BERGISCHE UNIVERSITÄT WUPPERTAL

Fakultät für Maschinenbau und Sicherheitstechnik

Fachgebiet Verkehrssicherheit und Zuverlässigkeit

Masterthesis

Zuverlässigkeitstechnik in der Industrie 4.0 und prädiktive Instandhaltung: Automatische Klassifizierung von Betriebszuständen durch maschinelles Lernen

vorgelegt von

Ekinsu Cin

Matrikel-Nr.: 1837831

Studiengang: Qualitätsingenieurwesen

Betreuer: Jun.-Prof. Dr. Antoine Tordeux

Zweitprüfer: Raphael Korbmacher

Ausgabe:18.04.2022

Abgabe:17.09.2022

Eidesstattliche Erklärung

Name, Vorname: Cin, Ekinsu

Erklärung

Hiermit, erkläre ich, dass ich die von mir eingereichte Masterarbeit selbstständig verfasst und keine andere als die angegebene Quelle und Hilfsmittel benutzt sowie Stellen der Abschlussarbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich bin damit einverstanden, dass die Arbeit durch Dritte eingesehen und unter Wahrung urheberrechtlicher Grundsätze zitiert werden darf.

Datum und Ort: ______________________________

Unterschrift: ______________________________

Inhaltsverzeichnis

Zusammenfassung

Prädiktive Instandhaltung ist an Bedeutung gewinnendes Thema für die Industrie 4.0, da diese Instandhaltungsstrategie die Ausfälle der Maschinen noch vor ihrem Eintreten mit verschiedenen Methoden erkennen kann.

In dem Theorieteil dieser Masterarbeit wird die Industrie 4.0 und ihre Zuverlässigkeitstechnik erläutert und es werden insbesondere die Instandhaltungsstrategien und die verschiedenen Methoden für die prädiktive Instandhaltung erörtert. In dem Analyseteil der Arbeit wird ein Datensatz mit verschiedenen Sensordaten anhand der Programmiersprache R untersucht, um die Methoden der prädiktiven Instandhaltung zu vergleichen. Der Schwerpunkt der Implementierung in die Programmierumgebung liegt auf der Klassifizierung des Betriebszustands der Maschine. Für diese Klassifizierung werden folgende statistische Methoden und die Algorithmen des maschinellen Lernens verwendet: Logistische Regression, Lineare Diskriminantfunktion, Naive Bayes, Decision Tree, Neural Network, Random Forest und Support Vector Machine. Erst wird untersucht, welche Methoden die Betriebszustände mit geringen Fehlern klassifizieren können. Die ersten Ergebnisse zeigen, dass die Algorithmen des maschinellen Lernens die Zustände mit niedrigeren Fehlerraten klassifizieren als statistische Methoden. Für optimalere Ergebnisse werden für Neural Network und Support Vector Machine neue Hyperparameter eingesetzt. Mit den neuen Parametern klassifizieren diese Algorithmen die Zustände mit geringen Fehlerraten, Neural Network weist für den betriebsbereiten Zustand 0,58 % Fehler auf und die Support Vector Machine 0,60 % Fehler. Die Fehlerquote für die Klassifizierung der ausgefallenen Zustände folgt mit jeweils 35,9 % und 41,3 %. Random Forest, ein weiterer Algorithmus des maschinellen Lernens, klassifiziert die Zustände ebenfalls mit niedrigen Fehlerraten.

Summary

Predictive maintenance is an increasingly important topic for Industry 4.0, as this maintenance strategy can detect machine failures even before they occur using various methods.

In the theory part of this master thesis, Industry 4.0 and its reliability technology are explained and additionally the maintenance strategies and different methods for predictive maintenance are discussed. In the analysis part of the thesis, a dataset with different sensor data is examined using the R programming language to compare the predictive maintenance methods. The focus of the implementation in the programming environment is on the classification of the operating state of the machine. The following statistical methods and machine learning algorithms are used for this classification: Logistic Regression, Linear Discriminant Analysis, Naive Bayes, Decision Tree, Neural Network, Random Forest and Support Vector Machine. The first step is to investigate which methods can classify the operating states with low error. The initial results show that the machine learning algorithms classify the states with lower error rates than statistical methods do. For the more optimal results, new hyper parameters are used for Neural Network and Support Vector Machine. With the new parameters, these algorithms classify the states with low error rates, Neural Network has 0.58% error for the ready state and Support Vector Machine has 0.60% error. The error rate for the classification of the failed states follows with 35.9% and 41.3% respectively. Random Forest, another machine learning algorithm, also classifies the states with low error rates.

Abkürzungsverzeichnis

ANN	Artifical Neural Network
CBM	Condition-Based Maintenance
CPS	Cyber- Physical Systems
DL	Deep Learning
DNN	Deep Neural Network
DT	Decision Tree
IoT	Internet of Things
LinR	Linear Regression
LDA	Linear Discriminant Analysis
LR	Logistic Regression
ML	Machine Learning
MSE	Mean Squared Error
NB	Naive Bayes
NN	Neural Network
PCA	Principal Components Analysis
PC	Principal Component
PdM	Predictive Maintenance
PHM	Prognostics and Health Management
RF	Random Forest
RFID	Radio-Frequency Identification
RUL	Remaining-useful-life
SR	Symbolic Regression
SVM	Support Vector Machine

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Physikalische, menschliche und digitale Welten haben einen großen Einfluss auf die Industrie 4.0. Mit diesen drei Einflüssen hat sich die Industrie bis jetzt viel verändert [7]. Wenn über das Thema Industrie 4.0 geredet wird, tauchen zwangsläufig die Begriffe Cyber-Physical Systeme (CPS), Internet of Things (IoT), Sensorik und Big Data auf. Diese Begriffe verändern die Art und Weise, wie man die Produkte und Dienstleistungen entwickeln, herstellen und anbieten kann. Industrie 4.0 bietet Möglichkeiten, komplexe Lieferketten und Vertriebsnetze effizienter, smarter, schneller, flexibler und widerstandsfähiger zu gestalten. [5] Dabei ist die Zuverlässigkeit ein wesentlicher Aspekt, um zielführende Entscheidungen zu treffen [2].

Für die smarte Produktion ist die Instandhaltung der Maschine wichtig. Prädiktive Instandhaltung überwacht den Zustand der Maschine und kann von selbstständigen Maschinen durch CPS, Internet of Things und Big Data Analyse realisiert werden. [11] Prädiktive Instandhaltung basiert auf einem idealerweise sensorisch durchgeführten Condition Monitoring, das eine kontinuierliche Überwachung relevanter Maschinenparameter wie z.B. Vibration und Temperatur ermöglicht [3].

Seit vielen Jahren werden verschiedene Veröffentlichungen über Industrie 4.0 publiziert. Im „Google Scholar“ werden insgesamt 220.000 Publikationen gefunden. Die Suche wurde mit dem Stichwort „Industry 4.0“ am 18.08.2022 durchgeführt. In der Abbildung 1 kann gesehen werden, wie viele Arbeiten in den letzten zehn Jahren veröffentlicht wurden. Besonders in den Jahren 2018 und 2020 wurden viele Veröffentlichungen getätigt und dieser Trend ist bis 2020 gestiegen. Jedoch ist die Anzahl in den letzten zwei Jahren gesunken.

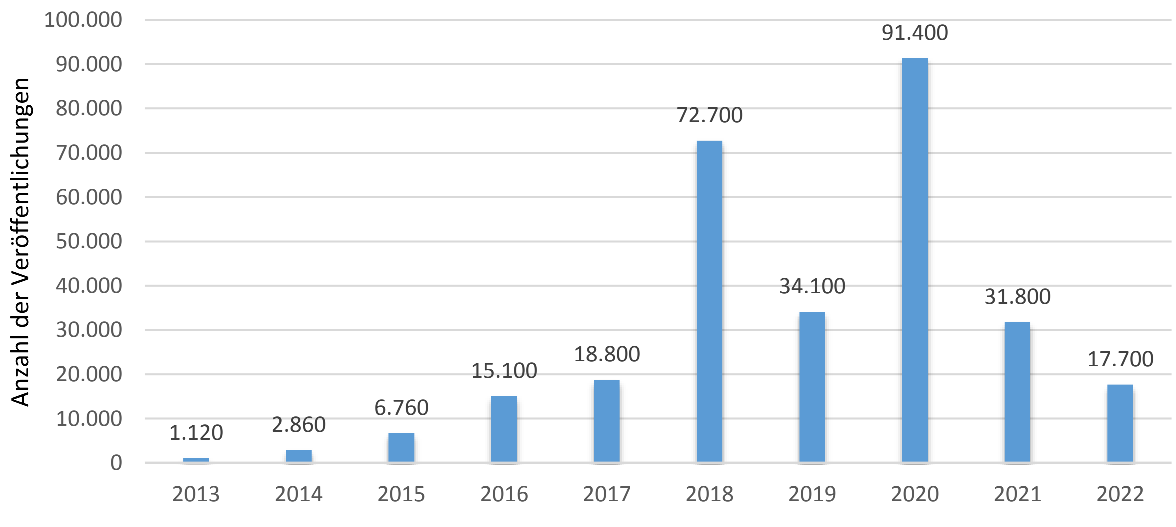

Abbildung 1: Anzahl der Veröffentlichungen mit dem Stichwort: „Industry 4.0“ Zugriff am 18.08.2022

Damit die Trends der prädiktiven Instandhaltung und Condition Monitoring in der Industrie 4.0 gesehen werden kann, wurden Veröffentlichungen im Google Scholar mit den Stichworten „Industry 4.0“ and „Predictive Maintenance“ und „Industry 4.0“ and „Predictive Maintenance“ and „Condition Monitoring“ gefunden und in der Abbildung 2 und Abbildung 3 gezeigt. Wie in der Abbildung 2 ersichtlich ist, zieht das Thema „prädiktive Instandhaltung“ die Aufmerksamkeit der Forscherinnen und Forscher auf sich. Obwohl in den Jahren 2013-2014 wenige Artikel veröffentlicht wurden, ist die Anzahl der Veröffentlichungen ab 2017 enorm gestiegen. Die Anzahl der Veröffentlichungen, die in den Jahren 2020 und 2021 publiziert wurden, zeigen, wie intensiv diese beiden Themen in den letzten Jahren erforscht wurden. Im Jahr 2019 wurden im Vergleich zu vergangenen Jahren wenige Veröffentlichungen über die Industrie 4.0 gefunden. Aber das Interesse an Industrie 4.0 und prädiktiver Instandhaltung hat sich trotzdem verstärkt, wie Abbildungen 1 und 2 widerspiegeln.

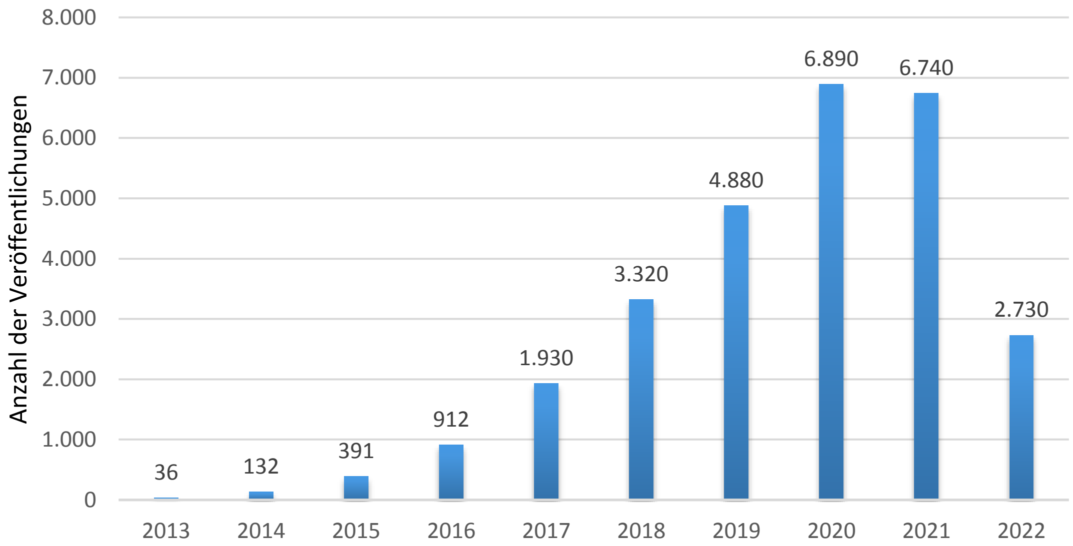

Abbildung 2: Anzahl der Veröffentlichungen mit den Stichworten: „Industry 4.0“ AND „Predictive Maintenance“ Zugriff am 18.08.2022

Wie schon erwähnt wurde, ist nicht nur prädiktive Instandhaltung sondern auch Condition Monitoring eines der Themen, die in den letzten Jahren erforscht wurde. In der Abbildung 3 ist zu sehen, wie der Trend sich mit der Zeit ändert. Die Suche wurde mit den Stichworten „Industry 4.0“ and „Predictive Maintenance“ and „Condition Monitoring“ durchgeführt, um den Zusammenhang zwischen prädiktiver Instandhaltung und Condition Monitoring zu ermitteln. Wie erwartet, hat Abbildung 3 eine ähnliche Darstellung wie Abbildung 2, weil Condition Monitoring auch ein wichtiger Begriff für die prädiktive Instandhaltung ist.

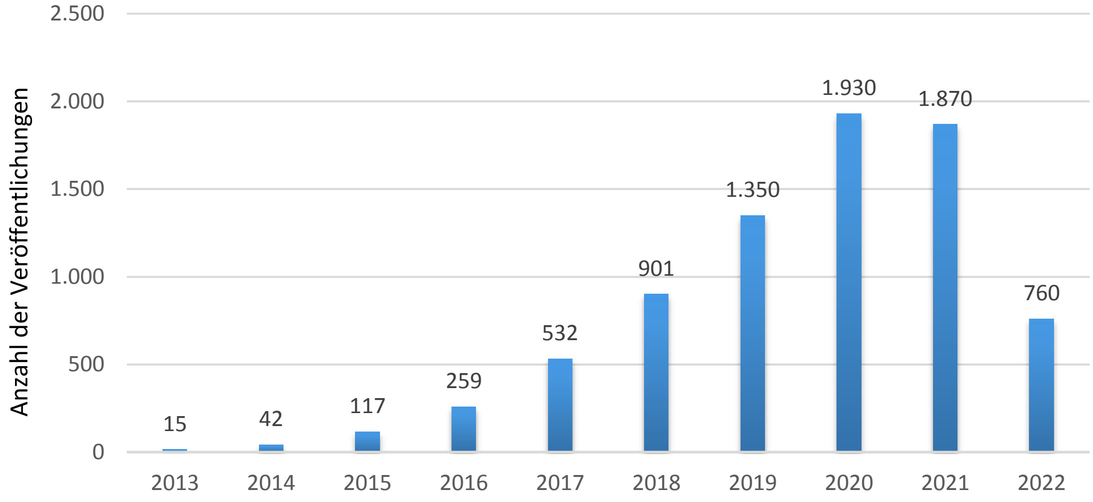

Abbildung 3: Anzahl der Veröffentlichungen mit den Stichworten: „Industry 4.0“ AND „Predictive Maintenance“ AND „Condition Monitoring“ Zugriff am 18.08.2022

1.1 Problemstellung

Mit der Entwicklung von Industrie 4.0 werden täglich viele verschiedene Daten durch Sensoren gesammelt. Die gesammelten Daten informieren uns besonders über die Zustände der Maschinen, die sich in der Produktion befinden. Der Zustand der Maschine ist wichtig für die Instandhaltungsstrategien. Besonders ist die prädiktive Instandhaltung in den letzten Jahren signifikant geworden, weil diese Instandhaltungsstrategien den Ausfall der Maschine prognostizieren und eine Wartung vor dem Ausfall der Maschine planen können. Deswegen sollten die durch Sensoren gesammelten Daten analysiert und ausgewertet werden. Das Problem ist, dass es noch nicht klar ist, welche Methode für diese Instandhaltung bessere Prognose erzielt. Deswegen werden in den letzten Jahren Methoden für die prädiktive Instandhaltung untersucht. Die statistische Methode und die Algorithmen des maschinellen Lernens können für die Zustandsprognose und Klassifizierung der Betriebszustände genutzt werden. Hierbei ist zu beachten, dass die Methoden, die die wenigsten Fehler darlegen, die Zustände vorhersagen oder klassifizieren sollten. Eine weitere Unklarheit ist, mit welchen Parametern die Algorithmen des maschinellen Lernens die beste Prognose und Klassifizierung aufweisen.

1.2 Zielsetzung

Das Ziel dieser Masterarbeit ist es, die Zuverlässigkeitstechnik, die Instandhaltungsstrategien und besonders die prädiktive Instandhaltung im Kontext von Industrie 4.0 zu betrachten. Die wichtigen Begriffe für die Industrie 4.0, Instandhaltungsstrategien und die Methoden für die datengesteuerten prädiktiven Instandhaltungsmodelle werden erklärt, damit diese Informationen mit dem Anwendungsbeispiel in Betracht gezogen werden können. Diese Arbeit befasst sich mit dem Ziel, die verschiedenen Methoden der prädiktiven Instandhaltung zu vergleichen und eine Entscheidung zu treffen, mit welchen Methoden eine bessere Klassifizierung des Betriebszustandes der Maschine erreicht werden kann. Dazu wird ein Datensatz von einer Maschine verwendet und der Zustand dieser Maschine mit verschiedenen Methoden der prädiktiven Instandhaltung klassifiziert. Die statischen Methoden und Algorithmen des maschinellen Lernens werden hier benutzt, um herauszufinden, wie gut diese Methoden die Zustände klassifizieren. Für diese Entscheidung wird die Fehlerrate der Methoden nach der Klassifizierung betrachtet. Neben der Entscheidung der Methoden wird auch festgestellt, mit welchen Parametern die Algorithmen des maschinellen Lernens bessere Ergebnisse bei der Klassifizierung ermitteln.

1.3 Aufbau der Arbeit

Diese Arbeit besteht aus fünf Kapiteln. Der grundlegende Aufbau der Masterarbeit ist in der Abbildung 4 zu zusehen.

Einleitung

Stand der Technik

- Industrie 4.0
- Zuverlässigkeitstechnik in der Industrie 4.0

Instandhaltungsstrategien

- Prädiktive Instandhaltung
- Condition Monitoring und die Modelle zur Fehlerprognose in der prädiktiven Instandhaltung
- Methoden für die datengesteuerten prädiktiven Instandhaltung-Modelle

Anwendungsbeispiel für die automatische Klassifizierung von Betriebszuständen

- Datenquelle und Programmiersprache
- Vorbereitung der Daten für die Analyse
- Klassifizierung von Betriebszuständen
- Auswertung der Ergebnisse

Fazit

Abbildung 4: Überblick des Aufbaus der Masterarbeit

Nach der Einleitung folgt das Kapitel „Stand der Technik“. Hier werden Industrie 4.0 und seine wichtigen Begriffe dargestellt, die Cyber- Physical Systems, Sensoren, Big Data und Internet of Things beinhalten. Nach der Erklärung der Begriffe werden die Zuverlässigkeitstechnik in der Industrie 4.0 und ihre Chancen und Herausforderungen erkundet.

Nach der Ermittlung der grundlegenden Informationen werden verschiedene Instandhaltungsstrategien erklärt. Weil die prädiktive Instandhaltung für die Industrie 4.0 essenziell ist, wird sowohl diese Instandhaltungsstrategie als auch Condition Monitoring erläutert. Die verschiedenen Modelle zur Fehlerprognose in der prädiktiven Instandhaltung werden auch in diesem Kapitel geschildert. Das wichtigste Modell für die prädiktive Instandhaltung ist das datengesteuerte Modell. Deswegen werden die Methoden für diese Modelle in statistische Modelle und Maschine Learning Methoden unterteilt und erklärt. Diese Methoden sind für die Zustandsprognose oder Klassifizierung von Bedeutung.

Ein Anwendungsbeispiel für die automatische Klassifizierung von Betriebszuständen wird in Kapitel 4 erörtert. Zu diesem Zweck werden erst die Datenquelle und die Anwendung der Programmiersprache R erklärt und dann die Daten für die Analyse vorbereitet. Nach der Vorbereitung der Daten werden die Betriebszustände mit verschiedenen Methoden klassifiziert. Die Methoden sind in statistische Modelle und Algorithmen des maschinellen Lernens unterteilt. Nach dem ersten Ergebnis wird diskutiert, ob die Algorithmen mit den verschiedenen Parametern eine bessere Klassifizierung erreichen können. Deswegen werden für zwei Algorithmen verschiedene Hyperparameter gesucht und untersucht, um herauszufinden, mit welchen Parametern die Algorithmen durchgeführt werden sollten. Nach der Eignung der Parameter werden die Endergebnisse mit den ersten Ergebnissen verglichen und die Änderungen werden erforscht. Zum Abschluss werden eine Beurteilung über die Ergebnisse sowie ein Ausblick auf die möglichen weiteren Studien betrachtet.

2. Stand der Technik

2.1 Industrie 4.0

Im achtzehnten Jahrhundert hat die erste industrielle Revolution mit der Nutzung der Dampfkraft angefangen. Elektrische Energie und das Fließband für die Massenproduktion führten Anfang des zwanzigsten Jahrhunderts die zweite industrielle Revolution an. In der dritten industriellen Revolution wurden Computer und Informationssysteme in die Produktion integriert, mit diesen Technologien wurden verschiedene computergestützte Systeme hergestellt. Infolgedessen wurden die Maschinen automatisiert, wodurch die Arbeitsbelastung reduziert, die Geschwindigkeit und die Genauigkeit erhöht und in einigen Fällen der Mensch vollständig ersetzt wurde. Die vierte industrielle Revolution beinhaltet nun den umfassenden Einsatz von Automatisierung und Datenaustausch in Fertigungsumgebungen. Das neue System nutzt fortschrittliche Technologien wie Cyber-Physical Systems, Internet of Things, und Advanced Analytics sowie Cloud Computing. Diese Revolution steht für smarte Fabrik, smarte Produktion und automatisierte Fertigung. [7,15] Abbildung 5 zeigt die industrielle Transformation.

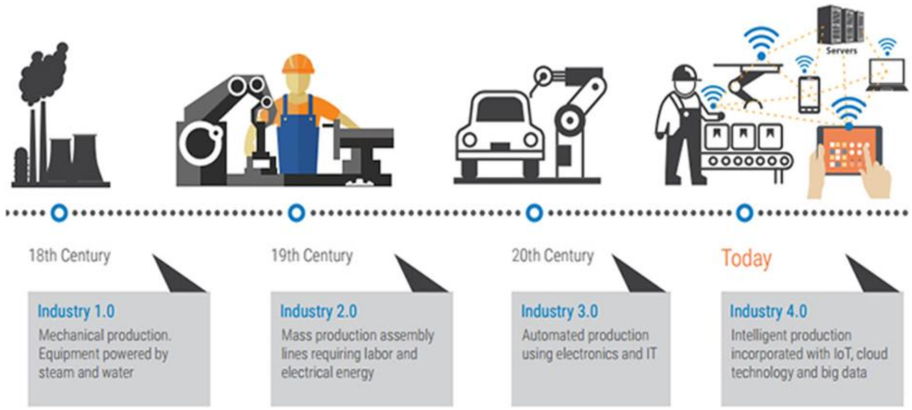

Abbildung 5: industrielle Revolution [10, S.133]

Die Produktion in der Industrie 4.0 kann mit Maschinen verglichen werden, die Dienstleistungen erbringen und Informationen mit Produkten in Echtzeit austauschen. Es gibt auch zusätzliche Funktionen in der Industrie 4.0 gemäß [10] und [8]:

- die Systemüberwachung und –diagnose wird erleichtert,
- durch ressourcensparendes Verhalten ist das System umweltfreundlicher und nachhaltiger,
- die Systeme sind effizienter,
- Wertschöpfungsprozesse sind anpassungsfähiger,
- Systeme reagieren auf Störungen flexibler.

Dieses Verständnis unterstützt die Idee, bereits existierende Black Factories bzw. „smart“ Fabriken zu generieren. Diese „smarten“ Fabriken betrachten den Produktions- und Herstellungsprozess aus einer neuen Perspektive. Smarte Fabriken sind Fabriken, in denen das Licht gelöscht wird und die Fertigungsumgebungen mit vollautomatischen Systemen ausgestattet sind und keine Anwesenheit von Menschen erfordern. Um eine smarte Fabrik aufzubauen, werden neue Technologien gebraucht. Diese Technologien beinhalten mehrere Software, fortschrittliche kollaborative Robotik, modulare und anpassbare Konfigurationen, Hochgeschwindigkeitssysteme für die Datenübertragung und andere. Es wird auch „smarte“ Lieferketten, ein smartes Wartungssystem, smarte Arbeit und so weiter gebraucht, damit ein vollständig smartes System erhalten wird. Natürlich werden die Produkte smarter, um den Funktions- und Nutzungsanforderungen gerecht zu werden. [10,7]

Die Umsetzung von Industrie 4.0 muss jedoch einige Grundanforderungen der produzierenden Industrie erfüllen. Anforderungen sind gemäß [7]:

- Unternehmensintegration und Interoperabilität
- Verteilte Organisation
- Modellbasierte Überwachung und Steuerung
- Heterogene Systeme und Umgebungen
- Offene und dynamische Struktur
- Kooperation und Zusammenarbeit
- Integration und Interoperabilität des Menschen mit Soft- und Hardwaresystemen
- Agilität, Skalierbarkeit und Fehlertoleranz
- Interdependente Netzwerke

Um die Industrie 4.0 besser zu verstehen, werden die Begriffe Cyber- Physikal Systeme, Big Data, Sensoren und Internet of Things in den nächsten Unterkapitel erläutert.

2.1.1 Cyber- Physical Systems (Cyber-Physische Systeme)

Moderne Industriesysteme sind komplexe Systeme, die physikalische, Software- und Netzwerkkomponenten in sogenannte Cyber-Physical Systems (CPS) integrieren [2]. CPS fördert die enge Verbindung und Koordination zwischen physischen Elementen und Computersoftware, während es Datenzugangs- und Datenverarbeitungsdienste bereitstellt und nutzt [7]. Das essenzielle Merkmal dieser Systeme ist, dass sie auf jedes generierte Feedback reagieren können. CPS sind die wesentlichen Bestandteile von Industrie 4.0 Implementierungen. [10] Es gibt verschiedene Vorteile, wenn Cyber-Physical Systems in der Produktion genutzt wird.

Die Vorteile des CPS sind gemäß [7] und [10]:

- Optimierung der Produktionsprozesse
- optimierte Produktanpassung,
- Ressource-effiziente Produktion,
- menschzentrierte Produktionsprozesse
- Prozess Monitoring
- Integration der verschiedenen Disziplinen in verschiedenen Domänen
- Konfiguration, Bereitstellung und Außerbetriebnahme in Echtzeit
- Selbstverhalten und Entscheidungsfindung.

Um ein CPS zu entwickeln, gibt es drei Phasen. Die erste Phase beinhaltet die Identifikationstechnologien wie RFID-Etikette, die eine eindeutige Identifikation ermöglichen. Die Speicherung und Analyse sollte als Hauptfunktion bereitgestellt werden. In der zweiten Phase gibt es Sensoren und Aktuators, die nur begrenzte Funktionen haben. In der dritten Phase werden die Daten gespeichert und analysiert. Das CPS ist netzwerkfähig und mit vielen Sensoren und Aktoren eingerichtet. [10]

Die Abbildung 6 nach [9] zeigt ein Aufbau eines CPS für die Intralogistik. In der physischen Welt werden die Daten von Physischen Objekten mit Sensoren gesammelt. Dann werden die Daten über Kommunikationsnetzwerke zur Speicherung oder Verarbeitung gesendet. Die Informationen können in der Enterprise-Ebene visualisiert und gesteuert werden. Nach der Steuerung der Informationen werden die Befehle zu den Aktoren gesendet. Die gesamte Kommunikation erfolgt über lokales Netzwerk (LAN) und Computer Netzwerk (Internet).

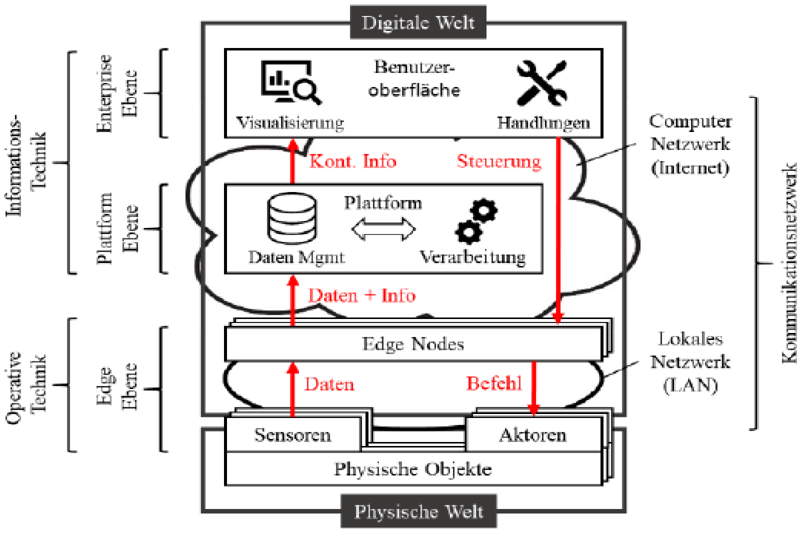

Abbildung 6: Aufbau eines Cyber-Physical Systems für die Intralogistik [9, S.7]

2.1.2 Sensoren

Ein Sensor ist nach [15] ein Gerät, das Inputstimulus erkennt. Die Inputstimuli können eine beliebige Größe, Eigenschaft oder Bedienung aus der physischen Umgebung sein und auf messbare digitale Signale reagieren. Der Inputstimulus kann beispielweise Druck, Kraft, Licht, Wärme, Bewegung sein. Der Output ist normalerweise eine elektrische Form eines Signals, wie Spannung, Strom, Frequenz usw., das in eine lesbare Anzeige umgewandelt wird.

