TU WIEN

Forschungsbereich Maschinenbauinformatik und Virtuelle Produktentwicklung

Überwachung und Anpassung eines Algorithmus zur Anomalieerkennung von Produktionsprozessdaten nach der Implementierung

Diplomarbeit

Werle Fabian

Matrikel.-Nr.: 01611156

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

DIPLOMARBEIT

Überwachung und Anpassung eines Algorithmus zur Anomalieerkennung von Produktionsprozessdaten nach der Implementierung

ausgeführt zum Zwecke der Erlangung des akademischen Grades Diplom-Ingenieur (DI)
unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Manfred Grafinger

E307-04 – Institut für Konstruktionswissenschaften und Produktentwicklung

Forschungsbereich Maschinenbauinformatik und Virtuelle Produktentwicklung

eingereicht an der Technischen Universität Wien

Fakultät für Maschinenwesen und Betriebswissenschaften

von

Werle Fabian

Matrikelnummer: 01611156

Ort, Datum ____

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Aus fremden Quellen wörtlich oder sinngemäß übernommene Zitate sind als solche kenntlich gemacht. Die Arbeit hat noch nicht anderweitig in gleicher oder ähnlicher Form zu Prüfungszwecken vorgelegen.

Die Arbeit darf über die Hochschulbibliothek zugänglich gemacht werden.

Ort, Datum	Unterschrift

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Kurzfassung

Machine Learning Algorithmen werden immer stärker in der heutigen Welt eingesetzt und sind speziell seit der Veröffentlichung von Chat-GPT in der breiten Bevölkerung angekommen. Eine Herausforderung, die viele Unternehmen bei der Anwendung von Machine Learning Algorithmen erleben, ist die Überführung eines Machine Learning Algorithmus aus der Entwicklungsumgebung in die Produktion. Aus diesem Grund bleiben viele der entwickelten Algorithmen reine Entwicklungsprojekte und schaffen es nicht, einen Mehrwert für das Unternehmen zu liefern.

Ziel der vorliegenden Arbeit ist es, geeignete Strukturen innerhalb der Thyssenkrupp Presta Steering zu entwickeln, um einen Algorithmus zur Anomalieerkennung nach gängigen Standards in die Produktionsumgebung überführen zu können. Anschließend soll ebendieser in der Produktionsumgebung laufen überwacht werden, um eine hohe Qualität der Vorhersagen gewährleisten zu können.

In diesem Zusammenhang wird eine geeignete Architektur und ein allgemeiner Prozess basierend auf dem Konzept von *Machine Learning Operations (MLOps)* entwickelt. Sowohl die implementierte Architektur als auch der Prozess wurde anschließend anhand der Überführung eines Algorithmus zur Anomalieerkennung in die Produktion getestet. Des Weiteren wurde für den vorliegenden Algorithmus eine Überwachungsstrategie ausgearbeitet, in der sowohl die Eingangsdaten als auch die Ausgangsdaten analysiert werden. Die Überwachungsstrategie wurde abschließend mit Hilfe von realen Produktionsdaten und künstlich veränderten Produktionsdaten getestet.

Die Implementierung zeigte, dass mit Hilfe der ausgearbeiteten Architektur eine vollständige Automatisierung der Machine Learning Operations erreicht werden kann. Zudem ermöglicht der ausgearbeitete Prozess für alle Prozessteilnehmenden eine klare und transparente Vorgehensweise bei der Überführung des Algorithmus in die Produktionsumgebung. Bei der Testung der nachgelagerten Überwachungsstrategie wurde ein Drift innerhalb der Eingangsdaten erkannt, worauf hin geeignete Gegenmaßnahmen ergriffen werden können.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Machine learning algorithms are becoming more and more prevalent in today's world and have become popular within the broad society especially since the release of Large Language Models as Chat-GPT. One challenge that many companies experience in the application of machine learning algorithms is the transition of a machine learning algorithm from the development environment to the production environment. For this reason, most developed algorithms remain pure development projects and fail to deliver value to the business.

The aim of this thesis is to develop suitable structures within Thyssenkrupp Presta Steering in order to be able to transfer an anomaly detection algorithm to the production environment according to common standards. Subsequently, this algorithm will be monitored in the production environment to guarantee a high quality of predictions.

In this context, a suitable architecture and a general process based on the concept of *Machine Learning Operations (MLOps)* will be developed. Both the implemented architecture and the process were subsequently tested by transferring an anomaly detection algorithm into production. Furthermore, a monitoring strategy was elaborated for the present algorithm, in which both the input data and the output data are analysed. Finally, the monitoring strategy was tested using real production data and artificially modified production data.

The implementation showed that with the help of the elaborated architecture a complete automation of the machine learning operations can be achieved. In addition, the elaborated process enables a clear and transparent procedure for all process participants while transferring the algorithm into the production environment. During the testing of the downstream monitoring strategy, a drift within the input data was detected, whereupon appropriate measures can be taken.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Inhaltsverzeichnis

1 Einleitung 1
1.1 Problemstellung 1
1.2 Ziel der Arbeit 3
1.3 Vorgehensweise 3
2 Theoretische Grundlagen 5
2.1 Künstliche neuronale Netze 5
2.1.1 Grundlagen von Künstlichen Neuronalen Netzen 5
2.1.2 Training von künstlichen neuronalen Netzen 7
2.1.3 Arten des Trainings von KNNs 9
2.1.4 Trainingsparameter bei KNNs 10
2.1.5 Autoencoder 11
2.2 Der Machine Learning Lebenszyklus 12
2.2.1 Grundlagen 12
2.2.2 Herausforderungen innerhalb des ML-Lebenszyklus 13
2.3 DevOps 17
2.3.1 Grundlagen agiler Arbeitsweisen 17
2.3.2 Scrum 17
2.3.3 Grundlagen von DevOps 18
2.3.4 Herausforderungen von DevOps im ML-Lebenszyklus 19
2.4 MLOps 21
2.4.1 MLOps Prozessmodell 21
2.4.2 MLOps Reifegradmodell 22
2.4.3 Lösung der Herausforderungen mit MLOps 24
2.5 Bereitstellung von ML-Services 25
2.5.1 Cloud-Computing 25
2.5.2 Container 26
2.5.3 Serverlose Funktionen 28
2.5.4 Verpacken von ML-Services 28
2.5.5 Zugriff auf ML-Services 30
2.6 Testen von ML-Systemen 31
2.6.1 Daten Tests 32
2.6.2 Modul Tests 32
2.6.3 Integration Tests 32
2.6.4 Modell Tests 33
2.7 Überwachung von ML-Systemen 35
2.7.1 Drifterkennung 35
2.7.2 Systemüberwachung 39
2.7.3 Weitere Überwachungsaspekte 40

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.8 Neutraining von ML-Modellen 40

3 Umsetzungsbeschreibung 41

3.1 IST Stand im Unternehmen 41

3.2 Vorgeschlagene Architektur 42

3.2.1 Azure DevOps Umgebung 43
3.2.2 Azure Machine Learning Umgebung 44
3.2.3 Azure Monitor Umgebung 44
3.2.4 Azure Blob Storage Umgebung 45
3.2.5 Erweiterte Softwareumgebung 45

3.3 Vorgeschlagener Erstellungsprozess 46

3.4 Vorgeschlagener Pipeline Prozess 48

3.4.1 CI-Pipeline 49
3.4.2 CT-Pipeline 49
3.4.3 CD-Pipeline 50

3.5 Vorgeschlagene Überwachung 51

3.5.1 Systemüberwachung 51
3.5.2 Driftüberwachung 51

4 Praktische Implementierung 53

4.1 Implementierung der vorgeschlagenen Architektur 53

4.2 Definition der Modellanforderungen 55

4.3 Datenerfassung 57

4.4 Datenbereinigung 58

4.5 Datenbeschriftung & Feature Engineering 59

4.6 Modellerstellung & Modelltraining 59

4.7 Modellbewertung 60

4.8 Modellbereitstellung 61

4.8.1 Automatisiertes Training vorbereiten 61
4.8.2 Modell in serverlose Funktion konvertieren 61
4.8.3 Lokales Testen der serverlosen Funktion 61
4.8.4 Erstellen eines Pull Requests & Code Review 62
4.8.5 Durchführung der CI - Pipeline 62
4.8.6 Kontinuierliches Training 63
4.8.7 Kontinuierliche Bereitstellung des trainierten Modells 66
4.8.8 Testen des bereitgestellten Services 67

4.9 Modellveröffentlichung und Überwachung 67

4.9.1 Kontinuierliche Überwachung des Modells 68
4.9.2 Kontinuierliche Überwachung des Systems 72
4.9.3 Analyse der Effektivität der verwendeten Drifterkennung 73

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 Fazit und Ausblick 76
5.1 Fazit 76
5.2 Ausblick 77
6 Verzeichnisse 78
6.1 Literaturverzeichnis 78
6.2 Abbildungsverzeichnis 86
6.3 Tabellenverzeichnis 88

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Abkürzungsverzeichnis

CD ... Continuous Deployment
CDE ... Continuous Delivery
CI ... Continuous Integration
CPU ... Central Processing Unit (Prozessor)
DevOps ... Development & Operations
engl ... Englisch
HDF ... Hierarchical Data Format
http ... Hypertext Transfer Protocol
IaaS ... Infrastructure as a Service
KNN ... Künstliche Neuronale Netze
ML ... Machine Learning
MLOps ... Machine Learning Operations
NaN ... Not a Number
NCSA ... National Center for Supercomputing Applications
NOK ... Fehlerhaft (nicht Okay)
OK ... Fehlerfrei (Okay)
ONNX ... Open Neural Network Exchange
PaaS ... Platform as a Service
PB ... Protobuf
PoC ... Proof of Concept
RAM ... Random Access Memory (Arbeitsspeicher)
REST ... Representation State Transfer
RPC ... Remote Procedure Call
SaaS ... Software as a Service
vgl. ... vergleiche
VM ... Virtuelle Maschine
z.B. ... zum Beispiel

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1 Einleitung

1.1 Problemstellung

Wir leben in einer stark datengetriebenen Welt, in der weltweit jeden Tag eine enorme Anzahl an Daten generiert werden. Organisationen versuchen einen wirtschaftlichen Mehrwert aus diesen Daten zu erhalten, indem Machine Learning (*ML*) Techniken und Datenanalysen auf die generierten Daten angewendet werden. Das Ziel dieser Anwendungen ist es, Prozesse innerhalb des Unternehmens durch automatisierte Arbeitsabläufe effizienter zu gestalten und Services gegenüber den jeweiligen Kund_innen weiter zu verbessern. Dafür werden beispielsweise Chatbots oder Analysen des bestehenden Verhaltens von Kund_innen herangezogen [1].

Aufgrund der Verringerung des Aufwandes zur Entwicklung von ML-Modellen wurden diese für wirtschaftliche Anwendungen zugänglich und können nun in der Breite entwickelt und verwendet werden. Durch eine Vielzahl an öffentlichen Datenbanken können Anwender schnell und unkompliziert auf Trainings- und Testdaten für die Entwicklung zurückzugreifen. Bibliotheken wie beispielsweise *Tensorflow*[1], *Keras*[2] oder *ScikitLearn*[3] ermöglichen es, innerhalb von Stunden passenden ML-Modelle zu entwickeln.

Während die Entwicklung von ML-Modellen schon stark vereinfacht wurde, stellt die Überführung der erstellten ML-Modelle in die Produktion, und damit die Lieferung eines realen Mehrwertes für das Unternehmen, eine Hürde für Unternehmen dar. Dies kann darauf zurückgeführt werden, dass ML-Modelle selbst nur ein kleiner Teil innerhalb des Systems sind, welche zur Anwendung in der Produktion notwendig sind. In der Realität ist die, für das Produktivsystem benötigte, technische Infrastruktur komplex [2]. Zudem ist für die Überführung der ML-Modelle in die Produktion ein Team aus unterschiedlichsten Entwickler_innen notwendig. Dies führt dazu, dass ML-Modelle in Unternehmen oftmals lediglich als Leuchtturmprojekte erstellt werden, um den Mehrwert dieser zu zeigen. Eine Überführung der ML-Modelle in die Produktion findet jedoch selten statt. Laut McKinsey schaffen es 36% der Unternehmen ihre ML-Algorithmen über die Pilotphase hinaus zu bringen und ein Gartner Report vermutet, dass lediglich 15% der ML-Projekte in die Produktion überführt werden [3], [4]. Ein Trend, um den dadurch entstehenden Herausforderungen entgegenzuwirken, nennt sich Machine Learning

[1] https://www.tensorflow.org/
[2] https://keras.io/
[3] https://scikit-learn.org/stable/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Operations (*MLOps*). Ähnlich wie das bereits etablierte DevOps zielt MLOps auf eine Verbesserung der Zusammenarbeit und Kommunikation zwischen den einzelnen Parteien innerhalb eines ML-Lebenszyklus ab [5].

Der zuvor beschriebene Umstand trifft auch für die *Thyssenkrupp Presta Steering* zu. Innerhalb des Unternehmens konnte mit Hilfe von Leuchtturmprojekten bewiesen werden, dass Machine Learning einen wertvollen Beitrag liefern kann. Jedoch wurde bis zu diesem Zeitpunkt kein Machine Learning Modell in die Produktion überführt. Dies soll mit Hilfe der vorliegenden Arbeit gelöst werden, indem eine Architektur sowie eine strukturierte Vorgehensweise bei der Überführung von ML-Algorithmen in die Produktion vorgeschlagen werden. Zudem werden geeignete Überwachungsmaßnahmen aufgezeigt, um einen Algorithmus für die Anomalieerkennung in der Produktionsumgebung überwachen zu können.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

1.2 Ziel der Arbeit

Vorhergehenden Arbeiten, welche innerhalb der *Thyssenkrupp Presta Steering* durchgeführt wurden, konnten im Rahmen eines Proof of Concept (PoC) beweisen, dass ML-Modelle einen wertvollen Beitrag für den Unternehmenserfolg liefern. Im vorliegenden Fall wurde eine Fehlerüberwachung eines kritischen Prozessschrittes ermöglicht. Diese Überwachung konnte durch eine Anomalieerkennung erreicht werden, welche Prozesskurven aus ebendiesem Prozess mit Hilfe eines Autoencoders analysiert. Im Rahmen der vorliegenden Arbeit soll eine Überführung des ML-Modells in ein Produktivsystem ermöglicht werden.

Die Probleme, welche durch die Anwendung von ML-Modellen in der Produktion entstehen, können in zwei Bereiche eingeteilt werden. Der erste Bereich entspricht der Überführung des Modells selbst in eine Produktionsumgebung, in der es einen Mehrwert für das Unternehmen bietet. Der zweite Bereich entspricht der Überwachung des ML-Modells, nachdem es in die Produktionsumgebung überführt wurde. Im Rahmen dieser Arbeit wird eine Architektur erarbeitet, wodurch ein Algorithmus zur Anomalieerkennung sowohl in die Produktion überführt als auch überwacht werden kann. Zudem wird ein klarer Prozessablauf erarbeitet, welcher eine strukturierte Überführung des vorliegenden Modells in die Produktion ermöglicht.

Sowohl die Architektur als auch der Prozessablauf sollen anhand des zuvor beschriebenen ML-Modells auf ihre Machbarkeit analysiert werden. Dabei soll die vorgeschlagene Architektur und der Prozessablauf auf der bestehenden Infrastruktur der *Thyssenkrupp Presta Steering* aufbauen und den Bedürfnissen der vorhandenen Produktionsumgebung gerecht werden. Zudem werden Möglichkeiten vorgestellt, um das bestehende ML-Modell in der Produktionsumgebung überwachen zu können.

Aus den Unternehmensanforderungen lassen sich somit zwei Forschungsfragen als Ziele dieser Arbeit ableiten:

Frage 1: Welche Strukturen sind im Produktionsumfeld der Thyssenkrupp Presta Steering notwendig, um einen Algorithmus zur Anomalieerkennung in die Produktion überführen zu können?

Frage 2: Wie lässt sich die Qualität von Algorithmen zur Anomalieerkennung in der Produktion überwachen und bewerten?

1.3 Vorgehensweise

Zur Beantwortung der zuvor definierten Forschungsfragen wird die vorliegende Arbeit in die drei Kapitel theoretische Grundlagen (*vgl. Kapitel 2*), Umsetzungsbeschreibung (*vgl. Kapitel 3*), praktische Implementierung (*vgl. Kapitel 4*) sowie Fazit & Ausblick (*vgl.* Kapitel 5) eingeteilt.

In Kapitel *2* werden die theoretischen Grundlagen für die vorliegende Problemstellung und mögliche Lösungen erläutert, die zum Verständnis der vorliegenden Arbeit notwendig sind. Innerhalb des Kapitels werden in Abschnitt 2.1 die Grundlagen von künstlichen neuronalen Netzen behandelt. Es werden die verschiedenen Arten an künstlichen neuronalen Netzen genannt. Ein Fokus wird auf das Training von künstlichen neuronalen Netzen sowie auf Autoencoder gelegt. Abschnitt 2.2 beschreibt die Grundlagen des Machine Learning Lebenszyklus und geht auf die

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Herausforderungen innerhalb eines Machine Learning Lebenszyklus ein. In Abschnitt 2.3 werden die theoretischen Grundlagen von DevOps behandelt. Es wird ein Überblick über die Entwicklung von DevOps sowie der agilen Arbeitsweise gegeben, um den vorgeschlagenen Prozess besser verstehen zu können. Abschnitt 2.4 erläutert MLOps, welches als Lösung der bestehenden Herausforderungen herangezogen wird. Es werden die theoretischen Grundlagen von MLOps erklärt und es wird eine Bewertungsmethodik eingeführt, um die entwickelte Lösung zu bewerten. Abschnitt 2.5 befasst sich mit der Theorie, die zur Bereitstellung eines ML-Modells in einem gesamten System notwendig ist. Es werden die Begriffe Cloud-Computing, Container, serverlose Funktionen definiert. Des Weiteren wird der Prozess des Verpackens sowie der Zugriff auf einen ML-Service beschrieben. In Abschnitt 2.6 werden die unterschiedlichen Verfahren zum Testen eines ML-Modells bzw. eines ML-Services erläutert. Anschließend werden in Abschnitt 2.7 die theoretischen Grundlagen zur Überwachung eines gesamten ML-Systems beschrieben. Es wird die Drifterkennung, sowie die allgemeine Systemüberwachung vorgestellt und es werden weitere Überwachungsaspekte genannt. Das Kapitel schließt mit der Theorie zum Neutraining von ML-Modellen ab, wobei hier ein Fokus auf die Strategien gelegt wird, um qualitativ hochwertige Ergebnisse erzielen zu können.

Kapitel 3 beinhaltet die Umsetzungsbeschreibung, in der die gewählte Vorgehensweise dokumentiert wird. Das Kapitel startet mit der Analyse des IST-Standes innerhalb des Unternehmens, um die Ausgangssituation dokumentieren zu können. In Abschnitt 3.2 wird die vorgeschlagene Architektur erläutert und es werden sämtliche in der Architektur verwendete Bausteine detailliert beschrieben. Abschnitt 3.3 erklärt den vorgeschlagenen Erstellungsprozess, der für die Überführung eines Modells angewendet werden soll. Anschließend werden in Abschnitt 3.4 die Prozesse der jeweiligen Pipelines erläutert. Das Kapitel schließt mit den vorgeschlagenen Überwachungsmaßnahmen für das in dieser Arbeit verwendete Modell ab.

In Kapitel 4 findet die praktische Implementierung der zuvor beschriebenen Vorschläge statt. Im ersten Abschnitt werden die Schritte aufgeführt, die zur Implementierung der vorgeschlagenen Architektur benötigt werden. Die darauffolgenden Schritte werden nach dem zuvor definierten Machine Learning Lebenszyklus durchlaufen. Hierbei wird ein spezieller Fokus auf die Modellbereitstellung, sowie die Modellveröffentlichung und Überwachung gelegt.

Die Arbeit schließt mit Kapitel 5 ab, indem das Fazit der vorliegenden Arbeit gezogen und ein Ausblick auf weitere zu untersuchende Aspekte gegeben wird.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2 Theoretische Grundlagen

Dieses Kapitel erläutert die theoretischen Grundlagen, welche für die nachfolgenden Kapitel benötigt werden. In einem ersten Schritt werden Grundlagen von künstlichen neuronalen Netzen (*vgl. Abschnitt 2.1*) erläutert. Abschnitt *2.2* definiert den Lebenszyklus von ML-Modellen. Darauf aufbauend werden in Abschnitt 2.3 die theoretischen Grundlagen von DevOps erläutert, um den aktuellen Stand und die damit einhergehenden Herausforderungen innerhalb des Unternehmens zu erläutern. In Abschnitt 2.4 werden die Grundlagen zu MLOps beschrieben, welches als Lösungsprinzip für das vorliegende Problem angewendet wird. Anschließend werden die technischen Grundlagen für eine Bereitstellung des ML-Systems (*vgl. Abschnitt 2.5*), das Testen von ML-Systemen (*vgl. Abschnitt 2.6*) ,die Überwachung von ML-Systemen (vgl. *Abschnitt 2.7*) sowie die unterschiedlichen Lernstrategien (vgl. *Abschnitt 2.8*) aufgezeigt.

2.1 Künstliche neuronale Netze

Dieser Abschnitt behandelt die grundlegenden Konzepte von Künstlichen Neuronalen Netzen (*KNN*) und es werden die Begriffe Machine Learning und Künstliche Neuronale Netze näher erläutert. Hierfür wird in ein Überblick über die Grundlagen von künstlichen neuronalen Netzen gegeben, indem der Aufbau eines KNN, sowie die Aktivierung von Neuronen erläutert wird. Des Weiteren werden die unterschiedlichen Arten des Modelltrainings beschrieben, bevor die Grundlagen eines Autoencoders definiert werden.

2.1.1 Grundlagen von Künstlichen Neuronalen Netzen

Künstliche Neuronale Netze sind Modelle, mit deren Hilfe Vorgänge im zentralen Nervensystem (ZNS) simuliert werden können. Dabei wird das Verhalten einzelner Neuronen und die gemeinsame Interaktion untereinander simuliert [6]. Im Unterschied zu klassischen statistischen Berechnungsverfahren zeichnen sich KNN durch Fehlertoleranz, Generalisierungsfähigkeit, Robustheit und Lernfähigkeit aus, wodurch KNN vor allem in der Mustererkennung eingesetzt werden [7].

Neuronen bilden die Grundbausteine von KNN und sind in unterschiedlichen Schichten (Layer) angeordnet [8]. Jedes Neuron ist mit den Neuronen der vorherigen Schicht verbunden und summiert die Eingangssignale in Abhängigkeit ihrer Relevanz auf [9]. Anschließend wird die Summe mit einer Aktivierungsfunktion verarbeitet. Dies kann mathematisch nach Gleichung (1) definiert werden.

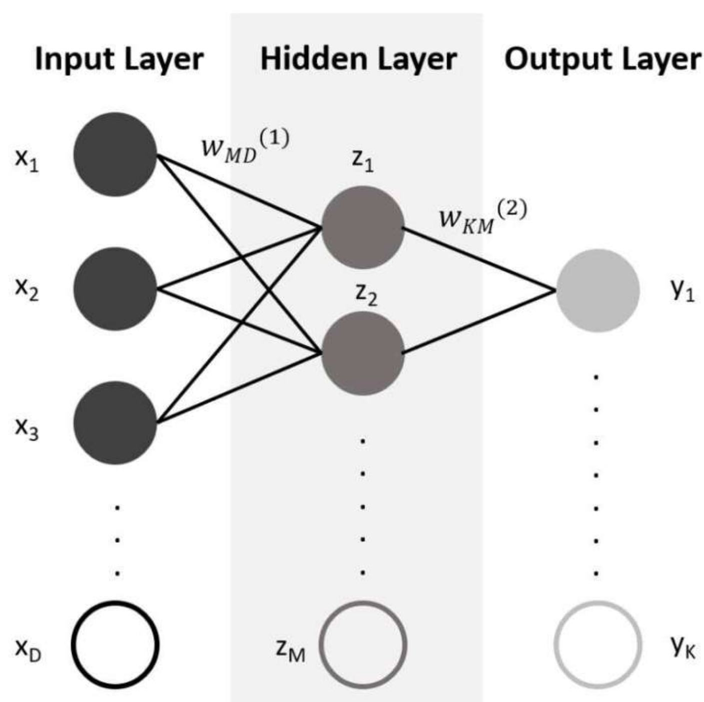
$$y(x) = h\left(\sum_{i=1}^{n} w_i \cdot x_i\right) \tag{1}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Hierbei sind $y(x)$ das Ausgangssignal, h eine zuvor definierte Aktivierungsfunktion, x das Eingangssignal und w die dem Eingangssignal zugeordnete Gewichtung. Ein Lernen der KNN erfolgt durch eine Anpassung der zugeordneten Gewichte und ist durch einen Trainingsdatensatz möglich [10].

Die einzelnen Schichten eines KNN werden in drei Kategorien unterteilt [9]. Die erste Schicht wird Eingangsschicht (*engl. Input Layer*) genannt und nimmt die Eingabewerte auf, die letzte Schicht eines KNNs wird Ausgangsschicht (*engl. Output Layer*) genannt und gibt die Output Variablen des KNNs aus. Dazwischen befindet sich eine innere Schicht (*engl. Hidden Layer*) die ihrerseits wiederum aus mehreren Schichten bestehen kann [6]. In Abbildung 2-1 ist schematisch der Aufbau eines dreischichtigen KNN dargestellt. Auf der linken Seite ist die Eingangsschicht erkennbar, welche drei Eingänge aufweist und bis zu D Eingänge aufweisen kann. Die innere Schicht besteht aus bis zu M Neuronen. Die Ausgangsschicht weist in der vorliegenden Abbildung einen Ausgang auf, sie kann ebenfalls bis zu K Ausgangsneuronen besitzen.

Abbildung 2-1: Aufbau eines künstlichen neuronalen Netzes



Das grundsätzliche Prinzip eines Zusammenschlusses von Neuronen wird in sämtlichen neuronalen Netzwerkarchitekturen angewendet. In den letzten Jahren wurde eine Vielzahl an unterschiedlichen Netzwerkarchitekturen wie Deep Belief Networks (DBN) [11], Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN) und Autoencoder entwickelt, welche jeweils für spezifische Anwendungsfelder eingesetzt werden können [12].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.1.2 Training von künstlichen neuronalen Netzen

Das Training von künstlichen neuronalen Netzen dient dazu, die Gewichtung der jeweiligen Neuronen innerhalb eines KNNs anzupassen. Der Trainingsvorgang selbst besteht aus den Phasen „Vorwärts gerichtete Propagation“ und „Rückwärtsgerichtete Optimierung“ [10]. Die nachfolgende Vorgehensweise ist an Busch [13] angelehnt.

Vorwärts gerichtete Propagation

In dieser Phase erfolgt der Informationsfluss von der Eingangsschicht hin zur Ausgangsschicht, wobei die einzelnen Aktivierungswerte a_j auf Basis der bestehenden Gewichte berechnet werden. Dazu werden in einem ersten Schritt *M* Linearkombinationen der Eingangsvariablen x_i nach Gleichung (2) definiert.

$$a_j = \sum_{i=1}^{D} {w_{ji}}^{(1)} x_i \tag{2}$$

Hierbei symbolisiert die hochgestellte (1) die Berechnung der Aktivierungswerte in der ersten Netzwerkschicht. Die Parameter w_{ji} bezeichnen die Gewichte der einzelnen Neuronen. Anschließend wird jede dieser Aktivierungswerte mit einer Aktivierungsfunktion h nach Gleichung (3) transformiert.

$$z_j = h(a_j) \tag{3}$$

Die daraus generierten Werte werden wiederum mit den Gewichten der nächsten Schicht linear kombiniert, bis zum Schluss die Aktivierungswerte der Ausgangsschicht nach Gleichung (4) erhalten werden können.

$$a_k = \sum_{j=1}^{M} {w_{kj}}^{(1)} z_j \tag{4}$$

Zum Schluss werden die Aktivierungswerte a_l mit einer geeigneten Aktivierungsfunktion σ nach Gleichung (5) transformiert, um die Ausgangswerte des KNNs zu erhalten.

$$y_k = \sigma(a_k) \tag{5}$$

Somit ist das neuronale Netzwerkmodell als nichtlineare Funktion einer Menge von Eingangsvariablen x_i zu einer Menge von Ausgangsvariablen y_k zu sehen, die durch einen Vektor *w* gesteuert wird.

Rückwärtsgerichtete Optimierung

Das Ziel dieser Phase ist das Erlernen der Gradienten der Verlustfunktion in Bezug auf die unterschiedlichen Gewichte, um durch eine Anpassung der Gewichte die Fehlerfunktion E zu minimieren. Da die Gradienten in Rückwärtsrichtung ausgehend von der Ausgangsschicht gelernt werden, wird diese Phase die Rückwärtsgerichtete Optimierung genannt [10]. Die Berech-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

nung der Fehlergradienten erfolgt durch die Kettenregel, um ein lokales Minimum der Fehlerfunktion E zu erhalten. Für die nachfolgende Berechnung wird eine sigmoidale Aktivierungsfunktion der Neuronen nach Gleichung (6) angenommen.

$$h(a) = \frac{1}{1 + e^{-a}} \tag{6}$$

Somit sind die Ausgabewerte eines Neurons zwischen 0 und 1. Des Weiteren wird die Fehlerfunktion eines Datenpunktes x_n im Trainingsdatensatz durch Gleichung (7) definiert.

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (t_k - y_k)^2 \tag{7}$$

Hierbei ist y_k der Ausgabewert des Ausgangsneutrons k und t_k der Zielwert des Ausgangsneutrons k. Die gesamte Fehlerfunktion des Netzwerks ist somit die Summe der einzelnen Terme und ist in Gleichung (8) dargestellt.

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) \tag{8}$$

Nachdem die gesamte Fehlerfunktion bekannt ist, kann diese nach dem Ausgangswert der Neuronen abgeleitet werden. Die Ableitung der Fehlerfunktion ist in Gleichung (9) und die Ableitung der Sigmoidfunktion in Gleichung (10) dargestellt.

$$\frac{\partial E_n}{\partial y_k} = y_k - t_k \tag{9}$$

$$h'(a) = h(a)(1 - h(a)) \tag{10}$$

Mit Hilfe der Kettenregel kann anschließend die Ableitung der Fehlerfunktion in Abhängigkeit der Gewichte berechnet werden (vgl. Gleichung (11) und Gleichung (12)).

$$\frac{\partial E_n}{\partial {w_{kj}}^{(2)}} = \frac{\partial E_n}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k} \cdot \frac{\partial a_k}{\partial {w_{kj}}^{(2)}} = \sum_{k=1}^{K} [(y_k - t_k) \cdot y_k(1 - y_k)] z_j \tag{11}$$

$$\begin{aligned} \frac{\partial E_n}{\partial {w_{ji}}^{(1)}} &= \frac{\partial E_n}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial a_j}{\partial {w_{ji}}^{(1)}} \\ &= \sum_{k=1}^{K} \left[(y_k - t_k) \cdot y_k(1 - y_k) \cdot {w_{kj}}^{(2)}\right] \cdot z_j\left(1 - z_j\right) \cdot x_i \end{aligned} \tag{12}$$

Eine Änderung der Gewichte Δw_{ji} sowie Δw_{kj} kann somit nach Gleichung (13) und Gleichung (14) berechnet werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\Delta w_{ji} = -\eta \frac{\partial E_n}{\partial w_{ji}^{(1)}} \tag{13}$$

$$\Delta w_{kj} = -\eta \frac{\partial E_n}{\partial w_{kj}^{(2)}} \tag{14}$$

Mit η wird die Lernrate festgelegt, welche die Gewichtsänderung zwischen den einzelnen Iterationsschritten definiert. Die Lernrate wird manuell durch den Anwender bestimmt [10].

2.1.3 Arten des Trainings von KNNs

Die gängigsten Herangehensweisen, um KNN zu trainieren, sind durch überwachtes und unüberwachtes Lernen. Neben diesen zwei Ansätzen existiert der Term bestärkendes Lernen (*engl. reinforcement learning*), sowie halb überwachtes Lernen (*engl. semi supervised learning*) [14]. In der vorliegenden Arbeit wird ein unüberwachtes ML-Modell verwendet, weshalb nachfolgend sowohl das unüberwachte Machine Learning als auch das überwachte Machine Learning als Vergleich vorgestellt werden.

Überwachtes maschinelles Lernen

Überwachtes maschinelles Lernen (*engl. supervised learning*) ist eine Form des maschinellen Lernens, bei welcher der Algorithmus auf Basis von vorhandenem Wissen trainiert wird. Dazu benötigt der Algorithmus vorab gelabelte Daten. Das daraus generierte Wissen kann anschließend durch den trainierten Algorithmus auf neue, nicht gelabelte Daten angewendet werden. Während des Trainings werden die gelabelten Ausgangsdaten den jeweilig zugrundeliegenden Eingangsdaten zugeordnet [15]. Zunächst werden analytische Aufgaben auf Basis der Trainingsdaten durchgeführt und anschließend werden kontingente Funktionen gebildet [16]. Die kontingenten Funktionen bestehen selbst aus einer Reihe von Trainingsmodellen [17].

Bei dieser Art des Lernens wird das neuronale Netz auf Basis der Trainingsdaten durch eine Reihe von Rückkopplungsreizen angeregt und die generierten Ausgabewerte werden kontinuierlich mit den Trainingsdaten verglichen. Wurden einige Iterationen durchgeführt, so wird der Fehler zwischen dem realen Ausgangssignal und dem gewünschten Ausgangssignal verglichen und die Netzwerkassoziationsgewichte der einzelnen Neuronen werden normalisiert [18]. Erreicht der Fehlerwert eine akzeptable Größe oder kann keine weitere Verbesserung festgestellt werden, so wird der Lernprozess unterbrochen und das trainierte Modell kann für neue nicht gelabelte Daten verwendet werden. Die überwachten Lernalgorithmen können weiter in Klassifizierungs- und Regressionsalgorithmen unterteilt werden [19].

Unüberwachtes maschinelles Lernen

Unüberwachtes maschinelles Lernen (*engl. unsupervised learning*) beschreibt eine Lernmethode von neuronalen Netzen, bei der nicht gelabelte Daten für das Training des Algorithmus verwendet werden. Das Netzwerk kennzeichnet das Muster der Eingabedaten, welches die numerische Anordnung der gesamten Sammlung von Eingabemustern reproduziert [20]. Es soll mit Hilfe des Trainings eine Funktion zur Beschreibung der Struktur der unbeschrifteten Daten erahnt werden. Unüberwachtes maschinelles Lernen wird auch adaptives oder selbstorgani-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

siertes Lernen genannt, da keine externe Basis existiert, die dem Netzwerk Informationen liefert. Es wird meist eingesetzt, um unstrukturierte Daten in Gruppen zu clustern oder Daten zu assoziieren. Dabei lernt das neuronale Netz, indem ihm Trainingsmuster in der Eingabeschicht gegeben werden. Die Netzwerkassoziationsgewichte werden anschließend mit Hilfe einer Art Rivalität zwischen den einzelnen Knoten der Ausgabeschicht bestimmt, wobei der Knoten mit dem höchsten Wert der erfolgreichste Kandidat ist [18]. Beispielhafte Netzwerkarchitekturen für diese Art des Lernens sind Autoencoder [21] und Deep Belief Networks (DBN) [22].

2.1.4 Trainingsparameter bei KNNs

Batch Größe

Die Batch Größe gibt die Anzahl der Stichproben während des Trainings an, die gleichzeitig verarbeitet werden, bevor das Modell aktualisiert wird. Die Batch Größe muss zwischen eins und der gesamten Trainingsdatenmenge liegen. In der Realität ist die Batch Größe durch den Arbeitsspeicher der Grafikkarte (GPU) bzw. des Prozessors (CPU) begrenzt. Die Batch Größe stellt einen der wichtigsten Hyperparameter dar, der in modernen ML-Systemen angepasst werden kann. Jedoch ist der Einfluss der Batch Größe nicht ganz einfach zu bestimmen. Generell kann aber gesagt werden, dass eine Batch Größe in Höhe des gesamten Trainingsdatensatzes zu einem globalen Optimum führt, jedoch dies zu Lasten der Geschwindigkeit geht. Wird ein Modell mit einer kleinen Batch Größe trainiert, so kann nicht garantiert werden, dass ein globales Optimum erreicht wird, jedoch erreichen diese oftmals schneller gute Ergebnisse, sowie eine bessere Generalisierung auf den Testdatensatz. Dies kann grob auf das Vorhandensein von Rauschen durch das Training kleinerer Batch Größen zurückgeführt werden. Nachdem Neuronale Netzwerke sehr anfällig für eine Überanpassung sind, wird davon ausgegangen, dass die Betrachtung vieler kleiner Stapel eine Überanpassung an den Trainingsdatensatz vermeidet [23].

