HAW HAMBURG

BACHELORTHESIS
Yannick Pascal Eisenschmidt

Vergleich unterschiedlicher Machine-Learning-Verfahren für das Voraussagen und Erklären der Ausfälle von Produktionsleitsystemen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Yannick Pascal Eisenschmidt

Vergleich unterschiedlicher Machine-Learning-Verfahren für das Voraussagen und Erklären der Ausfälle von Produktionsleitsystemen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr. Peer Stelldinger

Eingereicht am: 5. Juli 2022

Yannick Pascal Eisenschmidt

Thema der Arbeit

Vergleich unterschiedlicher Machine-Learning-Verfahren für das Voraussagen und Erklären der Ausfälle von Produktionsleitsystemen

Stichworte

Predictive-Maintenance, Explainable-AI, Machine-Learning, Produktionsleitsystem

Kurzzusammenfassung

Die Machine-Learning-Verfahren Entscheidungsbaum, Feed-Forward Netz und LSTM werden verwendet, um anhand von Sensordaten die Zeit bis zum nächsten Ausfall eines Produktionsleitsystems vorherzusagen. Modell-spezifische Erklärungsverfahren werden angewendet, um den für die Vorhersage verwendeten Sensorwerten ein Wichtigkeitsmaß zuzuordnen, was die Identifikation des Ausfallgrundes unterstützen soll. Durch Betrachtung der Vorhersagegenauigkeit und exemplarischer Betrachtung produzierter Erklärungen werden das LSTM-Netzwerk und das angepasste LRP-Verfahren als am für den Anwendungsfall geeignetsten ausgewählt.

Yannick Pascal Eisenschmidt

Title of Thesis

Comparison of Machine Learning Aproaches for Predicting and Explaining Faults of Manufacturing Execution Systems

Keywords

Predictive Maintenance, Explainable AI, Machine Learning, Manufacturing Execution System

Abstract

The machine learning methods decision tree, feed forward network and LSTM are used to predict the remaining useful life of an MES using collected sensor data. Model specific explanation methods are used to assign a relevancy to each sensor which helps identify the

cause of an outage. By comparing prediction accuracy and checking sample explanations the LSTM network and the modified LRP method are determined to be most suitable for the use-case.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

Diese Arbeit wurde in Zusammenarbeit mit dem Unternehmen Körber Pharma Software GmbH verfasst. Das Unternehmen vertreibt ein Produktionsleitsystem. Diese Art von System wird verwendet, um Herstellungsprozesse in Fabriken mit digitaler Unterstützung steuern zu können.

Dieses Kapitel beschreibt eine Herausforderung im Betrieb eines solchen Systems. Es wird das Forschungsthema vorgestellt, welches die Bewältigung dieser Herausforderung als Ziel hat. Außerdem wird das Vorgehen erklärt, mit welchem das Forschungsthema bewältigt wurde.

1.1 Problembeschreibung

Fertigungsprozesse sind von einer Reihe von verschiedenen Tätigkeiten abhängig. Diese Tätigkeiten lassen sich grob in drei Kategorien aufteilen.

Company-Management Unterstützende Tätigkeiten wie Personalverwaltung, Buchhaltung oder Marketing gehörten in diese Kategorie. Häufig werden sie in einem Enterprise-Resource-Planning (ERP) System abgebildet.

Production-Management Sowohl die Entwicklung neuer Produkte, als auch die Produktion dieser gehören in diese Kategorie. Für die eigentliche Produktion können Produktionsleitsysteme verwendet werden. Sie werden häufig auch Manufacturing-Execution-System (MES) genannt und sind dafür verantwortlich Produktionsprozesse digital abzubilden. Dafür müssen sie Aufträge und Ressourcen verwalten sowie Performance-Daten erfassen und speichern.

Production-Control & Automation Die für Fertigung eingesetzten Maschinen müssen gesteuert werden. Diese Kategorie an Tätigkeiten ist dafür verantwortlich. Im Fall

von vollständig automatisierten Arbeitsschritten können die notwendigen Parameter zum Beispiel über Supervisory-Control-and-Data-Acquisition (SCADA)-Systeme gesetzt werden.

[6, S. 8ff.]

Da Herstellungsprozesse auf die Verfügbarkeit eines MES, wie es von Körber Pharma Software vertrieben wird, angewiesen sind, können Ausfälle dieses Systems zu einem Produktionsstillstand führen. Aus verschiedenen Gründen — sei es eine Überlastung des Systems oder ein Programmierfehler in einer Anwendung — treten jedoch Ausfälle des Systems auf. Durch einen guten Entwicklungsprozess und robuste Software lassen sich diese Ausfälle minimieren, ausschließen kann man sie jedoch nicht. Um eine geeignete Gegenmaßnahme wie einen geregelten Neustart einleiten zu können, wäre eine Vorhersage, wann und warum ein Ausfall auftreten wird, vorteilhaft.

1.2 Forschungsthema

Das System, für welches Vorhersagen getroffen werden sollen, wird in verschiedenen Versionen und zum Teil mit individuellen Anpassungen bei den Kunden eingesetzt. Körber Pharma Software vertreibt neben dem MES auch eine Monitoring-Anwendung, welche bei einigen Kunden eingesetzt wird und in regelmäßigen Abständen Daten für verschiedene Metriken erfasst. Viele Kunden stellen die gesammelten Daten zur Verfügung.

Das Ziel der Arbeit soll es sein, mithilfe dieser Sensordaten ein Modell zu entwickeln, welches zum einen die Zeit bis zum nächsten Ausfall abschätzt und zum anderen eine Erklärung für die getroffene Schätzung gibt. Da ein möglichst geeignetes Modell entwickelt werden soll, werden verschiedene Verfahren verwendet, um die Problemstellung zu lösen. Die Ergebnisse der resultierenden Modelle werden verglichen, um die Verfahren auszuwählen, welche zum vielversprechendsten Resultat führen.

1.3 Vorgehen

Die Problemstellung besteht aus zwei Teilproblemen. Zum einen soll vorhergesagt werden können, *wann* der nächste Ausfall eintritt. Hierfür kann eine Vielzahl von Machine-Learning (ML)-Verfahren verwendet werden. Da es viele Auswahlmöglichkeiten gibt, soll-

te eine Vorauswahl getroffen werden. Mit jedem gewählten Verfahren soll dann ein Modell trainiert werden, welches die gewünschten Vorhersagen möglichst genau treffen kann.

Zum anderen soll ermittelt werden können, *warum* ein Ausfall für einen bestimmten Zeitpunkt vorhergesagt wurde. Für jedes in der Vorauswahl gewählte Verfahren muss also auch ein Erklärungsverfahren ausgewählt werden. Für die Lösung dieses zweiten Teilproblems ist es notwendig, überhaupt erst Vorhersagen treffen zu können, weshalb zuerst die Modelle trainiert werden müssen.

Für die Lösung beider Teilprobleme ist ein geeigneter Datenbestand notwendig. Daher muss im ersten Schritt eine Datenaufbereitung, wie sie in Kapitel 3 beschrieben wird, erfolgen.

2 Theoretischer Rahmen

Das Thema der Arbeit ist nun definiert. Es befasst sich mit Aspekten, die bereits in existierenden Forschungsbereichen behandelt werden. Im Folgenden wird erklärt, warum Predictive-Maintenance, Machine-Learning und Explainable-AI relevant für die Lösung der Problemstellung sind. Außerdem wird aufgezeigt, welche Methoden aus diesen Bereichen jeweils genutzt werden können. Mit diesem Hintergrundwissen kann dann mit der Entwicklung der Problemlösungen begonnen werden.

2.1 Predictive-Maintenance

Ein kontinuierlicher Betrieb von Fertigungsanlagen ist notwendig, um einen stetigen Produktionsfluss zu gewährleisten. Wenn die Fertigung eines Produktes durch Ausfälle eingeschränkt ist, hat dies einen Umsatzverlust zur Folge. Die Instandhaltung der Fertigungsanlagen ist daher wichtig für das Bestehen des Unternehmens. Mit der Instandhaltung sind jedoch Reparaturkosten verbunden. Mit einer Wartungsstrategie sollen die Reparaturkosten und Ausfallzeiten minimiert werden. Zwei offensichtliche Wartungsstrategien sind *Reactive-Maintenance* und *Preventive-Maintenance*. Bei ersterem werden Anlagen erst dann repariert, sobald sie nicht mehr funktionsfähig sind. Dadurch fallen zwar weniger Wartungseinsätze an, jedoch fallen so Anlagen häufiger aus. Beim Preventive-Maintenance werden solche Ausfälle durch regelmäßig durchgeführte Wartungen verringert. Durch *Predictive-Maintenance* werden die Vorteile beider Strategien kombiniert, indem Anlagen dann gewartet werden, wenn sie kurz davor sind, auszufallen. [9, S. 1]

Um diese Wartungseinsätze zu planen, ist es notwendig, den Status der betroffenen Maschine kontinuierlich mit Sensoren zu überwachen. Dabei werden zum Beispiel Temperaturen oder Vibrationen der Maschinen gemessen. Mithilfe der Sensordaten werden Fehler erkannt und vorhergesagt. [9, S. 5]

Um bevorstehende Ausfälle zu erkennen, wird ein Modell verwendet, welches die Sensordaten als Eingabe erhält und eine Ausgabe produziert. Diese Ausgabe kann unterschiedlicher Art sein.

- Sie gibt an, ob die überwachte Maschine innerhalb eines festen Zeitfensters ausfallen wird.
- Sie gibt an, wie lange die Maschine voraussichtlich noch im Betrieb bleiben wird. Diese Ausgabe kann in Tagen, Stunden oder anderen Perioden angegeben werden und wird als Remaining-Useful-Life (RUL) bezeichnet.
- Es wird ungewöhnliches Verhalten identifiziert. Diese Anomaly-Detection kann dann sinnvoll sein, wenn keine vorherigen Ausfälle bekannt sind.

[9, S. 3]

Üblicherweise wird Predictive-Maintenance eingesetzt, um physische Maschinen zu überwachen. Die installierten Sensoren erfassen dann zum Beispiel die Vibration oder Temperatur einzelner Komponenten. Allerdings kann man dieses Konzept auch auf Software übertragen. Die Maßnahmen zur Instandhaltung umfassen dann zum Beispiel das Durchführen eines geregelten Neustarts, statt des Austauschens einer fehlerhaften Komponente. Die Sensoren erfassen dann statt zum Beispiel Temperaturen einzelner Komponenten, Informationen, die üblicherweise von Monitoring-Anwendungen erfasst werden. Dazu gehören zum Beispiel CPU- oder Speicherauslastungen.

Um die Frage, *wann* mit einem Ausfall des MES zu rechnen ist, zu beantworten, kann das beschriebene Vorgehen angewendet werden.

2.2 Machine-Learning

Es gibt viele verschiedene Definitionen von Artificial-Intelligence (AI). Einige davon definieren AI über das Ziel, Maschinen zu erschaffen, die sich intelligent verhalten oder zumindest so wirken. ML ist eine Teildisziplin von AI, die sich damit befasst, Muster in Beobachtungen zu erkennen und basierend auf den Erkenntnissen zu reagieren. [10, S. 2]

Statt Regeln manuell aufzustellen, mit welchen der Zustand des überwachten Systems beurteilt werden kann, können Machine-Learning Algorithmen verwendet werden. Mit

diesen Algorithmen werden Modelle erzeugt, welche die richtigen Regeln erlernt haben. Die Verwendung solcher Algorithmen hat mehrere Vorteile [10, S. 693].

- Ein Entwickler muss nicht alle möglichen Situationen, die eintreten können, kennen, um diese abzudecken.
- Ein ML-Modell kann Veränderungen über die Zeit erlernen und muss nicht manuell angepasst werden.
- Die Regeln des Anwendungsgebiets müssen dem Entwickler des Modells nicht bekannt sein.

Diese Vorteile machen ML-Algorithmen für die Vorhersage von Ausfällen attraktiv. Die Situationen, in denen Ausfälle auftreten können sind nämlich vielfältig. Regeln, nach welchen Ausfälle auftreten, sind nicht immer offensichtlich. Darüber hinaus können sich diese auch verändern, wenn neue Installationen des Systems betrachtet werden.

Es gibt verschiedene Klassen von Lernverfahren, die sich dadurch unterscheiden, wie mit Feedback umgegangen wird. [10, S. 694f.]

Unsupervised-Learning Bei dieser Klasse von Verfahren gibt es keinen Feedbackmechanismus. Die Algorithmen dieser Klasse lernen nur basieren auf den Eingabedaten.

Reinforcement-Learning arbeitet mit Bestrafungen beziehungsweise Belohnungen, die basierend auf der zu einer Eingabe ausgewählten Rückmeldung ausgestellt wird.

Supervised-Learning verwendet Eingabedaten, für welche die tatsächliche Ausgabe bereits bekannt ist. Die verwendeten Algorithmen verwenden diese Informationen, um ein Modell zu erstellen, welches möglichst dieselben Ausgaben erzeugt.

In diesem Anwendungsfall, welcher sich mit der Vorhersage von Ausfällen eines Systems beschäftigt, stellen die Werte der überwachten Sensoren die Eingabedaten dar. Die dazu passenden Ausgaben können, wie in Kapitel 3.2.2 beschrieben wird, hergeleitet werden. Da also Eingabedaten und die zu erzielenden Ausgaben bekannt sind, sollen Lernverfahren des Supervised-Learning angewendet werden.

Wenn die gewünschten Ausgaben bei der Anwendung eines Supervised-Learning Algorithmus aus einem kontinuierlichen Wertebereich stammen, handelt es sich um ein Regressionsproblem. Von einem Klassifikationsproblem redet man, wenn die möglichen Ausgaben einer festen Menge an Kategorien entstammen. [10, S. 696]

Das Vorhersagen von Systemausfällen lässt sich sowohl als Regressions- als auch als Klassifikationsproblem formulieren. Man kann entweder Fragen, wann ein System ausfällt. Die Antwort bestünde in diesem Fall aus der Angabe einer Zeit, bis das System voraussichtlich ausfallen wird. Damit würde es sich um ein Regressionsproblem handeln. Fragt man stattdessen, ob das System zum Beispiel in den nächsten fünf Stunden ausfallen wird, handelt es sich um eine binäre Klassifikation.

Im ersten Fall ist der Informationsgehalt der Antwort höher, da so auch Ausfälle, die nach mehr als fünf Stunden zu erwarten sind, festgestellt werden können. Daher werden die untersuchten Algorithmen eingesetzt, um das Predictive-Maintenance Problem als Regression anzugehen.

Die ML-Algorithmen benötigen Daten, um passende Modelle zu erzeugen. Um die Qualität der Modelle zu ermitteln, werden ebenfalls Datensätze benötigt, mit welchen die Modell-Ausgaben und die erwarteten Ausgaben abgeglichen werden können. Die Cross-Validation Methode besagt, dass die insgesamt zur Verfügung stehenden Daten in zwei Datensätze für diese beiden Aufgaben aufgeteilt werden sollen. Da bei der Validierung des Modells dadurch ein unbekannter Datensatz verwendet wird, spiegelt die gemessene Qualität des Modells die bei Inbetriebnahme zu erwartende Qualität besser wider. [10, S. 708]

Durch diese Trennung des Datensatzes ist es außerdem möglich, Overfitting zu erkennen. Overfitting liegt vor, wenn ein Modell Muster aus den Trainingsdaten erlernt, die tatsächlich nicht existieren [10, S. 705]. Dieser Umstand äußert sich in einer guten Performance des Modells auf den Trainingsdaten, während das Modell die Validierungsdaten signifikant schlechter bewältigen kann.

Hyperparameter der Modelle werden üblicherweise basierend auf der Qualität der Ergebnisse des Trainingsdatensatzes angepasst. Die Modelle werden daher implizit darauf optimiert, möglichst gut Ergebnisse auf diesem Datensatz zu erzielen. Dieser Effekt wird dadurch vermieden, dass ein dritter Teil der verfügbaren Daten nur für das endgültige Testen des Modells verwendet wird. [10, S. 709] Dieses Vorgehen wird auch für die Entwicklung und Bewertung der später zu erstellenden Modelle angewendet.

2.3 Explainable-AI

Es gibt Anwendungsfälle, in denen neben der Genauigkeit eines Modells auch interessant ist, warum eine bestimmte Ausgabe auftritt oder wie diese zustande gekommen ist. Dazu gehören insbesondere Anwendungsfälle von Machine-Learning, bei denen ein Vertrauen in die Ausgaben des Modells wichtig ist. Beispielsweise das autonome Fahren stellt ein solches Anwendungsgebiet dar.

Explainable-AI (XAI) befasst sich damit, Erklärbarkeit und Interpretierbarkeit von Modellen herzustellen. Diese beiden Begriffe werden häufig äquivalent verwendet und haben keine formelle Definition. Manchmal wird jedoch zwischen ihnen unterschieden. Die folgenden Auffassungen der beiden Begriffe werden in dieser Arbeit verwendet.

Interpretierbarkeit befasst sich mit den Zusammenhängen zwischen den Ein- und Ausgaben eines Modells. Ein Modell ist interpretierbar, wenn es den Grund für eine Ausgabe für einen Benutzer verständlich macht.

Erklärbarkeit bezeichnet, wie verständlich die internen Abläufe des Modells für einen Benutzer sind. Der Benutzer soll also verstehen, wie der Entscheidungsprozess abläuft.

[5, S. 2f.]

Das zweite Teilproblem der Forschungsfrage ist es, zu klären, warum ein Modell vorhersagt, dass ein Ausfall bevorsteht. Daher sind Möglichkeiten Interpretierbarkeit der verwendeten Modelle herzustellen, für diese Arbeit interessant.

Man unterteilt diese Möglichkeiten in lokale Verfahren, die eine Erklärung für ein bestimmtes Ergebnis liefern, und globale Verfahren, die versuchen, das gesamte Modell zu erklären [5, S. 5]. Der Grund für einen bevorstehenden Ausfall ist abhängig von den Eingaben, die zu der Vorhersage des Ausfalls geführt haben. Daher sind nur die lokalen Interpretationsverfahren in dieser Arbeit relevant.

