Aktivierung	0,0222	9	399	19	23	90,667	0,321	0,281	736	61	0,3
5	Aus	7200	20	J48	An	0,25	4	Aktivierung	0,0333	9	399	19	23	90,667	0,321	0,281	734	61	0,3
8	An	4500	20	J48	Aus			Aktivierung	0,0667	10	393	25	22	89,556	0,286	0,312	383	13	0,299
9	An	4500	20	J48	Aus			Aktivierung	0,0222	10	392	26	22	89,333	0,278	0,312	383	13	0,294
9	An	4500	20	J48	Aus			Aktivierung	0,0167	10	392	26	22	89,333	0,278	0,312	383	13	0,294
12	Aus	4500	5	JRip				Verhältnis (NL)	0,2	12	379	39	20	86,889	0,235	0,375	874	88	0,289
29	Aus	3600	50	J48	An	0,25	2	Verhältnis (NL)	0,1	8	399	19	24	90,444	0,296	0,25	458	89	0,271
30	Aus	3600	50	J48	An	0,25	2	Verhältnis (L)		8	398	20	24	90,222	0,286	0,25	474	112	0,267
42	An	1800	20	Adaboost	100	3		Aktivierung	0,0167	6	410	8	26	92,444	0,429	0,188	226	15	0,261
125	An	4500	50	J48	Aus			Verhältnis (L)		9	385	33	23	87,556	0,214	0,281	396	59	0,243
261	An	1800	20	Adaboost	100	3		Verhältnis (L)		5	409	9	27	92	0,357	0,156	197	17	0,217
337	An	4500	5	RF	50	1000	sqrt	Verhältnis (L)		4	415	3	28	93,111	0,571	0,125	207	68	0,205
431	Aus	7200	50	RF	100	1000	sqrt	Aktivierung	0,0667	5	403	15	27	90,667	0,25	0,156	410	75	0,192
441	Aus	7200	20	Adaboost	1000	3		Aktivierung	0,0167	4	412	6	28	92,444	0,4	0,125	174	56	0,19
660	An	2700	5	RF	100	1000	sqrt	Aktivierung	0,0333	3	416	2	29	93,111	0,6	0,094	210	66	0,162
786	Aus	7200	20	Adaboost	100	3		Verhältnis (L)		3	413	5	29	92,444	0,375	0,094	532	112	0,15
786	Aus	7200	5	RF	100	1000	sqrt	Verhältnis (L)		3	413	5	29	92,444	0,375	0,094	368	75	0,15

RF = Random Forest,
Verhältnis (L) = Lernender Verhältnisklassifizierer, Verhältnis (NL) = Nicht Lernender Verhältnisk
aTTF = Durchschnittliche Warnzeit vor dem Fehler, mTTF = Minimale Warnzeit vor dem Fehler

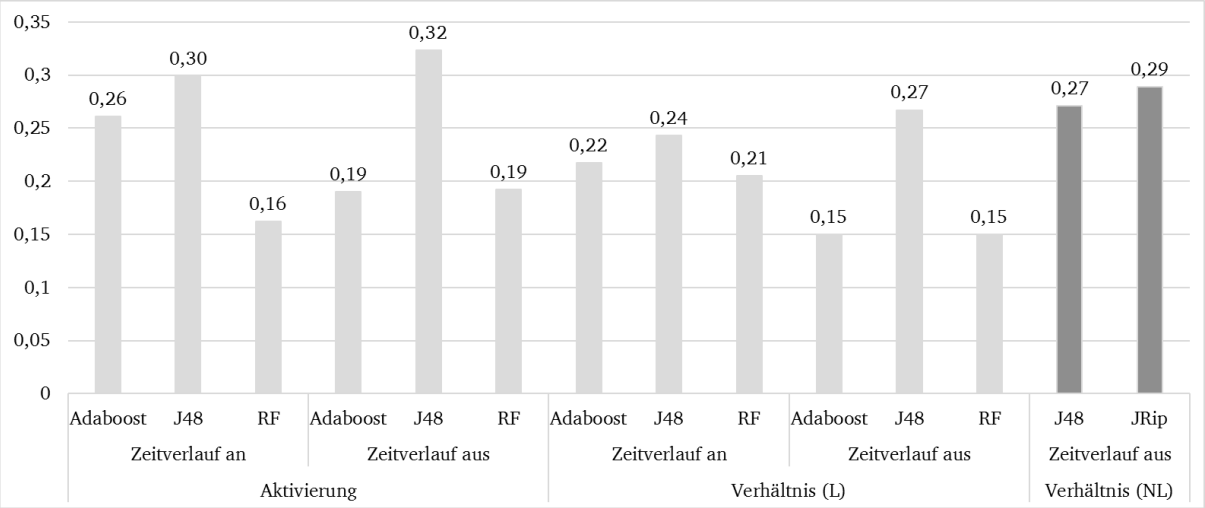

Abbildung 27: Vergleich der Konfigurationen nach F1-Measure (Kreuzvalidierung)

Wie Tabelle 4 (S. 65) und Abbildung 27 zu entnehmen, ergibt sich hier ein anderes Bild als zuvor. Bezogen auf den F1-Wert sind nur zwei Kombinationen der neuen Verfahren besser als die beiden Kombinationen der bisherigen Verfahren. Die restlichen 10 Kombinationen der neuen Verfahren schneiden hingegen bezogen auf die F1-Measure schlechter ab. Ein möglicher Grund hierfür könnte das pessimistische Kostenverhältnis der Tour-Ebenen Klassifizierers darstellen. Nichtdestotrotz ist auch hier die beste Konfiguration der neuen Verfahren mit einen F1-Wert in Höhe von 0,323 besser als die beste Konfiguration der alten Verfahren mit einem F1-Wert in Höhe von 0,289.

Die beste Konfiguration der neuen Verfahren (in Bezug auf die F1-Measure), ist durch die Kombination von ausgeschaltetem Verlaufsattributen und dem J48 Algorithmus in Kombination mit dem Aktivierungsklassifizierer auf Tour-Ebene gegeben. In diesem Fall wurden 10 der 32 Fehlertouren korrekt klassifiziert, also 31,2% der Stromrichterausfälle im Voraus erkannt. Auf der anderen Seite wurden 20 der Normaltouren als Fehlertouren klassifiziert. Dies entspricht einer Fehlalarmrate von 4,8%. Im Schnitt wurde die Warnung eines drohenden Stromrichter-ausfalls dabei 11,21 Stunden im Voraus herausgegeben, jedoch mindestens 1,02 Stunden zuvor.

Im Vergleich wird der beste F1-Wert der Konfigurationen mit alten Verfahren bei Verwendung des JRip Klassifkators erreicht. Die Konfiguration erkannte 12 der 32 Fehlertouren und detektierte somit 37,5% der Stromrichterausfälle im Voraus. Jedoch wurden bei dieser Konfiguration 39 Normaltouren fälschlicherweise als Fehlertouren klassifiziert, was eine Fehlalarmrate von 9,3% entspricht. Die Stromrichterausfälle konnten dabei im Schnitt 14,6 Stunden im Voraus erkannt werden, jedoch mindestens 1,5 Stunden zuvor.

Zusammenfassend kann also gesagt werden, dass die neuen Verfahren sowohl bezogen auf die Accuracy als auch auf die F1-Measure eine Verbesserung erreichen konnten. Dabei muss jedoch angemerkt werden, dass durch Auswahl des besten Modells die Klassifikationsleistung tendenziell überschätzt wird, da ja gerade die Parameter ausgewählt werden, die auf der Menge an Beispielen am besten funktioniert.[231] Aus diesem Grund folgt im kommenden Abschnitt die Anwendung der besten Konfigurationen auf eine zuvor ungesehene Testmenge. Da jedoch bei beiden Maßen unterschiedliche Konfigurationen, sowohl bei den neuen als auch bei den alten Verfahren die besten Ergebnisse lieferten, wurden insgesamt vier Konfigurationen auf die Testmenge angewandt.

4.3.2 Testmenge

Es sollten nun also die jeweils besten Konfigurationen (bezogen auf die Accuracy und F1-Measure der Kreuzvalidierung) der neuen und alten Verfahren auf die Testmenge angewandt und verglichen werden. Hierzu wurde die gesamte Trainingsmenge zum Lernen des Modells verwendet. Es wurde also auf insgesamt 450 Touren gelernt, wovon 32 Fehler- und 418 Normaltouren darstellten. Anschließend wurde das erzeugte Klassifikationsmodell auf die Testmenge angewandt, welche aus 8 Fehlertouren und 18.262 Normaltouren bestand. Ziel war es hierbei, die Leistung des alten und neuen Systems auf einer zuvor ungesehenen Menge zu messen und somit eine unverzerrte Leistungsbewertung zu erhalten. Die Ergebnisse (bezogen auf die Testmenge) sind in Tabelle 5 sortiert nach der Accuracy dargestellt.

Tabelle 5: Ergebnisse Versuchsreihe 1 (Testmenge)

Beste Konfiguration	Maße									
	TP	TN	FP	FN	Accuracy (%)	F1	Precision	Recall	aTTF (min)	mTTF (min)
Neue Verfahren (Accuracy)	0	18118	144	8	99,168	0	0	0		
Alte Verfahren (Accuracy)	0	17718	544	8	96,979	0	0	0		
Neue Verfahren (F1)	0	17419	843	8	95,342	0	0	0		
Alte Verfahren (F1)	0	16845	1417	8	92,2	0	0	0		

Zu erkennen ist, dass keine der Konfiguration eine der acht Fehlertouren korrekt klassifizieren konnte. Mit anderen Worten hat keine der Konfigurationen einen Stromrichterausfall im Voraus erkannt.

Hierfür sind mehrere Gründe vorstellbar. Ein Grund könnte die geringe Stichprobenanzahl an Fehlertouren darstellen. So ist es möglich, dass in der Testmenge durch Zufall nur solche

[231] Vgl. Varma;Simon (2006), S. 91:1 ff.

Stromrichterschäden enthalten sind, die auch während der Kreuzvalidierung nicht detektierbar waren. Zudem ergibt sich durch die geringe Stichprobenanzahl folgendes Phänomen: Nehmen wir an, die Wahrscheinlichkeit, eine Fehlertour zu erkennen, sei für alle Fehlertouren gleich und wir haben als Stichprobe die 8 Fehlertouren gegeben, die entweder als Fehler- oder Normaltour klassifiziert werden können. Diese Problemstellung spannt dabei ein 8-stufiges Bernoulli-Experiment auf, wobei die Erkennungsrate die Erfolgswahrscheinlichkeit darstellt.[232] Nehmen wir nun an, die beste Erkennungsrate der vier Konfigurationen aus der Kreuzvalidierung in Höhe von 37,5% wäre unverzerrt, so ist die Wahrscheinlichkeit keine der 8 Fehlertouren zu erkennen, gegeben durch

$$P(X \leq 0) = \binom{8}{0} * 0{,}375^0 * (1 - 0{,}375)^{8-0} = 0{,}0232 \approx 2{,}3\% \tag{13}$$

Also selbst in diesem Fall wäre die Wahrscheinlichkeit, keine der Fehlertouren zu erkennen noch bei 2,3%. Ein weiterer Grund könnte sein, dass die Erkennungsraten bei der Kreuzvalidierung durch die Auswahl der besten Parameterkonfigurationen sehr verzerrt sind, sodass das System auf zuvor ungesehenen Daten (und somit in der Realität) wesentlich schlechter abschneidet. Dies kann jedoch abschließend nur durch eine größere Stichprobe an Fehlertouren festgestellt werden.

Neben den erkannten Fehlertouren ist in Tabelle 5 (S. 67) ersichtlich, dass die beste Konfiguration der neuen Verfahren weiterhin eine höhere Accuracy als die beste Konfiguration der alten Verfahren besitzt. So werden bei der besten Konfiguration der neuen Verfahren nur 144 Normaltouren fehlklassifiziert, was bedeutet, dass in nur 0,79% der Normaltouren ein Fehlalarm herausgegeben wird. Bei der besten Konfiguration der alten Verfahren liegt die Fehlalarmrate hingegen bei 2,98%. Da bei den Normaltouren eine sehr große Stichprobe in Höhe von 18.262 Normaltouren vorhanden ist, besitzt diese Schätzung der Fehlalarmrate letztlich eine höhere Qualität als die der Erkennungsrate.

Zusammenfassend kann festgehalten werden, dass die Testmenge nur schwer eine Abschätzung der Erkennungsrate liefern kann. So könnten die schlechte Erkennungsrate an der geringen Anzahl an Fehlertouren in der Testmenge liegen oder aber darauf hindeuten, dass es durch die betrachteten Verfahren nicht möglich ist, einen Ausfall in der Realität im Voraus zu detektieren. Es konnte jedoch durch die Testmenge gezeigt werden, dass die Fehlalarmrate bei den neuen Verfahren bei zuvor ungesehenen Daten geringer ist als die der alten Verfahren.

[232] Weitere Informationen zu Bernoulli-Experimenten siehe Hartmann (2015), S. 514 ff.

5 Versuchsreihe 2: Multiklassen Vorhersage

Nachdem die erste Forschungsfrage im vorherigen Kapitel untersucht wurde, gilt es auch die zweite Forschungsfrage zu beantworten. Das Ziel der zweiten Forschungsfrage ist es, zu untersuchen, ob es neben der Warnung eines drohenden Stromrichterausfalls auch möglich ist, vorherzusagen, wie lange es noch bis zu diesem Ausfall dauert. Die Motivation besteht darin, dass der Lokführer abhängig von der vorhergesagten Zeit zum Stromrichterausfall unterschiedliche Strategien verfolgen kann. Wenn er beispielsweise nur noch eine halbe Stunde zur Verfügung hat, so kann er das nächste Abstellgleis anfahren. Sind jedoch noch 6 Stunden Zeit bis zum Ausfall, so ist es eventuell möglich, die Fahrt zu beenden und zur nächstgelegenen Werkstatt zu fahren. Die Idee ist also, nicht nur vorherzusagen, ob ein Stromrichterausfall in naher Zukunft droht, sondern im Falle eines drohenden Ausfalls dem Lokführer auch mitzuteilen, wie lange es noch bis zum Ausfall dauert.

Das Kapitel unterteilt sich (wie das vorherige) grundlegend in die Abschnitte Konzeption, Durchführung und Ergebnisse. Im Abschnitt *Konzeption* wird zunächst beschrieben, wie das System aus dem letzten Kapitel erweitert wurde, um neben der binären Warnung auch die Zeit zum Ausfall vorherzusagen. Der Abschnitt *Durchführung* beschreibt anschließend, wie diese Erweiterungen konkret in Java implementiert wurden und wie die Vorgehensweise der Evaluierung strukturiert war. Der letzte Abschnitt präsentiert letztlich die *Ergebnisse* der Evaluierung und beantwortet so, ob es durch die Erweiterungen möglich ist, die Zeit zum Stromrichterausfall vorherzusagen.

5.1 Konzeption

Im Folgenden wird erläutert, welche Anpassungen des im letzten Kapitel entwickelten System vorgenommen wurden. Die Grundidee war, die Vorhersage der Zeit in die Instanz-Ebene zu integrieren. Auf Tour-Ebene sollten die Vorhersagen anschließend aggregiert werden, um den Lokführer zu benachrichtigen, ob ein Stromrichterausfall droht und wie lange es bis dahin noch dauert. Zur Umsetzung dieser Idee wurden die blau dargestellten Ebenen aus Abbildung 28 (S. 70) angepasst bzw. erweitert, weshalb sich der Abschnitt in die Teile Labeling, Instanz-Ebenen Klassifizierung und Tour-Ebenen Klassifizierung unterteilt.

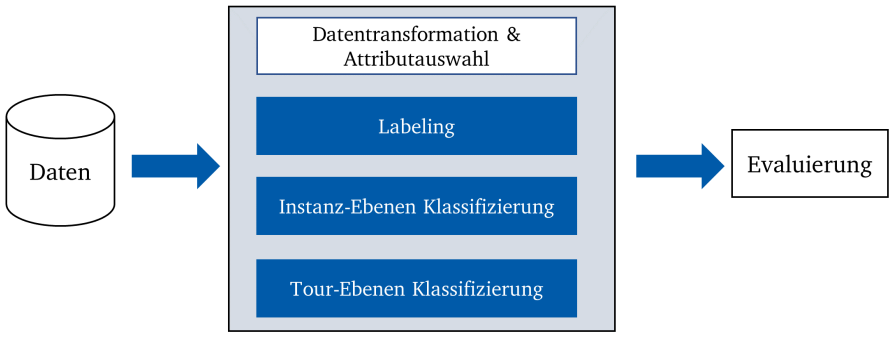

Abbildung 28: Vorhersagesystem mit Anpassungen (Versuchsreihe 2)

5.1.1 Labeling

Für die zusätzliche Vorhersage der Zeit war es zunächst notwendig, das Labeling auf Instanz-Ebene zu verändern. Anstatt also alle Instanzen in einem bestimmten Zeitfenster *z* vor dem Ausfall mit der positiven und alle anderen als negativen Klasse zu versehen, wird ein Multiklassen Ansatz in Anlehnung an Microsoft (2015) verwendet.[233] Die Grundidee ist in Abbildung 29 beispielhaft dargestellt.

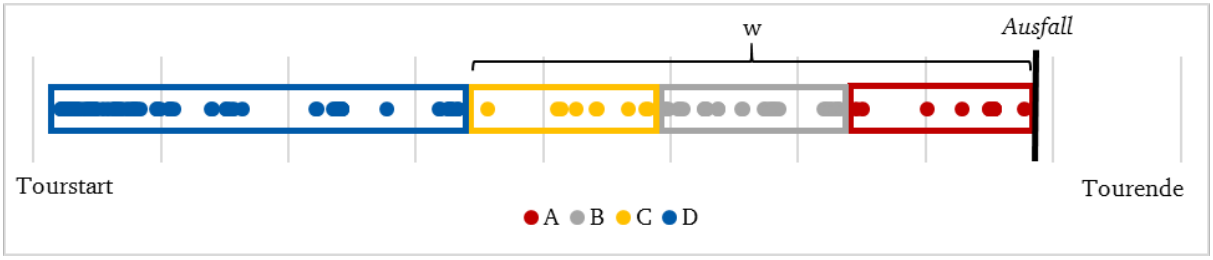

Abbildung 29: Labeling Multiklassen

In Abbildung 29 sind die Instanzen vor einem Ausfall als Punkte entlang der Zeitachse abgebildet, wobei die Klasse der Instanzen durch die Farben symbolisiert ist. Wie zu erkennen, werden die Instanzen jetzt nicht mehr nur in zwei Klassen aufgeteilt. So wird das Zeitintervall *w* nochmals in drei kleinere Zeitintervalle unterteilt, wobei diese jeweils gleich groß sind. Jede Instanz im Intervall *w* wird nun nicht mehr mit der positiven Klasse, sondern mit der Klasse des zugehörigen kleineren Zeitintervalls versehen. Im konkreten Beispiel sind dies die Klasse A, B oder C. Die Instanzen außerhalb des Intervalls *w* erhalten hingegen wie zuvor eine einheitliche Klasse, im Beispiel also die Klasse D.[234]

[233] Vgl. Microsoft (2015).

[234] In Anlehnung an Microsoft (2015).

Diese Labeling-Methode bietet den Vorteil, dass nun bei der Vorhersage differenziert werden kann, in welchem Zeitintervall die jeweilige Instanz vermutetet wird.[235] Anstatt nun also lediglich die Information zu erhalten, dass die Instanz vermutlich in einem Zeitintervall *w* vor dem Ausfall liegt, wird nun vorhergesagt, ob diese beispielsweise dem Intervall A, B oder C zugeordnet wird.