Trainingsdatenmenge

Die Trainingsdatenmenge ist ebenfalls ein wichtiger Einflussfaktor für die Performance von ML-Modellen. Generell gilt, je größer die Trainingsdatenmenge ist, desto besser ist die Performance eines ML-Modells. Jedoch nimmt die Performance eines ML-Modells im Allgemeinen logarithmisch zu. Das Aufbereiten von Trainingsdaten, welches sowohl die Generierung und Aufbereitung der Datensätze an sich als auch oftmals das Labeln der jeweiligen Datensätze beinhaltet, ist sehr ressourcenaufwändig. Deshalb muss sich die Frage gestellt werden, ob eine vergrößerte Trainingsdatenmenge noch einen sinnvollen Mehrwert für die Performance eines ML-Modells liefert. Laut Shahinfar et al. [24] liegt die optimale Trainingsdatenmenge bei etwa 100 – 500 Datensätzen pro zu identifizierender Klasse. Ab diesem Wendepunkt beginnt der zuvor starke Zuwachs der Performance abzuflachen. Dadurch steht der zusätzliche Aufwand, welcher durch die Vergrößerung der Trainingsdaten einher geht, in keinem sinnvollen Verhältnis zur Verbesserung der Performance.

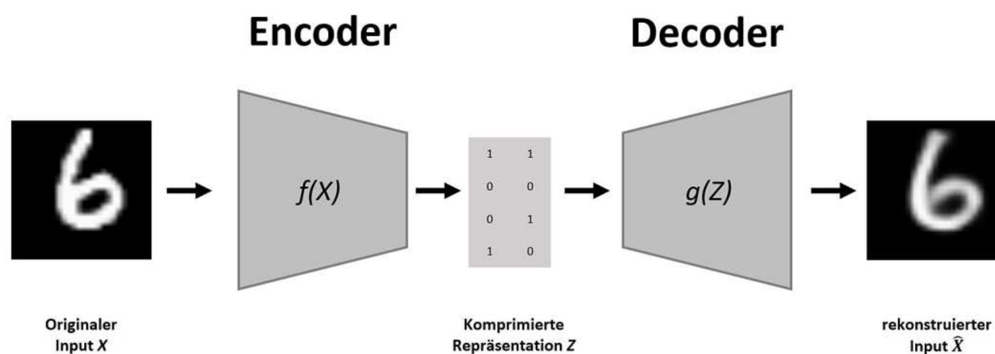
TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.1.5 Autoencoder

Wie in Abschnitt 2.1.1 angeführt, handelt es sich bei einem Autoencoder um eine spezielle Architektur von künstlichen neuronalen Netzen. Ein Autoencoder besteht im Wesentlichen aus zwei Teilen, dem Encoder und dem Decoder [25]. Mit Hilfe eines Encoders wird der erhaltene Input in eine aussagekräftige Repräsentation komprimiert, wodurch der Autoencoder eine Dimensionsreduktionstechnik darstellt. Anschließend wird durch einen Decoder eine Rekonstruktion der komprimierten Repräsentation durchgeführt, so dass der rekonstruierte Input so nah wie möglich zum originalen Input ist [26]. Somit besteht ein wesentlicher Unterschied im Vergleich zu herkömmlichen KNNs darin, dass Autoencoder eine Eingabe rekonstruieren. Im Gegensatz dazu versuchen klassische KNNs Zielwerte mit Hilfe von bestimmten Eingangswerten hervorzusagen [27]. Autoencoder können für unterschiedliche Aufgaben eingesetzt werden. So können Autoencoder in Empfehlungssystemen, für die Klassifikation, für Cluster-Aufgaben, in der Dimensionsreduktion sowie in der Anomalieerkennung eingesetzt werden [26].

Abbildung 2-2 zeigt schematisch den Aufbau eines Autoencoders, welcher hochdimensionale Bilddateien als Input erhält, diese mit Hilfe des Encoders komprimiert und anschließend mit Hilfe des Decoders rekonstruiert.

***Abbildung 2-2**: Schematische Darstellung eines Autoencoders (vgl. Bank et al. [26])*



Der Encoder kann als Funktion $z = f(x)$ und der Decoder als Funktion $r = g(z)$ angesehen werden [25]. Dabei bildet $f(x)$ den Datenpunkt x vom Datenraum in den Merkmalsraum ab und $g(z)$ erzeugt die darauf basierende Rekonstruktion des Datenpunktes x durch eine Abbildung von z vom Merkmalsraum in den Datenraum. Die beiden Funktionen $z = f(x)$ *und* $r = g(z)$ können auch als stochastische Funktionen $p_{Encoder}\ (z \mid x)$ und $p_{Decoder}\ (\hat{x} \mid z)$ ausgedrückt werden [25]. Hierbei ist $\hat{x}$ als Rekonstruktion von x anzusehen. Autoencoder werden meist wie klassische neuronale Netzwerke durch die in Abschnitt 2.1.2 vorgestellte Vorgehensweise durch vorwärts gerichtete Propagation und rückwärtsgerichtete Optimierung trainiert. Als Fehlerfunktion wird oftmals der mittlere absolute Fehler herangezogen, da dieser weniger anfällig gegenüber Ausreißern ist [28]. Dieser ist nach Gleichung definiert.

$$MAE(x) = \frac{1}{N}\sum_{i=1}^{N} \|x_i - \hat{x}_i\| \qquad (15)$$

Dabei entspricht $x = \{x_i\}_{i=1}^{N}$ dem originalen Input und $\hat{x} = \{\hat{x}_i\}_{i=1}^{N}$ der Rekonstruktion von x.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

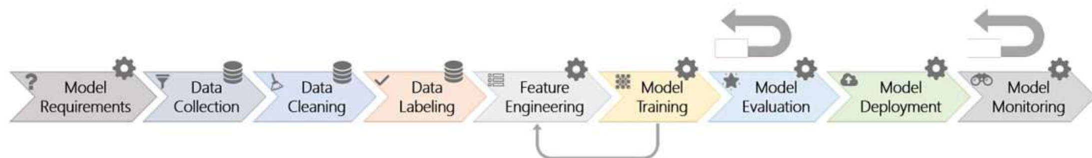
2.2 Der Machine Learning Lebenszyklus

Der folgende Abschnitt gibt eine Übersicht über den gesamten Lebenszyklus von ML-Modellen. Es wird der grundlegende Ablauf zur Erstellung von ML-Modellen erläutert und ein Fokus auf die Voraussetzungen der einzelnen Schritte gelegt. Anschließend werden detailliert die Herausforderungen des Machine Learning Ablaufs aufgezeigt.

2.2.1 Grundlagen

Amershi et al. [29] beschreiben ein Referenzmodell für einen Machine Learning Ablauf, welcher unabhängig des gewählten ML-Modells die einzelnen Schritte innerhalb des Lebenszyklus darstellt. Der in Abbildung 2-3 abgebildete Machine Learning Ablauf besteht aus neun Schritten, von denen einige eine Datenorientierung (*vgl. Datenbank-Symbol*) und andere eine Modellorientierung (*vgl. Zahnrad-Symbol*) aufweisen. Die einzelnen Stationen sind als einfacher linearer Ablauf dargestellt, sie sind jedoch untereinander durch mehrere Feedbackschleifen verbunden. Diese Feedbackschleifen werden durch die grauen Pfeile dargestellt. Nachfolgend werden die einzelnen Schritte nach Amershi et al. [29] beschrieben.

Abbildung 2-3: *vereinfacht dargestellter ML-Ablauf [29]*



1. **Definition der Modellanforderungen**: In dieser Phase werden die Ziele und Anforderungen an das jeweilige ML-Modell definiert. Die Anforderungen und Funktionen werden im Regelfall anhand der Aufgaben innerhalb des jeweiligen Sprints eines DevOps-Zyklus entschieden (*vgl. Abschnitt 2.3.1*). Zudem wird in dieser Phase aufbauend auf den Modellanforderungen die grundsätzliche Modellarchitektur zur Lösung der vorliegenden Problemstellung festgelegt.
2. **Datenerfassung:** Im Rahmen der Datenerfassung werden die unterschiedlichen Datensätze zusammengefasst und ineinander integriert. Die Daten können sowohl unternehmensinterne als auch Open-Source-Daten beinhalten. Der zusammengefasste Datensatz wird für das Training des Modells verwendet.
3. **Datenbereinigung:** Die Datenbereinigung beinhaltet das Entfernen ungenauer oder verrauschter Datenbereiche aus dem zuvor erfassten Datensatz, so dass für das Training ein optimaler Trainingsdatensatz vorliegt.
4. **Datenbeschriftung:** Bei der Datenbeschriftung werden jedem Datensatz eindeutige Bezeichnungen zugewiesen. Dieser Schritt wird für das Erstellen von Trainingsdatensätzen für überwachte ML-Modelle benötigt und kann bei unüberwachten ML-Modellen übersprungen werden. Beispielhaft kann hier das Kennzeichnen von Objekten auf Bildern zum anschließenden Training von Objekterkennungsalgorithmen genannt werden. Die notwendigen Kennzeichnungen können entweder von den Dateningenieuren

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

selbst, von spezifischen Fachexpert_innen oder aber auch von Crowdworker_innen in Online-Crowd-Sourcing-Plattformen erstellt werden. Dieser Schritt gilt als sehr zeit- und kostenintensiv.

5. **Feature Engineering:** Feature Engineering bezieht sich auf alle Aktivitäten, die zur Extraktion und Auswahl der informativen Merkmale innerhalb des zusammengefassten und bereinigten Datensatzes führen. Der Begriff des Feature Engineerings ist sehr breit gefasst und kann in die Begriffe der Feature Transformation, Feature Selektion, Feature Analyse und Feature Evaluation unterteilt werden. Ein Feature beschreibt ein Attribut oder eine Variable, welche einen Aspekt eines individuellen Datenobjektes beschreibt. Der Schritt des Feature Engineerings ist notwendig, um unbewusste Korrelationen zwischen Variablen zu unterbinden, indem nur relevante Variablen für das Training verwendet werden. Bei einigen Modellen (z.B. Convolutional Neural Networks) ist diese Phase weniger explizit ausgeführt und oft mit der nächsten Phase verschmolzen.

6. **Modelltraining:** Während des Modelltrainings werden die ausgewählten Modelle unter Verwendung der zuvor ausgewählten Features mit den zusammengefassten, bereinigten und beschrifteten Daten trainiert.

7. **Modellbewertung:** In der Phase der Modellbewertung bewerten die Datenwissenschaftler_innen das trainierte Modell anhand von Testdatensätzen unter der Verwendung von vordefinierten Metriken. In kritischen Bereichen kann diese Phase auch eine umfangreiche menschliche Bewertung beinhalten.

8. **Bereitstellung des Modells:** Verläuft die Modellbewertung positiv, so kann mit der Bereitstellung des Modells für den Endnutzer begonnen werden. In diesem Schritt wird das Modell verpackt, so dass es anschließend auf dem jeweiligen Zielsystem eingesetzt werden kann.

9. **Überwachung des Modells:** Nach der Bereitstellung wird das Modell kontinuierlich auf Fehler in der Praxis überwacht, so dass eine durchgängig hohe Performance des Modells garantiert werden kann.

2.2.2 Herausforderungen innerhalb des ML-Lebenszyklus

Machine Learning eignet sich sehr gut, um komplexe Vorhersagen in einer datengetriebenen Welt treffen zu können. Jedoch sind Machine Learning Systeme auch gewissen Risiken ausgesetzt, die nicht vernachlässigt werden dürfen. In diesem Abschnitt wird ein Überblick über die unterschiedlichen technischen Herausforderungen und Risiken gegeben, die im Rahmen des zuvor definierten ML-Lebenszyklus auftreten können. Nachfolgend werden diese Herausforderungen basierend auf der Arbeit von Sculley et al. [2] vorgestellt.

Eine Herausforderung, die den gesamten ML-Lebenszyklus umspannt, ist die Verwaltung von unterschiedlichen Phasen des ML-Lebenszyklus in Unternehmen durch unterschiedliche Teams. So ist ein Team aus Datenwissenschaftler_innen für die Erstellung der Modelle (Schritt 2 – Schritt 7) verantwortlich, während ein Team aus Softwareentwickler_innen die Bereitstel-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

lung des Machine Learning Systems übernimmt (Schritt 8). Dies führt dazu, dass über den gesamten ML-Lebenszyklus mehrere Personen und Abteilungen involviert sind. Diese teamübergreifende Komplexität weist bei einer mangelnden Kommunikation ein hohes Risiko des Informationsverlustes auf. Die Übergänge zwischen den einzelnen Schritten des ML-Lebenszyklus und den einzelnen Teams sind zudem oftmals unstrukturiert und nicht eindeutig definiert [2].

Des Weiteren können sich die einzelnen Bestandteile eines ML-Modells wie Code, Hyperparameter und Trainingsdaten während der Schritte 2 bis 7 stark ändern. Aus diesem Grund muss zur Vereinfachung der teamübergreifenden Kommunikation ein spezieller Fokus auf die genaue Dokumentation der Experimente gelegt werden. Eine unstrukturierte Vorgehensweise bei der Durchführung der einzelnen Schritte führt in der Regel zu nicht reproduzierbaren Versuchsergebnissen [29].

In der traditionellen Softwareentwicklung hat sich ein modulares Design des Softwarecodes durchgesetzt, um wartbaren Code für die Produktion zu entwickeln. Dies ist darauf zurückzuführen, dass strikte Abstraktionsgrenzen innerhalb des Codes helfen die logische Konsistenz von Informationen zu Ein- und Ausgängen zu erhalten [30]. Diese strikten Abstraktionsgrenzen können für Machine Learning Systeme nicht angewendet werden, da Machine Learning genau dann zum Einsatz kommt, wenn ein bestimmtes Verhalten nicht durch Code, sondern nur durch die zugrundeliegenden Daten ausgedrückt werden kann [2]. Jede Änderung in den dem Modell zugrundeliegenden Daten hat somit einen direkten Einfluss auf das Modell selbst. Als Beispiel hierfür kann ein ML-Modell mit $x_1 \ldots x_n$ Eingangsvariablen herangezogen werden. Verändert sich die Verteilung der einzelnen Datenpunkte in der Eingangsvariable x_1, so verändern sich bei einem erneuten Training des Modells automatisch die Gewichtung, Wichtigkeit und Verwendung der verbliebenen $n\text{-}1$ Eingänge des Modells. Somit wurde durch die geänderte Verteilung der Datenpunkte ein unterschiedliches ML-Modell im Vergleich zum Vorgängermodell erzeugt. Werden dem Trainingsdatensatz neue Eingänge hinzugefügt oder bestehende Eingänge weggelassen, so wird ebenfalls ein unterschiedliches ML-Modell erzeugt. Dies führt zu einer hohen Anzahl an unterschiedlichen ML-Modellversionen, welche wiederum zu unterschiedlichen Verhaltensweisen der ML-Modelle führen [31]. Durch diese hohe Anzahl an unterschiedlichen Versionen kann es leicht zu einem Kontrollverlust des gesamten ML-Systems kommen, wodurch beispielsweise die Fehlersuche stark verkompliziert wird. In vielen Unternehmen wird der Zusammenhang zwischen Code und den dafür verwendeten Daten innerhalb des ML-Lebenszyklus nicht erhoben, wodurch der Kontext von ML-Modellen über die Zeit verloren geht [32].

In der Produktion bestehen die Input Daten von ML-Modellen oftmals direkt aus Signalen anderer technischer Systeme. Diese Signale weisen eine Instabilität auf, wodurch sie sowohl qualitativ als auch quantitativ über die Zeit variieren können. Ein Grund für die Instabilität ist, dass die Verantwortung über das Input Signal in einem anderen Unternehmensbereich liegt, wie die Verantwortung für das ML-Modell. Wird ohne eine vorherige Kommunikation eine Änderung am Input Signal durchgeführt, so führt dies zu einem aufwändigen Diagnoseprozess, um die Änderungen zu entdecken. Neben Änderungen, welche durch Personen verschuldet werden, verändern sich auch die Daten in der realen Welt. Ein Problem, das daraus resultiert, ist die Verwendung von fixen Schwellwerten für die Ergebnisse von ML-Modellen. Hierbei wird beispielsweise ein Produkt unterhalb eines zuvor definierten Schwellwertes als OK eingestuft, während es

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

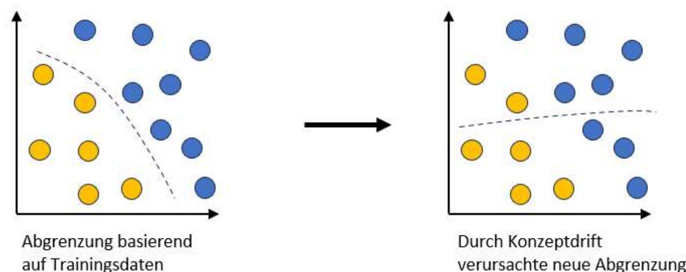
oberhalb des Schwellwerts als nicht OK (NOK) eingestuft wird. Diese Schwellwerte werden oft manuell durch die Dateningenieur_innen gesetzt. Wird ein Modell mit neuen Daten trainiert, so kann dieser manuell gesetzte Schwellwert nicht mehr dem aktuellen Schwellwert entsprechen, wodurch sich der Output des Systems in der Produktion verändert.

Ein weiteres Phänomen stellt der so genannte Konzept Drift dar, welcher auch realer Konzept Drift genannt wird [33]. Dieser tritt auf, wenn sich die Beziehung zwischen den Eingangsdaten und den Ausgangsdaten eines Modells verändert, während die Verteilung der Eingangsdaten konstant bleibt. Formell kann der (reale) Konzept Drift folgendermaßen definiert werden

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{16}$$

Dabei ist p_{t_0} die gemeinsame Verteilung der Eingangsvariable X und der Zielvariablen y zum Zeitpunkt t_0 und p_{t_1} die gemeinsame Verteilung der Eingangsvariable X und der Zielvariablen y zum Zeitpunkt t_1. Die Änderung der gemeinsamen Verteilung kann dabei nicht auf eine Änderung der Verteilung der Eingangsvariablen, sondern auf eine Änderung der Verteilung der Ausgangsvariablen zurückgeführt werden [33]. Diese Art von Drift tritt auf, wenn die Hypothesen, welche dem Modell zugrunde liegen, nicht mehr mit der realen Welt übereinstimmen. Der Konzept Drift ist schematisch in Abbildung 2-4 dargestellt. Auf der linken Seite ist eine Klassifikation von Punkten dargestellt, welche basierend auf den Trainingsdaten erfolgt. Durch die Zeit erfolgte ein Konzept Drift, wodurch sich zwar die Verteilung der Daten nicht veränderte, die Klassifikation der jeweiligen Punkte muss jedoch auf Basis einer anderen Funktion erfolgen, so dass die Realität abgebildet wird.

Abbildung 2-4: schematisch dargestellte Auswirkung des Konzept Drifts

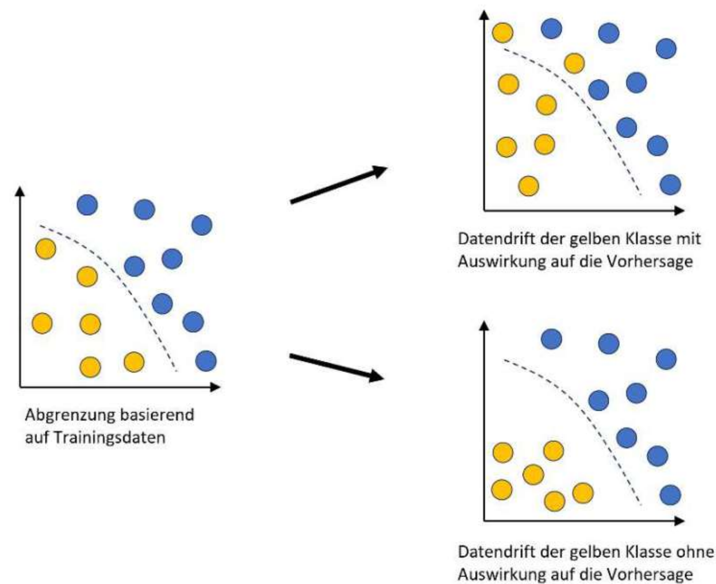


Neben dem Konzept Drift existiert der *Daten Drift*, bei dem sich die Input Daten eines ML-Systems über die Zeit verändern. [34] Diese Art von Drift wird auch virtueller Konzept Drift genannt [33], bei dem sich die Verteilung der Eingangsvariablen $\mathrm{p_{t_i}(X)}$ über die Zeit ändert. Dies kann eine Veränderung von $\mathrm{p_{t_i}(X, y)}$ zur Folge haben, jedoch muss dies nicht dezidiert sein. In diesem Aspekt unterscheidet sich der (reale) Konzept Drift vom Daten Drift. In Abbildung 2-5 ist schematisch der Einfluss des Daten Drifts auf einen Klassifikationsalgorithmus dargestellt. Dieser konnte während des Trainings eine geeignete Entscheidungsgrenze zwischen den gelben Punkten und den blauen Punkten finden. Durch einen Datendrift veränderte sich die Verteilung während der Produktion. Im rechten unteren Bild ist eine veränderte Verteilung dargestellt, bei der die Punkte niedrigere y-Werte aufweisen. In diesem Fall ist die Entscheidungsgrenze immer noch intakt und die Performance des ML-Modells wird nicht beeinflusst. Im rechten oberen Bild jedoch entwickelte sich die Verteilung negativ, indem die

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

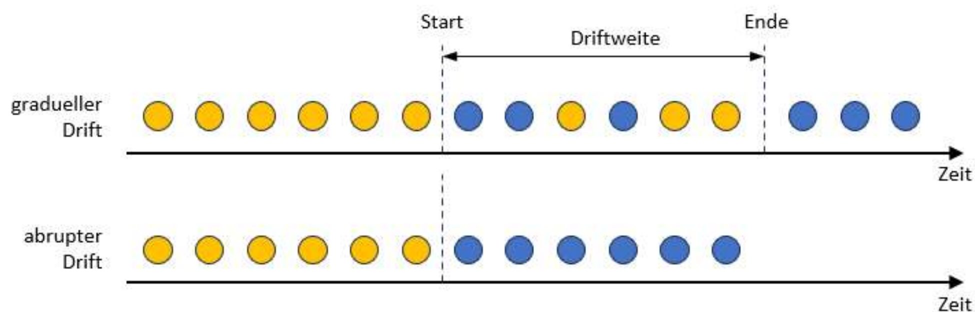
Punkte höhere y-Werte und kleinere x-Werte aufweisen. In diesem Fall verschlechtert sich die Performance des ML-Modells mit hoher Wahrscheinlichkeit [34].

Abbildung 2-5: schematisch dargestellte Auswirkung des Daten Drifts



Beide Arten von Drift können sowohl abrupt als auch graduell erfolgen [35], [36]. Im Fall eines abrupten Drifts erfolgt eine plötzliche Änderung der Eingangsdaten bzw. der zugrundeliegenden Hypothese, während sich bei einem graduellen Drift eine Änderung im Rahmen einer Übergangsphase einstellt. Die Übergangsphase zwischen dem Start und dem Ende des graduellen Drifts wird auch als Driftweite bezeichnet [37]. In Abbildung 2-6 ist eine schematische Darstellung der beiden Arten des Drifts dargestellt. Im oberen Teil ist der graduelle Drift erkennbar, in dem sich die Änderung von Kontext A zu Kontext B nach einer Übergangsphase mit einer bestimmten Driftweite einstellt, während im unteren Teil der abrupte Drift dargestellt ist, der sich ohne Übergangsphase einstellt.

Abbildung 2-6: schematische Darstellung des graduellen und abrupten Drifts



Es ist somit ersichtlich, dass während des ML-Lebenszyklus unterschiedliche Herausforderungen bestehen, welche einerseits die Kommunikation der unterschiedlichsten Teilnehmer über

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

den gesamten Lebenszyklus, der Vielzahl an unterschiedlichen Datenversionen und darauf aufbauenden ML-Modellen während des Trainings, sowie sich ändernde Eingangsdaten in der Produktivumgebung betreffen.

2.3 DevOps

DevOps hat sich mittlerweile zu einer Standard-Herangehensweise in der Softwareentwicklung etabliert, um Software effizient an die Wünsche der Kund_innen anzupassen. Es werden die Grundlagen der agilen Arbeitsweisen aufgezeigt, auf denen DevOps grundsätzlich basiert. Anschließend wird Scrum vorgestellt, bevor die theoretischen Grundlagen von DevOps erklärt werden. Zum Schluss werden die Probleme von DevOps im ML-Lebenszyklus aufgezeigt, um die Notwendigkeit der Implementierung von MLOps zu zeigen.

2.3.1 Grundlagen agiler Arbeitsweisen

Das Konzept der Agilität wurde gegen Ende der 1990er Jahre bekannt, als versucht wurde Lösungen für die bestehenden Probleme in den planorientierten Praktiken zu finden. Agile Projekte zeichnen sich durch eine offene Kommunikation zwischen heterogenen Interessengruppen, emergentes Verhalten in selbstorganisierenden Teams und eine Kultur der Offenheit und des Lernens aus [38]. Highsmith und Cockburn geben in diesem Zusammenhang an, dass die Neuheit an agilen Methoden nicht die Praktiken selbst, sondern die Anerkennung des Menschen als Haupttriebkraft des Projekterfolgs sind [39]. Dies wird mit einem Fokus auf eine hohe Effektivität und Manövrierbarkeit verbunden.

Heute gibt es mehrere etablierte agile Methoden, die von leichtgewichtigen Ansätzen mit starkem Fokus auf die Produktentwicklung wie *eXtreme Programming* (*XP*) und *Scrum* bis hin zu schwereren Methoden wie der projektmanagementorientierten *Dynamic Software Development Method* oder dem architekturorientierten *Scaled Agile Framework*® (SAFe®) reichen [40]. Jede Methodik hat ihre eigene Kultur, Praxis und Sprache. Beispielsweise drücken sowohl XP als auch Scrum ihre Prinzipien in humanistischen Begriffen aus (z. B. Respekt, Mut), während DSDM eher Personen anspricht, die sich in einer reifen Umgebung mit einem stärkeren strukturellen Fokus wohler fühlen [38].

2.3.2 Scrum

Das Scrum-Rahmenwerk besteht aus unterschiedlichen Rollen, Artefakten, Events und Regeln und wurde von Schwaber & Sutherland festgelegt [41]. Innerhalb eines Scrum-Rahmenwerks sind die Rollen eines Scrum-Teams als Produkteigentümer (*engl. product owner*) zur Verantwortung der Produktqualität, Scrum Master zur Verantwortung der Scrum-Umsetzung sowie die Entwickler_innen festgelegt. Das Kernelement von Scrum sind die periodischen *Sprints*, in denen eine zuvor definierte Anzahl an Aufgaben abgearbeitet werden. Die Sprints haben eine fixe Dauer von einem Monat oder kürzer, um eine Regelmäßigkeit zu gewährleisten. Ein Sprint startet direkt nach dem Ende des vorherigen Sprints. Die gesamte Arbeit, die notwendig ist, um die Aufgaben zu erfüllen, wird innerhalb dieser Sprints erledigt. Im *Produkt Backlog* sind sämtliche Aufgaben dokumentiert, die zur Verbesserung eines Produkts notwendig sind. Während der *Sprint Planung* werden die Aufgaben für einen Sprint vom gesamten *Scrum-Team* als

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Sprint-Ziel festgelegt. In täglichen Meetings (*engl. daily scrum*) wird der Fortschritt des Sprint-Ziels überprüft und wenn notwendig angepasst. Zum Ende des Sprints wird ein *Sprint-Review* durchgeführt, indem die Resultate des Sprints vorgestellt und notwendige Anpassungen für die Zukunft durchgeführt werden. Daran anschließend diskutiert das Scrum-Team in einer *Sprint-Retrospektive* Wege, um die Qualität und Effizienz in zukünftigen Sprints zu erhöhen [41].

2.3.3 Grundlagen von DevOps

DevOps ist ein Entwicklungsprinzip, welches als Erweiterung von Scrum für die Softwareentwicklung gesehen werden kann. Diese Erweiterung wurde nötig, da durch die Anwendung von agilen Arbeitsweisen innerhalb der Softwareentwicklung nicht alle Probleme bei der Entwicklung von Softwarecode gelöst werden konnten [42]. Durch DevOps soll die Zusammenarbeit zwischen den Bereichen Entwicklung (Dev) und Bereitstellung (Ops) von Software verbessert werden. Dafür wird die Kommunikation zwischen den beiden Bereichen erleichtert. Zudem erfolgt durch eine hohe Automatisierung eine Reduktion des von Menschen ausgelösten Fehlerrisikos. Des Weiteren wird ein schnelles Feedback von Kund_innen ermöglicht, indem neu geschriebener Code schnell in die Produktivumgebung gebracht wird. Dies führt dazu, dass sowohl Zeit als auch Kosten für die Entwicklungszyklen von Softwarecode minimiert werden können [43], [44]. Das ist darauf zurückzuführen, dass durch die verkürzte Zeit zur Veröffentlichung den Entwickler_innen die Möglichkeit für ein schnelles Scheitern und einem daraus resultierenden schnellen Adaptieren von Lösungen gegeben wird [45]. Aus technischer Sicht wurden diese Verbesserungen vor allem durch die Einführung der beiden Arbeitsprinzipien kontinuierlicher Integration und kontinuierlicher Bereitstellung erreicht. Diese Prinzipien bauen auf einer hohen Automatisierung auf, welche mit Hilfe von CI/CD Pipelines erreicht werden können. Eine CI/CD Pipeline entspricht einer fix definierten Abfolge von Aktionen, die der Reihe nach automatisiert abgearbeitet werden, um so den Code in die nächste Stufe zu überführen.

Kontinuierliche Integration (CI)

Damit Softwareentwickler_innen gemeinsam an einem Projekt arbeiten können, muss der Code in einem gemeinsam zugänglichen Speicher (*engl. repository*) abgelegt werden. Hierbei hat sich GIT als verteilte Versionsverwaltung von Code etabliert. Beispielhaft für grafische User-Interfaces, welche GIT implementiert haben, können GitHub[1] oder Azure Repos[2] neben weiteren Anbietern genannt werden. Bei dieser Art der Versionsverwaltung werden die einzelnen abgeänderten Kopien des bestehenden Codes in so genannten Zweigen (*engl. branches*) abgespeichert. Jeder Zweig stellt eine unterschiedliche Version der Software dar und dient zur Entwicklung von Features, ohne dass der Hauptzweig (*engl. main branch*) dadurch beeinflusst wird. Durch einen *commit* Befehl kann ein Zweig in einen anderen Zweig überführt werden,

[1] https://github.com/
[2] https://azure.microsoft.com/de-de/products/devops/repos

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

wodurch die jeweiligen Änderungen ineinander integriert werden. Kontinuierliche Integration (*engl. Continuous Integration - CI*) ist eine Softwareentwicklungspraktik, bei welcher der entwickelte Code regelmäßig mit dem schon bestehenden Code integriert wird, so dass Integrationsfehler schnell behoben werden können. Die Integration des neu erstellten Codes in den bestehenden Code erfolgt regelmäßig durch jede Entwickler_in und wird bei DevOps automatisiert mit Hilfe von CI-Pipelines durchgeführt. Die CI-Pipeline übernimmt das Testen des Codes durch eigens programmierte Testskripte und führt anschließend einen automatisierten Aufbau der Software durch, um die gesamte Software zudem auf ihre Ausführbarkeit zu testen [46]. Durch die regelmäßige Integration der Änderungen in den bestehenden Code erhalten Entwickler_innen regelmäßiges Feedback und der Code erhält insgesamt kleine inkrementelle Verbesserungen, wodurch insgesamt die Softwarequalität gesteigert werden kann [43].

Kontinuierliche Bereitstellung (CD)

Kontinuierliche Bereitstellung (*engl. Continuous Deployment - CD*) ist ein Entwicklungsprinzip, das darauf abzielt, jederzeit einen produktionsbereiten Code zur Verfügung zu haben. Dies wird erreicht, indem der erfolgreich mit Änderungen integrierte Code, automatisiert in einer produktionsähnlichen Umgebung bereitgestellt wird. Verläuft diese Bereitstellung erfolgreich, so kann die Software anschließend automatisiert in der Produktionsumgebung bereitgestellt werden. In diesem Punkt unterscheidet sich die kontinuierliche Bereitstellung vom Begriff der kontinuierlichen Lieferung (*engl. Continuous Delivery – CDE*), welche keine automatisierte Bereitstellung in die Produktionsumgebung beinhaltet [47]. Die automatisierte Bereitstellung in die produktionsähnliche Umgebung und anschließend in die Produktivumgebung erfolgt durch eine automatische CD-Pipeline, welche die Schritte Kompilieren von Code, Durchführung von Modul- und Akzeptanz Tests, Validierung der Codeabdeckung, Checken der Compliance und das Erstellen von Paketen enthält. Die kontinuierliche Bereitstellung ermöglicht es, dass Änderungen an der Software in kurzen Zyklen veröffentlicht werden und so schnell dem Anwender zur Verfügung gestellt werden. Dadurch erhalten Programmierer regelmäßiges Feedback und können Fehler, die durch die automatischen Tests innerhalb der Pipelines nicht abgedeckt sind, erkennen und beheben [48].

2.3.4 Herausforderungen von DevOps im ML-Lebenszyklus

Ein ML-System ist ebenfalls ein Softwaresystem, wodurch die Vorgehensweisen aus DevOps auch für ML-Systeme eingesetzt werden können. Jedoch gibt es einige Unterschiede zwischen ML-Projekten und traditioneller Software, die beachtet werden müssen [49].

1. **Teamzusammensetzung:** Ein Teil des Teams besteht aus Datenwissenschaftler_innen oder ML-Wissenschaftler_innen, die oftmals wenig Erfahrung im Bereich der Softwareentwicklung und der Entwicklung von Diensten auf Produktionsebenen aufweisen [49].
2. **Entwicklung:** Im Gegensatz zur traditionellen Softwareentwicklung ist Machine Learning eine experimentelle Vorgehensweise, bei der unterschiedliche Funktionen, Parameterkonfigurationen und Algorithmen miteinander kombiniert werden, um ein gutes Modell zu finden. Dazu müssen die Experimente reproduzierbar sein und es muss der

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Ausgang eines Experiments erfasst werden, um eine Vergleichbarkeit zu garantieren [49].

3. **Testen:** Zusätzlich zu den Einheits- und Integrationstests muss bei ML-Systemen eine Datenvalidierung, eine Qualitätsbewertung der trainierten Modelle und eine Modellvalidierung erfolgen [49].
4. **Bereitstellung:** Im Gegensatz zu reinen Code Systemen ist die Bereitstellung des ML-Systems komplexer, da die Bereitstellungspipeline (CD) mehrstufig ausgeführt werden muss, um das Modell automatisch neu zu trainieren [49].
5. **Produktion:** Die Leistung von ML-Systemen kann neben Fehlern im Code auch durch sich ändernde Daten (siehe *Abschnitt 2.2.2*) abnehmen, so dass zusätzliche Überwachungsmaßnahmen ergriffen werden müssen [49].