Die Umsetzung von Industrie 4.0 basiert auf Sensorik. Um eine effektive Entscheidung zu treffen, können Sensoren die Daten erfassen und analysieren. Mit Hilfe der Sensoren ist es für die Automatisierung möglich, in der Produktionslinie sich selbst zu optimieren. Durch die vereinfachte Integration und Analyse sind Smart Sensoren signifikant beim Design der Internet of Thing Technologien für die Produktion. [15] Smart Sensoren ermöglichen eine Selbstidentifizierung und Eigendiagnose. Die Entwicklung der Sensoren ist auch mit der Entwicklung der Industrialisierung verknüpft. Ähnlich wie bei Industrie 4.0 ist auch Sensor 4.0 zu nennen. Mechanischer Messwertaufnehmer wird als Sensor 1.0, elektrische Sensoren werden als Sensor 2.0, elektronische Sensoren werden als Sensor 3.0 und Smart Sensoren werden als Sensor 4.0 bezeichnet. Größere Zuverlässigkeit und höhere Genauigkeit, bessere Vernetzung und Informationsübermittlung und flexible Konfigurierbarkeit werden von Smarte Sensoren erwartet. [14]

2.1.3 Big Data

Während des Lebens der Produkte werden verschiedene Daten aus verschiedenen Ebenen wie über Rohmaterial, Qualitätskontrolle, Lieferung gesammelt und analysiert. Im Gegensatz zur Vergangenheit können in der heutigen Zeit die Daten mit Hilfe der Digitalisierung durch Computerprozesssensoren gesammelt, gespeichert und dadurch eine algorithmische Verarbeitung mit schnellen Zugriffsgeschwindigkeiten durchgeführt werden. Damit können die strukturierten Datenbestände, die sehr groß und vielfältig sind, für Aktionen, Prognosen und Planungen sehr schnell analysiert werden. Diese Prozesse fallen in den Bereich der Big Data. Die Rolle der Big Data ist wichtig für die Industrie 4.0, weil die Daten Unternehmen flexibler und leistungsfähiger beim Marktwettbewerb machen können. [7,8]

Die gesammelte Big Data soll verarbeitet und angewendet werden. Die Daten bestehen aus verschiedenen Arten von Parametern mit verschiedenen Formen, wie z.B. Bild, elektronisches Signal oder Video. Die Rohdaten sind nutzlos, deswegen sollten die Rohdaten in spezifische Informationsinhalte und Kontexte umgewandelt werden. Daher gibt es verschiedene Methoden wie Deep Learning, Cloud Computing und neuronale Netze. [7]

2.1.4 Internet of Things

Laut [10] ist Internet of Things (IoT) ein Netzwerk von physischen Geräten, Fahrzeugen, Gebäuden und anderen Elementen, die in Elektronik, Software, Sensoren, Aktoren und Netzwerkverbindungen eingebettet sind und es diesen Objekten ermöglichen, Daten zu sammeln und auszutauschen. Das IoT ermöglicht die Fernerkundung oder die Kontrolle von Objekten über die bestehende Netzwerkinfrastruktur und schafft Möglichkeiten für eine direktere Integration der physischen Welt in basierte Systeme. [10]

IoT und Industrie 4.0 haben ähnliche Konzepte. Effektivität und Produktivität in der Industrie 4.0 sind mit dem Einsatz von IoT Technologien gestiegen. Damit wurden auch neue Strategien und Optimierungen erschaffen. Es liegt in den Händen der menschlichen Intelligenz und des Wissens, sowohl Industrie 4.0 als auch IoT auf die effizienteste und effektivste Weise zu integrieren. [10]

2.2 Zuverlässigkeitstechnik in der Industrie 4.0

Zuverlässigkeitstechnik basiert auf mathematischen Prinzipien, Wahrscheinlichkeitstheorie und Statistik, damit die funktionellen Probleme in Komponenten und Systemen systematisch und konsequent mit dem Ziel eines zuverlässigen Designs analysiert werden können [1]. Zuverlässigkeit ist die Wahrscheinlichkeit, dass ein technisches System für einen bestimmten Zeitraum in einem bestimmten Ort korrekt funktioniert [2]. Die Zuverlässigkeit eines Produktionssystems beeinflusst verschiedene Effizienzkennzahlen, wie Produktkosten, Wartungskosten und Ersatzteilkosten. Eine geringe Zuverlässigkeit in einem Produktionssystem bedeutet nicht nur kostspielige Reparatur und kostspieligen Austausch sondern auch eine verringerte Produktion und einen verringerten Gewinn, weil Maschinen häufig ausfallen und der Produktionsprozess häufig unterbricht. [13]

Die Zuverlässigkeit der Ausstattung spiegelt die Mission wider, die Anforderungen und die erwartete Leistung des Produktionssystems zu erfüllen. Ausstattungen können je nach verwendetem Verfahren ausfallen. Die Märkte haben verschiedene Anforderungen wie z.B. Anforderungen an hohe Zuverlässigkeit und Haltbarkeit. Um den Anforderungen gerecht zu werden, ist es notwendig, das ganze System mit dem Design, der Produktion und den Anwendungen zu analysieren. Daher ist es erforderlich, die Zuverlässigkeit über den gesamten Produktlebenszyklus zu maximieren. Wenn es in dem Produktionssystem Unsicherheiten, wie Gerätezustand und Produktionsstrategien gibt und die Entscheidungen mit diesen Unsicherheiten verbunden sind, können Zuverlässigkeitsmodelle verwendet werden, um die Schlussfolgerungen zu unterstützen. In dem Kontext der

Entscheidungsfindung mit verschiedenen Unsicherheiten sind die Qualität und Quantität der Daten wichtig, um die optimalen Entscheidungen zu treffen. Daher unterstützen Big Data Analyse und Industrie 4.0 Anwendungen die Entscheidungsfindung. [2] Ein Zusammenhang zwischen Zuverlässigkeit, Entscheidungsfindung und Industrie 4.0 ist in der Abbildung 7 nach [2] zu sehen.

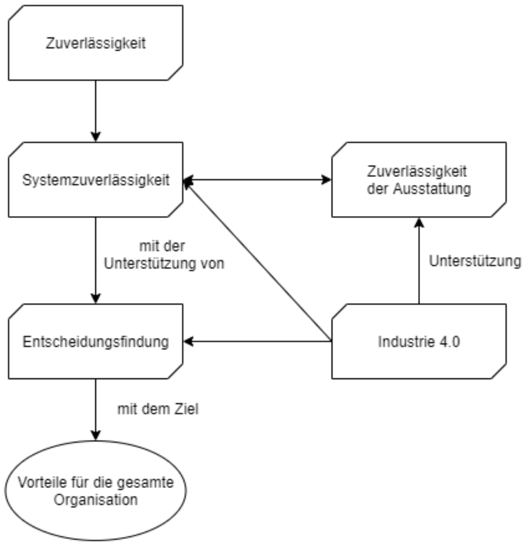

Abbildung 7: Zusammenhang der Zuverlässigkeit, Entscheidungsfindung und Industrie 4.0 nach [2, S.134]

Laut [13] haben die Produktionssysteme als Cyber-Physical Systems drei Faktoren, die ihre Zuverlässigkeit beeinflussen. Die sind Softwarezuverlässigkeit, Hardwarezuverlässigkeit und die Zuverlässigkeit durch menschliche Interaktion. Diese drei Arten von Zuverlässigkeit sowie deren Wechselwirkungen und Abhängigkeiten sollten berücksichtigt werden, um ein umfassendes Maß für die Zuverlässigkeit eines Produktionssystems zu erhalten. Mit der Entwicklung des IoT werden neue Möglichkeiten entwickelt, die Zuverlässigkeit von Systemen zu analysieren und bestehende Ansätze zu validieren. Die Systeme, die in einer Produktion eingesetzt sind, sind nicht sicherheitskritisch. Das heißt, dass die Ausfälle dieser Systeme keine Schäden für Menschen und Umwelt haben. Diese Ausfälle verursachen nur finanzielle Kosten. Die Entwicklung der Industrie 4.0 hat neue Möglichkeiten hervorgebracht, die die Art und Weise verändern können, wie die Zuverlässigkeit in Produktionssystemen analysiert und bewertet wird. Genauere Daten über verschiedene Produktionsprozesse, Ausfälle und Fehler können gesammelt, gespeichert und analysiert werden, weil sowohl IoT als auch CPS verwendet werden, um die Konnektivität zwischen verschiedenen Produktionskomponenten zu ermöglichen. Mit diesen gesammelten Daten können Hersteller bessere Entscheidungen treffen, um ihren Betrieb und ihre Rentabilität deutlich zu verbessern. Auf diese Weise können verschiedene Techniken wie Data Mining, Prozessoptimierung und Machine Learning angewendet werden. [13]

2.2.1 Herausforderungen und Chancen für Zuverlässigkeitstechnik

Industrie 4.0 weist Vorteile wie steigernde Produktionsvolumen, reduzierende Produktionsabfälle und schnelle Reaktion auf Kundenanforderungen auf. Neben diesen Vorteilen gibt es einige grundlegende Voraussetzungen, die erfüllt werden müssen. z.B. offene und dynamische Struktur in den Herstellungsprozessen, Kooperation und Zusammenarbeit, serviceorientierte Fertigungsplattformen usw. [7]. Angesichts der Vorteile und Anforderungen von Industrie 4.0 zeigt sich, dass Zuverlässigkeit auch neue Herausforderungen und Chancen birgt.

In [5] wurden einige Herausforderungen beim Prognostics and Health Management (PHM) für Zuverlässigkeit erläutert. PHM ist ein Forschungs- und Anwendungsgebiet, das die Art von Fehlern diagnostiziert, die Ausfallzeiten vorhersagt und deren Ausfälle proaktiv verwaltet [5]. PHM hat sich durch die umfassende Nutzung der neuesten Forschungsergebnisse der modernen Informationstechnologie und der künstlichen Intelligenz zu einer Lösung für das Gesundheitsmanagement von Geräten entwickelt [6]. Die Herausforderungen der Zuverlässigkeit sind gemäß [5] aufgelistet:

- Es ist notwendig, Methoden und Modelle zu entwickeln, die mit der Umwelt umgehen können, die die Bedingungen ändern, unter denen Degradation und Ausfall auftreten.
- Für effektiv informiertes PHM ist es notwendig, Informationen und Daten aus einer Komponente und einem System zu berücksichtigen.
- Es ist notwendig, die Integration und Interaktion von Informationen und Daten zu berücksichtigen, die auf mehreren Komponenten, die in einem System ausgeführt werden, überwacht werden.
- Der wirtschaftliche und sicherheitstechnische Wert eines PHM-Ansatzes für bedingte und/oder prädiktive Instandhaltung muss bewertet werden.

[7] hat in seinem Artikel das Prinzip von Industrie 4.0 vorgestellt und einige Herausforderungen und Chancen für die Zuverlässigkeitstechnik diskutiert. In der Tabelle 1 wurde eine Zusammenfassung für diese Herausforderungen und Chancen der verschiedenen Prinzipien, die in der Industrie 4.0 recherchiert werden, vorgestellt.

Tabelle 1: Zusammenfassung der Herausforderungen und Chancen für Zuverlässigkeitstechnik bezüglich [7]

	Herausforderungen	Chancen
System Modelling	Systemuntersuchung beim System-Modelling Unsicherheit im Mehrkomponentensysteme Sicherheit und Risiken des Systems	neue Zuverlässigkeitsmethoden
Big Data	dynamisches Verhalten des Systems Dataintegrität	Neural Networks Deep Learning Technik Verbesserung des Instandhaltung-Managements
CPS	Cyber-physical System-of-Systems-Interoperabilität CPS-basierte Produktionsanlagensteuerung Robuste digitale Fertigungsnetzwerke Robustheit und (Cyber-)Sicherheit der erzeugten Daten	Erhöhung der Systemstabilität und -flexibilität neue Konzepte und Instrumente für das Risikomanagement und -controlling.
Menschliche Rolle	Aus- und Weiterbildung des Menschen Widerstand gegen Veränderungen und gegen seine Interaktion mit der Maschine	Die Mensch-Maschine-Schnittstelle

3. Instandhaltungsstrategien

In der Industrie ist die Instandhaltung ein wichtiger Bestandteil, der sich auf die Betriebszeit und Effizienz von Maschinen auswirkt. Daher müssen Maschinenfehler identifiziert und korrigiert werden, um Produktionsausfälle zu vermeiden. [27] Instandhaltung ist ein ernstzunehmender Kostenfaktor in der Produktionsindustrie. Je nach Branche entfallen zwischen 15 und 70 Prozent der gesamten Produktionskosten auf Instandhaltungsaktivitäten [3]. Die Instandhaltungsstrategien, die vorhandene Ressourcen effizient nutzen und unnötige Kosten vermeiden, sind gefragt, damit die Gesamteffizient des Systems sichergestellt werden kann und die Wartungskosten auf einem niedrigen Niveau gehalten werden können [4]. In der Literatur finden sich vier unterschiedliche Instandhaltungsmethoden. Ein Überblick über die Instandhaltungsmethode ist in der Abbildung 8 zu zusehen.

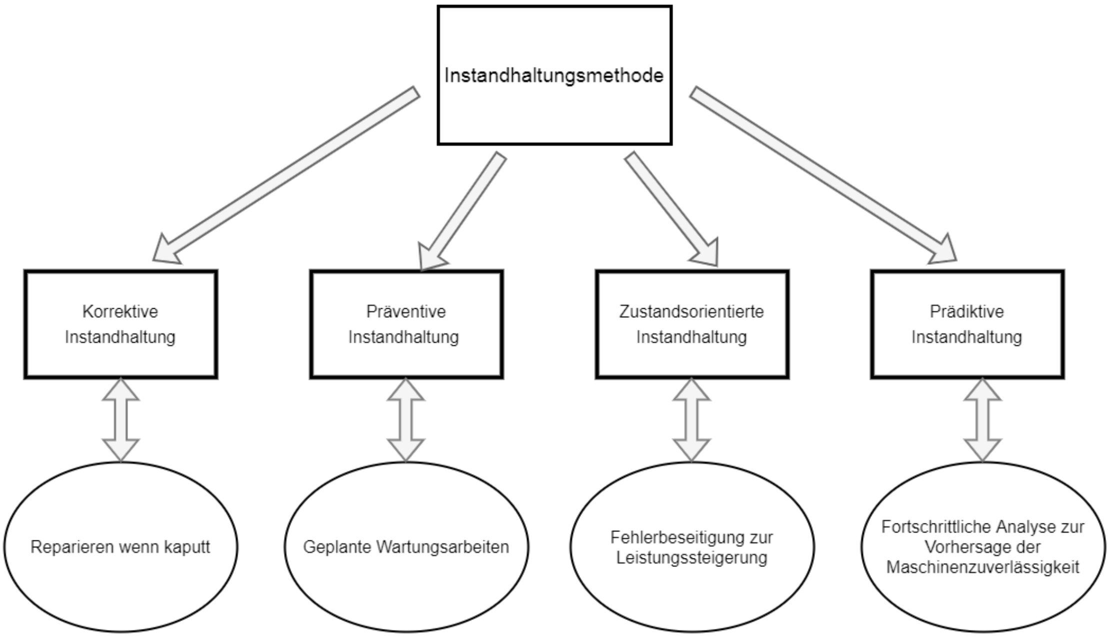

Abbildung 8: Instandhaltungstechniken

Die Instandhaltungstechniken werden zunächst so gemäß [12, 3, 6, 4,11] kategorisiert:

- Korrektive Instandhaltung (run 2 Failure/ Corrective Maintanence): Diese Instandhaltung findet immer nach Auftreten eines Fehlers statt. Deswegen ist diese Technik die einfachste. Die Reparatur der Maschine kann sofort oder zu einem späteren Zeitpunkt erfolgen.
- Präventive Instandhaltung (Preventive Maintenance): Diese Technik bezieht sich auf periodisch durchgeführte Wartungsarbeiten nach einem geplanten Zeitplan, um Ausfälle zu vermeiden. Diese Instandhaltungsmethode kann jedoch zu zusätzlichen

Betriebskosten und ungenutzter Lebensdauer führen und überwacht nicht den Zustand der Maschine, um Daten über die Maschine zu sammeln.

- Zustandsorientierte Instandhaltung (Condition-Based Maintenance, CBM): Diese Instandhaltungsstrategie überwacht den aktuellen Zustand einer Maschine. Die Wartungsarbeit wird nur durchgeführt, wenn sie tatsächlich notwendig ist. Erst nach einem oder mehreren sich verschlechternden Bedingungen des Prozesses kann der Prozess betrieben und Instandhaltungsmaßnahmen durchgeführt werden. Auf diese Weise können unnötige Wartungsarbeiten vermieden werden. Zustandsorientierte Instandhaltung kann normalerweise nicht vorausplant werden.
- Prädiktive Instandhaltung (Predictive Maintenance, PdM): Die Wartungsarbeit wird nur geplant, wenn es notwendig ist. Diese Instandhaltung basiert auf der kontinuierlichen Überwachung der Anlage bzw. der Maschine und verwendet eine Vorhersagemethode, um zu messen, wann die Wartungsarbeit erforderlich ist. So kann die Wartung geplant werden. Darüber hinaus ermöglicht die Instandhaltung eine frühzeitige Fehlererkennung basierend auf den historischen Daten, indem diese Vorhersagemethode wie Machine Learning verwendet wird.

Eine gute Instandhaltungsstrategie sollte den Zustand der Ausrüstung verbessern, die Ausfallraten der Ausrüstung reduzieren und die Wartungskosten minimieren und gleichzeitig die Lebensdauer der Ausrüstung maximieren. Aus diesem Grund ist die prädiktive Instandhaltung die prominenteste unter anderen Strategien und gewinnt im Zeitalter von Industrie 4.0 an Aufmerksamkeit. [27]

3.1 Prädiktive Instandhaltung

Neben anderen Instandhaltungsstrategien hat prädiktive Instandhaltung sich als eine der vielversprechendsten Strategien herausgestellt, weil diese Methode die Ausfallraten der Ausrüstung minimiert, den Zustand der Ausrüstung verbessert, die Lebensdauer der Ausrüstung verlängert und die Wartungskosten senken kann. Prädiktive Instandhaltung zieht die Aufmerksamkeit der Industrie auf sich und wird in der Industrie 4.0 dank seiner Fähigkeit, die Nutzung und Verwaltung von Assets zu optimieren, implementiert. [12] In der prädiktiven Instandhaltung ist es wichtig, dass die Fehler vorhergesagt werden. Die Prädiktive Instandhaltung verwendet Technologien des Condition Monitoring, um die Geräteleistung durch IoT-Systeme zu messen, die es ermöglichen, elektronische Geräte mit mechanischen und digitalen Maschinen zu verbinden und große Datenmengen zu sammeln. Die Daten werden gesammelt, um den Gerätezustand zu überwachen und Modelle zu erstellen, die Ausfälle verhindern helfen. [26]

Prädiktive Instandhaltung ist die am weitest entwickelte Instandhaltungsstrategie und vermeidet Überwartung oder Unterwartung. Die Vorteile von prädiktiver Instandhaltung lassen sich wie folgt zusammenfassen [30,11]:

- Höhere Zuverlässigkeit
- Höhere Verfügbarkeit
- Längere Lebensdauer der Maschine
- Verbesserte Produktqualität
- Höhere Anlagensicherheit
- Weniger Kosten für die Einzelteile und Arbeit
- Weniger Abfall bei Roh- und Verbrauchsmaterialien
- Energieeinsparungen

3.2 Condition Monitoring und die Modelle zur Fehlerprognose in der prädiktiven Instandhaltung

Prädiktive Instandhaltung basiert auf der Erfassung der Überwachungsdaten, die über den Gesundheitsstatus der Maschine informiert. Auf Basis dieser Überwachungsdaten kann das Remaining-useful-life (RUL) eingeschätzt werden. Condition Monitoring ist die kontinuierliche Erfassung relevanter Überwachungsdaten zur Abschätzung des RUL einer Maschine oder Komponente. Die Menge der erhobenen Daten ist hoch, weil der Zustand der Maschine kontinuierlich überwacht wird. [3]

Die Condition Monitoring-Technologie wird in zwei Kategorien unterteilt: Offline-Monitoring und Online-Monitoring. Offline-Condition Monitoring wird in bestimmten Zeitintervallen von Daten durch Sensoren an der Maschine durchgeführt. Nach dem Auslesen der Sensordaten werden die Analysen in einer Laborumgebung abseits der Maschine durchgeführt. Eine andere Methode ist die Online-Condition Monitoring. Sensoren in der Maschine sammeln ständig die Daten und vergleichen sie mit den akzeptablen Werten. In letzter Zeit ist die Online-Condition Monitoring aufgrund der Entwicklung von IoT-Lösungen sehr populär geworden. [29]

Condition Monitoring ist wichtig für die Klassifizierung der Betriebszustände, weil das Condition Monitoring die kontinuierlichen Überwachungen ermöglicht. Neben der Klassifizierung ist die Fehlerprognose für die Prädiktive Instandhaltung essenziell. Nach [9] gibt es verschiedene Methoden zur Fehlerprognose. Die sind physikalische Modelle, datengesteuerte Modelle und hybride Modelle. Die Methoden unterscheiden sich je nach den untersuchten Anwendungsfällen [9]. In den datengesteuerten Modellen spielen Machine Learning Algorithmen eine wichtige Rolle.

3.2.1 Physikalische Modelle zur Fehlerprognose in der prädiktiven Instandhaltung

Diese Modelle simulieren mit deterministischen oder stochastischen Prozessen dynamisches Systemverhalten und Merkmale, die den Versagensmechanismen wie Verschleiß oder Korrosion entsprechen. Diese physikalischen Modelle sind sehr präzise und ihre Ergebnisse können durch die zugrunde liegende Physik leicht interpretiert werden. Sie sind oft sehr komplex und brauchen Zeit in der Erstellung. [9]

3.2.2 Datengesteuerte Modelle zur Fehlerprognose in der prädiktiven Instandhaltung

Für die aus der Datenwissenschaft bekannte Prognose des „Remaining-useful-life (RUL)" werden datengesteuerte Modelle verwendet, da Systemmodelle oder systemspezifische Informationen nur eingeschränkt verfügbar sein dürfen. Merkmale werden hauptsächlich aus aussagekräftigen Datenquellen, wie einem Zustandsüberwachungssystem, extrahiert, um das Systemverhalten vollständig widerzuspiegeln. Mit Hilfe der Algorithmen des maschinellen Lernens oder statistischer Auswertung können die zukünftigen Zustände der Maschine prognostiziert werden. Zwar ist dieses Model auch für komplex verhaltende Systeme geeignet, aber die benötigten Datensätze für verschiedene Ausfallszenarien sind nicht passend bzw. können nur sehr zeit- und kostenintensiv ermittelt werden. [9]

3.2.3 Hybride Modelle zur Fehlerprognose in der prädiktiven Instandhaltung

Da es Nachteile hat, nur physikalische und datengestützte Modelle zu verwenden, hat sich eine dritte Möglichkeit im Sinne eines ganzheitlichen Ansatzes für hybride Modelle herausgebildet. Dieser Ansatz kombiniert die beiden Modelle seriell und/oder parallel und bietet so die Möglichkeit, einerseits weniger abhängig von Datensätzen zu sein und andererseits noch unbekannte Fehlerzustände erkennen zu können. [9]

3.3 Methoden für die datengesteuerten prädiktiven Instandhaltung-Modelle

In [12] sind fünf Kategorien festgelegt, die prädiktive Instandhaltung antreiben. Die sind smarte Sensoren, Netzwerke, Technologie Integration, Augmented Intelligence und Augmented Behavior. Smarte Sensoren sammeln die Maschinendaten oder Umgebungsdaten. Diese Daten werden durch das Netzwerk gespeichert oder über Bluetooth/ WiFi übertragen. Dieses Netzwerk ermöglicht Data Monitoring, um sicherzustellen, dass die Maschinen in einem gesunden Zustand sind. Technologie Integration ermöglicht Datenmanagement. Die Methoden für die Instandhaltung wie Machine Learning ersetzen Modelle mit historischen Daten, um Fehler vorherzusagen. Dies bietet proaktive Warnungen für zukünftige Wartungsanforderungen. Augmented Intelligence unterstützt die Datenverarbeitung und Datenanalyse. Augmented Behavior ermöglicht Virtualisierung, Computer- und Serviceplattformen über Apps, um den Betreiber zu unterstützen. Allerletzt wird ein automatischer Wartungsplan für die Produktionsplanung, die Planung von Wartungsaufgaben und die Zuweisung von Technikern erstellt. [12]

Datengesteuerte Modelle verwenden verschiedene Methoden, wie Regression, neuronale Netze, statistische Modelle, Simulationen usw. [9] Diese Methoden sind in statistische Methoden und Maschine Learning Methoden unterteilt.

3.3.1 Statistische Modelle zur prädiktiven Instandhaltung

Statistische Modelle verwenden mathematische Modelle und statistische Annahmen, um Stichprobendaten zu generieren und Vorhersagen über Prozesse zu treffen. Die Anwendung statistischer Modellierung auf Rohdaten hilft dabei, Beziehungen zwischen Variablen zu identifizieren, Vorhersagen zu treffen und die Datenanalyse strategisch anzugehen. [16] Informationen aus allen Maschinenausfalldaten ermöglichen die Vorhersage von Ausfällen anhand statistischer Informationen und ermöglichen so die Entwicklung einer prädiktiven Instandhaltungsstrategie. [28] Für die prädiktive Instandhaltung sind die Vorhersage der Ausfälle und das Klassifizieren der Betriebszustände signifikant. Die statistischen Methoden helfen bei der Prognose des Ausfalls der Maschine und einige statistische Methoden klassifizieren auch die Betriebszustände. Decision Tree, Linear Discriminant Analysis und Logistic Regression sind die wichtigen statistischen Methoden für die Klassifizierung.

3.3.1.1 Decision Tree

Decision Tree (DT) ist ein Netzwerksystem, das hauptsächlich aus Knoten und Zweigen besteht. Knoten umfassen die Wurzelknoten und Entscheidungsknoten.[12] Entscheidungsknoten werden verwendet, um beliebige Entscheidungen zu treffen und haben mehrere Verzweigungen, während Blattknoten die Aussage dieser Entscheidungen sind und keine weiteren Verzweigungen enthalten. Und die Zweige der Knoten stellen die Entscheidungsregel dar. [26] Abbildung 9 zeigt die Vorgehensweise des DT. Der Entscheidungsprozess beginnt mit der Wurzelknote. Dann wird der Wurzelknote durch Verzweigen verschiedene Knoten erzeugt. Das Entscheidungsmodell endet mit den Blattknoten.

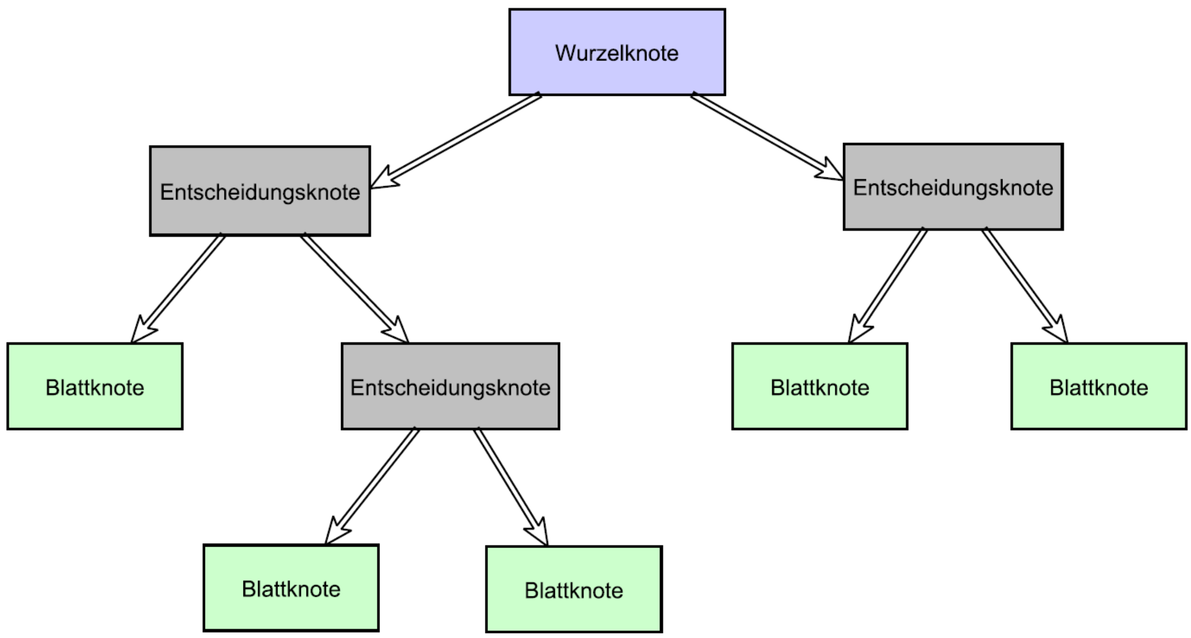

Abbildung 9: Decision Tree Modell

Das DT-Modell kann einen schwierigen Entscheidungsprozess in eine Sammlung von einfachen Entscheidungen zerlegen. Auf diese Weise bietet es eine einfach interpretierbare Lösung. Wichtig ist es zu bestimmen, welches Merkmal zuerst aufgeteilt werden soll. Das bedeutet, welches Merkmal im Datensatz eine entscheidende Rolle bei der Klassifizierung spielt. [6] DT-Klassifiziere haben in mehreren Bereichen, wie der Zeichenerkennung, der medizinischen Diagnose und der Spracherkennung, beträchtliche Popularität erlangt. [12]

3.3.1.2 Logistic Regression

Logistic Regression (LR) ist eine Klassifizierungsmethode, die eine niedrige Komplexität des Algorithmus beinhaltet. Das LR-Modell verwendet eine lineare Kombination von Merkmalen als Input und wendet eine nichtlineare Funktion an, um die Abbildung durchzuführen, sodass jedes Output in den entsprechenden Bereich (0 oder 1) fällt und eine wahrscheinlichkeitstheoretische Interpretation erhalten wird. LR wird verwendet, um kategoriale bedingte Variationen unter Verwendung eines gegebenen Satzes unabhängiger Variablen einzuschätzen. [6,12]

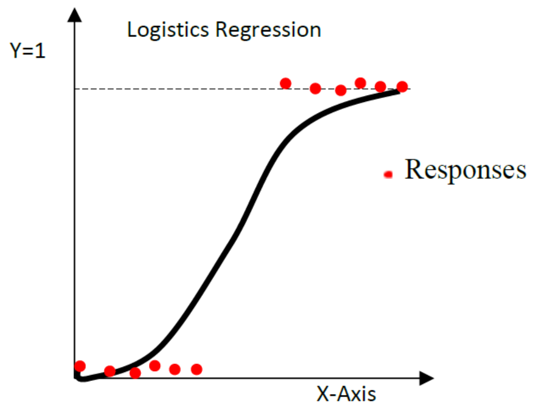

Abbildung 10: Logistic Regression [12, S.13]

In der Abbildung 10 ist ein logistisches Regression-Model zu sehen. In der Logistischen Regression ist die Funktion keine Gerade sondern logistische Funktion. Diese Funktion nimmt nur die Werte zwischen 0 und 1 und die Outputs befinden sich ebenfalls um die Werte 0 oder 1.