5.1.2 Instanz-Ebenen Klassifizierung

Durch Verwendung von mehr als zwei Klassen wird aus dem binären Problem auf Instanz-Ebene ein Multiklassen Problem. Da jedoch die zuvor verwendeten Instanz-Ebenen Klassifizierer von Grund auf mit Multiklassen Problemen umgehen können, war es nicht notwendig, diese anzupassen oder durch andere Algorithmen zu ersetzen. Dennoch wurde eine Anpassung auf Instanz-Ebene vorgenommen, die den SpreadSubsample Filter betrifft. Wie bereits zuvor beschrieben, reduziert dieser Filter die Mehrheitsklasse, sodass ein festes Verhältnis zwischen Mehrheits- und Minderheitsklasse entsteht. Da nun mehr als zwei Klassen existieren, würde der Filter in seiner Standardkonfiguration die Klasse mit der geringsten Anzahl an Instanzen als Minderheitsklasse und die Instanz mit der höchsten Anzahl an Instanzen als Mehrheitsklasse annehmen.[236] Die Idee war nun, den Filter so anzupassen, dass er kein festes Verhältnis zwischen Minderheits- und Mehrheitsklasse erzeugt, sondern ein Verhältnis zwischen der Klasse außerhalb von *w* und allen Klassen innerhalb von *w* (im Beispiel von Abbildung 29 (S. 70) also ein festes Verhältnis zwischen der Klasse D und der Klasse A, B und C).

5.1.3 Tour-Ebenen Klassifizierung

Da eine Instanz nun nicht mehr nur als positiv oder negativ klassifiziert wird, sondern mehr Klassen unterschieden werden, mussten die Tour-Ebenen Klassifizierer überarbeitet werden. Hierfür waren grundlegend zwei Anpassungen notwendig:

1. Zum einen musste geklärt werden, wie das Verhältnis bzw. das Aktivierungslevel berechnet wird. Dies bestimmt, wann eine Warnung über einen drohenden Stromrichterausfall herausgegeben bzw. wann eine Tour als „Fehlertour“ klassifiziert wird.

235 Vgl. Microsoft (2015).

236 Vgl. Inglis (2016).

2. Zum anderen war es notwendig zu klären, wie die Zeitvorhersage bei einem Überschreiten des Schwellenwerts (also beim Herausgeben einer Warnung) anhand der Instanz-Ebenen Vorhersagen gebildet wird.

Für die erste Anpassung wurde die Berechnung des Verhältnisses und des Aktivierungslevels neu definiert. Die Berechnung des Verhältnisses wurde dabei wie folgt geändert:

$$r = \frac{Anzahl\ Vorhersagen\ der\ Klassen\ innerhalb\ w}{Anzahl\ aller\ Vorhersagen} \tag{14}$$

Für die Berechnung des Verhältnisses r werden also alle Klassen innerhalb w als positive Klasse interpretiert. Im Beispiel aus Abbildung 29 (S. 70) würde r also wie folgt berechnet werden:

$$r = \frac{n(A) + n(B) + n(C)}{n(A) + n(B) + n(c) + n(D)} \tag{15}$$

Motiviert war diese Anpassung dadurch, dass die Herausgabe einer Warnung unabhängig von dem vorhergesagten Intervall sein soll. Eine Vorhersage von C, C, B soll also genau so zu einer Warnung führen wie C, B, B. Das primäre Ziel ist also die Herausgabe der Warnung, das sekundäre Ziel hingegen die korrekte Vorhersage der Zeit.

Nach dem gleichen Prinzip wurde auch die Berechnung des Aktivierungslevels angepasst. Anstatt nun das Aktivierungslevel um den Konfidenzwert der positiven Klasse zu erhöhen, wird das Level durch die Summe der Konfidenzwerte aller Klassen innerhalb von w erhöht. Im konkreten Beispiel würde das Aktivierungslevel a also um den Wert

$$\mathrm{P(A} \wedge \mathrm{B} \wedge \mathrm{C}) = P(A) + P(B) + P(C) \tag{16}$$

erhöht werden. Anstelle also nur die Konfidenz der vorhergesagten Klassen zu verwenden, wird das Aktivierungslevel um die Wahrscheinlichkeit erhöht, dass die Instanz in das Zeitintervall w fällt. Wie zuvor ist dies darin begründet, dass die Herausgabe einer Warnung und somit über die Klassifizierung einer Tour als „Fehlertour“ unabhängig vom vorhergesagten Zeitintervall innerhalb w stattfinden soll.

Um nun bei einer Klassifizierung einer Tour als „Fehlertour“ auch die Zeit zum Fehler bereitstellen zu können, musste im Rahmen der zweiten Anpassung bestimmt werden, wie die Zeitvorhersagen der Instanz-Ebene aggregiert werden. Dabei wurde entschieden, das am nächsten am Ausfall liegende Intervall zu verwenden, welches die Instanz-Ebene bisher vorhergesagt hat. Um dies zu verdeutlichen, sei der Vorhersageverlauf auf Instanz-Ebene aus Abbildung 30 (S. 73) gegeben. Der erste Vorhersageverlauf stellt dabei eine optimale Klassifizierung auf Instanz-Ebene dar, da Intervalle, die zeitlich weiter entfernt vom Ausfall sind, zuerst vorhergesagt

werden. Im zweiten Vorhersageverlauf wird jedoch eine Instanz dem Intervall B zugeordnet, bevor Instanzen wieder dem Intervall C zugeordnet werden. In diesem Fall wird also statt des Zeitintervalls C das Zeitintervall B vorhergesagt, sodass die Zeitvorhersage pessimistisch gestaltet ist. Die pessimistische Vorhersage wurde gewählt, da der Lokführer seine Strategie abhängig von der vorhergesagten Zeit wählt. Wird die Zeit überschätzt so ist es unter Umständen nicht möglich die Strategie bis zum Ende durchzuführen.

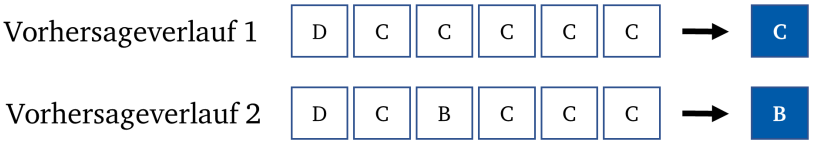

Abbildung 30: Vorhersage der Zeit

Zusammengefasst wird die Tour also unabhängig von dem vorhergesagten Zeitintervall als Fehler- oder Normaltour klassifiziert. Für die Zeitvorhersage bei Herausgabe einer Warnung wird hingegen die pessimistischste Vorhersage auf Instanz-Ebene verwendet.

5.2 Durchführung

Um nun die zweite Forschungsfrage zu beantworten, galt es zu überprüfen, ob und wie gut diese Konzepte funktionieren. Hierzu wurden die Erweiterungen zunächst implementiert und in das bestehende Java-Programm integriert. Mit Hilfe dieser Implementierung wurde anschließend die Evaluierung auf dem Lichtenberg-Hochleistungscluster der TU Darmstadt durchgeführt. Der folgende Abschnitt erläutert diese beiden Schritte und unterteilt sich deshalb in zwei Teile. Im ersten Teil wird erläutert, welche Klassen angepasst und welche neu erstellt wurden. Der zweite Teil beschreibt anschließend den Prozess der Evaluierung. Es wird also präsentiert, wie die Trainings- und Testdaten ausgewählt, welche Parameterkonstellationen ausprobiert und wie die daraus resultierenden Versuche auf dem Cluster ausgeführt wurden.

5.2.1 Implementierung

Für die Implementierung der neuen Konzepte wurde auf dem bestehenden System aus Versuchsreihe 1 aufgebaut, welches sich grundlegend in die PHP-Skripte und die Java-Klassen aufteilt. Da an der Datentransformation und Attributauswahl keine Änderungen vorgenommen werden musste, konnten die Touren und Instanzen aus der Datenbank unverändert

weitergenutzt werden. Somit war es nicht notwendig, neue PHP-Skripte zu erstellen oder die bestehenden anzupassen. Für die Implementierung der neuen Konzepte musste lediglich das Java-Projekt erweitert werden.

Die grundlegende Architektur der Java-Applikation ist in Abbildung 31 abgebildet. Dabei sind die Erweiterungen, die im Rahmen von Versuchsreihe 2 vorgenommen wurden, durch Unterstreichungen kenntlich gemacht. Diese sollen im Folgenden kurz erläutert werden.

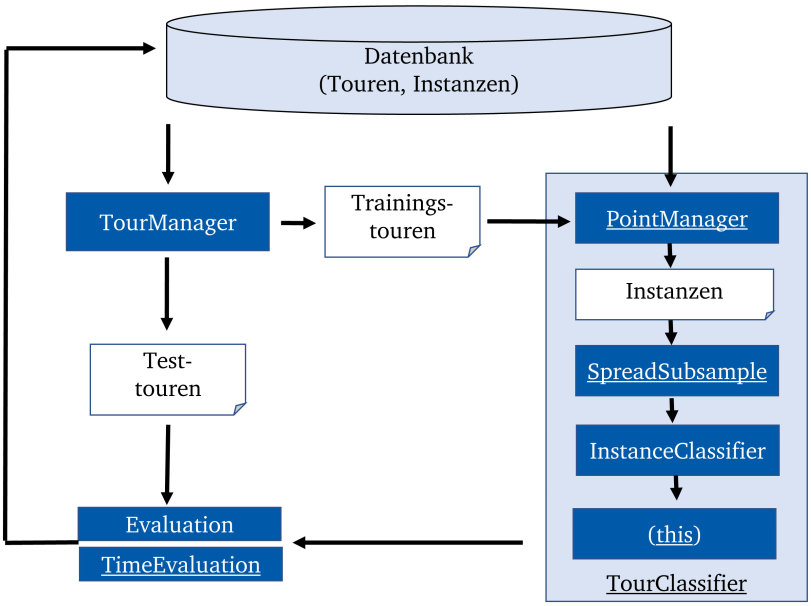

Abbildung 31: Anpassungen Java

Für das Labeling der Instanzen wurden zwei neue *PointManager* erstellt: der *MultiPointManager* (für den lernenden Verhältnisklassifizierer) und der *MultiPointTimeManager* (für den Aktivierungsklassifizierer). Diese besitzen wie auch im binären Fall die Aufgabe, die Instanzen herunterzuladen und mit den jeweiligen Label zu versehen. Im Unterschied zu den bisherigen setzen diese *PointManager* jedoch beim Labeling den Multiklassen-Ansatz um. Für die Implementierung des in Abschnitt 5.1.2 beschriebenen Verfahren zur Anpassung der Klassenverteilung wurde hingegen der *MultiSpreadSubsample* Filter entwickelt. Dieser basiert im Wesentlichen auf dem ursprünglichen Weka-internen *SpreadSubsample* Filter, erweitert diesen jedoch an einigen Stellen.

Die Änderungen auf Tour-Ebenen (siehe Abschnitt 5.1.3) wurden in den Klassen *MultiActivationTourClassifier* und *MultiRatioLearnTourClassifier* umgesetzt. Diese geben bei der Klassifizierung nicht nur zurück, ob die Tour als Fehler- oder Normaltour klassifiziert wird, sondern im Falle einer Fehlertour auch das vorhergesagte Zeitintervall. Um letztlich auch die

Zeitvorhersagen bewerten zu können, wurde die *MultiTimeEvaluation* Klasse entwickelt. Diese speichert die Multiklassen-Konfusionsmatrix und berechnet auf Basis derer die Accuracy, die Makro F1-Measure und den Micro F1-Measure der Zeitvorhersagen.

5.2.2 Vorgehensweise

Nachdem die neuen Ansätze implementiert waren, wurden diese evaluiert. Ziel war es, zu überprüfen, ob zum einen die Touren korrekt klassifiziert wurden, und zum anderen, ob die Vorhersage des Zeitintervalls korrekt war. Zu beachten ist hierbei, dass Zeitvorhersagen immer dann herausgegeben werden, wenn eine Tour als Fehlertour klassifiziert wurde. Es werden also auch Zeitvorhersagen ausgegeben, wenn eine Normaltour fälschlicherweise als Fehlertour klassifiziert wird. Da diese Zeitvorhersage in diesem Fall nur falsch sein kann, wurde sich dazu entschlossen nur Zeitvorhersagen in die Evaluierung einzubeziehen, die entstehen, wenn eine Fehlertour korrekterweise auch als Fehlertour eingeordnet wird.

Die restliche Vorgehensweise folgte weitgehend der Architektur aus der ersten Versuchsreihe (siehe Abschnitt 4.2.2). So wurden die gleichen 450 Touren (bestehend aus 32 Fehler- und 418 Normaltouren) als Trainingsmenge verwendet. Auf diesen wurde eine Kreuzvalidierung durchgeführt, um so die optimale Parameterkonfiguration zu ermitteln. Diese wurde anschließend auf die Testmenge angewandt, welche die gleichen 8 Fehler- und 18.262 Normaltouren umfasste.

Für die Konfiguration wurden die in Abbildung 32 abgebildeten Parameter bzw. Parameterwerte verwendet. Dabei sind die Parameter, die neu hinzugefügt oder deren Werte angepasst wurden, in dunkelblau dargestellt.

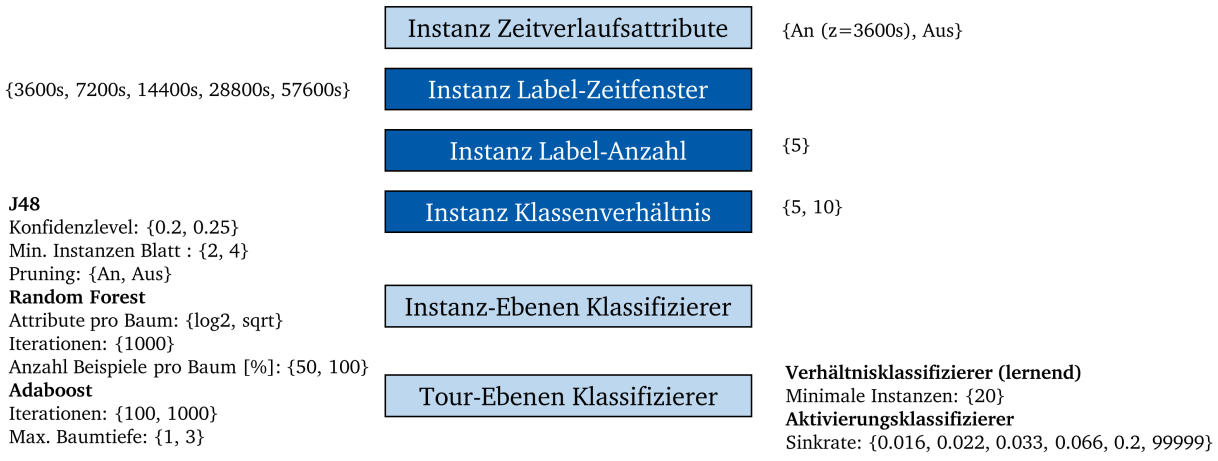

Abbildung 32: Konfigurationen Versuchsreihe 2

Wie zu erkennen ist, sind für das Labeling-Zeitfenster w im Vergleich zur Versuchsreihe 1 größere Werte verwendet worden. Dies ist darin begründet, dass nicht nur eine Erkennung des Musters vor dem Fehler stattfinden soll, sondern auch eine Zeitvorhersage. Wird das Zeitfenster w also relativ klein gewählt, so umfassen die resultierenden Zeitintervalle nur wenige Minuten. Aus diesem Grund wurden als Zeitfenster w die Werte 1h, 2h, 4h, 8h und 16 h verwendet. Dabei wird dieses Zeitintervall in jeweils vier kleinere Zeitintervalle aufgeteilt, was durch die Label-Anzahl = 5 definiert wird. Bei einem Zeitfenster von w = 1h entstehen also vier 15-Minuten Intervalle vor dem Ausfall. Bei einem Zeitfenster von w = 16h hingegen vier 4-Stunden Intervalle. Letztlich wurden auch die Parameterwerte für das Klassenverhältnis angepasst. Dabei wurden kleine Werte verwendet, da durch die teils großen Zeitfenster w sonst eine große Anzahl an Instanzen entstehen würde, was potentiell zu Speicher- und Zeitproblem bei der Berechnung führen kann.

Für jede dieser Konfiguration wurde ein Slurm-Skript erstellt. Insgesamt wurden so 1820 Konfigurationen auf dem Lichtenberg-Hochleistungscluster der TU Darmstadt ausgeführt. Dies erforderte letztlich ca. 4100 Stunden Rechenzeit, wobei auch hier wieder mehrere hunderte Skripte parallel ausgeführt werden konnten.

5.3 Ergebnisse

Im folgenden Abschnitt sollen nun die Ergebnisse der Evaluierung präsentiert und diskutiert werden. Hierzu unterteilt sich das Kapitel, wie in der vorherigen Versuchsreihe, in zwei Abschnitte. Im ersten Abschnitt werden die Ergebnisse der Kreuzvalidierung erläutert. Im zweiten Abschnitt wird letztlich beschrieben, wie gut die beste Konfiguration der Kreuzvalidierung auf der Testmenge abschneiden konnte.

5.3.1 Kreuzvalidierung

Wie in Kapitel 5.2.2 beschrieben, wurden alle 1820 Konfigurationen in Form von Slurm-Skripten auf dem Cluster ausgeführt. Dabei konnten 1819 Konfigurationen erfolgreich berechnet werden, während eine Konfiguration aufgrund von Zeitrestriktionen abgebrochen wurde. Dies ist darin begründet, dass diese Konfiguration das 24 Stunden Zeitlimit, welches auf dem Cluster eingehalten werden muss, erreicht hatte.

Um die Konfigurationen vergleichen bzw. sortieren zu können, war es notwendig, ein Maß zu finden, welche sowohl die Klassifikationsleistung in Bezug auf die Warnung bzw. die Touren,

als auch die Klassifikationsleistung in Bezug auf die Zeitvorhersage berücksichtigt. Dabei sollte die Erkennung der Touren ein höheres Gewicht erhalten als die korrekte Vorhersage der Zeit. Letztlich wurde folgende Zielfunktion Z definiert, wobei k die jeweilige Konfiguration darstellt:

$$Z(k) = 0{,}75 * \frac{F1_{Tour}(k)}{\max_h(F1_{Tour}(h))} + 0{,}25 * \frac{Accuracy_{Zeit}(k)}{\max_h(Accuracy_{Zeit}(h))} \quad (17)$$

Wie zu erkennen, wird zur Bewertung der Klassifikationsleistung auf Tour-Ebene die F1-Measure aufgrund der Ungleichverteilung der Klassen verwendet. Bei Bewertung der Zeitvorhersage wird hingegen die Accuracy eingesetzt. Beide Maße werden normalisiert, sodass beide Werte zwischen 0 und 1 annehmen. Die Gewichtung ist hingegen so gewählt, dass die Klassifikationsleistung in Bezug auf die Touren ein drei Mal höheres Gewicht einnimmt, wie die der Zeitvorhersage. Dabei steigt $Z(k)$, wenn bei der Tour-Vorhersage die Erkennungsrate steigt bzw. die Fehleralarmrate sinkt, oder bei der Zeit-Vorhersage der Anteil der korrekt klassifizierten Zeitintervalle steigt.

Mit Hilfe dieses Maßes können die Konfigurationen nun verglichen und sortiert werden. Die Ergebnisse der Kreuzvalidierung sind dabei in Tabelle 6 (S. 78) abgebildet, wobei sich auf die besten 20 Konfigurationen beschränkt wurde. Dabei stellt jede Reihe eine Konfiguration dar. Für jede Konfiguration sind deren Parameter und Evaluierungsmaße abgebildet. Dabei sind die Parameter nochmals in Instanz-Ebene und Tour-Ebene aufgeteilt. Die Evaluierungsmaße hingegen unterteilten sich in Tour- und Zeit-Maße. Die Tour-Maße messen dabei die Klassifikationsleistung bei der Tour-Vorhersage. Die Zeit-Maße hingegen messen die Leistung bei der Zeitvorhersage. Da es sich bei der Zeitvorhersage um ein Multiklassen-Problem handelt, wurde hierbei auf die Micro F1-Measure, Makro F1-Measure und die Accuracy Maße aus Kapitel 2.4.2 zurückgegriffen.