Darüber hinaus unterscheidet man zwischen modell-agnostischen und modell-spezifische Verfahren. Einige Verfahren, die Interpretierbarkeit schaffen, sind nur auf bestimmte Modelle wie zum Beispiel neuronale Netze anwendbar, weil sie vom Aufbau des Modells abhängig sind. Sie sind also modell-spezifisch. Die modell-agnostischen Verfahren hingegen betrachten das Modell als Black-Box und können daher mit jedem Modell angewendet werden. Sie arbeiten lediglich mit den Ein- und Ausgaben des Modells. [5, S. 5]

Molnar stellt in Kapitel 8 von „Interpretable Machine-Learning“ lokale, modell-agnostische Interpretationsverfahren vor. Diese haben gemeinsam, dass für die Interpretationen mehrere Eingaben vom Modell verarbeitet werden müssen. Dadurch dauert es wesentlich länger, zu einer Eingabe eine Erklärung zu generieren, weshalb für die in dieser Arbeit betrachteten Modelle spezifische Interpretationsverfahren bevorzugt werden.

3 Datenaufbereitung

Bevor Machine-Learning Modelle erstellt werden können, um die Predictive-Maintenance Aufgabe zu lösen, muss die dafür notwendige Datengrundlage geschaffen werden. In diesem Kapitel wird zunächst beschrieben, in welcher Form die Sensordaten zur Verfügung stehen. Anschließend werden die Schritte dargelegt, welche notwendig sind, um die Daten in ein geeignetes Format zu bringen. Außerdem findet eine globale Betrachtung der Sensordaten statt, um erste Aussagen über den Datensatz treffen zu können. Das Resultat dieser Vorverarbeitung wird dann ein reduzierter Datensatz sein, welcher auf die grundlegenden Bedürfnisse der ML-Verfahren angepasst ist und somit für das Training von Modellen verwendet werden kann.

3.1 Beschreibung der Datenstruktur

Jede Installation des MES besteht aus mehreren Services, die ein gemeinsames System darstellen. Jeder Service kann über unterschiedliche Sensoren verfügen. Typische Installationen bestehen aus einer Datenbank, einem Service für Druckaufträge und einem Cluster an Anwendungsservern. Die Anwendungsserver verfügen über dieselben Sensoren. Deren Werte werden jedoch unabhängig voneinander erfasst. Abbildung 3.1 zeigt die Anzahl der Sensoren pro Service einer typischen Installation mit einem Cluster der Größe 3.

Die Sensoren erfassen nicht nur numerische Werte, sondern auch Text wie Produktbezeichnungen oder Hostnamen. Diese textuellen Informationen verändern sich jedoch während des Betriebs nicht und können daher vernachlässigt werden. Darüber hinaus werden vereinzelt auch komplexere Datenstrukturen wie Prozesslisten oder Umgebungsvariablen durch Sensoren erfasst. Um den Umfang gering zu halten, werden diese Sensoren ebenso vernachlässigt. Die übrigen Sensoren, welche in der Mehrzahl der Installationen zur Verfügung stehen, sind in Tabelle A.1 mit ihrer Bezeichnung und einer Beschreibung aufgelistet.

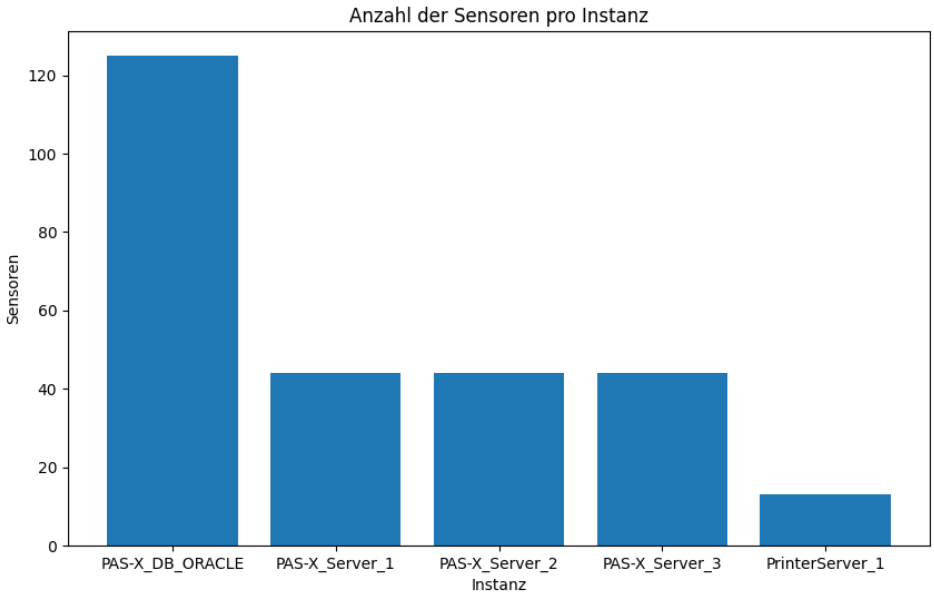


Abbildung 3.1: Anzahl der Sensoren pro Instanz einer Installation

Die Monitoring-Anwendung erfasst die Rohdaten der Sensoren von allen Instanzen und bündelt diese täglich in einem ZIP-Archiv. Pro Sensor gibt es eine Datei, welche Wertepaare bestehend aus einem Zeitstempel vom Erhebungszeitpunkt und dem Wert des Sensors enthält. Die Frequenz, mit welcher die Sensoren abgefragt werden, ist pro Sensor konfiguriert. Die Anzahl der Werte pro Datei ist also davon abhängig.

3.2 Schritte der Vorverarbeitung

Für das Erstellen von Machine-Learning Modellen ist ein Datensatz notwendig, welcher aus mehreren Einträgen besteht. Ein Eintrag soll aus mehreren Features und einem Label bestehen. Die Features sind die Sensordaten zu dem Zeitpunkt, welcher durch den Eintrag im Datensatz abgebildet wird. Das Label stellt die RUL der betrachteten Instanz dar.

Im Folgenden wird grob der Ablauf beschrieben, nach welchem die Sensordaten in das benötigte Format überführt wurden.

1. Die Sensordaten aller verfügbaren ZIP-Archive werden pro Instanz und Sensor in chronologischer Reihenfolge zusammengefügt. Ein Beispiel dafür ist durch die Ta-

belle 3.1b gegeben, welche die Einträge von Sensor A (Tabelle 3.1a) und Sensor B (Tabelle 3.1c) enthält.

2. Die unerwünschten Sensoren werden verworfen. Dazu gehören zum einen die in Kapitel 3.1 erwähnten Sensoren, welche keine numerischen Werte enthalten. Zum anderen gibt es auch Sensoren, welche keinen Informationsgehalt haben. Diese werden in Kapitel 3.3 identifiziert und sollten ebenso vernachlässigt werden.

3. Das niedrigste Intervall zwischen zwei Aufrufen eines Sensors beträgt eine Minute. Die Zeitstempel, welche mit einem Sensorwert erfasst werden, sind jedoch in Sekunden angegeben. Daher gibt es für jede Minute mehrere erfasste Zeitpunkte, zu welchen Werte von mehreren Sensoren erfasst werden. Für jeden einzelnen Sensor kann es in einer Minute jedoch nur einen Wert geben, weshalb alle Einträge innerhalb von einer Minute zu einem einzelnen zusammengefasst werden können. Dieser Schritt der Vorverarbeitung wurde mit den Beispieleinträgen in Tabelle 3.1b durchgeführt. Die resultierende Tabelle 3.1d enthält nur noch minütliche Einträge.

4. Der Datensatz besteht nun aus Einträgen im Minutenabstand. Jedoch gibt es nicht für jeden Sensor in jeder Minute einen Wert, da es Sensoren gibt, die beispielsweise nur jede Stunde abgerufen werden. Für das Training der Modelle ist es notwendig, für jeden Sensor zu jedem Zeitschritt einen Wert zu haben. Die Lücken werden durch unter der Berücksichtigung der vorherigen und nachfolgenden Werte gefüllt. Die verschiedenen Möglichkeiten dafür werden in Kapitel 3.2.1 verglichen.

5. Die Features des angestrebten Datensatzes sind durch die Sensoren bereits gegeben. Jedoch ist die RUL ist noch unbekannt. Sie wird aus einem der Sensoren berechnet, welcher anschließend nicht mehr zu den Sensoren zählt, aus denen die Feature-Daten bezogen werden. Das genaue Vorgehen ist in Kapitel 3.2.2 beschrieben.

Nach der Ausführung der beschriebenen Schritte sind die Mindestanforderungen an den Datensatz erfüllt. Für einige Machine-Learning Verfahren sind noch weitere Schritte notwendig, um ein geeignetes Modell zu erstellen. Diese werden in den modell-spezifischen Kapiteln beschrieben.

3.2.1 Berechnung der fehlenden Sensorwerte

Um die bei einigen Zeitschritten fehlenden Werte zu berechnen, werden drei Möglichkeiten in Betracht gezogen. Es könnte der letzte oder der nächste ausgelesene Sensorwert,

Tabelle 3.1: Beispielhafte Sensorwerte zur Veranschaulichung der Vorverarbeitung

(a) Beispiel-Feature A

Teitstempel	**Feature A**
1656842400	1
1656842460	2
1656842520	3
1656842580	4

(b) Sammlung aller Features

Teitstempel	**Feature A**	**Feature B**
1656842400	1	
1656842430		5
1656842460	2	
1656842490		6
1656842520	3	
1656842550		7
1656842580	4	
1656842610		8

(c) Beispiel-Feature B

Teitstempel	**Feature B**
1656842430	5
1656842490	6
1656842550	7
1656842610	8

(d) Kombination der Einträge

Datum	**Feature A**	**Feature B**
03.07.2022 12:00	1	5
03.07.2022 12:01	2	6
03.07.2022 12:02	3	7
03.07.2022 12:03	4	8

als Wert des betrachteten Zeitschritts verwendet werden. Alternativ kann zum Beispiel durch lineare Interpolation eine Schätzung basierend auch den bekannten Werten berechnet werden.

Die Verwendung des nächsten bekannten Wertes und alle Formen der Interpolation sind jedoch darauf angewiesen, dass ein Sensorwert eines späteren Zeitschritts bekannt ist. Das ist mit dem vorliegenden Datensatz möglich, jedoch spiegelt es nicht die vorgesehene Anwendung wider. Benutzt man nämlich ein trainiertes Modell, um in Echtzeit Vorhersagen über einen möglichen Ausfall zu treffen, liegen keine zukünftigen Sensorwerte vor. Es kann also nur auf die vorherigen Werte zugegriffen werden.

Aus diesem Grund werden die Lücken in den Sensordaten aufgefüllt, indem die Datensätze in chronologischer Reihenfolge durchlaufen werden und für unbekannte Werte der Wert des vorherigen Zeitschritts eingesetzt wird.

Für fehlende Werte vor dem Auftreten des ersten ausgelesenen Wertes eines Sensors kann jedoch kein Wert eingesetzt werden, da es keinen vorherigen gibt. Die ersten paar Datensätze mit derartigen Lücken müssen daher verworfen werden und können somit nicht für das Training der Modelle verwendet werden.

3.2.2 Aufbereitung der Remaining Useful Life

Jede Instanz verfügt über einen Sensor, welcher die „SystemUptime“ erfasst. Wenn dieser Sensor abgefragt wird, liefert er die Zeit in Millisekunden, die das System bereits verfügbar ist. Nach jedem Ausfall beginnt die Zählung wieder bei null.

Abbildung 3.2 zeigt die Werte dieses Sensors am Beispiel einer Installation. Die Ausfälle treten in dem betrachteten Zeitraum häufig parallel zueinander auf. Darüber hinaus ist zu erkennen, dass der dritte Anwendungsserver nach einer Weile endgültig außer Betrieb genommen wurde.

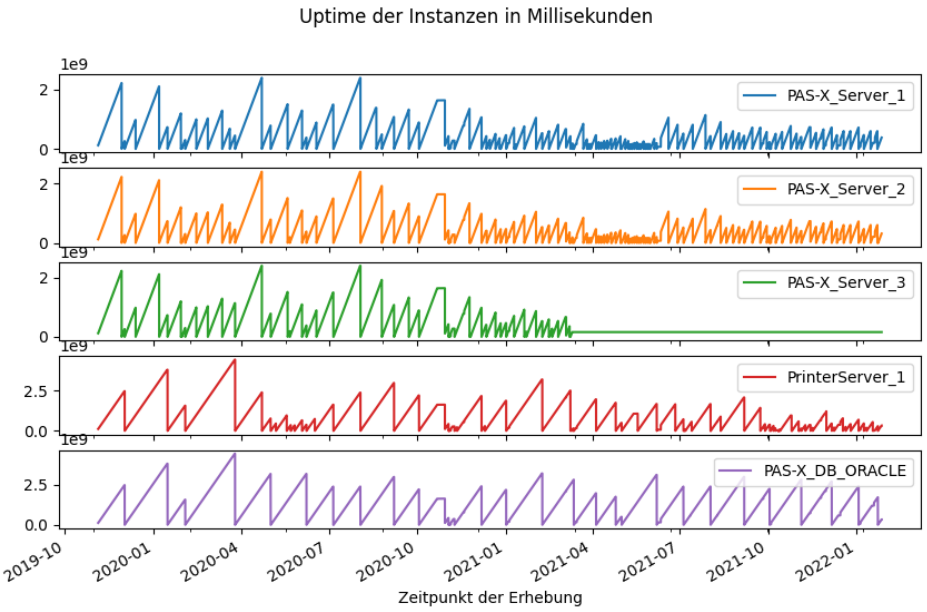


Abbildung 3.2: Verfügbarkeit der Services einer Installation

Der Verfügbarkeitssensor wird üblicherweise auf jeder Instanz einmal pro Stunde abgefragt. Wenn man ebenso wie bei den anderen Sensoren in jeder Lücke den letzten bekannten Wert einsetzt, ergibt sich eine sehr grobe Einstufung. Da allerdings bekannt ist, dass der „SystemUptime“ Sensor linear ansteigt, können die fehlenden Werte interpoliert werden. Im Gegensatz zu den restlichen Sensoren ist dies hier akzeptabel, da dieser Sensor nicht als Feature in den zu trainierenden Modellen verwendet wird.

In Abbildung 3.3 werden die beiden Möglichkeiten, fehlende Werte zu füllen, miteinander verglichen. Verwendet man den letzten bekannten Wert, so werden 60 aufeinander folgenden Zeitschritten dieselbe Verfügbarkeit zugeordnet. Die später daraus berechneten RULs werden dadurch ungenau, was den Fehler in den Vorhersagen erhöhen würde. Daher wird das im Folgenden beschriebene Verfahren verwendet, um die fehlenden Werte zu berechnen.

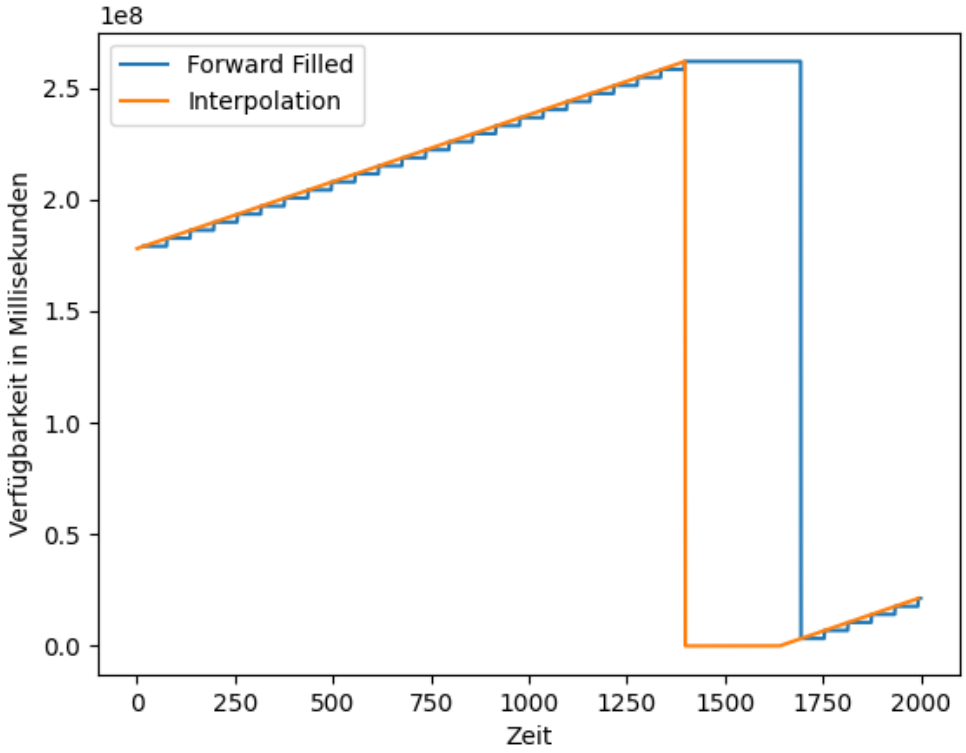


Abbildung 3.3: Vergleich der interpolierten mit den aufgefüllten Verfügbarkeitswerten

Weil jede Minute ein neuer Datensatz anfällt und die Verfügbarkeit in Millisekunden erfasst wird, gilt für die Steigung zwischen zwei Datenpunkten $\Delta t = 60 * 1000$, sofern das System in diesem Zeitraum nicht ausfällt. Wenn jedoch von zwei Datenpunkten der zweite eine geringere Verfügbarkeit angibt, ist zwischen den beiden Zeitpunkten ein Ausfall eingetreten. Mit der Steigung Δt kann für alle Zeitpunkte ab dem Neustart, für welche keine Verfügbarkeit erfasst wurde, ein Wert berechnet werden.

Da nicht bekannt ist, wie lange das System nicht verfügbar war, wird angenommen, dass der Ausfall beim letzten bekannten Datenpunkt eingetreten ist. Für die übrigen Zeitpunkte ohne eine Verfügbarkeit wird daher der Wert null angenommen. Aus diesem Grund ist der mit dem Interpolationsverfahren ermittelte Start des Ausfalls in Abbildung 3.3 vor dem, der sich durch das Ausfüllen der fehlenden Werte Ergeben würde.

Der Algorithmus 1 führt die beschriebene Interpolation durch, indem die Verfügbarkeiten rückwärts durchlaufen werden. Jeder durch den Sensor erfasste Verfügbarkeitswert wird beibehalten. Fehlende Werte werden berechnet, indem von dem vorherigen Wert die Steigung Δt subtrahiert wird (siehe Zeile 8). Dabei fällt die Verfügbarkeit jedoch nicht unter null.