3.3.1.3 Linear Regression

Linear Regression (LinR) bezieht sich auf die multiple lineare Kombination von Regressionskoeffizienten. Die Koeffizienten werden nach der verallgemeinerten Methode der kleinsten Quadrate berechnet. Linear Regression ist deterministisch und hat weniger Parameter. Linear Regressionsalgorithmen sind Zeitreihenregressionsalgorithmen, die bereits im Bereich der prädiktiven Instandhaltung verwendet werden. [12]

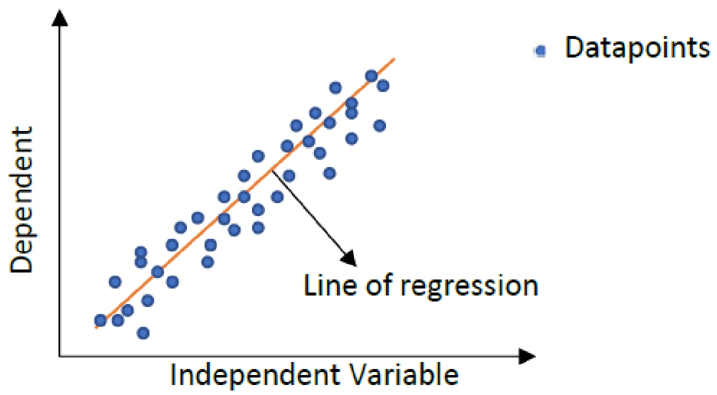

Abbildung 11:Linear Regression Modell [12, S.15]

Die lineare Regressionsanalyse untersucht und modelliert die funktionale Beziehung zwischen abhängigen Variablen und unabhängigen Variablen [26]. Auf der Abbildung 10 ist das Modell für die Lineare Regression dargestellt. Für die Lineare Regression ist die Regressionslinie eine gerade und die Datenpunkte sammeln sich um die Linie.

3.3.1.4 Symbolic Regression

Symbolic Regression (SR) bezeichnet Modelle in Form eines Syntaxbaums aus beliebigen mathematischen Symbolen, die leicht in einfache mathematische Funktionen umgewandelt werden. Top-down-Syntaxbäume werden zur Zielschätzung überprüft. Syntaxbäume wurden unter Verwendung der stochastischen genetischen Programmiertechnik entwickelt. [12] Ein Beispiel für Symbolic Regression ist in der Abbildung 11 zu sehen.

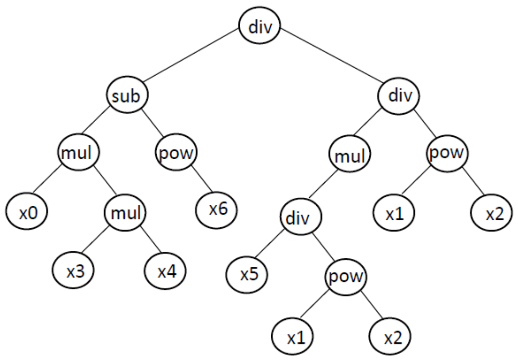

Abbildung 12: Ein Beispielmodell für Symbolic Regression [12, S.15]

3.3.1.5 Linear Discriminant Analysis

Linear Discriminant Analyse (LDA) wird zwischen unabhängigen Variablen und abhängigen Variablen durchgeführt. Es erlaubt, unabhängige Variable in die gleichen oder verschiedenen Gruppen zu unterteilen, je nachdem, ob sie die abhängige Variable beeinflussen. LDA kann in vielen Anwendungen verwendet werden. Dies sind beispielweise statistische Studien, Mustererkennung, die Charakterisierung der Objekt- oder Ereignisklasse. [29]

Die Klassifizierungsmetode für LDA funktioniert wie in der Gleichung 1:

$$G_j = c_{j0} + \sum c_{ji} x_j + \ln(n_j/N) \tag{1}$$

- G_j ist der Score der j-ten Gruppe,
- c_{j0} ist der konstante Wert für die j-te Gruppe,
- c_{ji} ist der Koeffizienten-Wert der i-ten Variablen und der j-ten Gruppe,
- x_j ist die i-te Variable,
- n_j ist die Anzahl der Beobachtungen innerhalb der j-ten Gruppe, und
- N ist die Gesamtzahl der Beobachtungen. [29]

3.3.2 Machine Learning Methoden zur prädiktiven Instandhaltung

Machine Learning (ML) wird häufig mit weitgehend verfügbaren Daten verwendet, um verschiedene Probleme zu lösen. Es wird auch in der Informatik und anderen Bereichen wie prädiktive Instandhaltung von Industrieanlagen verwendet, was eines der potenziellen Anwendungsgebiete für datengesteuerte Methoden ist [6]. ML-Algorithmen erfordern normalerweise das Sammeln großer Datenmengen der Ausfallstatusszenarien und der Gesundheitszustandsszenarien für das Modelltraining. Die Entwicklung von ML-Algorithmen umfasst die Auswahl historischer Daten, die Vorverarbeitung von Daten, die Modellauswahl, das Modelltraining, die Modellvalidierung und die Wartung. [12] Typische Aufgaben von ML sind Klassifikation, Regression, Lernassoziationen, Clustering und andere maschinelle Lernaufgaben wie Reinforcement Learning, Ranking-Lernen und Strukturvorhersage [31]. Die Algorithmen des maschinellen Lernens erstellen bei der prädiktiven Instandhaltung nicht nur die Prognose der Ausfälle der Maschine sondern auch die Klassifizierung der Betriebszustände. Weil diese Algorithmen mit schwierigen Situationen und mit vielen Daten bessere Entscheidungen treffen, haben die Algorithmen für die prädiktive Instandhaltung bessere Einsätze.

3.3.2.1 Support Vector Machine

Support Vector Machine (SVM) ist ein bekannter Algorithmus des maschinellen Lernens, die aufgrund ihrer hohen Genauigkeit sowohl für die Klassifizierung als auch für die Regressionsanalyse weit verbreitet ist. SVM wird definiert als statistisches Lernkonzept mit einem adaptiven computergestützten Lernverfahren. [12] In der prädiktiven Instandhaltung von Industrieanlagen wird SVM häufig verwendet, um einen bestimmten Zustand anhand des erfassten Signals zu identifizieren. Der Hauptzweck von SVM besteht darin, eine Hyperebene zu finden und die Datenpunkte auf beiden Seiten der Hyperebene korrekt aufzuteilen. [6] Eines der Hauptmerkmale von Support-Vektor-Maschinen ist die hohe Genauigkeit beim Trennen verschiedener Datenklassen, wodurch es möglich ist, den besten Punkt zum Trennen von Datenklassen zu bestimmen. [27] Die Hyperebene und zwei Datenklassen sind in der Abbildung 13 als ein Beispiel SVM-Modell dargestellt. In diesem linearen SVM-Modell sind die zwei Klassen auf den beiden Seiten der Hyperebene aufgeteilt.

Neben der linearen Klassifizierung können SVMs mit der nichtlinearen Klassifizierung umgehen, indem sie den Kernel-Trick verwenden, der auch als Kernel-Maschine bekannt ist [31].

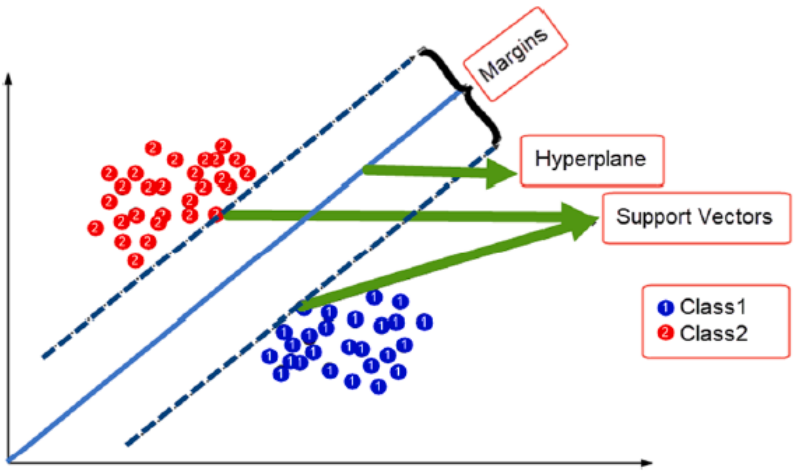

Abbildung 13: Ein SVM Beispielmodell mit zwei Klassen für die Klassifizierung [29, S.4]

3.3.2.2 Random Forest

RF ist ein Ensemble-Algorithmus, der aus mehreren DT-Klassifizieren besteht, und der Output wird gemeinsam durch diese einzelnen Entscheidungsbäume bestimmt. Es kann hochdimensionale Daten ohne Merkmalauswahl verarbeiten. Jeder Baum trifft für sich eine eigene Entscheidung. Aus diesen Entscheidungen liefert der Algorithmus eine endgültige Entscheidung. Außerdem ist die Geschwindigkeit des Entscheidungsprozesses hoch und die Generalisierungsfähigkeit ist gut. [6,12] In der Abbildung 14 ist ein Algorithmus des RF dargestellt. Die drei Bäume treffen erst seine eigenen Entscheidungen und am Ende liefert der RF-Algorithmus basierend auf die drei Entscheidungen eine RF-Vorhersage.

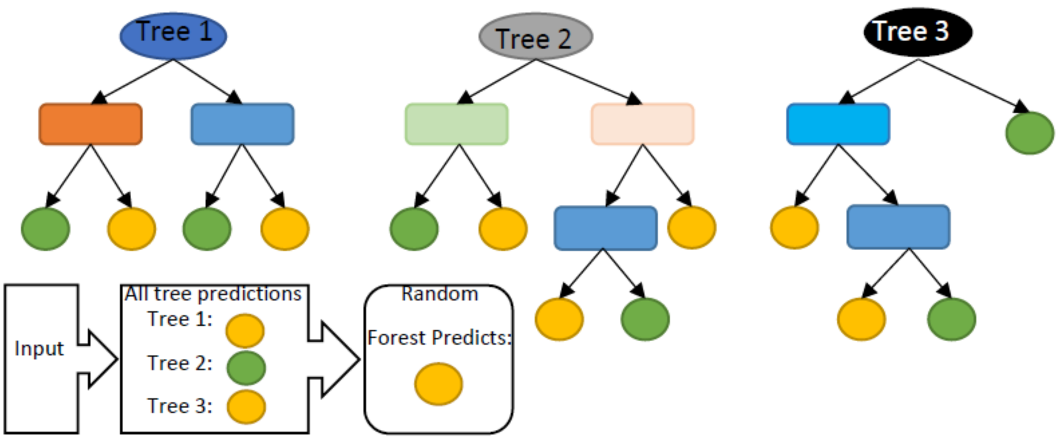

Abbildung 14: Random Forest Algorithmus [12, S.11]

Weil die Konstruktion und die Anwendung des DT mit großen Datensätzen schwer sind, wurde die Entwicklung des RF ermöglicht. Der Zweck des RF-Algorithmus besteht darin, mit mehr als einem Entscheidungsträger effizientere Ergebnisse zu erzielen als bei anderen Methoden. [29]

3.3.2.3 Artificial Neural Network und Deep Neural Network

Artificial Neural Network (ANN) ist eine intelligente Rechentechnik, die vom biologischen neuronalen Netz (NN) inspiriert wurde. Es ist ein massiv paralleles Rechensystem, das aus vielen einfachen Prozessoren mit vielen Verbindungen besteht, und es wurde entwickelt, um nichtlineare Probleme zu lösen. ANNs lernen die Grundgesetze anhand von Beispielen, die sich aus den gegebenen symbolischen Situationen ergeben, deswegen ist ANN attraktiv und weit verbreitet. ANN eignet sich besonders für Systeme mit großem Umfang, komplexer Struktur und unklaren Informationen. Gleichzeitig wurde ANN in mehreren industriellen Anwendungen mit Soft-Sensing und in prädiktiven Steuerungssystemen vorgeschlagen. [6,12] Die Architektur eines ANN ist in der Abbildung 15 dargestellt.

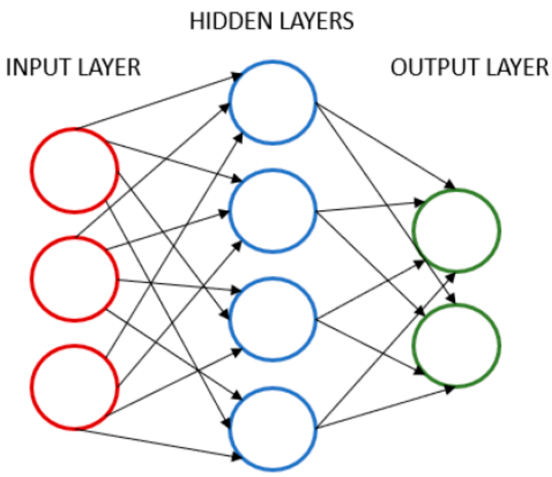

Abbildung 15: Neural Network Architektur [31, S.13]

Wie in der Abbildung 15 zu sehen ist, hat das Beispiel-Modell eines ANNs drei Schichten von Neuronen. Die erste (Input) Schicht ist mit einer verdeckten (Hidden) Schicht verbunden. Die verdeckte Schicht ist mit der letzten (Output) Schicht verbunden. Die Neuronen übertragen ein Signal von der ersten Schicht zur letzten Schicht. [31]

Deep Neural Network (DNN) ist ein tieferes neuronales Netz und bezieht sich auf ein NN mit vielen versteckten Schichten. Alle Schichten sind vollständig miteinander verbunden. DNNs können adaptive Fehlermerkmale extrahieren, wichtige Informationen effektiv präsentieren und eine intelligente Diagnose durchführen. Sie können auch die Erkennungsgenauigkeit verbessern und sind äußerst effektiv bei der Reduzierung von Fehlern in manuellen Konstruktionsmerkmalen. [6]

3.3.2.4 Deep Learning

Deep Learning (DL) ist eine Methode, die hierarchische ML-Algorithmen verwendet, um strukturierte Informationen aus großen Datensätzen zu extrahieren und hat in der künstlichen Intelligenz und in der Technologiebranche fast unvorstellbare Fortschritte gemacht. Außerdem sind Deep Learning Algorithmen im datengesteuerten PdM zu einer unverzichtbaren Methode geworden. Deep Learning kann als Brücke zwischen Sensordaten und datengesteuertem Gesundheitsmanagement der Maschine fungieren, weil es Merkmale aus den Originaldaten automatisch extrahieren und den Gesundheitszustand genau beschreiben kann. [6,13]

4. Anwendungsbeispiel für die automatische Klassifizierung von Betriebszuständen

4.1 Datenquelle und Programmiersprache

In dieser Arbeit wurde ein Datensatz verwendet, welcher durch eine offene Webseite Kaggle [17] bereitgestellt wurde. Dieser synthetische, prädiktive Instandhaltung-Datensatz wurde erstellt, da die realen prädiktiven Instandhaltung-Datensätze schwer zu beschaffen und besonders schwierig zu veröffentlichen sind [18]. Dieser Datensatz spiegelt am besten die realen Daten zur prädiktiven Instandhaltung wider, die in der Branche im Rahmen des Wissens und der Erfahrung angetroffen werden [18].

Der Datensatz hat 10 Spalten und 10.000 Datenpunkte. Die Spalten zeigen die Merkmale eines Datenpunktes. Die Merkmale sind gemäß [17] und [18]

- UDI: Eindeutige Kennung von 1 bis 10.000
- Product.ID: bestehend aus den Buchstaben L (low), M (medium) und H (high) als Produktqualitätsvariante und einer Seriennummer
- Type: Produktqualitätsvariante L, M oder H
- Air Temperature [K]: mit einem Random-Walk-Prozess erzeugt und später auf eine Standardabweichung von 2 K um 300 K normalisiert
- Process Temperature [K]: erzeugt durch einen Random-Walk-Prozess, normiert auf eine Standardabweichung von 1 K, addiert zur Lufttemperatur plus 10 K
- Rotational Speed [rpm]: berechnet aus einer Leistung von 2860 W, überlagert mit einem normalverteilten Lärmpegel
- Torque [Nm]: Drehmomentwerte verteilen sich normalerweise um 40 Nm mit einem Ïƒ = 10 Nm und keinen negativen Werten
- Tool Wear [min]: Die Qualitätsvarianten H/M/L addieren dabei 5/3/2 Minuten Werkzeugverschleiß zum gebrauchten Werkzeug hinzu.
- Target: Ausfallzustand (1) oder Betriebsbereitzustand (0)
- Failure Type: Ausfalltype (No Failure, tool wear failure, heat dissipation failure, power failure, overstrain failure, random failures)

Weil die Merkmale der Sensordaten für die prädiktive Instandhaltung wichtig sind, werden in dem Skript nur die entsprechenden Sensordaten und Ausfalldaten verwendet. Das bedeutet, dass für das Skript die Daten von „*air temperatur*", „*process temperatur*", „*rotational speed*" „*torque*" „*tool wear*" und „*target"* wichtig sind.

Der Datensatz wurde mit Hilfe der Programmiersprache “R“ durch RStudio durchgeführt. R ist eine Software für statistische Berechnungen (lineare und nichtlineare Modellierung, klassische statistische Tests, Zeitreihenanalyse, Klassifikation, Clustering…) und Grafiken. R wurde von Ross Ihaka und Robert Gentleman entwickelt und als Open-Source-Software kostenlos zur Verfügung gestellt [20]. Neben den Funktionen als Statistikpaket unterstützt R auch maschinelles Lernen oder Methoden der Künstlichen Intelligenz. R kann einfach über Pakete “packages” erweitert werden, damit diese Funktionen durchgeführt werden können [19,20]. Eine der Stärken von R besteht darin, dass gut gestaltete Plots von guter Qualität einfach erstellt werden können. Diese Diagramme können bei Bedarf mathematische Symbole und Formeln enthalten. [19]

4.2 Vorbereitung der Daten für die Analyse

Das Skript zu der Analyse wurde von Jun.-Prof. Dr. Antoine Tordeux, Leiter des Lehrstuhls für Verkehrssicherheit und Zuverlässigkeit an der Universität Wuppertal, erstellt und für diese Arbeit zur Verfügung gestellt.

Bevor der Datensatz analysiert wird, sollte die Software zu Beginn auf dem Stand „0“ sein. Das bedeutet, dass „Console“ und „Environment“ auf der Software gelöst werden sollen. Damit die Analyse und die Klassifizierung durchgeführt werden können, sollten die benötigten Pakete installiert werden (s. Anhang A).

Das Paket *„MASS“* ist die Abkürzung für „Modern Applied Statistics with S“ und bietet unterstützende Funktionen und Datasätze [21]. Das andere Paket „*corrplot*“ wird verwendet, um die Korrelationsmatrix zu erstellen. Für die Klassifizierung werden verschiedene statistische Methoden oder Algorithmen des maschinellen Lernens verwendet. Für die Algorithmen des maschinellen Lernens werden auch verschiede R Pakete gebraucht. Die Pakete und die genutzten R Codes sind in der Tabelle 2 aufgelistet.

Tabelle 2: R Pakete [22]

R Pakete	Algorithmen	R Code
rpart	Decision Tree	rpart
neuralnet	Neural Network	neuralnet
e1071	Support Vector Machine	svm
klaR	naive Bayes	NaiveBayes

Nach der Installation der Pakete wird der Datensatz gelesen. Hier ist es wichtig, dass die Datei für den Datensatz sich im sogenannten Arbeitsverzeichnis (Working Directory) befindet. Nachdem der Datensatz gelesen wurde, werden die relevanten Spalten des Datensatzes als Dataframe und Variable gespeichert. Einige der Sensordaten werden hier nicht berücksichtigt, weil die Daten wie z.B. ID Nummer oder Type des Produkts nicht für die Klassifizierung relevant sind. Die relevanten Messvariablen werden auf einem anderen Datenframe (*X*) zugeordnet. Diese Messvariablen sind „*air temperature*“, „*process temperature*“ „*rotational speed“*, *„torgue*“ und „*tool wear*“. Der Betriebszustand der Maschine wird einer Variable (*Y)* zugeordnet. Für den Betriebsbereitzustand hat *Y* den Wert 0 (*Y==0*) und für den ausgefallenen Zustand hat *Y* den Wert 1 (*Y==1*). Diese Zuordnung hilft bei der Klassifizierung.

4.2.1 Korrelationsanalyse und Hauptkomponentenanalyse

Um den Zusammenhang zwischen den Variablen zu sehen, wird eine Korrelationsanalyse durchgeführt (s. Anhang B). Die Berechnung der Korrelation ist dafür geeignet, die Abhängigkeit der zwei Variablen zu untersuchen. Eine Korrelation zeigt die Richtung (positiv oder negativ) und die Stärke einer linearen Beziehung zwischen zwei Variablen. Eine positive Korrelation liegt vor, wenn hohe Werte der Variablen A mit hohen Werten der Variablen B assoziiert sind. Es besteht eine negative Korrelation, wenn hohe Werte der Variablen A mit niedrigen Werten der Variablen B assoziiert sind. Die Stärke der Korrelation wird durch die Korrelationskoeffizienten bestimmt. Der Korrelationseffizient liegt zwischen -1 (starke negative Korrelation) und +1 (starke positive Korrelation). [20] Die Korrelation Matrix ist in der Abbildung 16 zu sehen.

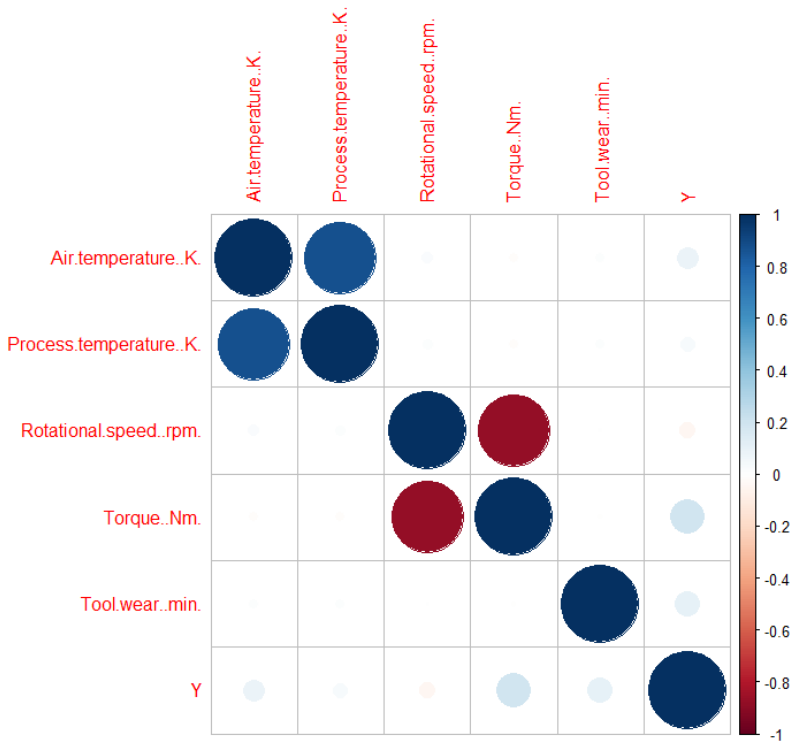

Abbildung 16: Korrelationsmatrix der Variablen

Die blaufarbigen Punkte zeigen positive Korrelationen und die rotfarbigen Punkte zeigen negative Korrelationen. Je dunkler der Punkt wird, desto stärker wird die Korrelation. Wenn die Punkte eine sehr helle Farbe aufweisen, ist der Korrelationskoeffizient fast null und die Variablen haben dann sehr weiche Korrelationen. Positive Korrelation heißt, dass die zwei Variablen einen linearen Zusammenhang haben. In der Abbildung lässt sich schlussfolgern, dass die „*Air Temperatur*“ und „*Process Temperature*“ einen linearen Zusammenhang haben. Aber zwischen „*Rational Speed*“ und „*Torque*“ gibt es einen negativen Zusammenhang. Der Zustand der Maschine (*Y*) hat leichte lineare Zusammenhänge mit Variablen außer „*Rational Speed*“. „*Rational Speed*“ und der Zustand der Maschine (*Y*) haben einen sehr leichten negativen Zusammenhang.

Nach der Korrelationsanalyse wurde eine Hauptkomponentenanalyse durchgeführt (s. Anhang B). Die Hauptkomponentenanalyse (Englisch für Principal Components Analysis, PCA) ist ein Ansatz, um einen niedrigdimensionalen Merkmalssatz aus einem großen Satz von Variablen abzuleiten. Wenn eine große Anzahl korrelierter Variablen konfrontiert werden, ermöglicht die Hauptkomponente, die Menge mit einer kleineren Anzahl repräsentativer Variablen zusammenzufassen, die zusammen den größten Teil der Variabilität in der ursprünglichen Menge erklären. [23]

Damit die Analyse zum besseren Verständnis führt, wird für jede Hauptkomponente (Englisch für Principal Component, PC) die Standardabweichung und der Anteil der Varianz berechnet. Die Daten zu der Analyse sind in der Tabelle 3 und in der Abbildung 17 zu sehen. *PC1*, *PC2* und *PC3* haben den größeren Anteil der Varianz.

Tabelle 3: Daten zur Hauptkomponentenanalyse

	PC1	PC2	PC3	PC4	PC5
Standard-abweichung	1,3823	1,3568	0,9998	0,35543	0,3500
Anteil der Varianz	38,21 %	36,82 %	19,99 %	2,527 %	2,245 %
Kumulativer Anteil der Varianz	38,21 %	75,03 %	95,02 %	97,550 %	100 %

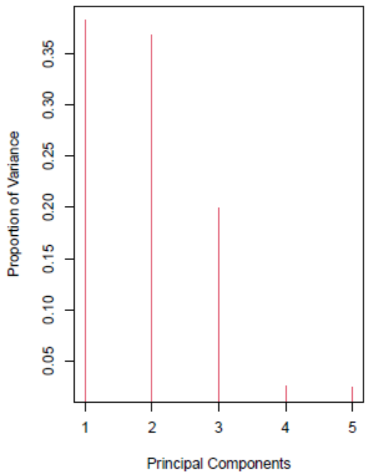

Abbildung 17: Anteil der Varianz der Hauptkomponenten

Die ersten drei Hauptkomponenten haben eine höhere Standardabweichung bzw. Varianz. Deswegen wurde der Zusammenhang zwischen dem Zustand der Maschine und den drei Hauptkomponenten mit dem Plot in der Abbildung 18 verglichen. Der Kreis zeigt den betriebsbereiten Zustand und das Dreieck zeigt den ausgefallenen Zustand. In der Abbildung 18 ist es ersichtlich, dass der betriebsbereite Zustand sich in der Mitte ansammelt.

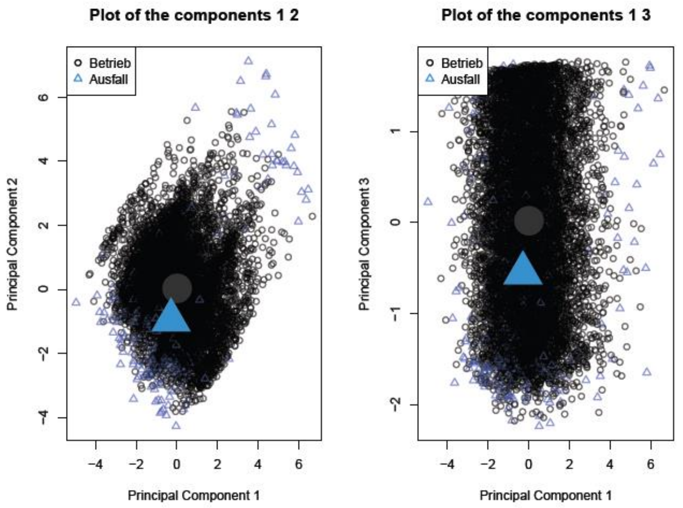

Abbildung 18: Plot of components 1-2 und 1-3

Zwischen den Hauptkomponenten und den Variablen wurde eine Korrelationsanalyse durchgeführt. Die Ergebnisse werden in der Tabelle 4 und auf der Abbildung 19 gezeigt. Hier wird deutlich, dass zwischen dem „*Rational Spee*d“ und *PC3* kein Zusammenhang besteht, aber „*Rational Speed“* und *PC1* einen linearen Zusammenhang aufweisen. „*Tool wear*“ und *PC1* haben keinen Zusammenhang, aber „*Tool wear“* und *PC3* haben einen sehr starken negativen Zusammenhang. All diese Korrelationen werden in der Abbildung 19 erkannt, weil da die Korrelationsbeziehungen kreisförmig angezeigt werden.