Tabelle 6: Ergebnisse Versuchsreihe 2 sortiert nach Zielfunktionswert (Kreuzvalidierung)

	Instanz-Ebene							Tour-Ebene		Tour Maße								Zeit Maße		
Rang	Zeitverlaufs-attribut	Label-Zeitfenster	Label-Anzahl	Klassen-Verhältnis	Klassifizierer	J48 Pruning RF Anzahl/Baum Adaboost Iteration	J48 Konf.-Level RF Iterationen Adaboost Baumtiefe	J48 Anzahl Blatt RF Anzahl Attr.	Klassifizierer	Akt. Sinkrate	TP	TN	FP	FN	Accuracy (%)	Precision	Recall	F1	Mikro F1	Makro F1	Accuracy (%)	Zielfunktion Z
1	Aus	57600	5	10	J48	An	0,25	2	Aktivierung	0,0667	12	380	38	20	87,111	0,24	0,375	0,293	0,333	0,166	33,3	0,778
2	Aus	57600	5	10	J48	An	0,5	4	Aktivierung	0,0667	14	374	44	18	86,222	0,241	0,438	0,311	0,143	0,067	14,3	0,775
3	Aus	57600	5	5	J48	An	0,5	4	Aktivierung	0,0667	10	392	26	22	89,333	0,278	0,312	0,294	0,3	0,12	30	0,774
4	Aus	57600	5	10	J48	An	0,5	2	Aktivierung	0,0167	13	378	40	19	86,889	0,245	0,406	0,306	0,154	0,067	15,4	0,765
5	Aus	14400	5	10	J48	An	0,25	4	Aktivierung	0,2	9	390	28	23	88,667	0,243	0,281	0,261	0,556	0,463	55,6	0,758
6	Aus	57600	5	10	J48	An	0,5	2	Aktivierung	0,0222	13	377	41	19	86,667	0,241	0,406	0,302	0,154	0,067	15,4	0,756
7	Aus	7200	5	10	J48	Aus			Verhältnis (L)		9	402	16	23	91,333	0,36	0,281	0,316	0	0	0	0,750
8	Aus	57600	5	10	J48	An	0,5	2	Aktivierung	0,0333	13	376	42	19	86,444	0,236	0,406	0,299	0,154	0,067	15,4	0,748
9	An	57600	5	5	J48	An	0,5	4	Aktivierung	0,0667	15	355	63	17	82,222	0,192	0,469	0,273	0,4	0,276	40	0,748
10	Aus	57600	5	10	J48	An	0,25	2	Aktivierung	0,0222	11	384	34	21	87,778	0,244	0,344	0,286	0,273	0,109	27,3	0,747
11	An	57600	5	5	J48	An	0,5	4	Verhältnis (L)		9	388	30	23	88,222	0,231	0,281	0,254	0,556	0,367	55,6	0,741
12	An	57600	5	5	J48	An	0,5	4	Aktivierung	0,0333	15	353	65	17	81,778	0,188	0,469	0,268	0,4	0,276	40	0,736
12	An	57600	5	5	J48	An	0,5	4	Aktivierung	0,2	15	353	65	17	81,778	0,188	0,469	0,268	0,4	0,333	40	0,736
14	Aus	57600	5	10	J48	An	0,5	2	Aktivierung	0,0667	13	374	44	19	86	0,228	0,406	0,292	0,154	0,067	15,4	0,732
14	Aus	57600	5	10	J48	An	0,5	4	Aktivierung	0,0333	13	374	44	19	86	0,228	0,406	0,292	0,154	0,073	15,4	0,732
16	Aus	57600	5	5	J48	An	0,25	4	Aktivierung	0,0667	9	393	25	23	89,333	0,265	0,281	0,273	0,333	0,12	33,3	0,731
17	An	7200	5	10	J48	An	0,5	2	Aktivierung	0,0333	11	389	29	21	88,889	0,275	0,344	0,306	0	0	0	0,726
18	Aus	57600	5	10	J48	An	0,25	2	Aktivierung	0,0167	11	384	34	21	87,778	0,244	0,344	0,286	0,182	0,089	18,2	0,724
19	Aus	57600	5	10	J48	An	0,5	4	Aktivierung	0,2	12	378	40	20	86,667	0,231	0,375	0,286	0,167	0,073	16,7	0,720
20	An	57600	5	5	J48	An	0,5	4	Aktivierung	0,0222	15	350	68	17	81,111	0,181	0,469	0,261	0,4	0,276	40	0,720

RF = Random Forest,
Verhältnis (L) = Lernender Verhältnisklassifizierer

Betrachtet man Tabelle 6 (S. 78), so fällt auf, dass alle Top-20 Konfigurationen den J48 Klassifizierer auf Instanz-Ebene verwenden. Auch nutzen die meisten Konfigurationen den Aktivierungsklassifizierer auf Tour-Ebene und ein Labeling-Zeitfenster von 16 Stunden (57.600). Die Häufigkeit des größten Zeitfensters könnte dabei unterschiedliche Gründe haben. Zum einen sind so die Zeitintervalle relativ groß (4-Stunden Intervalle), zum anderen könnte dies darin begründet sein, dass sich in diesem Zeitraum Indizien für den Ausfall des Stromrichters finden lassen. Bezüglich der Evaluierungsmaße fällt auf, dass der F1-Wert der Tour-Vorhersage auf einem ähnlichen Niveau wie auch bei Versuchsreihe 1 liegt. Bei der Zeitvorhersage wird hingegen bei den Top 20 Konfigurationen maximal ein Wert von 55,6% erreicht. Bezogen auf alle Konfigurationen, die mehr als 2 True Positives besitzen, existiert dabei maximal eine Accuracy von 66,7%.[237]

Die beste Konfiguration erreicht einen Zielfunktionswert von $Z = 0{,}778$. Dabei wurden 12 der 32 Fehlertouren erkannt, sodass die Erkennungsrate bei 37,5% liegt. Die Fehlalarmrate hingegen liegt bei 9,1%, da 38 der 418 Normaltouren als Fehlertouren klassifiziert wurden. Somit erreicht die Konfigurationen einen F1-Wert von 0,293. Die Genauigkeit der Zeitvorhersage liegt hingegen bei 33,33%. Die Konfusionsmatrix der Zeitvorhersage ist dabei in Tabelle 7 abgebildet, wobei das reale Intervall in den Reihen und die Vorhersage in den Spalten steht.

Tabelle 7: Konfusionsmatrix Zeitvorhersage

Realität/ Vorhersage	I(>16h)	I(12h-16h)	I(8h-12h)	I(4h-8h)	I(0h-4h)
I(>16h)	0	0	0	0	1
I(12h-16h)	0	0	0	1	1
I(8h-12h)	0	0	0	1	1
I(4h-8h)	0	0	1	**1**	0
I(0h-4h)	0	0	0	2	**3**

Wie zu erkennen ist, sind vier der vorhergesagten Zeitintervalle korrekt vorhergesagt worden. Bei den fehlerhaften Vorhersagen hingegen gibt es keine eindeutige Tendenz. So wird teilweise ein zu frühes, teilweise aber auch ein zu spätes Zeitintervall prognostiziert.

Zusammengefasst liegt der F1-Wert bei der Tour-Vorhersage auf einem ähnlichen Niveau wie auch bei Versuchsreihe 1. Die Zeitvorhersage erreicht jedoch nur eine geringe Genauigkeit. So liegt die Accuracy bei der besten Konfiguration (bezogen auf Z) bei nur 33,33%. Mit anderen Worten erhält der Lokführer in 66,66% der Fälle eine falsche Information über die Dauer bis zum Ausfall. Selbst bei der besten Accuracy (bei Konfigurationen mit mehr als 2 True Positives)

[237] Das Ergebnis aller Konfigurationen ist auf dem beiliegenden Datenträger mitgeliefert.

wird in 33,33% der Fälle das falsche Zeitintervall vorhergesagt. Zusätzlich muss noch angemerkt werden, dass durch die Auswahl des besten Modells die Klassifikationsleistung tendenziell überschätzt wird, da ja gerade die Parameter ausgewählt werden, die auf der Menge an Beispielen am besten funktionieren.[238] Die praktische Einsatzfähigkeit der Zeitvorhersage ist also bereits bei der Kreuzvalidierung bzw. beim Parametertuning zu vereinen. Um abschließend eine unverzerrte Messung der Klassifikationsleistung auf Tour-Ebene zu erhalten wurde die beste Konfiguration auf die Testmenge angewandt.

5.3.2 Testmenge

Um abschließend eine unverzerrte Messung der Klassifikationsleistung auf Tour-Ebene zu erhalten, wurde die beste Konfiguration auf die Testmenge angewandt. Hierzu wurden alle 450 Trainingstouren, bestehend aus 32 Fehler- und 418 Normaltouren, zur Erstellung des Klassifikationsmodells verwendet. Das gelernte Modell wurde anschließend auf 18.262 Touren angewandt, wobei hiervon 8 Touren Fehlertouren darstellen. Dies entspricht dem realen Verhältnis zwischen Fehler- und Normaltouren.

Das Ergebnis der Evaluierung ist in Tabelle 8 dargestellt:

Tabelle 8: Ergebnisse Versuchsreihe 2 (Testmenge)

Tour Maße								Zeit Maße			
TP	TN	FP	FN	Accuracy %	Precision	Recall	F1	Mikro F1	Makro F1	Accuracy %	Zielfkt.
0	16319	1943	8	89,321	0	0	0	-	-	-	-

Wie zu erkennen ist, wurde keiner der acht Fehlertouren erkannt, wohingegen 1943 der Normaltouren fälschlicherweise als Fehlertouren klassifiziert wurden. Somit liegt die Erkennungsrate bei 0%. Mögliche Erklärungen für dieses Ergebnis sind bereits in Versuchsreihe 1 erläutert. So könnte es zum einen an der geringen Fehlertour-Anzahl liegen, zum anderen jedoch darin begründet sein, dass die Kreuzvalidierung die Genauigkeit überschätzt und die Erkennungsrate in der Realität gering ist. Dies kann jedoch letztlich nur anhand einer größeren Stichprobe festgestellt werden. Die Fehleralarmrate liegt hingegen bei 10,64%. Da hierbei eine große Stichprobe von 18.262 Touren verwendet wird, ist dieser Wert eine gute Abschätzung der Fehlalarmquote. Dabei ist die Fehleralarmrate verglichen mit der besten F1 Konfiguration aus Versuchsreihe 1 auf der Testmenge (Fehlalarmrate: 4,62%) jedoch ca. 2,3-fach höher.

[238] Vgl. Varma;Simon (2006), S. 91:1 ff.

Verglichen mit der besten Accuracy Konfiguration auf der Testmenge (Fehlalarmrate 0,79%) sogar ca. 13,5-fach höher.

Bezogen auf die Zeitvorhersage existiert kein Accuracy-Wert. Grund hierfür ist, dass keine Warnung bei den Fehlertouren ausgegeben wird. Es wird also kein Zeitintervall bei den Fehlertouren vorhergesagt, weshalb auch keine Accuracy berechnet werden konnte.

Zusammengefasst kann gesagt werden, dass die Testmenge wie auch bei Versuchsreihe 1 nur schwer eine Abschätzung der Erkennungsrate auf Tour-Ebene liefern kann. Dennoch konnte festgestellt werden, dass die Fehleralarmrate um das 2,3-fache bzw. 13,7-fache höher ausfällt als beim binären Labeling von Versuchsreihe 1. Letztlich bleibt durch die Ergebnisse der Kreuzvalidierung festzuhalten, dass die Vorhersage der Zeitintervalle eine geringe Genauigkeit besitzt, sodass der praktische Einsatz des in diesem Kapitel entwickelten Systems zu verneinen ist. Letztlich liefert Versuchsreihe 2 somit keine Vorteile gegenüber Versuchsreihe 1, sodass Versuchsreihe 1 vorzuziehen ist.

6 Wirtschaftliche Analyse

Nachdem nun in Kapitel 4 und 5 die ersten beiden Forschungsfragen beantwortet wurden, folgt im folgendem Kapitel die Betrachtung der dritten Forschungsfrage. Diese untersucht, wie das entwickelte Predictive Maintenance System aus wirtschaftlicher Sicht zu bewerten ist. Um dies zu beantworten, unterteilt sich das Kapitel grundlegend in zwei Teile. Den ersten Teil, welches den Fokus der wirtschaftlichen Analyse darstellt, bildet die finanzielle Analyse bzw. die finanzielle Bewertung des Systems. Es soll also überprüft werden, ob es durch den Einsatz des Systems möglich ist, die Kosten zu reduzieren und somit einen monetären Vorteil zu erzielen. Im zweiten Teil folgt anschließend eine strategische Analyse bzw. Bewertung des Predictive Maintenance System. In diesem Teil wird folglich untersucht, ob neben finanziellen Gründen auch andere Gründe für die Einführung des Systems sprechen. Dabei wird sich aufgrund der Ergebnisse der beiden vorherigen Kapitel im Folgenden rein auf das binäre Vorhersagesystem ohne Zeitvorhersage (Versuchsreihe 1) beschränkt.

6.1 Finanzielle Analyse

Im folgenden Abschnitt wird untersucht, ob sich eine Investition in das System aus finanzieller Sicht lohnt bzw. ob die Leistung des Systems ausreicht, um eine Kostenreduktion zu erzeugen. Zur Beantwortung dieser Fragestellung wurden im Rahmen der finanziellen Analyse im Wesentlichen die folgenden drei Schritte durchgeführt:

1. Literaturrecherche nach bestehenden Bewertungsansätzen und Kostenmodellen
2. Adaption der Kostenmodelle auf die konkrete Problemstellung
3. Anwendung des adaptierten Kostenmodells auf das System (mit Hilfe beispielhafter Kostenwerte)

In Anlehnung an diese Aufteilung unterteilt sich der Abschnitt in drei Unterabschnitte, welche die Ergebnisse der jeweiligen Schritte darstellen. Dabei wird im ersten Unterabschnitt zunächst auf das Konzept der Lebenszykluskosten eingegangen. Anschließend folgt eine Beschreibung von Kostenmodellen zur Berechnung der Lebenszykluskosten, welche im Umfeld von Predictive Maintenance bzw. im Umfeld von Zügen entwickelt wurden. Im zweiten Unterabschnitt wird das im Rahmen der Arbeit entwickelte Kostenmodell beschrieben, welches die zuvor beschriebenen Kostenmodelle auf die konkrete Problemstellung adaptiert. Nachdem das spezifische Kostenmodell präsentiert wurde, folgt im dritten Unterabschnitt die Anwendung des Kosten-

modells. Da aus Aufwands- und Datenschutzgründen keine konkreten Kostendaten von Seiten der DB Cargo bereitgestellt werden konnten, erfolgt die Anwendung mit Hilfe beispielhafter Schätzwerte. Ziel des Kapitels ist folglich nicht eine finale finanzielle Bewertung, sondern die Entwicklung eines Instrumentes, welches von der DB Cargo leicht zur finanziellen Bewertung des Systems genutzt werden kann.

6.1.1 Literaturrecherche: Lebenszykluskosten und Kostenmodelle

Im Folgenden wird nun, wie bereits beschrieben, auf das Konzept der Lebenszykluskosten und auf die bereits entwickelten Kostenmodelle im Umfeld von Predictive Maintenance bzw. Zügen eingegangen.

6.1.1.1 Lebenszykluskosten

Begonnen werden soll dabei mit der Beschreibung des Konzepts der Lebenszykluskosten. Hierzu unterteil sich der Abschnitt in drei Abschnitte: Definition und Ziele, Prozess zur Ermittlung der Lebenszykluskosten und Kostenaufbruchsstruktur/Kapitelwertmethode.

6.1.1.1.1 Definition und Ziele

Für die Beschreibung des Konzepts der Lebenszykluskosten soll auf die Norm DIN EN 60300-3-3:2005-03 und den Norm-Entwurf DIN EN 60300-3-3:2014-09 zurückgegriffen werden.[239] Hier wird der Begriff Lebenszykluskosten definiert als „kumulierte Kosten eines Produkts über seinen Lebenszyklus“[240]. Dabei umfasst der Lebenszyklus das „Zeitintervall zwischen der Konzipierung und der Aussonderung eines Produkts“[241]. Das Produkt (bzw. die Einheit) kann dabei beispielsweise ein Bauelement, ein Gerät oder ein System, bestehend aus Hardware, Software und Personen, sein. Wichtig ist zu beachten, dass beim Konzept der Lebenszykluskosten nur die Ausgaben einer Einheit betrachtet werden, d.h. Einnahmen oder Wertsteigerungen, die im Laufe des Lebenszyklus anfallen, fließen nicht mit ein.[242]

239 Vgl. DIN Deutsches Institut für Normung e. V. (2005), S. 1 ff., DIN Deutsches Institut für Normung e. V. (2014), S. 1 ff.

240 DIN Deutsches Institut für Normung e. V. (2005), S. 6.

241 ebd.

242 Vgl. DIN Deutsches Institut für Normung e. V. (2014), S. 5 ff.

Die Ermittlung der Lebenszykluskosten ist letztlich ein Bewertungsprozess, in der die Kosten während des Lebenszyklus abgeschätzt und aufsummiert werden. Ziel der Ermittlung der Lebenszykluskosten ist die Bereitstellung eines Entscheidungsinstruments. So können die verschiedenen Optionen für den Entwurf oder Betrieb einer Einheit mit den jeweilig resultierenden Lebenszykluskosten versehen werden. Dies ermöglicht es, dem Entscheider die beste (in Form der kostengünstigsten) Option auszuwählen. Sollen die Alternativen nur relativ miteinander verglichen werden und ist keine detaillierte, absolute Betrachtung der Kosten notwendig, so müssen nur die Kostenarten betrachten werden, die für einen Vergleich notwendig sind. Letztlich müssen so weniger Kostendaten ermittelt bzw. gesammelt werden.[243]

6.1.1.1.2 Prozess zur Ermittlung der Lebenszykluskosten

Zur Ermittlung der Lebenszyklus existieren mehrere Prozessmodelle.[244] Das Modell nach DIN EN 60300-3-3:2014-09 unterteilt den Prozess (neben der Herstellung des organisatorischen Kontexts) dabei grundlegend in vier Schritte:[245]

1. Planung der Analyse
2. Festlegung der Vorgehensweise bei der Analyse
3. Analyse durchführen
4. Analyse abschließen

Im *ersten Schritt* soll das Ziel der Analyse festgelegt werden. Außerdem soll ein Plan für die Analyse erstellt werden, in welchem auch die notwendigen Ressourcen und Einschränkungen enthalten sind. Zudem ist es in dieser Phase sinnvoll, relevante Finanzparameter wie beispielsweise die Inflation zu ermitteln. Im *zweiten Schritt* gilt es die Vorgehensweise festzulegen. Zu nennen sind hier vor allem zwei Teilschritte. Zum einen sollte ein Lebenszykluskosten-Modell ausgewählt bzw. entwickelt werden, zum anderen die Kostenaufbruchsstruktur festgelegt werden. Dabei ist unter Kostenaufbruchsstruktur die Zerlegung der gesamten Lebenszykluskosten in einzelne Kostenarten zu verstehen.[246]

243 Vgl. Ebd., S. 7 f.

244 Vgl. New South Wales Treasury (2004), S. 6 ff, DIN Deutsches Institut für Normung e. V. (2014), S. 10 ff.

245 Vgl. DIN Deutsches Institut für Normung e. V. (2014), S. 10 ff.

246 Vgl. Ebd., S. 13 ff.

Im *dritten Schritt* wird die Analyse nun durchgeführt. Dabei werden die Kosten geschätzt und auf Basis dieser Schätzwerte die Lebenszykluskosten ermittelt. Um die Robustheit der Lebenszykluskosten zu ermitteln, kann hierbei eine Sensitivitätsanalyse durchgeführt werden. Zuletzt sollte eine Bewertung der Analyse stattfinden und beurteilt werden, ob die Analyseziele erreicht wurden. Im *vierten Schritt* werden zuletzt noch Folgetätigkeiten benannt und die Analyse dokumentiert.[247]

6.1.1.1.3 Kostenaufbruchsstruktur und Kapitalwertmethode

Wie bereits im dritten Schritt beschrieben, ist die Ermittlung der Kostenaufbruchsstruktur eine zentrale Aufgabe. Hierbei werden die gesamten Lebenszykluskosten in einzelne Kostenelemente, auch Kostenpakete genannt, zerlegt. Ziel dieser Zerlegung ist es, dass das Unternehmen die einzelnen Kostenpakete individuell ermitteln bzw. schätzen kann. Dabei wird eine Unterteilung anhand folgenden drei Achsen vorgeschlagen: [248]

1. Kostenart
2. System-/Arbeitsaufbruchsstruktur
3. Lebenszyklusphase

Die erste Achse definiert eine Unterscheidung anhand der Kostenart. Beispielsweise können die Kosten in Arbeitskosten, Materialkosten oder Energiekosten unterschieden werden. Die zweite Achse hingegen definiert eine Abgrenzung anhand der Gliederungsebenen eines Systems. Beispielsweise kann das System Auto in seine Teilsysteme wie Motor, Getriebe, Fahrwerk usw. unterteilt werden. Zuletzt kann eine Unterscheidung anhand der Lebenszyklusphasen (dritte Achse) durchgeführt werden.[249] Beispielsweise können dabei folgende Kostenkategorien unterschieden werden:[250]

- Investitionskosten vor Betrieb (Einmalig)
- Betriebs- /Wartungskosten und Risikokosten während des Betriebs (Laufend)

Um die Kostenströme zwischen Alternativen vergleichen zu können, wird eine Anwendung von

247 Vgl. Ebd., S. 18 ff.

248 Vgl. Ebd., S. 16 f.

249 Vgl. Ebd.

250 Vgl. Hokstad (1998), S. 10.

Finanztechniken wie beispielsweise der Kapitalwertmethode empfohlen.[251] Bei der Kapitelwertmethode werden die zukünftigen Zahlungsströme mittels eines Kalkulationszinssatzes (u.a. abhängig von Marktzins) auf ein Basisjahr diskontiert.[252] Es wird also der Gegenwartswert eines Zahlungsstroms zum Basisjahr berechnet. Der Kapitalwert entspricht also dem Wert zum Zeitpunkt des Basisjahres, welcher äquivalent mit dem Wert des Zahlungsstroms über die gesamte Laufzeit ist. Die Methode stellt also eine Möglichkeit bereit, einen Zahlungsstrom auf eine einzige Kennzahl zu reduzieren.[253] Letztlich kann der Kapitelwert K_0 wie folgt berechnet werden:

$$K_o = -A_o + \sum_{t=1}^{n} \frac{CF_t}{(1+i)^t} \tag{18}$$

wobei A_0 die Investitionszahlung im Basisjahr, CF_t den Zahlungsstrom im t-ten Jahr und i den Kalkulationszinssatz darstellt.[254]

Zusammenfassend kann gesagt werden, dass das Konzept der Lebenszykluskosten ein Instrument bereitstellt, welches es ermöglicht, eine Entscheidung über eine Investition zu treffen.