Algorithmus 1 Berechnung der fehlenden Verfügbarkeitswerte

```
1:  function INTERPOLATE(uptime, slope = 60000)
2:      uptime ← REVERSE(uptime)
3:      previous ← uptime[0]
4:      interpolated ← [previous]
5:      for i ← 1, LENGTH(uptime) do
6:          value ← uptime[i]
7:          if values = NaN then
8:              value ← MAX(previous − slope, 0)
9:          end if
10:         previous ← value
11:         PREPEND(interpolated, value)
12:     end for
13:     return interpolated
14: end function
```

Die RUL kann aus diesen Sensordaten berechnet werden, indem die Zeit der Verfügbarkeit nicht hoch-, sondern runtergezählt wird. In den Zeitschritten, zu welchen eine Instanz nicht verfügbar ist, soll die Verfügbarkeit weiterhin null sein.

Für die Umkehrung des Verlaufs der Verfügbarkeit gibt es zwei Alternativen.

1. Alle Werte in einer Zeitspanne, in welcher die Instanz kontinuierlich verfügbar war, werden in einer Liste erfasst und rückwärts sortiert.

2. Ebenso, wie bei Variante 1 werden die Verfügbarkeiten zwischen zwei Ausfällen gesammelt. Vom Maximum, welches das letzte Element der Sequenz ist, wird jedes Element subtrahiert.

Die zweite Variante wird für die Berechnung der RUL verwendet, da so auch Sequenzen verwendet werden können, die nicht bei null beginnen. Solche Sequenzen würden bei der ersten Variante dazu führen, dass die RUL bei einem Absturz nicht null beträgt. Würde man diese Variante verwenden, müsste der Anfang einiger Sensordaten daher verworfen werden. Die Monitoring-Anwendung beginnt die Datenerhebungen nämlich nicht notwendigerweise bei einem Systemstart.

Mit beiden Varianten ist es jedoch notwendig, die Sensordaten nach dem letzten Ausfall zu verwerfen. Der Grund dafür ist, dass der nächste Ausfallzeitpunkt, welche für die Bestimmung der RUL benötigt wird, für diese Zeitpunkte nicht vorliegt.

Zudem können Ausfälle nicht einfach dadurch identifiziert werden, dass zu dem Zeitpunkt durch den Sensor eine null erfasst wird. Schließlich wird dieser Sensor nur minütlich abgefragt und misst die Verfügbarkeit in Millisekunden. Stattdessen liegt immer dann ein Ausfall vor, wenn der erfasste Wert kleiner ist als der vorherige. Jedoch reicht es für die Berechnung der RUL nicht aus, das zweite beschriebene Verfahren auf die Sequenzen zwischen allen Ausfällen anzuwenden. Dadurch wären nämlich auch die Verfügbarkeiten mit dem Wert null während eines Ausfalls betroffen. Beim Anwenden des Verfahrens würden diese zu einem Plateau in der RUL führen. Um das zu verhindern, müssen immer das Ende eines Ausfalls und der Beginn des Nächsten identifiziert werden. Alle Werte dazwischen bilden den Zeitraum, in welchem die betrachtete Instanz kontinuierlich verfügbar war.

Diese Berechnung der RUL ist in Algorithmus 2 beschrieben. Die Funktion FIND_UPTIME_END identifiziert den Zeitpunkt des letzten Ausfalls *vor uptime_end*, während mit FIND_UPTIME_START der Zeitpunkt nach *downtime_start* ermittelt wird, bei welchem die Verfügbarkeit erstmalig wieder ansteigt. In dem eingegrenzten Bereich wird dann die RUL berechnet. Dieses Verfahren wird so lange mit Teilsequenzen wiederholt, bis der Anfang der Aufzeichnung erreicht ist (siehe Zeile 3).

3.3 Betrachtung der Sensordaten

Um ein grundlegendes Verständnis vom Datensatz zu gewinnen, werden diese anhand von Partial-Dependency-Plots (PDPs) näher betrachtet. PDPs zweigen die Abhängigkeit zwischen der Zielgröße (in diesem Fall die RUL) und einem einzelnen Feature auf. Dabei

Algorithmus 2 Berechnung der RUL aus der Verfügbarkeit

```
1: function CALCULATE_RUL(uptime)
2:     uptime_end ← LENGTH(uptime) - 1
3:     while uptime_end > 0 do
4:         max_uptime ← uptime[uptime_end]
5:         downtime_start ← 1+ FIND_UPTIME_END(uptime, uptime_end)
6:         uptime_start ← FIND_UPTIME_START(uptime, downtime_start)
7:         for i ← uptime_start, uptime_end do
8:             uptime[i] ← max_uptime − uptime[i]
9:         end for
10:        uptime_end ← downtime_start − 1
11:    end while
12:    return uptime
13: end function
```

wird vernachlässigt, dass die anderen Features ebenfalls eine Rolle spielen. Daher stellen PDPs nur eine Annäherung dar.

Abbildung A.1 enthält einen PDP für jeden Sensor. Auf der x-Achse ist jeweils der Wertebereich des Sensors abgebildet. Die y-Achse zeigt die RUL-Werte. Für jeden durch den Sensor erfassten Wert wird ein Datenpunkt auf dem Graphen mit der korrespondierenden RUL vermerkt.

Aus den Graphen lässt sich bereits abschätzen, wie relevant bestimmte Sensoren sein werden. Sensoren, die nur zwischen wenigen Werten schwanken und ähnliche RUL-Werte abdecken, sind eher irrelevant. Sensoren wie zum Beispiel „MemoryTenuredSediment“, bei denen eine Abhängigkeit zwischen den beiden Größen zu erkennen ist, spielen eine wichtige Rolle.

Bei so einer Betrachtung wird deutlich, dass einige Sensoren immer denselben Wert beibehalten und somit keinen Informationsgehalt haben. Die folgenden Sensoren zeigen dieses Verhalten.

- CodeCache.max
- IsAvailable
- JBossOperatingSystemAvailableProcessors
- MemoryPerm.max
- MemoryTenured.max
- MesgCache.HighMemMark
- MesgCache.MaxMemMark
- ThreadPoolQueueSize
- TotalPhysicalMemorySize
- jBossEdiDbConnectionPoolMax
- jBossEdiDbConnectionPoolMaxInUse
- jBossNativeDbConnectionPoolMax
- jBossNtxDbConnectionPoolMax
- jBossTxDbConnectionPoolMax

Wie einige Namen bereits vermuten lassen, liefern Sensoren in dieser Liste Werte zurück, welche von der Konfiguration des Systems oder der Hardware, auf der es betrieben wird, abhängig sind. Dies ist auch den Beschreibungen der Sensoren in der Tabelle A.1 zu entnehmen. Solche Sensoren können vernachlässigt werden.

Die Sensoren „IsAvailable“, „ThreadPoolQueueSize“ und „jBossEdiDbConnectionPoolMaxInUse“ gehören allerdings nicht in diese Kategorie. Der Sensor „IsAvailable“ liefert per Definition immer denselben Wert. Wenn kein Wert abgefragt werden kann, ist die Instanz nicht verfügbar. Da dieser Sensor niemals hilfreiche Informationen liefern wird, wird er ebenfalls nicht für Vorhersagen verwendet. Die anderen beiden Sensoren hingegen könnten potenziell hilfreich sein, weshalb sie bei der Anwendung der ML-Verfahren berücksichtigt werden sollten.

4 Umsetzung der Machine-Learning Verfahren

Die durch Monitoring-Anwendungen erfassten Sensordaten können nun für das Training von Machine-Learning Modellen verwendet werden, um Vorhersagen der RUL treffen zu können. Zunächst müssen Verfahren, die evaluiert werden sollen, ausgewählt werden. Jedes Unterkapitel wird sich dann einem der gewählten Verfahren widmen. Zu Beginn wird jeweils der Aufbau und die Funktionsweise des Verfahrens erklärt, um das notwendige Verständnis für dessen folgende Anwendung zu schaffen. Nachdem beschrieben wird, wie mit dem Verfahren ein Modell trainiert wurde, wird für jedes resultierende Modell eine Möglichkeit zur Interpretation der Ausgaben geschaffen. Am Ende des Kapitels wird also mit jedem ausgewählten ML-Verfahren ein Lösungskandidat präsentiert, welcher sowohl Vorhersagen treffen als auch erklären kann.

Ran u. a. [9] stellen in „A Survey of Predictive-Maintenance: Systems, Purposes and Approaches“ verschiedene Verfahren vor, die angewendet werden, um Predictive-Maintenance Aufgaben zu lösen. Diese verfügen alle über Vor- und Nachteile. Es wurden drei Verfahren ausgewählt, welche für diesen Anwendungsfall geeignet sein könnten.

Entscheidungsbaum Entscheidungsbäume verfügen verglichen mit anderen Verfahren über eine geringere Vorhersage-Genauigkeit. [9, S. 17] Dafür sind sie Vorhersagen von Entscheidungsbäumen leicht zu erklären, indem man den Pfad durch den Entscheidungsbaum nachverfolgt.

Feed-Forward Netz Entscheidungsbäume sind leicht zu interpretieren, werden vermutlich jedoch nur ungenaue Ergebnisse liefern. LSTMs hingegen eignen sich sehr gut für den Anwendungsfall, sind jedoch schwerer zu interpretieren. Feed-Forward Netze werden ausgewählt, da sie einen Kompromiss der anderen beiden Verfahren darstellen. Mit ihnen lässt sich häufig eine bessere Vorhersage-Genauigkeit erzielen. [9, S. 17] Dafür können Vorhersagen nicht so einfach erklärt werden.

LSTM Rekurrente neuronale Netze eignen sich gut für die Verarbeitung von Sequenzen. [9, S. 21] Das macht sie zu guten Kandidaten für diesen Anwendungsfall, da so die über einen längeren Zeitraum erhobenen Sensordaten als Eingabe für das Modell genutzt werden können. Das Modell kann somit Muster in diesen Daten erlernen, um geeignete Voraussagen zu treffen.

Long-Short-Term-Memorys (LSTMs) sind besondere rekurrente neuronale Netze, die Abhängigkeiten in den Eingabesequenzen auch über große Distanzen gut erkennen können. Daher werden Sie sind besonders häufig für die Verarbeitung von zeitlich abhängigen Daten verwendet. [9, S. 21]

4.1 Entscheidungsbäume

Entscheidungsbäume werden üblicherweise verwendet, um Klassifikationsprobleme zu lösen. Jedoch können sie ebenfalls für Regression verwendet werden [3, S. 305]. Ein Entscheidungsbaum besteht aus mehreren Knoten. Um für eine Eingabe eine Entscheidung zu treffen, wird der Baum beginnend bei dessen Wurzel durchlaufen. Mit jedem Knoten ist ein Prüfkriterium verbunden, das von einem Attribut der Eingabedaten abhängig ist. Beim Durchlaufen des Entscheidungsbaums mit einer Eingabe entscheidet die Auswertung des Prüfkriteriums beim aktuellen Knoten, welcher der Folgeknoten als nächstes betrachtet wird. [10, S. 698]

Mit jedem Blattknoten des Baums ist ein Ergebnis verbunden [10, S. 698]. Bei Klassifikationsproblemen ist dies die Klasse, welche der Eingabe zugeordnet wird. Bei einem Regressionsproblem ist es ein entsprechender Rückgabewert.

4.1.1 Konstruktion

Entscheidungsbäume können mit der Hilfe von Trainingsdaten konstruiert werden. Das Ziel solcher Konstruktionsalgorithmen ist es, einen Entscheidungsbaum zu erzeugen, der für die Trainingsdaten möglichst gute Ausgaben erzeugt und so klein wie möglich ist. [10, S. 699f.]

Die Größe der erzeugten Entscheidungsbäume gering zu halten, hat mehrere Vorteile. Zum einen sind große Entscheidungsbäume anfälliger für Overfitting [10, S. 705], zum

anderen müssen bei kleinen Bäumen weniger Tests durchgeführt werden, um eine Eingabe einzuordnen.

Quinlan entwickelte die Versionen ID3, C4.5 und C5.0 eines Algorithmus für die Konstruktion von Entscheidungsbäumen. Jedoch können mit diesen Algorithmen keine Entscheidungsbäume für Regression erstellt werden [3, S. 312]. Der Classification-and-Regression-Tree (CART) Algorithmus stellt eine Alternative zu diesen Algorithmen dar, die auch numerische Ausgaben ermöglicht [3, S. 307]. Da diese Art der Ausgabe für die vorherzusagenden RUL-Werte notwendig ist, wird der CART Algorithmus verwendet. Relevante Aspekte der Funktionsweise des CART-Algorithmus werden im Folgenden kurz beschrieben.

Ein Entscheidungsbaum wird erstellt, indem die Menge der Trainingsdaten in Teilmengen aufgeteilt werden. Für die Einordnung in diese Teilmengen werden die Attribute der Datensätze verwendet. Dieser Prozess wird rekursiv für alle gebildeten Teilmengen durchlaufen. In jedem Schritt wird das wichtigste Attribut für die Bildung der Teilmengen verwendet. [10, S. 700]

Die Wichtigkeit eines Attributs wird dadurch bestimmt, wie gut es die Trainingsdaten in möglichst homogene Teilmengen unterteilt. Bei der Regression sollen die Werte der Zielvariable in jeder Teilmenge möglichst nahe beieinander liegen. Die Summe der quadrierten Fehler (4.1) wird verwendet, um zu messen, wie gut ein Attribut diese Aufteilung realisiert. M bezeichnet hier die betrachtete Teilmenge, von der c der Durchschnitt der Zielvariable ist. Die Funktion f liefert die zu einem Datensatz der Trainingsdaten passende Ausgabe.

$$\sum_{x_i \in M} (f(x_i) - c)^2 \tag{4.1}$$

Dieses Fehlermaß wird für beide Teilmengen berechnet und addiert. Je wichtiger ein Attribut ist, desto geringer fällt diese Summe aus. [3, S. 307]

Bei Attributen mit einem kontinuierlichen Wertebereich gibt es mehrere Möglichkeiten, anhand eines Schwellwerts Teilmengen zu bilden. In so einem Fall müssen verschiedene Schwellwerte ausprobiert werden. [10, S. 707]

Die rekursive Konstruktion endet spätestens, wenn es entweder keine noch nicht verwendeten Attribute mehr gibt, um die Datensätze aufzuteilen, oder wenn es nicht mehr genug Datensätze an dem Knoten gibt, um diese auf Folgeknoten aufzuteilen. [10, S. 700f.]

4.1.2 Regularisierung

Wie bereits etabliert, neigen große Entscheidungsbäume zu Overfitting. Wenn ein Entscheidungsbaum mit Trainingsdaten erstellt wird, wächst dieser so lange, bis keine Datensätze mehr an den Knoten für das Aufteilen auf Folgeknoten vorhanden sind.

Es gibt zwei Möglichkeiten, die Größe eines Entscheidungsbaums einzuschränken. Einerseits kann ein Early-Stopping-Meachnismus etabliert werden, welcher den Konstruktionsalgorithmus für einen Teilbaum vorzeitig stoppt, wenn die Verbesserung unter einem festgelegten Schwellwert liegt. Andererseits kann der Entscheidungsbaum nach der Konstruktion vereinfacht werden. Dieses Vorgehen wird als Pruning bezeichnet und wird bevorzugt, da so scheinbar schlechte Aufteilungen auf Folgeknoten, die dann zu besseren Aufteilungen führen, nicht unterdrückt werden. [3, S. 308]

Das Cost-Complexity-Pruning Verfahren definiert für einen Baum T die Kostenfunktion $C_\alpha(T)$. Sie ist abhängig von dem Parameter $\alpha \geq 0$. $C(T_m)$ definiert für Regressionsbäume die Summe der quadrierten Fehler, wie sie in Gleichung 4.1 definiert ist, für den Blattknoten m im Baum T. $|T|$ steht für die Anzahl an Blattknoten des Baums T.

$$C_\alpha(T) = \sum_{m=1}^{|T|} C(T_m) + \alpha|T| \tag{4.2}$$

Die Kosten eines Entscheidungsbaums setzen sich also nach Gleichung 4.2 aus der Summe der Fehler aller Blattknoten und der Anzahl der Blattknoten zusammen. Um den Wert der Kostenfunktion zu minimieren, muss ein Kompromiss zwischen der Anzahl an Blattknoten sowie der Genauigkeit des Entscheidungsbaums gefunden werden. Der Parameter α legt das Verhältnis der beiden Größen fest. [3, S. 308]

Um mit einem festgelegten α einen Entscheidungsbaum zu vereinfachen, werden so lange innere Knoten zu Blattknoten umgewandelt, bis nur noch der Wurzelknoten übrig bleibt. In jedem Schritt wird dabei der innere Knoten gewählt, der das Fehlermaß $\sum_{m=1}^{|T|} C(T_m)$ am wenigsten erhöht. Von allen betrachteten Teilbäumen wird der Baum mit den geringsten Kosten $C_\alpha(T)$ als vereinfachter Entscheidungsbaum ausgewählt. [3, S. 308]

Es muss jedoch ein geeigneter Wert für den Parameter α gefunden werden. Hierfür kann man Cross-Validation verwenden. Mit verschiedenen Werten für α werden Entscheidungsbäume konstruiert. Für jeden Entscheidungsbaum werden dann die Genauigkeit mit den

Trainings- und Testdaten erfasst. In Abbildung 4.1 ist zu sehen, wie die Vorhersagegenauigkeit von der Wahl des Pruning-Parameters abhängig ist. Der Parameter sollte so gewählt werden, dass er den Fehler auf den Testdaten minimiert. Hier ist dies bei $\alpha \approx 0.1$ der Fall. Wenn $\alpha = 0$ gilt, entspricht der resultierende Entscheidungsbaum dem ungetrimmten Baum. In Abbildung 4.1 ist also deutlich zu erkennen, dass sich die Qualität des Modells durch diese Regularisierungsmethode verbessert hat.