Tabelle 4: Korrelationsanalyse zwischen Hauptkomponente und Variablen

	PC1	PC2	PC3	PC4	PC5
Air temperature	0.699072	-0.670165	0.015321	-0.150606	0.198138
Process temperature	0.69787	-0.67139	0.015726	0.15222	-0.19692
Rotational speed	0.68627	0.68303	0.003039	-0.20046	-0.14929
Torque	-0.68079	-0.68851	0.00049	-0.200707	-0.148919
Tool wear	0.02344	-0.01909	-0.99954	-0.000621	-0.000588

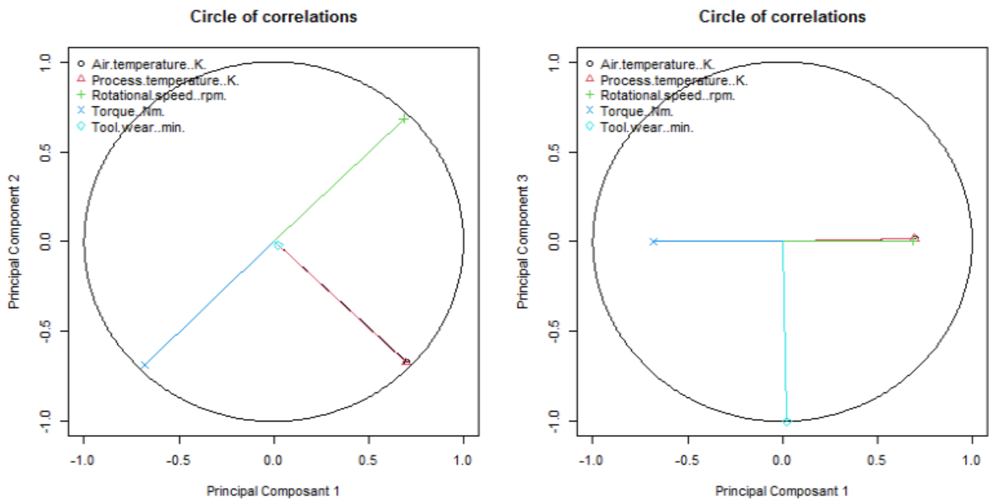

Abbildung 19: Circle of correlations

Bei der Klassifizierung von Betriebszuständen werden Hauptkomponenten für die Prognose benutzt. Die Hauptkomponentenanalyse wurde erst durchgeführt, um die Dimensionen des Datensatzes zu reduzieren. Die ersten drei Hauptkomponenten haben den hohen Varianzanteil. Eine neue Korrelationsmatrix wird in der Abbildung 20 erstellt, um zu testen, ob die ersten drei Hauptkomponenten mit den Zuständen der Maschine (*Y*) eine Korrelation verfügen. Hier wird dargestellt, dass die erste Hauptkomponente und der Zustand keine Korrelation besitzen. Der Zustand *Y* hat schwache negative Zusammenhänge mit den anderen Komponenten. Die höchste Korrelation gibt es zwischen *PC4* und dem Zustand der Maschine (*Y*). Deswegen werden bei der Zustandsklassifizierung alle fünf Hauptkomponenten genutzt, damit die Prognose bei der Zustandsklassifizierung bessere Ergebnisse erzielt.

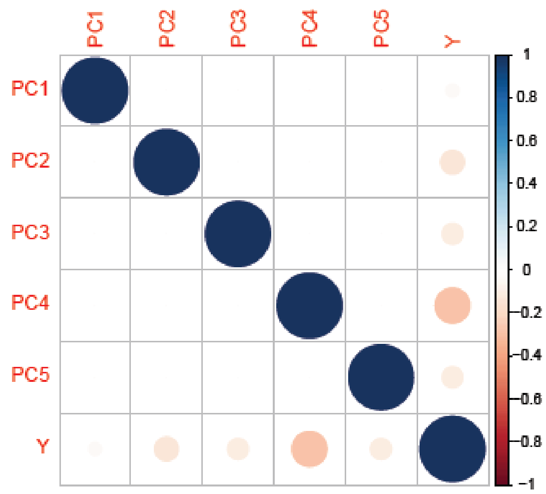

Abbildung 20: Korrelationsmatrix der Hauptkomponenten und des Zustands Y

4.3 Klassifizierung von Betriebszuständen

Nach den ersten Analysen und der Vorbereitung des Datensatzes kann der Datensatz weiterhin mit verschiedenen Algorithmen analysiert werden (s. Anhang C). Das Ziel ist es, herauszufinden, welcher Algorithmus für diesen Datensatz die richtige Vorhersage für den Betriebszustand und dem ausgefallenen Zustand mit dem geringsten Fehler trifft. Diese Fehler werden mit mittlerer quadratischer Abweichung (auf Englisch: mean squared error (MSE)) mit der Gleichung 2 gerechnet:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (2)$$

wobei y_i der beobachtete Wert, $\hat{y}_i$ der entsprechende vorhergesagte Wert und i die Anzahl der Beobachtungen sind. Um die Leistung einer statistischen Lernmethode für einen bestimmten Datensatz zu bewerten, wird gemessen, wie gut die vorhergesagten Daten mit den tatsächlich beobachteten Daten übereinstimmen. MSE wird berechnet, um zu quantifizieren, wie nahe der vorhergesagte Response-Wert für eine bestimmte Beobachtung am wahren Response-Wert für diese Beobachtung liegt. MSE ist ein kleiner Wert, wenn die vorhergesagten und tatsächlichen Werte sehr nahe beieinander liegen, und ein großer Wert, wenn sich die vorhergesagten und tatsächlichen Werte für einige Beobachtungen erheblich unterscheiden. [23]

Die Beobachtungen der Datenframes *X* werden erst in zwei Datenframes wie Training und Testing unterteilt. Trainingsdaten haben die meisten Beobachtungen von Datenframe *X* und der Rest der Daten werden Testdaten. Die Algorithmen lernen aus den Trainingsdaten. Die Testdaten sind von den Trainingsdaten unabhängig und werden bei dem Training nicht benutzt. Die Unterteilung in Training und Testing ist wichtig, damit die Ergebnisse besser verglichen werden können.

Für die Prognose werden drei Fehler berechnet. Erster Fehler (Error 1) zeigt, wie gut die Algorithmen betriebsbereite Zustände vorhersagen können. Zweiter Fehler (Error 2) zeigt, wie gut die Algorithmen ausgefallene Zustände vorhersagen können. Und der Total-Fehler (Total Error) zeigt, wie gut die Algorithmen für diese beiden Zustände vorhersagen können. Das heißt, dass der Total-Fehler aus Fehler 1 und Fehler 2 besteht. Das Ziel ist, diese Fehler zu minimieren und zu entscheiden, welcher Algorithmus die Zustände besser vorhersagen kann.

Für die Entscheidung werden sieben verschiedene Algorithmen benutzt. Die Algorithmen sind Logistische Regression (REG), Lineare Diskriminantfunktion (LDA), Naive Bayes (NB), Decision Tree (DT), Neural Network (NN), Random Forest (RF) und Support Vector Machine (SVM). Die Algorithmen werden untereinander in statische Modelle oder Machine Learning Methoden unterteilt. Mit dieser Unterscheidung kann nachher beschlossen werden, welche Methoden für die Prognose bessere Ergebnisse erzielen.

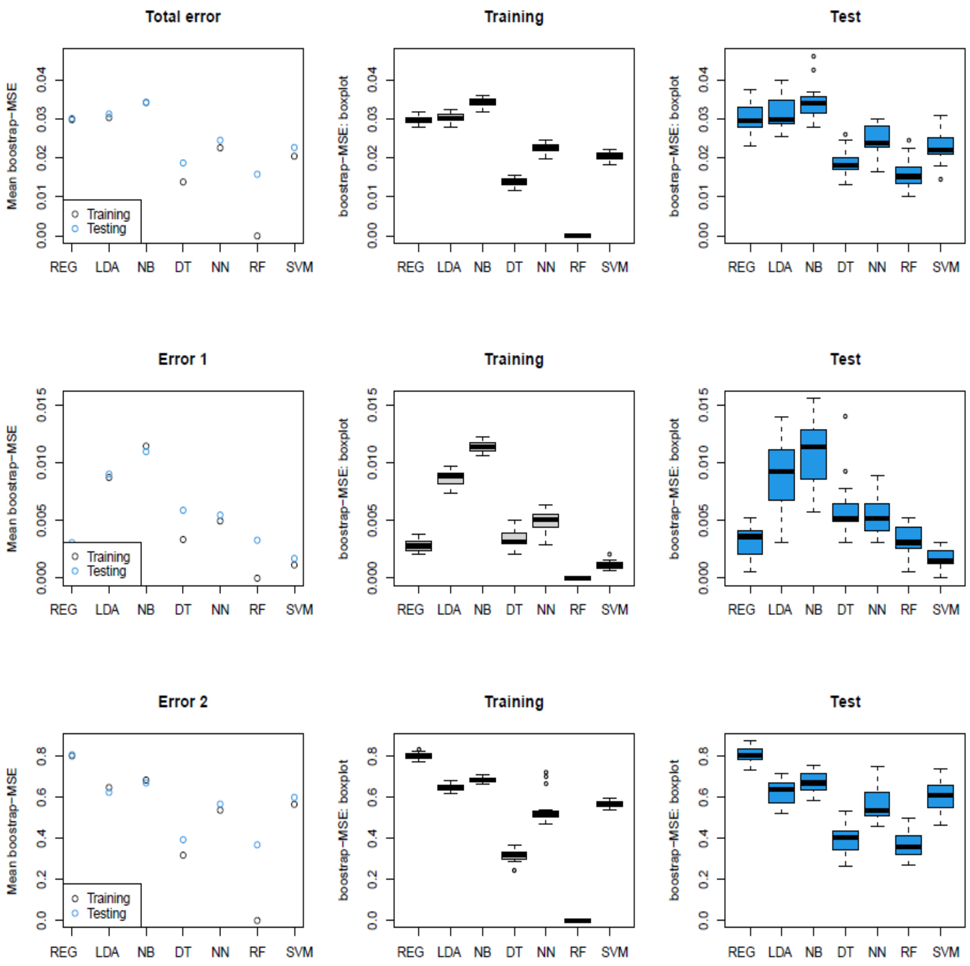

Abbildung 21: Erste Ergebnisse

Die Ergebnisse der Durchführung der Algorithmen sind auf der Abbildung 21 zu sehen. Die Algorithmen wurden mit 20 Wiederholungen durchgeführt. Deswegen zeigen alle Algorithmen 20 verschiedene Fehlerraten auf. Die Fehlerraten werden in den Boxplots verteilt. Mit dem Boxplot kann die Streuung dargestellt werden. Die Streuung besagt, wie weit die Fehlerraten um den Mittelwert herum verteilt sind. Wenn die 20 Fehlerraten ähnliche Werte haben, werden die Werte in dem Boxplot eng verteilt. Wenn die 20 Fehlerraten voneinander weitentfernte Werte haben, wird die Streuung in dem Boxplot breiter. Um die genauere Fehlerrate zu interpretieren, wird erwartet, dass die Streuung in dem Boxplot nicht breit ist.

Die ersten drei Plots zeigen die Ergebnisse für Total-Fehler. Die Punkte auf dem linken Plot sind die Mittelwerte der MSE für Training- und Testdaten. Die schwarzfarbigen Punkte zeigen den Mittelwert der Fehlerrate beim Training und die blaufarbigen Punkte sind Mittelwerte der MSE für Testing. Die anderen Boxplots zeigen, wie sich MSE bei allen Algorithmen sich verteilen. Die 20 MSE-Werte werden mit den Boxplots gezeigt. Die Mittelwerte dieser Fehlerrate sind als schwarze Linien bei den Boxplots zu sehen.

Bei dem Total-Fehler zeigt Random Forest die niedrigste Fehlerrate beim Training auf. Beim Test zeigt auch Random Forest die niedrige Fehlerrate auf. Die höchste Fehlerrate besteht bei dem Naive Bayes Algorithmus. Für die Prognose ist die Fehlerrate beim Test wichtig, weil die Testdaten von den Trainingsdaten unterschiedlich sind und der Algorithmus mit Testdaten nur die Betriebszustände prognostiziert.

Die anderen drei Plots, die in der Mitte zu sehen sind, zeigen die Prognose der betriebsbereiten Zustände. Hier weisen die Algorithmen sehr wenige Fehler auf, weil es in dem Datensatz viele Daten über betriebsbereite Zustände gibt und die Algorithmen deswegen bessere Prognosen vornehmen können. Logistische Regression prognostiziert nur mit 0,2 % Fehler die betriebsbereiten Zustände. Der SVM weist auch sehr wenige Fehlerrate für die Prognose auf. Wie bei Total-Fehler ergibt es in NB den höchsten Fehler bei der betriebsbereiten Zustandsprognose.

Die letzten drei Grafiken zeigen die Fehlerrate der ausgefallenen Zustandsprognose. Die Fehlerrate liegt sehr hoch, weil der Datensatz weinige Daten für ausgefallene Zustände hat. Deswegen haben die Algorithmen Schwierigkeiten die Zustände vorherzusagen. Beim Vergleich zwischen der betriebsbereiten Zustandsprognose und der ausgefallenen Zustandsprognose weist REG 80 % Fehler auf. RF und DT prognostizieren ausgefallene Zustände mit sehr wenigen Fehlern.

Für alle Zustandsprognosen erzielt RF gute Ergebnisse. SVM und REG prognostizieren gut den betriebsbereiten Zustand. Und beim Total-Fehler weisen RF und DT die wenigsten Fehler auf. Um die Fehlerprozente der Algorithmen zu minimieren, können die Algorithmen des maschinellen Lernens mit verschiedenen Hyperparametern nochmals durchgeführt werden. Mit Hilfe der Hyperparameter lernen die Algorithmen des maschinellen Lernens die Trainingsdatensätze neu und bewirken neue Ergebnisse. Um die Unterschiede zu sehen und die optimale Hyperparameter bestimmen zu können, werden die Hyperparameter bei Neural Network und Support Vector Machine geändert. Mit diesen Änderungen können die niedrigste Fehlerrate und die optimalen Parameter erlangt werden.

4.3.1 Bestimmung der optimalen Hyperparameter bei Neural Network

Bei den Algorithmen des maschinellen Lernens können die Hyperparameter des Algorithmus geändert werden, damit der Algorithmus den Datensatz besser verstehen und durchführen kann. Für NN sind die Neuronen der wichtigste Hyperparameter. Deswegen werden die Zahlen der Neuronen geändert und untersucht, mit wie vielen Neuronen NN die beste Prognose erlangt. In der Kapitel 4.3 wurde der Algorithmus mit zwei Neuronen durchgeführt. Diese Durchführung ist der erste Zustand. Alle anderen Ergebnisse werden mit diesem Zustand verglichen, um zu sehen, ob sich die Fehlerrate ändern wird. NN-Algorithmus wird erst mit drei Neuronen durchgeführt, damit der Unterschied gesehen werden kann. Die Änderung der Neuronen erfolgt durch den Parameter *„hidden“. „Hidden*“ wurde in Help R als „ein Vektor aus ganzen Zahlen, der die Anzahl der verborgenen Neuronen (Vertices) in jeder Schicht angibt“ [24], erklärt. Die Parameteränderung erfolgt z.B. mit *h=c(3).* Das heißt, dass dieser NN Algorithmus mit 3 Neuronen ausgeführt wird. Die Neuronen-Anzahl muss nicht nur einen Wert haben. Sie können auch z.B. mit *h=c(5,2)* durchgeführt werden. Mit dieser Parameteränderung klassifiziert der Algorithmus die Betriebszustände erst mit fünf Neuronen dann nochmals mit zwei Neuronen. Auf der nächsten Abbildung 22 werden die NN Ergebnisse gesehen, die mit drei Neuronen durchgeführt werden.

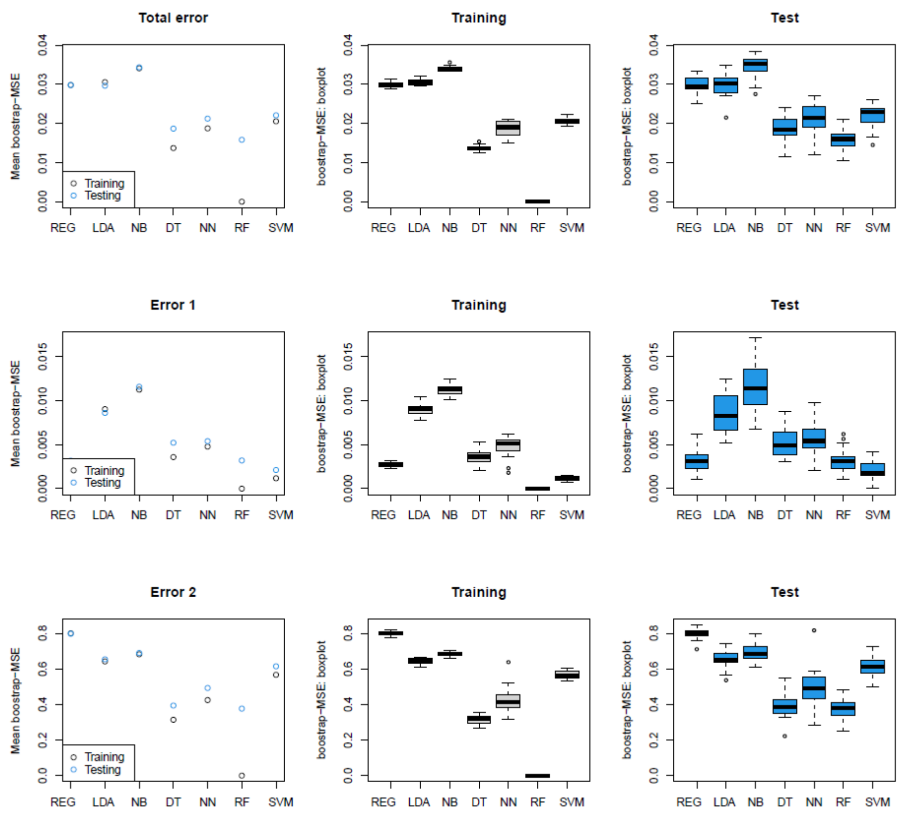

Abbildung 22: Untersuchung des NN Algorithmus mit drei Neuronen

Wenn die Ergebnisse verglichen werden, wird beobachtet, dass der Mittelwert der Fehlerrate beim Training mit drei Neuronen gesunken ist. Beim Testing sind die MSE-Werte sehr gering gesunken. Für Total-Fehler hat der Algorithmus NN beim Testing mit zwei Neuronen 0,022 MSE mit drei Neuronen 0,019 MSE. Das heißt, dass der Algorithmus für Total-Fehler mit zwei Neuronen 2,2% Testfehler und mit drei Neuronen 1,9% Testfehler aufweist.

Um den Unterschied der Änderungen genauer zu sehen, wird das Skript geändert (s. Anhang D). Mit den neuen Durchführungen können alle Änderungen der Neuronen genauer erkannt werden. Alle Boxplots zeigen die Ergebnisse der NN, die mit verschiedenen Zahlen der Neuronen durchgeführt werden. Weil die Durchführungszeiten des Skripts bei den Algorithmen des Maschinellen Lernens sehr hoch sind, wurde der Parameter „threshold“ geändert. „threshold“ ist ein nummerischer Wert, der die Schwelle für die partiellen Ableitungen der Fehlerfunktion als Abbruchkriterium angibt [24]. Je kleiner threshold ist, desto genauer sind die Ergebnisse. Aber mit sehr kleineren threshold-Werte dauert die Durchführungszeit sehr lang. Der NN Algorithmus wird in der Regel mit threshold=0,01 durchgeführt, damit die Ergebnisse genauer sind.

Auf der Abbildung werden verschiedene NN-Prognoseergebnisse gesehen. Die Nummer unter den Boxplots zeigen die geänderten Parameter. Die Algorithmen werden mit *threshold=0,1* durchgeführt. Die Parameter sind so aufgelistet:

- 1: *h=c(1)* :1 Neuron
- 2: *h=c(2)* :2 Neuronen
- 3: *h=c(3)* :3 Neuronen
- 4: *h=c(4)* :4 Neuronen
- 5: *h=c(5)* :5 Neuronen
- 6: *h=c(6)* :6 Neuronen
- 7: *h=c(7)* :7 Neuronen

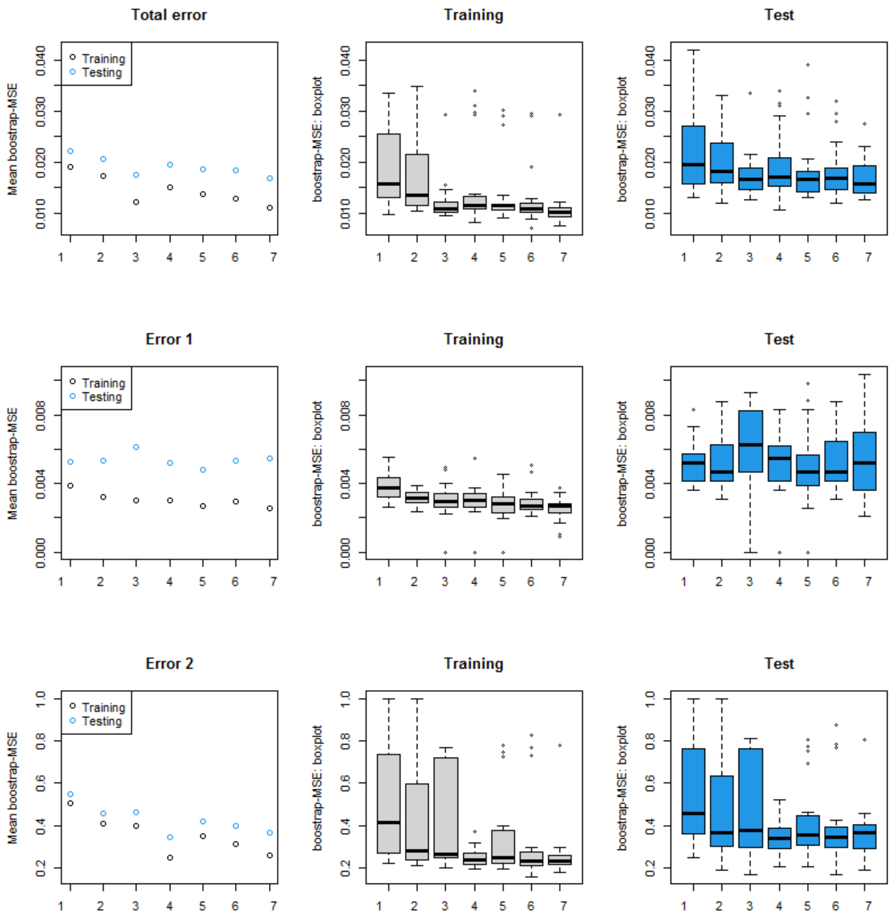

Abbildung 23: Untersuchung der verschiedenen Neuronenvariation 1

Die Fehlerquoten sinken mit der Erhöhung der Anzahl der Neuronen. Das optimale Ergebnis beim Training wird erzielt, wenn die Fehlerquote immer sinkt. Auch beim Testing wird verlangt, dass die Fehlerquote zuerst sinkt und dann an einem bestimmten Punkt wieder ansteigt. Der niedrigste Punkt zeigt das optimale Ergebnis an. Dieser Abstieg und Aufstieg sollten eine U-Form aufweisen. Auf dieser Abbildung 23 weisen die Ergebnisse nicht viele Unterschiede auf, deswegen wird der Algorithmus weiterhin mit verschieden Neuronen durchgeführt, um den optimalen Parameter zu finden.

Die wenigsten Fehler ergeben sich mit 5,6 und 7 Neuronen. Die Neuronen können auch mit den anderen Neuronen kombiniert werden. Z.B. kann der Algorithmus erst mit 7 Neuronen dann mit 2 Neuronen arbeiten. Die Algorithmen werden mit *threshold=0,5* durchgeführt. Die neuen geänderten Parameter bei den Algorithmen sind:

- 1: *h=c(5)*
- 2: *h=c(5,2)*
- 3: *h=c(5,3)*
- 4: *h=c(6)*
- 5: *h=c(6,2)*
- 6: *h=c(7)*
- 7: *h=c(7,2)*

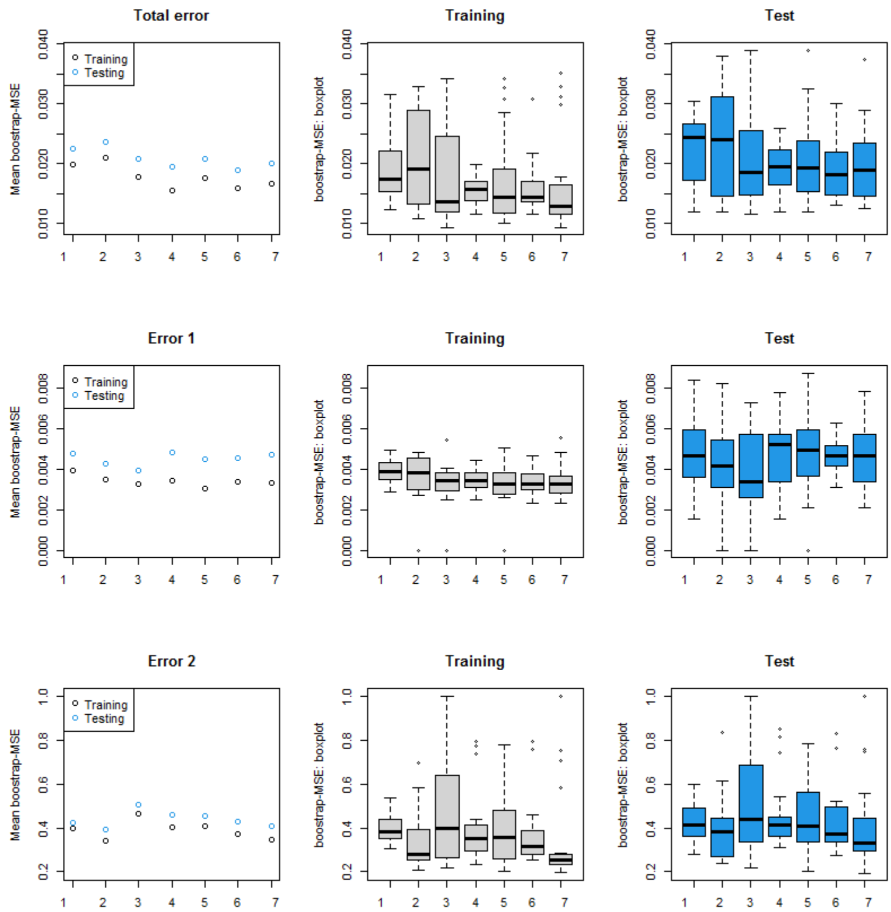

Abbildung 24: Untersuchung der verschiedenen Neuronenvariation 2

Hier wurden die verschiedenen Kombinationen der Neuronen ermittelt. Wenn NN neue Neuronenkombination besitzt z.B. zuerst 5 Neuronen dann 3 Neuronen, dann erlangt der Algorithmus bessere Ergebnisse. Bis jetzt wurden die Algorithmen maximal mit 7 Neuronen durchgeführt. Um den Unterschied bis 9 Neuronen zu sehen, wurde das Skript erst von 5 Neuronen bis 9 Neuronen durchgeführt. Zusätzlich wurden auch die Ergebnisse für die Kombination 5 Neuronen mit 3 Neuronen und 6 Neuronen mit 3 Neuronen verglichen. Alle Algorithmen werden mit *threshold=0,5* durchgeführt. Die neuen Parameter sind wie folgt:

- 1: *h=c(5)*
- 2: *h=c(6)*
- 3: *h=c(7)*
- 4: *h=c(8)*
- 5: *h=c(9)*
- 6: *h=c(5,3)*
- 7: *h=c(6,3)*

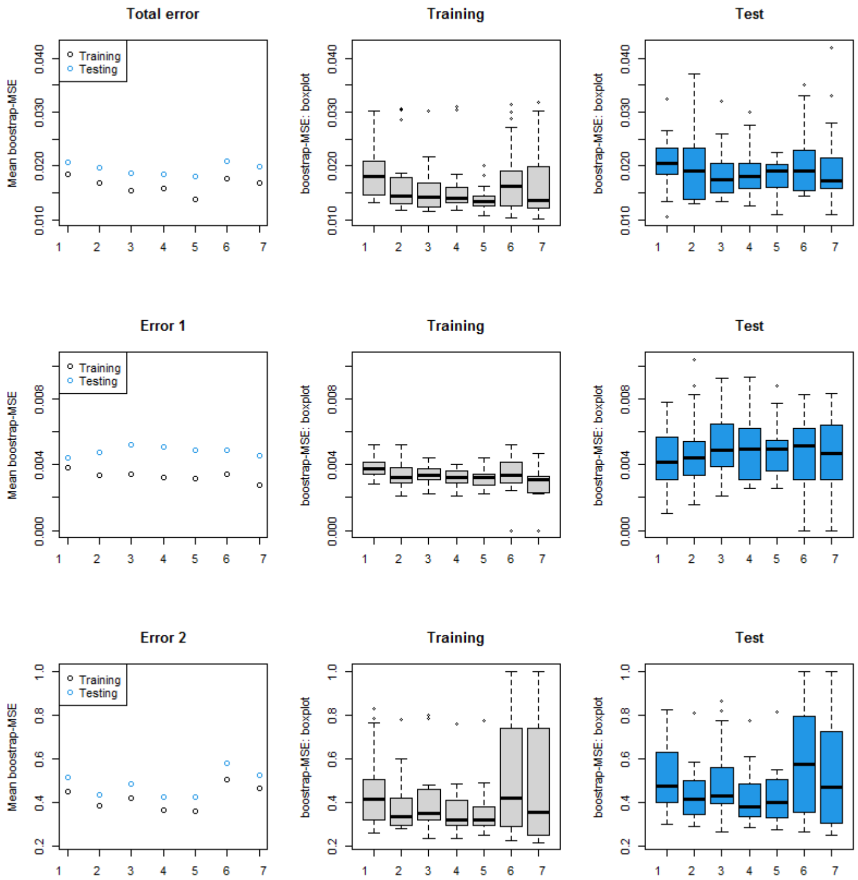

Abbildung 25: Untersuchung der verschiedenen Neuronenvariation 3

Die Trainingsfehler für die gesamten Fehler sind stätig von 5 Neuronen bis 7 Neuronen gesunken, aber die Testfehler für den gesamten Fehler sind erst bis 7 Neuronen gesunken und dann gestiegen. Für die optimale Fehlerrate kann erkannt werden, dass NN mit den 7 Neuronen eine bessere Prognose erzielt. Beim Vergleich der Kombination zwischen 5 und 3 Neuronen, 6 und 3 Neuronen wird ersichtlich, dass NN mit 6 und 3 Neuronen bessere Prognosen erzielt.