6.1.1.2 Relevante Kostenmodelle

Um mit Hilfe der Lebenszykluskosten analysieren zu können, ob sich das Predictive Maintenance System aus finanzieller Sicht lohnt, wurde sich am in Abschnitt 6.1.1.1.2 definierten Prozess orientiert. Zentral war dabei die Erstellung des Kostenmodells und der Kostenaufbruchsstruktur. Hierzu wurde in der Literatur nach bereits existierenden Kostenmodellen gesucht, die im Umfeld von Predictive Maintenance und dem Zugverkehr angesiedelt sind. Auf Basis dieser wurde das Kostenmodell für die konkrete Problemstellung entwickelt. Die relevanten Forschungsarbeiten bzw. Kostenmodelle sollen nun im Folgenden vorgestellt werden.

6.1.1.2.1 Kostenmodell nach Hokstad (1998)

Das Kostenmodell von Hokstad (1998) wurde im Rahmen des REMAIN[255]-Projektes der Forschungsorganisation SINTEF[256] entwickelt. Dabei war neben SINTEF auch die Deutsche Bahn

[251] Vgl. DIN Deutsches Institut für Normung e. V. (2014), S. 27.

[252] Vgl. Heesen (2010), S. 32 f.

[253] Vgl. Poggensee (2014), S. 105 ff.

[254] Vgl. Heesen (2010), S. 32 f.

[255] REMAIN = Modular System for Reliability and Maintenance Management in European Rail Transport.

[256] SINTEF = Stiftelsen for industriell og teknisk forskning (norwegisch).

AG, aber auch ein spanisches Eisenbahnunternehmen in das Projekt involviert. Ziel des Projektes war es, ein Lebenszykluskosten-Modell für Bahnanlagen zu entwickeln, welches eine wirtschaftliche Evaluation von verschiedenen Optionen erlaubt und somit eine Entscheidungsfindung möglich macht. Fokus bei der Entwicklung lag auf der Bahninfrastruktur wie Schienen oder Weichen, wohingegen die Anwendung bzw. die Modifikation des Modells auf Züge in dieser Arbeit nicht weiter betrachtet wurde.[257]

Das Kostenmodell von Hokstad (1998) folgt der Unterteilung der Kosten anhand der drei Achsen Kostenart, System-/Arbeitsaufbruchsstruktur und Lebenszyklusphase, welche in Abschnitt 6.1.1.1.3 erläutert wurden. Dabei ist die System-/Arbeitsaufbruchsstruktur auch bei diesem Modell individuell an die betrachtete Problemstellung anzupassen. Aus diesem Grund soll der Fokus auf den Kostenarten liegen, die gleichzeitig eine Einteilung in Lebenszyklusphasen vornehmen und somit den Kern des Modells bilden.[258]

Die Aufteilung der Kostenarten ist in Abbildung 33 dargestellt:

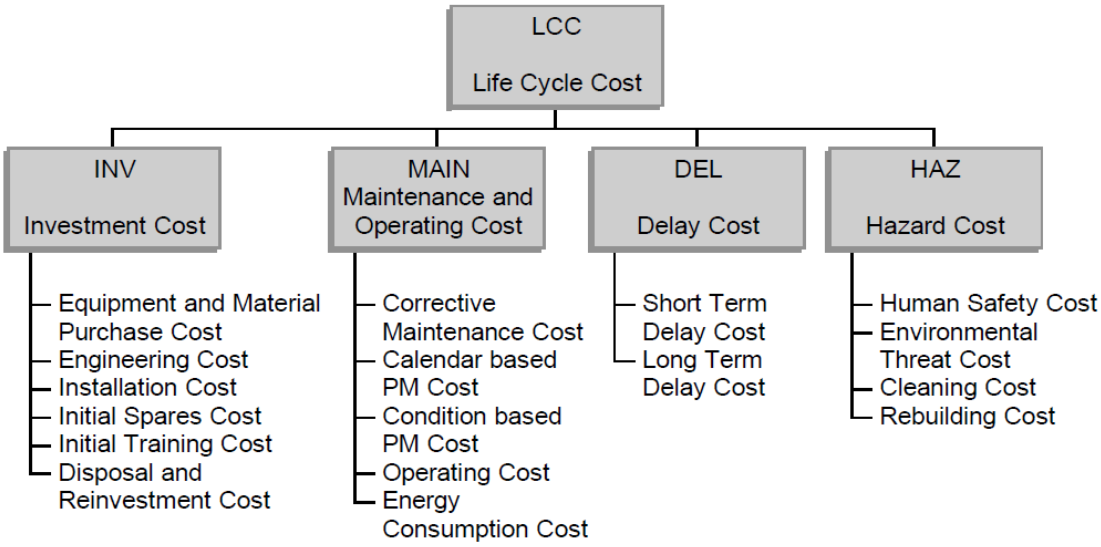

Abbildung 33: Kostenaufbruchsstruktur nach Kostenart[259]

Wie zu erkennen ist, unterteilen sich die Lebenszykluskosten LCC in vier Kostengruppen: Investitionskosten INV, Wartungs- und Betriebskosten $MAIN$, Verspätungskosten DEL und Gefahrenkosten HAZ. Es gilt somit:

$$LCC = INV + MAIN + DEL + HAZ \tag{19}$$

[257] Vgl. Hokstad (1998), S. 1 ff.

[258] Vgl. Ebd., S. 9 ff.

[259] ebd., S. 11.

Dabei stellen die Investitionskosten (INV) die einmaligen Kosten dar, während die restlichen Gruppen ($MAIN + DEL + HAZ$) Teil der laufenden Kosten sind. Jede der Kostengruppen unterteilt sich nochmals in detaillierte Kostenarten, welche im Folgenden genauer betrachtet werden sollen.[260]

Die *Investitionskosten* (INV) setzten sich aus sechs verschiedenen Kostenpaketen zusammen. Zum einen werden die Kosten für Teile und Materialien (EPC) betrachtet, welche im Rahmen der Akquisition des technischen Systems anfallen. Daneben werden aber auch Kosten für die Entwicklung (ENC), für die Installation (INC) und Kosten für die Ersatzteilhaltung (ISC) berücksichtigt. Auch werden Kosten für das Training der Mitarbeiter (ITC) und Entsorgungs- und Reinvestitionskosten zu den Investitionskosten (DIC) gerechnet. Es gilt also folglich:[261]

$$INV = EPC + ENC + INC + ISC + ITC + DIC \tag{20}$$

Die *Wartungs- und Betriebskosten* ($MAIN$) unterteilen sich in fünf grundlegende Kostenpakete. Die ersten drei Kostenpakete umfassen dabei die Kosten für die drei Wartungsansätze, welche in Kapitel 2.1.1 präsentiert wurden. So werden jährlichen Kosten für eine korrigierende Wartung (CMC), eine intervallbasierte Wartung (PMC) und eine zustandsbasierte Wartung ($CONC$) berücksichtigt. Dabei hängt jede dieser drei Kostenpakete wiederum von seinen jeweiligen Personalkosten, Ersatzteilkosten und Logistikosten ab. Beispielsweise werden so die Kosten für die korrigierende Wartung wie folgt berechnet:

$$CMC = NCM * (MHCM * MHR + SPCM) + LSCM \tag{21}$$

Dabei steht NCM für die Anzahl an Fehler pro Jahr, die durch korrigierende Wartung repariert wurden, $MHCM$ für die Personenstunden pro Reparatur[262], MHR für die Kosten pro Personenstunde, $SPCM$ für die Ersatzteilkosten pro Reparatur und $LCSM$ für die logistischen Kosten pro Jahr für die korrigierende Wartung.[263]

Zusätzlich zu den Wartungskosten werden die jährlichen Betriebskosten in Form von administrativen Kosten (ADC) und Energiekosten (ECC) berücksichtigt. Zusammenfassend gilt folglich für die jährlichen $MAIN$-Kosten: [264]

$$MAIN_{Annual} = CMC + PMC + CONC + ADC + ECC \tag{22}$$

[260] Vgl. Ebd., S. 10 ff.

[261] Vgl. Ebd., S. 14 f.

[262] Dabei ist MHCM = MTTR (Durchschnittliche Reparaturzeit) * NC (Anzahl Personen).

[263] Vgl. Hokstad (1998), S. 15 ff.

[264] Vgl. Ebd.

Die *Verspätungskosten* (DEL) unterteilen sich in zwei Kostenpakete, welche sich anhand der Verspätungsdauer unterscheiden: Kurzeit-Verspätungskosten und Langzeit-Verspätungskosten. Kurzzeit-Verspätungskosten sind dabei Kosten, die durch Verspätungen bis beispielsweise 30 Minuten entstehen. Diese setzten sich hauptsächlich aus Reputationsverlust und Passagierentschädigung zusammen. Langzeit-Verspätungskosten hingegen entstehen bei schwerwiegenderen Fehler, die durch Ausfall von Zügen, Passagierentschädigungen und alternativen Transportbedarf zustande kommen. Beide Kostenpakete werden dabei berechnet, indem die Anzahl an Verspätungen pro Jahr (NSD/NLD) mit den Kosten pro Verspätung (SDC/LDC) multipliziert werden. Es gilt also für die jährlichen DEL-Kosten:[265]

$$DEL_{Annual} = NSD * SDC + NLD * LDC \tag{23}$$

Bei den *Gefahrenkosten* (HAZ) werden jedoch einzelnen Kostenpakete nicht einzeln berechnet und aufsummiert. Anstatt dessen wird zur Berechnung der jährlichen HAZ-Kosten wie folgt vorgegangen:

$$HAZ_{Annual} = NHE * HEC \tag{24}$$

NHE steht dabei für die Anzahl an Katastrophen/Unfällen pro Jahr und HEC für die Kosten pro Katastrophe/Unfall.[266]

Das vorgestellte Modell wurde von Hokstad (1998) anschließend auf eine konkrete Problemstellung angewandt. Es wurde untersucht, ob sich die Einführung eines Zustandsüberwachungssystems (Condition Monitoring) für Weichen-Stellmotoren des Typs R2000 auf bestimmten Streckenabschnitten finanziell lohnt oder nicht. Grundlegend wurde dabei überprüft, ob Lebenszykluskosten der Weiche durch Condition Monitoring (CM) reduziert werden können. Es wurde also ermittelt, ob gilt:[267]

$$\Delta LCC = LCC_{Kein\,CM} - LCC_{CM} \geq 0 \tag{25}$$

Zusammenfassend liefert Hokstad (1998) also ein Kostenmodell, welche für den Einsatz auf Bahn-Infrastruktur konzipiert, bereits für die Bewertung eines Zustandsüberwachungssystems eingesetzt und in Kooperation mit der Deutschen Bahn AG entwickelt wurde.[268]

[265] Vgl. Ebd., S. 17 f.

[266] Vgl. Ebd., S. 18 f.

[267] Vgl. Ebd., S. 25 ff.

[268] Vgl. Ebd., S. 1 ff.

6.1.1.2.2 Kostenmodell von García Márquez et al. (2008)

Auch von García Márquez et al. (2008) wurde, in Kooperation mit dem britischen Bahn-Netzbetreiber Network Rail, ein Kostenmodell für die Bewertung eines Zustandsüberwachungssystem entwickelt, welches sowohl aus Software- als auch Hardwarekomponenten besteht. Wie auch bei Hokstad (1998), ist der Betrachtungsgegenstand dabei ein Stellmotor einer Zugweiche, der jedoch in dieser Studie vom Typ M63 ist. Dennoch unterscheiden sich die Kostenmodelle in einigen Punkten wie beispielsweise im Einbezug der Verspätungsminuten, weshalb im Folgenden das Modell genauer untersucht werden soll.[269]

Das grundlegende Lebenszykluskostenmodell, welches die Gesamtkosten Y berechnet, ist wie folgt formalisiert:

$$Y = \sum_{i=1}^{n} y_i = \lambda * \sum_{i=1}^{n} a_i * c_i^T \tag{26}$$

Dabei steht n für die Anzahl der betrachteten Kostengruppen, sodass y_i den Kapitalwert der Kostengruppe i darstellt. λ hingegen stellt den Kapitalwertfaktor dar, welcher die Diskontierung aus Abschnitt 6.1.1.1.3 bei einem über die Jahre konstanten Zahlungsstroms durchführt. a_i und c_i bezeichnen die i-te Zeile zweier Matrizen A und C. Die Matrix C beinhaltet die Kostenwerte pro Einheit in Form von Elementen c_{ij}. Die Matrix A hingegen besteht aus Elemente a_{ij}, die angeben, wie oft die Kosten c_{ij} anfallen.[270]

Für die Anwendung des Modells auf die Entscheidung über ein Zustandsüberwachungssystem wurden vier Kostengruppen identifiziert, sodass $n = 4$ gesetzt wurde. Die vier Gruppen bilden:

1. Kosten für die Investition in das Zustandsüberwachungssystem
2. Kosten für den Betrieb des Zustandsüberwachungssystems
3. Einsparungen in den Verspätungskosten
4. Einsparungen in den Wartungskosten

Wie an den Namen der Gruppen zu erkennen, stellen die einzelnen Gruppen bereits die Differenz der beiden Optionen (Zustandsüberwachungssystem / Kein Zustandsüberwachungssystem) dar. Es werden also nicht die Lebenszykluskosten beider Optionen im Gesamten berechnet und dann die Differenz gebildet, sondern bereits in den Gruppen die Differenz

[269] Vgl. García Márquez et al. (2008), S. 1175 ff.

[270] Vgl. Ebd., S. 1180.

betrachtet. Somit stellen die ersten beiden Gruppen die Mehrkosten des Zustandsüberwachungssystems dar. Die letzten beiden Gruppen hingegen betrachten die Einsparungen, die die Einführung des Zustandsüberwachungssystems zu Folge hätte. Die vier Kostengruppen sollen nun im Folgenden genauer beschrieben werden.[271]

Die *Kosten für die Investition in das Zustandsüberwachungssystem* stellen die einmaligen Kosten zur Einführung des Systems c_1 dar. Diese setzen sich aus fünf Kostenpaketen zusammen: Kapitalkosten c_{11}, Installationskosten c_{12}, Genehmigungskosten c_{13}, Schulungskosten c_{14} und Energie- und Kommunikationskosten c_{15}. Alle Kostenpakete bilden Mehrkosten, sodass diese ein negatives Vorzeichen erhalten. Fallen die jeweiligen Kostenpakete an, so ist $a_{1i} = 1$ zu setzen.[272]

Die *Kosten für den Betrieb des Zustandsüberwachungssystems* c_2 gehören zu den laufenden Kosten und unterteilen sich in vier Kostenpakete. Die ersten drei Pakete bilden jährliche Energie- und Kommunikationskosten c_{21}, jährliche Kosten für das technische Management c_{22} und die jährlichen Wartungskosten c_{23}. Fallen die Kostenpakete an, so ist wieder $a_{2i} = 1$ zu setzen. Zuletzt werden in dem Modell für diese Gruppe die Kosten für die Herausgabe falscher Alarme modelliert. Dabei beschreibt c_{24} die Kosten pro Fehler für zusätzliche Reparaturen, die durch das System anfallen. Korrespondierend hierzu stellt a_{24} die Häufigkeit von falschen Alarmen pro Jahr dar.[273]

Neben den Mehrkosten beschreibt die Gruppe *Einsparungen in den Verspätungskosten* c_3 laufende Einsparungen des Systems. Hierzu wird angenommen, dass durch bei Erkennung eines Fehlers im Voraus in dieser Situation keine Verspätungen mehr auftauchen. Aus diesem Grund werden die Kosten für eine Verspätungsminute c_{31} mit der Anzahl an jährlich eingesparten Verspätungsminuten a_{31} verrechnet. Die Gruppe ist also explizit von der Anzahl an jährlich eingesparten Verspätungsminuten abhängig. Da es hierbei um Einsparungen handelt, ist c_{31} mit einem positiven Vorzeichen zu versehen. Die eingesparten Verspätungsminuten a_{31} hingegen werden berechnet, indem die Effizienz des Systems η (Prozent der erkannten und vermiedenen Fehler) mit der durchschnittlichen jährlichen Anzahl an Fehlern $\overline{N}$ und Verspätungsminuten $\overline{d}$ multipliziert wird. Es gilt also $a_{31} = \eta * \overline{N} * \overline{d}$.[274]

271 Vgl. Ebd., S. 1180 f.

272 Vgl. Ebd., S. 1180.

273 Vgl. Ebd., S. 1180 f.

274 Vgl. Ebd., S. 1179 ff.

Die letzte Gruppe bilden die *Einsparungen in den Wartungskosten.* Hierzu unterteilt sich die Gruppe in zwei Kostenpakete. Dabei werden sowohl die Einsparungen in Form von weniger intervallbasierte Wartungen als auch die Kosten durch Wartungen, die durch das System entstehen, modelliert. Die Einsparungen an intervallbasierten Reparaturen werden dabei durch Multiplikation der Kosten pro Fehler bei intervallbasierter Wartung c_{41} mit der verringerten jährlichen Anzahl an intervallbasierter Wartung $a_{41} = \eta * \overline{N} \geq 0$ berechnet. Die Kosten, welche durch das System entstehen, werden hingegen durch Multiplikation der Kosten pro zusätzlicher Wartung des Systems $c_{42} = c_{24} \leq 0$ mit der jährlichen Anzahl an Wartungen des neuen Systems $a_{41} = \eta * \overline{N}$ ermittelt.[275]

6.1.1.2.3 Weitere Kostenmodelle

Neben diesen beiden Kostenmodellen, auf dem ein Großteil des im nächsten Abschnitt entwickelten konkreten Kostenmodells aufbaut, soll an dieser Stelle kurz auf die weiteren relevanten Arbeiten von Zhang et al. (2017) und May et al. (2015) eingegangen werden.