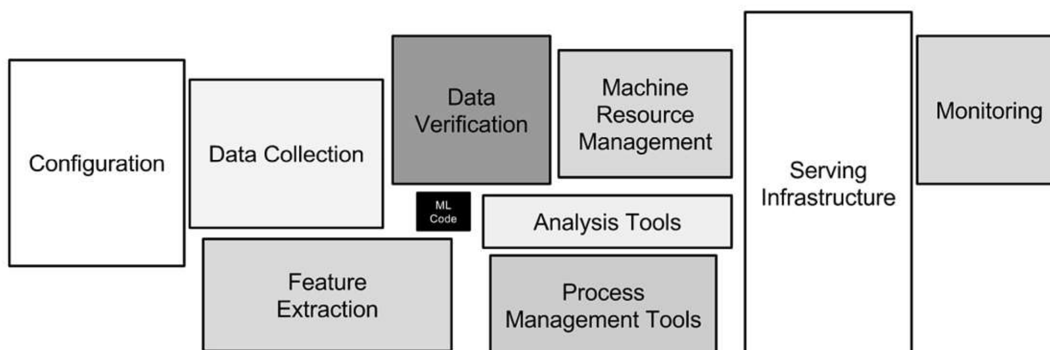
Diese Unterschiede, müssen dabei bei der Anwendung von Machine Learning Systemen berücksichtigt werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4 MLOps

MLOps ist ein Dachbegriff für Methoden, Technologien und Philosophien, die sich mit der Implementierung des beschriebenen ML-Lebenszyklus in die Produktionsumgebung befassen. Während DevOps schon mehrere Jahre in Unternehmen verwendet wird, ist MLOps noch ein neues Konzept, bei dem kein einheitlicher Standard festgelegt ist, sondern auf jede Problemstellung neu angewendet wird [50]. Grundsätzlich bedient sich MLOps den gleichen Techniken wie DevOps. Jedoch werden diese mit Machine Learning spezifischen Themenbereichen ergänzt, so dass die Defizite von DevOps bei der Anwendung von ML-Systemen behoben werden können. Dabei ist nicht nur ein Fokus auf die Entwicklung des Machine Learning Codes zu legen, sondern auch auf die gesamte Infrastruktur, welche für die Bereitstellung des ML-Modells notwendig ist. Wie in Abbildung 2-7 ersichtlich ist, stellt der ML-Code selbst bei der Überführung von Machine Learning Modellen in die Produktion nur einen Bruchteil des gesamten Systems dar. Neben dem ML-Modell wird eine breite Infrastruktur für die Bereitstellung von ML-Systemen in der Produktion benötigt. Beispielhaft dafür ist die Sammlung von Daten, die Überwachung des ML-Systems, sowie die Verifikation der Daten zu nennen [2].

Abbildung 2-7: Bildliche Darstellung der Infrastruktur zur Bereitstellung von ML-Systemen [2]



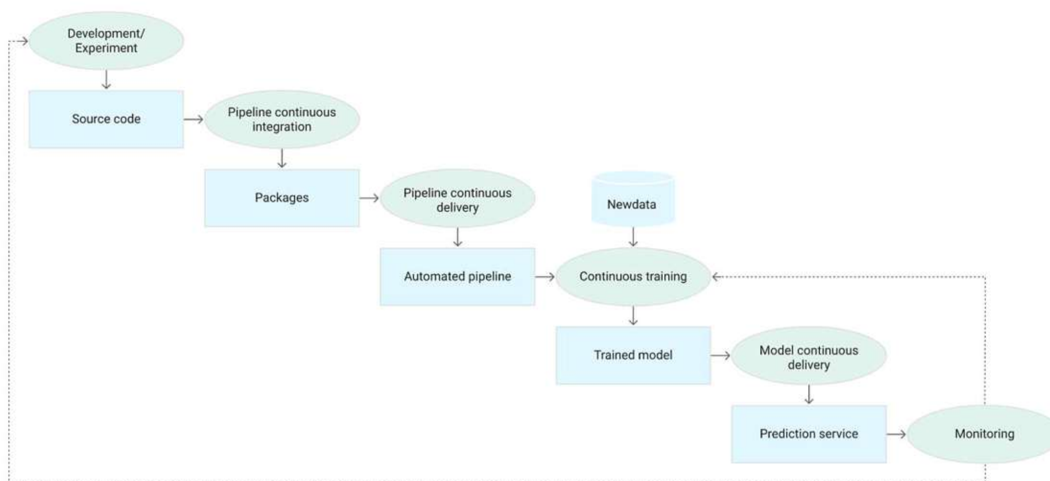
2.4.1 MLOps Prozessmodell

Der Ablauf von MLOps kann in unterschiedliche Stufen unterteilt werden, welche in Abbildung 2-8 dargestellt sind. Die erste Stufe beginnt in der linken oberen Ecke, von wo die unterschiedlichen Stufen nach rechts unten durchlaufen werden. Die erste Stufe ist die Experimentierstufe und beinhaltet die Schritte 2 – 7 des ML-Lebenszyklus (*vgl. Abschnitt 2.2*), in welchen ein ML-Modell erstellt wird. Die Schritte 2 – 7 innerhalb des ML-Lebenszyklus werden auch im Rahmen der Experimentierphase von MLOps manuell von Dateningenieur_innen durchgeführt. Aufbauend auf den entwickelten ML-Modellen wird eine Machine Learning Pipeline entwickelt, die automatisiert die Eingangsdaten verarbeitet, diese dem ML-Modell übergibt und die Ergebnisse ausgibt. Das Produkt dieser Phase ist ein Quellcode der ML-Pipeline, der in ein beliebiges Repository übertragen werden kann. Wird der neue Code in den bestehenden Code integriert,

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

so wird die kontinuierliche Integration gestartet. Dabei wird der Code ausgeführt und anschließend getestet. Die durch die Erstellung des Quellcodes generierten Pakete und Artefakte werden anschließend mit Hilfe der kontinuierlichen Bereitstellung in der jeweiligen Zielumgebung bereitgestellt. Damit kann die kontinuierliche Trainingspipeline mit dem neuen ML-Modell gestartet werden. Das kontinuierliche Training wird entweder durch einen fixen Zeitplan, oder durch definierte Trigger ausgeführt und trainiert das ML-Modell mit neuen Daten. Das trainierte Modell wird anschließend in den Modellspeicher übertragen und steht für die kontinuierliche Bereitstellung des ML-Modells bereit. Bei der kontinuierlichen Bereitstellung wird das ML-Modell als Service bereitgestellt, so dass das trainierte ML-Modell in der Produktivumgebung angewendet werden kann. Der Vorhersagedienst wird in der Produktivumgebung konstant überwacht und dient als Trigger für die Ausführung der Trainingspipeline oder die Entwicklung eines neuen ML-Modells [51].

Abbildung 2-8: Ablauf des MLOps Prozesses [49]



2.4.2 MLOps Reifegradmodell

In den vergangenen Jahren wurde sowohl von Google als auch von Microsoft ein Reifegradmodell für MLOps entwickelt, um den Fortschritt von Unternehmen auf diesem Gebiet besser darstellen zu können [49], [52]. Beide Modelle unterscheiden sich in ihrer Aufteilung. Das Modell von Google richtet sich nach dem Grad der Automatisierung von MLOps und weist drei unterschiedliche Stufen auf, das Modell von Microsoft fokussiert sich auf sämtliche technische Fähigkeiten des Systems und weist fünf unterschiedliche Reifegrade auf. Nachdem das Reifegradmodell von Microsoft detaillierter ist, wird nachfolgend dieses Reifegradmodell vorgestellt, um damit die Implementierung bewerten zu können.

Das Reifegradmodell von Microsoft (*Abbildung 2-9*) besteht aus insgesamt 5 unterschiedlichen Levels (*Level 0 – Level 4*), wobei jede höhere Stufe, die Eigenschaften der unteren Stufen beinhaltet und zusätzlich neue Eigenschaften dazu gewinnt [53].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In *Level 0* wird kein MLOps oder DevOps für Machine Learning Code im Unternehmen angewendet. Die Teams von Datenwissenschaftler_innen und Softwareentwickler_innen arbeiten unabhängig voneinander an der Fertigstellung des ML-Systems. Das Training des Modells erfolgt manuell durch das Team der Datenwissenschaftler_innen und wird anschließend manuell dem Team der Softwareentwickler_innen zur Verfügung gestellt. Auch das Testen des Modells, sowie der bereitgestellten Anwendung erfolgt manuell. Es gibt keine zentralisierte Nachverfolgung der Modell Performance und nach der Bereitstellung wird wenig Feedback über die Leistung der Anwendung an die Teams weitergegeben [53].

Auf *Level 1* greift das Team der Softwareentwickler_innen zur Erstellung des Codes, welcher für den ML-Service benötigt wird, auf DevOps zurück. Dadurch wird die ML-Anwendung automatisiert aufgebaut und getestet, so dass der Aufwand der einzelnen Veröffentlichungen verringert wird. Jedoch ist das Team der Softwareentwickler_innen für jedes neue ML-Modell auf das Team der Dateningenieur_innen angewiesen, welche das ML-Modell manuell an das Softwareentwickler_innen-Team weitergeben. Mit Hilfe einer Datenpipeline können die Dateningenieur_innen automatisch neue Daten sammeln. Des Weiteren wird die Performance der eingesetzten Modelle nicht vorhersagbar dokumentiert und es gibt wenig Feedback über die Funktion des Modells in der Produktion. Dies führt zu schwer nachzuverfolgenden und schwer reproduzierbaren Ergebnissen [53].

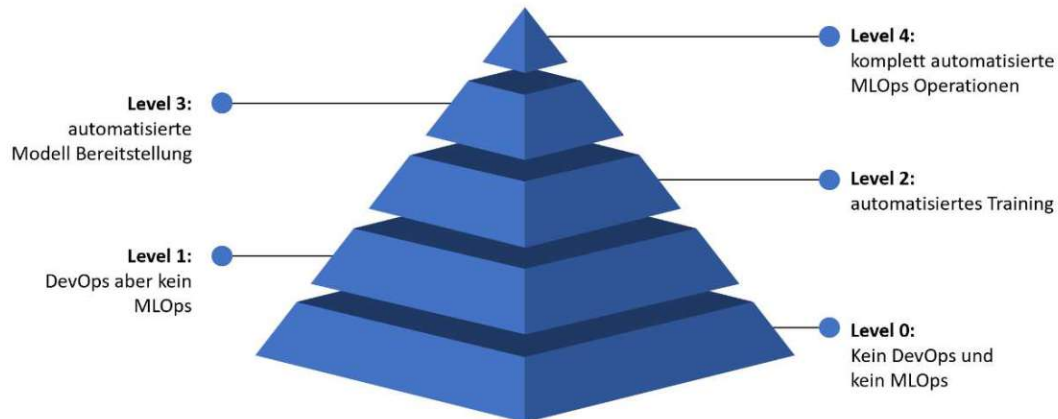
Level 2 zeichnet sich durch eine automatisierte Trainingsumgebung aus, wodurch das ML-Modell automatisch trainiert werden kann. Es kommt zu einer Versionisierung des Trainingscodes und der daraus resultierenden ML-Modelle. Die Resultate der einzelnen Experimente werden zentralisiert abgespeichert und ermöglichen es dadurch die Leistung nachzuverfolgen, sowie die Ergebnisse zu reproduzieren. Die Übergabe des ML-Modells an das Team der Softwareentwickler_innen erfolgt dennoch manuell [53].

Auf *Level 3* arbeiten die Softwareingenieur_innen gemeinsam mit dem Team der Datenwissenschaftler_innen, um das ML-Modell automatisch bereitstellen zu können. Es kann eine vollständige Nachverfolgbarkeit von der Bereitstellung des ML-Modells bis zu den Trainingsdaten garantiert werden. Zudem wird das ML-Modell automatisiert durch eine CD-Pipeline zur Verfügung gestellt, so dass integrierte A/B Tests der Modellleistung sowie automatisierte Tests des gesamten Codes ermöglicht werden. Dies führt zu einer verringerten Abhängigkeit von der Erfahrung der Datenwissenschaftler_innen bei der Implementierung des ML-Modells [53].

Level 4 entspricht der höchsten Stufe von MLOps, in der das ML-Modell in der Produktion überwacht und bei Bedarf automatisch neu trainiert wird. Dies führt zu einem System mit einer fast vollständigen Vermeidung von Ausfallszeiten. Die Metriken zur Überwachung des ML-Systems werden zentralisiert gespeichert und können jederzeit abgerufen werden [53].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 2-9: Darstellung des Reifegradmodells von Microsoft (vgl. [53])



2.4.3 Lösung der Herausforderungen mit MLOps

Damit verdeutlicht werden kann, welche Vorteile MLOps im Vergleich zu DevOps für den Bereich des Machine Learnings bietet, werden nachfolgend die Lösungen von MLOps für die Herausforderungen von DevOps angeführt, welche in Abschnitt 2.3.4 angeführt sind.

MLOps bietet eine durchgängige Dokumentation des Softwarecodes und der verwendeten Modelle bis auf den im Training verwendeten Datensatz und die dabei herrschenden Konditionen an. Dadurch kann einerseits die experimentelle Natur von ML-Modellen abgebildet werden und zudem ermöglicht die durchgängige Dokumentation eine klare Rückverfolgbarkeit der Modelle und eine mögliche Verwendung von vorherigen Versionen in der Produktion. Dadurch können die Modellversionen verwaltet werden und es wird ein regelmäßiges Update der Modelle ermöglicht. Baier et al. [54] geben in ihrer Arbeit an, dass speziell die Fähigkeit der durchgängigen Dokumentation ein wichtiges Merkmal erfolgreicher ML-Systemen in der Produktion ist. Durch die zentral abgespeicherten Modell- und Code-Versionen wird zudem die Zusammenarbeit zwischen den Datenwissenschaftler_innen und den Softwareentwickler_innen vereinfacht. In DevOps wird während des Monitorings ein Fokus auf die Gesundheit der zugrundeliegenden Infrastruktur gelegt [55]. MLOps erweitert diesen Fokus, indem zusätzlich ein konstantes Monitoring der Inputdaten, sowie der Modellvorhersagen implementiert wird und so eine durchgängige Überwachung des Vorhersageservices möglich wird. Zudem wird mit Hilfe einer kontinuierlichen Trainingspipeline der Aufwand für ein erneutes Training des ML-Modells reduziert und ein kontinuierliches Update des ML-Modells ermöglicht [53]. Des Weiteren ermöglicht die Verwendung von CI/CD Pipelines im MLOps Ablauf die Verwendung von automatisierten Tests, wodurch eine hohe Qualität garantiert werden kann [51].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.5 Bereitstellung von ML-Services

Damit ein ML-Modell einen Mehrwert für das Unternehmen liefern kann, muss es im Rahmen eines Vorhersageservices den Kund_innen bereitgestellt werden. Die Bereitstellung von ML-Services kann als Herausforderung angesehen werden, da dies von unterschiedlichsten Faktoren abhängig ist. Clouddienste bieten eine gute Möglichkeit, um eine Standardisierung bei der Bereitstellung von ML-Systemen zu ermöglichen und dadurch Probleme, die in dieser Phase auftreten, zu verringern [54]. Im folgenden Abschnitt werden die theoretischen Grundlagen und Begriffe für die Bereitstellung von ML-Systemen definiert.

2.5.1 Cloud-Computing

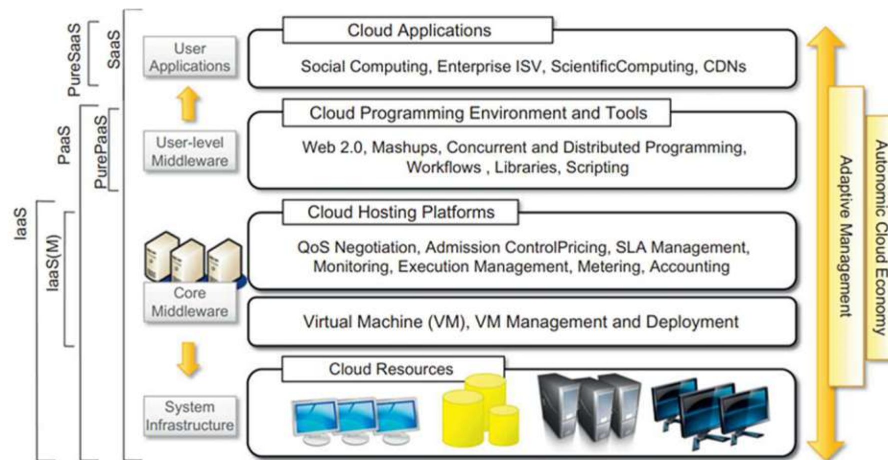
Clouds sind große, verteilte Rechenzentren (*engl. distributed systems*), die Ihre Services dritten Parteien bei Bedarf zur Verfügung stellen. Hierfür sind vor allem zwei Prinzipien wichtig, die Verteilte Systeme charakterisieren. So besteht ein verteiltes System aus einer Vielzahl an unabhängigen Komponenten, die aus Nutzersicht als eine einzige Einrichtung angesehen werden [56]. Das *National Institute of Standards and Technology* definiert Cloud Computing nach den folgenden fünf Merkmalen [57]:

- Die Leistungen von Cloud Services können von Nutzern einseitig und ohne menschliche Interaktion durch Selbstbedienung bereitgestellt werden.
- Die Leistungen sind mit Standardmechanismen nutzbar, sodass diese einer großen Gruppe zur Verfügung stehen.
- Die Ressourcen eines Cloud-Anbieters werden zusammengefasst, um die Nutzenden bedarfsgerecht bedienen zu können. Hierbei haben die Nutzenden keinen Einfluss darüber, welche Ressource für die Erbringung der Leistung genutzt wird.
- Die Leistung wird elastisch nach Bedarf der Nutzenden in der Höhe angepasst.
- Die erbrachte Leistung kann für die Nutzenden transparent anhand von messbaren Zahlen dokumentiert werden.

Eine Cloud-Computing Architektur kann in unterschiedliche Schichten eingeteilt werden und ist in *Abbildung 2-10* ersichtlich [58]. Die Basis für das Cloud-Computing stellt die System Infrastruktur dar, in der die für die Bereitstellung der Cloud Services notwendige Rechenleistung sowie die benötigten Datenbanken organisiert werden. Die Infrastruktur wird meist in Rechenzentren konzentriert, kann jedoch auch aus einem Netzwerk an individuellen Rechnern bestehen. Die physische Infrastruktur wird durch die Core Middleware verwaltet. Deren Ziel ist es, eine geeignete Laufzeitumgebung für die Anwendungen zur Verfügung zu stellen und die physischen Ressourcen optimal zu nutzen. Zu diesem Zweck werden Virtualisierungstechnologien, wie beispielsweise Virtuelle Maschinen, angewendet. Dadurch können Hardwareressourcen wie CPU oder Speicher partitioniert werden. Diese werden meist mit Speicher- und Netzwerksvirtualisierungstechnologien kombiniert, um eine vollständige Kontrolle der Infrastruktur zu ermöglichen.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abbildung 2-10: Architektur von Cloud Computing [58]



Die Kombination von Cloud-Hosting-Plattformen und -Ressourcen wird im Allgemeinen als *Infrastructure-as-a-Service-Lösung* (*IaaS*) bezeichnet. Diese Art von Services kann wiederum in zwei Kategorien eingeteilt werden. Einige Services bieten sowohl die Verwaltungsebene als auch die physische Infrastruktur an, während andere rein die Verwaltungsebne (IaaS(M)) anbieten. Hierbei wird die Verwaltungsebene häufig in andere IaaS-Lösungen integriert, die eine physische Infrastruktur bereitstellen. IaaS-Lösungen eignen sich für die Gestaltung der Systeminfrastruktur, jedoch bieten sie nur einen begrenzten Mehrwert für die Erstellung von Anwendungen. Diese Dienste können durch Cloud-Programmierumgebungen und Tools bereitgestellt werden, welche eine neue Schicht innerhalb der Architektur darstellen. Dies wird als Platform-as-a-Service (PaaS) bezeichnet, da die Nutzenden eine gesamte Entwicklungsplattform und nicht nur die reine Infrastruktur anwenden. PaaS-Lösungen umfassen in der Regel auch die Infrastruktur, die als Teil des Dienstes für die Nutzer gebündelt wird. Im Falle von Pure PaaS wird nur die Middleware auf Benutzerebene angeboten, die durch eine virtuelle oder physische Infrastruktur ergänzt werden muss. Die oberste Schicht innerhalb der Cloud-Computing Architektur umfasst Dienste, die auf der Anwendungsebene bereitgestellt werden. Diese werden als Software-as-a-Service (SaaS) bezeichnet. Hierbei handelt es sich meist um webbasierte Anwendungen, die auf weiteren Cloudservices basieren, um den Nutzenden die Dienste anbieten zu können. Dadurch können unabhängige Softwareanbieter ihre Dienste mit Hilfe von IaaS und SaaS-Lösungen über das Internet den Kund_innen bereitstellen [58].

2.5.2 Container

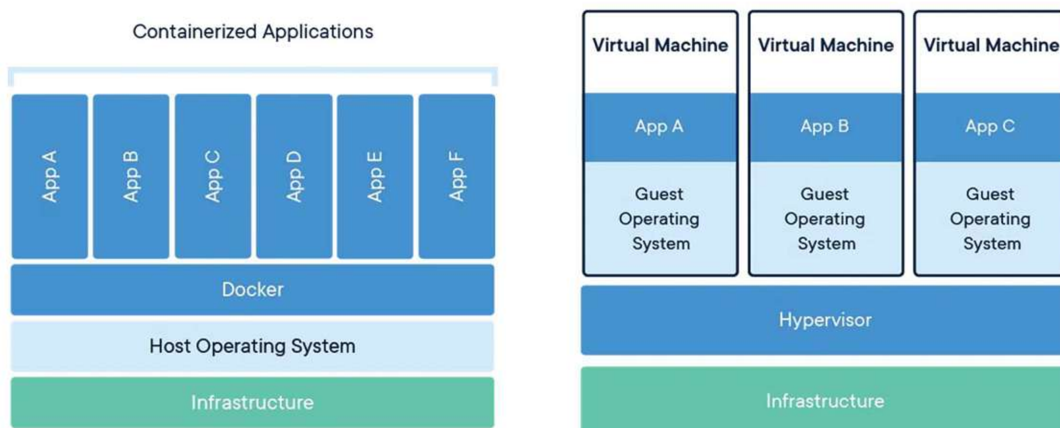
Eine weit verbreitete Bereitstellungsart von ML-Systemen ist das Verpacken von ML-Systemen in Containern und einer anschließenden Bereitstellung des Containers mit Hilfe eines Clouddienstes [59]. Ein Container ist ein Prozess oder eine Reihe an Prozessen, die über eine Gruppierung von Ressourcen verfügen. Diese können speziell den Prozessen zugewiesen werden. Dies geschieht einerseits mit Hilfe von so genannten Namensräume (*engl. Namespacing*), wodurch die Prozesse auf einen spezifischen Ressourcenbereich isoliert und Hardware für den Prozess limitiert werden können. Zusätzlich werden Kontrollgruppen (*engl. control groups*) verwendet, um weitere Ressourcen wie beispielsweise Prozessoren (CPU) oder Speicher für einen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Prozess zu limitieren [60]. Der Container kann beispielsweise einen ML-Service als laufenden Prozess beinhalten. Dieser Prozess sendet System Calls an den Kernel, welcher die einkommenden Calls analysiert und anschließend an die dem Container zugewiesenen Ressourcen weiterleitet.

Die für die Ausführung eines Programms notwendigen Abhängigkeiten und Konfigurationen werden in einer einzigen Binärdatei verpackt, die als Containerimage bezeichnet wird [61]. Ein Container ist eine Instanz eines Images, die das Programm ausführt. Damit kann ein Container in unterschiedlichsten Umgebungen wie beispielsweise auf einem privaten Laptop oder auch in einer Cloudumgebung erstellt werden [62]. Container kapseln die gesamte Softwareumgebung in einem einzelnen Paket und isolieren dadurch das ML-System und seine Abhängigkeiten von der Infrastruktur. Container bieten somit eine Übertragbarkeit in unterschiedliche Systeme und garantieren eine Konsistenz in allen Umgebungen. Im Vergleich zu virtuellen Maschinen (VM) benötigen Container einen Bruchteil des Arbeitsspeichers, welcher von virtuellen Maschinen benötigt wird. Dies ist darauf zurückzuführen, dass Container auf Betriebssystemebene virtualisieren, während virtuelle Maschinen auf Hardwareebene virtualisieren [62]. In *Abbildung 2-11* ist der Vergleich der Architektur zwischen einem Container auf der linken Seite und einer virtuellen Maschine auf der rechten Seite dargestellt.

Abbildung 2-11: Vergleich der Architektur eines Containers und einer virtuellen Maschine [63]



Um Container in einer Umgebung bereitzustellen, wird zu einem großen Teil auf *Docker*[1] zurückgegriffen [64]. Um *Docker* zu verwenden, wird ein *Docker Client* benötigt, um auf den *Docker Server* zuzugreifen. Dieser ist verantwortlich für die Erstellung von Containern und Images,

[1] https://www.docker.com/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sowie der Wartung von Containern [60]. Möchte beispielsweise ein Container mit Hilfe von Docker erstellt werden, so kann mit Hilfe des Docker Clients ein Befehl dazu ausgegeben werden. Der *Docker Client* übergibt den Befehl an den *Docker Server*. Daraufhin überprüft der *Docker Server*, ob das dafür passende Image auf dem lokalen Image Cache abliegt. Liegt das Image nicht ab, so wird dieses vom *Docker Hub* heruntergeladen. Anschließend verwendet der *Docker Server* das lokale Image, um eine Instanz des *Docker Containers* zu erstellen. Anschließend kann der *Docker Container* gestartet werden.

Werden mehrere Container gleichzeitig ausgeführt, so wird eine Containerorchestrierung wie beispielsweise *Kubernetes*[1] oder *Apache Mesos*[2] benötigt, um die Last gleichzeitig auf mehrere Container verteilen zu können. Ein Großteil der Infrastruktur sowie der Werkzeuge für ML-Services ist auf *Kubernetes* aufgebaut. Dies kann jedoch eine Schwierigkeit darstellen, da *Kubernetes*-Cluster teuer in der Bereitstellung sind [65].

2.5.3 Serverlose Funktionen

Zur Unterteilung einer Softwareanwendung wurden in den vergangenen Jahren unterschiedliche Lösungen entwickelt. Waren früher Softwareanwendungen als monolithischer Code ausgeführt, welcher starke Probleme bei der Entwicklung und auch Wartung mit sich brachte, so wurde dies mit Hilfe von Microservices weiterentwickelt. Ein Microservices stellt einen einzelnen Service der gesamten Anwendung dar, so dass eine Anwendung aus mehreren Microservices besteht. Durch diese klaren Abgrenzungen lässt sich eine Softwareanwendung einfacher entwickeln und anschließend warten. Eine Weiterentwicklung von Microservices stellen serverlose Funktionen da, bei denen jede einzelne Funktion als komplett isolierter Code erstellt wird. Eine serverlose Funktion ist dabei ein kleiner Codeblock, deren einzelner Zweck eine Reaktion auf ein Ereignis darstellt. Microservices hingegen reagieren auf Anforderungen häufig aus einer Schnittstelle. Sowohl Microservices als auch serverlose Funktionen werden in Containerumgebungen ausgeführt, wodurch eine hohe Flexibilität garantiert werden kann. Um serverlose Funktionen bereitzustellen können Frameworks wie beispielsweise *Apache Open Whisk*[3] oder *Knative*[4] verwendet werden, welche beide auf Kubernetes aufbauen [64].

2.5.4 Verpacken von ML-Services

ML-Frameworks, welche für die Erstellung eines ML-Modells benötigt werden, sind nicht immer für eine Produktionsumgebung ausgelegt, sondern haben meist einen unterschiedlichen Fokus. Unterschiedliche ML-Frameworks sind für unterschiedliche Anwendungsfelder während eines Modellerstellungsprozesses ausgelegt. Möchte eine Entwickler_in ein ML-Modell mit Hilfe von

[1] https://kubernetes.io
[2] https://mesos.apache.org/
[3] https://openwhisk.apache.org
[4] https://knative.dev

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Keras entwickeln und anschließend mit Hilfe von *PyTorch* bereitstellen, so ist ein Transport des Modells von einem Framework in das andere Framework notwendig [66]. Kolltveit et al. [64] notieren, dass eine Inkonsistenz von Frameworks eine Herausforderung bei der Bereitstellung von ML-Services ist. Zudem empfiehlt es sich ein ML-Modell zu verpacken, um die Experimentierphase besser von der Produktion zu trennen. Hierbei kann das Verpacken eines ML-Modells als serialisieren (*engl. serialize*) bezeichnet werden, bei dem die Konfiguration eines ML-Modells in einem allgemein lesbaren Dateiformat gespeichert wird [67]. Die Konfiguration eines Modells gibt an, welche und wie viele Schichten das ML-Modell enthält und wie diese Schichten miteinander verbunden sind. Mögliche Datenformate zur Verpackung eines ML-Modells sind *Open Neural Network Exchange*[1] (ONNX), *Hierarchical Data Format*[2] (HDF), *Pickle*[3] oder *Protobuf*[4] (PB). In Tabelle 2-1 ist die Kompatibilität der einzelnen Dateiformate (*vgl. Spalten*) mit den drei Frameworks *Tensorflow*, *Pytorch* sowie *SciKitLearn* basierend auf Schlittler [68] angegeben. Es kann gesehen werden, dass die Dateiformate ONNX und Pickle sämtliche Frameworks unterstützen, während die Dateiformate H5 und PB Tensorflow sowie Pytorch unterstützen.

Tabelle 2-1: Kompatibilität der Dateiformate (Spalten) mit den jeweiligen Frameworks (Zeilen) (vgl. [68])

	ONNX	**H5**	**Pickle**	**PB**
Tensorflow	ja	ja	ja	ja
Pytorch	ja	ja	ja	ja
SciKitLearn	ja	nein	ja	nein

ONNX ist der neueste Standard für den Transport von Neuronalen Netzwerken und wurde im Jahr 2017 durch *Microsoft* und *Facebook* veröffentlicht [66]. Wie der Name bereits erahnen lässt, ist ONNX ein offenes Format, welches von unterschiedlichen Unternehmen weiterentwickelt wird. ONNX kann als Programmiersprache angesehen werden, die auf mathematische Funktionen spezialisiert ist. Die einzelnen Schritte können auch als Diagramm dargestellt werden, weshalb mit ONNX implementierte Modelle oft auch als ONNX-Diagramm bezeichnet werden [69].

Das Datenformat H5 wird standardmäßig von *Keras* angewendet, bei dem die einzelnen Gewichte als HDF5 Format gespeichert werden. Die Modellstruktur selbst kann entweder als

[1] https://onnx.ai/
[2] https://portal.hdfgroup.org/display/support
[3] https://docs.python.org/3/library/pickle.html
[4] https://protobuf.dev/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

JSON-Datei, oder als YAML-Datei abgespeichert werden[60]. Das hierarchische Datenformat (HDF) wurde ursprünglich vom *National Center for Supercomputing Applications* (*NCSA*) für den Transport von großen wissenschaftlichen Datenmengen verwendet [70]. Hierbei können Tabellen und andere Daten in derselben Datei in einer beliebigen Verzeichnisstruktur abgelegt werden.

Pickle ist eine Standard *Python* Bibliothek zur Serialisierung bzw. Deserialisierung von Objekten. Hierbei werden Objekte in einem Byte-Format in einer Datei gespeichert. Bei einer späteren Verwendung kann das gespeicherte ML-Modell wieder in eine Objekthierarchie umgewandelt werden. Beide Vorgänge können auch als „pickling" bzw. „unpickling" bezeichnet werden [71].

Eine weitere Form der Speicherung stellen *Googles* Protobuf (PB) Dateien dar. Protocol Buffers sind sprachenneutrale, sowie plattformneutrale Mechanismen, um strukturierte Daten zu serialisieren [72]. Die Datei enthält sowohl die Definition des Graphen als auch die einzelnen Gewichte eines ML-Modells [60].

2.5.5 Zugriff auf ML-Services

Der Zugriff auf in Container bereitgestellte ML-Services erfolgt typischerweise mit Hilfe von REST (representation state transfer) oder durch RPC (remote procedure call) Interfaces.

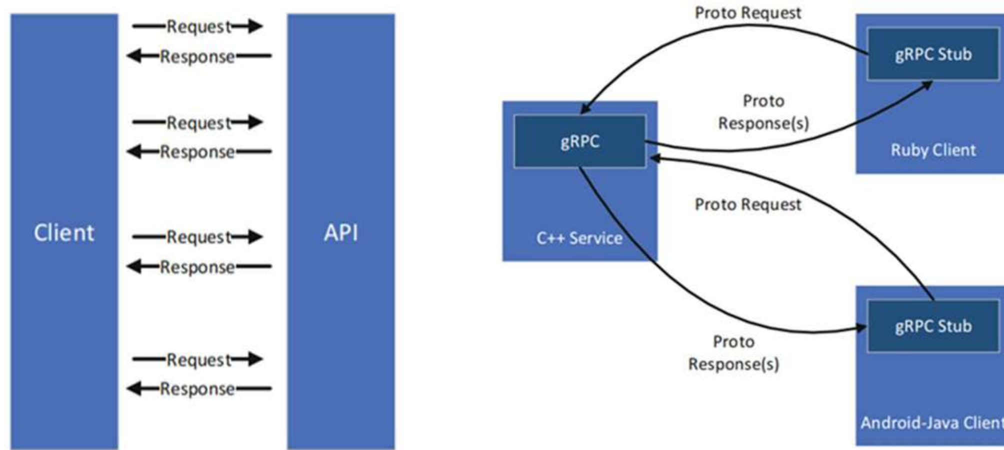
Bei REST handelt es sich im Wesentlichen um eine Anfrage/Antwort Beziehung zwischen einem Client und einem Server mittels http (Hypertext Transfer Protocol) und ist in Abbildung 2-12 links dargestellt. Es wird von einer Seite eine Anfrage angefordert, die andere Seite bearbeitet die Anfrage und sendet das Ergebnis zurück. Aus diesem Grund wird dieser Austausch auch Konversation bezeichnet. REST-Architekturen sind in der Regel für hochgradig modulare Mikrolösungen konzipiert [60].

Ein Beispiel für RPC ist gRPC, eine von *Google* entwickelte Remote Procedure Call-Platform, die hohe Leistung, niedrige Latenzzeiten und hohen Durchsatz bietet [60]. Die Kommunikation bei gRPC ist schematisch in Abbildung 2-12 rechts dargestellt. Ein gRPC-Kanal bietet eine Verbindung eines Clients zu einem gRPC-Server an einem bestimmten Port. Der Client ruft eine Methode auf einem gRPC – Stub auf, als ob es ein lokales Objekt wäre. Daraufhin wird der Server über den Request informiert. Hierbei kann im Gegensatz zu REST auch ein bidirektionales Streaming durchgeführt werden. Zudem ist gRPC auf Grund von serialisiertem PB zehn Mal schneller als REST [60].

Werden mehrere ML-Services bereitgestellt, so empfiehlt es sich einen konsistenten Standard wie beispielsweise OpenAPI zu verwenden, um die Systemintegration zu erleichtern [73].

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

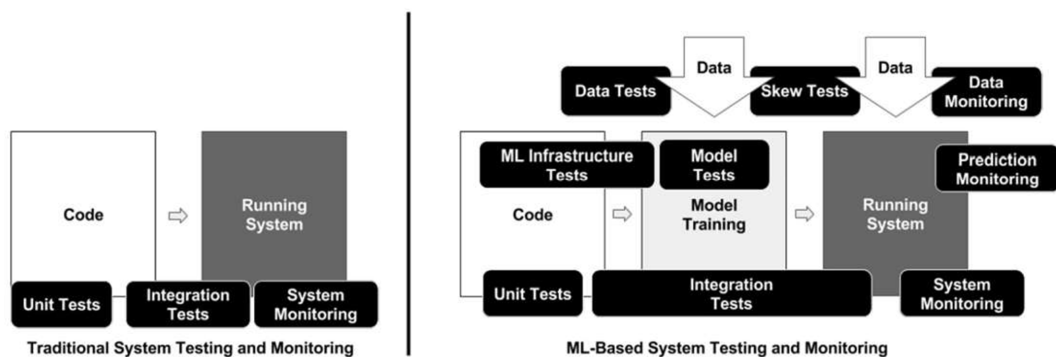
Abbildung 2-12: Schematische Darstellung der Kommunikation von REST API (links) und gRPC (rechts) [60]



2.6 Testen von ML-Systemen

Um ein zuverlässiges ML-System zu generieren, sollten im Rahmen der Teststrategie alle Fehler innerhalb des ML-Systems gefunden werden. Während viele dieser Tests in der Theorie als sinnvoll erachtet werden, geben Breck et al. [74] an, dass innerhalb von *Google* nur 20% der ML-Teams ihren ML-Code in der Produktion testen. Dabei ist das Testen von ML-Systemen um einiges komplexer im Vergleich zu traditionellen Code Systemen, wie in Abbildung 2-13 ersichtlich ist. Dies ist vor allem darauf zurückzuführen, dass das Verhalten eines ML-Systems stark von den zugrundeliegenden Daten abhängt, wodurch das Verhalten nicht direkt vorhergesagt werden kann. Dadurch müssen sowohl Code als auch Trainingsdaten getestet werden. Das trainierte Modell benötigt dieselben Produktionspraktiken wie kompilierter Code wie beispielsweise Debugging, Rollbacks auf vorherige Versionen und eine Überwachung des Systems in der Produktion [74].