Um die Ergebnisse besser zu begreifen, wird das Skript nochmals von 3 Neuronen bis 9 Neuronen durchgeführt. Damit soll festgestellt werden, dass die Trainingsfehler immer sinken, wenn die Zahl der Neuronen steigt und die Testfehler eine U-Form haben. Alle Algorithmen werden mit *threshold=0,5* durchgeführt. Die geänderten Parameter sind:

- 1: *h=c(3)*
- 2: *h=c(4)*
- 3: *h=c(5)*
- 4: *h=c(6)*
- 5: *h=c(7)*
- 6: *h=c(8)*
- 7: *h=c(9)*

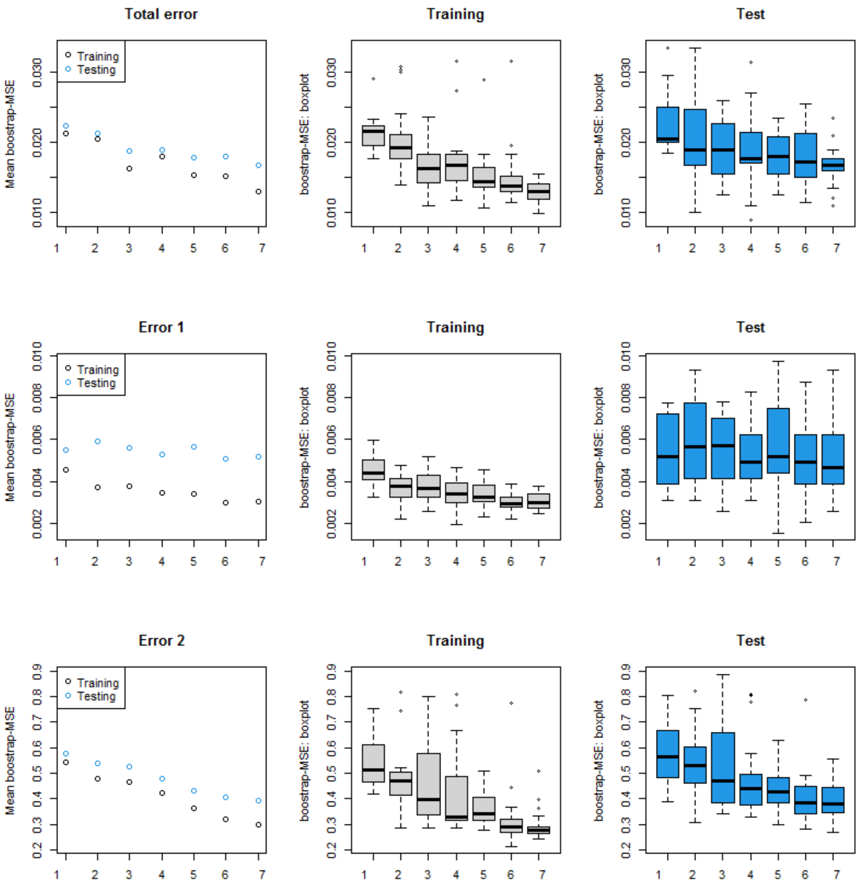

Abbildung 26: Untersuchung der verschiedenen Neuronenvariation 4

Die Ergebnisse sind auf der Abbildung 26 zu sehen. Die Algorithmen des maschinellen Lernens lernen immer nach Zufallsprinzip. Deswegen ändert sich die Fehlerrate immer ein wenig, wenn das Skript nochmals durchgeführt wird. Wie schon erwartet wurde, sind die Trainingsfehler mit der Erhöhung der Anzahl der Neuronen weiter gestiegen. Die Testfehler haben keine richtige U-Form. Deswegen wird die Zahl der Neuronen nochmals geändert, um zu sehen, wie die Ergebnisse sich verhalten, wenn die Algorithmen von 5 Neuronen bis 11 Neuronen erhöht werden. Alle Algorithmen werden mit *threshold=0,5* durchgeführt. Die neuen Parameter sind:

- 1*: h=c(5)*
- 2: *h=c(6)*
- 3: *h=c(7)*
- 4: *h=c(8)*
- 5: *h=c(9)*
- 6: *h=c(10)*
- 7: *h=c(11)*

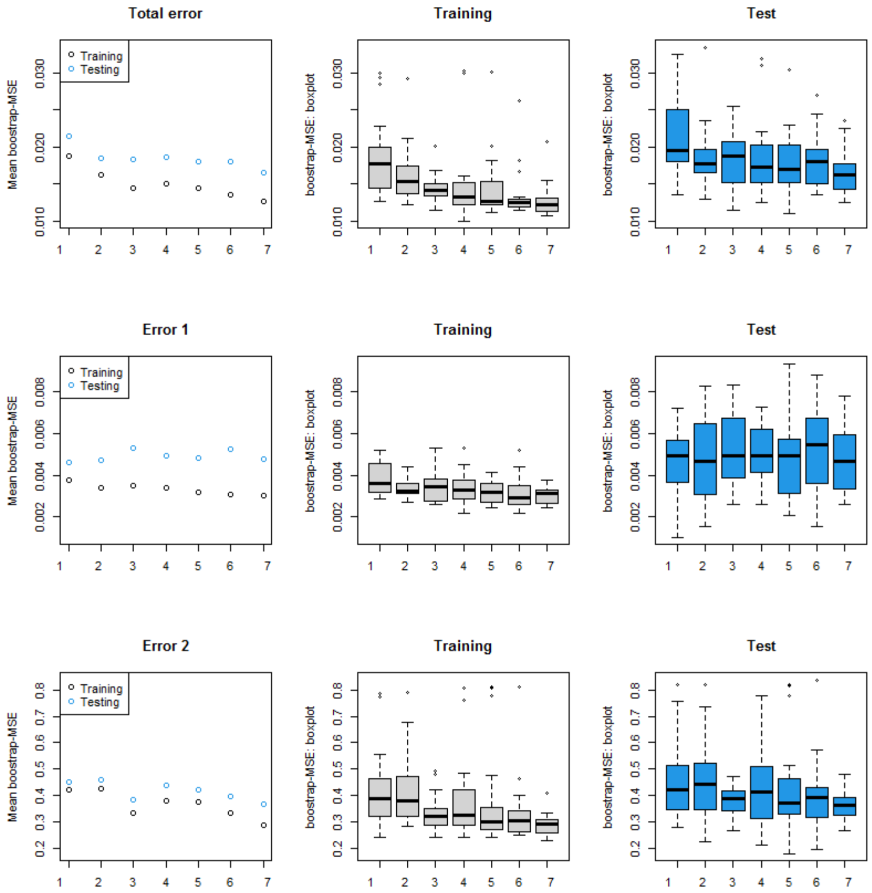

Abbildung 27: Untersuchung der verschiedenen Neuronenvariation 5

Es ist sehr schwer festzustellen, mit wie vielen Neuronen die Algorithmen bessere Ergebnisse erhalten. Weil es viele Wahrscheinlichkeiten für die Auswahl der Neuronen-Anzahl gibt, werden die Algorithmen nochmals mit verschiedenen Neuronen durchgeführt.

Weil die Ergebnisse gezeigt haben, dass die Fehlerraten mit den Kombinationen der Neuronen niedriger sind, werden die Algorithmen nochmals mit verschiedenen Neuronen-Kombinationen gestaltet. Hier werden die Algorithmen erst mit verschiedenen Neuronen und dann nochmal mit 3 Neuronen durchgeführt. Diese Algorithmen werden mit *threshold=0,5* durchgeführt. Die neuen Parameter sind:

- 1: *h=c(5,3)*
- 2: *h=c(6,3)*
- 3: *h=c(7,3)*
- 4: *h=c(8,3)*
- 5: *h=c(9,3)*
- 6: *h=c(10,3)*
- 7: *h=c(11,3)*

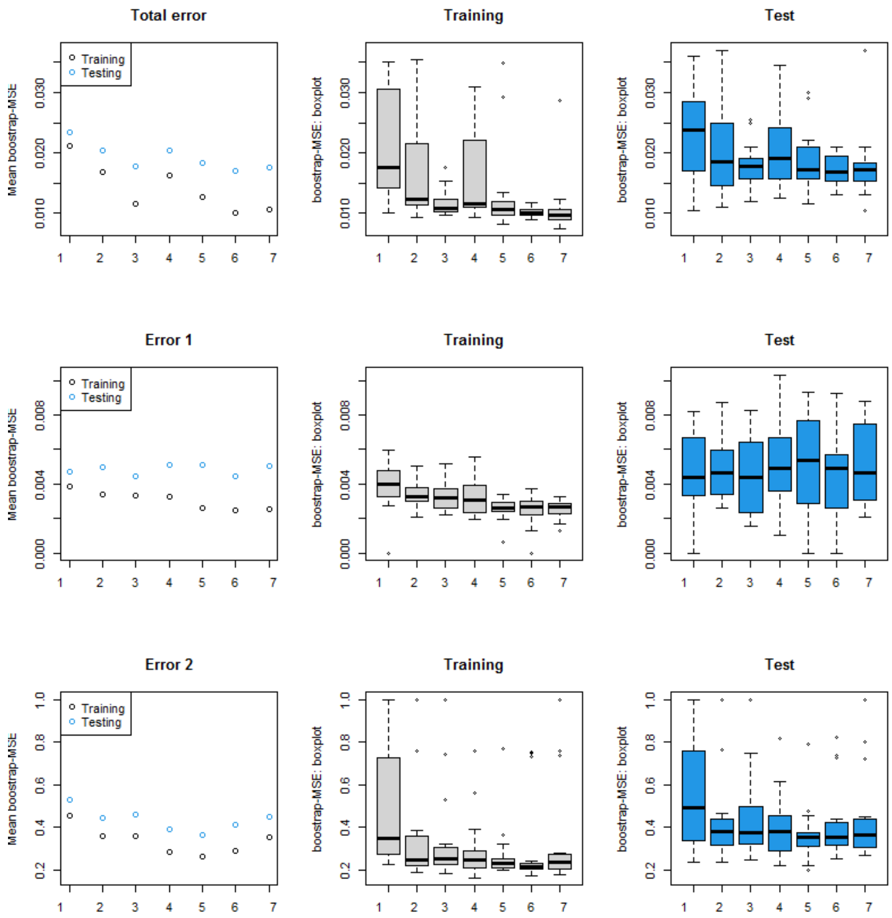

Abbildung 28: Untersuchung der verschiedenen Neuronenvariation 6

Der Algorithmus ergibt mit 7 und 3 Neuronen die optimale Fehlerquote. Der Algorithmus zeigt beim Total-Fehler 1,1 % Trainingsfehler und 1,7 % Testfehler auf. Für den betriebsbereiten Zustand hat die Prognose 0,03 % Trainingsfehler und 0,04 % Testfehler. Der Algorithmus prognostiziert den ausgefallenen Zustand mit 35 % Trainingsfehler und 45 % Testfehler. Um den Unterschied zwischen 7 Neuronen und 8 Neuronen zu sehen werden die Algorithmen erst mit der Kombination der 7 Neuronen und dann mit der Kombination der 8 Neuronen durchgeführt.

Die erste Durchführung findet mit verschiedenen Kombinationen der 7 Neuronen statt. Diese Algorithmen werden mit *threshold=0,5* durchgeführt. Die neuen Parameter sind:

- 1: *h=c(7)*
- 2: *h=c(7,2)*
- 3: *h=c(7,3)*
- 4: *h=c(7,4)*
- 5: *h=c(7,5)*
- 6: *h=c(7,6)*
- 7: *h=c(7,7)*

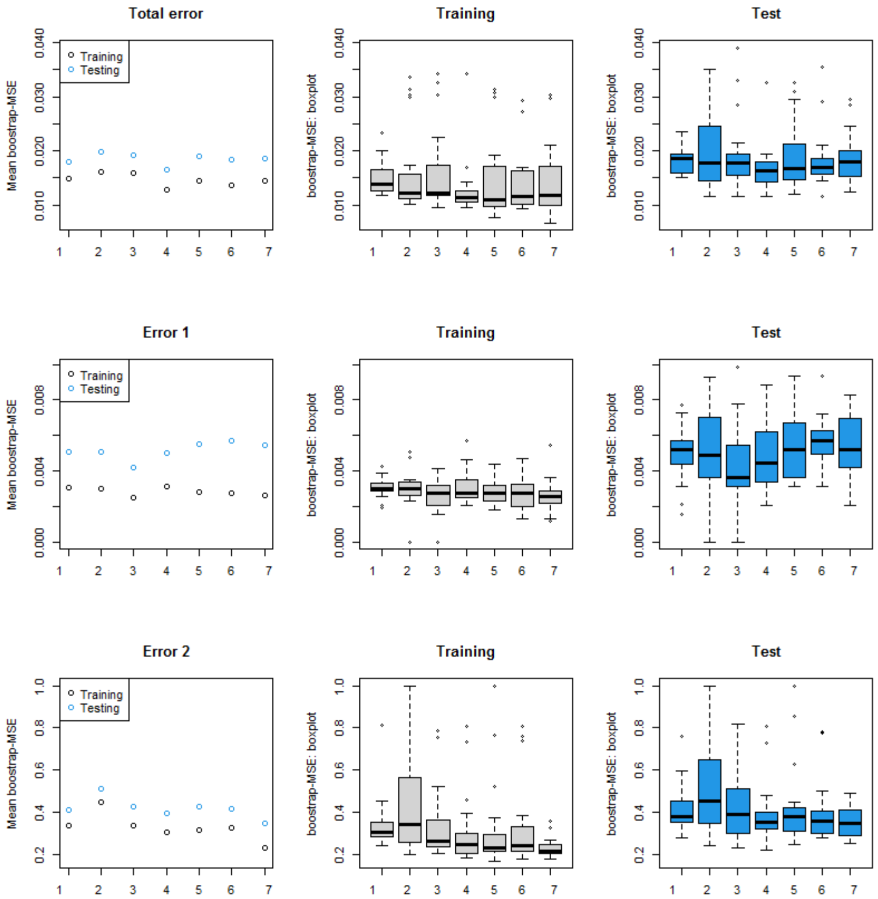

Abbildung 29: NN Parametern Kombinationen mit 7 Neuronen

Aus Abbildung 29 ist ersichtlich, dass NN seine besten Ergebnisse mit 7 und 4 Neuronen erzielt. Besonders für Total-Fehler und ausgefallenen Zustand hat der Algorithmus mit 7 und 4 Neuronen die geringste Fehlerrate. Die Fehlerraten verändern sich sehr wenig. Total-Fehler und Fehler des ausgefallenen Zustands haben eine geringe Fehlerrate mit 7 und 4 Neuronen als mit 7 und 3 Neuronen. Weil die Fehlerquote für Total-Fehler geringer ist, wird ausgesagt, dass NN Algorithmus am besten mit 7 und 4 Neuronen die besten Ergebnisse erzielt. Um den Unterschied der Kombinationen mit 8 Neuronen zu sehen, werden die NN Algorithmen nochmals ausgeführt. Die neuen Parameter sind:

- 1: *h=c(8)*
- 2: *h=c(8,2)*
- 3: *h=c(8,3)*
- 4: *h=c(8,4)*
- 5: *h=c(8,5)*
- 6: *h=c(8,6)*
- 7: *h=c(8,7)*

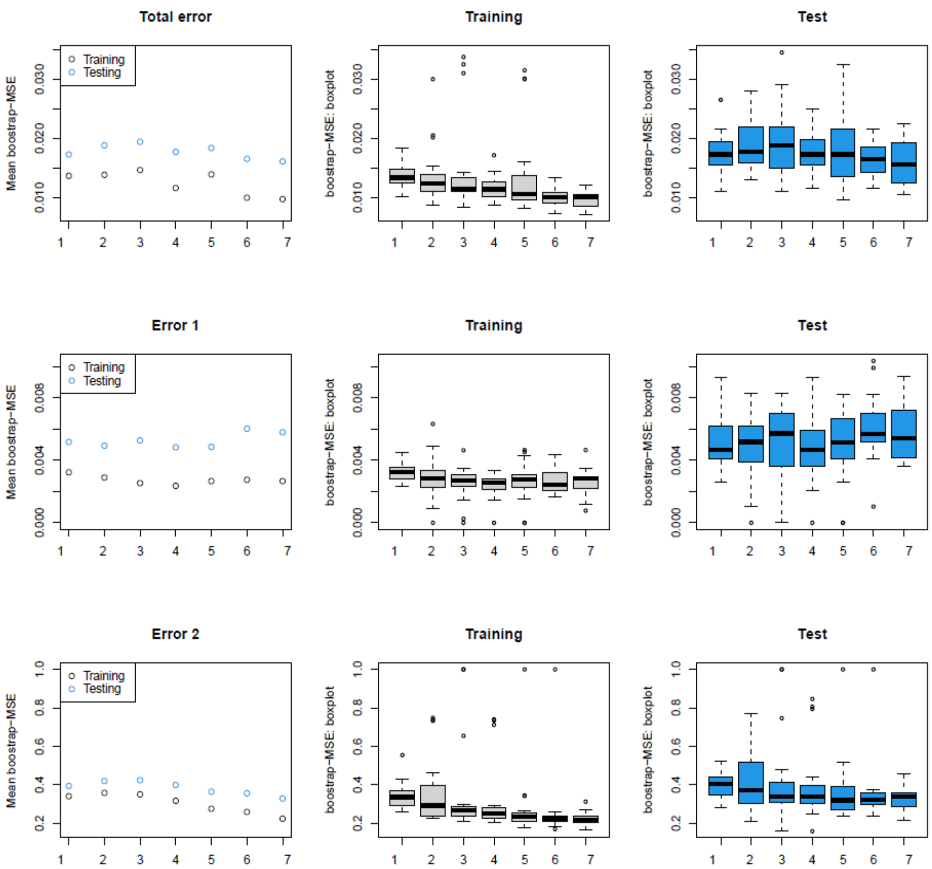

Abbildung 30: NN Parametern Kombinationen mit 8 Neuronen

Die optimalen Fehlerprozente ergibt der Algorithmus mit 8 und 4 Neuronen. Hier bekommt der Algorithmus beim Total-Fehler 1,1 % Trainingsfehler und 1,7 % Testfehler. Der Algorithmus zeigt für den betriebsbereiten Zustand 0,2 % Trainingsfehler und 0,4 % Testfehler und für den ausgefallenen Zustand 31 % Trainingsfehler und 39 % Testfehler auf.

Die endgültige Entscheidung wird zwischen den Parametern 7 und 4 Neuronen und 8 und 4 Neuronen getroffen. Für diese Entscheidungen werden die beiden NN Algorithmen mit den anderen Algorithmen nochmals durchgeführt und die Fehlerrate miteinander und besonders mit dem ersten Zustand verglichen. Für das erste Ergebnis wird NN mit dem Parameter *h=c(7,4)* und mit *threshold=0,2* durchgeführt.

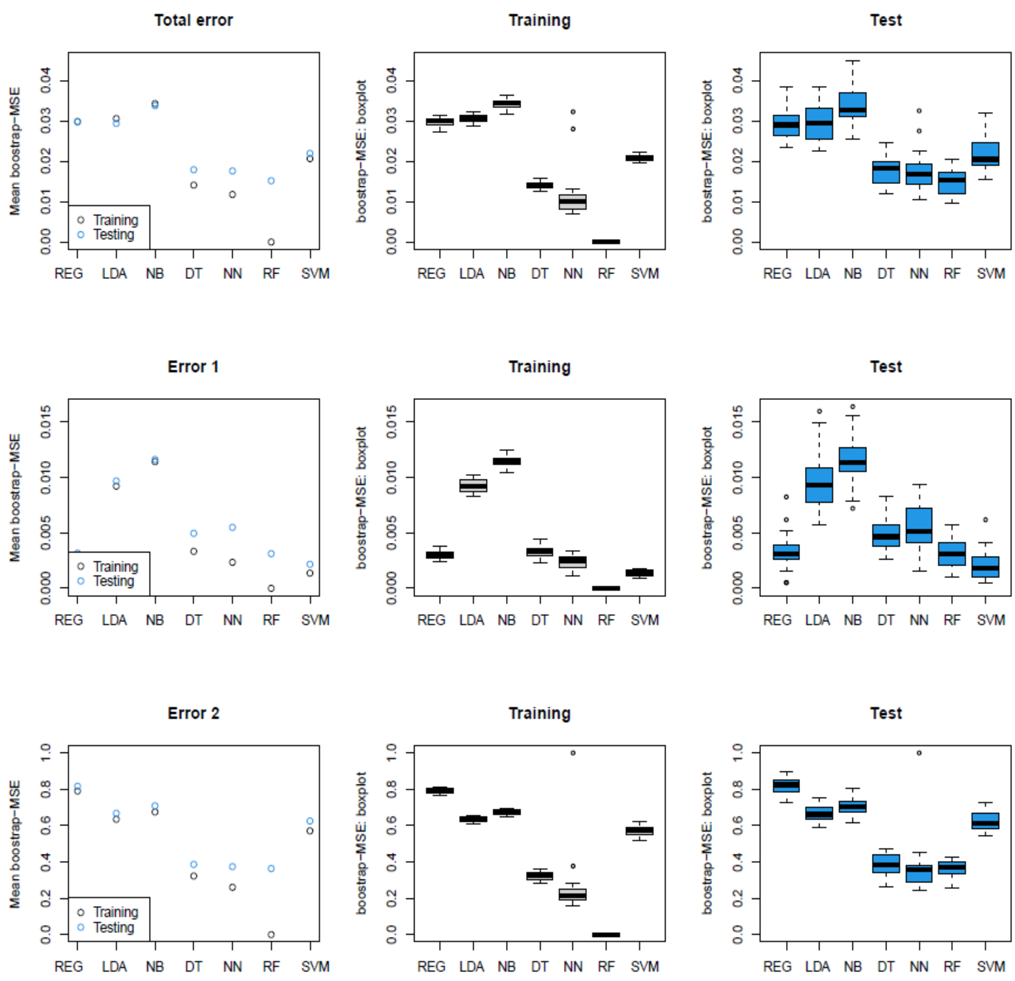

Abbildung 31: Alle Ergebnisse mit *h=(7,4)*

Für den Total-Fehler bekommt NN mit 7 und 4 Neuronen 1,1 % Trainingsfehler und 1,7 % Testfehler. Jedoch weist dieser Algorithmus 0,02 % Trainingsfehler und 0,05 % Testfehler für den betriebsbereiten Zustand und 26 % Trainingsfehler und 37 % Testfehler für den ausgefallenen Zustand auf.

Für einen anderen Vergleich wird der NN Algorithmus mit dem Parameter *h=c(8,4)* und mit *threshold=0,2* durchgeführt. Die Ergebnisse mit 8 und 4 Neuronen bei NN Algorithmus sind auf der Abbildung 25.

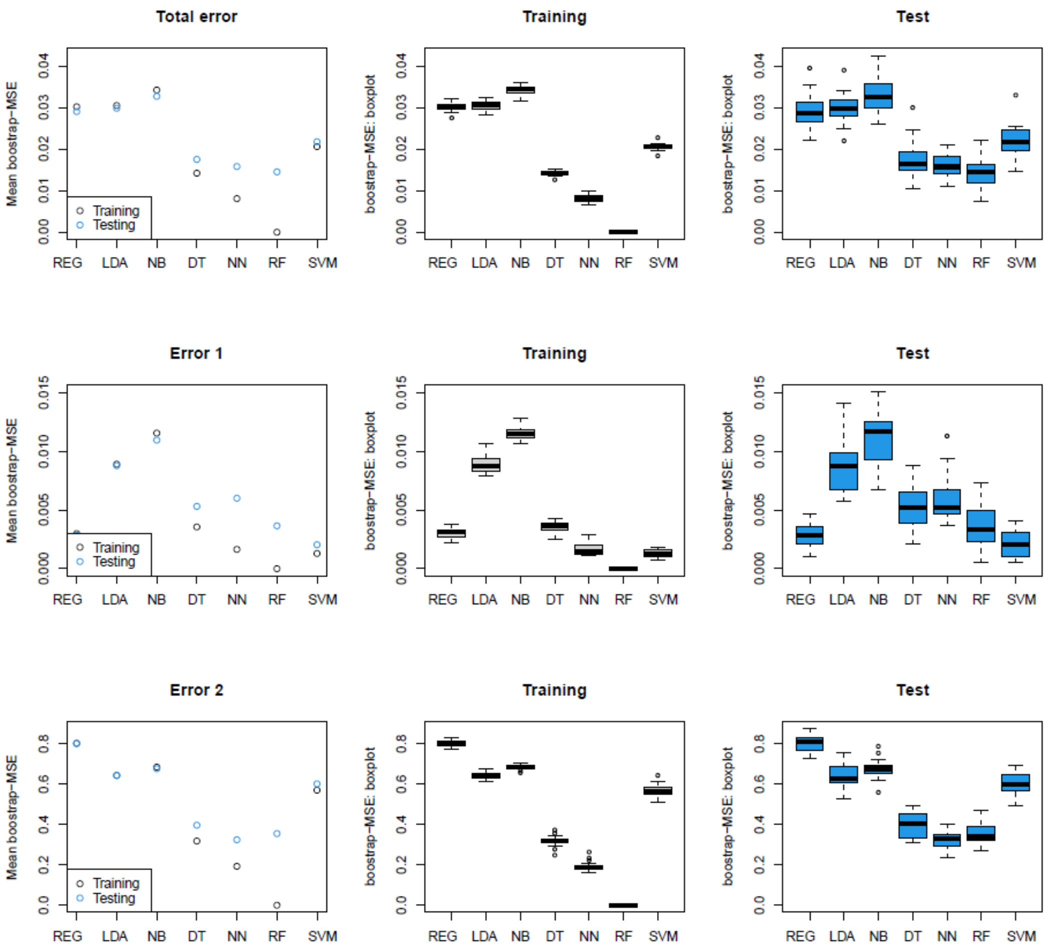

Abbildung 32: Alle Ergebnisse mit *h=(8,4)*

Die optimalen Fehlerquoten ergibt der NN Algorithmus mit 8 und 4 Neuronen. Für Total- Fehler zeigt der NN Algorithmus mit 8 und 4 Neuronen 0,8 % Trainingsfehler und 1,5 % Testfehler. Jedoch weist dieser Algorithmus 0,01 % Trainingsfehler und 0,05 % Testfehler für den ausgefallenen Zustand und 19 % Trainingsfehler und 32 % Testfehler für den ausgefallenen Zustand auf.

4.3.2 Bestimmung der optimalen Hyperparameter bei Support Vector Machine

Der Algorithmus Support Vector Machine hat zwei Hyperparameter, damit der Algorithmus eine bessere Prognose erzielen kann. Der erste Hyperparameter ist „*cost*“. Cost ist die „C“-Konstante des Regularisierungsterms in der Lagrange-Formulierung [25]. Der andere Hyperparameter ist „*gamma*“. Gamma ist ein Hyperparameter, der mit nichtlinearer SVM verwendet wird [25]. Ohne Parameteränderung weist SVM beim Total-Fehler 2 % Trainingsfehler und 2,1 % Testfehler, bei dem betriebsbereiten Zustand 0,01 % Trainingsfehler und 0,02 % Testfehler und bei dem ausgefallenen Zustand 57 % Trainingsfehler und 60 % Testfehler auf.

Das Skript wird neu geschrieben (s. Anhang E), damit die Ergebnisse der Parameteränderung genauer gestaltet werden können. Erst werden die Cost-Parameter bei SVM geändert. Für diese Durchführung haben die SVM-Algorithmen verschiedene Cost-Parameter. Diese Parameter sind *cost=0,1/ cost=1/ cost=5/ cost=10/ cost=20/ cost=50/ cost=100*. Mit dieser Änderung will beobachtet werden, welche Cost-Parameter die optimale Fehlerrate ergeben. Unter den Boxplots stehen die Werte der Cost-Parameter.

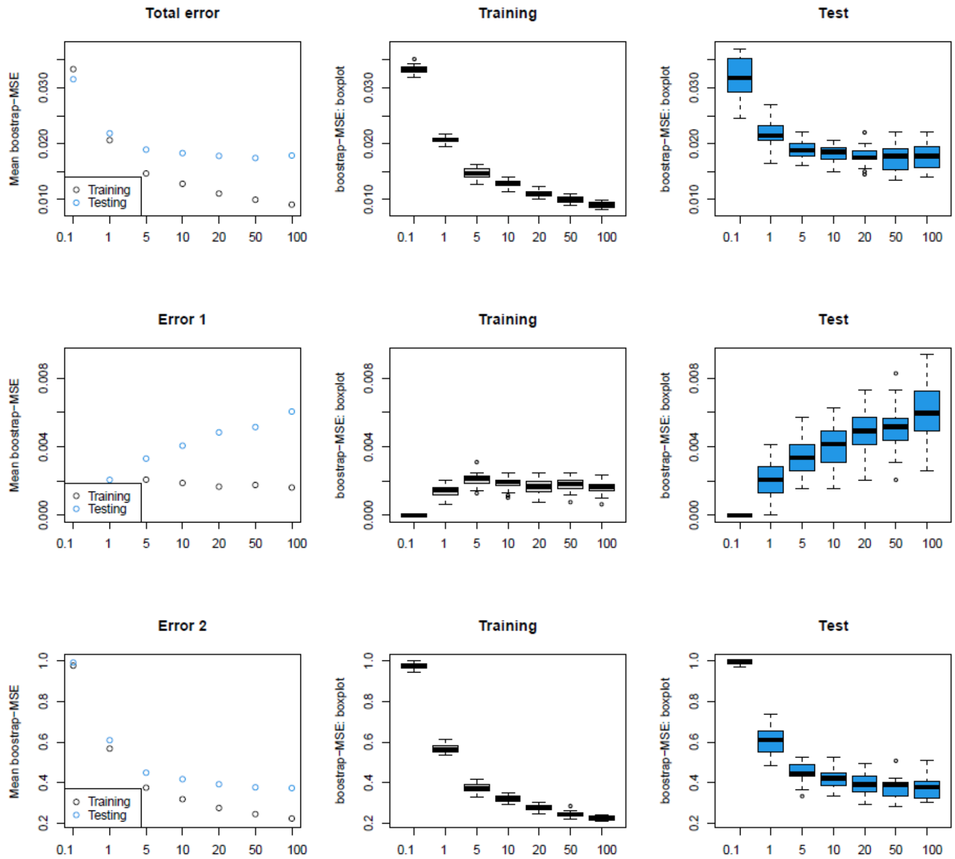

Abbildung 33: Varianten der Cost-Parameter

Um einen optimalen Wert zu erreichen, sollte die Fehlerrate beim Training immer sinken und beim Testing eine U-Form annehmen. Aus Abbildung 33 wird ersichtlich, dass die Trainingsfehler beim Total-Fehler stätig sinken. Aber die Testfehler sinken nur bis zu einem gewissen Punkt. Um die U-Form zu erreichen, werden die Algorithmen nochmals mit verschiedenen Cost-Parametern durchgeführt. Für diese Durchführung haben die Algorithmen die Parameter *cost=100/ cost=200/cost=300 / cost=500/ cost=600 / cost=800/ cost=1000.*

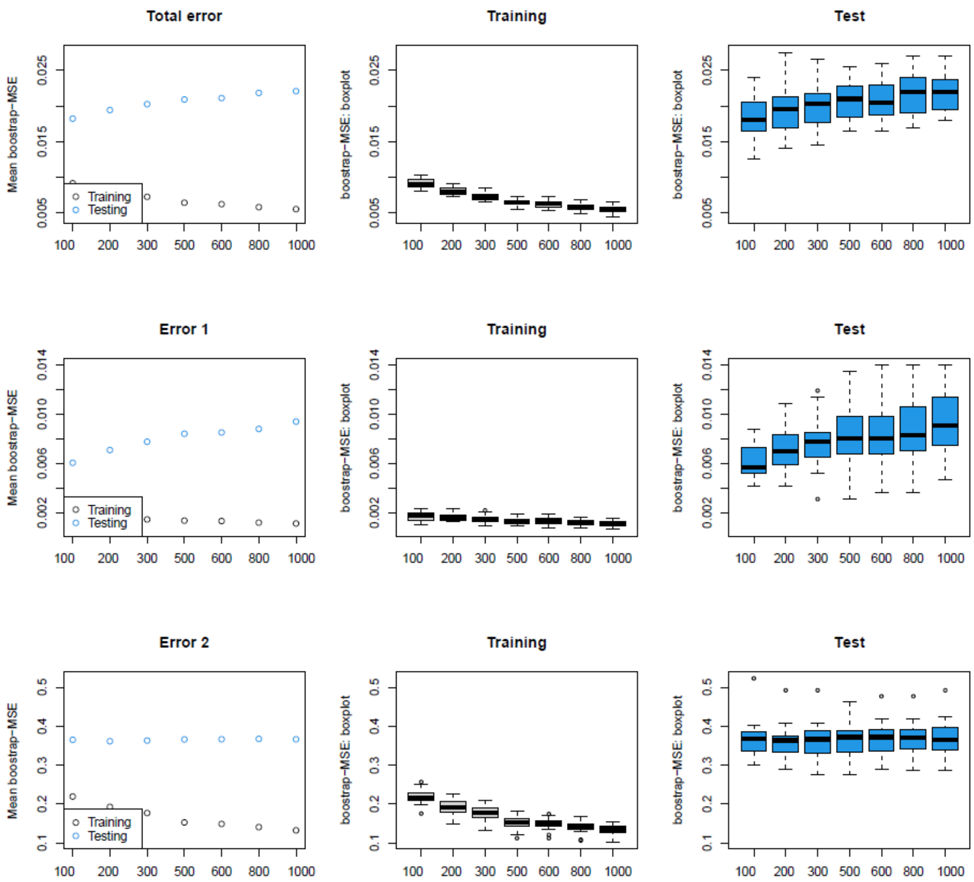

Abbildung 34: Varianten der Cost-Parameter 2

Die Trainingsfehler sind gesunken, aber die Testfehler sind gestiegen. Änderungen aus der ersten und zweiten Durchführung können kombiniert werden. Da das zweite Ergebnis eine Fortsetzung des ersten Ergebnisses ist, sollte die neue Durchführung helfen, die optimale Fehlerrate zu finden. Die Algorithmen werden mit den Parametern *cost=0,1/ cost=1/ cost=10/ cost=50/ cost=100/ cost=500/ cost=1000* durchgeführt.