Zhang et al. (2017) beschreiben ein Kostenmodell für die Evaluation eines Zustandsüberwachungssystems zur Detektion von Kugellagerschäden. Im Gegensatz zu den beiden zuvor beschriebenen Modellen wird hier also der Zug und nicht die Zuginfrastruktur in Form von Schienen oder Weichen betrachtet. Die Studie zeigt somit, dass Lebenszykluskosten-Modellen auch zur Evaluation von Wartungssystem bei Zügen bzw. Loks angewendet werden können.[276]

May et al. (2015) entwickelten ein Kostenmodell zur Bewertung von Zustandsüberwachungssystemen für Offshore Windturbinen. Dabei unterteilen Sie die Berechnung in die Kostengruppen: Ersetzung von Komponenten, Produktionsverlust, Betriebskosten und Kosten für das Überwachungssystem. Relevante Aspekte für diese Arbeit sind dabei folgende:[277]

- In der ersten und dritten Kostengruppe wird angenommen, dass durch eine frühzeitige Erkennung der Fehler der Schaden am Bauteil noch nicht so schwerwiegend ist. Somit werden weniger Ersatzteile benötigt, die Arbeiten können schneller durchgeführt werden und die Anzahl der Mitarbeiter kann reduziert werden.
- In der zweiten Kostengruppe wird angenommen, dass durch die frühzeitige Warnung das Reparaturteam früher losfahren kann, sodass die Ausfallzeit reduziert wird.

[275] Vgl. Ebd., S. 1181.

[276] Vgl. Zhang et al. (2017), S. 115 ff.

[277] Vgl. May et al. (2015), S. 903 f.

6.1.2 Entwicklung des konkreten Kostenmodells

Nachdem nun im vorherigen Kapitel das Konzept der Lebenszykluskosten und verschiedene Kostenmodelle im Umfeld von Predictive Maintenance und dem Bahnverkehr präsentiert wurden, folgt in diesem Abschnitt die Beschreibung des für die Arbeit entwickelten Kostenmodells zur Bewertung des entwickelten Predictive Maintenance System (PMS). Basis des konkreten Kostenmodells bildeten hierbei die zuvor vorgestellten Modelle, welche auf die konkrete Problemstellung adaptiert wurden.

Die grundlegende Struktur des Kostenmodells ist in Abbildung 34 dargestellt. Fokus lag dabei auf der Unterteilung nach *Kostenart*, die gleichzeitig die Kosten nach der *Lebenszyklusphase* unterteilt. Auf eine Unterteilung der Kostenpakete entlang der Achse *System-/Arbeitsaufbruchsstruktur* wurde hingegen verzichtet. Dabei wurden nur die Kostenarten modelliert, die das Predictive Maintenance System verändert. Ziel des Modells ist demnach zu berechnen, um welchen Betrag $\Delta LCC = LCC_{Kein\,PMS} - LCC_{PMS}$ das Predictive Maintenance System die Lebenszykluskosten der Lok reduziert bzw. erhöht.

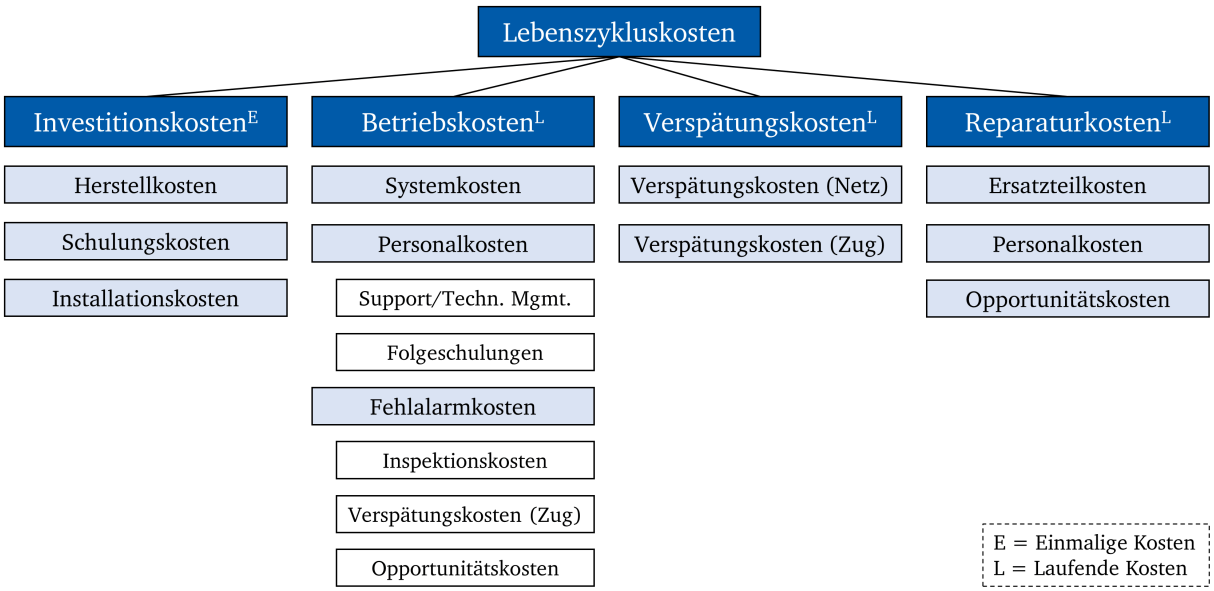

Abbildung 34: Konkretes Kostenmodell

Wie zu erkennen, unterteilt sich die Berechnung der Lebenszykluskosten in Anlehnung an García Márquez et al. (2008) in die vier verschiedene Kostengruppen: Investitionskosten, Betriebskosten, Verspätungskosten und Reparaturkosten. Dabei stellen die Investitionskosten ($INV \geq 0$) die einmaligen Kosten dar, die benötigt werden um das Predictive Maintenance System in den operativen Betrieb zu überführen. Die restlichen Kostengruppen bilden laufende

Kosten, die auf jährlicher Basis ermittelt werden und demnach diskontiert werden müssen. Dabei stellen die Betriebskosten ($\Delta BET_{Jahr} \geq 0$) die Mehrkosten durch den Betrieb des Systems dar. Die Verspätungskosten ($\Delta VER_{Jahr} \leq 0$) und Reparaturkosten ($\Delta REP_{Jahr} \leq 0$) bilden hingegen Minderkosten, also jene Kosten die durch das System eingespart werden können. Formal ausgedrückt gilt also:

$$\Delta LCC = \Delta INV + \sum_{t=1}^{T} \frac{\Delta BET_{Jahr} + \Delta VER_{Jahr} + \Delta REP_{Jahr}}{(1+i)^t} \tag{27}$$

Dabei bezeichnet i den Kalkulationszinssatz und T die durchschnittlich mögliche Einsatzdauer des Systems.

Bei der Erstellung des Modells wurden dabei verschiedene Annahmen getroffen, die dem Modell zugrunde liegen:

- Annahme 1: Die Diskontierung wird mit einem konstanten Kalkulationszinssatz i, welcher also über die komplette Einsatzdauer T konstant ist, und auf jährlicher Basis vorgenommen.
- Annahme 2: ($\Delta BET_{Jahr} + \Delta VER_{Jahr} + \Delta REP_{Jahr}$) ist über die komplette Einsatzdauer T konstant.

Diese Annahmen werden durch weitere Annahmen in den jeweiligen Kostengruppen ergänzt. In den folgenden Unterabschnitten soll nun die Berechnung bzw. Zusammensetzung der einzelnen Kostengruppen genauer erläutert werden.

6.1.2.1 Investitionskosten

Die Investitionskosten INV beschreiben die einmaligen Kosten, die für eine Einführung des Predictive Maintenance System notwendig sind. Da es sich hierbei um eine reine Softwarelösung handelt, unterscheidet sich die Zusammensetzung von den in Kapitel 6.1.1.2 beschrieben Modellen. So werden beispielsweise keine zusätzlichen physikalischen Teile benötigt. Für Modellierung der initialen Kosten eines Softwaresystem wurde neben Hokstad (1998) und García Márquez et al. (2008) deshalb vorwiegend auf Buxmann (2001) zurückgegriffen, welcher die einmaligen Kosten grundlegend in Anschaffungs- und Herstellkosten[278],

[278] Bestehend aus: Kaufpreis/Entwicklungskosten für Softwarekomponenten und Anschaffungsnebenkosten.

Personalkosten[279], Fremdleistungskosten[280], Installations- und Implementierungskosten[281] und sonstige Kosten[282] unterteilt.[283] Dabei wurden nur die Kosten aus der Kostenunterteilung von Buxmann (2001) verwendet, die auch in der konkreten Problemstellung anfallen.

Im Kostenmodell setzt sich die Berechnung der Investitionskosten INV aus drei Kostenpakete zusammen: Herstellkosten ΔHER, Schulungskosten (Personalkosten) ΔSCH und Installationskosten ΔINS. Es gilt also:

$$\Delta INV = \Delta HER + \Delta SCH + \Delta INS \tag{28}$$

Die Herstellkosten ΔHER stellen dabei die Entwicklungskosten der Software dar. Hiermit sind die Kosten gemeint, die für die Programmierung des Vorhersagemodells notwendig sind, sodass die Software online die Logdaten analysieren kann und der Lokführer eine entsprechende Warnung angezeigt bekommt. Die Schulungskosten (Personalkosten) ΔSCH hingegen bilden die Kosten, die für die initiale Schulung der Mitarbeiter anfallen. So muss den Mitarbeitern erläutert werden, dass es ein solches System gibt und wie sie sich im Falle einer Warnung verhalten sollen. Die Installationskosten ΔINS hingegen sind Kosten, welche anfallen, um das entwickelte System in Betrieb zu nehmen. Hierunter fallen die Bereitstellung des Servers, die Dokumentation, das Testen und das Ausrollen des Systems.

6.1.2.2 Betriebskosten

Unter den jährlichen Betriebskosten ΔBET_{Jahr} sind die Kosten zu verstehen, welche für den Betrieb des Predictive Maintenance Systems anfallen. Auch hier wurde neben den beschriebenen Modelle aus Kapitel 6.1.1.2 auf die Unterteilung der laufenden Kosten nach Buxmann (2001) zurückgegriffen. Dieser unterteilt die laufenden Kosten eines Softwaresystems grundlegend in Systemkosten[284], Personalkosten[285], Fremdleistungskosten[286] und sonstige

279 Bestehend aus: Kosten für interne Projektmitarbeiter/EDV Personal und Kosten für Schulungsmaßnahmen.

280 Bestehend aus: Kosten für externe Dienstleistungen.

281 Bestehend aus: Kosten für Datenbereitstellung, Dokumentation und das Testen.

282 Bestehend aus: Kosten für interne Organisation.

283 Vgl. Buxmann (2001), S. 26 ff.

284 Bestehend aus: Miete/Leasing/Lizenzgebühren, Wartungs- und Pflegekosten, Kosten für Benutzung des Rechenzentrums, Kosten für Datenschutz und Datensicherheit, Energie- und Leitungskosten.

285 Bestehens aus: Lohn- und Lohnnebenkosten und Schulungskosten (Folgeschulungen).

286 Bestehend aus: Kosten für regelmäßige externe Dienstleistungen.

Kosten[287].[288] Wie auch zuvor, werden wieder nur solche Kosten berücksichtigt, die in der konkreten Problemstellung relevant sind.

Die jährlichen Betriebskosten ΔBET_{Jahr} setzen sich dabei im Kostenmodell aus Systemkosten ΔSYS_{Jahr}, Personalkosten ΔPER_{Jahr} und den Kosten für Fehlalarme ΔFEH_{Jahr} zusammen. Es gilt folglich:

$$\Delta BET_{Jahr} = \Delta SYS_{Jahr} + \Delta PER_{Jahr} + \Delta FEH_{Jahr} \tag{29}$$

Die jährlichen Systemkosten ΔSYS_{Jahr} beschreiben die Kosten, welche für den jährlichen Betrieb des Predictive Maintenance Systems notwendig sind. Hierunter fallen die Kosten für den Betrieb des Servers (Energie, Softwarelizenzen und Datensicherungen) und Wartungskosten (Updates).

Die jährlichen Personalkosten ΔPER_{Jahr} setzen sich wiederum aus zwei verschiedenen Kostenpaketen zusammen. Diese sind die Kosten für Support bzw. für das technische Management ΔSUP_{Jahr} und Kosten für Folgeschulungen für neue Mitarbeiter ΔFOL_{Jahr}. Formal werden die jährlichen Personalkosten wie folgt berechnet:

$$\Delta PER_{Jahr} = \Delta SUP_{Jahr} + \Delta FOL_{Jahr} \tag{30}$$

Das letzte Kostenpaket ΔFEH_{Jahr} beschreibt in Anlehnung an García Márquez et al. (2008) die jährlichen Mehrkosten für Fehlalarme. Da der Lokführer nicht weiß, ob der Alarm fehlerhaft oder korrekt ist, wird er bei einem Alarm immer gleich vorgehen. Aus diesem Grund werden die Mehrkosten für einen Fehlalarm wie folgt berechnet:

$$\Delta FEH_{Jahr} = \overline{n}_{Fehlalarm,Jahr} * (\Delta INP + \Delta VEI + \Delta OPI) \tag{31}$$

Dabei ist $\overline{n}_{Fehlalarm,Jahr}$ die durchschnittliche Anzahl an Fehlalarmen pro Jahr, welche von der Qualität des Predictive Maintenance Systems abhängt. Für jeden Fehlalarm fallen dabei Inspektionskosten ΔINP, Verspätungskosten des Zuges ΔVEI und Opportunitätskosten ΔOPI an.

Die Inspektionskosten ΔINP sind die Kosten, die verursacht werden, um festzustellen, dass es sich um einen Fehlalarm handelt. So müssen die Werkstattmitarbeiter die Lokomotive eine gewisse Zeit untersuchen, um zu erkennen, dass der Stromrichter noch intakt ist. Die Inspektionskosten ΔINP sind somit abhängig vom Stundensatz der Mitarbeiter STM, der Anzahl

[287] Bestehend aus: Materialkosten und Kalkulatorische Kosten.

[288] Vgl. Buxmann (2001), S. 26 ff.

der beteiligten Mitarbeiter ANI und der durchschnittlichen Dauer der Inspektion ADI. Formal werden die Inspektionskosten ΔINP somit wie folgt berechnet:[289]

$$\Delta INP = STM * ANI * ADI \tag{32}$$

Die beiden weiteren Kostenpakete zur Berechnung der jährlichen Fehlalarmkosten $\Delta VEI = KVM * DDV_{PMS}$ (Verspätungskosten des Zuges) und $\Delta OPI = STO * ADI$ (Opportunitätskosten) werden hingegen in den beiden folgenden Unterabschnitten erläutert, da sie auch im Falle eines korrekten Alarms anfallen.

6.1.2.3 Verspätungskosten

ΔVER_{Jahr} beschreibt in Anlehnung an die Modelle von Hokstad (1998) und García Márquez et al. (2008) die jährlichen Kosteneinsparungen als Folge weniger Verspätungen bei Einsatz des Predictive Maintenance Systems. Die Annahme ist also, dass durch das Predictive Maintenance System die Verspätungen im Vergleich zum bisherigen Wert reduziert werden können. Hierzu werden die Kosteneinsparungen ΔVER_{Jahr} wie folgt berechnet:

$$\Delta VER_{Jahr} = \overline{n}_{Korrektalarm,Jahr} * (\Delta VEZ + \Delta VEN) \tag{33}$$

Dabei bezeichnet $\overline{n}_{Korrektalarm,Jahr}$ die durchschnittliche, jährliche Anzahl an korrekten Vorhersagen eines Stromrichterausfalls, welche von der Leistung des Vorhersagesystems abhängt. Diese werden multipliziert mit den Verspätungskosten, die pro korrekter Vorhersage eingespart werden können, welche wiederum aufgeteilt sind in: Einsparungen in den Verspätungskosten, welche durch die Blockierung des Netzes entstehen, ΔVEN und Einsparungen in Verspätungskosten, welche durch Verspätungen des eigenen Zuges am Zielort verursacht werden, ΔVEZ.

Die Verspätungskosten für Blockierung des Netzes sind Kosten, die durch Verspätungen anderer Züge im Netz entstehen, die also durch die Belegung eines Streckenabschnitts bei einem Ausfall des Stromrichters verursacht werden. Die Annahme ist dabei die folgende:

- Annahme 3:
    - o Kein Predictive Maintenance System: Bei einem Stromrichterausfall blockiert die Lok den Streckenabschnitt.

[289] In Anlehnung an die Berechnung der Wartungskosten von Hokstad (1998).

- Predictive Maintenance System: Bei einem Alarm fährt der Lokführer ein Abstellgleis/einen Bahnhof an, sodass er den Streckenabschnitt nicht blockiert.

Die Verspätungskosteneinsparungen für Blockierungen des Netzes pro korrekter Vorhersage werden dabei wie folgt berechnet:[290]

$$\Delta\text{VEN} = \text{KBM} * (\text{DDB}_{PMS} - \text{DDB}_{Kein\ PMS}) = \text{KBM} * (0 - \text{DDB}_{Kein\ PMS}) \quad (34)$$

Dabei ist KBM der Kostensatz pro Blockierungsminute eines Streckenabschnitts. DDB_{PMS} bezeichnet die durchschnittliche Blockierungsdauer pro korrekter Warnung des Predictive Maintenance Systems, welche aufgrund von Annahme 3 als 0 angenommen wird. $DDB_{Kein\ PMS}$ hingegen stellt die durchschnittliche Dauer der Blockierung bei einem Stromrichterausfall auf der Strecke dar. Dabei gilt nach Annahme 3: $DDB_{Kein\ PMS} \geq DDB_{PMS}$

Die Kosten für Verspätungen des Zuges bezeichnen die Kosten, welche durch eine Verspätung der Lok bzw. der Waggons am Zielort verursacht wird. Diese fallen durch den Lieferverzug beim Kunden, die verspätete Bereitstellung der Waggons für Folgefahrten und die zusätzlichen Lohnkosten für den Lokführer an. In Bezug auf die Verspätungskosten des eigenen Zugs liegt ähnlich zu May et al. (2015) folgende Annahme zugrunde:

- Annahme 4:
  - Kein Predictive Maintenance System: Der Lokführer besitzt keine Vorwarnzeit, sodass er erst beim endgültigen Ausfall eine Ersatzlok/Abschlepplok anfordern kann.
  - Predictive Maintenance System: Der Lokführer verständig schon bei Auftreten des Alarms die Leitstelle, die somit früher eine Ersatzlok/Abschlepplok schicken kann. Die Zeitdifferenz zwischen dem Anfordern der Ersatzlok/Abschlepplok und dem Abstellen der alten Lok auf einem Abstellgleis/in einem Bahnhof wird so eingespart.

Die Kosteneinsparungen ΔVEZ für Verspätungen des Zuges am Zielort pro korrekter Vorhersage werden wie folgt berechnet:[291]

$$\Delta VEZ = KVM * (DDV_{PMS} - DDV_{Kein\ PMS}) \quad (35)$$

Dabei ist KVM der Kostensatz für eine Verspätungsminute am Zielort, DDV_{PMS} die durchschnittliche Dauer der Verspätung am Zielort bei Einsatz des Predictive Maintenance Systems und

[290] In Anlehnung an die Berechnung der Verspätungskosten von García Márquez et al. (2008).

[291] In Anlehnung an die Berechnung der Verspätungskosten von ebd.

$DDV_{Kein\ PMS}$ die durchschnittliche Dauer der Verspätung, wenn kein Predictive Maintenance System eingeführt wird. Nach Annahme 4 wird dabei angenommen, dass $DDV_{PMS} \leq DDV_{Kein\ PMS}$, also die Dauer der Verspätung durch das System reduziert werden kann.

6.1.2.4 Reparaturkosten

Die Reparaturkosten stellen die Kosten dar, welche zur Reparatur des Stromrichters bzw. der defekten Komponenten anfallen.[292] Sie beschreiben also jene Kosten, die zur Beseitigung des in der Arbeit untersuchten Schaden verursacht werden. Dementsprechend bezeichnet ΔREP_{Jahr} die jährlichen Einsparungen der Reparaturkosten, welche durch die Einführung des Predictive Maintenance Systems ermöglicht werden. Die Annahme dabei ähnlich zu May et al. (2015) die Folgende:

- Annahme 5
    - Kein Predictive Maintenance System: Beim Eintreten des Stromrichterausfalls besitzt die Lok schwerwiegendere Defekte. Auch können Folgeschäden durch den Ausfall verursacht worden sein.
    - Predictive Maintenance System: Die Defekte sind bei frühzeitiger Detektion noch nicht so schwerwiegend und können einfacher/schneller/kostengünstiger repariert werden.