Abbildung 2-13: Vergleich der Testkomplexität zwischen traditionelle Code Systemen und ML-Systemen [74]



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Breck et al. [74] geben in ihrer Arbeit einen guten Überblick über das Testen von ML-Systemen für die Produktion. Die Arbeit ist jedoch vor allem auf Überwachte ML-Modelle ausgelegt, weshalb nachfolgend lediglich die Tests angeführt werden, die auch für unüberwachte ML-Modelle sinnvoll sind.

2.6.1 Daten Tests

Es ist hilfreich die Erwartungen an die für das Training verwendeten Daten als Schema festzuhalten, so dass diese automatisch überprüft werden können. Dadurch können grobe Fehler innerhalb der Daten ausgeschlossen werden. Dies kann beispielsweise die Annahme sein, dass eine ausgewachsene Person zwischen 1.40 m und 2.10 m groß ist, oder in dem vorliegenden Fall der Produktionsdaten, dass diese nur innerhalb eines gewissen Fensters liegen. Werden für das ML-System überwachte ML-Modelle verwendet, so muss dieser Aspekt noch stärker beleuchtet werden. Für das vorliegende unüberwachte ML-System ist diese Art von Daten Tests ausreichend. [74]

2.6.2 Modul Tests

Zudem müssen die einzelnen Module eines ML-Systems getestet werden, bevor diese in das ML-System integriert werden. Dies betrifft einerseits den Code zur Bearbeitung der Input Daten, als auch der Code zur Generierung des ML-Modells selbst [74].

Der Code zur Bearbeitung der Input Daten mag zwar oftmals simpel sein, jedoch ist speziell dieser Code für das richtige Verhalten eines ML-Modells ausschlaggebend. Zudem können Fehler innerhalb der Daten sehr schwer gefunden werden, speziell wenn der gleiche Fehler sowohl innerhalb der Test- und Trainingsdaten auftritt. Aus diesem Grund muss ein besonderes Augenmerk auf das Testen dieses Code Abschnittes gelegt werden [74].

Des Weiteren muss auch der Code zur Spezifikation des ML-Modells getestet werden, obwohl dies meist als Konfigurationsdateien angesehen werden. Doch auch diese Dateien können Fehler enthalten. Um ML-Spezifikationen zu testen können Testdaten erzeugt werden und das Modell kann einen Schritt darauf trainiert werden, wodurch schon die meisten Fehler auf Grund von Bibliotheken erkannt werden können. Zudem kann das ML-Modell nur eine kurze Zeit auf die Daten trainiert werden und es kann anschließend beispielsweise auf ein Abnehmen der Fehlerfunktion getestet werden. Wenn Modelle getestet werden, dann sollte besonders darauf geachtet werden, keine „goldenen Tests“ zu verwenden, bei denen ein Modell teilweise trainiert wird und mit den Ergebnissen des zuvor erstellten Modells verglichen wird. Diese Art von Tests sind schwer zu pflegen und geben wenig Aufschluss darüber, warum ein Unterschied eintritt [74].

2.6.3 Integration Tests

Eine vollständige ML-Pipeline besteht in der Regel aus der Zusammenstellung von Trainingsdaten, der Feature-Generierung, dem ML-Modelltraining, der ML-Modellüberprüfung und der Bereitstellung des trainierten und überprüften Modells in einem Produktionssystem. Auch wenn sich ein einzelnes Entwicklungsteam auf einen kleinen Teil des Prozesses konzentriert und diese mit Hilfe von Modul Tests überprüft, können in jeder Phase Fehler auftreten, die sich auf

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

die nachfolgenden Phasen auswirken. Aus diesem Grund muss ein vollautomatischer Test ausgeführt werden, der regelmäßig die gesamte Pipeline überprüft, um sicherzustellen, dass Daten und Code jede Phase erfolgreich durchlaufen können und dass das resultierende Modell gut funktioniert. Der Integrationstest sollte sowohl kontinuierlich als auch bei neuen Versionen von Modellen oder Servern laufen, damit Probleme erkannt werden, bevor diese die Produktionsumgebung erreichen. Da eine Ausführung der gesamten ML-Pipeline oftmals sehr lange dauert, empfiehlt es sich die Integrationstests mit einer Teilmenge von Trainingsdaten oder einem einfacheren Modell durchzuführen, so dass den Entwickler_innen schnelleres Feedback gegeben werden kann [74].

Bevor das Modell anschließend in die Produktionsumgebung gelangt, wird ein A/B Test durchgeführt. Dies ist notwendig, da Offline-Tests, auch wenn sie noch so umfangreich sind, nicht garantieren können, dass das Modell auch in der Produktionsumgebung gut funktionieren wird. Der Grund dafür liegt darin, dass in der Produktionsumgebung oftmals eine erhebliche Nicht-Stationarität oder andere Probleme auftreten, die den Nutzen historischer Daten einschränken können. Folglich besteht bei der Bereitstellung von neuen Modellen immer ein gewisses Risiko. Dabei können beispielsweise Unstimmigkeiten zwischen Modellartefakten und der bereitstellenden Infrastruktur erkannt werden, welche dadurch auftreten, dass sich die Modelle häufiger ändern wie der Code zur Bereitstellung der Modelle. Dadurch besteht die Gefahr, dass ein älteres System nicht in der Lage ist ein Modell bereitzustellen, welches mit einem neuen Code trainiert wurde. Um das Risiko neuer Modelle zu verringern, können neue Modelle schrittweise eingeführt werden, indem neue und alte Modelle gleichzeitig verwendet werden. Dabei wird anfangs nur ein geringer Teil des Datenverkehrs auf das neue Modell geleitet und anschließend schrittweise erhöht, wenn das Modell plausible Vorhersagen trifft [74].

Des Weiteren muss sichergestellt werden, dass Modelle schnell und sicher auf eine frühere Version zurückgesetzt werden können. Ein schnelles Zurücksetzen der Modelle ist eine wichtige Reaktion auf Probleme, die durch die Überwachung erkannt werden. Da es sich bei einem Zurücksetzen der Modelle um ein Notfallverfahren handelt, sollte dies regelmäßig geübt werden [74].

2.6.4 Modell Tests

Die Grundlage für Modell Tests liegt darin, dass jede Modellspezifikation ein Code Review erhält und in einem zentralen Speicher eingecheckt wird. Eine ordnungsgemäße Versionskontrolle der Modellspezifikationen kann dazu beitragen, dass das Training von ML-Modellen überprüfbar und somit die Reproduzierbarkeit verbessert wird. Speziell bei Vorfällen in der Produktionsumgebung ist es wichtig, den Code zu kennen, der zur Erstellung eines ML-Modells verwendet wurde [74].

Wurde ein Modell trainiert, so hilft es ein tiefgründiges Verständnis über das Modellverhalten aufzubauen, in dem der Testdatensatz nach wichtigen Dimensionen unterteilt wird. Wird beispielsweise ein ML-Modell für mehrere unterschiedliche Bauteile bzw. Bauteilspezifikationen verwendet, so ist es sinnvoll das Modellverhalten je Bauteil bzw. Bauteilspezifikation zu überprüfen. Bringt das Modell in einer oder mehreren Kategorien keine zufriedenstellende Leistung, so deutet es darauf hin, dass speziell diese Kategorie in den Trainingsdaten unterrepräsentiert ist [74].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Des Weiteren muss die Leistung des Modells vor dem direkten Einsatz in der Produktion validiert werden. Nach einer Überprüfung des Verhaltens kann das Modell anschließend für die Produktion freigegeben werden oder wiederum zurückgelegt werden. Dabei soll die Leistung des Modells sowohl auf eine langsame Verschlechterung als auch auf eine kurzfristige Veränderung im Vergleich zur vorherigen Modellversion überprüft werden. Zur Analyse auf eine langsame Verschlechterung ist es hilfreich das Modell anhand eines Validierungsdatensatz zu testen und dafür Schwellwerte zu definieren. Zur Analyse der abrupten Veränderung können die Vorhersagen des Modells mit den Vorhersagen des Vorgängermodells anhand realer Daten vergleichen werden [74].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.7 Überwachung von ML-Systemen

Nachdem ein ML-System erfolgreich getestet und in die Produktion überführt wurde, ist anschließend eine konstante Überwachung notwendig. In der Realität wird die Überwachung der bereitgestellten ML-Systeme oftmals anhand von Plausibilitätsprüfungen von Ergebnissen in der Produktion durchgeführt [54]. Für diese Plausibilitätsprüfungen ist jedoch meist ein kombiniertes Wissen von Datenwissenschaftler_innen und Fachexpert_innen notwendig, wodurch Plausibilitätsprüfungen sehr zeitaufwändig und kompliziert werden. Daher soll in der für diese Arbeit gewählten Lösung eine möglichst bedienerunabhängige Überprüfung und somit eine unkontrollierte Überwachung (*engl. unsupervised monitoring*) angewendet werden.

2.7.1 Drifterkennung

Im Allgemeinen kann eine Drifterkennung über zwei Arten erfolgen. Einerseits kann der Drift über die Erkennung der Fehlerrate (engl. *error rate-based (ERB) detectors*) und andererseits durch die Erkennung der Verteilung (engl. *data distribution-based (DDB) detectors*) erfolgen [37].

Unterteilung der Überwachungsdaten

Die unkontrollierte Überwachung von ML-Systemen fokussiert sich im Allgemeinen auf die Erkennung von Änderungen in der Verteilung der Eingangs- und Ausgangsdaten. [75]–[77] Dabei nutzen die Überwachungstechniken oftmals zwei Zeitfenster, welche anhand statistischer Tests miteinander verglichen werden. Ein Vorteil dieses Ansatzes ist, dass dabei sowohl ein abrupter Drift, als auch ein gradueller Drift erkannt werden kann [78]. Anfangs wurden dafür Methoden verwendet, die jeweils gesamte Zeitfenster miteinander vergleichen, während neuere Methoden Bootstrapping anwenden, um robustere Ergebnisse zu erreichen [79]–[81]. Dazu werden die Zeitfenster in Subregionen unterteilt, die anschließend miteinander verglichen werden. Anschließend werden die regionalen Unterschiede in der Dichte zusammengefasst, wodurch ein Drift erkennbar wird. Durch die Natur dieser Methode kann auch ein regionaler Drift innerhalb eines Datensatzes erkannt werden. Um die Datensätze aus zwei unterschiedlichen Zeiträumen in Subregionen aufzuteilen, gibt es unterschiedliche Ansätze. Dasu et al. [75] verwenden kdq Trees (engl. *quad-trees with scale with the size (k) and dimensionality (d) of the data*), um den Datensatz in mehrere Quadrate zu unterteilen, während Lu et al. [82] überlappende Kreise verwenden. Kdq Trees werden aktuell in einem Großteil der Drifterkennung eingesetzt, weshalb diese Methode nachfolgend genauer erläutert wird.

Die Unterteilung auf Basis von kdq-Trees nimmt eine *d*-dimensionale Verteilung der Daten in einem Hyperwürfel an. Um diesen *d*-dimensionalen Hyperwürfel zu unterteilen wird eine Kombination von *k-d Trees*, welche gut mit der Dimensionalität skalieren, und *quad-Trees*, welche den Raum in Quadrate unterteilen, angewendet. Dafür wird beispielsweise ein 2D-Raum alternierend entweder horizontal oder vertikal unterteilt, bis die Kanten der einzelnen Sub-Räume entweder die minimale Größe δ erreicht haben, oder eine minimale Anzahl τ an Punkten innerhalb des Sub-Raums erreicht wurde. Dabei sind τ und δ Parameter, die individuell festgelegt werden müssen. Diese Vorgehensweise kann ebenfalls für hohe Dimensionen angewendet werden. Die Unterteilung wird dabei für das Referenzfenster durchgeführt und anschließend für

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

das Testfenster übernommen [75]. Um die Unterteilung mit Hilfe von kdq-Trees auch für höhere Dimensionen zur Verfügung zu stellen, kann diese Methodik mit einer Principal Component Analyse (*PCA*) kombiniert werden. Hierbei wird die hohe Dimension der Eingangsdaten mit Hilfe der PCA reduziert, indem neue aufeinander orthogonale Komponenten gefunden werden, welche die Daten besser repräsentieren [77]. Anschließend werden die Daten auf diese Komponenten projiziert und eine Unterteilung mit Hilfe von kdq-Trees wird durchgeführt. Hierbei können die gleichen Metriken verwendet werden, wie in der niedrigdimensionalen Drifterkennung [37].

Metriken zur Drifterkennung

Um die Dichte innerhalb der Subregionen für das Referenzfenster und das Testfenster zu bestimmen können unterschiedliche Tests angewendet werden. Poenaru-Olaru et al. [37] vergleichen in ihrer Arbeit kdq-Trees, welche mit den folgenden Tests kombiniert werden:

Die Kullback-Leibler (KL) Divergenz beschreibt die relative Entropie von zwei Verteilungen und ist in Gleichung (17) nach [83] dargestellt. Hierbei ist x die Referenzverteilung und y die Verteilung, welche mit der Referenzverteilung verglichen werden soll. Somit stellt x die Verteilung innerhalb der Trainingsdaten und y die Verteilung innerhalb der Produktionsdaten dar.

$$d_{Kullback}(x, y) = \sum_{i=1}^{N} x_i \, log \frac{y_i}{x_i} \tag{17}$$

Die Manhattan (MH) Distanz ist ein Spezialfall der universellen Minkowski Distanz welche in Gleichung (18) nach [84] angegeben ist. Hierbei sind x und y zwei N-dimensionale Vektoren, die miteinander verglichen werden. Wird die Variable q = 1 gesetzt, so erhält man die Manhattan-Distanz. Die Distanz der beiden Vektoren wird hierbei als Summennorm berechnet. Diese ist in Gleichung (19) nach [84] dargestellt.

$$L_q(x, y) = \sqrt[q]{\sum_{i=1}^{N} |x_i - y_i|^q} \tag{18}$$

$$L_1(x, y) = \sum_{i=1}^{N} |x_i - y_i| \tag{19}$$

Die quadrierte Euklidische (*engl. Squared Euclidean (SE)*) Distanz, basiert ebenfalls auf der universellen Minkowski Distanz (*vgl. Gleichung (18)*). In diesem Fall wird q = 2 gesetzt, wodurch die Euklidische Distanz erhalten wird. Durch das Quadrieren der Euklidischen Distanz wird Gleichung (20) erhalten.

$$d = \sum_{i=1}^{N} |x_i - y_i|^2 \tag{20}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die Chebyshev (CBS) Distanz ist ebenfalls ein Spezialfall der universellen Minkowski Distanz (*vgl. Gleichung (18)*), bei der $q = \infty$ gesetzt wird. Dadurch ergibt sich Gleichung (21) nach [84]. Diese Gleichung wird auch Maximalwert-Abstandsberechnung genannt und bestimmt die maximale absolute Distanz zwischen den zwei N-dimensionale Vektoren x und y.

$$L_\infty(x, y) = \max_i |x_i - y_i| \tag{21}$$

Die Kosinusdistanz ist eine Semi-Pseudo Distanzfunktion und wird nach Gleichung (22) gebildet, wobei S_{cos} das Kosinusmaß der Vektoren x und y ist. Das Kosinusmaß beschreibt die Ähnlichkeit von Vektoren in einem Vektorraum mit Hilfe des Kosinus des eingeschlossenen Winkels zwischen zwei Vektoren bezüglich des Nullvektors. Die beschreibende Gleichung des Kosinusmaßes ist in Gleichung (23) nach [85] abgebildet. Dabei beschreibt $\langle x, y \rangle$ das Skalarprodukt der Vektoren und $\|x\|$ bzw. $\|y\|$ den Betrag der Vektoren [85].

$$d_{cos}(x, y) = 1 - S_{cos}(x, y) \tag{22}$$

$$S_{cos}(x, y) = \frac{\langle x, y \rangle}{\|x\| * \|y\|} \tag{23}$$

Die Bhattacharyya Distanz entspricht dem negativen natürlichen Logarithmus des Bhattacharyya Koeffiziennten und ist in Gleichung (24) nach [86] dargestellt. Der Bhattacharyya (BTC) Koeffizient gibt die Überschneidungswahrscheinlichkeit zwischen zwei Verteilungen an. Er wird nach [87] in Gleichung (25) für die N-dimensionalen Vektoren x und y berechnet. Der Koeffizient kann zwischen 0 (keine Überschneidung) und 1 (vollständige Überlappung) liegen. Wenn der Bhattacharyya-Koeffizient kleiner als 0,05 beträgt, so sind die Verteilungen signifikant unterschiedlich. Ist dieser Koeffizient größer als 0,95, so sind die Verteilungen signifikant ähnlich. Liegt der Wert zwischen diesen beiden Schwellwerten, so gibt es eine wahrscheinliche Überlappung. Es kann jedoch keine Aussage über die Signifikanz der Ähnlichkeit der Verteilungen getroffen werden [88].

$$D_{batt}(x, y) = -\ln C_{batt}(x, y)) \tag{24}$$

$$C_{batt}(x, y) = \sum_{i=1}^{N} \sqrt{x_i y_i} \tag{25}$$

In der Literatur wird die KL-Divergenz am meisten verwendet, jedoch konnten Poenaru-Olaru et al. [37] in ihren Tests zeigen, dass diese nicht immer die besten Resultate in Form von Latenz und Falsch-Positive Rate liefert. So lieferte die KL-Divergenz die beste Falsch-Positive Rate, jedoch lieferten die Bhattacharyya Distanz und die Kulsinski Distanz die niedrigste Latenz. Die

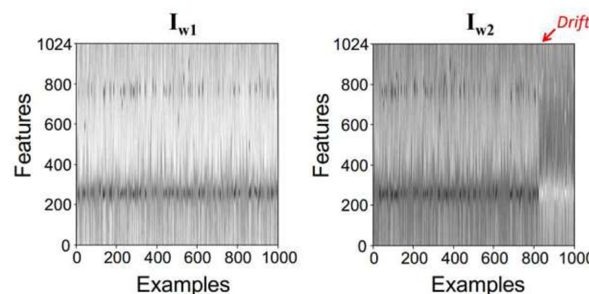
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Forscher zeigen in ihren Tests, dass keine optimale Kombination zwischen kql-Trees und Vergleichsmetrik gefunden werden konnte. Somit muss immer abgeschätzt werden, ob der Fokus in der Produktion auf einer niedrigen Falsch-Positive Rate oder einer niedrigen Latenz liegt.

Weitere datenverteilungsbasierte Überwachungen

Eine weitere Vorgehensweise ist die von Souza et al. [89] vorgestellte Bildbasierte Drifterkennung (*Image Based Drift Detector – IBDD*). Bei dieser Art von Drifterkennung wird eine zweidimensionale Repräsentation der Trainingsdaten sowie der Eingangsdaten erstellt und anschließend die Gleichheit der beiden Bilder analysiert. Der Vorteil dieser Vorgehensweise ist die schnelle Verarbeitung von hoch dimensionalen Datensätzen mit mehreren tausend Features. Der IBDD-Detektor ist unüberwacht und benötigt kein manuelles Labelling der Daten, um einen Drift zu erkennen. Zudem ist dieser modellunabhängig. Zur Drifterkennung werden zwei Fenster (*w1* und *w2*) mit der gleichen Fenstergröße *w* verglichen, wobei *w1* das Referenzfenster auf Basis der bekannten, stabilen Trainingsdaten ist und *w2* ein gleitendes Fenster ist, das kontinuierlich mit jeder neuen Eingangsprobe aktualisiert wird. Die d-dimensionalen Daten der beiden Fenster *w1* und *w2* werden anschließend in zwei unterschiedliche zweidimensionale Graustufenbilder konvertiert, wobei jeder Pixel den jeweiligen Wert eines Features innerhalb der Probe darstellt. Eine Spalte der Bilder repräsentiert eine Probe, und jede Zeile ein Feature der Probe. Dadurch kann ein Datensatz mit *q* Proben und *p* Features durch ein Bild der Höhe *p* und der Breite *q* dargestellt werden. In Abbildung 2-14 sind beispielhaft die bildhaften Datenverteilungen von zwei unterschiedlichen Zeitpunkten des *StarLightCurves* Datensatzes dargestellt. Der Datensatz enthält 1000 Proben, wobei jede Probe 1.024 Features enthält. Dadurch haben die Fenster eine Höhe von 1.024 Pixel und eine Breite von 1000 Pixel. Im rechten Bild (I_{W2}) ist zudem ein Datendrift zu erkennen, welcher gut durch eine Änderung des Kontrastes ersichtlich wird. Diese Kontraständerung erfolgt durch die Normalisierung der Daten des gleitenden Fensters.

Abbildung 2-14: Bildhafte Darstellung zweier Datenverteilungen [89]



Um die Gleichheit der beiden konvertierten Fenster zu ermitteln, wird die Mittlere Quadratische Abweichung (MSD) zwischen den beiden Bildern nach Gleichung (26) ermittelt.

$$MSD(I_{w1}, I_{w2}) = \frac{1}{p \times q} \sum_{i=1}^{p} \sum_{j=1}^{q} (I_{w1_{(i,j)}} - I_{w2_{(i,j)}})^2 \qquad (26)$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

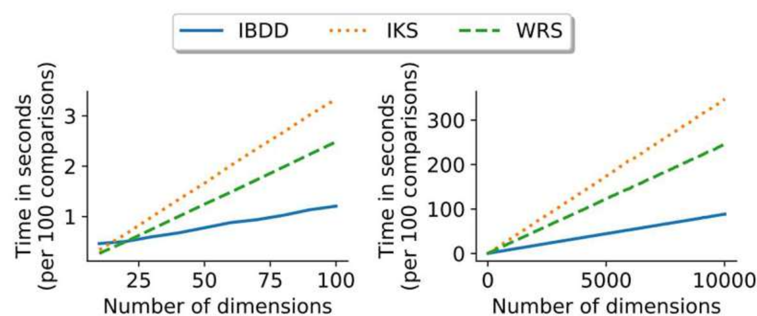
$$MSD(I_{w1}, I_{w2}) = MSD(I_{w2}, I_{w1}) \quad (27)$$

Die mittlere quadratische Abweichung misst die Differenz der Pixelstärken der zwei Bilder und hat die Eigenschaft einer nicht-negativität, sowie einer Symmetrie wie in Gleichung (27) dargestellt ist. Eine mittlere quadratische Abweichung von Null zeigt eine perfekte Gleichheit zwischen den Bildern auf. Zur Erkennung eines Daten Drifts muss die mittlere quadratische Abweichung einer Sequenz mit m Werten einen zuvor definierten maximalen Schwellwert überschreiten, oder einen minimalen Schwellwert unterschreiten. Die Schwellwerte werden im IBDD-Detektor in regelmäßigen Abständen, sowie bei Erkennung eines Drifts aktualisiert. Eine Aktualisierung nach der Erkennung eines Drifts ist notwendig, da sich das Referenzbild *w1* nicht ändert.

Bei Start des IBDD-Detektors werden die Schwellwerte geschätzt, indem das aus den Trainingsdaten erzeugte Bild *w1* mit weiteren Bildern verglichen wird, die durch k zufällige Permutationen der Trainingsdaten erzeugt wurden. Der obere Schwellwert wird durch den Mittelwert der Werte plus einfache Standardabweichung definiert, während der untere Schwellwert durch den Mittelwert der Werte minus der einfachen Standardabweichung definiert ist. Eine Aktualisierung der Schwellwerte findet nach dem gleichen Schema statt.

Diese Analyse des Daten Drifts ermöglicht eine schnellere Bearbeitung der einzelnen Proben im Vergleich zum inkrementellen Kolmogorov-Smirnov Test (IKS) oder zum Wilcoxon Test (WRS), wie in Abbildung 2-15 ersichtlich ist. Ab einer Dimensionalität der Daten von mehr als 25 ist eine zeitliche Verbesserung erkennbar, welche mit zunehmender Dimensionalität kontinuierlich zunimmt.

Abbildung 2-15: Vergleich der Bearbeitungsdauer in Abhängigkeit der Dimension [89]



2.7.2 Systemüberwachung

Eine weiterer essenzieller Überwachungsbereich ist die Überprüfung der Systemparameter eines ML-Systems. Dies beinhaltet beispielsweise die Überwachung der Trainingsgeschwindigkeit, der Latenzzeit, des Datendurchsatzes oder der *RAM*-Nutzung bei der Bereitstellung. Die Rechenleistung eines ML-Systems ist oft ein zentraler Aspekt bei der Skalierung eines ML-Systems. Eine schnelle Reaktion auf Änderungen dieser Parameter aufgrund von Änderungen wie beispielsweise der Daten, der Modellierung, der zugrunde liegenden Rechenbibliothek oder Infrastruktur ist für die Aufrechterhaltung eines leistungsfähigen Systems entscheidend [74].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.7.3 Weitere Überwachungsaspekte

Neben der Erkennung des Drifts ist auch eine Überwachung der Eingangswerte auf eine Plausibilität sinnvoll. Ungültige oder unplausible numerische Werte können während des Modelltrainings auftauchen ohne explizite Fehler auszulösen. Das Wissen, dass unplausible Werte aufgetreten sind, kann die Diagnose des Problems beschleunigen. Darum empfiehlt es sich explizit das anfängliche Auftreten von NaNs oder Unendlichkeiten zu überwachen, indem beispielsweise Grenzwerte für Nullwerte definiert werden und Warnungen ausgelöst werden, wenn die definierten Grenzwerte überschritten werden.

Die für das ML-System verwendeten Eingangsdaten werden meist aus einer Vielzahl anderer Systeme gespeist. Fallen diese Quellsysteme teilweise aus, oder werden auf andere Versionen upgedatet, so können sich die Eingangsdaten oft ändern und dadurch die Vorhersagegenauigkeit eines ML-Systems beeinträchtigen, ohne dass die Überwachung Fehler auslöst. Aus diesem Grund sollte sichergestellt werden, dass das Team, welches für die Bereitstellung des ML-Services verantwortlich ist, alle Ankündigungslisten der Quellsysteme abonniert hat, so dass bei einer Änderung der zugrundeliegenden Quellsysteme eine zusätzliche manuelle Plausibilitätsprüfung eingeführt werden kann.

2.8 Neutraining von ML-Modellen

Um ein Modell auf die sich laufend ändernde Produktionsumgebung anzupassen, ist ein regelmäßiges Lernen des ML-Modells unerlässlich. Dabei können zwischen zwei Arten von kontinuierlichem Lernen unterschieden werden. Passives Neutraining von ML-Modellen, welches in den Arbeiten [90]–[92] verwendet wird, beschreibt das kontinuierliche Neutraining und Updaten der Modelle nach festgelegten Zeitpunkten, unabhängig davon, ob ein Drift innerhalb der Daten erkannt wird. Dem gegenüber stehen aktive Ansätze [93], [94], welche einen Datendrift durch das Monitoring erkennen und anschließend ein erneutes Training des ML-Modells starten. Im Allgemeinen wird der aktive Ansatz dem passiven Ansatz vorgezogen, da diese im Allgemeinen einerseits eine geringere Anzahl an Trainings erforderlich machen und andererseits auch die Leistung des ML-Systems erhöhen können [89]. So konnten Ahmed et al. [95] in ihrer Arbeit herausfinden, dass durch die Anwendung des aktiven Ansatzes in der Anomalie Erkennung von Stromnetzen eine bis zu 10% verbesserte Leistung erzielt werden kann, als mit einem passiven Ansatz. Aktive Ansätze liefern auch zusätzliche Informationen über die Driftpunkte im Datenstrom und helfen den, für die Datenerzeugung verantwortlichen, Prozess zu verstehen.

Nikolov et al. [96] schlagen dabei vor, in einem ersten Schritt den Zeitpunkt zu erkennen, indem der Drift stattgefunden hat. Ab diesem Zeitpunkt werden die notwendigen Daten für das erneute Trainieren des ML-Modells gespeichert und anschließend wird das ML-Modell mit den neu gesammelten Daten trainiert. Dadurch konnte die Leistung der Algorithmen im Durchschnitt um 25% gesteigert werden. Anschließend wird das neu trainierte Modell automatisiert getestet und bei erfolgreichem Ergebnis in die Produktion überführt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3 Umsetzungsbeschreibung

Dieses Kapitel beschreibt die theoretische Vorgehensweise des *Kapitels 4*. In Abschnitt 3.1 wird der IST Stand im Unternehmen dargestellt. Abschnitt 3.2 beschreibt die vorgeschlagene Architektur und definiert die dafür notwendigen Softwareumgebungen. In Abschnitt 3.3 wird der vorgeschlagene Prozessablauf für die Erstellung eines Machine Learning Services dargestellt. In Abschnitt 3.4 wird anschließend der Prozessablauf für den vorliegenden ML-Service beschrieben.

3.1 IST Stand im Unternehmen

In diesem Abschnitt wird ein Überblick über den Status Quo innerhalb des Unternehmens gegeben. Dazu wird in einem ersten Schritt der aktuelle Stand beschrieben und anschließend mit Hilfe des Reifegradmodells die aktuelle Reife des Unternehmens in Hinblick auf MLOps erörtert.

In der *Thyssenkrupp Presta Steering* sind mehrere Teams von Datenwissenschaftler_innen mit der Entwicklung von Machine Learning Modellen beschäftigt. Die einzelnen Teams arbeiten jeweils in unterschiedlichen Fachabteilungen und erarbeiten fachspezifische ML-Modelle, welche anschließend übergeben werden können. Mit dem Schreiben dieser Arbeit ist dem Verfasser kein Machine Learning Modell bekannt, welches aktiv in der Produktion angewendet wird. Vielmehr fokussieren sich die unterschiedlichen Teams von Datenwissenschaftler_innen darauf, mit Hilfe von unterschiedlichen Proof of Concepts (*PoC*) das Vertrauen der einzelnen Fachabteilungen für dieses Thema zu bekommen. Die Softwareentwicklung, sowie die IT-Fachabteilung sind nicht direkt in die Arbeiten der Datenwissenschaftler_innen Teams eingebunden.

Wird nun das Reifegradmodell auf den aktuellen Stand der *Thyssenkrupp Presta Steering* angewendet, so lassen sich folgende Punkte erkennen. Die Data Science Teams arbeiten in Silos neben dem Team der Softwareentwicklung und der zentralen IT und stehen in keinem regelmäßigen Kontakt untereinander. Die Daten für die Entwicklung der Modelle werden von den Data Science Teams manuell zusammengestellt und das erstellte Modell wird als Modelldatei in der jeweiligen Ordnerstruktur lokal abgelegt. Aus diesem Grund ist der gesamte Prozess stark von der Expertise des Data Science Teams abhängig. Es findet zudem keine zentrale Rückverfolgung der einzelnen Experimente statt. Die Erstellung des ML-Modells erfolgt rein durch die Datenwissenschaftler_innen, welche anschließend das trainierte Modell manuell an die Softwareentwicklung weitergeben. Die Softwareentwicklung verwendet für die Erstellung der Software bereits DevOps aber noch kein MLOps. Somit lässt sich der aktuelle Stand der Operationalisierung von Machine Learning im Unternehmen in die vorletzte Stufe (*Stufe 1*) des MLOps Reifegradmodells charakterisieren.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2 Vorgeschlagene Architektur

Während im Unternehmen selbst aktuell die zweitletzte Stufe (*Stufe 1*) des Reifegradmodells vorherrschend ist, ist das Ziel dieser Architektur, die höchste Stufe von MLOps *(Stufe 4)* zu erreichen. In dieser Stufe wird nach *Microsoft* [52] der für das ML-Modell benötigt Code versionisiert und zusätzlich dazu werden ebenfalls die für das Training verwendeten Daten, sowie das trainierte und gepackte ML-Modell selbst versionisiert. Softwareingenieur_innen und Dateningenieur_innen arbeiten nicht mehr in Silos, sondern sie arbeiten miteinander und haben eine gemeinsame Arbeitsplattform mit deren Hilfe die Zusammenarbeit erleichtert wird. Die von den Dateningenieur_innen erstellten Modelle werden durch eine CT-Pipeline automatisch trainiert und mit einer CD-Pipeline automatisiert den Endkund_innen in der Produktionsumgebung zur Verfügung gestellt. Anschließend wird die Aussagekraft des ML-Modells konstant überwacht, so dass Fehler automatisiert erkannt werden können und zu einer Benachrichtigung bzw. zu einem erneuten Training des ML-Modells führen. Die vorgeschlagene Architektur ermöglicht es diese Anforderungen zu lösen.

Bei der Erstellung der Architektur wurde zudem die Unternehmensanforderung berücksichtigt, dass die Architektur auf den bereits vorhandenen Technologien und organisationalen Abläufen aufzubauen ist. Dadurch soll die Anzahl an unterschiedlichen Technologien im Unternehmen so gering wie möglich gehalten und die Änderungsmaßnahmen für das Team verringert werden. Aus diesen Anforderungen resultiert eine Architektur, die in weiten Teilen aus Elementen der Firma *Microsoft* besteht, da die *Thyssenkrupp Presta Steering Microsoft* als zentralen Baustein ihrer internen IT-Architektur verwendet. Abbildung 3-1 zeigt die dafür vorgeschlagene Architektur. Die Architektur ist an *Microsoft* [50] angelehnt und besteht aus vier primären Bestandteilen nämlich *Azure DevOps*[1], *Azure Machine Learning*[2], *Azure Monitor*[3] und *Azure Blob Storage*[4]. Dabei wird *Azure DevOps* für die Versionisierung des Codes, sowie für die automatisierten Pipelines verwendet. *Azure Machine Learning* wird für die Versionisierung der Modelle, der zugrundeliegenden Trainingsdaten, sowie für die Bereitstellung des ML-Modells als ML-Service verwendet. Die jeweiligen Daten werden von *Azure Machine Learning* in einem *Azure Blob Storage* gespeichert. Wurde ein ML-Service bereitgestellt, so kann dieser mit Hilfe von *Azure Monitor* überwacht werden, indem die wichtigsten Kennwerte gespeichert werden, und automatisiert Benachrichtigungen versendet werden können.

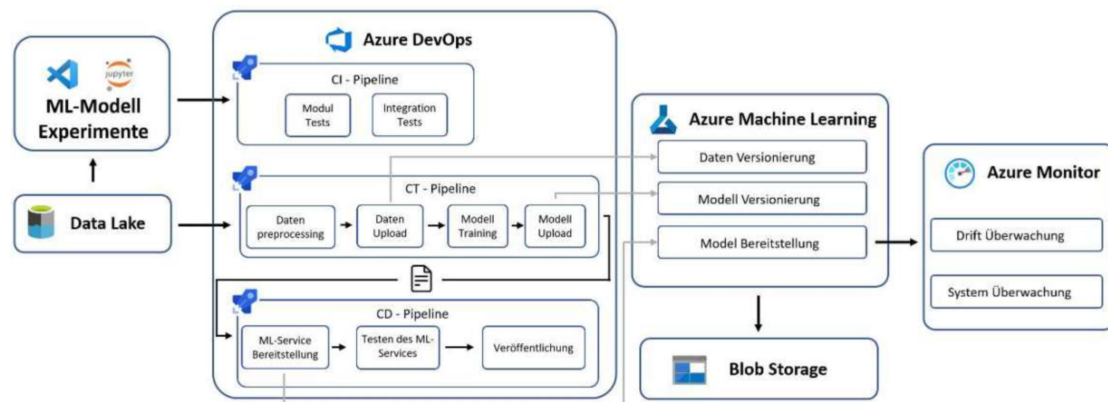
Nachfolgend werden detailliert die einzelnen Bestandteile der Architektur und ihre jeweiligen Beziehungen zueinander erläutert. Eine genauere Beschreibung der technischen Umsetzung findet in *Kapitel 4* statt.