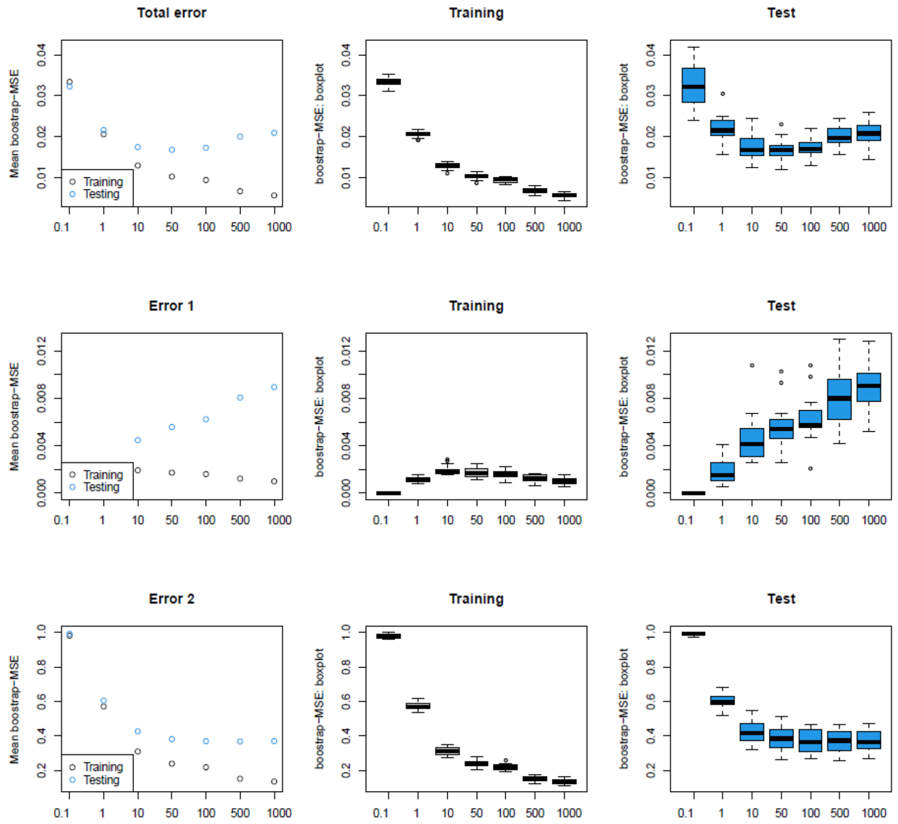

Abbildung 35: Varianten der Cost-Parameter 3

Wenn der SVM-Algorithmus mit dem Parameter *cost=100* durchgeführt wird, wird die optimale Fehlerquote gefunden. Mit den zwei Cost-Parametern 50 und 100 prognostiziert der Algorithmus ähnliche Fehlerraten. Deswegen sollte der Parameter *cost=50* für die anderen Änderungen berücksichtigt werden. Der SVM-Algorithmus mit *cost=100* liefert 0,9 % Trainingsfehler und 1,7 % Testfehler beim Total-Fehler, 0,1 % Trainingsfehler und 0,6 % Testfehler bei dem betriebsbereiten Zustand und 21 % Trainingsfehler und 37 % Testfehler beim ausgefallenen Zustand.

Die Fehlerrate für die Prognose ist für den Total-Fehler und für den ausgefallenen Zustand gesunken, aber für betriebsbereiten Zustand ist die Fehlerrate gestiegen. Deswegen wird die Durchführung erst nur mit der Änderung des Gamma-Parameters fortgesetzt. Danach werden die beiden Parameteränderungen kombiniert und es wird versucht, eine geringe und optimale Fehlerrate zu erzielen.

Um den optimalen Fehlerquote zu finden, werden die SVM-Algorithmen mit verschiedenen Gamma-Parameter durchgeführt (s. Anhang F). Die Parameter lauten *gamma= 0,01/ gamma=0,1/ gamma=0,5/ gamma=0,6/ gamma=0,8/ gamma=1/ gamma=5.*

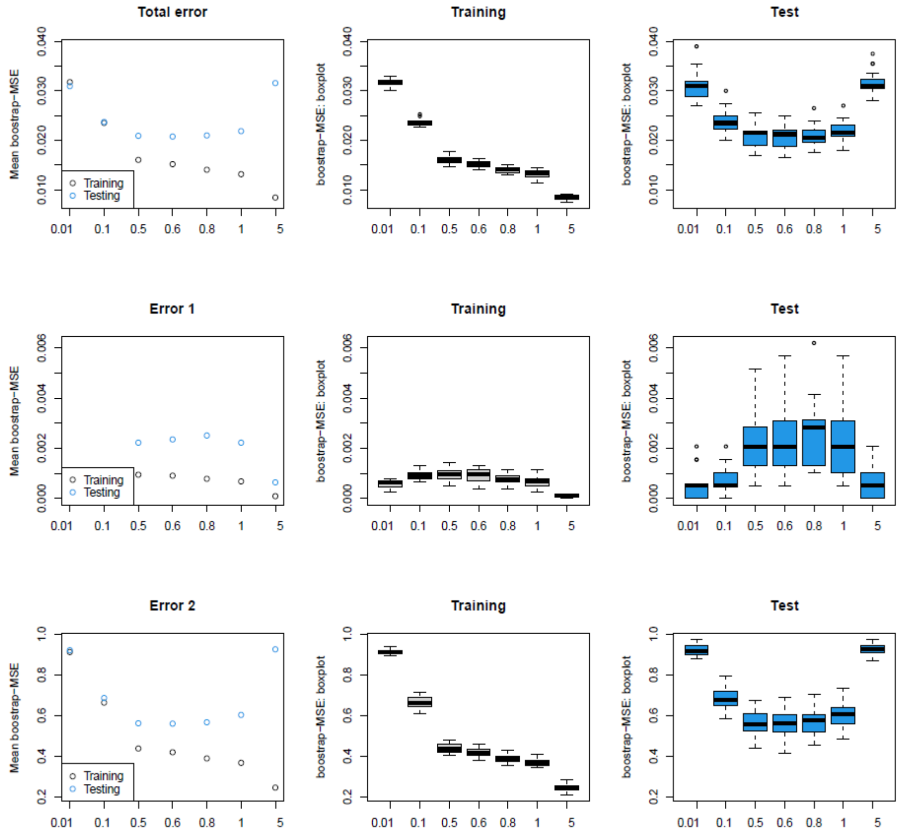

Abbildung 36: Varianten der Gamma-Parameter

Für den Total-Fehler und die ausgefallene Zustandsprognose weist der Algorithmus mit *gamma=0,8* die beste Fehlerrate auf, aber beim betriebsbereiten Zustand weist dieser Parameter mehr Testfehler als die anderen Varianten auf. Weil die Prognose für den ausgefallenen Zustand schwieriger ist, sollten die Ergebnisse des ausgefallenen Zustands mehr berücksichtigt werden. SVM prognostiziert gut, wenn der Algorithmus mit dem Gamma-Parameter durchgeführt wird. Besonders die Werte zwischen 0,5 und 1 erzielen bessere Prognosen mit wenigen Fehlern.

Um die optimale Fehlerquote zu finden, werden die beiden Parameter zusammen ausgeführt. Dafür werden erst die Gamma-Parameter *gamma=0,6, gamma=0,8* und *gamma=1* und die Cost-Parameter *cost=50* und *cost=100* vorgenommen. Die erste Durchführung wird mit den Parametern *cost=100* und *gamma=0,8* durchgeführt, weil die beiden Parameter separat die optimale Fehlerrat erreichen können.

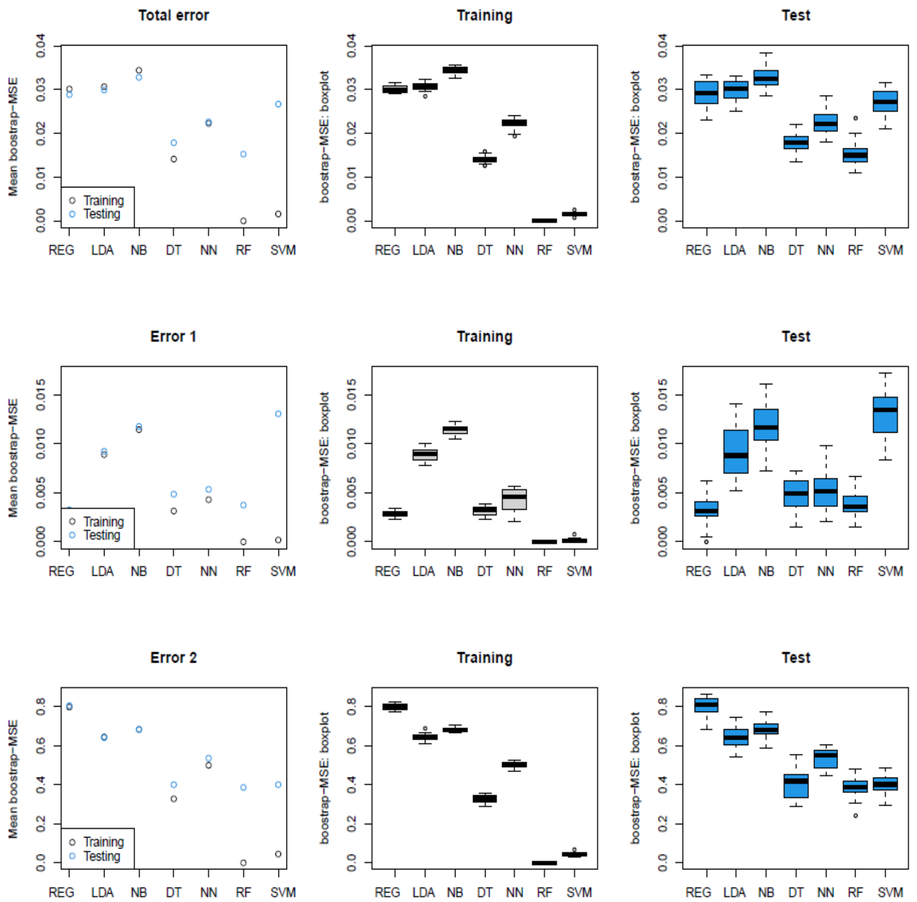

Abbildung 37: Untersuchung mit den Parametern *cost=100* und *gamma=0,8*

Nach der Kombination der zwei Hyperparameter des SVM-Algorithmus ist die Fehlerrate des Trainings viel gesunken, aber die Fehlerrate des Tests ist beim Total-Fehler und beim betriebsbereiten Zustand gestiegen. Ohne eine Parameteränderung weist der Algorithmus für den Total-Fehler 2,2 % Testfehler und für den betriebsbereiten Zustand 0,1 % Testfehler auf. Mit der Änderung der Parameter liegen die Prozente bei 2,6 % und 1,3 %. Beim ausgefallenen Zustand zeigt der Algorithmus 61 % Testfehler, wenn der Algorithmus keine Parameteränderung durchführt. Aber mit den zwei Parameteränderungen hat der Algorithmus nur 40 % Testfehler. Das heißt, dass der Algorithmus eine bessere Prognose für den ausgefallenen Zustand macht. Um die optimale Kombination der Hyperparameter zu finden, wird der SVM-Algorithmus mit verschiedenen Kombinationen durchgeführt. Zunächst wird der Algorithmus mit den Kombinationen, die ausgelistet sind, durchgeführt und in den Abbildungen werden die Ergebnisse gezeigt.

- *cost=100* und *gamma=1* Auf der Abbildung 38
- *cost=50* und *gamma=0,8* Auf der Abbildung 39
- *cost=50* und *gamma=1* Auf der Abbildung 40

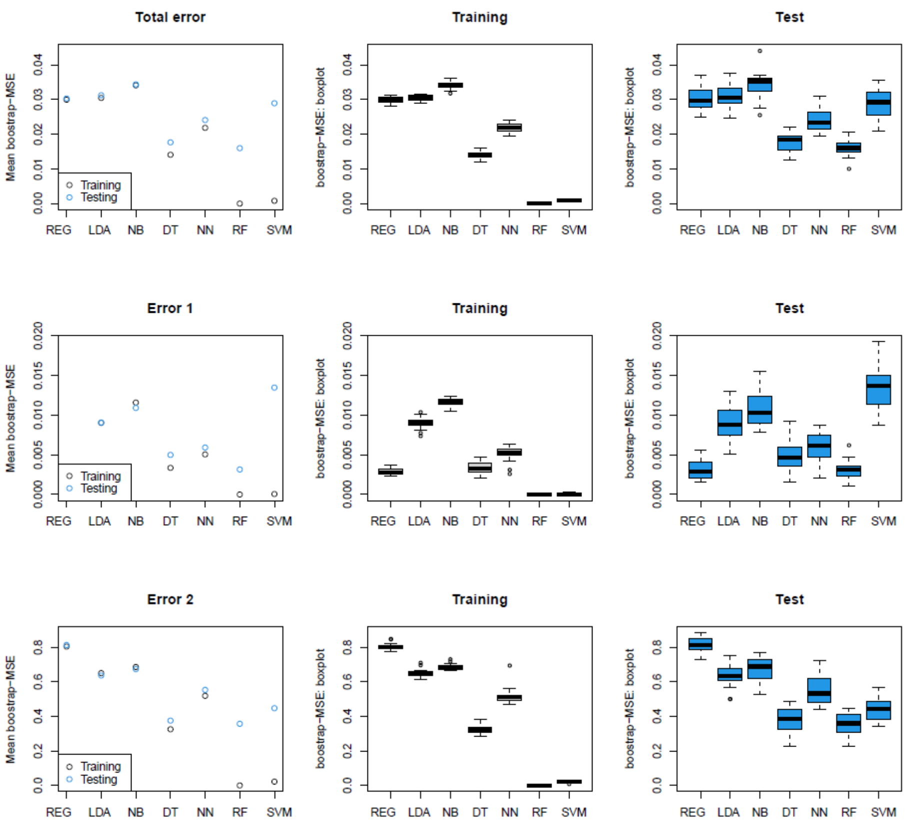

Abbildung 38: Parameter: *cost=100* und *gamma=1*

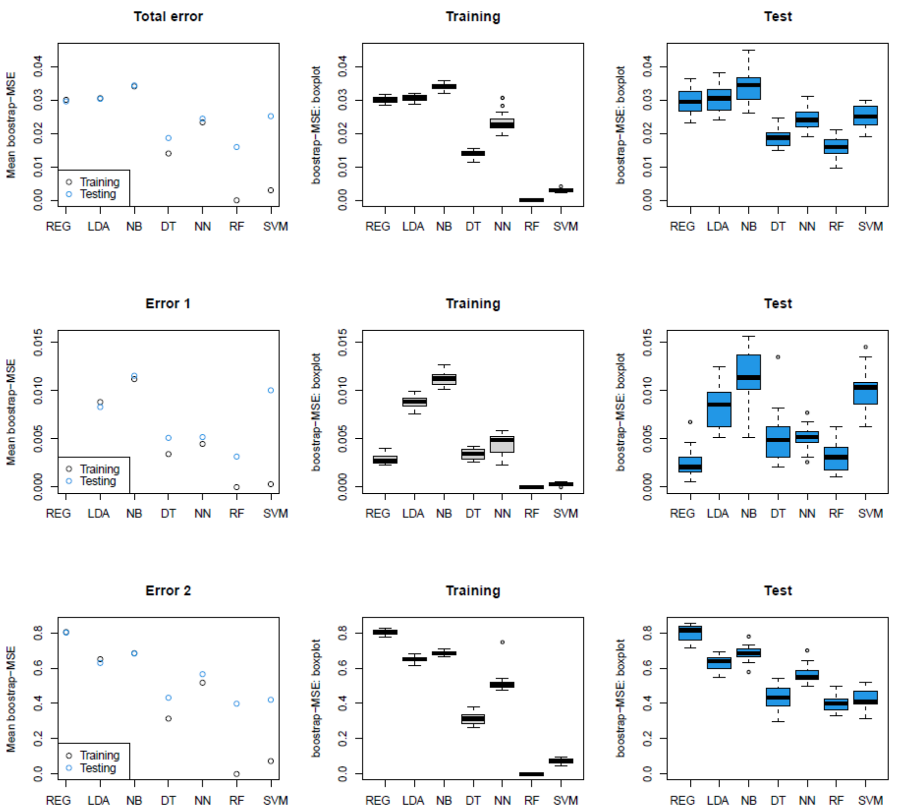

Abbildung 39: Parameter: *cost=50* und *gamma=0,8*

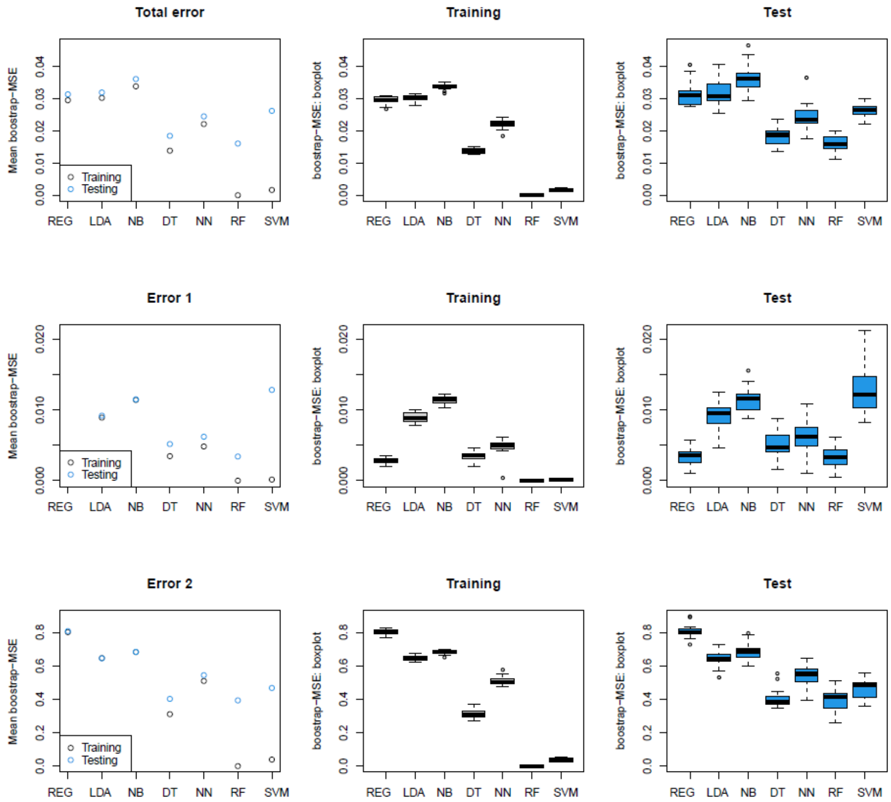

Abbildung 40: Parameter: *cost =50* und *gamma =1*

Auf den Abbildungen sind die Ergebnisse dargestellt. Für alle Kombinationen der Parameter sind die Trainingsfehlerprozente sehr gering. Die Testfehlerprozente sind hier entscheidend. Besonders bei den ausgefallenen Zuständen hat der SVM-Algorithmus mit den Parameter *gamma=0,8* und *cost=50* eine geringe Testfehlerquote. Weil es viele Kombinationen der Parameter für die Entscheidung gibt, wird ein neues Skript geschrieben (s. Anhang G), in dem nur die SVM-Algorithmen durchgeführt werden. Mit diesen neuen Durchführungen können die optimalen Parameter für SVM-Algorithmus gefunden werden. Die neuen Parameter für die Algorithmen:

- 1*: cost=1* und *gamma=0.8*
- 2: *cost=1* und *gamma=1*
- 3: *cost=10* und *gamma=0.8*
- 4: *cost=50* und *gamma=0.8*
- 5*: cost=50* und *gamma=1*
- 6: *cost=100* und *gamma=0.8*
- 7: *cost=100* und *gamma*=1

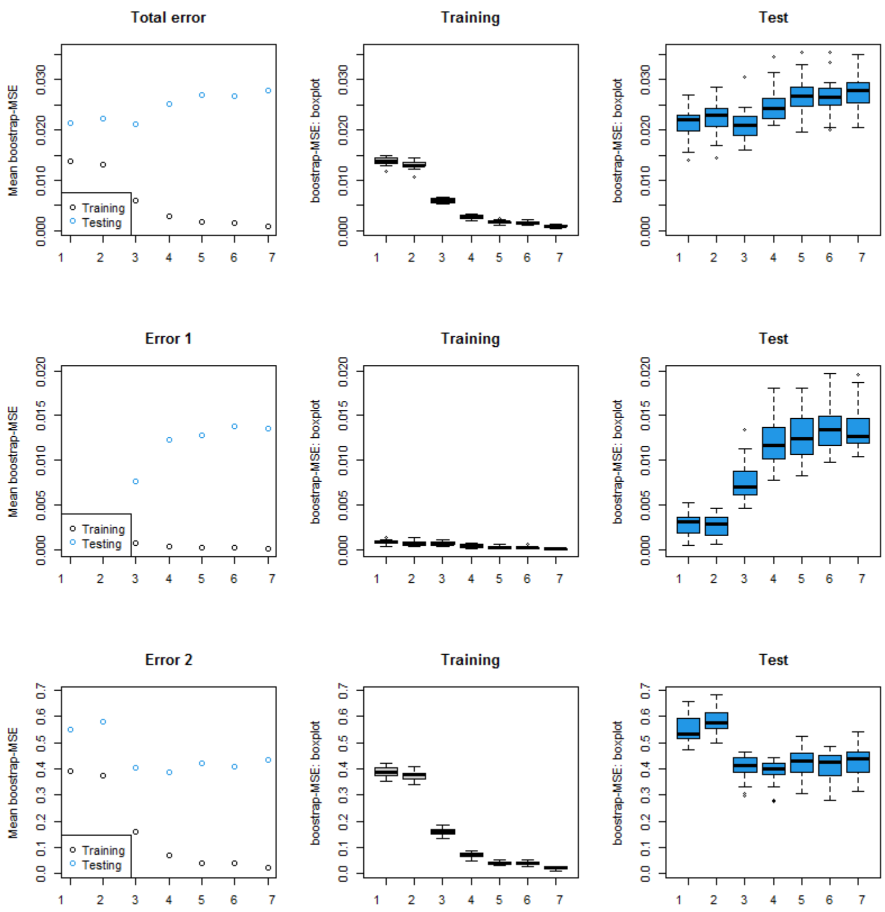

Abbildung 41: Cost- und Gamma-Parameter, Kombinationsvariante 1

Mit den gleichen Gamma-Parametern zeigt der Parameter *cost=1* eine hohe Trainings- und Testfehlerrate beim ausgefallenen Zustand auf. Mit der Steigerung des Wertes des Cost-Parameters sinken die Trainingsfehlerraten bei allen Zuständen. Aber die Testfehlerrate steigt beim betriebsbereiten Zustand. Damit die optimale Kombination gefunden werden kann, werden erst die Cost-Parameter 10,50 und 100 mit verschiedenen Gamma-Parameter kombiniert und durchgeführt. Mit diesen Kombinationen kann erst der optimale Gamma-Parameter festgelegt werden und dann können mit diesem Gamma-Parameter die verschiedenen Cost-Parameter ausgeführt werden. Die Parameter für die SVM-Algorithmen sind wie folgt aufgelistet:

- 1: *cost=10* und *gamma=0,01*
- 2: *cost=10* und *gamma=0,1*
- 3: *cost=10* und *gamma=0,5*
- 4: *cost=10* und *gamma=0,6*
- 5: *cost=10* und *gamma=0,8*
- 6: *cost=10* und *gamma=1*
- 7: *cost=10* und *gamma=5*

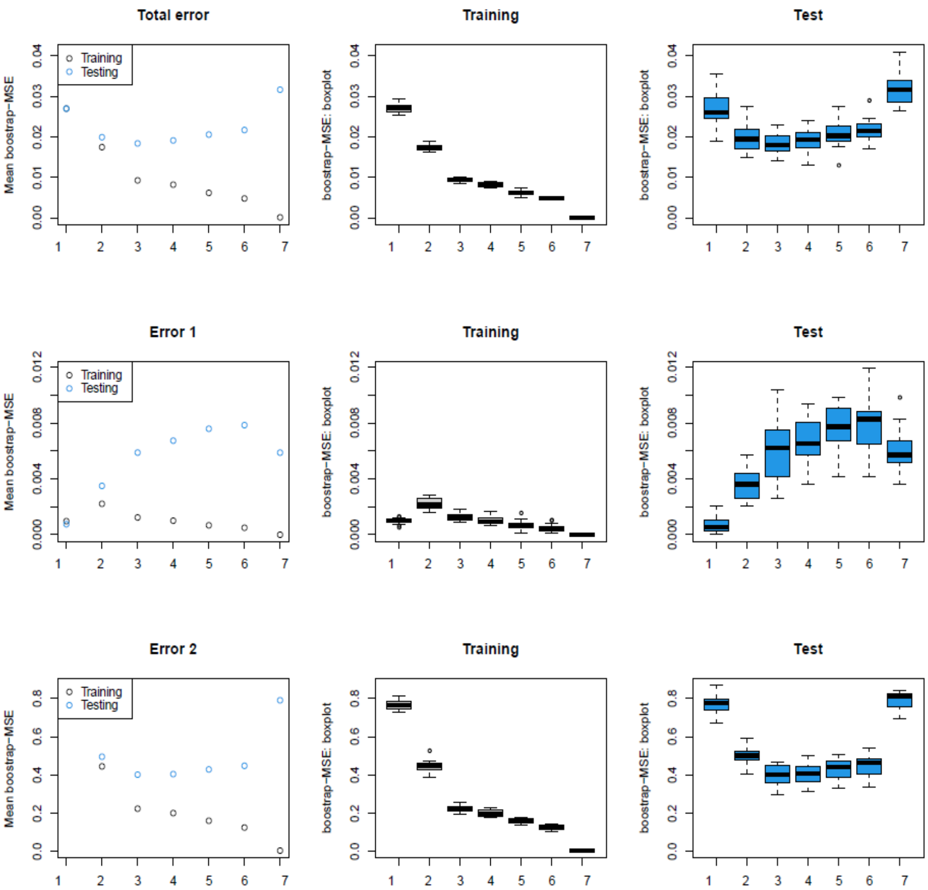

Abbildung 42: Untersuchung des konstanten Cost-Parameters von *cost=10* mit unterschiedlichen Gamma-Parametern

Aus den Ergebnissen nach Abbildung 42 sind zu erkennen, dass der optimale Gamma-Parameter mit dem Parameter *cost=10* für alle Zustände den Wert 0,6 hat. Für dieses Ergebnis ist besonders der ausgefallene Zustand entscheidend, weil die höchsten Fehlerraten beim ausgefallenen Zustand entstehen. Die Trainingsfehlerraten sind immer gestiegen, aber Testfehler haben eine U-Form. Und die optimalen Parameter sind nach dieser Variante *cost=10* und *gamma=0,6*, weil die Testfehlerraten ab diesem Punkt steigen. Um den optimalen Gamma-Parameter zu finden, werden im nächsten Schritt alle SVM-Algorithmen mit den Parametern *cost=50* und verschiedenen Gamma-Parametern ausgeführt.