Die jährlichen Einsparungen der Reparaturkosten setzen sich dabei wie folgt zusammen bzw. werden wie folgt berechnet:[293]

$$\Delta REP_{Jahr} = \overline{n}_{Korrektalarm,Jahr} * (\Delta ETK + \Delta PKR + \Delta OPP) \quad (36)$$

Dabei ist $\overline{n}_{Korrektalarm,Jahr}$ wie auch im vorherigen Kapitel die durchschnittliche, jährliche Anzahl an korrekten Vorhersagen eines Stromrichterausfalls, welche von der Leistung des Vorhersagesystems abhängt. Diese wird multipliziert mit den Einsparungen in den Reparaturkosten pro korrekter Vorhersage ($\Delta ETK + \Delta PKR + \Delta OPP$). Dabei bezeichnet ΔETK die Einsparungen in den Ersatzteilkosten, ΔPKR die Einsparungen in den Personalkosten und ΔOPP die Reduzierung der Opportunitätskosten.

[292] In Anlehnung an die Wartungskosten von Hokstad (1998) und García Márquez et al. (2008).

[293] In Anlehnung an die Aufteilung der Wartungskosten von Hokstad (1998).

Wie bereits beschrieben, bezeichnet ΔETK die Einsparungen pro korrekter Vorhersage bezüglich der Ersatzteilkosten. Diese wird also berechnet, indem die Kosten für die Ersatzteile mit und ohne das System vergleichen werden:

$$\Delta ETK = ETK_{PMS} - ETK_{Kein\ PMS} \tag{37}$$

Dabei wird nach Annahme 5 davon ausgegangen, dass $ETK_{Kein\ PMS} \geq ETK_{PMS}$, also, dass die Ersatzteilkosten durch das Predictive Maintenance System reduziert werden können.

Neben den Ersatzteilkosten werden die Einsparungen pro korrekter Vorhersage ΔPKR in der Berechnung berücksichtigt. Diese bezeichnen die Personalkosten, welche für die Reparatur des Schadens anfallen. Formal werden die Einsparungen in den Personalkosten ΔPKR in Anlehnung an Hokstad (1998) wie folgt berechnet:

$$\Delta PKR = STM * (ANR_{PMS} * ADR_{PMS} - ANR_{kein\ PMS} * ADR_{Kein\ PMS}) \tag{38}$$

Dabei bezeichnet STM, wie bereits in Kapitel 6.1.2.2 beschrieben, den Stundensatz der Werkstattmitarbeiter, ANR die Anzahl der für die Reparatur benötigen Mitarbeiter und ADR die durchschnittliche Dauer der Reparatur. Auf Grund von Annahme 5 wird dabei davon ausgegangen, dass bei einer frühzeitigen Warnung die Anzahl der Mitarbeiter für eine Wartung geringer ausfällt, also $ANR_{kein\ PMS} \geq ANR_{PMS}$. Des Weiteren wird angenommen, dass die Reparatur weniger Zeit in Anspruch nimmt, also $ADR_{PMS} \leq ADR_{Kein\ PMS}$.

Neben den Ersatzteilkosten und Personalkosten werden bei den Reparaturkosten die Opportunitätskosten berücksichtigt. Diese fallen an, da die Lok während der Reparatur nicht benutzt werden kann, die Lok also keine Einnahmen für das Unternehmen generiert. Die Einsparungen in den Opportunitätskosten pro Warnung ΔOPP werden dabei wie folgt berechnet:

$$\Delta OPP = STO * (ADR_{PMS} - ADR_{Kein\ PMS}) \tag{39}$$

Dabei ist STO der Stundensatz für Nichteinsatz der Lok und $(ADR_{PMS} - ADR_{Kein\ PMS})$ die Reduktion der Reparaturzeit in Stunden.

6.1.3 Anwendung des konkreten Kostenmodells

Nachdem das Kostenmodell im vorherigen Abschnitt vorgestellt wurde, soll nun die Anwendung des Modells erläutert werden. Da, wie bereits einleitend erwähnt, keine Daten von Seiten der DB Cargo bereitgestellt werden konnten, erfolgt die Anwendung mit Hilfe beispielhafter Schätzwerte. Somit ist es bei Verfügbarkeit der realen Schätzwerte leicht nachzuvollziehen, wie das System aus finanzieller Sicht bewertet werden kann.

Das Kapitel unterteilt sich dabei in drei Abschnitte, welche gleichzeitig die Schritte bilden, die bei der beispielhaften Anwendung des Kostenmodells durchlaufen wurden. Zunächst wird im ersten Abschnitt beschrieben, wie die Parameter des Kostenmodells mit Hilfe eines Fragebogens ermittelt wurden. Im zweiten Teil wird aufgezeigt, wie das mit den konkreten, beispielhaften Schätzwerten versehene Kostenmodell für eine kostensensitive Evaluierung verwendet wurde. Im letzten Abschnitt wird schlussendlich die Verwendung der kostensensitiven Evaluierung zur finanziellen Bewertung des Systems beschrieben.

6.1.3.1 Ermittlung der Schätzwerte

Zur Ermittlung der Schätzwerte, welche für die Anwendung des Modells notwendig sind, wurde ein Fragebogen in Form einer Excel-Tabelle entwickelt, welcher die zu schätzenden Werte in einer hierarchischen Form abfragt. Dabei ist jedem Schätzwert eine Beschreibung als Hilfestellung angefügt. Da jedoch, wie einleitend beschrieben, keine realen Schätzwerte bereitgestellt werden konnte, wurde der Fragebogen mit beispielhaften Werten versehen. Dabei wurde versucht, die Kostenwerte möglichst realistisch zu schätzen. Der Fragebogen inklusiver aller beispielhaften Schätzwerte ist im Anhang B beigefügt. Zur besseren Übersicht sind die darin enthaltenen Werte (mit den jeweiligen Parametern) in komprimierter Form in Tabelle 9 dargestellt. Neben diesen Werten wurde die Nutzungsdauer auf $T = 20$ Jahre und der Kalkulationszinssatz in Anlehnung an das Bundesfinanzministerium auf $i = 1{,}0\%$ gesetzt.[294]

Tabelle 9: Parameter und Schätzwerte (Fragebogen)

Investitionskosten		Betriebskosten		Verspätungskosten		Reparaturkosten	
Parameter	Wert	Parameter	Wert	Parameter	Wert	Parameter	Wert
ΔHER	22.400	ΔSYS_{Jahr}	6.000	KBM	0	ETK_{PMS}	200
ΔSCH	1.000	ΔSUP_{Jahr}	6.720	DDB_{PMS}	0	$ETK_{Kein\ PMS}$	50.000
ΔINS	5.600	ΔFOL_{Jahr}	0	$DDB_{Kein\ PMS}$	0	ANR_{PMS}	2
		STM	50	KVM	2,5	$ANR_{kein\ PMS}$	4
		ANI	2	DDV_{PMS}	60	ADR_{PMS}	8
		ADI	6	$DDV_{Kein\ PMS}$	120	$ADR_{Kein\ PMS}$	24
						STO	100

[294] Vgl. Bundesministerium der Finanzen (2016), S. 2.

6.1.3.2 Kostensensitive Evaluierung

Mit den Schätzwerten aus dem Fragebogen wurde anschließend eine kostensensitive Evaluierung[295] durchgeführt. Idee dabei ist es, die Konfigurationen aus Kapitel 4.3.1 nach ihrem ΔLCC zu sortieren, um so die beste Konfiguration in Bezug auf die Lebenszykluskosten auswählen zu können. Die Leistung einer Konfiguration soll durch ihren ΔLCC bewertet werden.

Dabei sind im Kostenmodell zur Berechnung von ΔLCC zwei Parameter abhängig von der Leistung des Predictive Maintenance Systems: $\overline{n}_{Korrektalarm,Jahr}$ und $\overline{n}_{Fehlalarm,Jahr}$. $\overline{n}_{Korrektalarm,Jahr}$ stellt dabei die durchschnittliche, jährliche Anzahl an korrekten Vorhersagen eines Stromrichterausfalls dar.[296] Diese kann im Falle der Kreuzvalidierung demnach wie folgt geschätzt werden:

$$\begin{aligned}\overline{n}_{Korrektalarm,Jahr} &= \frac{TP}{Zeitraum\ der\ Daten} * \frac{Gesamtanzahl\ der\ Fehlertouren}{Stichprobenanzahl\ der\ Fehlertouren} \\ &= \frac{TP}{3\ Jahre} * \frac{40}{32} \approx 0{,}417 * TP\end{aligned} \tag{40}$$

Hierbei existiert der Faktor $\frac{40}{32}$, sodass die Anzahl an True Positive von der Stichprobe auf die Grundgesamtheit skaliert werden.

$\overline{n}_{Fehlalarm,Jahr}$ hingegen beschreibt die durchschnittliche jährliche Anzahl an Fehlalarmen.[297] Auch diese kann auf Grundlage der verfügbaren Maße bei der Kreuzvalidierung wie folgt geschätzt werden:

$$\begin{aligned}\overline{n}_{Fehlalarm,Jahr} &= \frac{FP}{Zeitraum\ der\ Daten} * \frac{Gesamtanzahl\ der\ Normaltouren}{Stichprobenanzahl\ der\ Normaltouren} \\ &= \frac{FP}{3\ Jahre} * \frac{91.309}{418} \approx 72{,}814 * FP\end{aligned} \tag{41}$$

Hierbei existiert wiederum ein Faktor $\frac{91.309}{418}$, sodass die Anzahl an False Positive von der Stichprobe auf die Grundgesamtheit skaliert werden.

Letztlich ist es also möglich ΔLCC durch die Maße FP und TP für jede Konfiguration zu berechnen. Gegeben den Schätzwerten kann ΔLCC also als eine Funktion $\Delta LCC(TP, FP)$ verstanden werden, welche auf jede der Konfigurationen angewandt werden kann. Die Herleitung

[295] Wie beispielweise auch bei Prytz et al. (2013), S. 121.

[296] Siehe Kapitel 6.1.2.3.

[297] Siehe Kapitel 6.1.2.2.

und Anwendung der Funktion $\Delta LCC(TP, FP)$ für die im vorherigen Abschnitt festgelegten Schätzwerte soll nun im Folgendem beschrieben werden.

Zu Bestimmung von $\Delta LCC(TP, FP)$ wurden zunächst die Schätzwerte aus Tabelle 9 (S. 101) in die Formeln von Kapitel 6.1.2 eingesetzt.[298] Hieraus ergibt sich:

$$\begin{aligned} \Delta INV &= 29.000 \\ \Delta BET_{Jahr} &= 12.720 + 1.350 * \overline{n}_{Fehlalarm,Jahr} \\ \Delta VER_{Jahr} &= -150 * \overline{n}_{Korrektalarm,Jahr} \\ \Delta REP_{Jahr} &= -55.400 * \overline{n}_{Korrektalarm,Jahr} \end{aligned} \tag{42}$$

Anschließend wurde $\overline{n}_{Korrektalarm,Jahr}$ und $\overline{n}_{Fehlalarm,Jahr}$ durch Formel (40) und (41) ersetzt. Somit sind alle Kostengruppen nun abhängig von TP und FP:

$$\begin{aligned} \Delta INV &= 29.000 \\ \Delta BET_{Jahr} &\approx 12.720 + 98.299 * FP \\ \Delta VER_{Jahr} &\approx -63 * TP \\ \Delta REP_{Jahr} &\approx -23.102 * TP \end{aligned} \tag{43}$$

Schlussendlich kann $\Delta LCC(TP, FP)$ dann wie folgt berechnet werden:

$$\begin{aligned} \Delta LCC(TP, FP) &= \Delta INV + \sum_{t=1}^{T} \frac{\Delta BET_{Jahr} + \Delta VER_{Jahr} + \Delta REP_{Jahr}}{(1+i)^t} \\ &=^{299} \Delta INV + \left(\Delta BET_{Jahr} + \Delta VER_{Jahr} + \Delta REP_{Jahr}\right) \sum_{t=1}^{T} \frac{1}{(1+i)^t} \\ &= 29.000 + (12.720 + 98.299 * FP - 23.165 * TP) * \sum_{t=1}^{20} \frac{1}{(1+0{,}01)^t} \\ &\approx 29.000 + (12.720 + 98.299 * FP - 23.165 * TP) * 18{,}0456 \\ &\approx 258.540 + 1.773.864 * FP - 418.026 * TP \end{aligned} \tag{44}$$

Mit Hilfe der Funktion $\Delta LCC(TP, FP)$ wurde nun eine kostensensitive Evaluierung durchgeführt, d.h. für jede Konfiguration wurde ihr $\Delta LCC(TP, FP)$ ermittelt. Die besten zehn Konfigurationen sind dabei in Tabelle 10 (S. 104) dargestellt.

298 Eine detaillierte Berechnung ist im Anhang C gegeben.

299 Umformung in Anlehnung an García Márquez et al. (2008), S. 1185.

Tabelle 10: Kostensensitive Evaluierung

	Instanz-Ebene							Tour-Ebene		Maße		
Rang	Zeitverlaufs-attribut	Label-Zeitfenster	Klassen-Verhältnis	Klassifizierer	J48 Pruning RF Anzahl/Baum Adaboost Iteration	J48 Konf.-Level RF Iterationen Adaboost Baumtiefe	J48 Anzahl Blatt RF Anzahl Attr.	Klassifizierer	Verh. Schwellenwert Akt. Sinkrate	TP	FP	ΔLCC(TP,FP)
1	An	3600	50	Adaboost	1000	1		Akt.	0,2	3	0	-995.538
1	An	3600	50	Adaboost	100	1		Akt.	0,2	3	0	-995.538
3	An	3600	50	Adaboost	1000	1		Akt.	0,0667	2	0	-577.512
3	An	3600	50	Adaboost	100	1		Akt.	0,0667	2	0	-577.512
3	An	3600	50	Adaboost	100	1		Akt.	0,0333	2	0	-577.512
3	An	3600	50	Adaboost	1000	1		Akt.	0,0333	2	0	-577.512
3	An	3600	50	Adaboost	100	1		Akt.	0,0222	2	0	-577.512
3	An	3600	50	Adaboost	1000	1		Akt.	0,0222	2	0	-577.512
9	Aus	1800	20	Adaboost	100	1		Akt.	0,0333	1	0	-159.486
9	Aus	1800	20	Adaboost	1000	1		Akt.	0,0333	1	0	-159.486

6.1.3.3 Interpretation des Ergebnisses

Mit Hilfe der Kostenevaluation auf dem vorherigen Abschnitt kann nun eine finanzielle Bewertung des Predictive Maintenance System getätigt werden. Dabei wird angenommen, dass die beste Konfiguration des vorherigen Kapitels die Leistungsfähigkeit des Systems beschreibt. Wichtig ist jedoch anzumerken, dass durch Auswahl des besten Modells im Rahmen der Kreuzvalidierung die Klassifikationsleistung tendenziell überschätzt wird, sodass es sich hierbei um eine optimistische Bewertung des Systems handelt.

Wie in Tabelle 10 zu erkennen, erreicht die beste Konfiguration bei den beispielhaften Schätzwerten einen Wert von $\Delta LCC = -995.538$. Da $\Delta LCC \leq 0$ ist, werden durch das Predictive Maintenance System die Lebenszykluskosten der Lok bezogen eine Nutzungsdauer von 20 Jahren verringert. Somit lässt sich also im Falle der Beispieldaten sagen, dass sich das System in der aktuellen Version auf Basis der vorliegenden Daten finanziell rechnen kann. Für eine finale und realistische Bewertung des Systems ist es jedoch, wie bereits einleitend beschrieben notwendig, dass die realen Schätzwerte zur Berechnung von ΔLCC verwendet werden.

6.2 Nicht-Finanzielle Analyse

Neben der Bewertung einer IT-Investition aus einer rein finanzieller Sichtweise, werden IT-Investitionen in der Literatur auch anhand nicht-monetärer Kriterien bewertet. So entwickelten beispielweise Jonen et al. (2004) eine Balanced IT-Decision Card, bei denen IT-Projekte aus sechs verschiedenen Perspektiven bewertet werden können: Sicherheitsperspektive, Prozessperspektive, Innovationsperspektive, Mitarbeiterperspektive, Kundenperspektive und Finanzperspektive.[300] Beispielweise macht es so vielleicht aus der Finanzperspektive keinen Sinn, eine neue Firewall einzuführen, jedoch lässt sich die Firewall eventuell durch eine erhöhte IT-Sicherheit rechtfertigen. Im Folgenden soll deshalb erläutert werden, welche nicht-monetären Vorteile für die Einführung des Predictive Maintenance Systems sprechen bzw. welche nicht-monetären Gründe für eine Umsetzung sprechen.

Ein möglicher nicht-monetärer Grund ist, dass seitens der DB Cargo eine *Digitale Transformation* im Asset Management bzw. der Instandhaltung beschlossen wurde. Diese Strategie umfasst intelligente Fahrzeuge, eine zentrale Datenerhebung, umfangreiche Analyseverfahren und eine Automatisierung der Prozesse.[301] Das System könnte einen ersten Schritt in der Umsetzung dieser Strategie darstellen und somit als Leuchtturm-Projekt im Rahmen der Digitalen Transformation bzw. für die Einführung von Predictive Maintenance dienen. So könnte man durch das System weitere Investitionen in den Ausbau von Predictive Maintenance rechtfertigen und eine höhere Aufmerksamkeit des Managements erhalten.

Ein weiterer nicht-finanzieller Grund ist, dass die Logdaten eine *sehr große Datenmenge* zur Verfügung stellen, welche potentiell viele Informationen beinhalten, die bisher aber kaum genutzt werden. Die Einführung des Systems hätte somit den Vorteil, dass es eines der ersten Projekte darstellt, die einen Mehrwert aus den Daten in Form zusätzlicher Informationen generieren.

Auch die *Sensibilisierung der Mitarbeiter* für den Umgang mit der Digitalisierung könnte für die Einführung des Systems sprechen. So könnte die korrekte Vorhersage von Stromrichterschäden die Mitarbeiter vom Nutzen der Digitalisierung überzeugen, damit Barrieren abbauen und somit die Akzeptanz der Mitarbeiter für neue Technologien steigern.

Durch die Vorhersage von Stromrichterausfällen kann das System zur Steigerung der Zuverlässigkeit der Züge und zur Reduzierung der Verspätungen führen. Dies könnte die

[300] Vgl. Jonen et al. (2004), S. 196 ff.

[301] Vgl. DB Cargo (2017).

Kundenzufriedenheit erhöhen, da die Waren rechtzeitig bzw. mit einer geringen Verspätung geliefert werden. Somit ist es möglich, dass das *Image* der DB Cargo mittelfristig steigt und dazu führt, dass neue Kunden akquiriert werden können.

Zusammenfassend existieren also mehrere nicht-finanzielle Gründe, die für eine Investition in das System sprechen. Somit kann es unabhängig von der finanziellen Bewertung sinnvoll sein, dass System in den operativen Betrieb zu überführen.