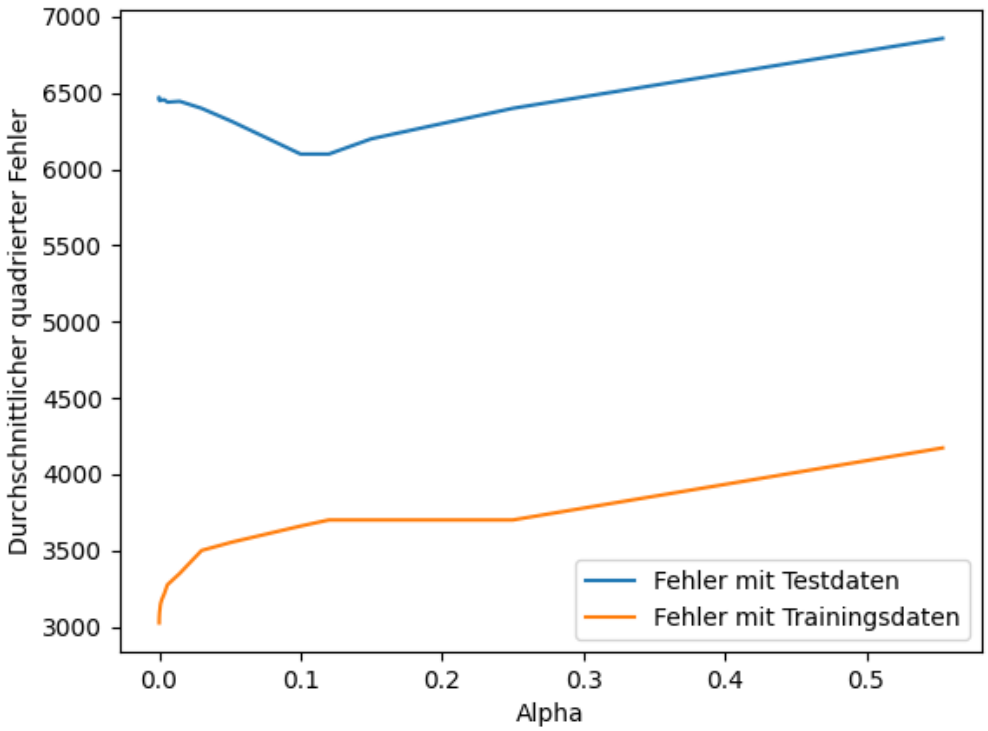


Abbildung 4.1: Vergleich verschiedener Werte des Pruning-Parameters

4.1.3 Interpretation

Eine Stärke von Entscheidungsbäumen ist, dass Ausgaben aufgrund der simplen Struktur der Bäume leicht zu erklären sind [10, S. 707]. Dafür muss lediglich der Pfad durch den Entscheidungsbaum nachvollzogen werden. Die Bedingungen, welche erfüllt sein müssen, damit eine Eingabe diesem Pfad folgt, müssen mit einem „und“ verknüpft werden. Immer,

wenn der dadurch entstehende Ausdruck durch eine Eingabe erfüllt wird, wird dieselbe Ausgabe produziert. [10, S. 698f.]

Mit dem beschriebenen Vorgehen lassen sich einzelne Vorhersagen, die mit den Sensordaten eines Zeitpunktes erzeugt wurden, genau erklären. Bei großen Entscheidungsbäumen werden die produzierten Ausdrücke jedoch sehr lang und somit unübersichtlich. Es ist außerdem unklar wie wichtig die einzelnen Teilbedingungen für die Einordnung der Vorhersage sind. So ist es zum Beispiel in der folgenden Erklärung der Fall. Sie wurde mit einem ungetrimmten Entscheidungsbaum erzeugt und erklärt die Vorhersage einer RUL von 200.

> GarbageCollectionNewTimeDelta: $1316.0 \leq 70784.5$
> und RunningSessionInfoNr: $143.0 > 0.8956395089626312$
> und JMSQueueDemonProcess.depth: $0.0 \leq 3842244608.0$
> und jBossTxDbConnectionPoolMaxInUse: $36.0 \leq 124.5$
> und JMSQueueDemonProcess.depth: $0.0 \leq 3713531904.0$
> und MesgCache.SoftenedSize: $0.0 \leq 13.5$
> und JBossOperatingCommittedVirtualMemorySize: $22704328704.0 > 22672896000.0$
> und ProcessCpuTimeAvg: $0.0248061134625006 \leq 78.5$
> und MemoryEdenSpaceUsage.max: $3752853504.0 > 33.951229095458984$
> und MesgCache.TotalCacheSize: $18.0 \leq 303.0$
> und MemoryPerm.percental: $36.83133274316788 > 33.91458511352539$
> und JBossSystemServerInfoActiveThreadCount: $871.0 > 557.5$
> und SyntheticMonitor-DBCall: $1.725105 \leq 4.521528005599976$
> und jBossTxDbConnectionPool.percental: $0.0 > -2.0$

Stattdessen soll daher die Wichtigkeit jedes Sensors direkt berechnet werden. Dafür wird bestimmt, wie sehr jeder Sensorwert die Vorhersage des Modells verändert. Das ist möglich, indem die durchschnittliche Vorhersage eines Knotens, der diesen Sensor verwendet, und die des aufgrund des Sensorwertes ausgewählten Folgeknotens bestimmt werden. Die Differenz der beiden Vorhersagen drückt die Relevanz des betrachteten Sensors aus. Diese durchschnittliche Vorhersage wird genauso berechnet, wie es bereits in den Blattknoten gemacht wird. Der Wert der Zielvariable von jedem Datensatz, der einen Knoten durchläuft, wird erfasst, um im Anschluss deren Durchschnittswert zu bestimmen. [7, S. 104f.]

Wenn also M_1 die Menge der Datensätze ist, die für die Definition eines Knotens verwendet wurden und $M_2 \subset M_1$ die Teilmenge eines Folgeknotens ist, dann lässt sich mit

Gleichung 4.3 berechnen, wie sehr das Attribut des Folgeknotens das Ergebnis beeinflusst. Die Funktion f liefert ebenso wie in Gleichung 4.1 den Wert der Zielvariable.

$$\frac{1}{|M_1|} \sum_{x_i \in M_1} f(x_i) - \frac{1}{|M_2|} \sum_{x_i \in M_2} f(x_i) \tag{4.3}$$

Dieser Wert wird für jedes Attribut, welches auf dem Pfad durch den Entscheidungsbaum verwendet wird, berechnet. Alle Sensoren, die nicht verwendet wurden, haben eine Wichtigkeit von null. Stellt man die Wichtigkeiten der Sensoren in einem Balkendiagramm dar, ist für den Benutzer schnell sichtbar, wie relevant einzelne Sensoren sind und ob diese sich positiv oder negativ auf die Zielvariable auswirken.

Mit derselben Eingabe, welche auch für die textuelle Erklärung verwendet wurde, wird ein solches Balkendiagramm erzeugt (siehe Abbildung 4.2).

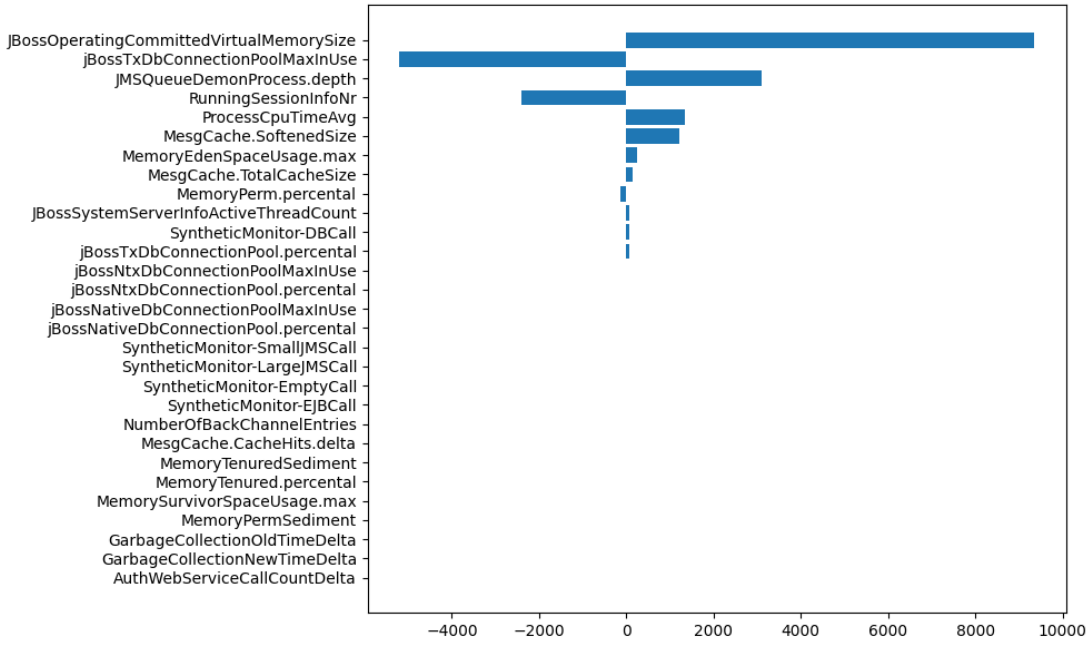


Abbildung 4.2: Beispielhafte lokale Erklärung für einen Entscheidungsbaum

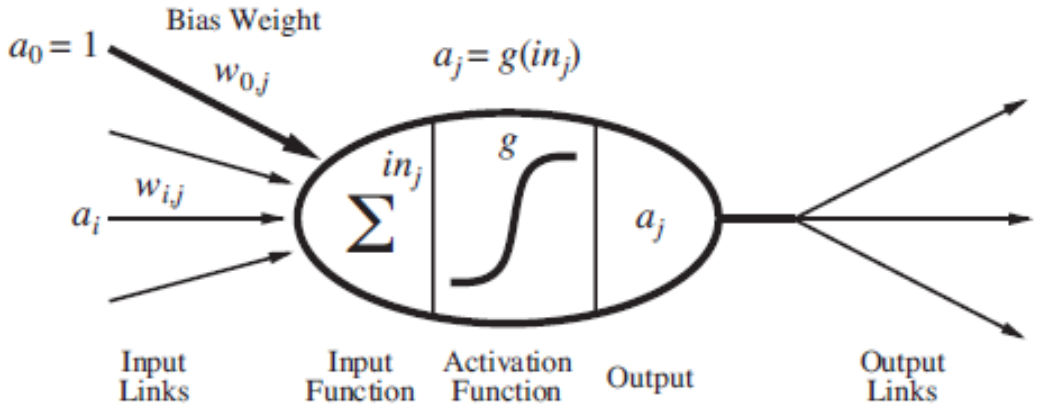


Abbildung 4.3: Aufbau eines künstlichen Neurons (Quelle: [10, S. 728])

4.2 Feed-Forward Netze

Künstliche neuronale Netze sind inspiriert von Erkenntnissen der Neurowissenschaften über den Aufbau des Gehirns. Sie bestehen aus künstlichen Neuronen, die zusammengeschaltet sind. Ein Neuron, wie es in Abbildung 4.3 zu sehen ist, kann mehrere ein- und ausgehende Verbindungen haben. Mit jeder Verbindung von einem Neuron i zu dem Neuron j ist ein Gewicht w_{ij} assoziiert. Die Aktivierung a_i gibt Signalstärke des Neurons i an. $a_0 = 1$ ist eine besondere Aktivierung, da sie konstant ist und von keinem anderen Neuron kommt. Über das korrespondierende Gewicht wird ein Bias eingeführt.

Die Aktivierung eines Neurons wird, wie in Gleichung 4.4 zu sehen ist, berechnet, indem die Funktion g auf die Summe der gewichteten Aktivierungen angewendet wird. Da diese Funktion die Aktivierung des Neurons angibt, wird sie als Aktivierungsfunktion bezeichnet. n steht in der Gleichung für die Anzahl der eingehenden Verbindungen. [10, S. 727f.]

$$a_j = g(\sum_{i=0}^{n} w_{ij} * a_i) \tag{4.4}$$

Bei einem Feed-Forward Netzwerk sind die Neuronen so verbunden, dass sich keine Zyklen bilden. Häufig werden die Netze in Schichten organisiert. Die Neuronen einer Schicht werden dabei mit den Neuronen der Folgeschicht verbunden. Die Eingabeschicht verfügt dabei über so viele Neuronen, wie es Dimensionen in den Eingabedaten gibt. [10, S. 729]

Die Anzahl der Neuronen in der Ausgabeschicht hängt von der Problemstellung ab. Bei der Regression gibt es typischerweise ein Neuron, an welchem der Ausgabewert gelesen werden kann. Bei Klassifikationsproblemen gibt es pro Klasse ein Neuron, an welchem die Ausgabe die Wahrscheinlichkeit der Zuordnung zu dieser Klasse ist. [3, S. 392] Zwischen den Ein- und Ausgabeschichten kann es beliebig viele weitere Schichten geben. Diese werden Hidden-Layer genannt [10, S. 729].

4.2.1 Vorverarbeitung

Die Konvergenz des Lernverfahrens wird beschleunigt, wenn die Netzeingaben standardisiert sind. Das bedeutet, dass die Eingabewerte für jedes Feature um null zentriert sind (4.5) und dieselbe Varianz σ^2 aufweisen (4.6). [4, S. 8]

$$\mu = \mu_1, \mu_1, \ldots, \mu_i = \frac{1}{|M|} \sum_{m \in M} z_i^m = 0 \tag{4.5}$$

$$\sigma^2 = \sigma_1^2, \sigma_2^2, \ldots, \sigma_i^2 = \frac{1}{|M|} \sum_{m \in M} (z_i^m)^2 \tag{4.6}$$

Die beiden Gleichungen müssen daher für jedes Feature i erfüllt sein. M bezeichnet hier die Menge der Datensätze. z_i^m ist der Wert des Features i im Datensatz m.

Häufig wird so standardisiert, dass die Varianz $\sigma^2 = 1$ gilt. Dafür muss für jedes Feature i das arithmetische Mittel μ_i und die Standardabweichung σ_i, welche die Quadratwurzel der Varianz ist, bestimmt werden. Durch das Standardisieren wird dann nach Gleichung 4.7 für jeden Wert z_i^m der standardisierte Wert z'^m_i berechnet.

$$z'^m_i = \frac{z_i^m - \mu_i}{\sigma_i} \tag{4.7}$$

Abbildung 4.4 zeigt den Effekt des Standardisierens am Beispiel eines Sensors. Die Verteilung der ursprünglichen Sensorwerte bleibt erhalten. Der einzige Unterschied zeigt sich in der Skalierung der x-Achse. Zum einen liegen die Werte nun um null. Zum anderen fallen sie insgesamt niedriger aus, da $\sigma^2 = 1$ gilt. Bei der Anwendung von Cross-Validation darf kein Wissen über die Validierungsdaten in das Training der Modelle mit einfließen. Aus diesem Grund werden die Standardabweichungen und Varianzen, welche für das Skalieren aller Daten notwendig sind, nur auf den Trainingsdaten erhoben.

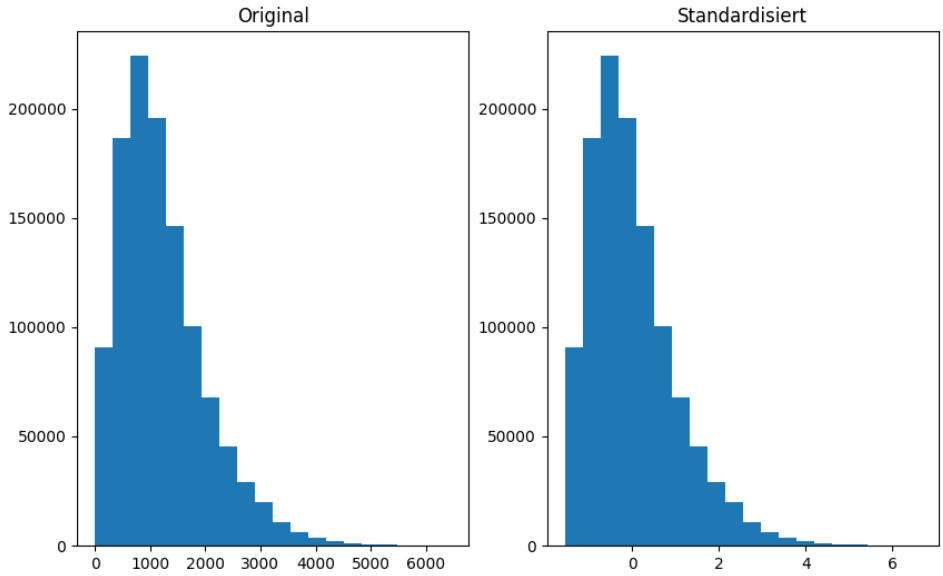


Abbildung 4.4: Histogramme des Sensors „AuthWebServiceCallCountDelta“ vor und nach dem Standardisieren mit $\mu = 1218{,}28$ und $\sigma = 785{,}86$

4.2.2 Wahl der Hyperparameter

Der Aufbau eines neuronalen Netzes beeinflusst den Lernerfolg maßgeblich. Es sollte daher eine geeignete Anzahl an Hidden-Layer und enthaltenen Neuronen gewählt werden. Ebenso sollte eine geeignete Aktivierungsfunktion gewählt werden.

Üblicherweise wird Cross-Validation verwendet, um eine geeignete Wahl der Hyperparameter zu ermitteln. Es werden also Modelle mit verschiedenen Hyperparametern trainiert. Die Hyperparameter, deren Modelle die Validierungsdaten am besten verarbeiten, werden gewählt. [10, S. 737]

Die bekanntesten Aktivierungsfunktionen sind Recified-Linear-Unit (ReLU), Hyperbolic-Tangent (TanH) und Sigmoid. [2, S. 6] Deren Verlauf ist Abbildung 4.5 zu entnehmen. Während TanH und Sigmoid auf den Bereich zwischen 0 beziehungsweise -1 und 1 begrenzt sind, deckt ReLU jeden Wert ≥ 0 ab. ReLU wird in jedem Fall als Aktivierungsfunktion des einzigen Neurons in der Ausgabeschicht verwendet, da negative Werte nicht zulässig sind und das Modell somit den gesamten Wertebereich der RUL abdecken kann.