---

[1] https://azure.microsoft.com/de-de/products/devops
[2] https://azure.microsoft.com/de-de/products/machine-learning
[3] https://azure.microsoft.com/de-de/products/monitor
[4] https://azure.microsoft.com/de-de/products/storage/blobs/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 3-1: Vorgeschlagene Architektur



3.2.1 Azure DevOps Umgebung

Die *Azure DevOps* Umgebung wird, wie der Name suggeriert, im Allgemeinen für die Umsetzung von DevOps (*vgl. Abschnitt 2.3*) in Unternehmen herangezogen und besteht selbst aus unterschiedlichen Services [97]. Diese Umgebung ist bereits im Unternehmen etabliert, so dass diese Umgebung als Kern der Architektur definiert wird. Für die vorliegende Architektur sind speziell zwei Services von Bedeutung. Der erste Service nennt sich *Azure Repos*[1] und wird für die Versionisierung des Codes verwendet. *Azure Repos* ist ein GIT-Speicher, der die Versionisierung des Codes mit Hilfe von unterschiedlichen Zweigen übernimmt. Im vorliegenden Fall wird für jeden ML-Service ein eigenes Projekt angelegt und jede Änderung eines Features wird in einem neuen Zweig durchgeführt. Azure Repos bietet eine direkte Schnittstelle zu weiteren Services. Wurde beispielsweise in einem Experiment eine verbesserte Version einer Modellarchitektur entwickelt oder die Datenaufbereitung in der ML-Pipeline angepasst und diese Änderung in den *main* Zweig übernommen, so wird anschließend automatisch die CI-Pipeline ausgeführt.

Der zweite Service nennt sich *Azure Pipelines*[2] und wird für die automatisierte Durchführung der definierten Pipelines verwendet. Konkret sollen mit Hilfe dieses Services vier unterschiedliche Pipelines, nämlich eine CI-Pipeline, eine CT-Pipeline, eine CD-Pipeline sowie eine Daten Pipeline ausgeführt werden. Die detaillierte Beschreibung der einzelnen Pipelines wird in Abschnitt 3.4 erläutert.

[1] https://azure.microsoft.com/de-de/products/devops/repos/
[2] https://azure.microsoft.com/de-de/products/devops/pipelines/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2.2 Azure Machine Learning Umgebung

Die *Azure Machine Learning* Umgebung wird in der vorgeschlagenen Architektur einerseits für die automatisierte Bereitstellung des ML-Services als Endpunkt verwendet. Die Bereitstellung des ML-Services erfolgt mit Hilfe einer *Azure Container Instanz*[1], welche vor allem auf Grund ihrer einfachen Bedienung für das vorliegende Konzept verwendet wird. Alternativ kann ein ML-Service innerhalb der *Azure Machine Learning* Umgebung auch mit Hilfe eines *Azure Kubernetes Service*[2] bereitgestellt werden. Diese Möglichkeit wird bevorzugt bei hohen Auslastungen angewendet, da hierbei ein ML-Service in mehreren Containern gleichzeitig bereitgestellt und somit die Last auf mehrere Container verteilt wird [98]. Da es sich in dieser Arbeit um einen Prototypen handelt, bei dem keine hohen Auslastungen zu erwarten sind, wird die *Azure Container Instanz* als Bereitstellungsservice bevorzugt. Neben der Bereitstellung des ML-Services übernimmt die *Azure Machine Learning* Umgebung auch die Versionierung der relevanten Modelldaten, die während des kontinuierlichen Trainings erstellt werden. Es werden innerhalb der *Azure Machine Learning* Umgebung sowohl die dem Modell zugrunde liegenden Trainingsdaten, als auch das ML-Modell selbst versionisiert. Des Weiteren werden die Logs des gesamten Trainingslaufes abgespeichert. Die *Azure Machine Learning* Umgebung selbst ist mit einem eigenen *Blob Storage* verbunden, in dem die Daten abgespeichert werden.

3.2.3 Azure Monitor Umgebung

Nach der Überführung des trainierten Modells in die Test- und Produktionsumgebung ermöglicht die *Azure Monitor*[3] Umgebung eine genaue Überwachung des bereitgestellten ML-Systems. Die *Azure Monitor* Umgebung besteht wiederum selbst aus mehreren unterschiedlichen Services, die für eine Überwachung verwendet werden.

In der vorliegenden Arbeit wird der Service *Protokolle* für eine Auswertung der übermittelten Daten verwendet. Dieser Service wird speziell dafür herangezogen, den Drift innerhalb der Eingangsdaten und der Ausgangsdaten auswerten zu können.

Der Service *Dashboards* wird verwendet, um die zuvor erstellten Auswertungen graphisch in Form eines Dashboards darzustellen, so dass der Nutzer einen direkten Überblick über den aktuellen Status des Systems bekommt und Unregelmäßigkeiten schnell erkannt werden können. Des Weiteren bietet dieser Service die Möglichkeiten bei Unregelmäßigkeiten automatisierte Warnungen an die Nutzer zu senden.

[1] https://azure.microsoft.com/de-de/products/container-instances/
[2] https://azure.microsoft.com/de-de/products/kubernetes-service/
[3] https://azure.microsoft.com/de-de/products/monitor

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Zudem wird der Service *Application Insights* genutzt, um den Status des Containers zu überwachen. Mit Hilfe dieses Services wird die Anzahl der fehlerhaften Anforderungen, die Serverantwortzeit, die Anzahl der Serveranforderungen, sowie die Verfügbarkeit des Services automatisiert überwacht.

3.2.4 Azure Blob Storage Umgebung

Die *Azure Blob Storage* Umgebung ist direkt mit der Azure Machine Learning Umgebung verbunden, um die relevanten Daten eines Machine Learning Modells zu speichern. Der *Azure Blob Storage* selbst ist ein Objektspeicher und für die Speicherung einer großen Menge an unstrukturierten Daten optimiert [99]. Der Blob Storage ist direkt mit dem Arbeitsplatz innerhalb der *Azure Machine Learning* Umgebung verbunden, so dass sämtliche relevante Daten automatisiert abgelegt werden.

3.2.5 Erweiterte Softwareumgebung

Neben den zuvor erwähnten Umgebungen, welche den Kern der vorgeschlagenen Architektur ausmachen, ist noch eine erweiterte Softwareumgebung für die Implementierung von MLOps notwendig.

Die erweiterte Softwareumgebung besteht aus einem *Azure Data Lake*[1], welcher für die Speicherung der relevanten Prozessdaten verwendet wird. Das Speichern der Daten innerhalb des Data Lakes wurde vom Unternehmen bereits implementiert, weshalb dies eine Schnittstelle zur schon bestehenden Infrastruktur ist. Im Unternehmen wird dieser Data Lake als zentraler Speicher der Maschinendaten verwendet. Die Daten aus dem Data Lake werden mit Hilfe der Daten Pipeline innerhalb der *Azure DevOps* Umgebung automatisiert für das Training aufbereitet.

Eine weitere Schnittstelle besteht zu den einzelnen Modell-Experimenten, die von den Datenwissenschaftler_innen durchgeführt werden. Die Experimente selbst sind kein zentraler Aspekt dieser Arbeit, weshalb diese nur kurz erwähnt werden. Es soll aber angemerkt werden, dass die Experimente sowohl in *Visual Studio Code*[2] und auch in der *Jupyter Notebook*[3] Umgebung erstellt werden können. *Visual Studio Code* ermöglicht eine direkte Anbindung zu *Azure DevOps*, so dass ohne großen Aufwand eine Versionisierung des Codes mit *Azure DevOps* möglich ist. Aus diesem Grund ist diese Variante der Möglichkeit mit Jupyter Notebooks zu bevorzugen, in denen der aus den Experimenten resultierende Code manuell in die *Azure DevOps* Umgebung überführt werden muss. Des Weiteren wird empfohlen, die Experimente nach dem CRISP-DM Zyklus zu erstellen [100].

---

[1] https://azure.microsoft.com/de-de/products/storage/data-lake-storage/
[2] https://code.visualstudio.com/
[3] https://jupyter.org/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Sowohl der Data Lake als auch die jeweiligen Modell-Experimente beinhalten eine gegenseitige Schnittstelle, da auch die für die Experimente benötigten Kurvendaten von diesem zentralen Speicher heruntergeladen werden müssen und anschließend für die Experimente der Datenwissenschaftler_innen verwendet werden können.

3.3 Vorgeschlagener Erstellungsprozess

In diesem Abschnitt wird der Prozess zur Erstellung eines ML-Services basierend auf einem Algorithmus zur Anomalieerkennung definiert. Es wird erläutert, wie die Erstellung eines ML-Services mit Hilfe der vorgeschlagenen Architektur ermöglicht wird. In Abbildung 3-2 ist der an *iguazio* [101] angelehnte Prozess dargestellt. Das Ziel des vorgeschlagenen Erstellungsprozesses ist es, die bereits bestehenden DevOps Praktiken innerhalb des Unternehmens zu übernehmen und bei Bedarf zu erweitern, sodass eine einfache Integration ermöglicht wird.

Wie bei Scrum üblich, ist der Erstellungsprozess auch in der *Thyssenkrupp Presta Steering* durch regelmäßige Sprints gekennzeichnet, in denen die definierten Ziele erfüllt werden sollen. Des Weiteren sind innerhalb des Teams die Rollen des Product Owners, des Scrum Masters, sowie die Entwickler_innen festgelegt. Der Product Owner verantwortet die Produktqualität und erstellt in Absprache mit den Kund_innen Aufgaben, die im Rahmen der regelmäßigen Sprints erfüllt werden sollen, um so die Bedürfnisse der Kund_innen zu decken. Hierbei werden die Aufgaben der Kund_innen im Product Backlog gespeichert. Beim Start eines neuen Sprints werden die Aufgaben mit der höchsten Priorität ausgewählt und als Sprint Ziel festgelegt. Der Scrum Master überwacht den gesamten Scrum Ablauf und moderiert die einzelnen Termine. Die Entwickler_innen erfüllen im Rahmen des Sprints die an sie gestellten Aufgaben und sind dabei in regelmäßiger Abstimmung mit dem Product Owner.

Der Erstellungsprozess beginnt mit der durch den Product Owner gestellten Anforderung einen ML-Service zu entwickeln bzw. einen bestehenden anzupassen als ein Sprint Ziel. Anschließend wird im zentralen Codespeicher ein neues Projekt angelegt. Besteht schon ein ML-Projekt welches angepasst werden soll, so kann die Entwicklung durch die Erstellung eines Unterzweiges durchgeführt werden. In beiden Fällen werden die Anforderungen durch Stories innerhalb eines Sprints definiert. Darauf aufbauend werden durch die Dateningenieur_innen Experimente durchgeführt, um die gestellten Anforderungen erfüllen zu können. Die Experimente werden nach dem CRISP-DM Zyklus absolviert und entsprechen den Schritten 2 – 7 innerhalb des ML-Lebenslaufes (vgl. Abschnitt 2.2). Die durchgeführten Änderungen werden in regelmäßigen Abständen im Codespeicher abgespeichert.

Anschließend erfolgt eine Analyse der Leistung des erstellten Modells, sodass sichergestellt wird, dass das erstellte ML-Modell einen Mehrwert entsprechend den in der Story definierten Anforderungen bietet. Konnte kein Mehrwert festgestellt werden, so muss der vorherige Schritt nochmals wiederholt werden. Diese Schritte werden mehrmals hintereinander wiederholt, bis das gewünschte Ergebnis erreicht wird.

Konnte ein Mehrwert nachgewiesen werden, so wird nachfolgend das Modell in eine ML-Pipeline konvertiert, um als ML-Service innerhalb eines Containers funktionieren zu können. Diese ML-Pipeline beinhaltet die Aufbereitung der übergebenen Daten, das Trainieren der auf-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

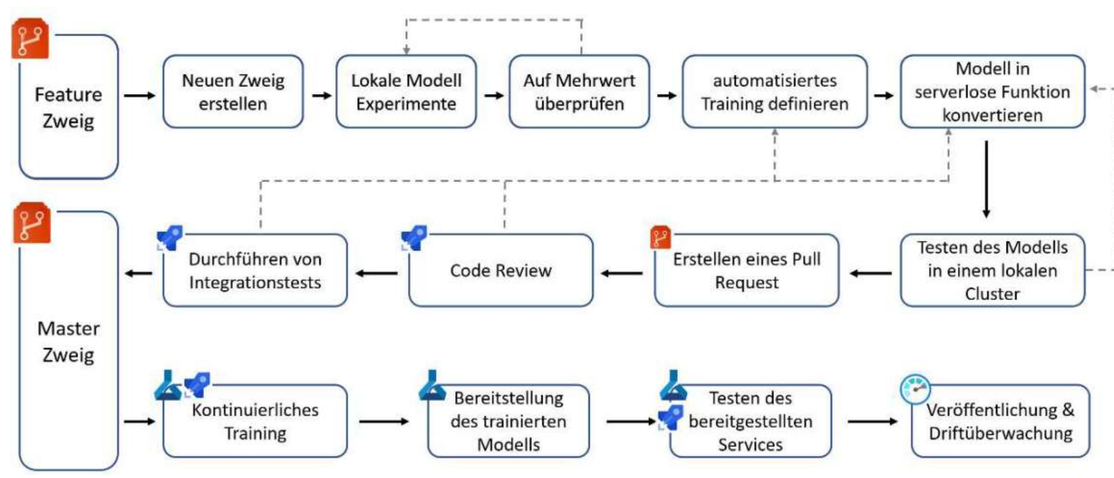
bereiteten Daten, die Verarbeitung der aufbereiteten Daten durch das trainierte ML-Modell, sowie die Ausgabe der relevanten Daten, was neben dem Modellergebnis auch die Resultate des Monitorings sind.

Anschließend wird die erstellte ML-Pipeline in einem lokalen Cluster bereitgestellt und getestet. Dadurch können einerseits Fehler innerhalb des Bereitstellungsprozesses, als auch Fehler innerhalb der ML-Pipeline gefunden werden. Wird ein Fehler gefunden, so muss dieser im vorhergehenden Schritt gelöst werden. Dies benötigt in der Regel mehrere Feedbackschleifen, bis alle Fehler innerhalb der ML-Pipeline ausgebessert wurden. Nachdem sämtliche Fehler innerhalb der ML-Pipeline behoben sind, kann schließlich ein *Pull Request* erstellt werden, welcher den Start der Integration des neuen Features in den bestehenden Code des *master*-Zweiges markiert. Eine oder mehrere Entwickler_innen überprüfen je nach Anforderung den Code und geben ihn in weiterer Folge entweder frei, oder weisen ihn mit Änderungsvorschlägen zurück. Diese müssen anschließend eingearbeitet werden. Wurde das Code Review ohne weitere Änderungsvorschläge abgeschlossen, so wird im nächsten Schritt die kontinuierliche Integrationspipeline (vgl. Abschnitt 3.4.1) gestartet, wodurch die Änderungen in den bestehenden *master*-Zweig integriert und getestet werden.

Danach steht der zusammengeführte Code zur Überführung in die Produktion zur Verfügung. Dazu wird das kontinuierliche Training gestartet, welches aktuelle Trainingsdaten bekommt und das ML-Modell mit diesen aktuellen Trainingsdaten trainiert. Das trainierte ML-Modell wird anschließend als ML-Service in einer Testumgebung bereitgestellt und abschließende Tests werden durchgeführt. Sind diese erfolgreich, so kann der Service in der Produktionsumgebung bereitgestellt werden und eine kontinuierliche Überwachung des Drifts, sowie der Leistung wird durchgeführt.

Der vorgeschlagene Prozess entspricht somit dem in *Abschnitt 2.4.1* vorgestellten allgemeinen MLOps Prozessmodell, welches eine Vereinfachung der Operationalisierung von ML-Services zum Ziel hat.

Abbildung 3-2: Vorgeschlagener Prozess zur Erstellung eines ML-Services

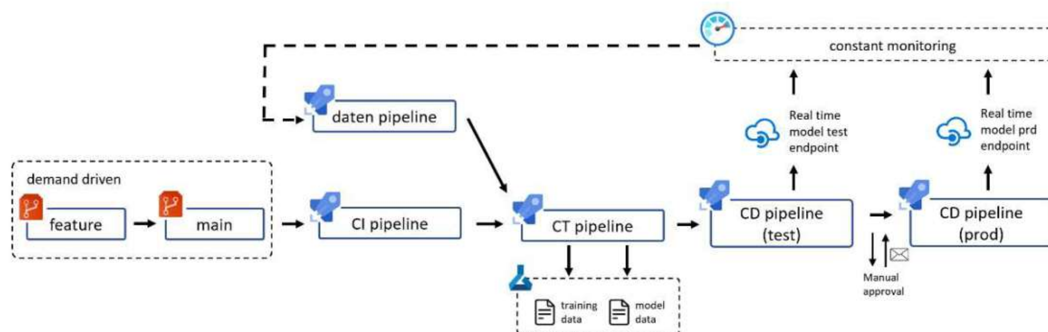


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.4 Vorgeschlagener Pipeline Prozess

In der vorhergehenden Prozessbeschreibung wurden unterschiedliche Pipelines angesprochen, welche für die erfolgreiche Umsetzung von MLOps in einem Unternehmen benötigt werden. In diesem Abschnitt wird noch einmal detailliert auf die einzelnen Pipelines eingegangen. Abbildung 3-3 gibt einen allgemeinen Überblick über die einzelnen Pipelines und ihre Verbindung zueinander.

Abbildung 3-3: Vorgeschlagener Pipeline Prozess



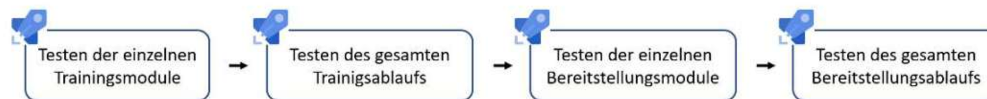
Der Prozess wird einerseits durch die Daten Pipeline gestartet, welche durch einen detektierten Drift gestartet wird. Andererseits kann der Prozess auch durch ein manuelles Überführen eines *feature* Zweiges in den *main* Zweig mittels eines *commit* Befehls innerhalb der *Azure DevOps* Umgebung erfolgen, welches je nach Bedarf von Datenwissenschaftler_innen gegeben wird. Dies ist beispielsweise der Fall, wenn die Datenwissenschaftler_in innerhalb der Experimentierphase ein neues ML-Modell erstellt hat, welches bessere Ergebnisse wie das bestehende ML-Modell erzielt und dieses in eine funktionierende ML-Pipeline konvertiert wurde. Anschließend wird automatisiert die CI-Pipeline innerhalb der *Azure DevOps* Umgebung durchlaufen, in welcher die für das Training verwendeten Daten, sowie das gepackte ML-Modell automatisiert in der *Azure Machine Learning* Umgebung abgespeichert werden. Nachfolgend wird automatisch der erste Teil der CD-Pipeline durchlaufen. In diesem Teil wird das ML-Modell automatisch in die Testumgebung überführt und nach der Überführung mittels zuvor definierter Tests überprüft. Erfolgen die Tests positiv, so können Produktionsdaten an das überführte ML-Modell gesendet werden und die Performance des ML-Modells kann in der realen Umgebung überwacht werden. Des Weiteren wird der für den jeweiligen Machine Learning Dienst zuständigen Personengruppe automatisiert eine Nachricht zugesendet, in der diese auf ein neues bereitgestelltes Modell aufmerksam gemacht wird. Diese kann anschließend mit Hilfe des eingerichteten Dashboards in der Azure Monitor Umgebung die Performance des neuen Modells mit der Performance des schon bestehenden Modells in der Produktionsumgebung vergleichen. Wird auch bei der Verarbeitung von realen Produktionsdaten eine Verbesserung des Modells erkannt, so wird der zweite Teil der CD-Pipeline von der zuständigen Personengruppe manuell gestartet und das Modell wird automatisch in die Produktion überführt. Ab diesem Zeitpunkt generiert das Modell einen Mehrwert innerhalb des Unternehmens.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.4.1 CI-Pipeline

Die CI-Pipeline wird durch die Erstellung eines *Pull Request* auf den *master* Zweig des Codespeichers, sowie einem darauffolgend positiven Code Review gestartet und dient zur Integration der neu erstellten Features in den *master* Zweig. Der Kern der CI-Pipeline ist die Überprüfung der Codequalität, der Module des ML-Services sowie der ML-Trainingspipeline. Für das vorliegende Problem wurden mehrere Tests basierend auf den Empfehlungen von Abschnitt 2.6 implementiert. In Abbildung 3-4 ist schematisch der Ablauf der CI – Pipeline dargestellt.

Abbildung 3-4: Ablauf der CI – Pipeline



In einem ersten Schritt wird der Code zur Bearbeitung der Input Daten getestet. Dieser ist oftmals simpel gestaltet, jedoch hat er einen großen Einfluss auf die Leistung des gesamten ML-Services. Dazu wird der Code mit einer kleinen Anzahl von Testdaten auf die erwartete Ausgabe getestet. Des Weiteren wird der Code zur Spezifizierung des ML-Modells auf Plausibilität getestet. Die Spezifizierung des ML-Modells beinhaltet dabei die Anzahl der Epochs, der Batch Größe, sowie den Validierungssplit.

Im nächsten Schritt wird mit Hilfe einer geringen Anzahl an Trainingsdaten das Verhalten des Modells während des Trainings untersucht. Hierbei wird vor allem darauf geachtet, dass während des Modelltrainings eine Konvergenz der Fehlerfunktion eintritt. Ein weiterer Aspekt, der im Rahmen der CI-Pipeline analysiert wird, ist die Erstellung der gewünschten Artefakte.

Des Weiteren wird die Integration des neuen Codes in den gesamten Bereitstellungsablauf getestet. Dafür werden die einzelnen Module getestet, bevor anschließend der gesamte Bereitstellungsablauf überprüft wird. Wie in *Abschnitt 2.6.3* beschrieben, ist das Testen der gesamten ML-Pipeline ein zeitaufwändiger Prozess, wodurch dies nur mit einer kleinen Teilmenge der Daten durchgeführt wird, um ein schnelles Ergebnis zu liefern. Der ML-Service wird dabei aufgebaut und der gesamte ML-Service wird mit Beispieldaten getestet.

3.4.2 CT-Pipeline

Die kontinuierliche Tainingspipeline (*kurz: CT-Pipeline*) ist für das automatisierte Neutraining des ML-Modells, sowie für die darauffolgenden Modelltests und die Versionisierung des ML-Modells verantwortlich. Die CT-Pipeline wird entweder durch einen erfolgreichen Durchlauf der CI-Pipeline oder durch einen externen Trigger gestartet. Dieser Trigger kann beispielsweise ein detektierter Datendrift, oder ein zeitlich gesteuertes Neutraining des ML-Modells sein. Wie in Abschnitt 2.8 beschrieben, erzielt ein ereignisgesteuerter Trigger eine stark verbesserte Leistung im Vergleich zu einem zeitgesteuerten Trigger, weshalb in der vorliegenden Arbeit ein ereignisgesteuerter Trigger bei Erkennung eines Drifts angewendet wird. Abbildung 3-5 stellt die jeweiligen Schritte innerhalb der CT-Pipeline schematisch dar.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 3-5: Schritte innerhalb der CT – Pipeline

In der CT-Pipeline werden die notwendigen Trainingsdaten aus dem Data Lake geladen und für das Training aufbereitet. Anschließend werden die aufbereiteten Trainingsdaten validiert, um eine sinnvolle Modellausgabe garantieren zu können. Die Validierung beinhaltet nach Abschnitt 2.6.1 eine Überprüfung der Daten auf das Vorhandensein von überproportional vielen Null Werten und NaN-Werten, sowie die Überprüfung, ob die Trainingsdaten in einem vorgegebenen Wertebereich liegen. Die Daten werden anschließend abgespeichert, so dass jederzeit eine Rückverfolgung der Daten mit dem trainierten Modell ermöglicht wird. In einem nächsten Schritt wird das Modell mit den zuvor validierten Daten trainiert. Zudem wird ebenfalls das trainierten Modells abgespeichert, so dass auch hierbei eine Rückverfolgung ermöglicht wird und im Rahmen der Bereitstellung auf das trainierte ML-Modell zugegriffen werden kann.

3.4.3 CD-Pipeline

Die kontinuierliche Bereitstellungspipeline (*kurz: CD-Pipeline*) ist für die Bereitstellung des ML-Modells als ML-Service verantwortlich. Die CD-Pipeline ist zweistufig ausgeführt, so dass in einem ersten Schritt der ML-Service in einer Testumgebung veröffentlicht wird. In dieser Testumgebung wird der Service automatisiert getestet. Nach Abschnitt 2.6 wird der Service in dieser Stufe zwei Tests unterzogen. Der erste Test analysiert anhand von Beispieldaten eine einwandfreie Bearbeitung dieser durch das gesamte ML-System. Der zweite Test vergleicht die Performance des vorhandenen ML-Systems in der Produktion mit dem ML-System in der Testumgebung. Dieser wird auch A/B Test genannt. Dazu empfängt der ML-Service bereits in der Testumgebung Daten aus der Produktionsumgebung und verarbeitet diese. Anschließend wird eine Benachrichtigung an die jeweils verantwortlichen Personen gesendet, so dass die Ergebnisse manuell überprüft werden können. Verlaufen die Tests positiv und der ML-Service liefert zufriedenstellende Ergebnisse, so wird der zweite Teil der CD-Pipeline manuell durch die jeweils verantwortliche Person gestartet und der ML-Service wird in der Produktionsumgebung veröffentlicht. Ab diesem Zeitpunkt steht der ML-Service den Endnutzern zur Verfügung und generiert somit einen unternehmerischen Mehrwert.

Die CD-Pipeline kann manuell nach Abschluss der CT-Pipeline aktiviert werden. Es wird empfohlen, die CD-Pipeline gleich nach Abschluss der CT- Pipeline zu aktivieren, um die Zeit für die Überführung eines neuen ML-Modells in die Produktionsumgebung zu minimieren. Die Überführung des ML-Services in die Produktion wird manuell durch die jeweils zuständige Person gestartet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.5 Vorgeschlagene Überwachung

Wurde das ML-Modell als ML-Service in die Produktion überführt, so erfolgt nachfolgend eine kontinuierliche Überwachung. Wie in Abschnitt 2.7 beschrieben, werden zwei unterschiedliche Bereiche des ML-Modells überwacht. Der erste Bereich ist die Systemüberwachung, in dem relevante Daten bezüglich der Systemgesundheit überwacht werden. Der zweite Bereich ist die Driftüberwachung der Eingangs- und Ausgangsdaten.

3.5.1 Systemüberwachung

Zur Überwachung des Systems werden die in Abschnitt 2.7.2 beschriebenen Metriken herangezogen. So wird das Machine Learning System durch die Analyse der Parameter Latenzzeit, Datendurchsatz, sowie die RAM – Nutzung überwacht. Überschreiten die Werte jeweils zuvor definierte Schwellwerte, so führt dies zu einer automatisierten Benachrichtigung.

3.5.2 Driftüberwachung

Für eine Driftüberwachung des bereitgestellten ML-Services ist sowohl eine Überwachung der Eingangsdaten als auch eine Überwachung der Ausgangsdaten nötig (vgl. Abschnitt 2.7.1), da eine reine Überwachung der Eingangsdaten eine hohe Anzahl an falschen Alarmen produzieren kann. Denn nicht jeder Datendrift innerhalb der Eingangsdaten hat auch eine Auswirkung auf die Leistungsfähigkeit des ML-Systems selbst [102]. Sculley et al. [2] geben an, dass in einem funktionierenden ML-System die Verteilung des Ausgansdaten etwa gleich der Verteilung der zugrundeliegenden Eingangsvariablen ist. Somit wird ein relevanter Datendrift sowohl in der Eingangsdatenüberwachung als auch in der Ausgangsdatenüberwachung detektiert. Wird ein Datendrift lediglich in der Eingangsdatenüberwachung detektiert, so handelt es sich sehr wahrscheinlich um einen harmlosen Datendrift, wodurch kein erneutes Training des ML-Modells erforderlich ist. Wird ein Drift nur in den Ausgangsdaten erkannt, so deutet dies auf einen Konzept Drift hin, wodurch eine genauere Analyse des gesamten ML-Systems notwendig ist.

Überwachung der Eingangsdaten

Die Überwachung der Eingangsdaten stellt auf Grund ihrer hochdimensionalen Eigenschaft eine Herausforderung dar, da ein Großteil der Überwachungsmethoden nur auf eine Dimension angewendet werden können. Werden diese Methoden auf hochdimensionale Eingangsdaten angewendet, so wird für die Drifterkennung ein hoher Rechenaufwand benötigt. Zudem kann dadurch eine hohe falsche Drifterkennung auftreten [89]. Aus diesem Grund können nicht alle in Abschnitt 2.7.1 genannten Überwachungsmethoden für die Überwachung der Eingangsdaten angewendet werden. Durch eine geringe Bearbeitungsdauer des IBDD – Drifterkennungsalgorithmus im Vergleich zu anderen Algorithmen wird der IBDD - Drifterkennungsalgorithmus für eine Überwachung der Eingangsdaten verwendet.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Überwachung der Ausgangsdaten

Die Überwachung der Ausgangsdaten ist im Vergleich zur Überwachung der Eingangsdaten einfacher, da es sich bei den Ausgangsdaten um einen eindimensionalen Wert handelt. Aus diesem Grund können hierfür sämtliche in Abschnitt 2.7.1 genannten Überwachungsmethoden angewendet werden. Um auch regionale Drifts erkennen zu können, wird eine Driftüberwachung der Ausgangsdaten mithilfe von kdq-Trees implementiert. Es wird die größte Distanz innerhalb der mit kdq-Trees unterteiltenTrainingsdaten herangezogen und diese mit der Verteilung der Produktionsdaten verglichen. Ist diese höher, als die Verteilung der Trainingsdaten, so wird ein Drift innerhalb der Ausgangsdaten detektiert. Wurde ein Drift erkannt, so wird dieser in der Azure Monitor – Umgebung ausgegeben und es erfolgt eine automatisierte Benachrichtigung an die verantwortlichen Personen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4 Praktische Implementierung

Aufbauend auf der in Kapitel 3 definierten Umsetzungsbeschreibung wird in diesem Kapitel die praktische Implementierung der vorliegenden Arbeit bei der *Thyssenkrupp Presta Steering* dokumentiert, so dass die in Abschnitt 1.2 definierten Forschungsfragen beantwortet werden können. Die Implementierung startet mit dem Aufbau der vorgeschlagenen Architektur. Anschließend wird die Erstellung eines Services zur Anomalieerkennung nach dem in Abschnitt 2.2 vorgestellten ML-Lebenszyklus durchgeführt. Es werden in jedem Schritt die notwendigen Tätigkeiten vorgestellt, um den Algorithmus in die Produktion zu überführen. Die ersten Schritte des ML-Lebenszyklus werden kurz abgehandelt, da für die vorliegende Arbeit ein bestehender Algorithmus zur Anomalieerkennung herangezogen wird. Der letzte Schritt des ML-Lebenszyklus, welcher die Bereitstellung des ML-Modells in der Produktionsumgebung umfasst, wird anschließend detailliert beschrieben.

4.1 Implementierung der vorgeschlagenen Architektur

In einem ersten Schritt wird die in Abschnitt 3.2 vorgeschlagene Architektur implementiert. Der Kern der Architektur besteht aus der bereits im Unternehmen verwendeten *Azure DevOps* – Umgebung, welche zur Verwendung als MLOps – Plattform mit weiteren Diensten verknüpft wird. Es wird eine Verbindung zwischen der *Azure DevOps* Umgebung und der *Azure ML* – Umgebung benötigt. Zudem wird eine Verbindung zum Data Lake benötigt, um die Produktionsdaten für die Modellexperimente heranziehen zu können. Zur Durchführung der Modellexperimente mit Hilfe von Jupyter Notebooks in der *Azure ML* – Umgebung werden zudem diverse GIT – Befehle angewendet.

Implementierung der Azure ML- Umgebung

Die *Azure ML*-Umgebung wurde bislang nicht im Unternehmen verwendet, weshalb diese im Rahmen der vorliegenden Arbeit implementiert werden muss. Dazu wird eine Ressourcengruppe innerhalb der *Azure* – Umgebung erstellt, die sämtliche notwendigen Ressourcen innerhalb der *Azure* – Umgebung organisiert [103]. Beispielhaft dafür kann der im nächsten Schritt definierte *Azure ML* Arbeitsplatz, sowie die dazugehörige *Azure Monitor* – Umgebung genannt werden.

Basierend auf der Implementierung der Ressourcengruppe wird anschließend der Machine Learning Arbeitsplatz erstellt. Innerhalb des Arbeitsplatzes werden sämtliche für die Erstellung des Machine Learning Services notwendigen Artefakte gespeichert und analysiert [104]. Zudem bietet der Arbeitsplatz die Möglichkeit diverse Berechtigungen neu zu vergeben, sodass lediglich eine genau definierte Personengruppe einen Zugang zu den relevanten Informationen besitzt.

Darauf aufbauend erfolgt die Implementierung einer cloudbasierten Compute Instanz, die für die Durchführung der Aufträge innerhalb der *Azure ML* – Umgebung benötigt wird [105]. Es können unterschiedlichste Compute Instanzen mit jeweils unterschiedlichen Spezifikationen

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

ausgewählt werden. Für die vorliegende Arbeit wird die Compute Instanz „STANDARD_E2A_V4“ herangezogen, die einen guten Kompromiss zwischen Preis und Rechenleistung liefert. Nun steht die *Azure ML* – Umgebung zur vollen Verwendung bereit.

Verbindung zwischen Azure DevOps und Azure ML

Die Verbindung zwischen der *Azure DevOps* - Umgebung und der *Azure ML* – Umgebung wird durch die Definition einer Service Verbindung (*engl. Service Connection*) innerhalb der *Azure DevOps* Umgebung erstellt. Mit Hilfe der Service Verbindung können definierte Befehle innerhalb der *Azure Pipeline* an die *Azure ML* – Umgebung gestellt und von dieser ausgeführt werden.

Zur Implementierung der Service Verbindung wird das bestehende *Azure* Abonnement sowie die bei der Erstellung der *Azure ML* – Umgebung definierte Ressourcengruppe und der Machine Learning Arbeitsplatz benötigt. Somit besteht eine aktive Verbindung zwischen *Azure DevOps* und *Azure ML*, welche in den nächsten Schritten benötigt wird.

Verwendung von Jupyter Notebooks in der Azure ML Umgebung

Zur Vereinfachung des *Python* Bibliothek Managements, sowie zur zentralisierten Speicherung des Codes in einem zentralisierten Codespeicher empfiehlt es sich die notwendigen Experimente innerhalb der *Azure ML*-Umgebung im Bereich Notebooks durchzuführen.

Hierbei kann das bereits angelegte Repository innerhalb der *Azure DevOps* Umgebung mit einem GIT-Befehl in den Bereich geklont werden und steht anschließend zur Durchführung der notwenigen Experimente bereit. Mit weiteren GIT-Befehlen kann der veränderte Code in das bestehende Repository integriert werden.

Verbindung zum Data Lake

Zur Lösung der vorliegenden Problemstellung durch die notwendigen Modell Experimente wird ein Zugang zu den abgelegten Produktionsdaten benötigt. Dazu wird ein Service Client herangezogen, welcher basierend auf interaktiven Browser Anmeldeinformationen die Berechtigung zum Data Lake Zugriff überprüft. Mit Hilfe des Service Clients kann auf den Data Lake zugegriffen werden und die notwendigen Produktionsdaten werden anschließend zur weiteren Bearbeitung heruntergeladen.

Weitere Verbindungen

Sämtliche weitere Verbindungen, wie beispielsweise zu *Azure Monitor* oder dem *Azure Machine Learning* – Data Lake sind bereits innerhalb der *Azure* Ressourcengruppe integriert und benötigen keine weitere Betrachtung bei der Implementierung der Architektur. Somit ist die Architektur bereit zur Implementierung des ML-Services.