7 Zusammenfassung und Ausblick

Die Zielsetzung der Masterarbeit bestand in der Beantwortung der drei zu Beginn vorgestellten Forschungsfragen. Hierzu wurden in Kapitel 2 zunächst die theoretischen Grundlagen gelegt, bevor in Kapitel 3 die vorliegenden Daten und das bisherige System von Kauschke et al. (2016) vorgestellt wurde. Nach diesen grundlegenden Ausführungen wurde in Kapitel 5, 6 und 7 die einzelnen Forschungsfragen untersucht. Die Ergebnisse sollen im Folgenden kurz zusammengefasst werden:

Ziel der ersten Forschungsfrage war es zu klären, ob es möglich ist, die Vorhersagequalität des bisherigen Predictive Maintenance Systems von Kauschke et al. (2016) durch Verwendung alternativer Konzepte bzw. Ansätze zu verbessern. Hierzu wurden Änderungen in drei der vier grundlegenden Bestandsteile des Systems vorgenommen. So wurde ein gleitendes Zeitfenster bei der Datentransformation eingeführt, welches es ermöglicht, auch den zeitlichen Verlauf der Diagnosecodes abzubilden. Auch wurde bei Instanz-Ebenen Klassifizierung drei alternative Lernalgorithmen genutzt: AdaBoost, Random Forest und J48. Dabei wurden die Parameter auf dieser Ebene durch eine Kreuzvalidierung an das Problem angepasst, sodass auch der J48 Klassifizierer eine Neuerung darstellt. Zuletzt wurden auf Tour-Ebene zwei neue Klassifizierer entwickelt, welche bei der Klassifikation einer Tour als Fehlertour eine Warnung an den Lokführer herausgeben. Der lernende Verhältnisklassifizierer lernt dabei automatisch ein Klassifikationsmodell, welches Normal- und Fehlertouren anhand des Anteils der positiven Vorhersagen auf Instanz-Ebene trennt. Beim Aktivierungsklassifizierer hingegen wird für jede Tour ein Aktivierungslevel verwendet, welches bei einer positiven Vorhersage auf Instanz-Ebene ansteigt, jedoch mit der Zeit um einen festen Betrag abfällt. Das Tour-Ebenen Klassifikationsmodell wird auch hier automatisch auf Grundlage der Aktivierungsverläufe bei Normal- und Fehlertouren erstellt.

Die neuen und alten Verfahren wurden anschließend implementiert und mit Hilfe des Hochleistungsclusters der TU Darmstadt evaluiert. Die Ergebnisse bei der Kreuzvalidierung zeigen, dass die neuen Verfahren sowohl bezogen auf die Accuracy als auch auf die F1-Measure eine Verbesserung erreichen konnten. Die beste Konfiguration bei der Kreuzvalidierung konnte 5 (Accuracy) bzw. 10 (F1-Measure) der 32 Fehlertouren erfolgreich erkennen. Bei der Evaluierung auf der Testmenge wurde hingegen keine der 8 Fehlertouren erkannt. Dennoch konnten die neuen Verfahren durch weniger Fehlalarme eine höhere Accuracy erreichen.

Ziel der zweiten Forschungsfrage war es zu untersuchen, ob es bei einer Warnung vor einem drohenden Stromrichterausfall auch möglich ist, vorherzusagen, wie lange es noch bis zu

diesem Ausfall dauert. Motivation hierfür war, dass der Lokführer mit diesen zusätzlichen Informationen entscheiden kann, ob er beispielsweise direkt ein Abstellgleis anfährt oder die Fahrt noch planmäßig beendet. Um diese Zeitvorhersage zusätzlich zur binären Warnung zu ermöglichen, wurde das zuvor entwickelte System angepasst bzw. erweitert. Die größte Anpassung war dabei die Einführung von Multiklassen-Labels auf Instanz-Ebene, die jeweils verschiedene Zeitintervalle vor dem Ausfall darstellen. Diese Einführung hatte zur Folge, dass auch die Instanz-Ebenen und Tour-Ebenen Klassifizierung auf den Umgang mit Multiklassen angepasst werden musste. Nachdem die Anpassungen bzw. Erweiterungen implementiert wurden, wurde auch hier eine Evaluierung mit Hilfe des Hochleistungsclusters der TU Darmstadt durchgeführt. Ergebnis der Kreuzvalidierung war hierbei, dass die binäre Klassifikation in Fehler- und Normaltouren auf einem ähnlichen Leistungsniveau (bezogen auf den F1-Wert) wie bei Forschungsfrage 1 liegt, jedoch die Genauigkeit der Zeitvorhersage gering ausfällt. Somit zeigen die Ergebnisse, dass sich die Zeitvorhersage mit Hilfe der verwendeten Methode nicht für einen praktischen Einsatz eignet.

Das Ziel der dritten Forschungsfrage war es, das entwickelte Predictive Maintenance System aus wirtschaftlicher Sicht zu untersuchen. Hierzu wurde im Rahmen der finanziellen Analyse ein Kostenmodell entwickelt, welches das System auf Basis der Erhöhungen bzw. Einsparungen in den Lebenszykluskosten bewertet. Da aus Aufwands- und Datenschutzgründen keine Kostendaten von der DB Cargo vorlagen, wurde die Anwendung des Kostenmodells mit Hilfe von beispielhaften Kostenwerten erläutert. Neben der finanziellen Analyse wurde eine nicht-finanzielle Analyse durchgeführt. Hierbei wurden Gründe erarbeitet, welche aus nicht-monetärer Sichtweise für eine Investition in das System sprechen. Letztlich liefert das Kapitel somit ein Instrument, welches es der DB Cargo ermöglicht, das Predictive Maintenance System aus finanzieller als auch aus nicht-finanzieller Sichtweise zu bewerten und somit eine fundierte Entscheidung über eine Investition erlaubt.

Die vorliegende Arbeit stellt schlussendlich einen wichtigen Schritt für die Einführung eines Predictive Maintenance Systems zur Vorhersage von Stromrichterschäden dar. Dabei wurden bereits während der Bearbeitung einige potentielle Erweiterungsmöglichkeiten identifiziert:

Zum einen wäre es interessant zu prüfen, ob andere Algorithmen auf Instanz-Ebene die Klassifikationsleistung erhöhen können. So beschränkt sich die vorliegende Arbeit auf die Klassifizierer AdaBoost, Random Forest und J48. Besonderes Support Vektor Maschinen und Neuronale Netzwerke könnten hierbei eine mögliche Alternative darstellen. Zudem wäre es bei der Zeitvorhersage möglich, einen Regressions-Ansatz zu verwenden. Anstatt also ein Zeitintervall vor dem Fehler zu ermitteln, könnte so die Zeit bis zum Fehler als numerischer Wert

vorhersagt werden. Dies hätte den Vorteil, dass keine konkreten Zeitintervalle definiert werden müssten. Auch wäre es vorteilhaft, dass System auf einer größeren Anzahl an Stromrichterausfällen zu testen. So ist die Evaluation aktuell auf 40 Ausfälle limitiert. Beispielsweise könnten hierfür eventuell Daten, die weiter in der Vergangenheit liegen, genutzt werden. Letztlich wäre es interessant, das entwickelte System auf andere Fehlertypen wie beispielsweise den „Zentralschraubenbruch“ anzuwenden und somit zu prüfen, wie gut es bei anderen Fehlern abschneidet.

Anhang

Anhang A: Übersicht der PHP Skripte und Java Klassen

Tabelle 11: PHP-Skripte

Skriptname	Beschreibung
dataset_binary.php	Erstellt Instanzen ohne Zeitverlaufsattributen und speichert dies in der Datenbank ab.
dataset_header_binary.php	Erstellt den Header für Instanzen ohne Zeitverlaufs-attributen und speichert dies in der Datenbank ab.
dataset_window.php	Erstellt Instanzen mit Zeitverlaufsattribute und speichert dies in der Datenbank ab.
dataset_header_window.php	Erstellt den Header für Instanzen mit Zeitverlaufsattribute und speichert dies in der Datenbank ab.
time_to_failure.php	Berechnet die Zeit zum Fehler für jede Instanz und aktualisiert die Instanzen in der Datenbank.
dbconnect.php	Erstellt eine Verbindung zum Datenbankserver.
functions.php	Allgemeine Funktionen (bspw. Bestimmung des Ausfallzeitpunktes in einer Fehlertour)

Tabelle 12: Java-Klassen

Package/Klassenname	Beschreibung
evaluation	
MultiTimeEvaluator	Evaluiert die Zeitvorhersagen bei Versuchsreihe 2. Berechnet hierfür die in Abschnitt 2.4.2 beschriebenen Multiklassen-Maße.
TimeEvaluator	Abstrakte Klasse für die Evaluation von Zeitvorhersagen
filter	
MultiSpreadSubsample	SpreadSubsample Filter für Verwendung bei Multiklassen. Implementation der Anpassung aus Abschnitt 5.1.2.
masterthesis	
Main	Startet die Durchführung der Experimente gegeben der spezifischen Konfiguration.
pointmanager	
BinaryPointManager	Lädt die Instanzen aus der Datenbank herunter und versieht diese mit einer Klasse. Version ohne Zeitattribut und für Versuchsreihe 1 (Binäre Problemstellung)
BinaryTimePointManager	Lädt die Instanzen aus der Datenbank herunter und versieht diese mit einer Klasse. Version mit Zeitattribut und für Versuchsreihe 1 (Binäre Problemstellung)
MultiPointManager	Lädt die Instanzen aus der Datenbank herunter und versieht diese mit einer Klasse. Version ohne Zeitattribut und für Versuchsreihe 2 (Multiklassen Problemstellung)
MultiTimePointManager	Lädt die Instanzen aus der Datenbank herunter und versieht diese mit einer Klasse. Version mit Zeitattribut und für Versuchsreihe 2 (Multiklassen Problemstellung)

PointManager	Abstrakte Klasse von der alle PointManager erben müssen. Definition grundlegender Funktionen, welche in der konkreten Klasse implementiert werden müssen.
skripts	
BatchCleaner	Löscht alle Slurm-Skripte, die nicht in einer spezifizierten Liste vorkommen.
BatchCreatorBinary	Erstellt alle Slurm-Skripte der neuen Methoden bei der Kreuzvalidierung von Versuchsreihe 1 (Binäre Problemstellung)
BatchCreatorBinaryBenchmark	Erstellt alle Slurm-Skripte der alten Methoden bei der Kreuzvalidierung von Versuchsreihe 1 (Binäre Problemstellung)
BatchCreatorBinaryDownload	Erstellt Slurm-Skripte für Versuchsreihe 1 (Binäre Problemstellung), welche die Instanzen/Touren herunterladen und diese auf dem Cluster abspeichern. Somit wird die Datenbanklast erheblich reduziert, da die Daten auf dem Dateiensystem des Clusters liegen.
BatchCreatorMulti	Erstellt alle Slurm-Skripte für Versuchsreihe 2 (Multiklassen Problemstellung)
BatchCreatorMultiDownload	Erstellt Slurm-Skripte für Versuchsreihe 2 (Multiklassen Problemstellung), welche die Instanzen/Touren herunterladen und diese auf dem Cluster abspeichern. Somit wird die Datenbanklast erheblich reduziert, da die Daten auf dem Dateiensystem des Clusters liegen.
BatchTesten	Erstellt die Slurm-Skripte für die Testmenge
tourclassifier	
BinaryActivationTourClassifier	Implementation des Aktivierungsklassifizierers aus Versuchsreihe 1. Siehe Abschnitt 4.1.3.2
BinaryRatioLeanTourClassifer	Implementation des lernenden Verhältnisklassifizierers aus Versuchsreihe 1. Siehe Abschnitt 4.1.3.1
BinaryRatioTourClassifier	Implementation des nicht lernenden Verhältnisklassifizierers aus Versuchsreihe 1. Siehe Abschnitt 3.2.3.2
DecisionStumpCostSensitiv	Umsetzung des kostensensitiven DecisionStumps aus Abschnitt 4.1.3.1
MultiActivationTourClassifier	Implementation des Aktivierungsklassifizierers aus Versuchsreihe 2. Siehe Abschnitt 5.1.3
MultiRatioLearnTourClassifier	Implementation des lernenden Verhältnisklassifizierers aus Versuchsreihe 2. Siehe Abschnitt 5.1.3
TimeTourClassifier	Interface für alle Tour-Ebenen Klassifizierer, welche eine Zeitvorhersage durchführen können.
TourClassifier	Abstrakte Klasse für alle Tour-Ebenen Klassifizierer. Implementiert zusätzlich Methoden, welche die Vorhersagen in der Datenbank abspeichern.
tourmanager	
TourManager	Verantwortlich für die Auswahl, das Labeln und das Erstellen von Touren

Anhang B: Fragebogen Kostenmodell

Tabelle 13: Fragebogen für das Kostenmodell

Ebene 0	Ebene 1	Ebene 2	Ebene 3	Erklärung	Kein PM-System	PM-System	Einheit	Kommentar
Investitionskosten (Einmalig)					0,00	29000,00	€	
	Herstellungskosten			*Entwicklungskosten der PM-Software, sodass ein Online-Einsatz möglich ist.*	0,00	22400,00	€	
	Schulungskosten (Personalkosten)			*Kosten für die initiale Schulung der Mitarbeiter*	0,00	1000,00	€	
	Installationskosten			*Kosten für:* *- Bereitstellung des Servers* *- Dokumentation des Systems* *- Testen des Systems* *- Roll-Out des Systems*	0,00	5600,00	€	
Betriebskosten (Laufend)								
	Systemkosten			*Kosten für:* *- Betrieb Server (Energie + Softwarelizenzen + Datensicherung)* *- Wartungskosten (Updates)*	0,00	6000,00	€/Jahr	
	Personalkosten				0,00	6720,00	€/Jahr	
		Support/Technisches Management		*Kosten für Support bzw. für das technische Management*	0,00	6720,00	€/Jahr	
		Folgeschulungen		*Kosten für Folgeschulungen (für neue Mitarbeiter)*	0,00	0,00	€/Jahr	
	Kosten für Fehlalarm			*Kosten für einen Fehlalarm*	0,00	1350,00	€/Fehlalarm	
		Inspektionskosten Stromrichter		*Kosten für Inspektion der Lok/des Stromrichters*	0,00	600,00	€/Fehlalarm	
			Kostensatz Mitarbeiter	*Stundensatz für Werkstattmitarbeiter*	50,00		€/(h*Mitarbeiter)	
			Anzahl Mitarbeiter	*Anzahl der beteiligten Mitarbeiter*	0,00	2,00	Mitarbeiter	
			Dauer inspektion	*Durchschnittliche Dauer der Inspektion eines nicht defekten Stromrichters*	0,00	6,00	h	
Verspätungskosten (Laufend)					300,00	150,00	€/Ausfall	
	Kosten für Verspätungen im Netz			*Kosten durch Verspätungen anderer Züge. Diese resultieren aus der Belegung des Streckenabschnitts der defekten Lok.*	0,00	0,00	€/Ausfall	
		Kosten für Blockierungsminute eines Streckenabschnitt		*Kostensatz für die Blockierung eines Streckenabschnitt*	0,00		€/min	
		Dauer der Blockierung		*Durchschnittliche Dauer der Streckenblockierung*	0,00	0,00	min	**Annahme:** Kein PM-System: Bei einem Stromrichterausfall blockiert die Lok den Streckenabschnitt. PM-System: Bei einem Alarm fährt der Lokführer ein Abstellgleis/einen Bahnhof an, sodass er den Streckenabschnitt nicht blockiert. Blockierungsdauer(Kein PM) > Blockierungsdauer (PM)
	Kosten für Verspätungen des Zugs			*Kosten für die Verspätung der Lok am Ziebrt.*	300,00	150,00	€/(Ausfall bzw. Alarm)	
		Kosten für Verspätungsminute		*Kosten für eine Verspätungsminute verursacht durch:* *- Lieferverzug* *- Verspätete Bereitstellung Waggons für Folgefahrten* *- Personalkosten Lokführer*	2,50		€/min	

		Dauer der Verspätung		*Durchschnittliche Dauer der Verspätung am Zielort*	120,00	60,00	min	**Annahme:** Kein PM-System: Der Lokführer besitzt keine Vorwarnzeit, sodass er erst beim endgültigen Ausfall eine Ersatzlok/Abschlepplok anfordern kann. PM-System: Der Lokführer verständig schon bei Auftreten des Alarms die Leitstelle, die somit früher eine Ersatzlok/Abschlepplok schicken kann. Die Zeitdifferenz zwischen dem Anfordern der Ersatzlok/Abschlepplok und dem Abstellen der alten Lok auf einem Abstellgleis/in einem Bahnhof wird so eingespart. Verspätungsdauer(Kein PM) > Verspätungsdauer (PM)
Reparaturkosten (Laufend)					57200,00	1800,00	€/Ausfall	
	Reparaturkosten				57200,00	1800,00	€/Ausfall	**Annahme:** Kein PM-System: Beim Eintreten des Stromrichterausfalls besitzt die Lok schwerwiegendere Defekte. Auch können Folgeschäden durch den Ausfall verursacht worden sein. PM-System: Die Defekte sind bei frühzeitiger Detektion noch nicht so schwerwiegend und können einfacher/schneller/kostengünstiger repariert werden.
		Ersatzteilkosten		*Ersatzteilkosten zur Reparatur*	50000,00	200,00	€/Ausfall	Erstatzteilkosten(Kein PM) >= Ersatzteilkosten(PM)
		Personalkosten		*Personalkosten für die Durchführung der Reparatur*	4800,00	800,00	€/Ausfall	
			Kostensatz Mitarbeiter	*Kostensatz für Werkstattmitarbeiter*	50,00		€/(h*Mitarbeiter)	
			Anzahl Mitarbeiter	*Anzahl der beteiligten Mitarbeiter*	4,00	2,00	Mitarbeiter	Anzahl Mitarbeiter(Kein PM) >= Anzahl Mitarbeiter(PM)
			Dauer Reparatur	*Durchschnittliche Dauer der Reparatur*	24,00	8,00	h	Reparaturdauer(Kein PM) >= Reparaturdauer(PM)
		Oppurtunitätskosten			2400,00	800,00	€/Ausfall	
				Kostensatz für Nichteinsatz der Lok	100		€/h	
Allgemeine Angaben								
	Einsatzdauer			Wie lange werden die Loks (BR 185) im Durchschnitt noch verwendet? Wie lange kann das PM-System im Durchschnitt noch eingesetzt werden?	20		Jahre	

Anhang C: Umformungen Kostenmodell

$$\begin{aligned}\boldsymbol{\Delta INV} &= \Delta HER + \Delta SCH + \Delta INS \\ &= 22.400 + 1.000 + 5.600 \\ &= 29.000\end{aligned} \tag{45}$$

$$\begin{aligned}\boldsymbol{\Delta BET_{Jahr}} &= \Delta SYS_{Jahr} + \Delta PER_{Jahr} + \Delta FEH_{Jahr} \\ &= \Delta SYS_{Jahr} + \left(\Delta SUP_{Jahr} + \Delta FOL_{Jahr}\right) + \Delta FEH_{Jahr} \\ &= 6.000 + 6.720 + 0 + \Delta FEH_{Jahr} \\ &= 12.720 + \Delta FEH_{Jahr} \\ &= 12.720 + \overline{n}_{Fehlalarm,Jahr} * (\Delta INP + \Delta VEI + \Delta OPI) \\ &= 12.720 + \overline{n}_{Fehlalarm,Jahr} \\ &\qquad * (STM * ANM * ADI + KVM * DDV_{PMS} + STO * ADI) \\ &= 12.720 + \overline{n}_{Fehlalarm,Jahr} * (50 * 2 * 6 + 2{,}5 * 60 + 100 * 6) \\ &= 12.720 + 1.350 * \overline{n}_{Fehlalarm,Jahr}\end{aligned} \tag{46}$$

$$\begin{aligned}\boldsymbol{\Delta VER_{Jahr}} &= \overline{n}_{Korrektalarm,Jahr} * (\Delta VEZ + \Delta VEN) \\ &= \overline{n}_{Korrektalarm,Jahr} \\ &\qquad * \left(KVM * (DDV_{PMS} - DDV_{Kein\ PMS}) + KBM\right. \\ &\qquad \left. * (0 - DDB_{Kein\ PMS})\right) \\ &= \overline{n}_{Korrektalarm,Jahr} * \left(2{,}5 * (60 - 120) + KBM * (0 - 0)\right) \\ &= -150 * \overline{n}_{Korrektalarm,Jahr}\end{aligned} \tag{47}$$

$$\begin{aligned}\boldsymbol{\Delta REP_{Jahr}} &= \overline{n}_{Korrektalarm,Jahr} * (\Delta ETK + \Delta PKR + \Delta OPP) \\ &= \overline{n}_{Korrektalarm,Jahr} \\ &\qquad * \left((ETK_{PMS} - ETK_{Kein\ PMS}) + STM\right. \\ &\qquad * (ANR_{PMS} * ADR_{PMS} - ANR_{kein\ PMS} * ADR_{Kein\ PMS}) + STO \\ &\qquad \left. * (ADR_{PMS} - ADR_{Kein\ PMS})\right)\end{aligned} \tag{48}$$

$$= \overline{n}_{Korrektalarm,Jahr} * \left((200 - 50.000) + 50 * (2 * 8 - 24 * 4) + 100 * (8 - 24)\right)$$

$$= \overline{n}_{Korrektalarm,Jahr} * (-49.800 - 4000 - 1.600)$$

$$= -55.400 * \overline{n}_{Korrektalarm,Jahr}$$

Literaturverzeichnis

Accenture (2014): Big Success With Big Data: Executive Summary. https://www.accenture.com/sa-en/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Industries_14/Accenture-Big-Data-POV.pdf, Abruf am 2017-03-11.