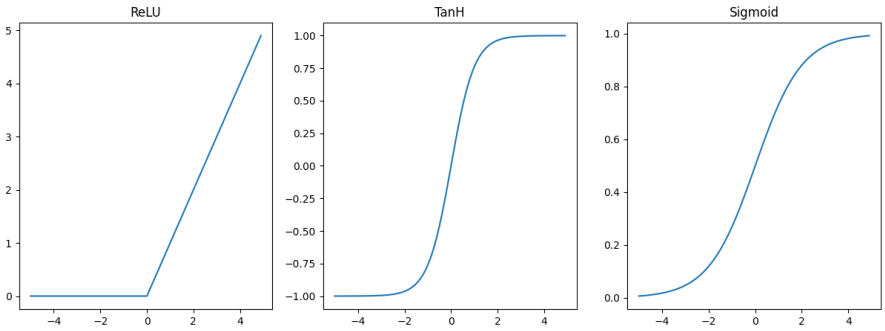


Abbildung 4.5: Vergleich häufig verwendeter Aktivierungsfunktionen

In den verbleibenden Schichten des Modells können jedoch auch andere Aktivierungsfunktionen verwendet werden.

Es werden nun mit jeder Aktivierungsfunktion 20 Feed-Forward Netze mit unterschiedlichen Ausprägungen an Hidden-Layern antrainiert. Von jedem Modell wird im Anschluss das Fehlermaß auf Testdaten bestimmt. Abbildung 4.6 enthält ein Boxplot-Diagramm pro Aktivierungsfunktion. Diese stellen die Streuung der Fehlermaße der verschiedenen Modelle dar. Die Fehlermaße der ReLU Aktivierungsfunktion liegen deutlich unter denen der anderen Aktivierungsfunktionen, weshalb sie nicht nur für die Ausgabeschicht sondern auch für die Hidden-Layer verwendet werden sollte.

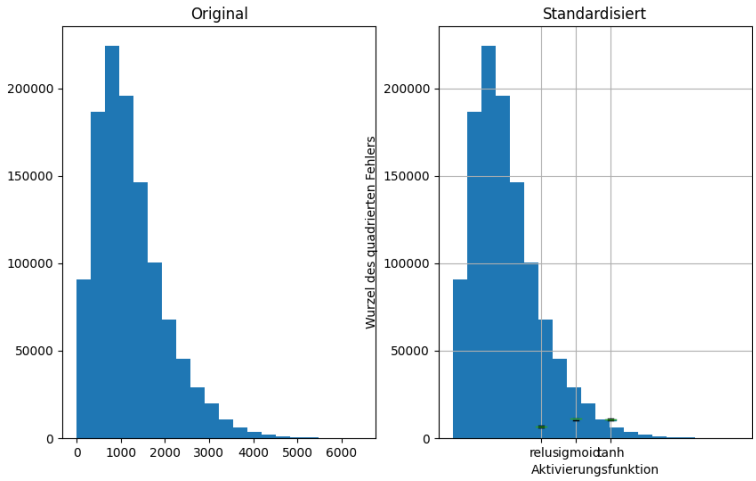


Abbildung 4.6: Streuung des Fehlermaßes bei der Verwendung unterschiedlicher Aktivierungsfunktionen

Um eine geeignete Anzahl an Hidden-Layer sowie Neuronen in diesen zu ermittelt, werden weitere Netze in verschiedenen Konfigurationen antrainiert. Dabei wird nun immer die ReLU Aktivierungsfunktion verwendet. Getestet werden Netze mit bis zu vier Hidden-Layer mit maximal 26 Neuronen. Diese Neuronenanzahl wurde gewählt, da sie der Anzahl an verwendeten Sensoren entspricht und weitere Neuronen somit irrelevant wären.

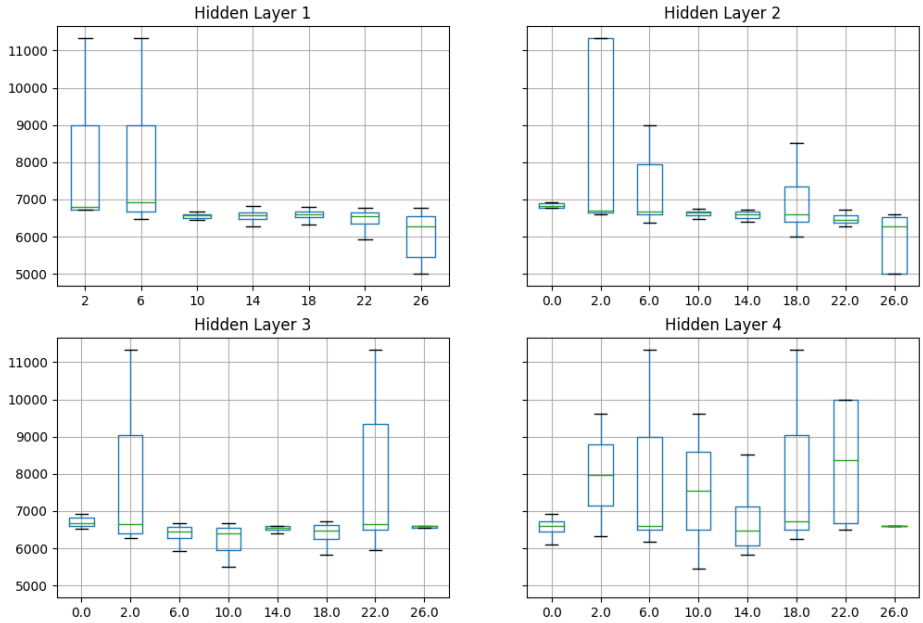


Abbildung 4.7: Streuung des Fehlermaßes bei unterschiedlichen Netzstrukturen

Die Abbildung 4.7 zeigt ebenfalls Boxplot-Diagramme. In diesem Fall gibt es eins für jede getestete Neuronen-Anzahl pro Hidden-Layer. Das Fehlermaß, welches für ein Modell mit weniger als vier Schichten ermittelt wurde, wird bei den fehlenden Schichten jeweils unter null Neuronen erfasst. Es ist zu erkennen, dass die ermittelten Fehler bei vier Hidden-Layern stark schwanken, was als Zeichen für potenzielles Overfitting interpretiert wird. Daher sollten nur drei Hidden-Layer verwendet werden. Die Neuronenanzahlen werden so gewählt, dass die entsprechenden Boxplot-Diagramme möglichst geringe Fehlermaße abdecken. Folglich sollte ein Netz mit 26, 26 und 10 Neuronen in den Hidden-Layer verwendet werden.

4.2.3 Interpretation

Um auch für Feed-Forward Netzwerke Interpretierbarkeit zu gewährleisten, wird Layer-wise-Relevance-Propagation (LRP) eingesetzt. LRP ist ein modell-spezifisches Verfahren, bei welchem Schicht für Schicht pro Neuron ein Wichtigkeitsmaß berechnet wird.

Berechnet man dieses beginnend von der Ausgabeschicht rückwärts bis zur Eingabeschicht, lässt sich somit die Relevanz von jedem Feature für eine Eingabe ermitteln. Dafür muss die Eingabe bereits vom Netzwerk verarbeitet worden sein, da die Aktivierungen aller Neuronen im Algorithmus benötigt werden. [8, S. 197]

In jedem Iterationsschritt wird Gleichung 4.8 verwendet, um die Relevanz R_j eines Neurons j aus der Relevanz R_k des Neurons k in der Folgeschicht zu berechnen. Dabei sei a_j die Aktivierung des Neurons j und w_{jk} das Gewicht der Verbindung zwischen den Neuronen j und k. Bei Beginn des Algorithmus in der Ausgabeschicht gilt $R_k = a_k$.

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_j a_j w_{jk}} R_k \tag{4.8}$$

Die Relevanz eines Neurons ist also abhängig von dessen Aktivierung und der Stärke der vorhandenen Verbindungen. Durch den Nenner $\sum_j a_j w_{jk}$ wird die Relevanz so skaliert, dass die Summe aller Werte pro Schicht der der vorherigen Schichten entspricht. [8, S. 198]

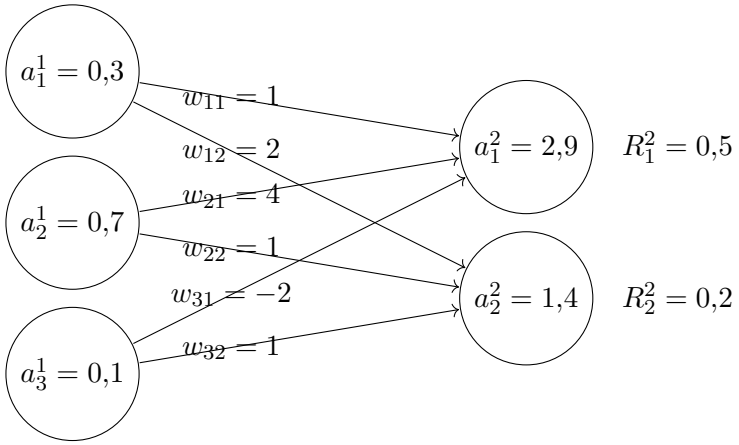


Abbildung 4.8: Beispiel eines Schrittes in der Durchführung des LRP-Algorithmus

In Abbildung 4.8 ist eine beispielhafte Situation gegeben, wie sie bei der Durchführung des Algorithmus eintreten könnte. Im Folgenden wird veranschaulicht, wie die Wichtig-

keitsmaße für die drei Neuronen mit den Angaben der Grafik berechnet werden.

$$
\begin{aligned}
R_1^1 &= \frac{a_1^1 w_{11}}{a_1^1 w_{11} + a_2^1 w_{21} + a_3^1 w_{31}} * R_1^2 + \frac{a_1^1 w_{12}}{a_1^1 w_{12} + a_2^1 w_{22} + a_3^1 w_{32}} * R_2^2 \\
&= \frac{0,3 * 1}{2,9} * 0,5 + \frac{0,3 * 2}{1.4} * 0,2 \approx 0{,}14 \\
R_2^1 &= \frac{a_2^1 w_{21}}{a_1^1 w_{11} + a_2^1 w_{21} + a_3^1 w_{31}} * R_1^2 + \frac{a_2^1 w_{22}}{a_1^1 w_{22} + a_2^1 w_{22} + a_3^1 w_{32}} * R_2^2 \\
&= \frac{0,7 * 4}{2,9} * 0,5 + \frac{0,7 * 1}{1.4} * 0,2 \approx 0{,}58 \\
R_3^1 &= \frac{a_3^1 w_{31}}{a_1^1 w_{11} + a_2^1 w_{21} + a_3^1 w_{31}} * R_1^2 + \frac{a_3^1 w_{32}}{a_1^1 w_{12} + a_2^1 w_{22} + a_3^1 w_{32}} * R_2^2 \\
&= \frac{0,1 * -2}{2,9} * 0,5 + \frac{0,1 * 1}{1.4} * 0,2 \approx -0{,}02
\end{aligned}
$$

4.3 LSTM

LSTMs stellen eine Unterart der Recurrent-Neural-Networks (RNNs) dar. Solche Netze verfügen über zyklische Verbindungen. Diese Zyklen werden umgesetzt, indem jedes Neuron der betroffenen Schicht immer zwei Eingaben erhält. Die eine sind wie auch bei einem Feed-Forward Netzwerk die Aktivierungen der Neuronen der vorherigen Schicht. Die andere Eingabe stellt den internen Zustand der rekurrenten Schicht dar und ist von den vorherigen Aktivierungen ihrer Neuronen abhängig. [2, S. 11]

$$h_t = g(W x_t + U h_{t-1} + b) \tag{4.9}$$

$$y_t = V h_t \tag{4.10}$$

g : Aktivierungsfunktion
W : Matrix der Gewichte für eingehende Verbindungen
U : Matrix der Gewichte für rekurrente Verbindungen
V : Matrix der Gewichte für ausgehende Verbindungen
b : Bias-Vektor
h_t : Zustandsvektor zum Zeitpunkt t
x_t : Eingabevektor zum Zeitpunkt t
y_t : Ausgabevektor zum Zeitpunkt t

Gleichung 4.9 zeigt, wie der interne Zustand eines RNN für eine gesamte Schicht berechnet wird. Im Gegensatz zu der Gleichung 4.4, welche die Berechnung einer Aktivierung in Feed-Forward Netzen zeigt, wird hier nicht die Aktivierung eines einzelnen Neurons berechnet. Stattdessen werden die Aktivierungen der gesamten Schicht gleichzeitig ermittelt, da diese Notation kompakter ist.

Das Summieren der durch die eingehenden Verbindungen gewichteten Aktivierungen der vorherigen Schicht, erfolgt durch eine Matrizenmultiplikation für alle Neuronen der betrachteten Schicht gleichzeitig. Die Gleichung 4.11 soll verdeutlichen, wieso diese beiden Darstellungsformen äquivalent sind. Die Gewichtsmatrix enthält für jede Verbindung zwischen den Neuronen der vorherigen und der aktuellen Schicht ein Gewicht. Eine Zeile der Matrix enthält also alle eingehenden Verbindungen für ein Neuron. Durch Multiplikation der Matrix mit dem Vektor der Aktivierungen der vorherigen Schicht werden die Gewichte der eingehenden Verbindungen jedes Zielneurons mit den entsprechenden Aktivierungen multipliziert und aufsummiert. Um die Aktivierungen der aktuellen Schicht zu berechnen, muss dann nur noch die Aktivierungsfunktion g auf jedes Element des resultierenden Vektors angewendet werden.

$$\text{Zielneuronen}\left\{\overbrace{\begin{pmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ w_{10} & w_{11} & \dots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & \dots & w_{mn} \end{pmatrix}}^{\text{Quellneuronen}} * \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} w_{00}a_0 + w_{01}a_1 + \cdots + w_{0n}a_n \\ w_{10}a_0 + w_{11}a_1 + \cdots + w_{1n}a_n \\ \vdots \\ w_{m0}a_0 + w_{m1}a_1 + \cdots + w_{mn}a_n \end{pmatrix} \tag{4.11}$$

Die Berechnung des internen Zustands eines rekurrenten Netzes entspricht also der Berechnung der Aktivierungen in einem Feed-Forward Netz mit der Ausnahme, dass weitere Verbindungen zu dem vorherigen internen Zustand hinzukommen. Der berechnete interne Zustand wird mit einer weiteren Gewichtsmatrix für ausgehende Verbindungen multipliziert, bevor er von der folgenden Schicht verwendet wird. Dies zeigt die Gleichung 4.10.

Traditionelle RNNs haben jedoch den Nachteil, dass sie nicht gut mit Abhängigkeiten zwischen weit auseinander liegenden Eingaben umgehen können. Der Grund dafür ist, dass der interne Zustand besonders stark von den zuletzt verarbeiteten Eingaben ist. LSTM-Netzwerke verwenden Gatter, um zu bestimmen welche Informationen in den internen Zustand aufgenommen oder aus diesem entfernt werden. [2, S. 13]

Es gibt verschiedene Varianten der LSTM-Zelle. Sie unterscheiden sich häufig in den verwendeten Gattern voneinander. Abbildung 4.9 zeigt die am häufigsten verwendete

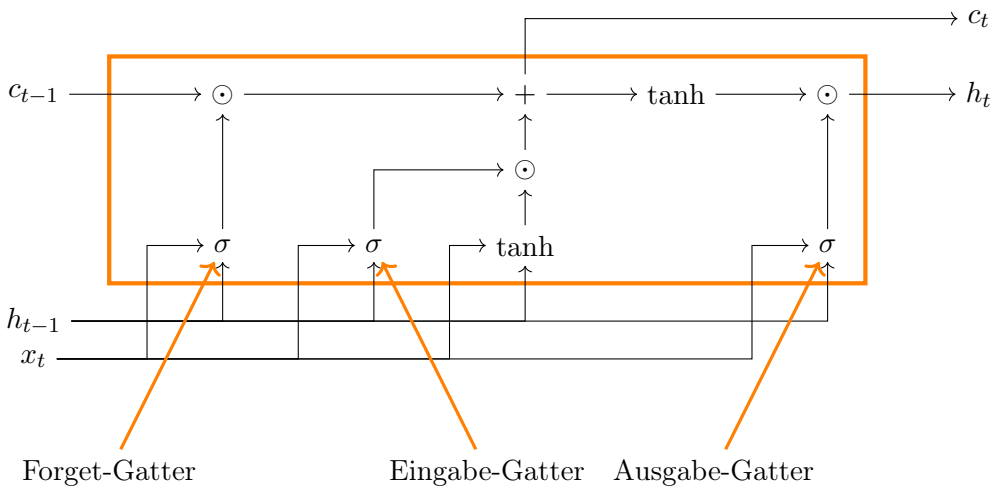


Abbildung 4.9: Aufbau einer LSTM-Zelle

Variante. Sie ist auch in dem Deep-Learning-Framework „Keras“ implementiert, welche für die folgende Konstruktion des LSTM-Netzwerks verwendet wird. Daher wird nur diese Variante betrachtet.

Jede Zelle verfügt über zwei interne Zustände. Zum einen gibt es den Cell-State c_t und zum anderen den Hidden-State h_t. In einem Berechnungsschritt werden die internen Zustände c_{t-1} und h_{t-1} sowie die aktuelle Aktivierung x_t verwendet. Für jedes Gatter sowie für die Verarbeitung der Eingabe x_t gibt es wie bei einem RNN (siehe Gleichung 4.9) gewichtete Verbindungen zu der vorherigen Schicht, rekurrente Verbindungen und eigene Bias-Werte. Alle Gatter verwenden die Sigmoid-Aktivierungsfunktion, da deren Wertebereich zwischen null und eins liegt. Durch elementweise Multiplikation ($\odot$) eines Gatter-Vektors mit einem zweiten Vektor wird bestimmt, wie viel der einzelnen Werte beibehalten wird. Für die Verarbeitung der Ein- und Ausgaben der LSTM Schicht wird üblicherweise die tanh Aktivierungsfunktion verwendet.