Die Parameter sind folgende wie folgt ausgelistet:

- 1: *cost=50* und *gamma=0,01*
- 2: *cost=50* und *gamma=0,1*
- 3: *cost=50* und *gamma=0,5*
- 4: *cost=50* und *gamma=0,6*
- 5: *cost=50* und *gamma=0,8*
- 6: *cost=50* und *gamma=1*
- 7: *cost=50* und *gamma=5*

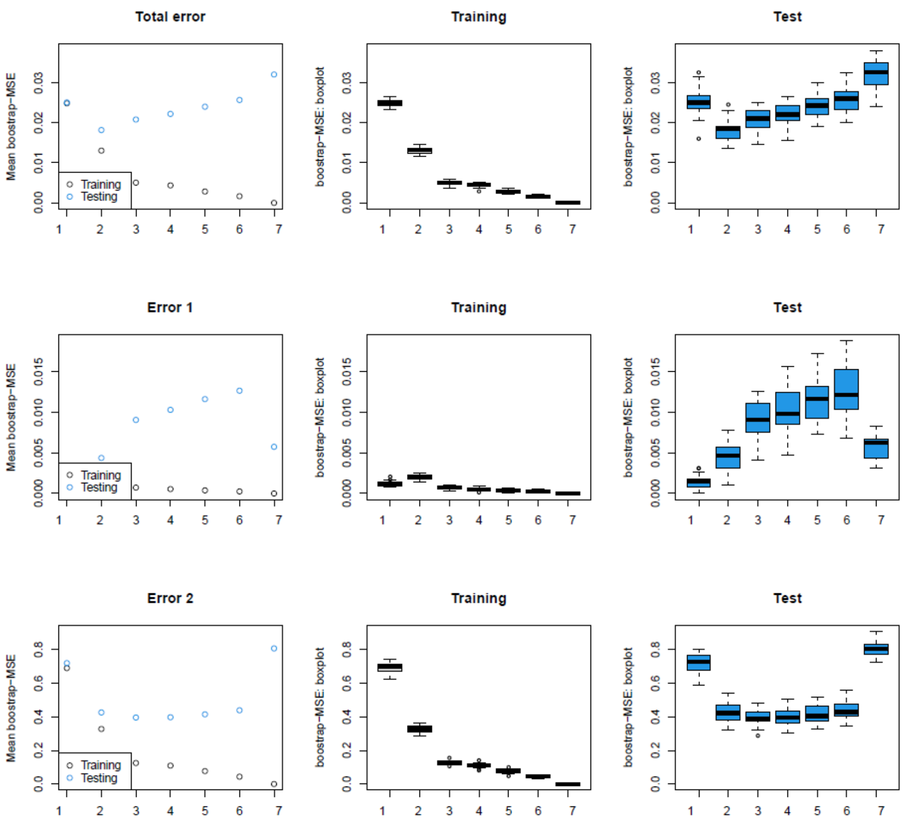

Abbildung 43: Untersuchung des konstanten Cost-Parameters von cost=50 mit unterschiedlichen Gamma-Parametern

Die Ergebnisse mit den Parametern *cost=50* und verschiedenen Gamma-Parametern weisen auch ähnliche Ergebnisse wie mit den Parametern *cost=10* auf. Für den ausgefallenen Zustand weisen die Algorithmen vom Parameter *gamma=0,1* bis *gamma=1* ähnliche Fehlerquote auf. Der Algorithmus prognostiziert den Total-Fehler mit *gamma=0,1* und *cost=50* mit einer niedrigen Fehlerrate. Für die weitere Entscheidung der optimalen Gamma-Parameter werden die SVM-Algorithmen mit den Parametern *cost=100* und verschiedenen Gamma-Parametern durchgeführt. Die Parameter sind:

- 1: *cost=100* und *gamma=0,01*
- 2: *cost=100* und *gamma=0,1*
- 3: *cost=100* und *gamma=0,5*
- 4: *cost=100* und *gamma=0,6*
- 5: *cost=100* und *gamma=0,8*
- 6: *cost=100* und *gamma=1*
- 7: *cost=100* und *gamma=5*

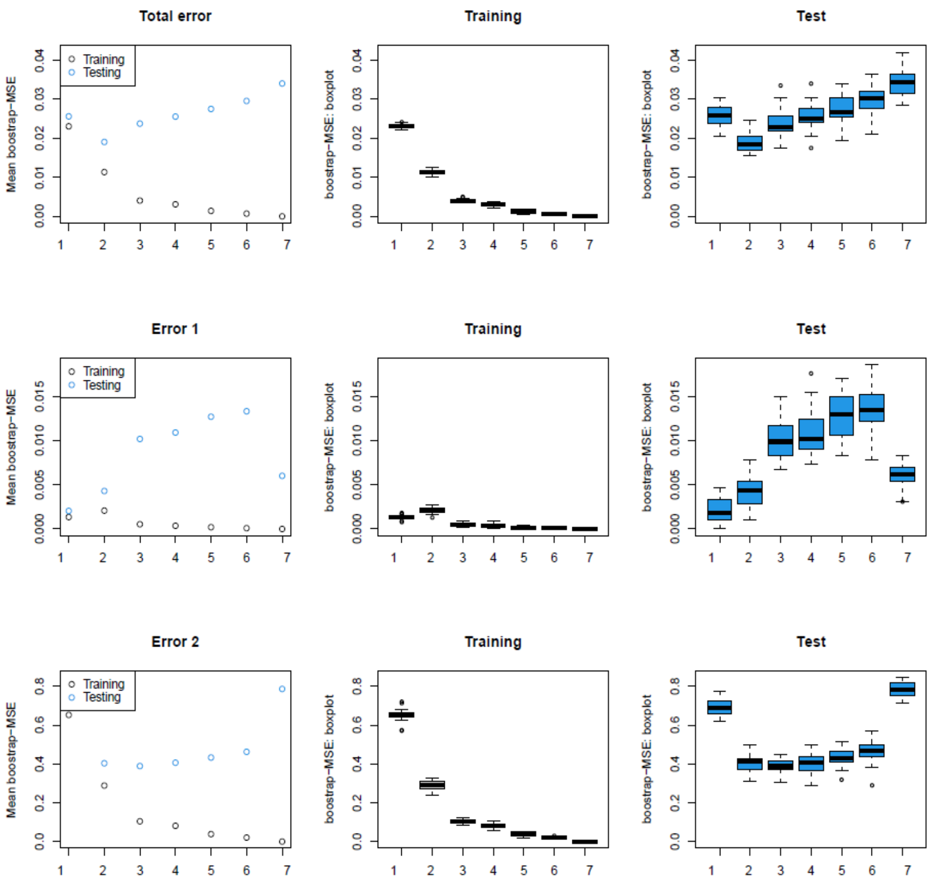

Abbildung 44: Untersuchung des konstanten Cost-Parameters von cost=100 mit unterschiedlichen Gamma-Parametern

Die Ergebnisse sind ähnlich wie beim letzten Versuch mit cost=50 und verschiedenen Gamma-Parametern. Für den ausgefallenen Zustand weisen die Algorithmus mit den Parametern *gamma=0,1*, *gamma=0,5* und *gamma=0,6* wenige Fehlerraten auf. Der Algorithmus prognostiziert den Total-Fehler besser mit den Parametern *cost =100* und *gamma=0,1*. Aus diesen Ergebnissen ist es schwer zu identifizieren, welcher Parameter besser ist. Bei den Kombinationen mit *cost=10* wird das optimale Ergebnis mit dem Parameter *gamma=0,6* erreicht. Aber bei den Kombinationen mit *cost=50* und *cost=100* weist der Algorithmus eine bessere Fehlerrate mit *gamma=0,1* auf. Aus diesem Grund werden als nächstes die Algorithmen mit *gamma=0,1* und verschiedenen Cost-Parametern durchgeführt. Dann werden die gleichen Kombinationen mit *gamma=0,6* durchgeführt. Die neuen Parameter sind wie folgt:

- 1: *cost=1* und *gamma=0,1*
- 2: *cost=10* und *gamma=0,1*
- 3: *cost=20* und *gamma=0,1*
- 4: *cost=50* und *gamma=0,1*
- 5: *cost=60* und *gamma=0,1*
- 6: *cost=80* und *gamma=0,1*
- 7: *cost=100* und *gamma=0,1*

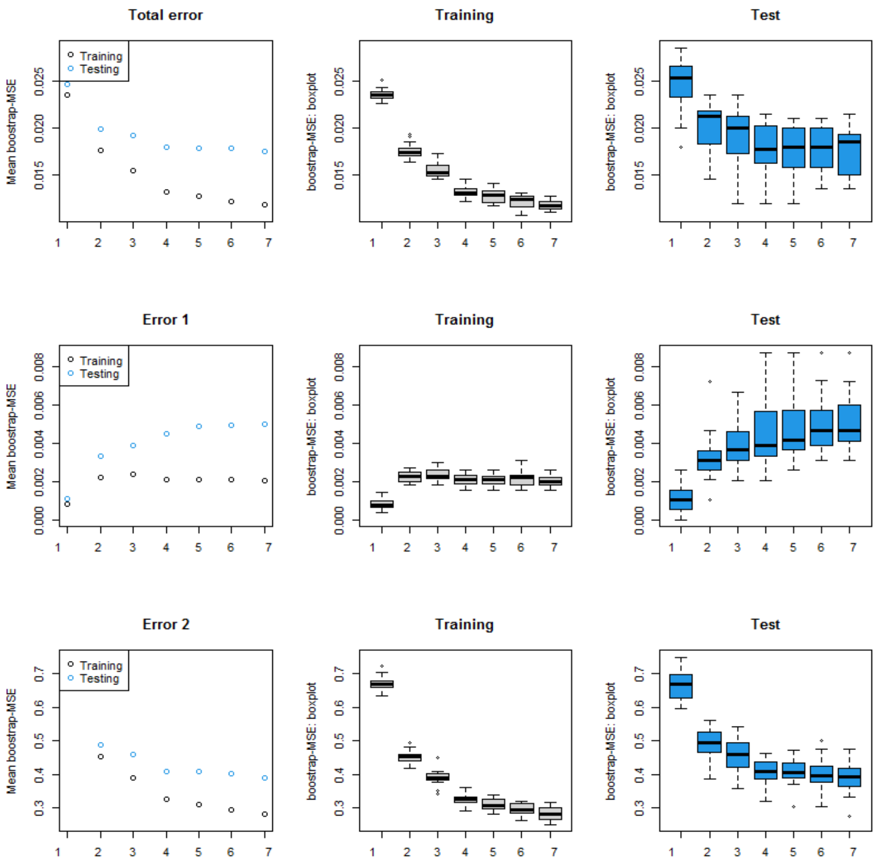

Abbildung 45: Untersuchung des konstanten Gamma-Parameters von gamma=0,1 mit unterschiedlichen Cost-Parametern

Nach diesem Versuch (s. Abbildung 45) ist schwer zu entscheiden, welche Cost-Parameter mit *gamma=0,1* die optimale Prognose erzielt. Aus den drei Zuständen können aber die optimalen Parameter als *cost=50* und *gamma=0,1* besagt werden, weil besonders bei dem ausgefallenen Zustand der Algorithmus mit diesen Parametern eine niedrige Fehlerrate aufweist. Der Algorithmus ergibt mit diesen Parametern 1,3 % Trainingsfehler und 1,9 % Testfehler für den Total-Fehler, 0,2 % Trainingsfehler und 0,4 % Testfehler für den betriebsbereiten Zustand und 32,5 % Trainingsfehler und 41 % Testfehler für den ausgefallenen Zustand.

Als letztes werden die SVM-Algorithmen mit den Parametern *gamma=0,6* und mit verschiedenen cost Parametern durchgeführt. Nach dieser Durchführung können die optimalen Parameter gefunden werden. Die Parameter sind wie folgt:

- 1: *cost=1* und *gamma=0,6*
- 2: *cost=10* und *gamma=0,6*
- 3: *cost=20* und *gamma=0,6*
- 4: *cost=50* und *gamma=0,6*
- 5: *cost=60* und *gamma=0,6*
- 6: *cost=80* und *gamma=0,6*
- 7: *cost=100* und *gamma=0,6*

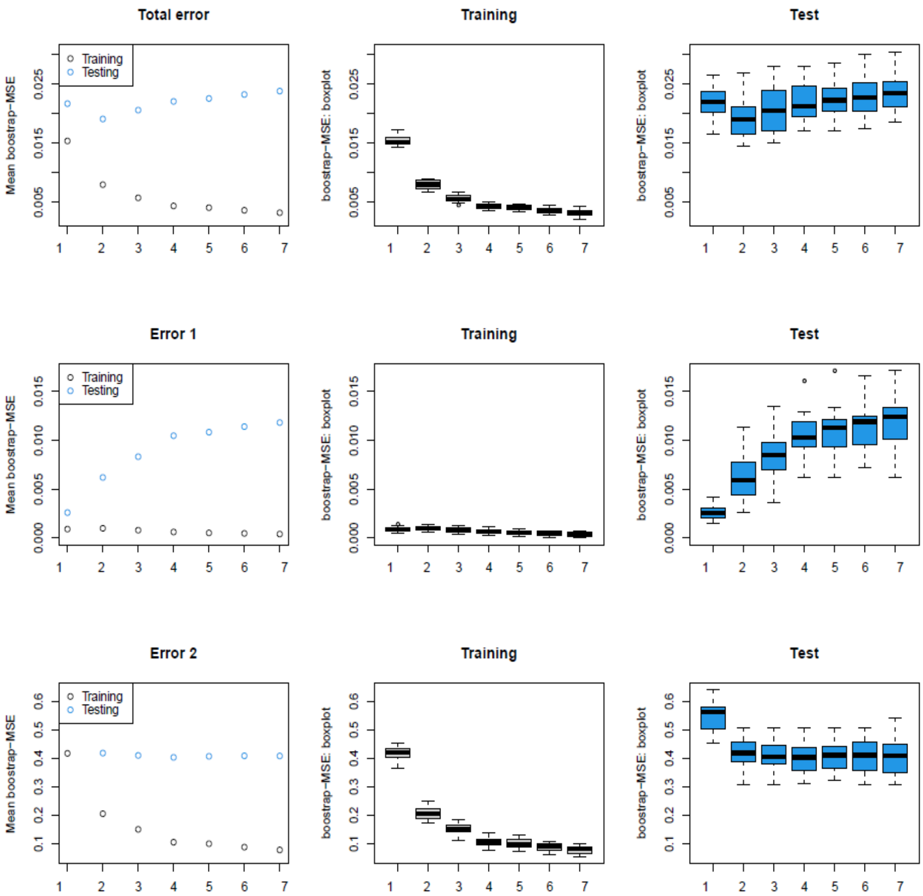

Abbildung 46: Untersuchung des konstanten Gamma-Parameters von gamma=0,6 mit unterschiedlichen Cost-Parametern

Aus der Abbildung 46 wird ersichtlich, dass die Trainingsfehlerrate bei allen Zuständen gesunken ist. Beim Total-Fehler hat der Algorithmus mit dem Parameter *cost=10* und *gamma=0,6* die niedrigste Testfehlerrate. Beim betriebsbereiten Zustand sind die Testfehlerraten immer gestiegen. Für den ausgefallenen Zustand haben die Cost-Parameter ab *cost=10* sehr ähnliche Fehlerquote. Wenn alle drei Zustände berücksichtigt werden, können die optimalen Hyperparameter als *cost=10* und *gamma=0,6* identifiziert werden. Mit diesen Parametern weist der Algorithmus 0,7 % Trainingsfehler und 1,9 % Testfehler für den Total-Fehler, 0,1 % Trainingsfehler und 0,6 % Testfehler für den betriebsbereiten Zustand und 20 % Trainingsfehler und 41 % Testfehler für den ausgefallenen Zustand auf.

Aus den letzten zwei Versuchen können die optimalen Parameter festgelegt werden. Zwischen den Parametern *cost=50* und *gamma=0,1* und *cost=10* und *gamma=0,6* weist der Algorithmus ähnliche Fehlerquote auf. Aber mit den Parameter *cost=10* und *gamma=0,6* prognostiziert der Algorithmus die Fehlerrate zuverlässiger, das bedeutet, dass die Fehlerrate in dem Boxplots wenig auseinander liegen. Aus diesem Grund können die optimalen Parameter als *gamma=0,6* und *cost=10* identifiziert werden.

4.4 Auswertung der Ergebnisse

Die ersten Ergebnisse wurden im Kapitel 4.3 erläutert. Damit die Endergebnisse mit den ersten Situationen verglichen werden können, werden die ersten Ergebnisse, in dem die Algorithmen keine Parameteränderung haben, hier nochmals erklärt.

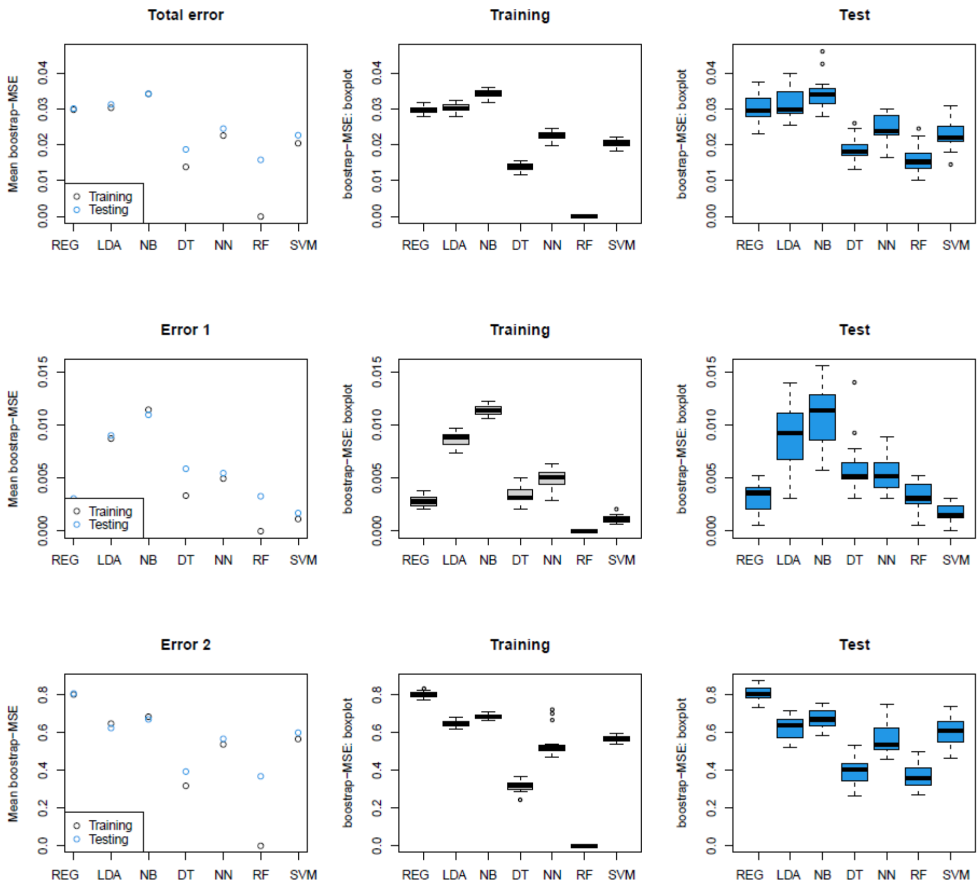

Abbildung 47: Erste Ergebnisse ohne Parameteränderung

Auf der Abbildung 47 sind die Ergebnisse von den Algorithmen, die keine Parameteränderung haben, zu sehen. Für die Prognose sind Testfehlerquoten wichtig, weil diese Daten unabhängig von den Trainingsdaten mit den Algorithmen durchgeführt werden. Für den ausgefallenen Zustand haben RF und DT eine niedrige Fehlerrate. REG prognostiziert diesen Zustand mit 80 % Fehler. REG kann aber gut den betriebsbereiten Zustand vorhersehen. Für die betriebsbereite Zustandsprognose weisen alle Algorithmen eine sehr niedrige Fehlerquote auf. Und für den Total-Fehler erzielen die Algorithmen RF und DT eine niedrige Fehlerquote. Alle Mittelwerte der auftretenden Fehler werden in einer Tabelle nochmals dargestellt, damit die Veränderungen nachher genauer betrachtet werden können.

Tabelle 5: Mittelwert der Fehlerrate der Algorithmen bei der Klassifizierung

Algorithmen	Situation	Total Error	Error 1	Error 2
REG	Training	2.90 %	0.29 %	79.00 %
	Test	3.10 %	0.28 %	80.00 %
LDA	Training	3.00 %	0.89 %	63.80 %
	Test	3.00 %	0.97 %	66.50 %
NB	Training	3.40 %	1.11 %	68.00 %
	Test	3.40 %	1.22 %	69.00 %
DT	Training	1.30 %	0.33 %	31.60 %
	Test	1.80 %	0.60 %	41.40 %
NN	Training	2.20 %	0.49 %	51.90 %
	Test	2.30 %	0.56 %	56.80 %
RF	Training	0.00 %	0.00 %	0.00 %
	Test	0.15 %	0.37 %	38.00 %
SVM	Training	2.00 %	0.12 %	55.00 %
	Test	2.20 %	0.18 %	61.40 %

In der Tabelle 5 sind alle Mittelwerte der Fehlerrate in Prozenten genauer zu sehen. NN weist mit zwei Neuronen 2,2 % Trainingsfehler und 2,3 % Testfehler beim Total-Fehler, 0,49 % Trainingsfehler und 0,56 % Testfehler beim betriebsbereiten Zustand und 51 % Trainingsfehler und 56 % Testfehler beim ausgefallenen Zustand auf. SVM erreicht 2 % Trainingsfehler und 2,2 % Testfehler beim Total-Fehler, 0,12 % Trainingsfehler und 0,18 % Testfehler beim betriebsbereiten Zustand und 55 % Trainingsfehler und 61 % Testfehler beim ausgefallenen Zustand.

In den Kapiteln 4.3.1 und 4.3.2 wurden die optimalen Parameter für NN und SVM gefunden. Der optimale NN-Parameter ist *h=c(8,4),* das heißt, dass der Algorithmus erst mit 8 Neuronen dann mit 4 Neuronen durchgeführt wird. Die optimalen SVM-Parameter sind *cost=10* und *gamma=0,6*. Diese zwei Algorithmen werden mit diesem Parameter und die restlichen Algorithmen werden ohne Änderungen durchgeführt. Die neuen Ergebnisse und die Fehlerrate der verschiedenen Zustände sind auf der Abbildung 48 zu sehen.

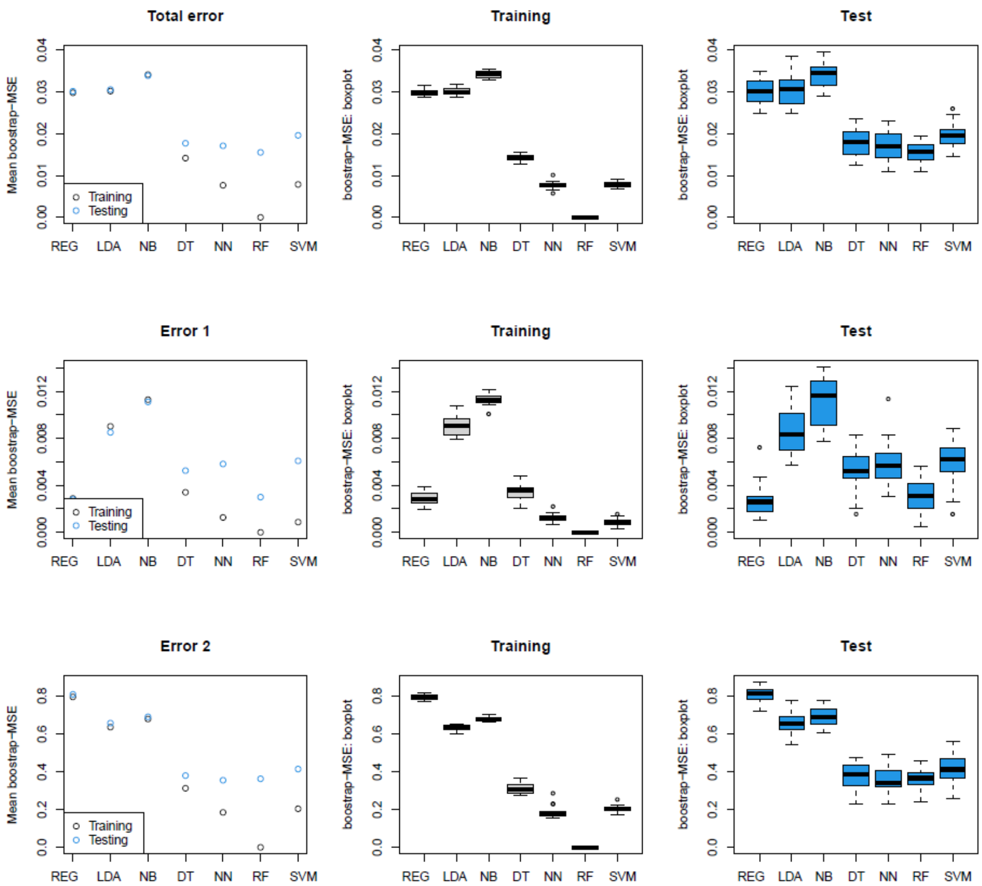

Abbildung 48: Endergebnisse mit der Parameteränderung

Mit der Parameteränderung erreichen NN und SVM eine bessere Prognose. Die Mittelwerte der auftretenden Fehler für NN und SVM werden in der Tabelle 6 dargestellt. In der Tabelle 6 ist zu sehen, welche Fehlerrate die Algorithmen vor und nach der Parameteränderung ermittelt haben. NN prognostiziert mit 8 und 4 Neuronen den betriebsbereiten Zustand mit 0,58% Fehler und den ausgefallenen Zustand mit 35,90 % Fehler. Ohne Parameteränderung erreicht der Algorithmus 0,56 % Fehler für die betriebsbereite Zustandsprognose und 56,80 % Fehler für die ausgefallene Zustandsprognose. Beim Total-Fehler ist die Fehlerrate von 2,30 % auf 1,70 % gesunken. SVM prognostiziert mit den Parametern *cost=10* und *gamma=0,6* den betriebsbereiten Zustand mit 0,6% Fehlern und den ausgefallenen Zustand mit 41,39 % Fehlern. Die Fehlerquote bei der betriebsbereiten Zustandsprognose und der ausgefallenen Zustandsprognose waren ohne Änderung: 0,18 % und 61,40 %. Beim Total-Fehler sind die auftretenden Fehler von 2,20 % auf 1,96 % gesunken. Die Ergebnisse zeigen, dass die Algorithmen mit der Parameteränderung die ausgefallenen Zustände besser vorhersagen können.

Tabelle 6: Mittelwert der Fehlerrate für NN und SVM vor und nach der Parameteränderung bei der Klassifizierung

Algorithmen	Situation	Total Error	Error 1	Error 2
NN-Alt	Training	2,20 %	0,49 %	51,90 %
	Test	2,30 %	0,56 %	56,80 %
NN-Neu	Training	0,76 %	0,12 %	18,40 %
	Test	1,70 %	0,58 %	35,90 %
SVM-Alt	Training	2,00 %	0,12 %	55,00 %
	Test	2,20 %	0,18 %	61,40 %
SVM-Neu	Training	0,78 %	0,08 %	20,30 %
	Test	1,96 %	0,60 %	41,39 %

Die Mittelwerte der auftretenden Fehler werden in Prozenten für alle Algorithmen in der Tabelle 7 präsentiert. Aus der Abbildung 48 und Tabelle 7 sind zu sehen, welche Algorithmen die bessere Prognose stellen können. NN, RF, SVM und DT prognostizieren die ausgefallenen Zustände besser als REG, LDA und NB. Dieses Ergebnis zeigt, dass die Algorithmen des maschinellen Lernens bessere Prognosen für die ausgefallenen Zustände stellen können. Die Algorithmen prognostizieren den betriebsbereiten Zustand sehr gut, weil die Algorithmen sehr niedrige Fehlerrate für die betriebsbereite Zustandsprognose aufweisen. Die Ergebnisse für den Total-Fehler sind ähnlich wie beim ausgefallenen Zustand. Die Algorithmen DT, NN, RF und SVM weisen wenige Fehler als REG, LDA und NB auf.

Tabelle 7: Mittelwerte der Fehlerrate bei der Klassifizierung Endergebnisse

Algorithmen	Situation	Total Error	Error 1	Error 2
REG	Training	2,98 %	0,28 %	79,54 %
	Test	3,00 %	0,27 %	80,08 %
LDA	Training	3,00 %	0,90 %	63,50 %
	Test	3,00 %	0,85 %	65,07 %
NB	Training	3,40 %	1,11 %	67,84 %
	Test	3,40 %	1,10 %	69,00 %
DT	Training	1,40 %	0,33 %	31,24 %
	Test	1,70 %	0,52 %	37,94 %
NN	Training	0,76 %	0,12 %	18,40 %
	Test	1,70 %	0,58 %	35,90 %
RF	Training	0,00 %	0,00 %	0,00 %
	Test	1,50 %	0,30 %	36,20 %
SVM	Training	0,78 %	0,08 %	20.30 %
	Test	1,96 %	0,60 %	41,39 %

5. Fazit

Die Bedeutung der Industrie 4.0 und der prädiktiven Instandhaltung nimmt weiter zu. Mit Hilfe von CPS, IoT, Sensoren und Big Data sind die Systeme zuverlässiger geworden. Mit den Chancen, die diesen Prinzipen mitbringen, treten jedoch neue Herausforderungen für die Zuverlässigkeitstechnik auf. Es gibt immer mehr neue Daten von Maschinen, die gesammelt und analysiert werden müssen. Diese Daten können mit Sensoren gesammelt und mit verschiedenen Methoden analysiert werden. Unter prädiktiver Instandhaltung wird eine Instandhaltungsstrategie verstanden, bei der der Ausfall eines technischen Systems durch Condition Monitoring mittels Sensoren vorhergesagt werden kann. Die prädiktive Instandhaltung ist deshalb so innovativ, weil diese Instandhaltungsstrategie die Ausfälle der Maschinen und die unnötigen Wartungskosten vermeiden kann.

Für die prädiktive Instandhaltung ist die Klassifizierung der Betriebszustände essenziell. Die Betriebszustände können in betriebsbereite oder ausgefallene Zustände klassifiziert werden. Die durch Sensoren gesammelten Daten können die Betriebszustände durch die statischen Methoden oder die Algorithmen des maschinellen Lernens klassifizieren.