Asthana, Anand N.; Khorana, Sangeeta (2013): Unlearning Machine Learning: The Challenge of Integrating Research in Business Applications. In: Middle-East Journal of Scientific Research, 15 (2), S. 266-271.

Breiman, Leo (1996): Bagging Predictors. In: Machine Learning, 24 (2), S. 123-140.

Breiman, Leo (2001): Random Forests. In: Machine Learning, 45 (1), S. 5-32.

Bundesministerium der Finanzen (2016): 1. Personalkosten in der Bundesverwaltung für Kostenberechnungen/WU 2. Sachkosten in der Bundesverwaltung für Kostenberechnungen/WU 3. Kalkulationszinssätze für WU;. http://www.bundesfinanzministerium.de/Content/DE/Standardartikel/Themen/Oeffentliche_Finanzen/Bundeshaushalt/personalkostensaetze.html, Abruf am 2017-02-18.

Buxmann, Peter (2001): Informationsmanagement in vernetzten Unternehmen: Wirtschaftlichkeit, Organisationsänderungen und der Erfolgsfaktor Zeit. Deutscher Universitätsverlag, Wiesbaden.

Carbonell, Jaime G.; Michalski, Ryszard S.; Mitchell, Tom M. (1983): An Overview of Machine Learning. In: Michalski, Ryszard S.; Carbonell, Jaime G.; Mitchell, Tom M. (Hrsg.): Machine Learning: An Artificial Intelligence Approach. Springer-Verlag, Berlin, Heidelberg, S. 3-23.

Caruana, Rich; Niculescu-Mizil, Alexandru (2006): An Empirical Comparison of Supervised Learning Algorithms. In: Cohen, William W.; Moore, Andrew (Hrsg.): Proceedings of the 23rd International Conference on Machine Learning. ACM, Pittsburgh, Pennsylvania, USA, S. 161-168.

Chapelle, Olivier; Scholkopf, Bernhard; Zien, Alexander (2006): Semi-Supervised Learning. MIT Press, Cambridge, London.

Cohen, William W. (1995): Fast Effective Rule Induction. In: Prieditis, Armand; Russell, Stuart J. (Hrsg.): Machine Learning Proceedings 1995. Morgan Kaufmann, Tahoe City, California, USA, S. 115-123.

DB Cargo (2015): Digitalisierung? Wir sind mittendrin! http://www.dbcargo.com/rail-deutschland-de/news_media/uebersicht_magazin/8977572/railways_15_01_seite_12.html, Abruf am 2017-03-11.

DB Cargo (2016): Zahlen & Fakten. https://www.dbcargo.com/rail-deutschland-de/unternehmen_db_cargo/ueber_dbcargo_de/zahlen_und_fakten.html, Abruf am 2017-03-11.

DB Cargo (2017): „ampulse“: Asset & Maintenance Digital Lab der DB - Hintergrundinformationen und Themen im Überblick. https://dk.dbcargo.com/file/rail-deutschland-en/12100012/HhrnXZ2Fgt66Wqv_X3BjqQ8csdo/13273904/data/Asset_Maintenance_Digital_Lab.pdf, Abruf am 2017-02-16.

DIN Deutsches Institut für Normung e. V. (2005): DIN EN 60300-3-3:2005-03 Zuverlässigkeitsmanagement – Teil 3-3: Anwendungsleitfaden – Lebenszykluskosten (IEC 60300-3-3:2004); Deutsche Fassung EN 60300-3-3:2004. Beuth Verlag, Berlin.

DIN Deutsches Institut für Normung e. V. (2014): DIN EN 60300-3-3:2014-09 (Entwurf) Zuverlässigkeitsmanagement – Teil 3-3: Anwendungsleitfaden – Lebenszykluskosten (IEC 56/1549/CD:2014). Beuth Verlag, Berlin.

Fawcett, Tom (2006): An introduction to ROC analysis. In: Pattern Recognition Letters, 27 (8), S. 861-874.

Fayyad, Usama M.; Piatetsky-Shapiro, Gregory; Smyth, Padhraic (1996): From Data Mining to Knowledge Discovery in Databases. In: AI Magazine, 17 (3), S. 37-54.

Flach, Peter (2012): Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press, Cambridge u.a.

Frank, Eibe (2016a): Class J48. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html, Abruf am 2017-01-12.

Frank, Eibe (2016b): Class REPTree. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/REPTree.html, Abruf am 2017-01-16.

Frank, Eibe; Hall, Mark A.; Witten, Ian H. (2016): The Weka Workbench - Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques (Fourth Edition). http://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf, Abruf am 2017-01-20.

Frank, Eibe; Trigg, Len (2016): Class AdaBoostM1. http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AdaBoostM1.html, Abruf am 2017-01-16.

Freund, Yoav; Schapire, Robert E. (1996): Experiments with a New Boosting Algorithm. In: Saitta, Lorenza (Hrsg.): Machine Learning: Proceedings of the Thirteenth International Conference. Morgan Kaufmann, Bari, Italien, S. 148-156.

Freund, Yoav; Schapire, Robert E. (1997): A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. In: Journal of Computer and System Sciences, 55 (1), S. 119-139.

Fürnkranz, Johannes; Widmer, Gerhard (1994): Incremental Reduced Error Pruning. In: Cohen, William W.; Hirsh, Haym (Hrsg.): Proceedings of the 11th International Conference on Machine Learning (ML 94). Morgan Kaufmann, New Brunswick, New Jersey, USA, S. 70-77.

García Márquez, Fausto Pedro; Lewis, Richard W.; Tobias, Andrew M.; Roberts, Clive (2008): Life cycle costs for railway condition monitoring. In: Transportation Research Part E: Logistics and Transportation Review, 44 (6), S. 1175-1187.

Gartner (2016): Top 10 Strategic Technology Trends for 2017. https://www.gartner.com/doc/3471559?srcId=1-6595640685, Abruf am 2017-03-11 (Login erforderlich).

General Electric; Accenture (2015): Industrial Internet Insights Report for 2015. https://www.accenture.com/t20150523T023646__w__/us-en/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Global/PDF/Dualpub_11/Accenture-Industrial-Internet-Insights-Report-2015.pdf, Abruf am 2017-03-11.

Hall, Mark; Frank, Eibe; Holmes, Geoffrey; Pfahringer, Bernhard; Reutemann, Peter; Witten, Ian H. (2009): The WEKA Data Mining Software: An Update. In: ACM SIGKDD Explorations Newsletter, 11 (1), S. 10-18.

Han, Jiawei; Kamber, Micheline; Pei, Jian (2012): Data Mining: Concepts and Techniques. 3. Aufl., Morgan Kaufmann, Amsterdam u.a.

Hartmann, Peter (2015): Mathematik für Informatiker: Ein praxisbezogenes Lehrbuch. 6. Aufl., Springer Vieweg, Wiesbaden.

Hashemian, H. M.; Bean, Wendell C. (2011): State-of-the-Art Predictive Maintenance Techniques*. In: IEEE Transactions on Instrumentation and Measurement, 60 (10), S. 3480-3492.

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2009): The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2. Aufl., Springer-Verlag, New York.

He, Haibo; Garcia, Edwardo A. (2009): Learning from Imbalanced Data. In: IEEE Transactions on Knowledge and Data Engineering, 21 (9), S. 1263-1284.

Heesen, Bernd (2010): Investitionsrechnung für Praktiker: Fallorientierte Darstellung der Verfahren und Berechnungen. 1. Aufl., Gabler Verlag, Wiesbaden.

Hewlett Packard Enterprise (2016): Sense Making in an IOT World: Sensor Data Analysis with Deep Learning. http://on-demand.gputechconf.com/gtc/2016/presentation/s6773-natalia-vassilieva-sensor-data-analysis.pdf, Abruf am 2017-03-11.

Hokstad, Per (1998): Life Cycle Cost Analysis in Railway Systems. SINTEF Industrial Management: Safety and Reliability, Trondheim.

Huang, Barbara F. F.; Boutros, Paul C. (2016): The parameter sensitivity of random forests. In: BMC Bioinformatics, 17 (1), S. 1-13 (331).

IDG Research Services (2016): Studie Internet of Things 2016. https://www.sap.com/germany/docs/download/2016/12/50280bc8-9b7c-0010-82c7-eda71af511fa.pdf, Abruf am 2017-03-11.

Inglis, Stuart (2016): Class SpreadSubsample. http://weka.sourceforge.net/doc.stable/weka/filters/supervised/instance/SpreadSubsample.html, Abruf am 2017-01-12.

Jardine, Andrew K. S.; Lin, Daming; Banjevic, Dragan (2006): A review on machinery diagnostics and prognostics implementing condition-based maintenance. In: Mechanical Systems and Signal Processing, 20 (7), S. 1483-1510.

Jonen, Andreas; Lingnau, Volker; Müller, Jochen; Müller, Paul (2004): Balanced IT-Decision-Card Ein Instrument für das Investitionscontrolling von IT-Projekten. In: Wirtschaftsinformatik, 46 (3), S. 196-203.

Kauschke, Sebastian (2014): Nutzung bahnbezogener Sensordaten zur Vorhersage von Wartungszyklen. Knowledge Engineering Group, Technische Universität Darmstadt, Diplomarbeit: 94 Seiten, http://www.ke.tu-darmstadt.de/lehre/arbeiten/diplom/2014/Kauschke_Sebastian.pdf.

Kauschke, Sebastian; Fürnkranz, Johannes; Janssen, Frederik (2016): Predicting Cargo Train Failures: A Machine Learning Approach for a Lightweight Prototype. In: Calders, Toon; Ceci, Michelangelo; Malerba, Donato (Hrsg.): Proceedings of the 19th International Conference on Discovery Science (DS 2016). Springer International Publishing, Bari, Italy, S. 151-166.

Kauschke, Sebastian; Janssen, Frederik; Schweizer, Immanuel (2015): On the Challenges of Real World Data in Predictive Maintenance Scenarios: A Railway Application. In: Bergmann, Ralph; Görg, Sebastian; Müller, Gilbert (Hrsg.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. Trier, Deutschland, S. 121-132.

KDnuggets (2014): What main methodology are you using for your analytics, data mining, or data science projects? Poll (Oct 2014). http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html, Abruf am 2017-01-10.

Kirkby, Richard (2016): Class Random Forest. http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html, Abruf am 2017-01-16.

Knowledge Engineering Group (o. J.-a): Predictive Maintenance in a Railway Scenario. http://www.ke.tu-darmstadt.de/projects/predictive-maintenance-for-train-engines, Abruf am 2017-03-11.

Knowledge Engineering Group (o. J.-b): Research. http://www.ke.tu-darmstadt.de/research, Abruf am 2017-03-11.

Krishnamurthy, Lakshman; Adler, Robert; Buonadonna, Phil; Chhabra, Jasmeet; Flanigan, Mick; Kushalnagar, Nandakishore; Nachman, Lama; Yarvis, Mark (2005): Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea. In: Redi, Jason K. (Hrsg.): Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems. ACM, San Diego, California, USA, S. 64-75.

Lavesson, Niklas; Davidsson, Paul (2006): Quantifying the Impact of Learning Algorithm Parameter Tuning. In: Cohn, Anthony (Hrsg.): Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06) Band 1, AAAI Press, Boston, Massachusetts, USA, S. 395-400.

Létourneau, Sylvain; Famili, Fazel; Matwin, Stan (1999): Data Mining to Predict Aircraft Component Replacement. In: IEEE Intelligent Systems and their Applications, 14 (6), S. 59-66.

Li, Hongfei; Parikh, Dhaivat; He, Qing; Qian, Buyue; Li, Zhiguo; Fang, Dongping; Hampapur, Arun (2014): Improving rail network velocity: A machine learning approach to predictive maintenance. In: Transportation Research Part C: Emerging Technologies, 45, S. 17-26.

Ling, Charles X.; Yang, Qiang; Wang, Jianning; Zhang, Shichao (2004): Decision Trees with Minimal Costs. In: Brodley, Carla (Hrsg.): Proceedings of the twenty-first International Conference on Machine Learning. ACM, Banff, Alberta, Kanada, S. 1-8 (69).

Lomax, Susan; Vadera, Sunil (2013): A Survey of Cost-Sensitive Decision Tree Induction Algorithms. In: ACM Computing Surveys, 45 (2), S. 1-35 (16).

May, Allan; McMillan, David; Thöns, Sebastian (2015): Economic analysis of condition monitoring systems for offshore wind turbine sub-systems. In: IET Renewable Power Generation, 9 (8), S. 900-907.

McKinsey Global Institute (2016): The Age of Analytics: Competing in a Data-Driven World. http://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Analytics/Our%20Insights/The%20age%20of%20analytics%20Competing%20in%20a%20data%20driven%20world/MGI-The-Age-of-Analytics-Full-report.ashx, Abruf am 2017-03-11.

Microsoft (2015): AzureML Team for Microsoft: Predictive Maintenance: Step 1 of 3, data preparation and feature engineering. https://gallery.cortanaintelligence.com/Experiment/Predictive-Maintenance-Step-1-of-3-data-preparation-and-feature-engineering-2, Abruf am 26-01-2017.

Mitchell, Tom M. (1997): Machine Learning. McGraw-Hill, Maidenhead.

Mitchell, Tom M. (1999): Machine Learning and Data Mining. In: Communications of the ACM, 42 (11), S. 30-36.

Mobley, R. Keith (2002): An Introduction to Predictive Maintenance. 2. Aufl., Butterworth-Heinemann, Amsterdam u.a.

Molina, María De Mar; Luna, José María; Romero, Cristobal; Ventura, Sebastián (2012): Meta-Learning Approach for Automatic Parameter Tuning: A Case Study with Educational Datasets. In: Yacef, Kalina; Zaïane, Osmar; Hershkovitz, Arnon; Yudelson, Michael; Stamper, John (Hrsg.): Proceedings of the 5th International Conference on Educational Data Mining. Chania, Griechenland, S. 180-183.

Moura, Márcio das Chagas; Zio, Enrico; Lins, Isis Didier; Droguett, Enrique (2011): Failure and reliability prediction by support vector machines regression of time series data. In: Reliability Engineering & System Safety, 96 (11), S. 1527-1534.

New South Wales Treasury (2004): Total Asset Management: Life Cycle Costing Guideline. Sydney.

Nowlan, F. Stanley; Heap, Howard F. (1978): Reliability-Centered Maintenance. Dolby Access Press.

Poggensee, Kay (2014): Investitionsrechnung: Grundlagen – Aufgaben – Lösungen. 3. Aufl., Springer Gabler, Wiesbaden.

Prytz, Rune; Nowaczyk, Slawomir; Rögnvaldsson, Thorsteinn; Byttner, Stefan (2013): Analysis of Truck Compressor Failures Based on Logged Vehicle Data. In: Stahlbock, Robert; Weiss, Gary M. (Hrsg.): Proceedings of the 9th International Conference on Data Mining (DMIN). CSREA Press, Las Vegas, Nevada, USA, S. 118-124.

Prytz, Rune; Nowaczyk, Sławomir; Rögnvaldsson, Thorsteinn; Byttner, Stefan (2015): Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data. In: Engineering Applications of Artificial Intelligence, 41, S. 139-150.

Quinlan, J. Ross (1986): Induction of Decision Trees. In: Machine Learning, 1 (1), S. 81-106.

Quinlan, J. Ross (1993): C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, Kalifornien.

Quinlan, J. Ross (1999): Simplifying decision trees. In: International Journal of Human-Computer Studies, 51 (2), S. 497-510.

Roemer, Michael J; Byington, Carl S; Kacprzynski, Gregory J; Vachtsevanos, George (2006): An Overview of Selected Prognostic Technologies With Application to Engine Health Management. In: ASME Turbo Expo 2006: Power for Land, Sea, and Air. Band 2, American Society of Mechanical Engineers, Barcelona, Spain, S. 707-715.

Russell, Stuart; Norvig, Peter (2012): Künstliche Intelligenz: Ein moderner Ansatz. 3. Aufl., Pearson, München u.a.

Salfner, Felix; Malek, Miroslaw (2007): Using Hidden Semi-Markov Models for Effective Online Failure Prediction. In: Huai, Jinpeng; Baldoni, Roberto; Yen, I·Leng (Hrsg.): Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007). IEEE Computer Society, Beking, China, S. 161-174.

Sammut, Claude; Webb, Geoffrey I. (2010): Encyclopedia of Machine Learning. 1. Aufl., Springer Science & Business Media, New York.

Samuel, Arthur L. (1959): Some Studies in Machine Learning Using the Game of Checkers. In: IBM Journal of Research and Development, 3 (3), S. 210-229.

Schuld, Maria; Sinayskiy, Ilya; Petruccione, Francesco (2015): An introduction to quantum machine learning. In: Contemporary Physics, 56 (2), S. 172-185.

scikit-learn (2016): RandomForestClassifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, Abruf am 2017-01-16.

Shannon, Claude E. (1948): A Mathematical Theory of Communication. In: Bell System Technical Journal, 27 (3), S. 379-423.

Slurm (2016): Documentation. https://slurm.schedmd.com/, Abruf am 2017-03-17.

Sokolova, Marina; Lapalme, Guy (2009): A systematic analysis of performance measures for classification tasks. In: Information Processing & Management, 45 (4), S. 427-437.

Susto, Gian Antonio; Beghi, Alessandro; De Luca, Cristina (2012): A Predictive Maintenance System for Epitaxy Processes Based on Filtering and Prediction Techniques. In: IEEE Transactions on Semiconductor Manufacturing, 25 (4), S. 638-649.

Susto, Gian Antonio; Schirru, Andrea; Pampuri, Simone; McLoone, Seán; Beghi, Alessandro (2015): Machine Learning for Predictive Maintenance: A Multiple Classifier Approach. In: IEEE Transactions on Industrial Informatics, 11 (3), S. 812-820.

Varma, Sudhir; Simon, Richard (2006): Bias in error estimation when using cross-validation for model selection. In: BMC Bioinformatics, 7 (1), S. 1-8 (91).

Witten, Ian H.; Frank, Eibe (2005): Data Mining: Practical Machine Learning Tools and Techniques. 2. Aufl., Morgan Kaufmann, Amsterdam u.a.

Xu, Xin; Frank, Eibe (2016): Class JRip. http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/JRip.html, Abruf am 2017-01-08.

Yam, Richard C. M.; Tse, Peter W.; Li, L.; Tu, P. (2001): Intelligent Predictive Decision Support System for Condition-Based Maintenance. In: The International Journal of Advanced Manufacturing Technology, 17 (5), S. 383-391.

Yan, Jihong; Koç, Muammer; Lee, Jay (2004): A prognostic algorithm for machine performance assessment and its application. In: Production Planning & Control, 15 (8), S. 796-801.

Yang, Chunsheng; Létourneau, Sylvain (2005): Learning to Predict Train Wheel Failures. In: Grossman, Robert L.; Bayardo, Roberto; Bennett, Kristin; Vaidya, Jaideep (Hrsg.): Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, Chicago, Illinois, USA, S. 516-525.

Yann-Chang, Huang (2003): Evolving Neural Nets for Fault Diagnosis of Power Transformers. In: IEEE Transactions on Power Delivery, 18 (3), S. 843-848.

Yilboga, Halis; Eker, Ömer F.; Güçlü, Adem; Camci, Fatih (2010): Failure Prediction on Railway Turnouts Using Time Delay Neural Networks. In: Pedrycz, Witold; Ruspini, Enrique; Lecce, Vincenzo Di; Micheli-Tzanakou, Evangelia (Hrsg.): 2010 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications. IEEE, Taranto, Italien, S. 134-137.

Zhang, Di; Hu, Hao; Roberts, Clive; Dai, Lei (2017): Developing a life cycle cost model for real-time condition monitoring in railways under uncertainty. In: Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit, 231 (1), S. 111-121.