Mit Gleichung 4.15 wird der neue Cell-State berechnet. Dieser setzt sich aus zwei Komponenten zusammen. $i_t \odot z_t$ ist der Anteil der Eingabe, welchen das Eingabe-Gatter als relevant erachtet. Die zweite Komponente ist $f_t \odot c_{t-1}$. Mit dem Forget-Gatter werden so irrelevante Informationen des vorherigen Cell-State verworfen. Die Ausgabe der Schicht in Form des Hidden-State wird mit Gleichung 4.17 ermittelt. Nach Anwendung

der Aktivierungsfunktion wird mithilfe des Ausgabe-Gatters erneut gefiltert.

$$z_t = \tanh(W_z x_t + U_z y_{t-1} + b_z) \quad \text{Eingabe} \tag{4.12}$$

$$i_t = \sigma(W_i x_t + U_i y_{t-1} + b_i) \quad \text{Eingabe-Gatter} \tag{4.13}$$

$$f_t = \sigma(W_f x_t + U_f y_{t-1} + b_f) \quad \text{Forget-Gatter} \tag{4.14}$$

$$c_t = i_t \odot z_t + f_t \odot c_{t-1} \quad \text{Cell-State} \tag{4.15}$$

$$o_t = \sigma(W_o x_t + U_o y_{t-1} + b_o) \quad \text{Ausgabe-Gatter} \tag{4.16}$$

$$y_t = o_t \odot \tanh(c_t) \quad \text{Hidden-State} \tag{4.17}$$

4.3.1 Verarbeitung der Sensordaten als Sequenzen

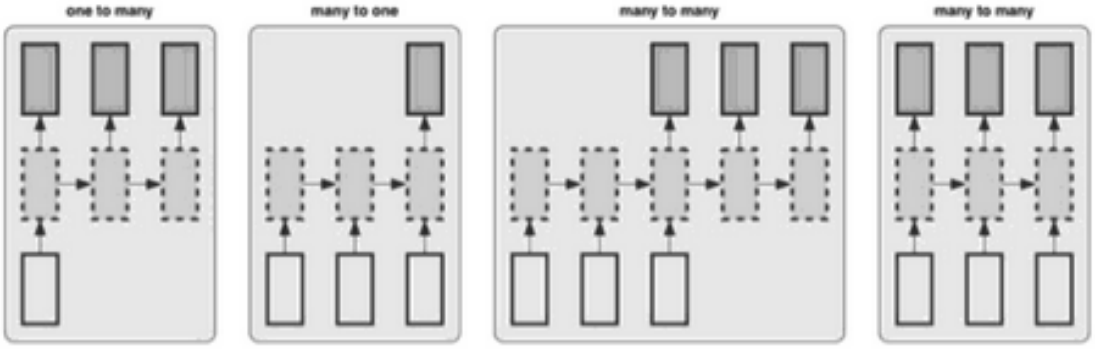


Abbildung 4.10: Betriebsmodi von rekurrenten neuronalen Netzen (Quelle: [11, S. 535])

Da Ausgaben eines RNN von vorherigen Eingaben abhängig sind, eignen sie sich, um Zusammenhänge in Sequenzen von Daten zu finden oder um Sequenzen zu erzeugen. Abbildung 4.10 zeigt verschieden Möglichkeiten für den Umgang mit Ein- und Ausgaben.

One-To-Many Mit einer einzigen Eingabe wird eine Ausgabesequenz erzeugt. Dieser Modus ist zum Beispiel für die Generierung von Abbildungstiteln sinnvoll. Nach der ersten Eingabe wird der interne Zustand folglich ohne Einfluss weiterer Eingabevektoren berechnet.

Many-To-One Für eine gesamte Eingabesequenz wird genau eine Ausgabe generiert. Die Klassifikation von Texten wäre ein mögliches Anwendungsgebiet für diesen Modus. Hierfür wird y_t nach der Berechnung des letzten internen Zustandes berechnet. Es werden also nur dann Signale an die Folgeschicht übermittelt.

Many-To-Many Sowohl die Netzwerkeingaben als auch die Ausgaben sollen Sequenzen sein. Dafür kann direkt zu jedem Element der Sequenz eine Ausgabe erzeugt werden. Damit entspricht die Länge der erzeugten Sequenz zwingend der der Eingabesequenz.

Alternativ können, die anderen beiden Modi kombiniert werden, um erst mit einem Encoder einen internen Zustand zu erstellen, welcher die gesamte Sequenz beschreibt. Anschließend wird mit diesem als einzige Eingabe für den Decoder die Ausgabesequenz erzeugt.

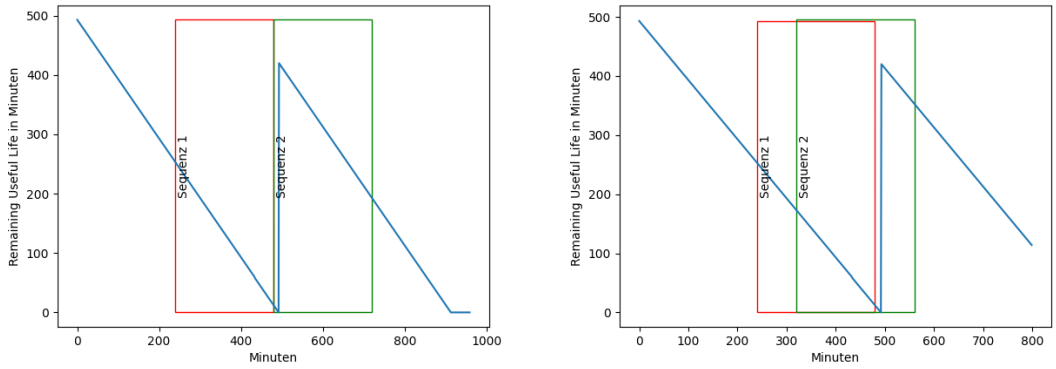
[11, S. 535]

Wenn ein RNN verwendet werden würde, um Vorhersagen zum Zustand eines Produktionsleitsystems in Echtzeit zu treffen, würden neue Sensordaten, sobald sie verfügbar sind, als Modelleingabe verwendet werden. Mit jeder neuen Eingabe muss also eine entsprechende Vorhersage erzeugt werden. Daher muss ein Many-To-Many Betriebsmodus verwendet werden. Bei der Verarbeitung mit einem Encoder und einem Decoder muss die vollständige Sequenz vorliegen, bevor Ausgaben erzeugt werden. Daher sollen die Ausgaben stattdessen direkt erzeugt werden.

Die Sensordaten der durch Monitoring-Anwendungen überwachten Produktionsleitsysteme stellen pro Instanz eine einzige Sequenz dar. Für jeden Zeitpunkt in der Sequenz gibt es durch die Sensoren als Features einen entsprechenden Wert. Die RUL-Werte stellen die Ausgabesequenz dar. Allerdings fallen diese Sequenzen sehr lang aus, da minütlich Sensordaten erfasst werden und einige Monitoring-Anwendungen bereits über mehrere Jahre hinweg Daten erheben. Damit sind diese Sequenzen zu lang, um ein rekurrentes Netz damit zu trainieren.

Um dieses Problem zu lösen, wird die Sequenz in kürzere Teilsequenzen aufgeteilt, welche für das Training verwendet werden können. Dabei kann es jedoch passieren, dass eine Teilsequenz mit einem Systemausfall beginnt. Die Anzeichen für diesen Ausfall befinden sich also in den vorherigen Zeitschritten. Da diese jedoch zu der davor liegenden Teilsequenz gehören, würde ein mit diesen Teilsequenzen trainiertes Modell für die Zeitpunkte um den Ausfall herum keine guten Vorhersagen erzeugen. Diese Situation wird durch die beiden aufeinander folgenden Teilsequenzen in Abbildung 4.11a dargestellt.

Da die Vorhersagen in der Nähe eines Ausfalls besonders interessant sind, sollten derartige Situationen behandelt werden. Aus diesem Grund werden die Teilsequenzen so erzeugt, dass sie sich überschneiden. Die Sequenzlänge s wurde auf 4,00 h festgelegt, da in diesem



(a) $o = 240$ (b) $o = 80$

Abbildung 4.11: Aufteilen der Sensordatensequenz mit Versatz o

Intervall auch für die nur stündlich abgefragten Sensoren einige Werte vorliegen. Bei dem in Abbildung 4.11a beschriebenen Problemfall entspricht der Versatz o zwischen den Anfängen der Teilsequenzen der Sequenzlänge. Um das Trainieren mit nahezu identischen Sensordaten zu vermeiden, sollte der Versatz nicht zu gering gewählt werden. Aus diesem Grund wurde der für das Training tatsächlich verwendete Versatz auf $o = 80$ festgelegt.

4.3.2 Interpretation

Das LRP-Verfahren kann mit einigen Anpassungen auch für die Interpretation von Vorhersagen eines LSTM verwendet werden. Da ein LSTM im betrachteten Anwendungsfall Sensordaten mehrerer Zeitpunkte verwendet, ist es damit nicht nur möglich die Relevanz auf die Sensoren aufzuteilen. Vielmehr kann die Relevanz auch zeitlich aufgeteilt werden.

Arras u. a. unterscheiden zwischen drei Berechnungsschritten bei einem Forward-Pass in einem LSTM-Netzwerk. Für jeden Schritt muss definiert werden, wie die Relevanz bei der Anwendung von LRP umverteilt wird.

Linear-Mappings Diese Art von Berechnung bestimmt die Aktivierung eines Neurons basierend auf den Aktivierungen verbundener Neuronen [1, S. 8]. In Gleichung 4.4 wurde dies für Feed-Forward Netze vorgestellt. Bei LSTMs ist diese Berechnungsart bei der Verarbeitung der Netzeingabe in Gleichung 4.12 wiederzufinden. In diesem Fall wird sie zweimal angewendet, da sowohl rekurrente Verbindungen also auch

Verbindungen zur vorherigen Schicht beachtet werden müssen. Die Umverteilung der Relevanz von dem betrachteten Neuron zu den Verbundenen kann so erfolgen, wie es in Kapitel 4.2.3 für das gewöhnliche LRP-Verfahren beschrieben wurde [1, S. 8].

Gated-Interactions Bei der Berechnung des Cell-State (Gleichung 4.15) und des Hidden-State (Gleichung 4.17) werden Gatter-Aktivierungen mit anderen Signalen wie beispielsweise der Eingabe z_t multipliziert. Die Relevanz, die in einem Schritt des LRP-Verfahrens einem Produkt zugeordnet wird, muss auf die beiden Faktoren aufgeteilt werden. Die einfachste Möglichkeit diese Aufteilung vorzunehmen, ist die gesamte Relevanz dem Signal zuzuordnen und das jeweilige Gatter zu vernachlässigen [1, S. 8f.].

Accumulation Eine Akkumulation findet insbesondere bei der Berechnung des Cell-State (Gleichung 4.15) statt [1, S. 10f.]. Der neue Cell-State wird mit zwei Summanden berechnet. Der eine besteht aus den Signalen vom vorherigen Cell-State, welche beibehalten werden, während der andere Summand aus der Netzeingabe berechnet wird. Arras u. a. scheinen keine Empfehlung für die Aufteilung der Relevanz auf die beiden Summanden abzugeben. Es ist jedoch naheliegend, dass die Relevanz proportional auf beide Summanden aufgeteilt werden sollte.

Mit diesen Informationen wurde LRP für LSTMs implementiert. Das Ziel soll dabei sein, zu bestimmen, wie relevant jede verwendete Eingabe für *einen* Zeitpunkt in der Ausgabesequenz ist. Dafür wird zunächst jedem Neuron der Ausgabeschicht zu jedem Zeitschritt eine Relevanz zugeordnet. Es wird angenommen, dass der zu betrachtende Zeitpunkt den letzten Eintrag in der Eingabesequenz darstellt. Die Relevanz der Neuronen der letzten Schicht zu diesem Zeitpunkt ist deren Aktivierungen gleichzusetzen, wie es auch beim gewöhnlichen LRP-Verfahren der Fall ist. Zu allen anderen Zeitpunkten sei die Relevanz null. Mit den Aktivierungen der Ausgabeschicht kann wie bei einem Feed-Forward Netzwerk LRP angewendet werden, bis eine LSTM Schicht erreicht ist.

Eine Ausgabe eines LSTM ist von der aktuellen und allen vorherigen Eingaben abhängig, nicht jedoch von den folgenden. Es muss also möglich sein, die Relevanz auf alle vorherigen Eingaben zu verteilen. Deswegen erfolgt die LRP beginnend mit dem letzten Zeitschritt der Sequenz, wie es in Zeile 4 vom Algorithmus 3 dargestellt ist.

In jedem Schritt wird dann die Relevanz auf die aktuellen Netzeingaben und die internen Zustände aufgeteilt (siehe Zeile 7). Der Anteil, welcher auf die Netzeingaben aufgeteilt

wurde, bestimmt, wie relevant die aktuellen Sensorwerte sind. Die Relevanz der internen Zustände wird verwendet, um dieselbe Aussage für die vorherigen Zeitschritte zu treffen.

Algorithmus 3 LRP-Algorithmus für LSTM-Netzwerke

```
1: function LSTM_LRP(sequence, hidden_states, relevances, net)
2:     state_relevance ← [0, 0, ...]
3:     input_relevances ← []
4:     for i ← LENGTH(features), 0 do
5:         relevance ← relevances[i] + state_relevance
6:         features ← sequence[i]
7:         input_relevance, state_relevance ← LSTM_LRP_STEP(features, relevance, net)
8:         PREPEND(input_relevances, input_relevance)
9:     end for
10:    return input_relevances
11: end function

12: function LSTM_LRP_STEP(features, relevance, net)
13:     input_rel, state_rel ← SPLIT_RELEVANCE([net.input_product, net.cell_state_product], rel)
14:     feature_rel, recurrent_rel ← SPLIT_RELEVANCE([net.input_activations, net.recurrent_activations, input_rel)
15:     sensor_rel ← LRP_STEP(feature_rel, net.input_weights, net.input_activations)
16:     state_rel ← state_rel + LRP_STEP(recurrent_rel, net.recurrent_weights, net.recurrent_activations)
17:     return sensor_rel, state_rel
18: end function

19: function SPLIT_RELEVANCE(activations, relevance)
20:     total_activations ← SUM(activations)
21:     partial_relevances ← []
22:     for i ← LENGTH(activations), 0 do
23:         partial_relevances[i] ← relevance * (activations[i]/total_activations)
24:     end for
25:     return partial_relevances
26: end function
```

Um das Verfahren für die Aufteilung der Relevanz in einen bestimmten Zeitschritt herzuleiten, können die Gleichungen 4.12 bis 4.17 rückwärts durchlaufen werden. Die Relevanz R_{y_t}, die den internen Zuständen zugeordnet ist, muss nach auf Gleichung 4.17 auf o_t und c_t aufgeteilt werden. Da Gattern nach Arras u. a. keine Relevanz zugeordnet werden muss, gilt $R_{y_t} = R_{c_t}$. Die Gleichung 4.15 wiederum enthält sowohl eine Akkumulation, als auch Verwendungen von Gattern. Da letztere vernachlässigt werden können, muss die Relevanz auf die Eingabe z_t und den vorherigen Cell-State c_{t-1} aufgeteilt werden. Die proportionsgerechte Aufteilung der Relevanz wird durch die Funktion in Zeile 19 von Algorithmus 3 vorgenommen.

Die Relevanz der Eingabe R_{z_t} kann über die Verbindungen der Neuronen, wie sie durch die Gleichung 4.12 ausgedrückt werden, noch weiter verteilt werden. Hier werden zwei Linear-Mappings verwendet. Bevor dafür das normale LRP-Verfahren angewendet werden kann, muss die Relevanz erneut auf die beiden Summanden aufgeteilt werden, wie es auch beim Cell-State der Fall war.

Ein Schritt des LRP-Verfahrens wird benötigt, um über die rekurrenten Verbindungen eine Relevanz für den vorherigen Hidden-State zu ermitteln (siehe Zeile 16). Bei den Verbindungen zur vorherigen Schicht können mehrere Schritte notwendig sein, da das Netzwerk vor der rekurrenten Schicht noch Feed-Forward Schichten enthält. In der Implementierung in Algorithmus 3 wurde jedoch angenommen, dass es keine Schichten vor der LSTM Schicht gibt. Die Relevanz des internen Zustands setzt sich aus der Summe $R_{c_{t-1}} + R_{y_{t-1}}$ zusammen.

5 Auswertung

Mit jedem der drei betrachteten Verfahren wurde nun ein Modell erzeugt, welches eine Abschätzung für die RUL eines Produktionsleitsystems abgeben kann. Zudem kann für die Abschätzungen aller Modelle eine Erklärung erzeugt werden, indem für die verwendeten Sensorwerte jeweils ein Maß der Relevanz hergeleitet wird. In diesem Kapitel soll nun bestimmt werden, wie geeignet die verwendeten Verfahren für diesen Anwendungsfall sind.

Dafür wird zunächst verifiziert, dass die Erklärungsverfahren korrekt funktionieren. Anschließend werden die von den verschiedenen Modellen produzierten Vorhersagen miteinander verglichen. Die Ergebnisse werden anschließend diskutiert, um ein bevorzugtes Verfahren auszuwählen. Darauf folgend werden die Ergebnisse der Bachelorarbeit kritisch hinterfragt, um Optimierungspotenzial zu identifizieren.

5.1 Verifikation der Erklärungsverfahren

Während die verwendeten neuronalen Netze und Entscheidungsbäume in getesteten Softwarebibliotheken implementiert sind, mussten die Erklärungsverfahren selbst implementiert werden. Da dabei das Risiko von Programmierfehlern höher ist, sollen diese nun verifiziert werden. Die Erklärungen von vorhergesagten RUL-Werten von Produktionsleitsystemen sind dafür allerdings ungeeignet, weil die erwarteten Relevanz-Werte nicht bekannt sind, sondern nur im Einzelfall von Experten abgeschätzt werden können.

Stattdessen wird ein künstlich erzeugter Datensatz für die Validierung verwendet. Aus diesem soll die tatsächliche Relevanz jedes Features klar hervorgehen. Daher besteht der Datensatz aus fünf Features, für welche jeweils Zufallszahlen zwischen null und eins als Werte ausgewählt werden. Das Label zu jedem Eintrag im Datensatz ist die Summe aller Zufallszahlen. Die Relevanz jedes Features muss also proportional zu dessen verwendetem Wert sein. Für LSTMs werden Sequenzen als Eingaben benötigt. Dasselbe Prinzip kann

jedoch auch hier angewendet werden. Die Labels sind dann die Summe der Zufallszahlen über alle Features und die gesamte Sequenz.