In Rahmen dieser Arbeit wurde ein Datensatz verwendet, in dem sich die Sensordaten und die Informationen über die Betriebszustände befinden. Der Datensatz wurde analysiert und mit Hilfe der folgenden Methoden klassifiziert: Logistische Regression, Lineare Diskriminantfunktion, Naive Bayes, Decision Tree, Neural Network, Random Forest und Support Vector Machine. Die ersten Ergebnisse zeigen, dass die Art der Methoden signifikant ist. Die Algorithmen des maschinellen Lernens klassifizieren die Zustände mit weniger Fehlerraten als der statistischen Methoden. Die Algorithmen des maschinellen Lernens haben auch verschiedene Hyperparameter. Mit der Änderung der Parameter klassifizieren die Algorithmen auch mit verschiedenen Fehlerraten. Die optimalen Parameter wurden für die Neural Network und Support Vector Machine identifiziert und die Fehlerraten der Klassifizierung der Betriebszustände sind mit dieser Parameteränderung gesunken. Mit den neuen Parametern weist Neural Network beim betriebsbereiten Zustand 0,58 % Fehler und beim ausgefallenen Zustand 35,9 % Fehler auf. Die Fehlerquote für Support Vector Maschine folgt mit 0,6 % und 41,3 %. Random Forest, der auch ein Algorithmus des maschinellen Lernens ist, klassifiziert die Zustände auch mit niedrigen Fehlerraten.

Für die zukünftigen Forschungen ist es möglich, mit vielen neuen Daten zu arbeiten und die Zustände neu zu klassifizieren. Für so eine Datensammlung können mit vielen Sensoren gearbeitet werden. Je mehr Daten gesammelt werden, desto zuverlässiger wird die Analyse. Wenn ein Datensatz viel mehr Informationen über die ausgefallenen Zustände aufweist, können die Methoden mit niedrigen Fehlern die ausgefallenen Zustände klassifizieren. Die Algorithmen des maschinellen Lernens können diesen Zustand auch besser lernen und die besseren Prognosen stellen. Eine andere Möglichkeit ist neue Hyperparameter für die Algorithmen zu finden. Die neuen Parameter können systematisch geändert und die Fehlerrate untersucht werden, um die optimalen Parameter zu identifizieren.

Wie nun verdeutlicht wurde, ist die prädiktive Instandhaltung eine gute Option für die Industrie. Es ist nicht möglich, in einer sehr kurzen Zeit eine optimale Instandhaltungsstrategie zu erstellen. Aber mit Hilfe der verschiedenen Methoden wird diese Instandhaltungsstrategie in vielen Feldern benutzt und die Zuverlässigkeit der Systeme verbessert.

Literatur- / Quellenverzeichnis

[1] Coit, D. W.; Zio, E. (2019): The evolution of system reliability optimization. In: Reliability Engineering & System Safety, 192.

[2] Hoffmann Souza, M. L.; Da Costa, C. A.; Oliveira Ramos, G. de; Da Rosa Righi, R. (2020): A survey on decision-making based on system reliability in the context of Industry 4.0. In: Journal of Manufacturing Systems, 56, S. 133–156.

[3] Krupitzer, C.; Wagenhals, T.; Züfle, M.; Lesch, V.; Schäfer, D.; Mozaffarin, A.; Edinger, J.; Becker, C.; Kounev, S. (2020): A Survey on Predictive Maintenance for Industry 4.0. arXiv: 2002.08224

[4] Bink, R.; Zschech, P. (2018): Predictive Maintenance in der industriellen Praxis. In: HMD Praxis der Wirtschaftsinformatik, 55 (3), S. 552–565.

[5] Zio, E. (2016): Some Challenges and Opportunities in Reliability Engineering. In: IEEE Transactions on Reliability, 65 (4), S. 1769–1782

[6] Zhang, W.; Yang, D.; Wang, H. (2019): Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey. In: IEEE Systems Journal, 13 (3), S. 2213–2227.

[7] Farsi, M. A.; Zio, E. (2019): Industry 4.0: Some Challenges and Opportunities for Reliability Engineering. In: International Journal of Reliability, Risk and Safety: Theory and Application, 2 (1), S. 23–34.

[8] Brödner, P. (2015): Industrie 4.0 und Big Data – wirklich ein neuer Technologieschub? In: Hirsch-Kreinsen, H.; Ittermann, P.; Niehaus, J. (Hrsg.): Digitalisierung industrieller Arbeit. Nomos, S. 232–251.

[9] Schadler, M.; Hafner, N.; Landschützer, C. (2019): Konzepte und Methoden für prädiktive Instandhaltung in der Intralogistik. In: Logistics Journal: Proceedings, Vol. 2019, S. 1-11.

[10] Oztemel, E.; Gursev, S. (2020): Literature review of Industry 4.0 and related technologies. In: Journal of Intelligent Manufacturing, 31 (1), S. 127–182.

[11] Drakaki, M.; Karnavas, Y. L.; Tzionas, P.; Chasiotis, I. D. (2021): Recent Developments Towards Industry 4.0 Oriented Predictive Maintenance in Induction Motors. In: Procedia Computer Science, S. 943–949.

[12] Cinar, Z. M.; Abdussalam Nuhu, A.; Zeeshan, Q.; Korhan, O.; Asmael, M.; Safaei, B. (2020): Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. In: Sustainability, 12 (19) 8211.

[13] Lazarova-Molnar, S.; Mohamed, N. (2019): Reliability Assessment in the Context of Industry 4.0. In: Procedia Computer Science, 151 (2019), S. 691–698.

[14] Schütze, A.; Helwig, N. (2017): Sensorik und Messtechnik für die Industrie 4.0. In: tm - Technisches Messen, 84 (5), S. 310–319.

[15] Javaid, M.; Haleem, A.; Singh, R. P.; Rab, S.; Suman, R. (2021): Significance of sensors for industry 4.0: Roles, capabilities, and applications. In: Sensors International, 2 (2021) 100110.

[16] Statistical Modeling: Online verfügbar unter https://www.omnisci.com/technical-glossary/statistical-modeling , zuletzt geprüft am 21.11.2021.

[17] Bansal, S.: Machine Predictive Maintenance Classification. Online verfügbar unter https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification , zuletzt geprüft am 05.08.2022

[18] Matzka, S. (2020): Explainable Artificial Intelligence for Predictive Maintenance Applications. In: 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), S. 69-74.

[19]R Project: Online verfügbar unter www.r-project.org/ , zuletzt geprüft am 29.05.2022.

[20] Heesen, B. (2021): Data Science und Statistik mit R. Wiesbaden: Springer Fachmedien Wiesbaden.

[21] R Project/ Packages: MASS: Support Functions and Datasets for Venables and Ripley's MASS (2022): Online verfügbar unter https://cran.r-project.org/web/packages/MASS/index.html , zuletzt geprüft am 05.08.2022.

[22] R Project/ Packages: Online verfügbar unter https://cran.r-project.org/web/packages/available_packages_by_name.html#available-packages-R , zuletzt geprüft am 05.08.2022.

[23] James, G.; Witten, D.; Hastie, T.; Tibshirani, R. (2013): An Introduction to Statistical Learning. New York, NY: Springer New York.

[24] R Project/ Package „Neuralnet" (2019): Online verfügbar unter https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf , zuletzt geprüft am 20.06.2022.

[25] R Project/ Package „e1071" (2022): Online verfügbar unter https://cran.r-project.org/web/packages/e1071/e1071.pdf, zuletzt geprüft am 27.06.2022.

[26] Arena, F.; Collotta, M.; Luca, L.; Ruggieri, M.; Termine, F. G. (2022): Predictive Maintenance in the Automotive Sector: A Literature Review. In: Mathematical and Computational Applications, 27 (2), S. 1–21.

[27] Carvalho, T. P.; Soares, F. A. A. M. N.; Vita, R.; Da Francisco, R. P.; Basto, J. P.; Alcalá, S. G. S. (2019): A systematic literature review of machine learning methods applied to predictive maintenance. In: Computers & Industrial Engineering, 137 (2019) 106024.

[28] Spendla, L.; Kebisek, M.; Tanuska, P.; Hrcka, L. (Hrsg.) (2017): Concept of Predictive Maintenance of Production Systems in Accordance with Industry 4.0. IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI), S. 000405-000410.

[29] Cakir, M.; Guvenc, M. A.; Mistikoglu, S. (2021): The experimental application of popular machine learning algorithms on predictive maintenance and the design of IIoT based condition monitoring system. In: Computers & Industrial Engineering, 151 (2021) 106948.

[30] Selcuk, S. (2017): Predictive maintenance, its implementation and latest trends. In: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 231 (9), S. 1670–1679.

[31] Kudelina, K.; Vaimann, T.; Asad, B.; Rassõlkin, A.; Kallaste, A.; Demidova, G. (2021): Trends and Challenges in Intelligent Condition Monitoring of Electrical Machines Using Machine Learning. In: Applied Sciences, 11 (6), 2761.

Inhaltsverzeichnis der Anhänge

Anhang A

```
## Alle Variablen löschen:
rm(list=ls())

## Pakete installieren
library(MASS)
library(rpart)
library(rpart.plot)
library(neuralnet)
library(randomForest)
library(e1071)
library(klaR)
library(corrplot)

## Working directory
setwd("C:/Users/Desktop/Machine_failure")

## Datensatz lesen
data=read.csv("predictive_maintenance.csv",header=TRUE)
names(data);summary(data)
Y=data[,9]==1
X=data[,4:8]
```

Abbildung A.1: Vorbereitung der Daten für die Analyse

Anhang B

```
## Korrelation
cor(cbind(X,Y))
corrplot(cor(cbind(X,Y)))

## Principal component analysis

plot_component=function(a,b){
  plot(Z$x[,a],Z$x[,b],main=paste("Plot of the components",a,b),
       xlab=paste("Principal Component",a),
       ylab=paste("Principal Component",b),xlim=c(min(Z$x[,a]),
       max(Z$x[,a])),ylim=c(min(Z$x[,b]),
       max(Z$x[,b])),pch=1+as.integer(Y),col=rgb(0,0,as.integer(Y),.5))
  legend("topleft",c("Betrieb","Ausfall"),col=c(1,4),pch=1:2)
  lines(c(mean(Z$x[!Y,a]),mean(Z$x[Y,a])),c(mean(Z$x[!Y,b]),mean(Z$x[Y,b])),
        type='p',pch=c(16,17),cex=5,col=c(rgb(.2,.2,.2),4))}

plot_circle_correlation=function(a,b){
  mm="Circle of correlations"
  plot(cos(seq(0,2*pi,.01)),sin(seq(0,2*pi,.01)),main=mm,
       xlab=paste("Principal Composant",a),ylab=paste("Principal Component",b),
       type='l',xlim=c(-1,1),ylim=c(-1,1))
  for(i in 1:length(cor(X,Z$x)[,b])){
    segments(0,0,cor(X,Z$x)[i,a],cor(X,Z$x)[i,b],col=i);
    lines(cor(X,Z$x)[i,a],cor(X,Z$x)[i,b],type='p',pch=i,col=i)}
  cc=1:ncol(X);legend("topleft",names(X)[cc],pch=cc,col=cc,cex=1,bty='n')}

Z=prcomp(X,scale.=T);summary(Z);cor(X,Z$x)

nbPC=5
PC=as.data.frame(Z$x[,1:nbPC])

par(mfrow=c(2,3))
plot(Z$sdev^2/sum(Z$sdev^2),xlab="Principal Components",
     ylab="Proportion of Variance",type='h',col=2)
plot_component(1,2)
plot_component(1,3)
plot_circle_correlation(1,2)
plot_circle_correlation(1,3)
corrplot(cor(cbind(PC,Y)))
```

Abbildung B.1: Korrelationsanalyse und Hauptkomponentenanalyse

Anhang C

```
## Klassifizierung

MSE=function(M,Y,k){
  mse=mean((M-Y)^2)
  if(k==2) mse=mean((M[!Y]-Y[!Y])^2)
  if(k==3) mse=mean((M[Y]-Y[Y])^2)
  mse}

algo_name=function(){
  axis(1,at=c(1,3,5,7),lab=c("REG     ","NB","NN","   SVM"))
  axis(1,at=c(2,4,6),lab=c("LDA ","DT","RF"))}

par(mfrow=c(3,3))

B=20;K=3
for(k in 1:K){

  train=matrix(0,B,7)
  test=matrix(0,B,7)

  for(b in 1:B){

    ## Cross-validation sampling
    cc=NULL;n=nrow(X)
    cc[1:n]=T;cc[sample(1:n,.2*n)]=F

    ## Logistische Regression
    algo_REG=glm(Y[cc]~.,data=PC[cc,],family=binomial(logit))
    train[b,1]=MSE(predict(algo_REG,PC[cc,])>0,Y[cc],k)
    test[b,1]=MSE(predict(algo_REG,PC[!cc,])>0,Y[!cc],k)

    ## Lineare Diskriminantfunktion
    algo_LDA=lda(Y[cc]~.,data=PC[cc,])
    train[b,2]=MSE(as.numeric(predict(algo_LDA,PC[cc,])$class)-1,Y[cc],k)
    test[b,2]=MSE(as.numeric(predict(algo_LDA,PC[!cc,])$class)-1,Y[!cc],k)

    ## Naive-Bayes
    algo_NB=NaiveBayes(as.factor(Y[cc])~.,data=PC[cc,])
    train[b,3]=MSE(round(predict(algo_NB,PC[cc,])$posterior[,2]),Y[cc],k)
    test[b,3]=MSE(round(predict(algo_NB,PC[!cc,])$posterior[,2]),Y[!cc],k)

    ## Decision tree
    algo_DT=rpart(Y[cc]~.,data=X[cc,],method="class")
    train[b,4]=MSE(as.numeric(predict(algo_DT,X[cc,],type="class"))-1,Y[cc],k)
    test[b,4]=MSE(as.numeric(predict(algo_DT,X[!cc,],type="class"))-1,Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(2),
                        err.fct="sse",threshold=0.1)
      if(!is.null(algo_NN$result.matrix)) break}
    train[b,5]=MSE(round(predict(algo_NN,PC[cc,]))[,1],Y[cc],k)
    test[b,5]=MSE(round(predict(algo_NN,PC[!cc,]))[,1],Y[!cc],k)
```

Abbildung C.1: Klassifizierung der Betriebszustände mit statistichen Methoden und Algorithmen des maschinellen Lernens

```
    ## Random Forest
    algo_RF=randomForest(as.factor(Y[cc])~.,data=X[cc,])
    train[b,6]=MSE(as.numeric(predict(algo_RF,X[cc,]))-1,Y[cc],k)
    test[b,6]=MSE(as.numeric(predict(algo_RF,X[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=1,gamma=1)
    train[b,7]=MSE(as.numeric(predict(algo_SVM,PC[cc,]))-1,Y[cc],k)
    test[b,7]=MSE(as.numeric(predict(algo_SVM,PC[!cc,]))-1,Y[!cc],k)
  }

  fehler_train=apply(train,2,mean)
  print(fehler_train)
  fehler_test=apply(test,2,mean)
  print(fehler_test)
  yl=range(train,test)
  main1="Total error";if(k==2)main1="Error 1";if(k==3)main1="Error 2"
  plot(apply(train,2,mean),xlab="",ylab="Mean boostrap-MSE",xaxt='n',
       ylim=yl,main=main1);algo_name()
  lines(apply(test,2,mean),type='p',col=4)
  legend("bottomleft",c("Training","Testing"),pch=c(1,1),col=c(1,4))
  boxplot(train,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',
          ylim=yl,main="Training");algo_name()
  boxplot(test,xlab="",ylab="boostrap-MSE: boxplot",
          xaxt='n',ylim=yl,col=4,main="Test");algo_name()
}
```

Abbildung C.2: Klassifizierung der Betriebszustände mit statistichen Methoden und Algorithmen des maschinellen Lernens

Anhang D

```
## Classification

MSE=function(M,Y,k){
  mse=mean((M-Y)^2)
  if(k==2) mse=mean((M[!Y]-Y[!Y])^2)
  if(k==3) mse=mean((M[Y]-Y[Y])^2)
  mse}

algo_name=function(){
  axis(1,at=c(1,3,5,7),lab=c("1      ","3","5","   7"))
  axis(1,at=c(2,4,6),lab=c("2 ","4","6"))}

par(mfrow=c(3,3))

B=20;K=3
for(k in 1:K){

  train=matrix(0,B,7)
  test=matrix(0,B,7)

  for(b in 1:B){

    ## Cross-validation sampling
    cc=NULL;n=nrow(X)
    cc[1:n]=T;cc[sample(1:n,.2*n)]=F

    ## Neural network
    repeat{
      algo_NN_1=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_1$result.matrix)) break}
    train[b,1]=MSE(round(predict(algo_NN_1,PC[cc,]))[,1],Y[cc],k)
    test[b,1]=MSE(round(predict(algo_NN_1,PC[!cc,]))[,1],Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN_2=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,1),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_2$result.matrix)) break}
    train[b,2]=MSE(round(predict(algo_NN_2,PC[cc,]))[,1],Y[cc],k)
    test[b,2]=MSE(round(predict(algo_NN_2,PC[!cc,]))[,1],Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN_3=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,2),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_3$result.matrix)) break}
    train[b,3]=MSE(round(predict(algo_NN_3,PC[cc,]))[,1],Y[cc],k)
    test[b,3]=MSE(round(predict(algo_NN_3,PC[!cc,]))[,1],Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN_4=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,3),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_4$result.matrix)) break}
    train[b,4]=MSE(round(predict(algo_NN_4,PC[cc,]))[,1],Y[cc],k)
    test[b,4]=MSE(round(predict(algo_NN_4,PC[!cc,]))[,1],Y[!cc],k)
```

Abbildung D.1: Klassifizierung der Betriebszustände mit Neural Network und verschiedenen Hyperparametern

```
    ## Neural network
    repeat{
      algo_NN_5=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,4),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_5$result.matrix)) break}
    train[b,5]=MSE(round(predict(algo_NN_5,PC[cc,]))[,1],Y[cc],k)
    test[b,5]=MSE(round(predict(algo_NN_5,PC[!cc,]))[,1],Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN_6=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,5),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_6$result.matrix)) break}
    train[b,6]=MSE(round(predict(algo_NN_6,PC[cc,]))[,1],Y[cc],k)
    test[b,6]=MSE(round(predict(algo_NN_6,PC[!cc,]))[,1],Y[!cc],k)

    ## Neural network
    repeat{
      algo_NN_7=neuralnet(Y[cc]~.,data=PC[cc,],linear.out=F,h=c(8,6),
                          err.fct="sse",threshold=.5)
      if(!is.null(algo_NN_7$result.matrix)) break}
    train[b,7]=MSE(round(predict(algo_NN_7,PC[cc,]))[,1],Y[cc],k)
    test[b,7]=MSE(round(predict(algo_NN_7,PC[!cc,]))[,1],Y[!cc],k)

  }

  fehler_train=apply(train,2,mean)
  print(fehler_train)
  fehler_test=apply(test,2,mean)
  print(fehler_test)
  yl=range(train,test)
  main1="Total error";if(k==2)main1="Error 1";if(k==3)main1="Error 2"
  plot(apply(train,2,mean),xlab="",ylab="Mean boostrap-MSE",xaxt='n'
       ,ylim=yl,main=main1);algo_name()
  lines(apply(test,2,mean),type='p',col=4)
  legend("topleft",c("Training","Testing"),pch=c(1,1),col=c(1,4))
  boxplot(train,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl
          ,main="Training");algo_name()
  boxplot(test,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          col=4,main="Test");algo_name()
}
```

Abbildung D.2: Klassifizierung der Betriebszustände mit Neural Network und verschiedenen Hyperparametern

Anhang E

```
## Classification

MSE=function(M,Y,k){
  mse=mean((M-Y)^2)
  if(k==2) mse=mean((M[!Y]-Y[!Y])^2)
  if(k==3) mse=mean((M[Y]-Y[Y])^2)
  mse}

algo_name=function(){
  axis(1,at=c(1,3,5,7),lab=c("0.1      ","5","20","    100"))
  axis(1,at=c(2,4,6),lab=c("1 ","10","50"))}

par(mfrow=c(3,3))

B=20;K=3
for(k in 1:K){

  train=matrix(0,B,7)
  test=matrix(0,B,7)

  for(b in 1:B){

    ## Cross-validation sampling
    cc=NULL;n=nrow(X)
    cc[1:n]=T;cc[sample(1:n,.2*n)]=F

    ## Support Vector Machine
    algo_SVM_0.1=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=0.1)
    train[b,1]=MSE(as.numeric(predict(algo_SVM_0.1,PC[cc,]))-1,Y[cc],k)
    test[b,1]=MSE(as.numeric(predict(algo_SVM_0.1,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_1=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=1)
    train[b,2]=MSE(as.numeric(predict(algo_SVM_1,PC[cc,]))-1,Y[cc],k)
    test[b,2]=MSE(as.numeric(predict(algo_SVM_1,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_10=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=5)
    train[b,3]=MSE(as.numeric(predict(algo_SVM_10,PC[cc,]))-1,Y[cc],k)
    test[b,3]=MSE(as.numeric(predict(algo_SVM_10,PC[!cc,]))-1,Y[!cc],k)


    ## Support Vector Machine
    algo_SVM_50=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=10)
    train[b,4]=MSE(as.numeric(predict(algo_SVM_50,PC[cc,]))-1,Y[cc],k)
    test[b,4]=MSE(as.numeric(predict(algo_SVM_50,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_100=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=20)
    train[b,5]=MSE(as.numeric(predict(algo_SVM_100,PC[cc,]))-1,Y[cc],k)
    test[b,5]=MSE(as.numeric(predict(algo_SVM_100,PC[!cc,]))-1,Y[!cc],k)
```

Abbildung E.1: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Cost-Parametern

```
    ## Support Vector Machine
    algo_SVM_500=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=50)
    train[b,6]=MSE(as.numeric(predict(algo_SVM_500,PC[cc,]))-1,Y[cc],k)
    test[b,6]=MSE(as.numeric(predict(algo_SVM_500,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_1000=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=100)
    train[b,7]=MSE(as.numeric(predict(algo_SVM_1000,PC[cc,]))-1,Y[cc],k)
    test[b,7]=MSE(as.numeric(predict(algo_SVM_1000,PC[!cc,]))-1,Y[!cc],k)
  }

  fehler_train=apply(train,2,mean)
  print(fehler_train)
  fehler_test=apply(test,2,mean)
  print(fehler_test)
  yl=range(train,test)
  main1="Total error";if(k==2)main1="Error 1";if(k==3)main1="Error 2"
  plot(apply(train,2,mean),xlab="",ylab="Mean boostrap-MSE",xaxt='n',
       ylim=yl,main=main1);algo_name()
  lines(apply(test,2,mean),type='p',col=4)
  legend("bottomleft",c("Training","Testing"),pch=c(1,1),col=c(1,4))
  boxplot(train,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          main="Training");algo_name()
  boxplot(test,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          col=4,main="Test");algo_name()
}
```

Abbildung E.2: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Cost-Parametern

Anhang F

```
## Classification

MSE=function(M,Y,k){
  mse=mean((M-Y)^2)
  if(k==2) mse=mean((M[!Y]-Y[!Y])^2)
  if(k==3) mse=mean((M[Y]-Y[Y])^2)
  mse}

algo_name=function(){
  axis(1,at=c(1,3,5,7),lab=c("0.01     ","0.5","0.8","   5"))
  axis(1,at=c(2,4,6),lab=c("0.1 ","0.6","1"))}

par(mfrow=c(3,3))

B=20;K=3
for(k in 1:K){

  train=matrix(0,B,7)
  test=matrix(0,B,7)

  for(b in 1:B){

    ## Cross-validation sampling
    cc=NULL;n=nrow(X)
    cc[1:n]=T;cc[sample(1:n,.2*n)]=F

    ## Support Vector Machine
    algo_SVM_0.01=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=0.01)
    train[b,1]=MSE(as.numeric(predict(algo_SVM_0.01,PC[cc,]))-1,Y[cc],k)
    test[b,1]=MSE(as.numeric(predict(algo_SVM_0.01,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_0.1=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=0.1)
    train[b,2]=MSE(as.numeric(predict(algo_SVM_0.1,PC[cc,]))-1,Y[cc],k)
    test[b,2]=MSE(as.numeric(predict(algo_SVM_0.1,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_0.5=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=0.5)
    train[b,3]=MSE(as.numeric(predict(algo_SVM_0.5,PC[cc,]))-1,Y[cc],k)
    test[b,3]=MSE(as.numeric(predict(algo_SVM_0.5,PC[!cc,]))-1,Y[!cc],k)


    ## Support Vector Machine
    algo_SVM_0.6=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=0.6)
    train[b,4]=MSE(as.numeric(predict(algo_SVM_0.6,PC[cc,]))-1,Y[cc],k)
    test[b,4]=MSE(as.numeric(predict(algo_SVM_0.6,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_0.8=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=0.8)
    train[b,5]=MSE(as.numeric(predict(algo_SVM_0.8,PC[cc,]))-1,Y[cc],k)
    test[b,5]=MSE(as.numeric(predict(algo_SVM_0.8,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_1=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=1)
    train[b,6]=MSE(as.numeric(predict(algo_SVM_1,PC[cc,]))-1,Y[cc],k)
    test[b,6]=MSE(as.numeric(predict(algo_SVM_1,PC[!cc,]))-1,Y[!cc],k)
```

Abbildung F.1: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Gamma-Parametern

```
    ## Support Vector Machine
    algo_SVM_5=svm(as.factor(Y[cc])~.,data=PC[cc,],gamma=5)
    train[b,7]=MSE(as.numeric(predict(algo_SVM_5,PC[cc,]))-1,Y[cc],k)
    test[b,7]=MSE(as.numeric(predict(algo_SVM_5,PC[!cc,]))-1,Y[!cc],k)
  }

  fehler_train=apply(train,2,mean)
  print(fehler_train)
  fehler_test=apply(test,2,mean)
  print(fehler_test)
  yl=range(train,test)
  main1="Total error";if(k==2)main1="Error 1";if(k==3)main1="Error 2"
  plot(apply(train,2,mean),xlab="",ylab="Mean boostrap-MSE",xaxt='n',
       ylim=yl,main=main1);algo_name()
  lines(apply(test,2,mean),type='p',col=4)
  legend("bottomleft",c("Training","Testing"),pch=c(1,1),col=c(1,4))
  boxplot(train,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          main="Training");algo_name()
  boxplot(test,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          col=4,main="Test");algo_name()
}
```

Abbildung F.2: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Gamma-Parametern

Anhang G

```
## Classification

MSE=function(M,Y,k){
  mse=mean((M-Y)^2)
  if(k==2) mse=mean((M[!Y]-Y[!Y])^2)
  if(k==3) mse=mean((M[Y]-Y[Y])^2)
  mse}

algo_name=function(){
  axis(1,at=c(1,3,5,7),lab=c("1      ","3","5","    7"))
  axis(1,at=c(2,4,6),lab=c("2 ","4","6"))}

par(mfrow=c(3,3))

B=20;K=3
for(k in 1:K){

  train=matrix(0,B,7)
  test=matrix(0,B,7)

  for(b in 1:B){

    ## Cross-validation sampling
    cc=NULL;n=nrow(X)
    cc[1:n]=T;cc[sample(1:n,.2*n)]=F

    ## Support Vector Machine
    algo_SVM_0.1=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=1,gamma=0.8)
    train[b,1]=MSE(as.numeric(predict(algo_SVM_0.1,PC[cc,]))-1,Y[cc],k)
    test[b,1]=MSE(as.numeric(predict(algo_SVM_0.1,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_1=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=1,gamma=1)
    train[b,2]=MSE(as.numeric(predict(algo_SVM_1,PC[cc,]))-1,Y[cc],k)
    test[b,2]=MSE(as.numeric(predict(algo_SVM_1,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_10=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=10,gamma=0.8)
    train[b,3]=MSE(as.numeric(predict(algo_SVM_10,PC[cc,]))-1,Y[cc],k)
    test[b,3]=MSE(as.numeric(predict(algo_SVM_10,PC[!cc,]))-1,Y[!cc],k)


    ## Support Vector Machine
    algo_SVM_50=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=50,gamma=0.8)
    train[b,4]=MSE(as.numeric(predict(algo_SVM_50,PC[cc,]))-1,Y[cc],k)
    test[b,4]=MSE(as.numeric(predict(algo_SVM_50,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_100=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=50,gamma=1)
    train[b,5]=MSE(as.numeric(predict(algo_SVM_100,PC[cc,]))-1,Y[cc],k)
    test[b,5]=MSE(as.numeric(predict(algo_SVM_100,PC[!cc,]))-1,Y[!cc],k)

    ## Support Vector Machine
    algo_SVM_500=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=100,gamma=0.8)
    train[b,6]=MSE(as.numeric(predict(algo_SVM_500,PC[cc,]))-1,Y[cc],k)
    test[b,6]=MSE(as.numeric(predict(algo_SVM_500,PC[!cc,]))-1,Y[!cc],k)
```

Abbildung G.1: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Cost-und Gamma-Parametern

```
    ## Support Vector Machine
    algo_SVM_1000=svm(as.factor(Y[cc])~.,data=PC[cc,],cost=100,gamma=1)
    train[b,7]=MSE(as.numeric(predict(algo_SVM_1000,PC[cc,]))-1,Y[cc],k)
    test[b,7]=MSE(as.numeric(predict(algo_SVM_1000,PC[!cc,]))-1,Y[!cc],k)
  }

  fehler_train=apply(train,2,mean)
  print(fehler_train)
  fehler_test=apply(test,2,mean)
  print(fehler_test)
  yl=range(train,test)
  main1="Total error";if(k==2)main1="Error 1";if(k==3)main1="Error 2"
  plot(apply(train,2,mean),xlab="",ylab="Mean boostrap-MSE",xaxt='n',
       ylim=yl,main=main1);algo_name()
  lines(apply(test,2,mean),type='p',col=4)
  legend("topleft",c("Training","Testing"),pch=c(1,1),col=c(1,4))
  boxplot(train,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          main="Training");algo_name()
  boxplot(test,xlab="",ylab="boostrap-MSE: boxplot",xaxt='n',ylim=yl,
          col=4,main="Test");algo_name()
}
```

Abbildung G.2: Klassifizierung der Betriebszustände mit Support Vector Machine und verschiedenen Cost-und Gamma-Parametern