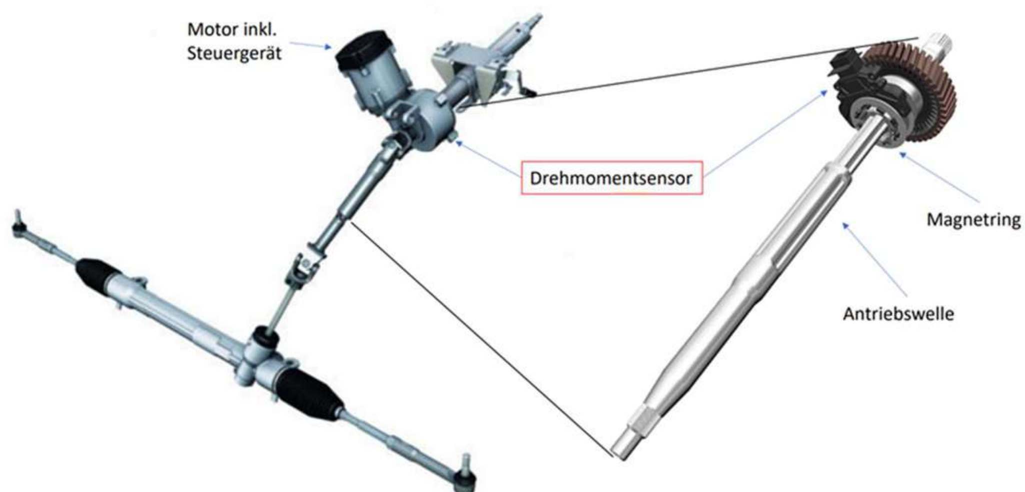
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

4.2 Definition der Modellanforderungen

Im Rahmen der Arbeit soll ein ML-Modell gefunden werden, welches in der Lage ist auf Basis von historischen Prozessdaten fehlerhafte Produktionsdaten eines Einpressvorganges von Drehmomentsensoren auf eine Antriebswelle des Lenkgetriebes in Echtzeit zu erkennen.

Der Vorgang stellt einen der kritischsten Prozessschritte innerhalb des gesamten Montageprozesses einer elektrischen Servolenkung (*engl. Electronic Power Steering (EPS) – System*) dar. Der aufgepresste Drehmomentsensor erfasst den Lenkeinschlag des EPS-Systems und leitet diesen an das EPS-Steuermodul zurück. Mit Hilfe dieser erfassten Kenngröße unterstützt anschließend der elektrische Servomotor die Lenkbewegung. Erfolgt der Einpressvorgang des Drehmomentsensors nicht einwandfrei, so kann die Funktionsweise des gesamten EPS-Systems gestört sein. Die endgültige Position des Drehmomentsensors auf der Antriebswelle ist in Abbildung 4-1 dargestellt.

Abbildung 4-1: Einbauposition eines Drehmomentsensors im Lenkstrang (Thyssenkrupp Presta Steering)



Um einen einwandfreien Aufpressvorgang bei ausgelieferten EPS-Systemen zu garantieren, werden die resultierenden Prozessdaten basierend auf manuell definierten, statischen Parametern analysiert. Eine Software führte eine Transformation der Rohdaten in eine Kraft-Weg Kurve durch und berechnete innerhalb manuell definierter Zonen die relevanten Werte. Dies sind beispielsweise die minimale und maximale Kraft innerhalb der jeweiligen Zone. Bei einer Überschreitung bzw. Unterschreitung des zuvor definierten Schwellwertes werden die Kurven als Anomalie gekennzeichnet und die betreffende Baugruppe wird aus dem Produktionsprozess entfernt. Durch die manuelle Definition der Zonen sowie der Schwellwerte ist die bisherige Vorgehensweise nicht für eine dynamische Anpassung ausgelegt und es besteht die Gefahr, dass bedeutende Kurveneigenschaften außerhalb der Zonen womöglich nicht erfasst werden. Dieser Umstand soll mit Hilfe eines Anomalieerkennungsalgorithmus behoben werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Dazu soll ein bestehendes Modell zur Anomalieerkennung verwendet werden, welches bereits für andere Prozesse innerhalb des Unternehmens angewendet wird und sich im Bereich der Fehlererkennung von Produktionsdaten bewährt hat. Das Modell basiert auf einem Autoencoder (*vgl. Abschnitt 2.1.5*) Ein Autoencoder ist für diese Anforderung sehr gut geeignet, da dieser gut mit der hohen Dimensionalität der zu überwachenden Datensätze umgehen kann. Die hohe Dimensionalität der Prozessdaten basiert auf der hohen Anzahl an einzelnen Datenpunkten, welche für die Dokumentation der jeweiligen Prozesskurven benötigt werden. Besteht eine Prozesskurve beispielsweise aus 1000 einzelnen Datenpunkten, so hat der entsprechende Datensatz eine Dimensionalität von 1000.

Projektimplementierung

Neben der Definition der Modellanforderungen wird zudem in diesem ersten Schritt auf Basis des in Abschnitt 3.3 vorgeschlagenen Erstellungsprozesses ein neues Projekt im zentralen Codespeicher angelegt, um sämtliche relevanten Daten abzuspeichern. In der vorliegenden Arbeit wird dafür *Azure DevOps Repos* verwendet. In Abbildung 4-2 ist die verwendete Ordnerstruktur dargestellt. Die Ordnerstruktur soll für jedes Projekt nach einem einheitlichen Schema aufgebaut sein, um eine gute Übersichtlichkeit zu ermöglichen. Dazu werden die Unterordner des Projektes in zwei Kategorien eingeteilt, welche durch den ersten Zeichenblock erkennbar sind. Die Dateien mit der Kennzeichnung „*auto*“ sind für das Ablegen von relevanten Dateien zur automatisierten Bereitstellung des ML-Modells vorgesehen, während die Dateien mit der Kennzeichnung „*man*“ für Dateien vorgesehen sind, die in der manuellen Experimentierphase, sowie in der anschließenden manuellen Testphase benötigt werden.

Abbildung 4-2: generische Ordnerstruktur des ML-Projektes

```
Projektname
  |_ auto_Deployment
  |_ auto_Package_requirement
  |_ auto_Tests
  |_ auto_Training
  |_ man_Data
  |_ man_Experiments
  |_ man_Test_deployment
```

Im Ordner „*auto_Deployment*“ sind sämtliche Dateien untergebracht, die für die automatisierte Bereitstellung des ML-Services benötigt werden. Innerhalb des Ordners wird die „*score.py*“ Datei gespeichert. Die Namensgebung ist auf die englische Bezeichnung „*scoring data*“ zurückzuführen, dass die Verarbeitung von neuen Daten durch ein bereits trainiertes Modell bezeichnet [106]. Diese Datei ist der Kern des ML-Services, da sie die ML-Pipeline definiert. Die „*scoringConfigStaging.yml*“ Datei definiert die Softwareumgebung innerhalb des für den ML-Service erstellten Containers, sodass die einzelnen Prozessschritte des ML-Services einwandfrei durchlaufen werden können. Die Datei „*aciDeploymentConfig.yml*“ definiert die genaue Spezifikation des Containers, der zur Bereitstellung des ML-Services benötigt wird. Hierbei bezieht sich die Bezeichnung „aci“ auf eine *Azure Container Instanz*. Des Weiteren liegt in dem Ordner die Datei „*inferenceConfig.yml*“ als Konfigurationsdatei für den Bereitstellungsbefehl ab. Hierbei bezeichnet „*inference*“ die Anwendung eines trainierten ML-Modells auf neue Daten.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Im Ordner „auto_*Package_requirement*“ befinden sich Dateien, die für die Installation der notwendigen Softwarepakete innerhalb der CI-Pipeline und CT-Pipeline benötigt werden. In der Datei „*requirements.txt*“ sind die zu installierenden Pakete aufgelistet. Die Datei „*install_requirements.sh*“ ist für die Ausführung des Installationsbefehls vorgesehen und beinhaltet die Basisbefehle zur Installation der Pakete in der jeweiligen Umgebung.

Im Ordner „*auto_Tests*“ sind die Testskripte abgelegt, welche im Rahmen der Pipelines automatisiert durchlaufen werden. Die einzelnen Testskripte werden in einem späteren Abschnitt beschrieben. Grundsätzlich wird für jeden Test innerhalb des ML-Lebenszyklus ein einzelnes Testskript erstellt, um eine Abkapselung der einzelnen Testdurchlaufe und somit eine verbesserte Fehlersuche zu erreichen.

Im Ordner „*auto_Training*“ sind sämtliche Dateien gespeichert, die für das automatisierte Training des ML-Modells benötigt werden. Unter dem Namen „*autoencoder_1500.py*“ ist das Modell selbst abgespeichert, welches anschließend trainiert werden soll. Mit der Datei „*train_machine-data.runconfig*“ wird der für das Training benötigte Container definiert. Die Datei „*conda_dependencies.yml*“ beinhaltet sämtliche zur Erstellung der Trainingsumgebung innerhalb der Azure Container Instanz notwendigen Bibliotheken. Die Datei „*train_aml.py*“ beinhaltet Befehle, die zur Dokumentation des Trainingslaufes in *Azure ML* benötigt werden. Die Datei „train.py“ enthält rein die zum Training des Modells benötigten Befehle. Die Datei „*train_test.py*“ beschreibt die durchgeführten Datentests, bevor die Daten für das Training verwendet werden.

Der Ordner „*man_Data*“ beinhaltet die für die Experimentierphase aufbereiteten Datensätze, welcher für das initiale Training des ML-Modells herangezogen wird.

Im Ordner „*man_Experiments*“ befindet sich der Code, der für die Experimentierphase benötigt wird. Dies umfasst beispielsweise den Code zur Aufbereitung der Daten sowie den Code zur Durchführung der einzelnen Modellexperimente und der anschließenden Validierung.

4.3 Datenerfassung

Der gesamte Aufpressvorgang eines Drehmomentsensors auf die Antriebswelle eines Lenkgetriebes wird durch eine Kraft-Weg Kurve dargestellt und soll mit Hilfe des ML-Modells überwacht werden. Dazu werden historische Kraft-Weg Kurven des Aufpressvorganges herangezogen und zur weiteren Analyse verwendet.

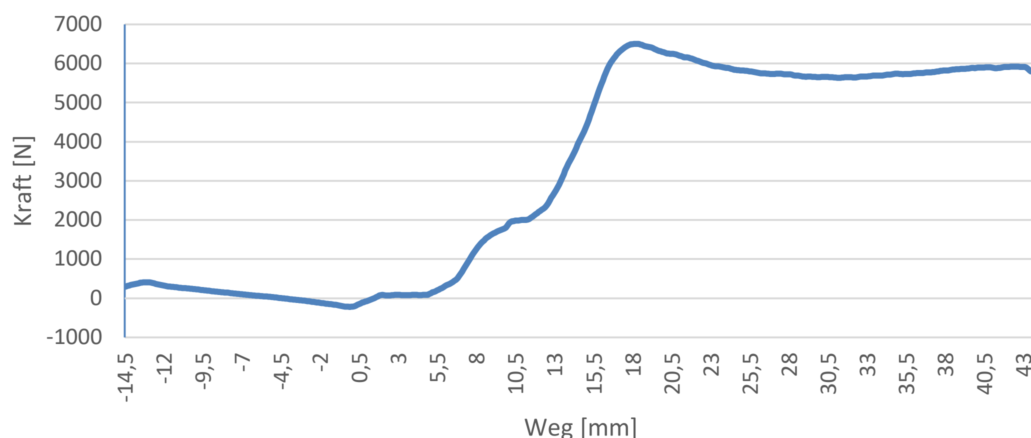
Die Rohdatenkurven liegen komprimiert als zpg-Dateien auf den Data Lakes der jeweiligen Werke vor, in denen der Aufpressprozess durchgeführt wird. Hierbei wird innerhalb der Data Lakes unterschieden, ob es sich um einwandfreie Prozesskurven (OK), oder fehlerhafte Prozesskurven (NOK) handelt. Die Rohdaten sind in einer JSON-ähnlichen Struktur innerhalb der zpg-Dateien abgespeichert, wobei neben den Kurvendaten noch weitere relevante Merkmale der Kurve, wie beispielsweise Maschinennummer, genaue Fertigungszeit und Standort erfasst werden. Mit Hilfe eines bestehenden Scripts können die komprimierten Rohdaten heruntergeladen, entschlüsselt und für die weitere Datenverarbeitung lokal abgespeichert werden. Für das Training des Autoencoders werden nur einwandfreie Prozesskurven herangezogen. Anschließend werden die relevanten Bereiche innerhalb der abgespeicherten zpg-Dateien extrahiert, um lediglich die Kraft-Weg-Kurve als Datensatz zu erhalten.

TU Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die Datenpunkte der extrahierten Rohdaten weisen keine konsistenten Abschnitte und Anzahl auf, wodurch die Erstellung eines Trainingsdatensatzes direkt aus den extrahierten Rohdaten nicht möglich ist. Aus diesem Grund werden die Kurvendaten in einem nächsten Schritt kubisch interpoliert, um eine zuvor definierte Anzahl an Datenpunkten mit gleichen Schrittweiten zu erhalten.

Die interpolierten Daten können anschließend zu einem Trainingsdatensatz zusammengefügt werden. Daraus kann eine Kraft-Weg-Kurve wie in Abbildung 4-3 entnommen werden. Hierbei ist ersichtlich, dass der Weg in der x- und die Kraft in der y-Achse angegeben ist. Die Aufnahme der Kraft-Weg-Kurve beginnt bei einem Abstand von -14,5 mm und endet bei einem Abstand von 43 mm. Im Bereich von -14,5 mm bis 0mm wirkt eine geringe wechselnde Kraft auf den Kraftsensor. Diese kann durch die Beschleunigung und anschließende Abbremsung des Drehmomentsensors vor dem Einpressvorgang erklärt werden. Ab einem Abstand von 0 mm wechselt die Kraft in den positiven Bereich, bevor ab 5,5 mm die Kraft stetig bis zum Maximalwert bei etwa 18 mm zunimmt. Bis zum Ende der Kraftaufnahme bleibt die Kraft konstant etwas unter dem Maximalwert bestehen. In diesem Zeitraum findet der Großteil des Aufpressvorganges statt, bei dem zuerst der höhere Haftreibungswiderstand überwunden werden muss, bevor sich ein etwas geringerer Gleitreibungskoeffizient einstellt. Der letzte Abschnitt der Kraft-Weg-Kurve wird im Rahmen der Aufbereitung nicht weiter betrachtet, da hier keine konsistenten Rohdaten existieren. Zudem ist der letzte Abschnitt nicht für die Qualität des Aufpressvorganges entscheidend.

Abbildung 4-3: Kraft-Weg Kurve eines als OK eingestuften Kraftverlaufes



4.4 Datenbereinigung

Um die Qualität der Trainingsdaten hochzuhalten, wird anschließend eine Sichtkontrolle der Daten durchgeführt, um mögliche grobe Fehler innerhalb der Trainingsdaten erkennen zu können. Wie in Abschnitt 4.3 bereits beschrieben, werden für das Training des ML-Modells nur fehlerfreie Kurven herangezogen, um den Rekonstruktionsfehler des Autoencoders bei diesen Kurven so gering als möglich zu erzeugen. Treten optisch fehlerhafte Kurven innerhalb des

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Trainingsdatensatzes auf, so werden diese manuell aus dem Trainingsdatensatz entfernt. Anschließend kann der Trainingsdatensatz für die weiteren Schritte herangezogen werden.

4.5 Datenbeschriftung & Feature Engineering

Da es sich bei dem vorhandenen ML-Modell um einen Autoencoder handelt, der keine Beschriftung der Daten benötigt, kann dieser Schritt übersprungen werden. Im gesamten ML-Lebenszyklus gehört dieser Schritt zu den aufwändigsten Schritten, weshalb hierbei klar der Vorteil eines Autoencoders im Vergleich zu anderen ML-Modellen erkennbar ist. Auch ein Feature Engineering, bei dem die Relevanz der einzelnen Merkmale eines Datensatzes untersucht wird und mögliche irrelevante Merkmale weggelassen werden, wird für Autoencoder nicht benötigt.

4.6 Modellerstellung & Modelltraining

Das in der vorliegenden Arbeit verwendete Modell basiert auf einem bereits bekannten Modell, weshalb hierbei lediglich auf den Adaptierungsprozess und nicht auf die Erstellung des Modells selbst eingegangen wird.

Der Autoencoder besteht aus einem Encoder, der die Kurvendaten in eine latente Repräsentation komprimiert. Darauf aufbauend wird die latente Repräsentation mit Hilfe eines Decoders rekonstruiert. Da es sich bei dem vorliegenden Modell um ein bereits funktionierendes Modell handelt, muss zur Anwendung für den vorliegenden Fall lediglich die Anzahl der Eingangsneuronen an die Dimensionalität des Datensatzes angepasst werden.

Um das Modell auf die vorliegenden Kurvendaten zu trainieren, wird ein zweistufiger Ansatz gewählt. In einem ersten Schritt wird der Autoencoder mit den zuvor erstellten Trainingsdaten trainiert und der Rekonstruktionsfehler wird bestimmt. Anschließend werden die Kurven mit einem Rekonstruktionsfehler im 90. Perzentil aussortiert, um mögliche, im Rahmen der Datenbereinigung, nicht erkannten Anomalien ausschließen zu können. Das Modell wird mit den verbliebenen Trainingsdaten neu trainiert und der Schwellwert für die Anomalieerkennung wird beim Wert des 90. Perzentils angenommen. Dieser Schwellwert wird in das Modell übernommen, wodurch das Modell für die Anomalieerkennung zur Verfügung steht.

Zur Veranschaulichung des Einflusses der Trainingsdatengröße wird das Modell mit 4.000 Kurven sowie mit 8.000 Kurven trainiert. Der Trainingsprozess wird für beide Trainingsdatensätze durchgeführt und das daraus generierte Modell wird an einem zuvor manuell erstellten Testdatensatz geprüft. Die Ergebnisse sind in Tabelle 4-1 dargestellt. Es ist klar ersichtlich, dass durch das Training des Modells mit dem bereinigten Datensatz die Anzahl an richtig erkannten Anomalien stark gesteigert werden konnte, während die Anzahl an falsch detektierten Anomalien gleichbleibt, bzw. sich sogar verringert.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

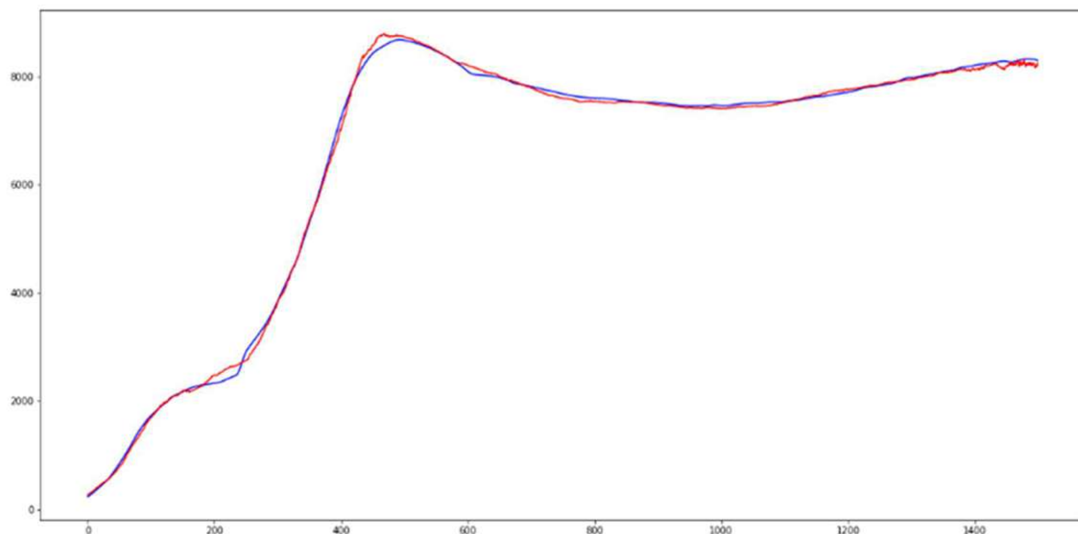
Tabelle 4-1: Vergleich der beiden Trainingsvorgänge

	True Positives	True Negatives	False Positves	False Negatives
4.000 Kurven				
Einfaches Training	4	2	1	3
Doppeltes Training	4	5	1	0
8.000 Kurven				
Einfaches Training	4	2	1	3
Doppeltes Training	5	5	0	0

4.7 Modellbewertung

Im nächsten Schritt erfolgt die Bewertung der Performance des gewählten Modells anhand eines Testdatensatzes. Dafür wird derselbe Testdatensatz, wie in Abschnitt 4.6 herangezogen, weshalb Tabelle 4-1 für die Modellbewertung verwendet werden kann. Es ist ersichtlich, dass sämtliche Vorgehensweisen eine gute Leistung liefern, wobei das doppelte Training mit einem Trainingsdatensatz von 8.000 Kurven die beste Performance leistet. Aus diesem Grund kann die Leistungsfähigkeit des gewählten Modells bestätigt werden und es kann mit der Operationalisierung des ML-Modells begonnen werden, indem das trainierte und gepackte Modell lokal abgespeichert wird. Abbildung 4-4 veranschaulicht die Leistung des trainierten Autoencoders, indem eine reale Kraft-Weg Kurve mit einer rekonstruierten Kraft-Weg Kurve verglichen wird.

Abbildung 4-4: Reale Kraft-Weg Kurve (blau), sowie durch den Autoencoder rekonstruierte Kraft-Weg Kurve (rot)



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.8 Modellbereitstellung

Der vorletzte Schritt des Machine Learning Lebenszyklus ist die Bereitstellung des trainierten Modells in der Produktionsumgebung und der Anwendung von MLOps für dieses Projekt. Dieser Schritt selbst ist sehr umfangreich, weshalb dieser nachfolgend weiter in die einzelnen Schritte des Bereitstellungsprozesses unterteilt wird.

4.8.1 Automatisiertes Training vorbereiten

Das zuvor definierte Training, welches im Rahmen der Modellexperimente durchgeführt wurde, wird in diesem Schritt für die automatisierte Trainingspipeline übernommen. Dafür müssen sämtliche im Ordner „*auto_training*“ abgelegten Dateien an den Trainingsprozess angepasst werden. Innerhalb der Datei „*conda_dependencies.yml*“ werden die für das Training notwendigen Softwarepakete adaptiert, während die Datei „*train_machinedata.runconfig*“ die allgemeine Umgebung definiert und in der Regel nicht weiter angepasst werden muss. Die Datei „*train.py*“ organisiert die einzelnen Schritte, die für das Modelltraining benötigt werden, wodurch hier die meisten Anpassungen notwendig sind. In der Datei „*train_aml.py*“ wird eine Anpassung der zu referenzierenden Bezeichnungen benötigt.

4.8.2 Modell in serverlose Funktion konvertieren

Innerhalb des bereits angelegten Ordners „*auto_Deployment*“ werden die Dateien zur Erstellung der serverlosen Funktion gespeichert. Dies betrifft die „*score.py*“-Datei, welche als Kern der serverlosen Funktion gesehen werden kann. Darin erfolgt die Anpassung der „*init*“-Funktion, welche für das Laden des Modells aus der *Azure ML* – Umgebung benötigt wird. Des Weiteren kommt es zu einer Anpassung der „*run*“-Funktion, welche zur Ausführung der ML-Pipeline benötigt wird. Die „*run*“ Funktion definiert die genaue Vorgehensweise zur Erstellung einer Modellvorhersage, sowie die Tests im Rahmen der kontinuierlichen Überwachung. In dem vorliegenden Fall definiert die „*run*“-Funktion die Übernahme der Daten aus der API-Schnittstelle, die Vorverarbeitung der Daten in eine für das Modell lesbare Form, sowie die Erstellung einer Vorhersage durch das Modell. Anschließend werden die notwendigen Ausgabedaten definiert. Der letzte Schritt protokolliert die notwendigen Überwachungsparameter und übermittelt die Modellantwort an den Sender der Eingangsdaten. Die „*scoringConfig*“ – Datei definiert die dafür notwendige Softwareumgebung, welche an die benötigten Softwarepakete angepasst werden muss.

4.8.3 Lokales Testen der serverlosen Funktion

Im nächsten Schritt wir die erstellte Funktion, sowie die gesamte Bereitstellung der Funktion lokal getestet, um eine schnelle und unkomplizierte Fehlerbehebung gewährleisten zu können. Hierfür wird der bereits erstellte Ordner „*man_test_deployment*“ benötigt. In diesem Ordner liegen sowohl Testdaten für das Testen des bereitgestellten Services, als auch das gepackte ML-Modell, welches im Rahmen des Tests manuell bereitgestellt wird. Zudem ist in diesem Ordner in der Datei „*test_deployment.ipynb*“ der Code zur manuellen lokalen Bereitstellung des ML-Services abgelegt. Treten Fehler bei der Ausführung auf, so müssen die für die Bereitstellung

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

notwendigen Dateien angepasst werden. Wurden sämtliche Fehler im Rahmen der lokalen Bereitstellung behoben, so kann der ML-Service als zusätzlicher Test in einer Azure Container Instanz bereitgestellt werden. Der dafür notwendige Code ist ebenfalls in der Datei „*test_deployment.ipynb*“ abgelegt.

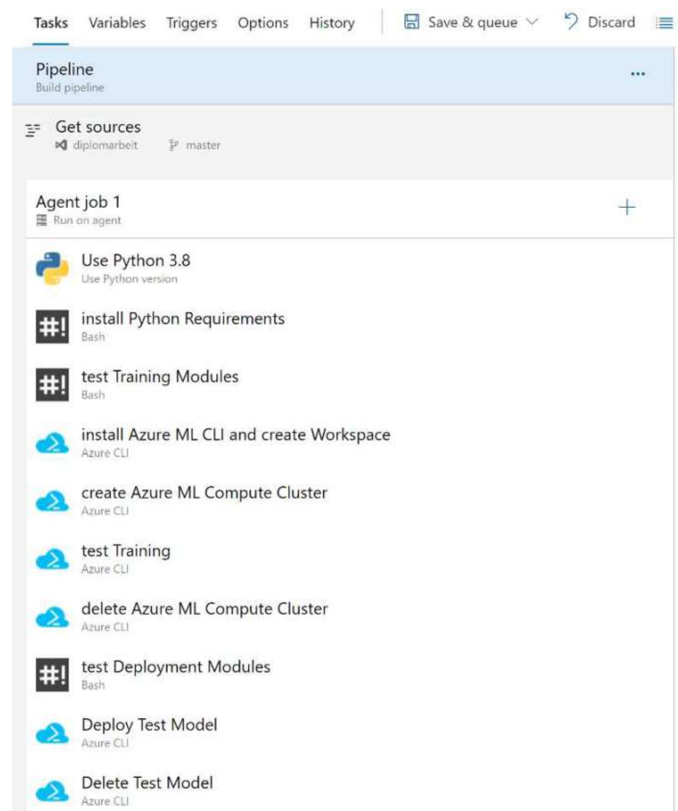
4.8.4 Erstellen eines Pull Requests & Code Review

Treten auch bei der Bereitstellung des ML-Services in der Container Instanz keine Fehlermeldungen auf, so kann anschließend ein Pull Request durch die Entwickler_innen gestellt werden. Auf Grund der Erstellung des Pull Request bekommen andere Entwickler_innen im Rahmen des Code Reviews eine automatisierte Benachrichtigung per Mail, um die Qualität des entwickelten Codes innerhalb der *Azure DevOps* Umgebung zu prüfen und gegebenenfalls Änderungsvorschläge einzubringen. Kommt es zu Änderungsvorschlägen, so müssen die Dateien der serverlosen Funktion angepasst und diese anschließend lokal getestet werden.

4.8.5 Durchführung der CI - Pipeline

Der Kern der CI-Pipeline ist das Testen der definierten Schritte innerhalb der Trainingspipeline, sowie der Module des ML-Services. Für die Durchführung der CI-Pipeline wird die Funktion *Pipelines* innerhalb der *Azure DevOps* - Umgebung herangezogen. In Abbildung 4-5 sind die einzelnen Schritte der kontinuierlichen Integrationspipeline dargestellt.

Abbildung 4-5: Ablauf der kontinuierlichen Integrationspipeline innerhalb der Azure DevOps Umgebung



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die CI-Pipeline wird nach einem erfolgreichen Abschluss des Code Reviews gestartet, und die in *Abschnitt 2.6* definierten Testfälle werden durchlaufen. Die Testfälle werden in der Datei „*test_integration.py*“ innerhalb des Ordners „*auto_tests*“ definiert. Für den vorliegenden Fall werden einerseits Modultests, sowie ein automatisierter Aufbau des ML-Services getestet. Erfolgen die Integrationstests ohne Fehler, so steht anschließend die getestete Codebasis für die automatisierte Trainingspipeline zur Verfügung.

Testfall 1: Modultests im Trainingsablauf

Der erste Abschnitt der kontinuierlichen Integrationspipeline analysiert die korrekte Programmierung der einzelnen Module. In diesem Schritt wird das Preprocessing der Trainingsdaten getestet, indem bereits bekannte Daten dem Preprocessing zugeführt werden und das Ergebnis mit dem erwarteten Ergebnis verglichen wird.

Testfall 2: Integrationstest des Trainingsablaufs

Anschließend erfolgt die Durchführung eines gesamten Trainingsdurchlaufes, in einem Compute Cluster. Mit Hilfe des Integrationstests kann die korrekte Angabe der benötigten Bibliotheken getestet werden. Der Trainingsdurchlauf selbst wird mit einer geringen Anzahl an Trainingsdaten durchgeführt und das Ergebnis des Trainings wird anschließend analysiert.

Testfall 3: Modultests der Modellbereitstellung

In diesem Schritt wird die korrekte Verarbeitung der Eingangsdaten des ML-Services überprüft. Dabei wird der für die Verarbeitung vorgesehene Code mit einer kleinen Anzahl an Testdaten ausgeführt, und das anschließende Ergebnis wird mit dem zu erwarteten Ergebnis verglichen. In dem vorliegenden Fall werden die Eingangsdaten als JSON-Datei übergeben und anschließend in einen *Python*-Array umgewandelt. Im Rahmen des Testfalls wird die Dimensionalität des Arrays auf Richtigkeit überprüft. Zudem wird das Auftreten von fehlerhaften Werten innerhalb des Arrays untersucht.

Testfall 4: Integrationstests der Modellbereitstellung

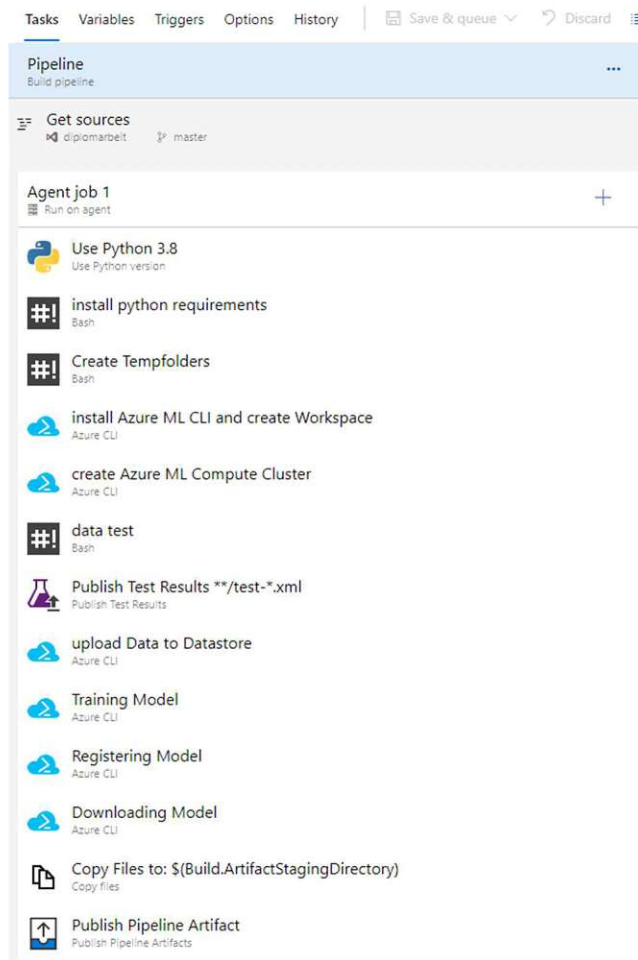
Zum Schluss wird die automatisierte Bereitstellung eines bereits trainierten Modells getestet. Erfolgen sämtliche Schritte ohne Fehlermeldung, so steht die Codebasis für das kontinuierliche Training zur Verfügung.

4.8.6 Kontinuierliches Training

Wurde die Codebasis für das Training, sowie die Bereitstellung des ML-Services in den *master*-Zweig überführt, so kann anschließend die automatisierte Trainingspipeline erstellt werden, um ein automatisiertes Neutraining zu ermöglichen. Dazu wird ebenfalls eine Pipeline in der *Azure DevOps* – Umgebung erstellt. Für die Erstellung der CT – Pipeline müssen die einzelnen Schritte der Pipeline, sowie zusätzliche Variablen und der gewählte Trigger für den Pipelinestart definiert werden. Als Betriebssystem für die CT – Pipeline wird die aktuellste *Ubuntu* - Version definiert. Die Pipeline wird nicht automatisiert gestartet, sondern bei Bedarf manuell aktiviert, sodass ein Neutraining nur dann möglich ist, wenn dieses auch explizit gewünscht ist.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-6: Ablauf der kontinuierlichen Trainingspipeline innerhalb der Azure DevOps Umgebung



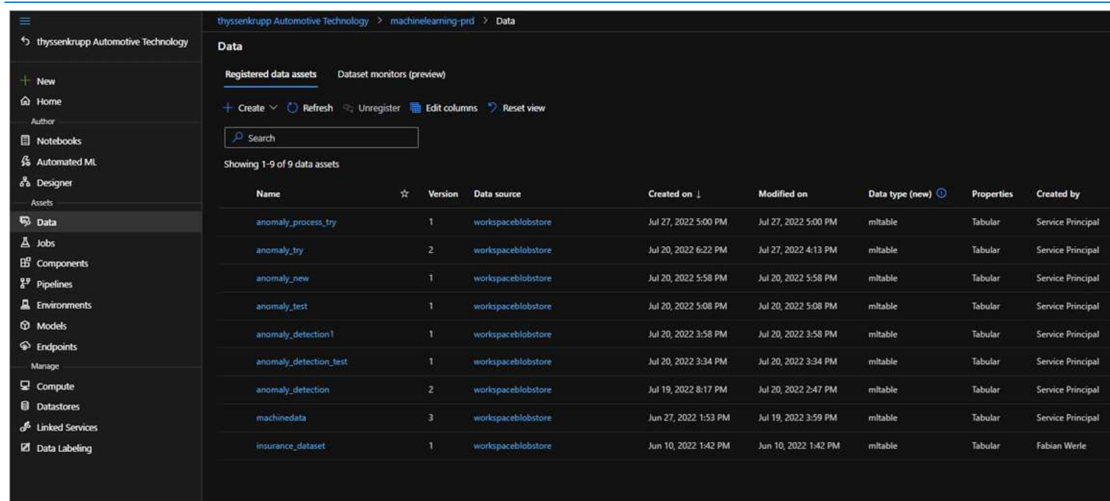
Der Ablauf der CT – Pipeline beginnt mit der Definition der Python Version, sowie der Installation der Softwarevoraussetzungen. Anschließend werden drei temporäre Ordner zur Speicherung der Trainingsdaten, Metadaten sowie der Modelldaten erstellt. Diese Ordner werden im letzten Schritt der CT - Pipeline veröffentlicht, sodass die kontinuierliche Bereitstellungspipeline darauf zurückgreifen kann. Im nächsten Schritt wird innerhalb der Pipeline die *Azure ML* – Erweiterung installiert, um spezifische Befehle für *Azure ML* verwenden zu können und es wird der Arbeitsplatz definiert, in dem die CT – Pipeline das Training durchführt. Anschließend wird ein Compute-Cluster innerhalb der *Azure ML* – Umgebung erstellt, der für die Ausführung des Modelltrainings verantwortlich ist. Mit diesem Schritt ist die Initialisierung der Umgebung beendet, und es kann mit den relevanten Schritten innerhalb der Pipeline gestartet werden.