Ein Entscheidungsbaum, ein Feed-Forward Netz und ein LSTM werden dann trainiert, um die Summenbildung durchzuführen. Mit den jeweiligen Erklärungsverfahren wird dann für jedes Sample die Relevanz der Eingabewerte bestimmt. Die Relevanzwerte und die tatsächlichen Eingaben werden dann auf den Bereich zwischen null und eins skaliert, um sie vergleichbar zu machen. Nun kann die durchschnittliche Differenz zwischen den Eingaben und den Relevanzwerten bestimmt werden. Der Durchschnitt dieser Werte über alle Elemente des generierten Datensatzes wird verwendet, um die Qualität der Erklärungsverfahren zu beurteilen.

Abbildung 5.1 veranschaulicht das Vorgehen anhand einer Eingabe. Diese ist in 5.1a zu sehen. In 5.1b sind die ermittelten Wichtigkeitswerte abgebildet. In beiden Fällen sind die Werte bereits skaliert. Die Fehler, welche in 5.1c zu sehen sind, zeigen wie gut die einzelnen Wichtigkeitswerte sind. In diesem Beispiel wurde insbesondere den Werten einiger Features in den Schritten 7 und 8 der Eingabesequenz eine zu hohe Wichtigkeit zugeordnet, was sich in den ermittelten Differenzen widerspiegelt.

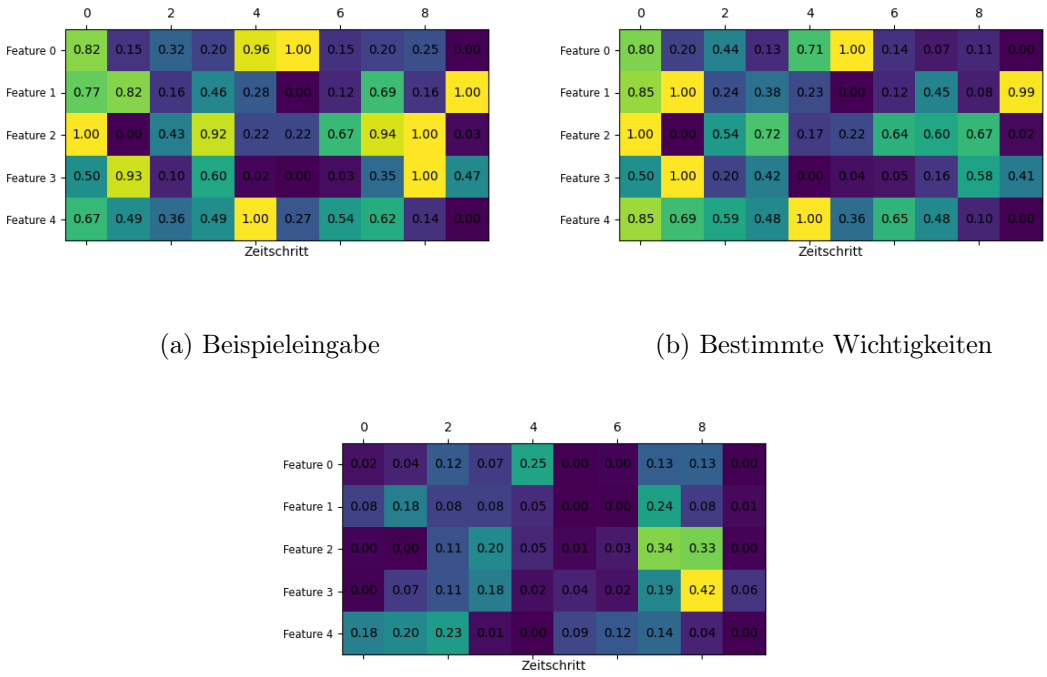
Tabelle 5.1 zeigt die Ergebnisse der Verifikation. Für jedes Verfahren sind dort zwei Fehlermaße als durchschnittlicher absoluter Fehler angegeben. Zum einen wird dadurch die Vorhersagegenauigkeit charakterisiert. Zum anderen ist erkennbar, wie akkurat den Features eine Relevanz zugeordnet wird. Beide Varianten von LRP liefern mit Fehlern von weniger als 10 Prozent akzeptable Ergebnisse.

Tabelle 5.1: Fehler der verschiedenen Erklärungsverfahren mit synthetischen Daten

Verfahren	**Fehler der Vorhersagen**	**Fehler der Wichtigkeiten**
Entscheidungsbaum	1,742	0,396
Feed-Forward Netz	0,293	0,010
LSTM	0,385	0,057

Die mit dem Entscheidungsbaum ermittelten Erklärungen sind jedoch nicht optimal. Das lässt sich allerdings dadurch erklären, dass die Ausgaben des Entscheidungsbaums verglichen mit denen anderen Verfahren ebenfalls schlechter sind.

Es gibt einen Grund für die schlechtere Genauigkeit des Entscheidungsbaums, die mit der Struktur der Bäume zusammenhängt. Entscheidungsbäume können nämlich nicht gut mit additiven Zusammenhängen zwischen Features umgehen. Damit ist gemeint, dass



(a) Beispieleingabe

(b) Bestimmte Wichtigkeiten

(c) Absolute Differenz aus Eingaben und Wichtigkeiten

Abbildung 5.1: Fehlerermittlung mit dem Erklärungsverfahren für LSTMs

sich das Ergebnis von der Summe mehrerer Features abhängig ist, wie es im verwendeten Datensatz der Fall ist. Während diese Summe in neuronalen Netzen durch Verbindungen zu einem Neuron gebildet werden kann, muss dies in einem Entscheidungsbaum über viele verschiedene Knoten abgebildet werden, die jeweils mit verschiedenen Schwellwerten prüfen. [3, S. 313]

5.2 Vergleich der Modelle

Es sollen nun die geeignetsten Vorhersage- und Erklärungsverfahren ausgewählt werden. Für den Vergleich der Vorhersagegenauigkeit werden die Fehlermaße auf Validierungsdaten ermittelt. Diese sind Tabelle 5.2 zu entnehmen. Das LSTM-Netzwerk weist den geringsten durchschnittlichen Fehler auf. Auch die Wurzel der quadrierten Fehler, welche Ausreißer stärker bewertet, liegt signifikant unter denen der anderen Modelle.

Somit ist das LSTM-Netzwerk am besten geeignet, um die RUL des untersuchten Produktionsleitsystems vorherzusagen. Es ist dabei zu beachten, dass ein Fehler von 3782,00 min bedeutet, dass Ausfälle im Durchschnitt zwei bis drei Tage zu früh oder gar zu spät in der vorhergesagte RUL widergespiegelt werden. Ein dementsprechend großer Puffer muss gewählt werden, um rechtzeitig vor einem Ausfall Gegenmaßnahmen einleiten zu können.

Tabelle 5.2: Fehler der trainierten Modelle auf Validierungsdaten

Verfahren	**Mean-Absolute-Error**	**Root-Mean-Squared-Error**
Entscheidungsbaum	5775,939	9283,665
Feed-Forward Netz	5184,149	7751,189
LSTM	3782,003	5663,850

Wie bereits im vorherigen Kapitel angesprochen wurde, werden die Gründe für Ausfälle nicht dokumentiert und sind somit nicht bekannt. Das macht eine objektive Beurteilung der Qualität der Erklärungsverfahren unmöglich. Dennoch wurden exemplarisch mit den Erklärungsverfahren Relevanzwerte ermittelt und auf Plausibilität geprüft. Dabei wurden folgende Beobachtungen gemacht.

- Die LRP-Verfahren für Feed-Forward Netze und LSTMs verteilen die Relevanz häufig auf dieselben Sensoren. Die Relevanzwerte des Entscheidungsbaums weichen häufig stark von denen der anderen Verfahren ab, was sich mit den Ergebnissen des Tests auf künstlichen Daten in Kapitel 5.1 deckt.
- Die für das LSTM ermittelten Relevanzwerte lassen deutlich erkennen, dass es bestimmte Zeitschritte gibt, die von besonderem Interesse sind. Ein Beispiel sei in Abbildung 5.2 gegeben. In diesem Fall fällt unter anderem der 60. Zeitschritt der Eingabesequenz besonders auf. In diesem wirken sich unter anderem die Werte von zwei Sensoren stark negativ auf die RUL aus.

Da die Erklärungen für Vorhersagen des LSTM in den betrachteten Fällen sehr ähnlich mit denen für das Feed-Forward Netz waren und zudem noch relevante Zeitschritte identifizieren können, wird das LSTM-Netzwerk mit dem modifizierten LRP-Verfahren als bevorzugte Methode für Vorhersagen und Erklärungen von Ausfällen des Produktionsleitsystems ausgewählt.

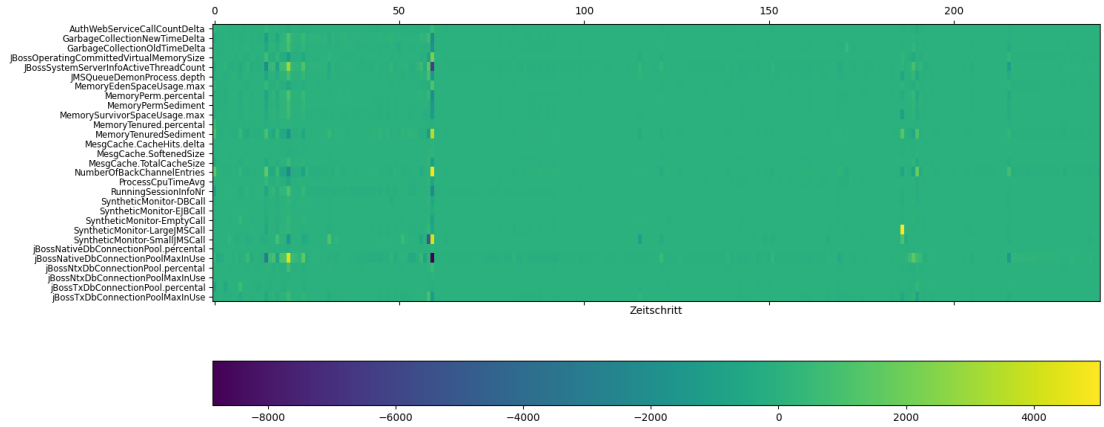


Abbildung 5.2: Relevanzmatrix für die Erklärung einer LSTM-Vorhersage

5.3 Kritik

Mit den erzielten Ergebnissen ist deutlich geworden, wie geeignet die betrachteten Machine-Learning-Verfahren für das Vorhersagen von Ausfällen in diesem Anwendungsfall sind. Jedoch gibt es noch Verbesserungspotenzial, da die für das Training und Testen der Modelle verwendeten Label verunreinigt sind.

Wie in Kapitel 3.2.2 beschrieben ist, wird die RUL aus den „SystemUptime“ Sensoren berechnet. Diese Umwandlung basiert auf der Annahme, dass jedes Mal, wenn die Werte dieses Sensors wieder bei null beginnen, die überwachte Instanz ausgefallen ist. Jedoch kann die Verfügbarkeit des überwachten Systems auch aus anderen Gründen eingeschränkt sein.

Beispielsweise könnte es für eine Softwareaktualisierung heruntergefahren werden. Solche Ereignisse werden bei der Berechnung der RUL fälschlicherweise ebenfalls als Ausfälle kategorisiert. Instanzen, die für eine Softwareaktualisierung heruntergefahren werden, weisen jedoch nicht die Merkmale auf, die auf einen Ausfall hinweisen. Solche Ereignisse sollten, wenn sie nicht besonders behandelt werden, dazu führen, dass die RUL von allen Verfahren gleichermaßen zu hoch eingeschätzt wird.

Setzt man eines der Machine-Learning-Verfahren ein, um Ausfälle vorherzusagen, kann es außerdem sein, dass, weil ein Ausfall bevorsteht, ein System zu einem besser geeigneten Zeitpunkt neu gestartet wird. Dieser Neustart des Systems würde bei der Berechnung der RUL ebenfalls wie ein kurzer Ausfall behandelt werden. Benutzt man die neu erfassten

Sensordaten dann, um das verwendete Modell zu verbessern, ergibt sich eine Feedbackschleife, durch welche die vorhergesagte RUL immer weiter sinkt.

Die genannten Effekte ließen sich umgehen, wenn man für jede Zeitspanne wüsste, ob es sich um einen Ausfall oder ein geregeltes Herunterfahren des Systems handelt. Diese Information wird allerdings nicht manuell erfasst. Es könnte stattdessen möglich sein, aus den Logdateien des Systems zu bestimmen, auf welche Weise das System beendet wurde. Jedoch liegen diese für die meisten Systeme ebenfalls nicht vor.

Um zu ermitteln, wie effektiv die Verfahren zum Erklären von Ausfällen sind, wurden lediglich ausgewählte Systeme zu bestimmten Zeitpunkten betrachtet. Die erfassten Relevanz-Werte wurden dann mit der Einschätzung eines Experten verglichen. Diese Methode zum Vergleichen der Verfahren ist jedoch anfällig für Fehleinschätzungen, da nicht sicher bekannt ist, warum ein Ausfall tatsächlich eingetreten ist oder ob nicht vielleicht ein geregelter Neustart durchgeführt wurde.

Um die genannten Probleme zu lösen, muss für jede Zeitspanne, die ein System nicht verfügbar ist, ein Grund bekannt sein. Diese Information ist jedoch für die vorliegenden Sensordaten nicht trivial zu beschaffen. Möchte man die Vorhersage- und Erklärungsverfahren jedoch in Echtzeit einsetzten können, sollten diese Informationen für eine genauere Validierung der Verfahren vorliegen.

5.4 Weiterentwicklung

Neben der genannten Kritik gibt es zwei Aspekte, in denen sich eine weitere Untersuchung lohnen könnte.

Zum einen wurde von drei betrachteten ML-Verfahren eins ausgewählt, welches die besten Ergebnisse liefert. Jedoch besteht nicht die Notwendigkeit sich auf ein Verfahren festzulegen. Stattdessen können Vorhersagen mit mehreren Modellen erzeugt werden. Aus diesen wird dann eine gemeinsame Vorhersage bestimmt, indem der Durchschnitt gebildet wird. Dieses Vorgehen ist als Ensemble-Learning bekannt. [10, S. 748] Bei der Anwendung von Ensemble-Learning wäre in diesem Fall jedoch noch unklar, wie für die durchschnittlichen Vorhersagen Erklärungen generiert werden können.

Zum anderen legen die genaueren Vorhersagen des LSTM nahe, dass der zeitliche Verlauf bestimmter Sensoren hilfreich ist, um gute Vorhersagen zu treffen. Den verwendeten Entscheidungsbäumen und Feed-Forward Netzen fehlten diese Informationen.

Bei anderen Anwendungen solcher Modelle für Predictive-Maintenance werden jedoch künstliche Features konstruiert, um Informationen des zeitlichen Verlaufs zu berücksichtigen [9, S. 15]. Beispiele dafür könnten die durchschnittliche Steigung oder die Varianz der Werte eines bestimmten Zeitfensters sein.

Literaturverzeichnis

[1] Arras, Leila ; Arjona-Medina, José ; Widrich, Michael ; Montavon, Grégoire ; Gillhofer, Michael ; Müller, Klaus-Robert ; Hochreiter, Sepp ; Samek, Wojciech: Explaining and Interpreting LSTMs. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning.* Springer International Publishing, 09 2019, S. 211–238

[2] Burkov, Andriy: *Neural Networks and Deep Learning.* Kap. 6. In: *The Hundred-Page Machine Learning Book*, Kindle Direct Publishing, 2019. – ISBN 978-1-9995795-0-0

[3] Hastie, T. ; Tibshirani, R. ; Friedman, J.H.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009 (Springer series in statistics). – ISBN 978-0-387-84884-6

[4] LeCun, Yann ; Bottou, Léon ; Orr, Genevieve B. ; Müller, Klaus-Robert: Efficient BackProp. In: Montavon, Grégoire (Hrsg.) ; Orr, Genevieve B. (Hrsg.) ; Müller, Klaus-Robert (Hrsg.): *Neural Networks: Tricks of the Trade (2nd ed.)* Bd. 7700. Springer, 2012, S. 9–48. – ISBN 978-3-642-35288-1

[5] Linardatos, Pantelis ; Papastefanopoulos, Vasilis ; Kotsiantis, Sotiris: Explainable AI: A Review of Machine Learning Interpretability Methods. In: *Entropy* 23 (2020), 12, S. 18

[6] Meyer, H. ; Fuchs, F. ; Thiel, K.: *Manufacturing Execution Systems (MES): Optimal Design, Planning, and Deployment.* McGraw-Hill Education, 2009. – ISBN 9780071626026

[7] Molnar, Christoph: *Interpretable Machine Learning.* 2019. – ISBN 0-244-76852-8

[8] Montavon, Grégoire ; Binder, Alexander ; Lapuschkin, Sebastian ; Samek, Wojciech ; Müller, Klaus-Robert: *Layer-Wise Relevance Propagation: An Overview.*

S. 193–209. In: Samek, Wojciech (Hrsg.) ; Montavon, Grégoire (Hrsg.) ; Vedaldi, Andrea (Hrsg.) ; Hansen, Lars K. (Hrsg.) ; Müller, Klaus-Robert (Hrsg.): *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Cham : Springer International Publishing, 2019. – ISBN 978-3-030-28954-6

[9] Ran, Yongyi ; Zhou, Xin ; Lin, Pengfeng ; Wen, Yonggang ; Deng, Ruilong: A Survey of Predictive Maintenance: Systems, Purposes and Approaches. In: *arXiv e-prints* (2019), Dezember, S. 36

[10] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 3. Prentice Hall, 2010. – ISBN 0-13-604259-7

[11] Wichert, Andreas ; Sa-Couto, Luis: *Machine Learning: A Journey to Deep Learning with Exercises and Answers*. World Scientific, 2021. – ISBN 978-981-12-3405-7

A Anhang

Tabelle A.1: Übersicht der verwendbaren Sensoren

Sensorname	**Beschreibung**
AuthWebServiceCallCountDelta	This sensor provides the number of requests for PAS-X authentication by PAS-X central login. This requests will be executed by the Jboss web server. The measured value is the number of requests since the last measurement of this sensor.
CodeCache.max	This sensor provides the size of memory (in bytes) that can be used for the Java Code Cache. This size is configured with the startup options of the JVM. Debug purpose only.
GarbageCollectionNewTimeDelta	This sensor provides the time in ms spent for the JavaVirtualMachines GarbageCollections on Newspace in the last measurement cycle.
GarbageCollectionOldTimeDelta	This sensor provides the time in ms spent for the JavaVirtualMachines GarbageCollections on Oldspace in the last measurement cycle.
jBossEdiDbConnectionPoolMax	This sensor provides the size (number of database connections) configured for JCoffee EDI (External Data Interface) connection pool of PAS-X application server.
jBossEdiDbConnectionPoolMaxInUse	This sensor provides the maximum number of used connections of JCoffee EDI (External Data Interface) connection pool since the last reset of the statistic. Typically start of the PAS-X of PAS-X application server.
JBossIsAvailableWithOutStore	This sensor checks if the JBoss is online and accessible the value is 1. This sensor is evaluated every 10 seconds typically. No value is stored in the monitoring database. The sensor is used internally for the dashboard status display only.
jBossNativeDbConnectionPool.percental	This sensor provides the percentage of used database connections of JBoss native connection pool of PAS-X application server.
jBossNativeDbConnectionPoolMax	This sensor provides the size (number of database connections) configured for JBoss native connection pool of PAS-X application server.
jBossNativeDbConnectionPoolMaxInUse	This sensor provides the maximum number of used connections of JBoss native connection pool since the last reset of the statistic. Typically start of the PAS-X of PAS-X application server.
jBossNtxDbConnectionPool.percental	This sensor provides the percentage of used database connections of direct JDBC (non transactional)connection pool of PAS-X application server.
jBossNtxDbConnectionPoolMax	This sensor provides the size (number of database connections) configured for direct JDBC (non transactional) connection pool of PAS-X application server.
jBossNtxDbConnectionPoolMaxInUse	This sensor provides the maximum number of used connections of direct JDBC (non transactional) connection pool since the last reset of the statistic. Typically start of the PAS-X of PAS-X application server.
JBossOperatingCommittedVirtualMemorySize	This sensor provides the memory size being used by application server JavaVirtualMachine in bytes.
JBossOperatingSystemAvailableProcessors	This sensor provides the number of cores available in the application server.
JBossSystemServerInfoActiveThreadCount	This sensor provides the number of threads being used by the application server Jboss.