Der erste relevante Schritt beinhaltet einen Test der Trainingsdaten, um die Qualität der Daten gewährleisten zu können. Der für den Test verwendete Code ist im Ordner „*auto_training*“ in der Datei „*train_test.py*“ abgespeichert. Die Validierung beinhaltet nach Abschnitt 2.6.1 eine Überprüfung der Daten auf das Vorhandensein von überproportional vielen Null Werten und NaN-Werten, sowie die Überprüfung, ob die Trainingsdaten in einem vorgegebenen Wertebereich liegen. Die Resultate des Tests werden veröffentlicht, so dass sie jederzeit innerhalb der

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

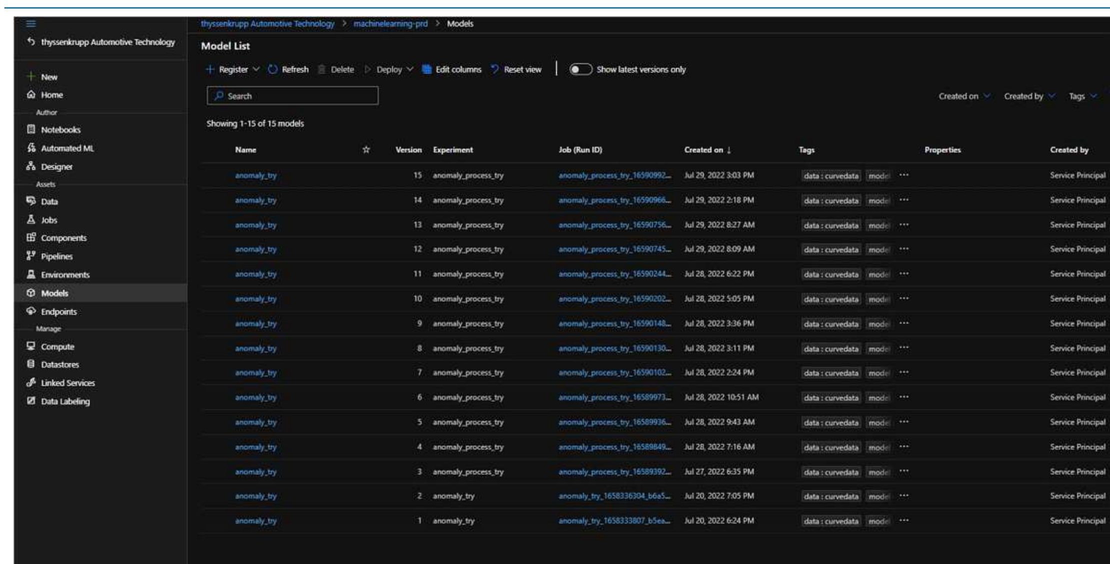
Azure DevOps – Umgebung überprüft werden können. Anschließend werden die Daten in die *AzureML* – Umgebung hochgeladen, um eine Rückverfolgbarkeit der Daten zu gewährleisten. In Abbildung 4-7 sind die registrierten Datensätze in *Azure ML* dargestellt.

Abbildung 4-7: Registrierte Datensätze in der Azure-ML Umgebung



Der zweite relevante Schritt ist das Training des Modells durch den zuvor definierten Compute Cluster. Dafür müssen innerhalb des Compute Clusters die Softwarevoraussetzungen installiert werden, bevor mit dem Training begonnen werden kann. Die Dokumentation des Trainingsablaufes wird innerhalb des temporären Metadaten – Ordners abgespeichert. Darauf aufbauend wird das Modell ebenfalls in der *Azure ML* Umgebung registriert, um die Nachverfolgbarkeit zu gewährleisten. Das trainierte ML Modell wird zudem innerhalb des temporären Modellordners gespeichert.

Abbildung 4-8: Registrierte Modelle in der Azure-ML Umgebung



Im letzten Schritt werden die temporären Dateien veröffentlicht, so dass sie für die kontinuierliche Bereitstellungspipeline genutzt werden können.

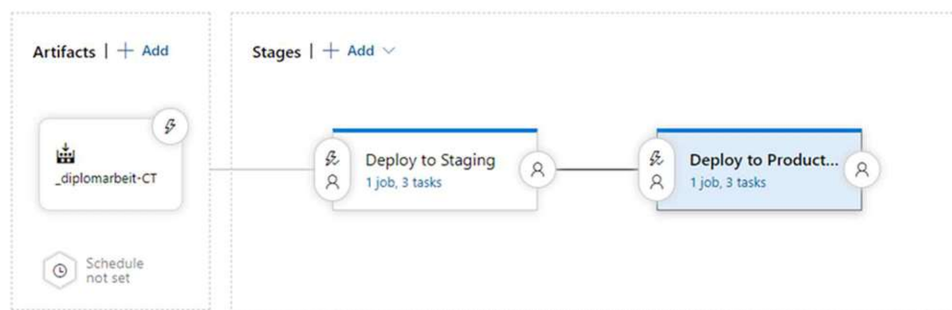
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Somit sind nach Durchlauf der CT – Pipeline sämtliche relevanten Dateien innerhalb der *Azure ML* – Umgebung gespeichert und das trainierte ML Modell wurde für die automatisierte Bereitstellung abgespeichert.

4.8.7 Kontinuierliche Bereitstellung des trainierten Modells

Für die Bereitstellung des trainierten Modells wird die Funktion *Releases* innerhalb der *Azure DevOps* Umgebung verwendet. Diese greift auf die bereits durch die CT - Pipeline veröffentlichten Artefakte zu und veröffentlicht den Service in einer produktionsähnlichen Umgebung, bevor der Service anschließend in der Produktionsumgebung veröffentlicht wird. Sowohl die Veröffentlichung in der produktionsähnlichen Umgebung als auch in der Produktionsumgebung wird manuell durch eine zuvor definierte Personengruppe gestartet, um unbewusste Veröffentlichungen zu unterbinden. Die Veröffentlichung in beiden Umgebungen verläuft nach dem gleichen Ablauf, jedoch unter anderem Namen. Auf Grund des gleichen Ablaufs wird dieser einmal erklärt.

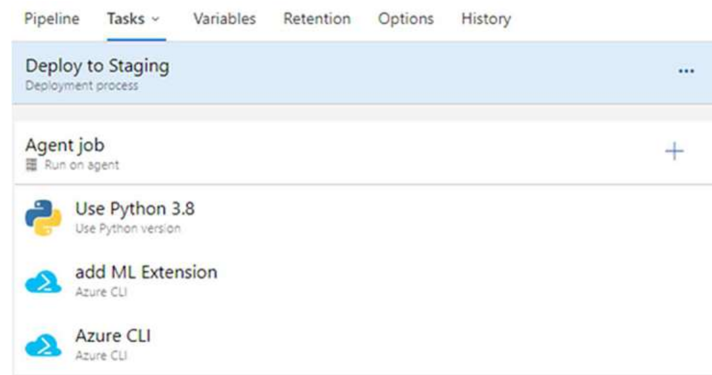
Abbildung 4-9: Stufen der kontinuierlichen Bereitstellungspipeline innerhalb der Azure DevOps Umgebung



Als Betriebssystem für die CD-Pipeline wird die aktuellste *Ubuntu* – Version definiert. Im ersten Schritt wird zudem die angewendete Python Version festgelegt. Anschließend erfolgt innerhalb der Pipeline die Installation der *Azure ML* Erweiterung. Im letzten Schritt wird das Modell über die *Azure ML* Umgebung in einer *Azure Compute Instanz* bereitgestellt. Die Spezifikationen der Azure Compute Instanz sind im Ordner „*auto_deployment*“ innerhalb der Datei „*aciDeploymentConfigStaging.yml*“ definiert. Nach erfolgreichem Abschluss des Schrittes steht der Service in der jeweiligen Umgebung zur Verfügung und kann beliebig angesteuert werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-10: Ablauf der kontinuierlichen Bereitstellungspipeline innerhalb der Azure DevOps Umgebung zur Bereitstellung in der produktionsähnlichen Umgebung



Es wird empfohlen die Bereitstellung des Modells in der Produktionsumgebung mit Hilfe von *Kubernetes* durchzuführen. Durch die Verwendung von *Kubernetes* ist eine Orchestrierung von mehreren Containern möglich, wodurch die Leistungsfähigkeit des Services weiter gesteigert werden kann. Im Rahmen dieser Arbeit wird aus Kostengründen sowie zur Vereinfachung der Arbeit auf eine Implementierung mit Hilfe von Kubernetes verzichtet. *Azure* bietet einen integrierten Kubernetes Service an, welcher ähnlich wie eine *Azure Container Instanz* im Rahmen der Bereitstellungspipeline angesteuert werden kann.

4.8.8 Testen des bereitgestellten Services

Nach der Bereitstellung des ML – Services in der produktionsähnlichen Umgebung wird der Service anschließend getestet. Es werden wie in Abschnitt 3.4.3 beschrieben zwei unterschiedliche Tests durchgeführt.

Der erste Test analysiert anhand von Beispieldaten eine einwandfreie Bearbeitung dieser durch das gesamte ML-System. Es werden zuvor definierte Kurvendaten dem Service übergeben und die zurückbekommene Antwort wird auf Plausibilität überprüft.

Der zweite Test vergleicht die Performance des vorhandenen ML-Systems in der Produktion mit dem ML-System in der Testumgebung und wird auch A/B Test genannt. Dazu empfängt der ML-Service bereits Daten aus der Produktionsumgebung und verarbeitet diese. Die Ergebnisse werden verglichen und in einem Dashboard bereitgestellt. Anschließend erfolgt eine Benachrichtigung an die verantwortliche Personengruppe zur manuellen Überprüfung der Ergebnisse.

4.9 Modellveröffentlichung und Überwachung

Verlaufen die Ergebnisse des Modelltests zufriedenstellend, dann wird der Service in die Produktionsumgebung überführt. Die Überführung in die Produktionsumgebung erfolgt nach der bereits in Abschnitt 4.8.7 erklärten Vorgehensweise. Mit diesem Schritt empfängt der Service reale Daten aus der Produktion und bewertet diese. Um die Qualität des bereitgestellten Services kontinuierlich garantieren zu können, werden unterschiedliche Überwachungsmaßnahmen getroffen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.9.1 Kontinuierliche Überwachung des Modells

Zur kontinuierlichen Überwachung von ML-Modellen und im speziellen Fall im Bereich der Driftüberwachung haben sich bereits unterschiedliche Programme und Services etabliert, welche herangezogen werden können. Jedoch sind diese nicht für die Drifterkennung von hochdimensionalen Daten ausgelegt und fokussieren sich zudem zu einem großen Teil auf kategorische Daten, wodurch diese nicht für den vorliegenden Anwendungsfall geeignet sind. Zudem bieten sämtliche bestehende Lösungen keine Integration in die bestehende Architektur. Aus diesem Grund wurde die Überwachung des Modells mit Hilfe einer eigenen Lösung implementiert, die nachfolgend vorgestellt wird.

In Abbildung 4-11 ist der gewählte Ablauf zur Überwachung des Modells dargestellt. Die Überwachung ist in den ML-Service integriert und speichert nur bei erkannten Auffälligkeiten die relevanten Werte in die Azure Monitor Umgebung.

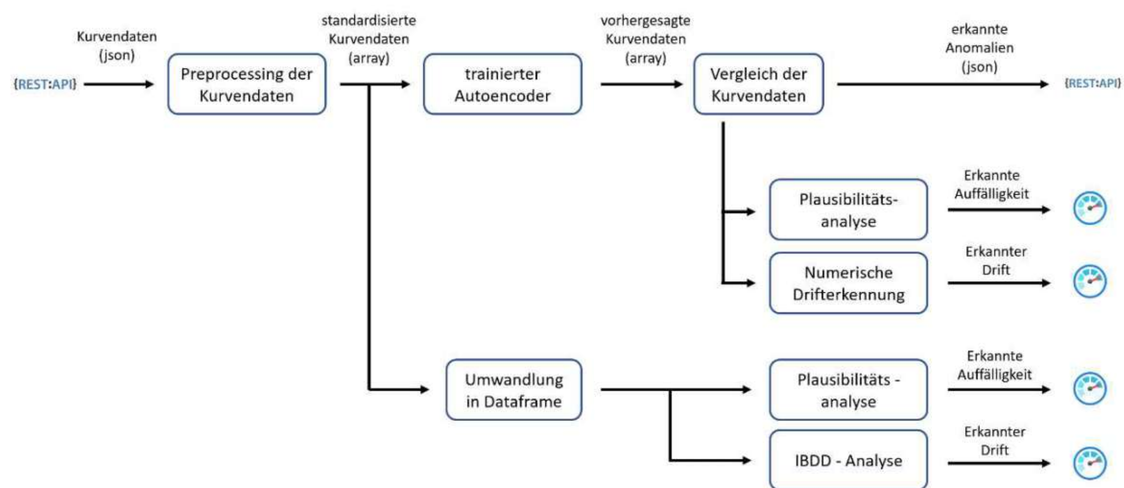
Der Prozess beginnt mit den, mittels REST API, an den ML-Service übergebenen Kurvendaten, die als json – Datei übergeben werden. Anschließend erfolgt ein Preprocessing der Kurvendaten, um diese in ein Array umzuwandeln, und zu standardisieren. Die standardisierten Kurvendaten werden an den trainierten Autoencoder weitergegeben, der diese rekonstruiert. Durch einen Vergleich der Kurvendaten und das Bilden des Rekonstruktionsfehlers können Anomalien erkannt werden, die mittels REST API wieder an den Sender der erhaltenen Kurvendaten zurückgesendet werden.

Die Überwachung der Eingangsdaten erfolgt durch das Abgreifen der standardisierten Kurvendaten. Diese werden anschließend in einen Dataframe umgewandelt und sowohl der Plausibilitätsanalyse als auch der in Abschnitt 2.7.1 vorgestellten bildbasierten Drifterkennung (IBDD) zugeführt. Wurde eine Auffälligkeit bzw. ein Drift erkannt, so wird diese anschließend an die *Azure Monitor* Umgebung übergeben.

Die Überwachung der Ausgangsdaten erfolgt durch das Abgreifen der Differenz zwischen den standardisierten Kurvendaten und den vorhergesagten Kurvendaten. Diese wird sowohl der Plausibilitätsprüfung als auch der numerischen Drifterkennung zugeführt und es erfolgt die Überwachung der Ausgangsdaten. Wurde eine Auffälligkeit bzw. ein Drift erkannt, so wird diese an die Azure Monitor Umgebung übergeben.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-11: Darstellung der Überwachung des ML-Modells



Nachfolgend wird detailliert die Implementierung der Überwachung der Eingangsdaten, sowie der Ausgangsdaten beschrieben.

Überwachung der Eingangsdaten

Die Überwachung der Eingangsdaten wird in eine Plausibilitätsanalyse, sowie die Drifterkennung eingeteilt. Die Plausibilitätsprüfung überprüft die Werte auf das Vorkommen von Nullwerten bzw. NaN – Werten und es wird untersucht, ob sich die Werte innerhalb zuvor definierter Grenzen bewegen. Im vorliegenden Fall werden die beidseitigen Grenzen der standardisierten Kurvendaten mit einem Schwellwert von 5 angenommen, so dass extreme Ausreißer innerhalb der Kurvendaten erkannt werden können. Werden Ausreißer detektiert, so wird je Ausreißer ein Zähler aktiviert. Überschreitet der Zähler einen zuvor definierten Schwellwert, wird eine Meldung an die Überwachung weitergeleitet und der Zähler wird wieder auf null gesetzt. Der Schwellwert des Zählers wird mit 10 Ausreißern festgelegt.

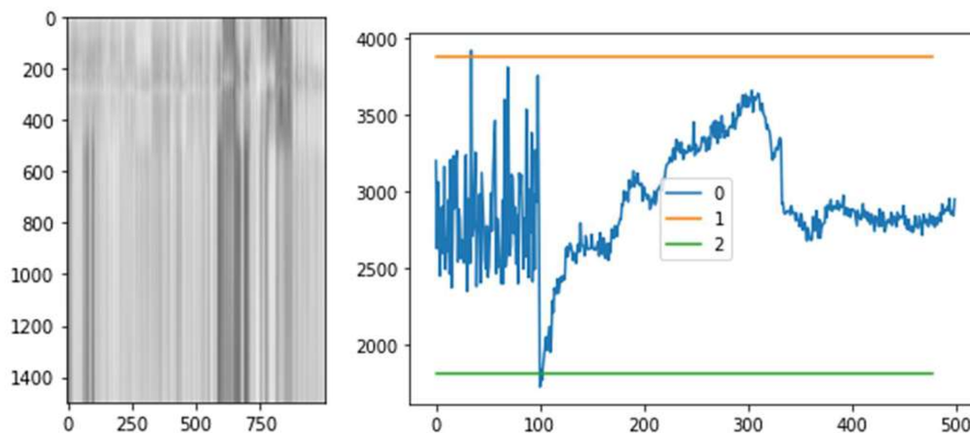
Die Drifterkennung wird auf Basis des IBDD - Drifterkennungsalgorithmus durchgeführt, der in Abschnitt 2.7.1 detailliert erklärt wird. Die Drifterkennung übernimmt die in einen DataFrame umgewandelten standardisierten Kurvendaten und wertet diese aus. Zur Ermittlung der Grenzen für die Drifterkennung wird ein Anfangsschwellwert definiert. Dazu wird insgesamt 100-mal die Verteilung innerhalb von zufällig generierten Kombination von Trainingskurven ermittelt. Die Anfangsschwellwerte werden anschließend mit dem Mittelwert der Verteilungen zuzüglich, bzw. abzüglich der zweifachen Standardabweichung definiert. Damit können 90% der im Test vorkommenden Verteilungswerte abgedeckt werden. Wurde ein Drift erkannt, so wird dieser anschließend an die Azure Monitor – Umgebung weitergeleitet.

In Abbildung 4-12 ist beispielhaft auf der linken Seite ein generiertes Bild der Kurvendaten dargestellt. Das Bild besteht aus insgesamt 1.500 vertikalen Pixeln, welche jeweils einem Datenpunkt der Kurve entsprechen, sowie aus 1.000 horizontalen Pixel, die der Anzahl der Kurven entsprechen. Auf der rechten Seite ist der Verlauf der Verteilungen dargestellt. Die orange Linie entspricht dem oberen Schwellwert und die grüne Linie dem unteren Schwellwert. Es ist er-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

kennbar, dass in den ersten 100 Datenpunkten hohe Abweichungen vorherrschen. Diese entsprechen den Abweichungen in der Verteilung der zufällig generierten Kombinationen an Testdaten. Die anschließenden Werte sind die aus den Produktionsdaten enthaltenen Werte. Hierbei ist erkennbar, dass die Produktionsdaten keinen Datendrift aufweisen.

Abbildung 4-12: Ergebnisse des IBDD – Drifterkennungsalgorithmus, mit dem zu analysierenden Bild (links) und dem Output (rechts)



Überwachung der Ausgangsdaten

Die Überwachung der Ausgangsdaten des Modells deckt mehrere Aspekte ab. Die Ausgangsdaten werden kontinuierlich auf Nullwerte und nicht plausible Werte überprüft. Des Weiteren wird auch der Mittelwert der Ausgangsdaten kontinuierlich gebildet und auf grobe Abweichungen hin untersucht. Zudem erfolgt eine Überwachung des Drifts durch die in Abschnitt 2.7.1 beschriebenen Metriken.

Die Plausibilitätsprüfung der Ausgangsdaten erfolgt mit Hilfe einer Erkennung von Nullwerten sowie bei Überschreiten eines zuvor gesetzten Schwellwertes. Als Schwellwert wird der Wert 1 herangenommen, welcher nur bei extremen Abweichungen vorkommt. Werden Ausreißer detektiert, so wird je Ausreißer ein Zähler aktiviert. Überschreitet der Zähler einen zuvor definierten Schwellwert, wird eine Meldung an die Überwachung weitergeleitet und der Zähler wird wieder auf null gesetzt. Der Schwellwert des Zählers wird mit 10 Ausreißern festgelegt.

Die Analyse des Mittelwerts wird durch Bilden des Mittelwerts der vergangenen 100 Datenpunkte durchgeführt. Überschreitet der Mittelwert das 1.5-fache des höchsten Trainingsmittelwerts, so wird ebenfalls ein Zähler aktiviert. Überschreitet der Zähler einen zuvor mit 5 Ausreißern definierten Schwellwert, so wird eine Meldung an die Überwachung weitergeleitet und der Zähler wird auf null gesetzt.

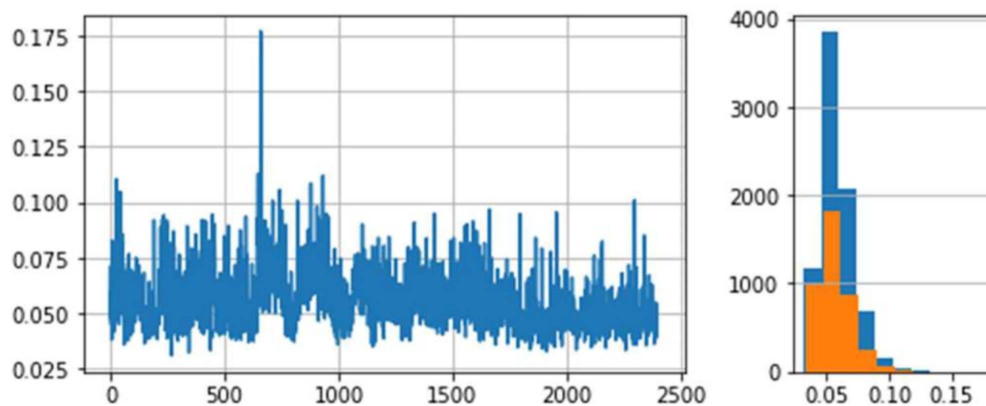
Des Weiteren werden die Ausgangsdaten auf eine Veränderung der Verteilung überwacht, die einen Datendrift darstellen kann. Hierbei wird die in Abschnitt 2.7.1 beschriebene Vorgehensweise, sowie die unterschiedlichen Metriken zur Driftüberwachung herangezogen. Mit Hilfe von kdq-Trees werden die Trainingsdaten in unterschiedliche Bereiche zu jeweils 100 Datenpunkten unterteilt, um auch Distanzen innerhalb der Trainingsdaten erfassen zu können. Die höchste Distanz innerhalb der Trainingsdaten zuzüglich einem Puffer von 10% entspricht anschließend

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

dem Schwellwert zur Drifterkennung. Wird dieser Schwellwert für die Produktionsdaten überschritten, so wird ein Drift detektiert. Zur Verstärkung der Robustheit wird ein Drift nur dann an die Überwachung weitergeleitet, wenn dieser von mehreren Metriken gleichzeitig erkannt wird.

Nachfolgend ist ein Beispiel von Ausgangsdaten dargestellt, bei dem die ersten 8.000 Datenpunkte aus dem Training entstanden sind und die nächsten 4.500 Datenpunkte aus der Produktion entnommen wurden. Auf der rechten Seite sind die Histogramme der Trainingsdaten, sowie der Produktionsdaten aufgezeichnet. Hierbei kann kein starker Unterschied in der Verteilung erkannt werden.

Abbildung 4-13: Darstellung von Ausgangsdaten des ML-Services, wobei die ersten 8000 Datenpunkte die Trainingsdaten und die nächsten 4500 die Produktionsdaten darstellen



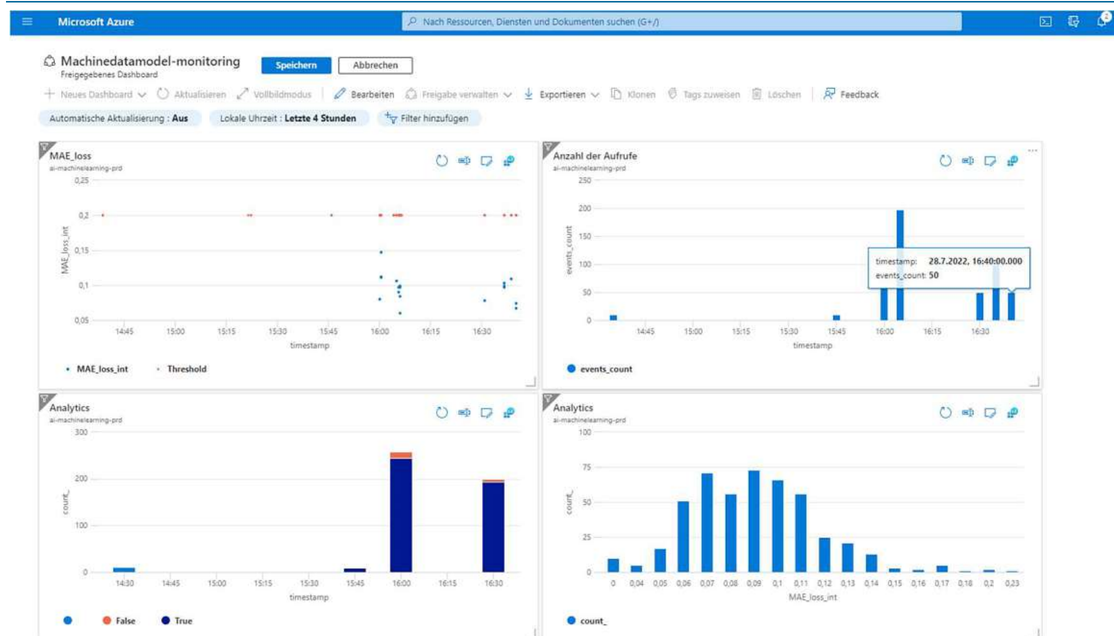
Die Produktionsdaten werden mit den sieben Metriken Kullback-Leibler Divergenz, Manhattan (MH) Distanz, quadrierte euklidische Distanz, Chebyshev Distanz, Kosinusdistanz und Bhattacharyya Distanz zur Drifterkennung auf einen Datendrift untersucht. Der dafür verwendete Code wurde auf Basis von Poenaru-Olaru et al. [37] implementiert und für den vorliegenden Fall angepasst. Für das vorliegende Beispiel wurde durch die Kullback-Leibler Divergenz regelmäßig ein Drift innerhalb der Daten erkannt, während die anderen Metriken keinen Drift detektierten. Somit kann die Kullback-Leibler Divergenz als sehr sensitiv angesehen werden. Dies steht in einem Widerspruch zur Arbeit von Poenaru-Olaru et al. [37], in der die Kullback-Leibler Divergenz die geringste Falsch-Positiv Rate aufweist. Dieser Unterschied kann mit den unterschiedlichen Daten erklärt werden.

Auswertung der Überwachungsdaten

Wird eine Meldung an die Überwachung getätigt, indem beispielsweise ein Zähler einen Schwellwert überschreitet, oder ein Drift innerhalb der Daten erkannt wird, so wird dieser in der Azure Monitor – Umgebung geloggt. In dieser Umgebung werden die Daten unstrukturiert gespeichert, weshalb sie anschließend mit Hilfe der Kusto query language (KQL), einer SQL-ähnlichen Programmiersprache, für die einzelnen Diagramme ausgewertet werden. Anschließend können innerhalb der *Azure Monitor* Umgebung Regeln für die einzelnen Diagramme erstellt werden, um automatisierte Benachrichtigungen zu senden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-14: Beispielhafter Aufbau einer individuell programmierten Azure Monitor - Umgebung zur Modellüberwachung

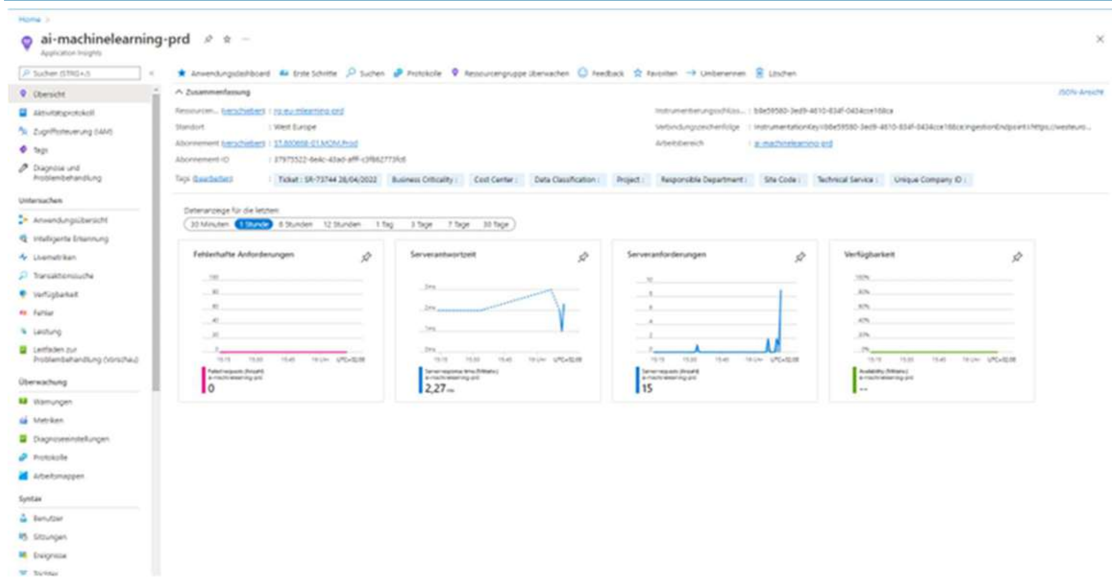


4.9.2 Kontinuierliche Überwachung des Systems

Ein weiterer Aspekt ist die Überwachung des gesamten ML-Services auf diverse Unstimmigkeiten. Im Rahmen der Arbeit wird die Überwachung des ML-Services mit Hilfe der *Azure Application Insights* realisiert. Diese übernimmt automatisiert die Überwachung des Systems, indem die Serverantwortzeiten, sowie die Anzahl der Überwachungen analysiert werden. Auch hier können spezifische Regeln definiert werden, bei denen es zu einer anschließenden automatisierten Meldung an eine zuvor definierte Personengruppe kommt. Somit kann die gesamte Überwachung des Systems gewährleistet werden.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

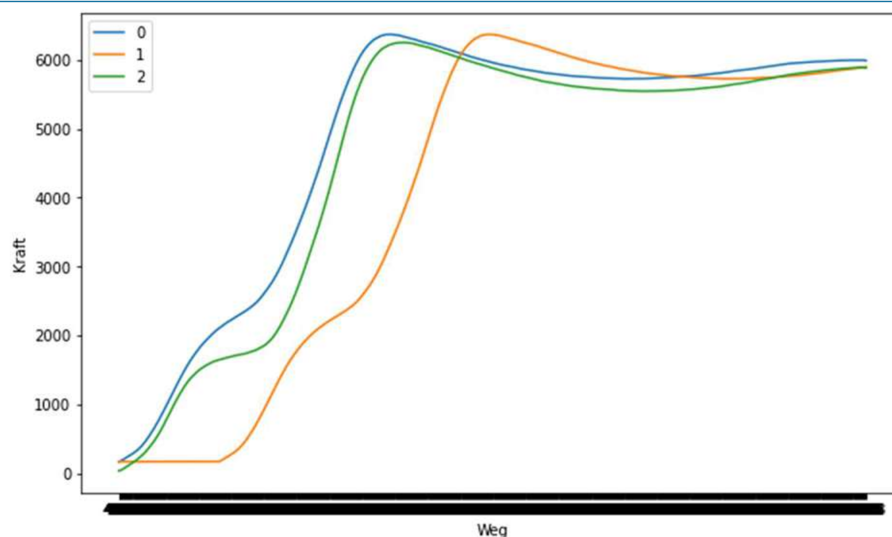
Abbildung 4-15: Aufbau der Azure Application Insights – Umgebung, in der die Überwachung des Systems erfolgt



4.9.3 Analyse der Effektivität der verwendeten Drifterkennung

Zum Testen der vorliegenden Drifterkennung wurden die Drifterkennungen mit zwei unterschiedlichen Arten an Kurvendaten getestet und anschließend analysiert. Im ersten Schritt wurden den Kurven, welche vier Monate später erzeugt wurden auf einen Drift analysiert. Im zweiten Schritt wurden künstlich angepasste Kurvendaten analysiert, die einen verspäteten Kraftanstieg aufweisen. Dieser verspätete Kraftanstieg kann beispielsweise bei einer verkürzten Aufpresslänge gesehen werden, welche durch die Einführung eines neuen Produkts mit einer solchen Spezifikation in den zu überwachenden Maschinen erfolgen kann.

Abbildung 4-16: Unterschiedliche Kurven in der Analyse (Mittelwerte)

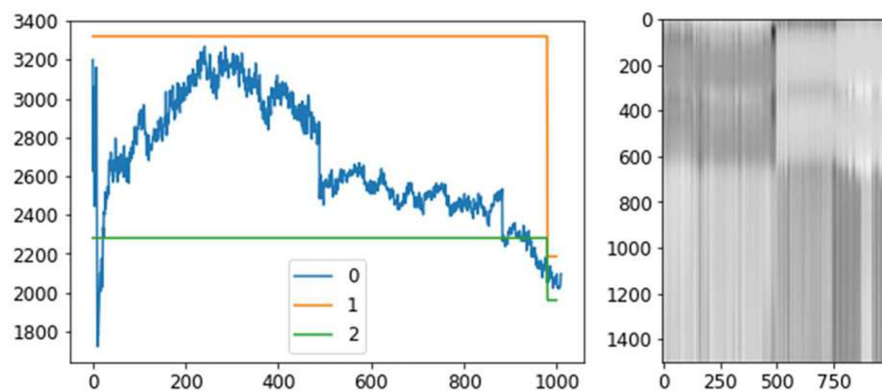


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In Abbildung 4-16 sind die drei unterschiedlichen Arten an Kurven jeweils dargestellt. Kurve 2 beschreibt eine durchschnittliche Trainingskurve (5000) Kurven. Kurve 0 beschreibt eine durchschnittliche Kurve (5000 Kurven) des gleichen Prozesses, die etwa 4 Monate später entstanden ist. Hierbei ist erkennbar, dass diese Kurve ein weniger stark ausgeprägtes Plateau innerhalb des Kraftanstiegs aufweist. Kurve 1 entspricht der angepassten Kurve, mit einem verspäteten Kraftanstieg, basierend auf den 4 Monate später entstandenen Kurven. Die unterschiedlichen Arten an Kurvendaten wurden sowohl durch die Eingangsdatenanalyse als auch durch die Ausgangsdatenanalyse auf einen vorliegenden Drift untersucht.

In Abbildung 4-17 ist das Ergebnis der Eingangsdatenanalyse dargestellt. Hierbei wurde die Grundverteilung, sowie die Grenzwerte zur Drifterkennung mit den Trainingsdaten des Modells definiert. Anschließend wurde der Drifterkennungsalgorithmus auf die später erzeugten Kurven, sowie die angepassten Kurven angewendet. In der linken Abbildung markieren die ersten 500 Datenpunkte die später erzeugten Produktionsdaten und die nächsten 500 Datenpunkte markieren die künstlich angepassten Produktionsdaten. Es ist erkennbar, dass die später erzeugten Produktionsdaten innerhalb der zuvor definierten Grenzen liegen, und somit kein Drift erkannt wird. Bei den künstlich angepassten Produktionsdaten fällt die Bildverteilung abrupt ab, sodass diese am unteren Rand liegt. Durch eine anschließende weitere Veränderung der Eingangsdaten wird ein Drift innerhalb der Daten erkannt. In der rechten Abbildung ist zudem der Wechsel zwischen den beiden Arten der Produktionsdaten erkennbar.