Tabelle A.1: Übersicht der verwendbaren Sensoren

Sensorname	**Beschreibung**
jBossTxDbConnectionPool.percental	This sensor provides the percentage of used database connections of Jcoffee transactionalconnection pool of PAS-X application server.
jBossTxDbConnectionPoolMax	This sensor provides the size (number of database connections) configured for Jcoffee transactional connection pool of PAS-X application server.
ProcessCpuTimeAvg	This sensor provides the average CPU utilization of the java process in nano seconds. The measured time is the time in the last measurement cycle.
SyntheticMonitor	This is a 'wrapper'-sensor of the group 'synthetic monitoring'. The results can be found in the sensors: SyntheticMonitor-EmptyCallTime SyntheticMonitor-DatabaseCallTime SyntheticMonitor-EJBCallTime SyntheticMonitor-SmallJMSServiceFullCallTime and SyntheticMonitor-LargeJMSServiceFullCallTime. The unit of all measured values is ms.
SystemUptime	This sensor provides the time from the start of the service up to now in milliseconds.
ThreadPoolQueueSize	This sensor provides the number of thread requests waiting to be scheduled in threads. This value shall by typically 0 indicating that the thread pool is big enough and all thread requests will be processed. If the value is bigger than 0 for a longer time Werum service shall be asked to check the system.
TotalPhysicalMemorySize	This sensor provides the physical memory size of the server.
jBossTxDbConnectionPoolMax	This sensor provides the size (number of database connections) configured for Jcoffee transactional connection pool of PAS-X application server.
jBossTxDbConnectionPoolMaxInUse	This sensor provides the maximum number of used connections of Jcoffee transactional connection pool since the last reset of the statistic. Typically start of the PAS-X of PAS-X application server.
JMSQueueDemonProcess.depth	This sensor provides the number of waiting JMS messages in the demon process queue of the application server (statistical use for Jboss optimization only).
MemoryEdenSpaceUsage.max	This sensor provides the maximum memory utilization of the JVM YoungGeneration/Eden memory area in bytes.
MemoryPerm.max	This sensor provides the size of the JVM PermSpace in bytes.
MemoryPerm.percental	This sensor provides the current utilization of the JVM permanet space memory area in percent.
MemorySurvivorSpaceUsage.max	This sensor provides the size of the JVM SurvivorSpace in bytes.
MemoryTenured.max	This sensor provides the size of the JVM OldSpace/Tenured memory area in bytes.
MemoryTenured.percental	This sensor provides the current utilization of the JVM OldSpace/Tenured memory area in percent.
MesgCache.CacheHits.delta	This sensor provides the number of application server JVM hard referenced message accessess in the last measurement cycle. This is statistical data for optimizing the JVM.

Tabelle A.1: Übersicht der verwendbaren Sensoren

Sensorname	**Beschreibung**
`MesgCache.HighMemMark`	This sensor provides the value of the application server JBoss MessageCache HighMemMark in Mbyte. This is typically a paramaterized value and shall be set to 80% of the total heapspace. JMS messages will start to be moved into the database for releasing heap memory when the total heap utilization excesses that value. The PAS-X default is 80% of the heap space of the J-Boss. The heapspace is parameterized in the file run.bat with -Xmx. In doubt compare with the result of the sensor InputArguments-JVM showing the -Xmx argument or review the file \server\pasx\deploy-hasingleton\jms\oracle-jdbc2-service.xml (cluster) or \server\pasx\deploy\jms\oracle-jdbc2-service.xml (non cluster)
`MesgCache.MaxMemMark`	This sensor provides the value of the application server JBoss MesssageCache MaxMemMark in Mbyte. This is typically a paramaterized value and shall be set to 90% of the total heapspace. All JMS Messages will be moved into the database for releasing heap memory when the total heap utilization excesses that value. The PAS-X default is 90% of the heap space of the J-Boss. The heapspace is parameterized in the file run.bat with -Xmx. In doubt compare with the result of the sensor InputArgumentsJVM showing the -Xmx argument or review the file \server\pasx\deploy-hasingleton\jms\oracle-jdbc2-service.xml (cluster) or \server\pasx\deploy\jms\oracle-jdbc2-service.xml (non cluster)
`MesgCache.SoftenedSize`	This sensor provides application server JBoss JMS MessageCache total number of messages softened since the last boot.
`MesgCache.TotalCacheSize`	This sensor provides application server JBoss total number of messages that are being managed by the message cache.

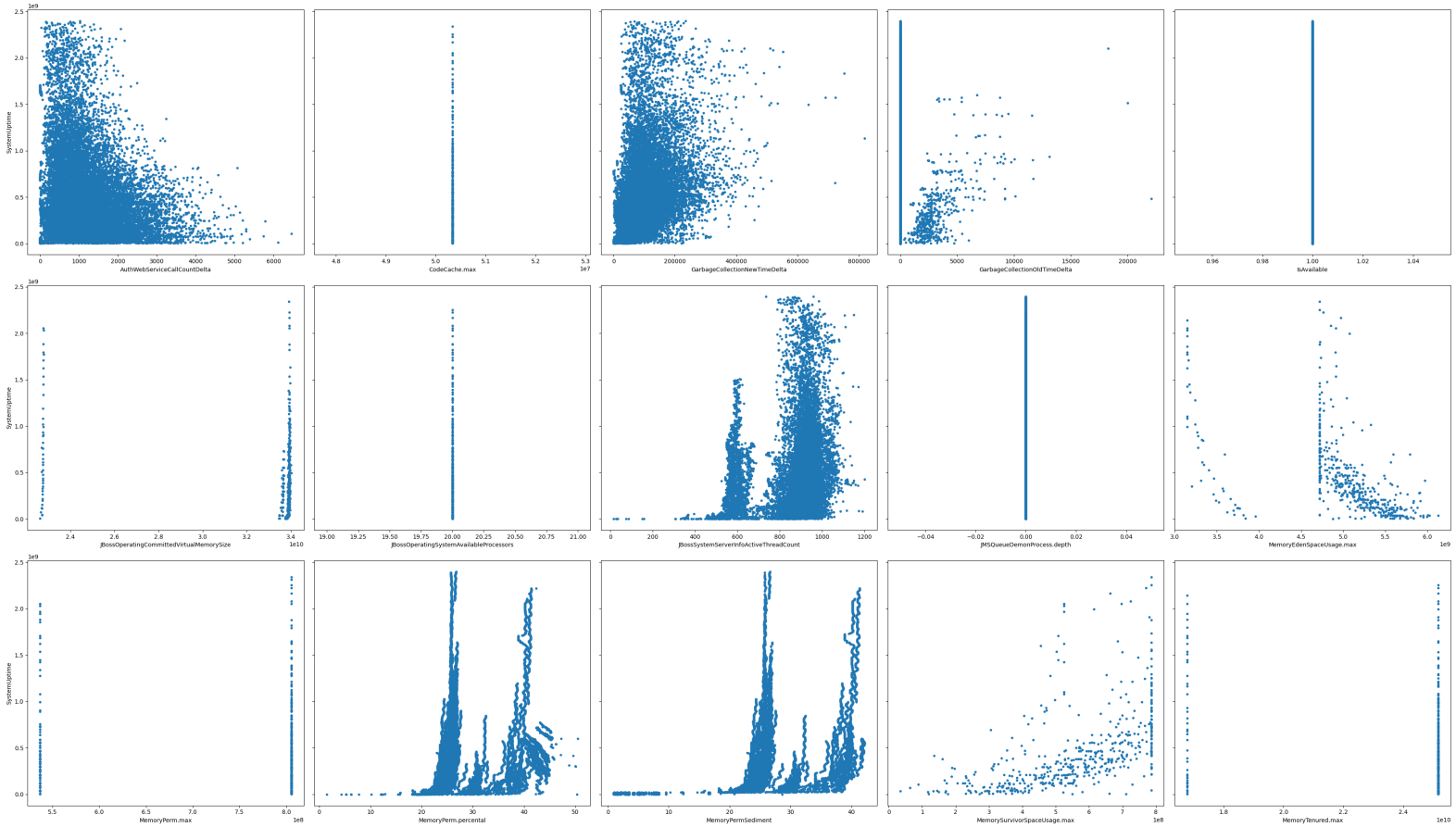


Abbildung A.1: Partial Dependency Plots der Sensoren eines Anwendungsservers

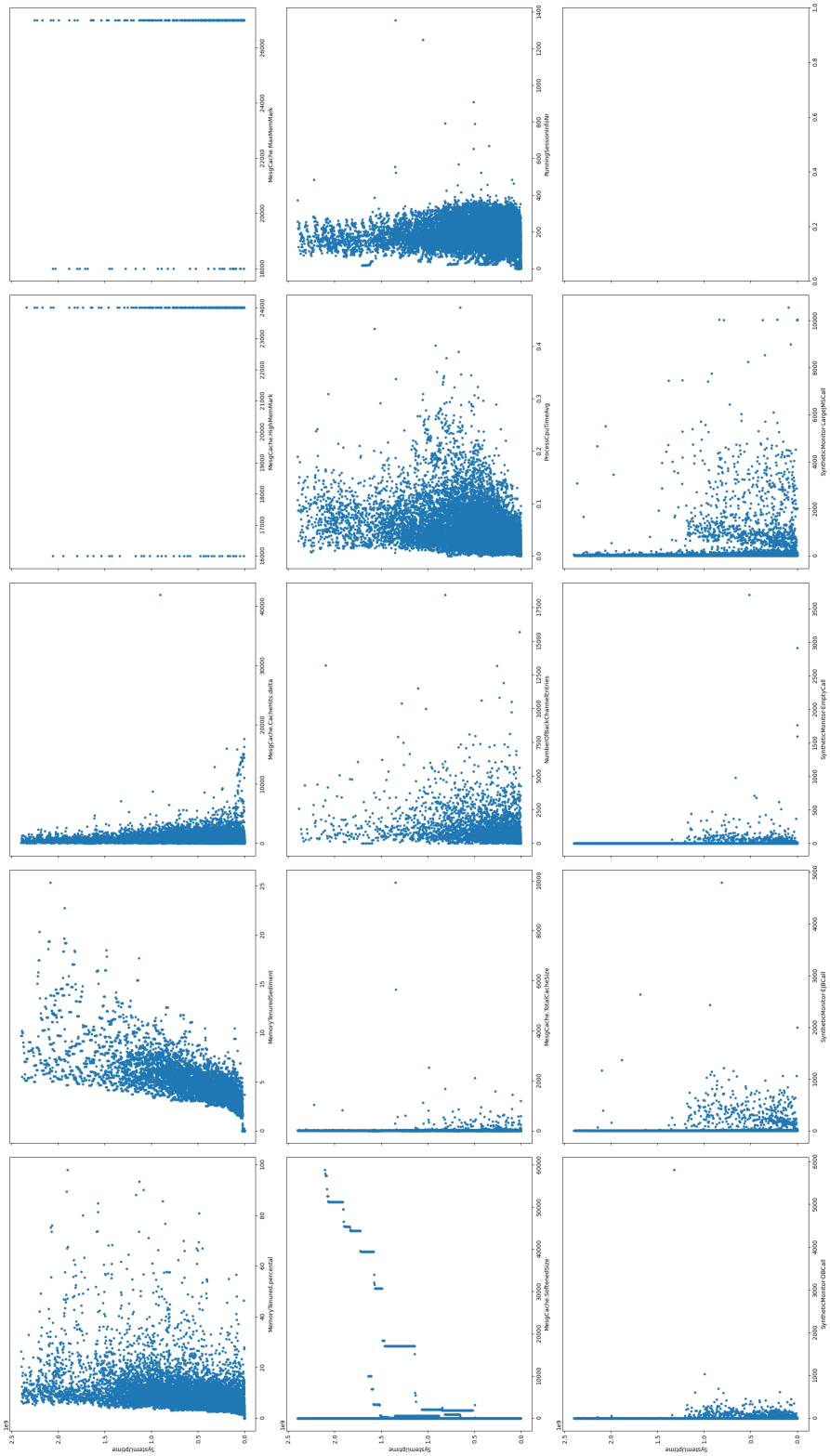


Abbildung A.1: Partial Dependency Plots der Sensoren eines Anwendungsservers

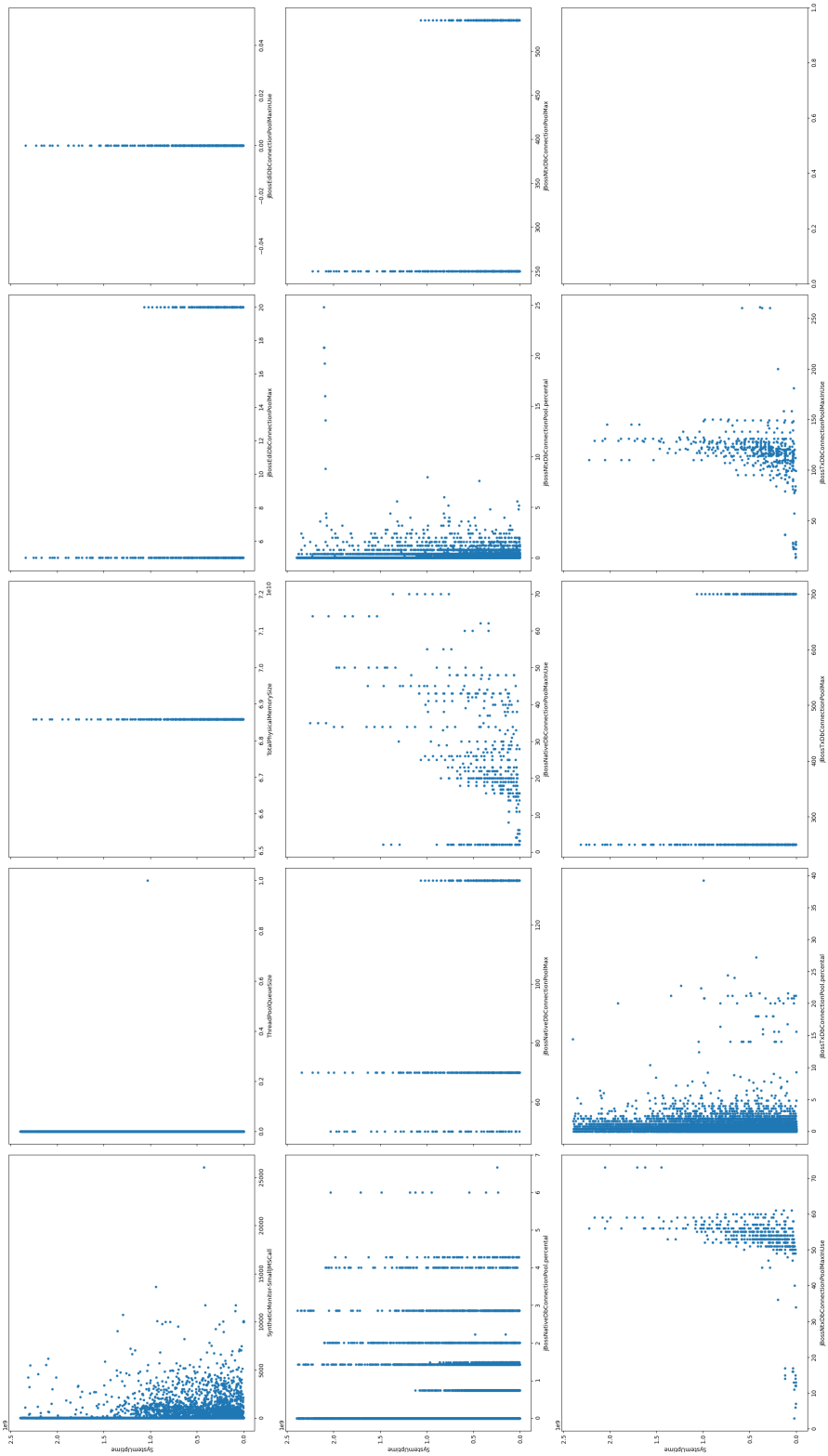


Abbildung A.1: Partial Dependency Plots der Sensoren eines Anwendungsservers

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

[redacted]	04.07.2022	[redacted]
Ort	Datum	Unterschrift im Original