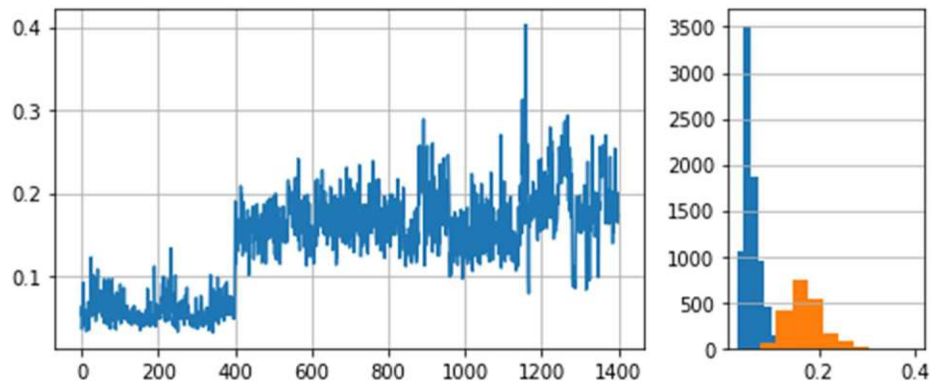
Abbildung 4-17: Ergebnisse der Eingangsdatenanalyse auf spätere Produktionsdaten, sowie künstlich angepasste Produktionsdaten



Die gleichen Daten werden anschließend der Ausgangsdatenanalyse zugeführt, welche zuvor durch das trainierte Modell verarbeitet werden. In Abbildung 4-18 ist das Ergebnis der Ausgangsdatenanalyse dargestellt. Die ersten 400 Punkte bilden die Trainingsdaten ab, die nächsten 500 Punkte stellen die 4 Monate später erstellten Produktionsdaten dar und die letzten 500 Punkte stellen die angepassten Produktionsdaten dar. Es ist gut ersichtlich, dass die Performance des trainierten Algorithmus im Vergleich zu den Trainingsdaten abnimmt und der Rekonstruktionsfehler steigt welches allgemein als *Training-Serving Skew* verstanden wird. Jedoch kann auch gesehen werden, dass der Einfluss der manuellen Veränderung der Kurven keine signifikante Änderung in der Verteilung aufweist, jedoch die Maximalwerte der Rekonstruktionsfehler zunehmen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-18: Darstellung von Ausgangsdaten des ML-Services, wobei die ersten 8000 Datenpunkte die Trainingsdaten und die nächsten 4500 die Produktionsdaten darstellen



Aus den Ergebnissen der Drifterkennung können folgende Rückschlüsse gezogen werden. Durch die Eingangsdatenüberwachung wurde ein Drift innerhalb der Eingangsdaten erkannt, während die Ausgangsdatenüberwachung keinen Drift entdeckt hat. Wird dieser Umstand nach der in Abschnitt 2.2.2 beschriebenen Theorie analysiert, so wurde mittels der implementierten Driftüberwachung ein Datendrift detektiert, während ein Konzeptdrift ausgeschlossen wird. Dies deckt sich auch mit den Ergebnissen des ML-Modells, welches zwar eine veränderte Datenbasis erhält. Die Funktion des ML-Modells bleibt jedoch trotz der veränderten Datenbasis erhalten.

Aus diesem Grund kann gesagt werden, dass die implementierte automatisierte Drifterkennung funktioniert und eine konstante Überwachung des ML-Modells ermöglicht wurde.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 Fazit und Ausblick

In diesem Kapitel wird ein Fazit über die durchgeführte Arbeit gezogen und die Arbeit nach dem in Abschnitt 2.4.2 behandelten MLOps – Reifegradmodell bewertet. Anschließend wird ein Ausblick auf weitere Verbesserungsmaßnahmen gegeben, um den in dieser Arbeit verfolgten Ansatz weiter auszubauen.

5.1 Fazit

Der Untersuchungsgegenstand der vorliegenden Arbeit war die Operationalisierung eines ML-Modells. Diese wurde anhand eines Autoencoders, welcher Anomalien in Produktionsdaten erkennen soll, gezeigt.

Die Ergebnisse lassen erkennen, dass sowohl die Architektur als auch der vorgeschlagene Prozess für die Überführung eines Algorithmus zur Anomalieerkennungs in die Produktion geeignet ist. Mit Hilfe der Architektur konnte jeder Schritt des vorgeschlagenen Erstellungsprozesses abgearbeitet werden. Zudem bietet die Architektur die geforderte Nähe zu bisherigen Lösungen durch die komplette Verwendung von bisher genutzten Services und Anwendungen, welche innerhalb der Microsoft Azure Umgebung liegen. Somit konnte eine wichtige Unternehmensvorgabe erfüllt werden. Der Erstellungsprozess bietet klar definierte Schritte, um ein bestehendes Modell in die Produktion zu überführen und dieses anschließend zu überwachen. Zudem konnte mit Hilfe des vorgeschlagenen Überwachungsprozesses die Leistungsfähigkeit des gesamten ML-Services konstant überwacht und bewertet werden. Die Überwachung wurde mit Hilfe einer Eingangsdatenüberwachung, einer Ausgangsdatenüberwachung sowie einer Systemüberwachung realisiert. Sowohl die Eingangsdatenüberwachung als auch die Ausgangsdatenüberwachung wurde auf Basis von realen und angepassten Produktionsdaten überprüft und es konnte bewiesen werden, dass die Überwachung in der Lage ist, einen Drift innerhalb der Produktionsdaten zu erkennen. Basierend auf der Erkennung des Drifts kann anschließend das Modell neu trainiert werden, um die Leistungsfähigkeit des Systems konstant zu halten.

Des Weiteren soll der erstellte MLOps - Prozess nach dem MLOps – Reifegradmodell bewertet werden. Durch die Anwendung der in dieser Arbeit vorgestellten Architektur, sowie des vorgestellten Erstellungsprozesses kann der Reifegrad von der vorletzten Stufe (Stufe 1) des MLOps – Reifegradmodells auf die höchste Stufe (Stufe 4) gehoben werden. In dieser Stufe arbeiten Dateningenieur_innen und Softwareentwickler_innen gemeinsam an der Entwicklung von neuen ML-Services. Des Weiteren besteht eine automatisierte Trainingsumgebung, in der das ML-Modell automatisiert neu trainiert werden kann. Es kommt zu einer Versionisierung des Trainingscodes, der Trainingsdaten und der daraus resultierenden Modelle. Des Weiteren werden die Ergebnisse zentralisiert abgespeichert, sodass die Leistungen nachverfolgbar sind. Eine automatisierte Bereitstellungspipeline stellt das Modell in der Produktionsumgebung bereit und führt Tests durch. Eine konstante Überwachung des Modells in der Produktionsumgebung

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

ermöglicht ein gezieltes Neutraining des Modells bei Bedarf. Die dafür notwendigen Metriken werden zudem historisch abgespeichert und können jederzeit abgerufen werden.

5.2 Ausblick

Die Operationalisierung eines Algorithmus zur Anomalieerkennung stellt lediglich einen kleinen Teil von Machine Learning Algorithmen dar, die in Unternehmen verwendet werden können. Neben der Anomalieerkennung von Produktionskurven bieten sich auch weitere Anwendungsfälle für die Operationalisierung von Machine Learning Algorithmen an. Im Folgenden wird auf mögliche Erweiterungen der bestehenden Arbeit, sowie auf weiterführende Analysen eingegangen.

Implementierung einer Datenpipeline

In der vorliegenden Arbeit wurde zur Vereinfachung der Architektur auf eine Implementierung der Datenpipeline verzichtet. Die Implementierung der universellen Datenpipeline erfordert ein weitreichendes Wissen über die Datenanalyse, sowie weitere Services zur automatisierten Datenaufbereitung. Um das volle Potential der Architektur ausnutzen zu können, ist eine Datenpipeline notwendig.

Anwendung der vorgeschlagenen Konzepte im Unternehmensalltag

Die vorliegende Arbeit versteht sich als Proof of Concept in der Operationalisierung von Machine Learning Algorithmen zur Anomalieerkennung. Aus diesem Grund wurden die Implikationen der vorgeschlagenen Konzepte auf den Unternehmensalltag nicht untersucht. Vielmehr wurde der Fokus auf die Machbarkeitsanalyse einer Operationalisierung gelegt. Mit Hilfe des Feedbacks aus dem Unternehmensalltag können die vorgeschlagenen Konzepte weiter auf die Unternehmensanforderungen angepasst werden und es wird eine weitreichende Anwendung der Konzepte innerhalb des Unternehmens ermöglicht. Zudem können nach einem definierten Zeitraum die Auswirkungen der Konzepte auf die Unternehmensleistung gemessen werden.

Ausweitung der Anwendung auf weitere ML-Modelle

In dieser Arbeit wurde ein Autoencoder als ML-Modell zur Anomalieerkennung herangezogen, der unüberwacht auf die vorliegenden Produktionsdaten trainiert wird. Im Rahmen von weiterführenden Arbeiten kann untersucht werden, ob die vorliegende Architektur, sowie die Prozesse für weitere Arten von ML-Modellen geeignet sind. Zudem können Änderungsvorschläge gegeben werden, die eine Operationalisierung von weiteren Arten von ML-Modellen basierend auf der vorgeschlagenen Architektur ermöglichen. Ein weiterer Aspekt ist die Überwachung von unterschiedlichen ML-Modellen, die ebenfalls weiterführend analysiert werden kann.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Verzeichnisse

6.1 Literaturverzeichnis

[1] T. Weitzmann, “Understanding The Benefits And Risks Of Using AI In Business,” 2023. Accessed: Jul. 12, 2023. [Online]. Available: https://www.forbes.com/sites/forbesbusinesscouncil/2023/03/01/understanding-the-benefits-and-risks-of-using-ai-in-business/

[2] D. Sculley *et al.*, “Hidden Technical Debt in Machine Learning Systems,” 2015.

[3] A. White, “Gartner: Our Top Data and Analytics Predicts for 2019.” Accessed: Feb. 10, 2023. [Online]. Available: https://blogs.gartner.com/andrew_white/2019/01/03/our-top-data-and-analytics-predicts-for-2019/

[4] McKinsey, “The imperatives for automation success,” 2020.

[5] S. Makinen, H. Skogstrom, E. Laaksonen, and T. Mikkonen, “Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?,” in *Proceedings - 2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI, WAIN 2021*, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 109–112. doi: 10.1109/WAIN52551.2021.00024.

[6] Traeger M., Eberhart A., and Geldner G., “Grundsätzliche Funktions-beschreibung eines künstlichen neuronalen Netzwerks,” *Anaesthesist*, vol. 11, pp. 1055–1061, 2003, doi: 10.1007/s00101-003-0576-x.

[7] S. Strecker and A. Wi, “Künstliche Neuronale Netze-Aufbau und Funktionsweise CORE View metadata, citation and similar papers at core.ac.uk provided by Giessener Elektronische Bibliothek,” 1997, Accessed: Jul. 10, 2022. [Online]. Available: http://wi.uni-giessen.de

[8] J. Lawrence, *Neuronale Netze: Computersimulation biologischer Intelligenz*. Systhema-Verlag, 1992.

[9] S. S. Haykin, *Neural networks and learning machines*. Prentice Hall/Pearson, 2009.

[10] C. C. Aggarwal, “Neural Networks and Deep Learning,” 2018.

[11] R. Salakhutdinov and G. Hinton, “DEEP BELIEF NETWORKS,” 2007.

[12] H. Abdel-Jaber, D. Devassy, A. al Salam, L. Hidaytallah, and M. El-Amir, “A Review of Deep Learning Algorithms and Their Applications in Healthcare,” *Algorithms*, vol. 15, no. 2. MDPI, Feb. 01, 2022. doi: 10.3390/a15020071.

[13] K. Busch, “Unüberwachte Musterentdeckung in sequentiellen Daten miiels Autoencoder und Clustering Unsupervised Paaern Discovery in Sequential Data Using Autoencoder and Clustering.”

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[14] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, “A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science,” 2020, pp. 3–21. doi: 10.1007/978-3-030-22475-2_1.

[15] M. van Gerven and S. Bohte, “Editorial: Artificial neural networks as models of neural information processing,” *Frontiers in Computational Neuroscience*, vol. 11. Frontiers Media S.A., Dec. 19, 2017. doi: 10.3389/fncom.2017.00114.

[16] M. W. Libbrecht and W. S. Noble, “Machine learning applications in genetics and genomics,” *Nature Reviews Genetics*, vol. 16, no. 6. Nature Publishing Group, pp. 321–332, May 18, 2015. doi: 10.1038/nrg3920.

[17] “(Prentice Hall Series in Artificial Intelligence) Stuart Russell, Peter Norvig - Artificial Intelligence_ A Modern Approach-Prentice Hall (2010)”.

[18] H. U. Dike, Y. Zhou, K. K. Deveerasetty, and Q. Wu, “Unsupervised Learning Based On Artificial Neural Network: A Review,” in *2018 IEEE International Conference on Cyborg and Bionic Systems, CBS 2018*, Institute of Electrical and Electronics Engineers Inc., Jan. 2019, pp. 322–327. doi: 10.1109/CBS.2018.8612259.

[19] S. B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” 2007.

[20] Institute of Electrical and Electronics Engineers and IEEE Control Systems Society. Chapter Malaysia, *Proceedings, 2017 IEEE 13th International Colloquium on Signal Processing & Its Application (CSPA 2017) : 10-12 March 2017 : conference venue, Parkroyal Penang Resort, Batu Ferringhi Beach, 11100 Penang, Malaysia*.

[21] J. Qian, Z. Song, Y. Yao, Z. Zhu, and X. Zhang, “A review on autoencoder based representation learning for fault detection and diagnosis in industrial processes,” *Chemometrics and Intelligent Laboratory Systems*, vol. 231. Elsevier B.V., Dec. 15, 2022. doi: 10.1016/j.chemolab.2022.104711.

[22] M. Nilashi *et al.*, “Remote tracking of Parkinson’s Disease progression using ensembles of Deep Belief Network and Self-Organizing Map,” *Expert Syst Appl*, vol. 159, Nov. 2020, doi: 10.1016/j.eswa.2020.113562.

[23] “Effect of batch size on training dynamics | by Kevin Shen | Mini Distill | Medium.” Accessed: Jan. 28, 2023. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#id_token=eyJhbGciOiJSUzI1NiIsImtpZCI6IjI3NDA1MmEyYjY0NDg3NDU3NjRlNzJjMzU5MDk3MWQ5MGNmYjU4NWEiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL2FjY291bnRzLmdvb2dsZS5jb20iLCJuYmYiOjE2NzQ5MDQ2MTIsImF1ZCI6IjIxNjI5NjAzNTgzNC1rMWs2cWUwNjBzMnRwMmEyamFtNGxqZGNtczAwc3R0Zy5hcHBzLmdvb2dsZXVzZXJjb250ZW50LmNvbSIsInN1YiI6IjExMTY0NTM4NzIzMDM0MTExNzgxOSIsImVtYWlsIjoiZmFiaWFuLndlcmxlM0BnbWFpbC5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiYXpwIjoiMjE2Mjk2MDM1ODM0LWsxazZxZTA2MHMydHAyYTJqYW00bGpkY21zMDBzdH

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

RnLmFwcHMuZ29vZ2xldXNlcmNvbnRlbnQuY29tIiwi-
bmFtZSI6IkZhYmlhbiBXZXJsZSIsInBpY3R1cmUiOiJodH-
RwczovL2xoMy5nb29nbGV1c2VyY29udGV-
udC5jb20vYS9BRWRGVHA1blhyX2pOMy1sLUdleHJMWUFXTHlEdnNjbHBxdTlRUW
9uOVFlMT1zOTYtYylsImdpdmVuX25hbWUi-
OiJGYWJpYW4iLCJmYW1pbHlfbmFtZSI6IldlcmxlIiwiaWF0IjoxNjc0OTA0OTEyLCJleH
AiOjE2NzQ5MDg1MTIsImp0aSI6ImIz-
ZjhmMjk4ZGFlZDZjMmM1NzI2ZDY2OWIyOGJhYmE1ZGM3YmE0OWEifQ.qeove9djg
gDlRk8OrEHVKlwp7H-
yWeFZIBLuy0bH5xKVdVxesYn5JthT137_iH8g8k6_Abax6PQCdAJFZiZP0arhyMQ-
yoE1ceRsE5hqIk8_rQvu4jzJmiDCc0rmnLLuitj0Ss6-S2-FFIM0F-
iXK3vvnVC9M5wcmytxfrsygmIcfPp-
msJp2iE1b9on7Ud3J3BmCme9cXRINlHxvKkZ11DkvuNmklcOJHNXixKn-
cIoCjFxWfqaXY0SQGs1YDQSS0uTrywmub0212JWcaspucGr-
HG7__pc2N_7HOrrNsRKFRqbThog5ErafmMVS1Is65skau9w0xfT4HWx-i8k9eYw

[24] S. Shahinfar, P. Meek, and G. Falzon, "'How many images do I need?' Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring," *Ecol Inform*, vol. 57, May 2020, doi: 10.1016/j.ecoinf.2020.101085.

[25] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and Its Various Variants," in *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, Institute of Electrical and Electronics Engineers Inc., Jan. 2019, pp. 415–419. doi: 10.1109/SMC.2018.00080.

[26] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2003.

[27] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017, doi: 10.1016/j.neucom.2016.12.038.

[28] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, "On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression," Aug. 2020, doi: 10.1109/LSP.2020.3016837.

[29] S. Amershi *et al.*, "Software Engineering for Machine Learning: A Case Study," 2019, Accessed: Jul. 11, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/

[30] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[31] Y. Zhao and Z. Zhao, "MLOps Scaling ML in an Industrial Setting," 2658.

[32] R. Garcia, V. Sreekanti, N. Yadwadkar, D. Crankshaw, J. E. Gonzalez, and J. M. Hellerstein, "Context: The Missing Piece in the Machine Learning Lifecycle".

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[33] J. GAMA, I. ZLIOBAITE, A. BIFET, M. PECHENIZKIY, and A. BOUCHACHIA, “A Survey on Concept Drift Adaptation,” in *ACM Comput. Surv. 1, 1, Article 1* , 2013. doi: 10.1145/0000000.0000000.

[34] S. Ackerman, O. Raz, M. Zalmanovici, and A. Zlotnick, “Automatically detecting data drift in machine learning classifiers,” Nov. 2021, [Online]. Available: http://arxiv.org/abs/2111.05672

[35] R. S. M. Barros and S. G. T. C. Santos, “A large-scale comparison of concept drift detectors,” *Inf Sci (N Y)*, vol. 451–452, pp. 348–370, Jul. 2018, doi: 10.1016/j.ins.2018.04.014.

[36] E. S. Babüroğlu, A. Durmuşoğlu, and T. Dereli, “Novel hybrid pair recommendations based on a large-scale comparative study of concept drift detection,” *Expert Syst Appl*, vol. 163, Jan. 2021, doi: 10.1016/j.eswa.2020.113786.

[37] L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeyer, “Are Concept Drift Detectors Reliable Alarming Systems? -- A Comparative Study,” Nov. 2022, [Online]. Available: http://arxiv.org/abs/2211.13098

[38] A. Cockburn, “Agile Software Development,” 2000.

[39] J. Highsmith and A. Cockburn, “Agile Software Development: The Business of Innovation,” *Computer (Long Beach Calif)*, vol. 34, no. 9, pp. 120–122, 2001, [Online]. Available: http://computer.org/

[40] A. Moran, “Managing Agile.”

[41] K. Schwaber and J. Sutherland, “The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game,” 2020.

[42] M. Hüttermann, “Introducing DevOps,” *DevOps for Developers*, pp. 15–31, 2012, doi: 10.1007/978-1-4302-4570-4_2.

[43] B. Fitzgerald and K. J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, vol. 123, pp. 176–189, Jan. 2017, doi: 10.1016/j.jss.2015.06.063.

[44] A. Dyck, R. Penners, and H. Lichter, “Towards definitions for release engineering and DevOps,” in *Proceedings - 3rd International Workshop on Release Engineering, RELENG 2015*, Institute of Electrical and Electronics Engineers Inc., 2015, p. 3. doi: 10.1109/RELENG.2015.10.

[45] J. Angara, S. Prasad, and G. Sridevi, “DevOPs project management tools for sprint planning, estimation and execution maturity,” *Cybernetics and Information Technologies*, vol. 20, no. 2, pp. 79–92, 2020, doi: 10.2478/cait-2020-0018.

[46] M. Fowler, “Continuous Integration.” Accessed: Jul. 13, 2022. [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html

[47] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

vol. 5. Institute of Electrical and Electronics Engineers Inc., pp. 3909–3943, 2017. doi: 10.1109/ACCESS.2017.2685629.

[48] L. Chen, “Continuous delivery: Huge benefits, but challenges too,” *IEEE Softw*, vol. 32, no. 2, pp. 50–54, Mar. 2015, doi: 10.1109/MS.2015.27.

[49] Google, “MLOps: Continuous Delivery und Pipelines zur Automatisierung im maschinellen Lernen | Google Cloud.” Accessed: Jul. 12, 2022. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=de

[50] Microsoft, “Machine learning operations (MLOps) framework to upscale machine learning lifecycle with Azure Machine Learning - Azure Architecture Center | Microsoft Docs.” Accessed: Jul. 13, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-technical-paper

[51] “MLOps: Continuous Delivery und Pipelines zur Automatisierung im maschinellen Lernen | Cloud Architecture Center | Google Cloud.” Accessed: Mar. 12, 2023. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=de

[52] Microsoft, “Machine Learning operations maturity model - Azure Architecture Center | Microsoft Docs.” Accessed: Jul. 13, 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model

[53] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, “MLOps -- Definitions, Tools and Challenges,” Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.00162

[54] L. Baier, S. Seebacher, and R. paper Baier, “CHALLENGES IN THE DEPLOYMENT AND OPERATION OF MACHINE LEARNING IN PRACTICE Blockchain Business Networks View project (BigDieMo) Geschäftsmodelle 4.0-Entwicklung eines methodischen Baukastens zur Gestaltung von Big Data Dienstleistungen View project CHALLENGES IN THE DEPLOYMENT AND OPERATION OF MACHINE LEARNING IN PRACTICE,” 2019. [Online]. Available: https://www.researchgate.net/publication/332996647

[55] L. E. Lwakatare, P. Kuvaja, and M. Oivo, “Dimensions of DevOps,” in *Agile Processes in Software Engineering and Extreme Programming. XP 2015.*, vol. 212, 2015, pp. 212–217. doi: https://doi.org/10.1007/978-3-319-18612-2_19.

[56] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing*. 2013.

[57] P. M. Mell and T. Grance, “The NIST definition of cloud computing,” Gaithersburg, MD, 2011. doi: 10.6028/NIST.SP.800-145.

[58] R. Buyya, C. Vecchiola, and S. T. Selvi, “Cloud Computing Architecture,” in *Mastering Cloud Computing*, 2013, pp. 111–140.

[59] A. B. Kolltveit and J. Li, “Operationalizing Machine Learning Models - A Systematic Literature Review,” in *Proceedings - Workshop on Software Engineering for Responsible AI, SE4RAI 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1–8. doi: 10.1145/3526073.3527584.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[60] P. Fergus and C. Chalmers, *Applied Deep Learning*. 2022.

[61] “Was ist cloudnativ? | Microsoft Learn.” Accessed: Feb. 10, 2023. [Online]. Available: https://learn.microsoft.com/de-de/dotnet/architecture/cloud-native/definition

[62] “Was sind Container? | Google Cloud.” Accessed: Feb. 10, 2023. [Online]. Available: https://cloud.google.com/learn/what-are-containers?hl=de

[63] Docker, “Comparing Containers and Virtual Machines.” Accessed: Oct. 21, 2023. [Online]. Available: https://www.docker.com/resources/what-container/

[64] A. B. Kolltveit and J. Li, “Operationalizing Machine Learning Models - A Systematic Literature Review,” in *Proceedings - Workshop on Software Engineering for Responsible AI, SE4RAI 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1–8. doi: 10.1145/3526073.3527584.

[65] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, “Demystifying mlops and presenting a recipe for the selection of open-source tools,” *Applied Sciences (Switzerland)*, vol. 11, no. 19, Oct. 2021, doi: 10.3390/app11198861.

[66] S. Singh, N. Singh, and V. Singh, “Comparative Analysis of Open Standards for Machine Learning Model Deployments,” in *ICT Systems and Sustainability*, vol. 321, M. Tuba, S. Akashe, and A. Joshi, Eds., in Lecture Notes in Networks and Systems, vol. 321. , Singapore: Springer Nature Singapore, 2022. doi: 10.1007/978-981-16-5987-4.

[67] “Save, serialize, and export models | TensorFlow Core.” Accessed: Jul. 15, 2023. [Online]. Available: https://www.tensorflow.org/guide/keras/serialization_and_saving

[68] R. Schlittler, “Machine Learning Operations-vom Modell als Datei zum Prototyp mit Userinteraktion Bachelorarbeit im Studiengang Wirtschaftsinformatik.” [Online]. Available: https://robert-schlittler.medium.com/let-users-interact-with-your-onnx-pmml-model-with-ml-starter-

[69] “ONNX Concepts - ONNX 1.15.0 documentation.” Accessed: Jul. 15, 2023. [Online]. Available: https://onnx.ai/onnx/intro/concepts.html

[70] “About Us - The HDF Group.” Accessed: Jul. 15, 2023. [Online]. Available: https://www.hdfgroup.org/about-us/

[71] “pickle — Python object serialization — Python 3.11.4 documentation.” Accessed: Jul. 15, 2023. [Online]. Available: https://docs.python.org/3/library/pickle.html

[72] “Protocol Buffers Documentation.” Accessed: Jul. 15, 2023. [Online]. Available: https://protobuf.dev/

[73] A. Lopez Garcia *et al.*, “A cloud-based framework for machine learning workloads and applications,” *IEEE Access*, vol. 8, pp. 18681–18692, 2020, doi: 10.1109/ACCESS.2020.2964386.

[74] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction.” [Online]. Available: https://pair-code.github.io/facets/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[75] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, "An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams."

[76] J. Shao, Z. Ahmadi, and S. Kramer, "Prototype-based learning on concept-drifting data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, 2014, pp. 412–421. doi: 10.1145/2623330.2623609.

[77] A. Qahtan, B. Alharbi, S. Wang, and X. Zhang, "A PCA-based change detection framework for multidimensional data streams," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Aug. 2015, pp. 935–944. doi: 10.1145/2783258.2783359.

[78] IEEE Computational Intelligence Society, International Neural Network Society, Institute of Electrical and Electronics Engineers, and B. C.) IEEE World Congress on Computational Intelligence (2016 : Vancouver, *2016 International Joint Conference on Neural Networks (IJCNN) : 24-29 July 2016, Vancouver, Canada.*

[79] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing *," 2006. Accessed: Feb. 21, 2023. [Online]. Available: http://www.lsi.upc.edu/

[80] L. Todorovski, N. Lavrac, and K. P. Jantke, "Lecture Notes in Artificial Intelligence 4265 Subseries of Lecture Notes in Computer Science."

[81] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski, "Decision trees for mining data streams based on the mcdiarmid's bound," *IEEE Trans Knowl Data Eng*, vol. 25, no. 6, pp. 1272–1279, Jun. 2013, doi: 10.1109/TKDE.2012.66.

[82] N. Lu, G. Zhang, and J. Lu, "Concept drift detection via competence models," *Artif Intell*, vol. 209, no. 1, pp. 11–28, Apr. 2014, doi: 10.1016/j.artint.2014.01.001.

[83] N. Pasumarthi and L. Malleswari, "AN EMPIRICAL STUDY AND COMPARATIVE ANALYSIS OF CONTENT BASED IMAGE RETRIEVAL (CBIR) TECHNIQUES WITH VARIOUS SIMILARITY MEASURES," 2016.

[84] G. Retscher and J. Joksch, "Comparison of Different Vector Distance Measure Calculation Variants for Indoor Location Fingerprinting," 2016, pp. 53–76.

[85] I. Schmitt, "Ähnlichkeitsmaße," in *Ähnlichkeitssuche in Multimedia-Datenbanken*, 2006, pp. 215–259.

[86] G. Goos *et al.*, "On the Use of Bhattacharyya Distance as a Measure of the Detectability of Steganographic Systems," in *Transactions on Data Hiding and Multimedia Security*, vol. 3, 1973, pp. 23–32.

[87] K. A. Lee, C. H. You, H. Li, T. Kinnunen, and K. C. Sim, "Using discrete probabilities with bhattacharyya measure for SVM-Based speaker verification," *IEEE Trans Audio Speech Lang Process*, vol. 19, no. 4, pp. 861–870, 2011, doi: 10.1109/TASL.2010.2064308.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[88] T. Guillerme and N. Cooper, “Effects of missing data on topological inference using a Total Evidence approach,” *Mol Phylogenet Evol*, vol. 94, pp. 146–158, Jan. 2016, doi: 10.1016/j.ympev.2015.08.023.

[89] V. M. A. Souza, A. R. S. Parmezan, F. A. Chowdhury, and A. Mueen, “Efficient unsupervised drift detector for fast and high-dimensional data streams,” *Knowl Inf Syst*, vol. 63, no. 6, pp. 1497–1527, Jun. 2021, doi: 10.1007/s10115-021-01564-6.

[90] L. Z. Guo, Z. Zhou, and Y. F. Li, “RECORD: Resource Constrained Semi-Supervised Learning under Distribution Shift,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, Aug. 2020, pp. 1636–1644. doi: 10.1145/3394486.3403214.

[91] V. M. A. Souza, D. F. Silva, G. E. A. P. A. Batista, and J. Gama, “Classification of evolving data streams with infinitely delayed labels,” in *Proceedings - 2015 IEEE 14th International Conference on Machine Learning and Applications, ICMLA 2015*, Institute of Electrical and Electronics Engineers Inc., Mar. 2016, pp. 214–219. doi: 10.1109/ICMLA.2015.174.

[92] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista, “Data stream classification guided by clustering on nonstationary environments and extreme verification latency,” in *SIAM International Conference on Data Mining 2015, SDM 2015*, Society for Industrial and Applied Mathematics Publications, 2015, pp. 873–881. doi: 10.1137/1.9781611974010.98.

[93] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3171, pp. 286–295, 2004, doi: 10.1007/978-3-540-28645-5_29.

[94] A. Bifet and R. Gavaldà, “Learning from Time-Changing Data with Adaptive Windowing *,” 2006. Accessed: Feb. 21, 2023. [Online]. Available: http://www.lsi.upc.edu/

[95] A. Ahmed, K. S. Sajan, A. Srivastava, and Y. Wu, “Anomaly Detection, Localization and Classification Using Drifting Synchrophasor Data Streams,” *IEEE Trans Smart Grid*, vol. 12, no. 4, pp. 3570–3580, Jul. 2021, doi: 10.1109/TSG.2021.3054375.

[96] I. A. Nikolov *et al.*, “Seasons in Drift: A Long-Term Thermal Imaging Dataset for Studying Concept Drift,” APA, 2021. [Online]. Available: https://openreview.net/forum?id=Ljjqeg-BNtPi

[97] “Azure DevOps Services | Microsoft Azure.” Accessed: Mar. 12, 2023. [Online]. Available: https://azure.microsoft.com/de-de/products/devops#overview

[98] “Verwalteter Azure Kubernetes Service (AKS) | Microsoft Azure.” Accessed: Mar. 12, 2023. [Online]. Available: https://azure.microsoft.com/de-de/products/kubernetes-service/

[99] “Informationen zum Blob(objekt)speicher - Azure Storage | Microsoft Learn.” Accessed: Mar. 15, 2023. [Online]. Available: https://learn.microsoft.com/de-de/azure/storage/blobs/storage-blobs-overview

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[100] I. Karamitsos, S. Albarhami, and C. Apostolopoulos, “Applying devops practices of continuous automation for machine learning,” *Information (Switzerland)*, vol. 11, no. 7, pp. 1–15, Jul. 2020, doi: 10.3390/info11070363.

[101] iguazio, “Breaking the Silos Between Data Scientists, Engineers and DevOps with New MLOps Practices | Iguazio.” Accessed: Mar. 15, 2023. [Online]. Available: https://www.iguazio.com/sessions/breaking-the-silos-between-data-scientists-engineers-and-devops-with-new-mlops-practices/

[102] E. Oikarinen, H. Tiittanen, A. Henelius, and K. Puolamäki, “Detecting virtual concept drift of regressors without ground truth values,” *Data Min Knowl Discov*, vol. 35, no. 3, pp. 726–747, May 2021, doi: 10.1007/s10618-021-00739-7.

[103] “Manage resource groups - Azure portal - Azure Resource Manager | Microsoft Learn.” Accessed: Aug. 15, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal

[104] “What is a workspace? - Azure Machine Learning | Microsoft Learn.” Accessed: Aug. 15, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/concept-workspace?view=azureml-api-2

[105] “What is an Azure Machine Learning compute instance? - Azure Machine Learning | Microsoft Learn.” Accessed: Aug. 15, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/concept-compute-instance?view=azureml-api-2

[106] T. Tuor, S. Wang, K. K. Leung, and K. Chan, “Distributed Machine Learning in Coalition Environments: Overview of Techniques,” in *21st International Conference on Information Fusion (FUSION)*, 2018, pp. 814–821.

6.2 Abbildungsverzeichnis

Abbildung 2-1: Aufbau eines künstlichen neuronalen Netzes 6
Abbildung 2-2: Schematische Darstellung eines Autoencoders (vgl. Bank et al. [26]) 11
Abbildung 2-3: vereinfacht dargestellter ML-Ablauf [29] 12
Abbildung 2-4: schematisch dargestellte Auswirkung des Konzept Drifts 15
Abbildung 2-5: schematisch dargestellte Auswirkung des Daten Drifts 16
Abbildung 2-6: schematische Darstellung des graduellen und abrupten Drifts 16
Abbildung 2-7: Bildliche Darstellung der Infrastruktur zur Bereitstellung von ML-Systemen [2] 21
Abbildung 2-8: Ablauf des MLOps Prozesses [49] 22
Abbildung 2-9: Darstellung des Reifegradmodells von Microsoft (vgl. [53]) 24
Abbildung 2-10: Architektur von Cloud Computing [58] 26
Abbildung 2-11: Vergleich der Architektur eines Containers und einer virtuellen Maschine [63] 27

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abbildung 2-12: Schematische Darstellung der Kommunikation von REST API (links) und gRPC (rechts) [60] 31
Abbildung 2-13: Vergleich der Testkomplexität zwischen traditionelle Code Systemen und ML-Systemen [74] 31
Abbildung 2-14: Bildhafte Darstellung zweier Datenverteilungen [89] 38
Abbildung 2-15: Vergleich der Bearbeitungsdauer in Abhängigkeit der Dimension [89] 39
Abbildung 3-1: Vorgeschlagene Architektur 43
Abbildung 3-2: Vorgeschlagener Prozess zur Erstellung eines ML-Services 47
Abbildung 3-3: Vorgeschlagener Pipeline Prozess 48
Abbildung 3-4: Ablauf der CI – Pipeline 49
Abbildung 3-5: Schritte innerhalb der CT – Pipeline 50
Abbildung 4-1: Einbauposition eines Drehmomentsensors im Lenkstrang (Thyssenkrupp Presta Steering) 55
Abbildung 4-2: generische Ordnerstruktur des ML-Projektes 56
Abbildung 4-3: Kraft-Weg Kurve eines als OK eingestuften Kraftverlaufes 58
Abbildung 4-4: Reale Kraft-Weg Kurve (blau), sowie durch den Autoencoder rekonstruierte Kraft-Weg Kurve (rot) 60
Abbildung 4-5: Ablauf der kontinuierlichen Integrationspipeline innerhalb der Azure DevOps Umgebung 62
Abbildung 4-6: Ablauf der kontinuierlichen Trainingspipeline innerhalb der Azure DevOps Umgebung 64
Abbildung 4-7: Registrierte Datensätze in der Azure-ML Umgebung 65
Abbildung 4-8: Registrierte Modelle in der Azure-ML Umgebung 65
Abbildung 4-9: Stufen der kontinuierlichen Bereitstellungspipeline innerhalb der Azure DevOps Umgebung 66
Abbildung 4-10: Ablauf der kontinuierlichen Bereitstellungspipeline innerhalb der Azure DevOps Umgebung zur Bereitstellung in der produktionsähnlichen Umgebung 67
Abbildung 4-11: Darstellung der Überwachung des ML-Modells 69
Abbildung 4-12: Ergebnisse des IBDD – Drifterkennungsalgorithmus, mit dem zu analysierenden Bild (links) und dem Output (rechts) 70
Abbildung 4-13: Darstellung von Ausgangsdaten des ML-Services, wobei die ersten 8000 Datenpunkte die Trainingsdaten und die nächsten 4500 die Produktionsdaten darstellen 71
Abbildung 4-14: Beispielhafter Aufbau einer individuell programmierten Azure Monitor - Umgebung zur Modellüberwachung 72
Abbildung 4-15: Aufbau der Azure Application Insights – Umgebung, in der die Überwachung des Systems erfolgt 73
Abbildung 4-16: Unterschiedliche Kurven in der Analyse (Mittelwerte) 73
Abbildung 4-17: Ergebnisse der Eingangsdatenanalyse auf spätere Produktionsdaten, sowie künstlich angepasste Produktionsdaten 74

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abbildung 4-18: Darstellung von Ausgangsdaten des ML-Services, wobei die ersten 8000 Datenpunkte die Trainingsdaten und die nächsten 4500 die Produktionsdaten darstellen 75

6.3 Tabellenverzeichnis

Tabelle 2-1: Kompatibilität der Dateiformate (Spalten) mit den jeweiligen Frameworks (Zeilen) (vgl. [68]) 29

Tabelle 4-1: Vergleich der beiden Trainingsvorgänge 60

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub