Anomaly Detection im Monitoring des Produktivbetriebs der ODAV AG anhand der Analyse der Error-Count Metrik

Benjamin Eder (M.Sc.)
ODAV AG
Gesellschaft für Informatik und Telekommunikation
Ernst-Heinkel-Str. 11, 94315 Straubing, Germany
E-Mail: b.eder@odav.de

Benjamin Neumann (Dipl.-Inf. (FH))
ODAV AG
Gesellschaft für Informatik und Telekommunikation
Ernst-Heinkel-Str. 11, 94315 Straubing, Germany
E-Mail: b.neumann@odav.de

Professor Dr. Frank Herrmann
Ostbayerische Technische Hochschule Regensburg
Innovationszentrum für Produktionslogistik
und Fabrikplanung
Galgenbergstraße 32, 93053 Regensburg, Germany
E-Mail: frank.herrmann@oth-regensburg.de

SCHLÜSSELWÖRTER

Anomaly Detection, Machine Learning

ABSTRAKT

Viele Firmen entwickeln Software und stellen diese auch gleichzeitig dem Kunden auf Servern bereit. Da es nötig ist, dass der Betrieb der Software 24 Stunden am Tag sieben Tage die Woche gewährleistet wird, soll ein System entwickelt werden, das Anomalien automatisch im Betrieb der Server erkennt und dies einer entsprechenden Stelle, wie zum Beispiel dem betreibenden Rechenzentrum, meldet.

1 EINLEITUNG

Da viele Firmen selbst entwickelte Software den Kunden auch bereitstellen, zum Beispiel in einem eigenen Rechenzentrum, wird es auch immer wichtiger sicherzustellen, dass die Software für den Kunden 24 Stunden am Tag sieben Tage die Woche zur Verfügung steht. Somit werden Systeme benötigt, die überwachen, ob die Server erreichbar sind und die Software funktionsfähig ist.

Deshalb soll am Beispiel der ODAV AG (ODAV AG, Gesellschaft für Informatik und Telekommunikation 1969), die zur Bereitstellung ihrer Software ein eigenes Rechenzentrum betreibt, ein System entwickelt werden, das Anomalien automatisch im Produktivbetrieb erkennen kann und damit die aktuelle manuelle Methode optimiert. Anomalien beschreiben dabei Muster in Daten, die nicht mit einer genau definierten Vorstellung von normalem Verhalten übereinstimmen (Chandola, Banerjee und Kumar 2009). Diese können aus verschiedenen Gründen in den Daten auftreten, zum Beispiel durch böswillige Aktivitäten wie Cyber-Angriffe und terroristische Aktivitäten oder durch Systemausfälle.

Zur Entwicklung des Systems werden zunächst in Abschnitt 2 die Probleme, die mithilfe des Systems gelöst werden sollen, definiert. Anschließend wird in Abschnitt 3 ein Konzept erarbeitet, wie solche Anomalien am besten erkannt werden können. Anhand dieses Konzeptes soll dann in Abschnitt 4 ein System implementiert werden, das die Anomalien erkennt und entsprechend verantwortliche Personen im Falle einer Anomalie benachrichtigt. Schließlich soll in Abschnitt 5 evaluiert werden, wie das implementierte System die Abläufe in der Firma beeinflusst.

2 PROBLEMSTELLUNG

Im aktuellen Produktivbetrieb der ODAV AG (ODAV AG, Gesellschaft für Informatik und Telekommunikation 1969) werden Logmeldungen aus dem Produktivsystem gesammelt, jedoch nicht weitergehend automatisch analysiert. Die Probleme, die anhand der Analyse der Logmeldungen gelöst werden sollen, werden in Abschnitt 2.2 erläutert. Zunächst soll jedoch in Abschnitt 2.1 erläutert werden, wie das Produktivsystem aktuell aufgebaut ist.

2.1 Aktueller Aufbau des Produktivsystems

Das Produktivsystem für alle Kunden ist mit Kubernetes (The Kubernetes Authors 2014), einem System zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerbasierten Anwendungen, aufgebaut. Darin sind alle Anwendungen als Container bereitgestellt, sodass die Kunden auf diese zugreifen können.

Die Logmeldungen dieser Anwendungen werden in einem Elasticsearch Server (Elasticsearch B.V. 2010), einer verteilten Such- und Analyse-Engine, gesammelt und für mindestens sechs Monate datenschutzkonform gespeichert. Eine Meldung kann dabei immer einem bestimmten Zeitpunkt zugeordnet werden. Zudem wird zwischen verschiedenen Typen unterschieden: *Info*, *Warn* und *Error*.

Im Folgenden bezeichnet die *Error-Count Metrik* die Anzahl der Logmeldungen des Typs *Error* innerhalb eines bestimmten Zeitintervalls. Das Zeitinter-

vall wird hier und im Folgenden als die vergangenen fünf Minuten definiert.

2.2 Probleme im bestehenden System

Im aktuellen Produktivsystem sind die folgenden Probleme vorhanden. Diese sollen zunächst erfasst werden, um im Verlauf der Arbeit entsprechend bearbeitet werden zu können.

a. **Erkennen von Angriffen auf das System**
Ein Angreifer kann mittels eines Brute-force Angriffes (Dave 2013) versuchen, sich Zugang zu den Anwendungen zu verschaffen. Dies kann beispielsweise geschehen, indem er versucht, das Passwort eines Benutzers, dessen Anmeldename er kennt, zu erraten, was im Authentifizierungsserver zu fehlgeschlagenen Authentifizierungsversuchen führt, die als Fehler erfasst werden.

   Im aktuellen System können Angriffe durch die Firewall der Firma verhindert werden. Diese erkennt einen Angriff dadurch, dass viele Zugriffe innerhalb einer bestimmten Zeitspanne auf einen Port erfolgen. Die Erkennung erfolgt damit auf der Netzwerkebene entsprechend des Sieben-Schichten-Modells (Day und Zimmermann 1983). Problematisch dabei ist, dass die Firewall auch alamieren könnte, wenn viele Benutzer gleichzeitig auf die Systeme zugreifen, da diese nicht erkennt, ob ein Zugriff einen Fehler verursacht oder nicht. Ein Problem ist somit, dass kein System existiert, dass einen Angriff auf Applikationsebene erkennt.

b. **Erkennen von ausgefallenen Anwendungen**
Viele Anwendungen im Produktivsystem stellen Schnittstellen für andere Anwendungen zur Verfügung. Diese Schnittstellen können jedoch aus verschiedensten Gründen temporär nicht erreichbar sein. So können zum Beispiel Container, in denen die Client- und Serveranwendungen bereitgestellt werden, abgestürzt sein. Auch können die Anwendungen selbst in einen kritischen Fehler gelaufen sein und funktionieren deshalb nicht mehr. Tritt ein solcher Fall ein, können die entsprechenden Schnittstellen nicht mehr erreicht werden. Dies führt auf Seite der aufrufenden Anwendungen zu Fehlern, welche im Elasticsearch Server erfasst werden.

   Im aktuellen Überwachungssystem wird eine Fehlermeldung angezeigt, sollte ein Pod über einen längeren Zeitraum nicht erreichbar sein. Um einen reibungslosen Betrieb aller Systeme zu gewährleisten, sollte dies automatisiert innerhalb kurzer Zeit angezeigt werden. Ein Problem des Systems ist somit, dass ausgefallene Anwendungen nicht sofort erkannt werden.

c. **Erkennen von ausgefallenen Nodes**
Das Kubernetes-Cluster der Firma besteht aus mehreren Nodes, welche zusammen alle Anwendungen bereitstellen. Eine Node ist dabei eine virtuelle oder physische Maschine (The Kubernetes Authors 2014). Nun kann jedoch eine oder mehrere Nodes unerwartet ausfallen, was darin resultiert, dass einige Anwendungen nicht mehr verfügbar sind. Ist dies der Fall, versucht das Kubernetes System zunächst, die Anwendungen der ausgefallenen Node(s) auf die restlichen zu verteilen, was jedoch unter Umständen die Antwortzeiten der Anwendungen verlangsamt und im schlimmsten Fall einen Timeout bei einer anfragenden Anwendung erzeugt. Auch kann es passieren, dass die vorhandenen Ressourcen der restlichen Nodes nicht ausreichen, um die ausgefallenen Anwendungen bereitzustellen. Dies führt dazu, dass einige Anwendungen eventuell gar nicht mehr verfügbar sind, was wiederum zu Fehlern bei anfragenden Anwendungen führt.

   Im aktuellen Überwachungssystem wird eine Fehlermeldung angezeigt, sollte eine Node über einen längeren Zeitraum nicht erreichbar sein. Um einen reibungslosen Betrieb aller Systeme zu gewährleisten, sollte dies automatisiert innerhalb kurzer Zeit angezeigt werden. Ein Problem ist somit, dass ausgefallene Nodes im Kubernetes Cluster erst nach einiger Zeit erkannt werden.

Die genannten Probleme sollen durch das im Folgenden beschriebene Lösungskonzept vermieden werden.

3 LÖSUNGSKONZEPT

Zunächst soll kurz erläutert werden, was *Anomaly Detection* (Chandola, Banerjee und Kumar 2009) ist und wie dies im Allgemeinen im aktuellen System eingesetzt werden kann. Anschließend soll sowohl ein simpler Ansatz als auch ein weiterer mithilfe von Machine Learning konzipiert werden. Dabei werden verschiedene Modelle für das Problem erstellt und erläutert. Schließlich sollen die erarbeiteten Ansätze miteinander verglichen werden, um den besten Ansatz aus diesen auszuwählen und in einem Überwachungssystem umzusetzen.

3.1 Einsatz von Anomaly Detection

Die Erkennung von Anomalien, genannt *Anomaly Detection*, bezieht sich auf das Problem, Muster in Daten zu finden, die nicht dem erwarteten Verhalten entsprechen (Chandola, Banerjee und Kumar 2009). Dabei resultieren aus Anomalien in einer Vielzahl von Anwendungsbereichen bedeutende und oft kritische Informationen, die genutzt werden können. Bezogen auf diese Definition soll im Produktivsystem

der Verlauf der *Error-Count Metrik* auf Muster untersucht werden, die während eines normalen Betriebes des Systems nicht erwartet werden. Dafür muss zunächst definiert werden, wie die erfassten Daten auszusehen haben, damit diese miteinander verglichen werden können.

Im Folgenden besteht der Verlauf eines Tages aus 288 Datenpunkten, die die Daten der *Error-Count Metrik* des entsprechenden Tages darstellen. Diese Datenpunkte werden in einem Graphen dargestellt, wobei $t = 0$ den *Error-Count* um fünf Minuten nach Mitternacht des entsprechenden Tages abbildet. Dies ist so gewählt, damit der Verlauf eines Tages nur Fehlermeldungen berücksichtigt, die auch an diesem Tag aufgetreten sind. Abbildung 1 stellt einen unauffälligen Verlauf an einem Arbeitstag dar.

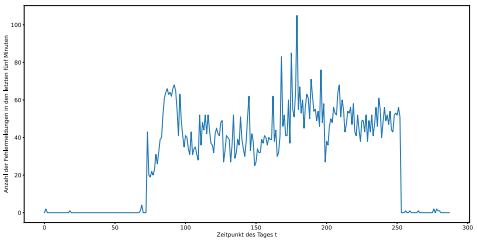


Abbildung 1: Unauffälliger Verlauf der *Error-Count Metrik* an einem Arbeitstag

Eine Anomalie ist für die vorliegenden Daten, wenn ein Datenpunkt deutlich höher angesiedelt ist als die restlichen Datenpunkte. Im Verlauf aus Abbildung 1 wäre dies zum Beispiel der Fall, wenn innerhalb von fünf Minuten mehr als 150 Fehlermeldungen eingegangen wären. Da das Maximum dieses Tages bei ungefähr 100 Fehlermeldungen pro fünf Minuten liegt, wäre dies eine Abweichung von 50 Prozent. Dies weist auf eine mögliche Störung im System hin.

Da die durchschnittlichen Verläufe der Tage abhängig vom Wochentag anders ausfallen, werden die Daten nach Wochentag gruppiert analysiert. Dies wird bei allen Ansätzen berücksichtigt.

3.2 Einfache Anomaly Detection

Zunächst soll ein simples Konzept zur *Anomaly Detection* entwickelt werden. Dabei wird ein Grenzwert erstellt, der während des Produktivbetriebes nicht überschritten werden darf. Auf einen unteren Grenzwert wird verzichtet, da bei allen in Abschnitt 2.2 genannten Szenarien die Anzahl der Fehlermeldungen ansteigt.

Ein Ansatz dafür ist das Errechnen eines Grenzwertes anhand der Verläufe vorheriger gleicher Wochentage. Mit D_i als die Datenpunkte des Wochentages vor i Wochen, errechnet sich der Grenzwert g wie folgt:

$$g(n) = \left(\frac{\sum_{i=1}^{n} Q_{0,9}(D_i)}{n} \right) \cdot 1,25 \qquad (1)$$

Wie in Gleichung 1 zu erkennen ist, werden für die Berechnung des Grenzwertes die Daten der letzten n Wochen verwendet. Aus den Daten der jeweiligen Tagesverläufe wird das 90-Prozent-Quantil gebildet. Dies wurde gewählt, damit einzelne höhere Werte, die eine Anomalie anzeigen könnten, nicht berücksichtigt werden. Anschließend wird der Mittelwert der Quantile errechnet. Um einen Toleranzbereich zu erzeugen, wird dieser Wert schließlich um 25 Prozent erhöht. Bei mehreren Versuchen mit verschiedenen Werten, bei denen für dieselben Tagesverläufe die Grenzwerte errechnet wurden, stellte sich dieser Wert als bester heraus, da mit diesem die meisten Anomalien erkannt werden.

Der errechnete Grenzwert g wird nun für den aktuellen Tag zur *Anomaly Detection* verwendet. Dabei wird die *Error-Count Metrik* für den aktuellen Arbeitstag periodisch gegen dessen Grenzwert geprüft. Dieser Grenzwert ist dabei konstant für einen bestimmten Arbeitstag. Sollte zu irgendeinem Zeitpunkt der *Error-Count* größer als der Grenzwert sein, so wird dies als Anomalie angenommen.

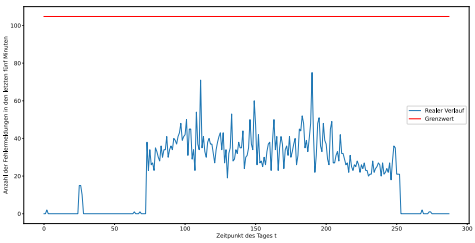


Abbildung 2: Prüfung eines vollständigen Arbeitstages gegen dessen Grenzwert

In Abbildung 2 ist ein Beispiel für die Prüfung gegen den Grenzwert dargestellt. Die blaue Kurve stellt dabei die *Error-Count Metrik* des Arbeitstages dar und die rote Gerade den Grenzwert für diesen Arbeitstag. Der Grenzwert beträgt in diesem Beispiel ungefähr 105 Fehlermeldungen pro fünf Minuten und wurde anhand der Daten der vorausgegangenen sechs Wochen berechnet. Die Werte während des Tages überschreiten diesen zu keinem Zeitpunkt, weshalb dieser Tag als unauffällig eingestuft werden würde. Jedoch ist zu Beginn des Tages, als die Werte noch nahezu null betragen, ein kleiner Ausschlag zu erkennen. Obwohl dort eine hohe Wahrscheinlichkeit für eine Anomalie vorliegt, da außerhalb der regulären Arbeitszeiten der Kunden kaum Fehlermeldungen auftreten sollten, wird dieser Ausschlag nicht erkannt.

Als Problem stellt sich dabei heraus, dass der Grenzwert auf den Peaks der vorhergehenden Tage

basiert. Dies führt dazu, dass bei der Berechnung des Grenzwertes lediglich die hohen Werte einfließen und somit der Grenzwert die geringere Anzahl von Fehlermeldungen außerhalb der regulären Arbeitszeiten nicht beachtet.

3.3 Anomaly Detection mit Machine Learning

Ein weiterer Ansatz zur *Anomaly Detection* ist, Machine Learning Modelle zu verwenden, um mithilfe dieser die Anomalien zu erkennen. Im Folgenden sollen verschiedene Modelle vorgestellt werden, wie dies im vorliegenden Fall umgesetzt werden könnte. Zunächst werden jedoch noch allgemeine Bedingungen definiert, die bei der Konzeption aller Modelle gelten.

Für alle Modelle stehen die Daten von acht Monaten zur Verfügung, mit denen diese trainiert werden. Die vorhandenen Daten sind für alle Modelle gleich. Das Trainieren eines Modells bezieht sich hier darauf, dass ein Algorithmus entwickelt wird, der, wie der Mensch auch, mit der Zeit und mit Erfahrung besser in einer bestimmten Aufgabe wird (Ketkar und Santana 2017).

Um die Modelle trainieren zu können, werden die Daten vor dem eigentlichen Training aufbereitet. Dabei werden diese zunächst nach Wochentag aufgeteilt, damit für jeden Wochentag ein separates Modell erstellt werden kann. Da die durchschnittlichen Verläufe an den verschiedenen Wochentagen unterschiedlich sind, hat dies den Vorteil, dass die Daten eines Wochentages die Vorhersagen der anderen Wochentage nicht beeinflussen. An dieser Stelle ist zu erwähnen, dass dies auch zu erreichen wäre, indem man den Wochentag als weitere Eingabe neben den Datenpunkten verwendet, was in diesem Paper jedoch nicht weitergehend analysiert wird.

Anschließend werden die Daten so aufbereitet, dass die Modelle als Eingabe den Verlauf eines Tages erhalten, also 288 Datenpunkte. Um einen einzelnen Tagesverlauf für die kommende Woche der erhaltenen Daten vorherzusagen, erhält das Modell als anzustrebende Ausgabe den Verlauf des Tages in der folgenden Woche. Somit hat das Modell als Ausgabe ebenfalls 288 Datenpunkte. Letztlich wird das letzte Datenpaar aus den Trainingsdaten entfernt und als Testdatum verwendet, das beim Trainingsprozess des Modells nicht berücksichtigt wird.

Als Vergleich wird bei beiden Modellen anhand des Testdatenpaares der Verlauf eines Tages vorhergesagt. Da im späteren Betrieb die Tagesverläufe dem Modell nicht bekannt sind, wie zum Beispiel die Verläufe des Testdatenpaares, kann damit ein Verlauf vorhergesagt werden, wie er auch im späteren Betrieb vorhergesagt werden würde. Dieser Verlauf wird mit dem tatsächlichen Verlauf des Tages verglichen.

Als erste Variante kann dabei eine Kurve abgeleitet werden, die die euklidische Distanz zwischen dem vorhergesagten und dem realen Verlauf darstellt. Diese wird dabei verwendet, da sie die Differenz zwischen dem vorhergesagten und dem realen Wert an einem bestimmten Zeitpunkt darstellt und so die Vorhersage direkt mit dem realen Wert verglichen wird. Anhand der Kurve der euklidischen Distanz soll dann vorhergesagt werden, ob eine Anomalie vorliegt. Dabei wird der *Training Loss* des entsprechenden Modells als Grenzwert für die Werte der euklidischen Distanz am aktuellen Tag verwendet. Überschreitet die Distanz zwischen Vorhersage und realem Wert zu irgendeinem Zeitpunkt diesen Grenzwert, wird dies als Anomalie angenommen. Ein Vorteil dieser Variante ist, dass sowohl der Grenzwert als auch die euklidische Distanz zwischen Vorhersage und realem Wert schnell berechnet werden können. Ein Nachteil dabei ist, dass der errechnete Grenzwert konstant für den ganzen Tag ist.

Eine zweite Variante zur Vorhersage einer Anomalie ist die Festlegung eines Grenzwertes für den *Error-Count*. Dabei werden alle vorhergesagten Werte für den aktuellen Tag um einen gewissen Prozentsatz erhöht, um einen Toleranzbereich zu erzeugen. In mehreren Versuchen wurden für dieselben Tage mehrere Grenzwerte mit verschiedenen Toleranzbereichen errechnet. Bei einem Prozentsatz von 50 Prozent stellte sich dabei heraus, dass mit diesem Wert die meisten Anomalien erkannt wurden. Die dadurch entstehende Kurve dient schließlich als Grenzwert. Es existiert in diesem Falle ein separater Grenzwert für jeden Zeitpunkt des Tages. Überschreitet der reale Wert den zum Zeitpunkt gehörigen Grenzwert, wird dies als Anomalie angenommen. Ein Vorteil dieser Variante ist, dass der Grenzwert schnell berechnet werden kann. Ein Nachteil ist, dass der Grenzwert theoretisch an manchen Stellen null betragen könnte und somit bereits eine Fehlermeldung zur Meldung einer Anomalie führen könnte.

3.3.1 Modell: Autoregressiv

Zunächst soll ein autoregressives Modell vorgestellt werden. Die dabei verwendeten Werte und Funktionen wie die Anzahl der Schichten, die Anzahl der Neuronen pro Schicht, die Aktivierungsfunktion, die Anzahl der Trainingsepochen, der Optimizer, die *Learning Rate* sowie die *Loss Function* sind dabei in verschiedenen Versuchen verändert und getestet worden. Mit den im Folgenden beschriebenen Werten und Funktionen erzielt das Modell die besten Ergebnisse, weshalb diese letztlich auch verwendet werden.

Abbildung 3: Architektur des erstellten autoregressiven Modells

In Abbildung 3 ist die Architektur dieses Modells dargestellt. Es benötigt als Eingabe 288 Datenpunk-

te und erhält dafür die Daten eines vollständigen vergangenen Tages. Die Daten werden anschließend in drei *Hidden Layer* (Ketkar und Santana 2017), alle Schichten vor der Ausgabeschicht, weitergegeben. In diesem Modell sind dies drei *Dense Layer*. Dies sind Schichten, bei denen alle Neuronen mit den Neuronen der vor- und nachgelagerten Schicht vollständig vernetzt sind. Die erste Schicht enthält 512, die zweite 1024 und die dritte wieder 512 Neuronen. Als Aktivierungsfunktion wird bei allen drei Schichten die *hyperbolische Tangensfunktion*

$$f(x) = \tanh(x)$$

verwendet. Alternativ könnte hier auch die Rectified Linear Unit (ReLU) als Aktivierungsfunktion verwendet werden. Während der Konzeption konnte jedoch festgestellt werden, dass mit dieser die Vorhersagen teilweise nur einen konstanten Wert lieferten und somit nicht verwendet werden konnten. Die Ausgabeschicht besteht ebenfalls aus einem *Dense Layer* mit 288 Neuronen und erzeugt damit eine Ausgabe von 288 Datenpunkten, die den Verlauf der *Error-Count Metrik* für den Tag in der folgenden Woche beschreibt.

Das Trainieren des Modells erfolgt in 100 Epochen. Dabei wird der *Adam Optimizer* verwendet, da dies die gängigste Methode ist. Dessen *Learning Rate* wird in Abhängigkeit der aktuellen Trainingsepoche festgelegt. Die Funktion dafür ist wie folgt definiert, wobei e die aktuelle Epoche repräsentiert:

$$lr(e) = \begin{cases} 0,1 & \text{falls } e < 50 \\ 0,01 & \text{sonst} \end{cases} \text{ mit } e \in [0, 99] \quad (2)$$

Wie in Gleichung 2 zu erkennen ist, ist die *Learning Rate* in den ersten 50 Epochen des Trainings mit 0,1 definiert. Anschließend wird sie für die restlichen Epochen auf 0,01 verringert. Als *Loss Function* für den Trainingsprozess wird der *Mean Squared Error*

$$\sum_{i=1}^{n} (y - \hat{y})^2 \quad (3)$$

verwendet, da dieser in solchen Modellen ein Standardvorgehen ist.

Vorhersage im Vergleich zum realen Verlauf In Abbildung 5a ist die Vorhersage des Modells im Vergleich zum realen Verlauf der *Error-Count Metrik* dargestellt. Der Tag ist dabei der gleiche wie in Abschnitt 3.2.

Es ist zu erkennen, dass der vorhergesagte Verlauf über den ganzen Tag hinweg etwas höher ist als der reale Verlauf. Dies liegt jedoch unter anderem daran, dass die Fehlermeldungen in den Wochen zuvor etwas höher waren, vor allem zu den Zeiten vor und nach Arbeitsbeginn der Kunden. Diese sind hingegen auch in der Vorhersage gut zu erkennen und werden vom Modell berücksichtigt. Ebenfalls kann man die zwei Peaks im realen Verlauf in der Vorhersage erkennen.

Was beim realen Verlauf auffällt, ist der Ausschlag am frühen Morgen, der im Folgenden als Anomalie definiert wird.

Euklidische Distanz Wie in Abschnitt 3.1 beschrieben, soll zunächst mithilfe der euklidischen Distanz versucht werden, Anomalien zu erkennen. Dabei wird die euklidische Distanz zwischen einem Punkt in der Vorhersage und dem realen Wert zu diesem Zeitpunkt errechnet. Mit D_R als den realen Verlauf und D_P als den vorhergesagten Verlauf sowie D_R^i und D_P^i als den Werten aus dem jeweiligen Verlauf zum Zeitpunkt i, wird die Funktion zur Berechnung der euklidischen Distanz in Abhängigkeit zum Zeitpunkt wie folgt definiert:

$$e(i) = |D_R^i - D_P^i| \text{ mit } i \in [0, 288[\quad (4)$$

Die euklidische Distanz wird nun anhand der Funktion aus Gleichung 4 für alle Zeitpunkte berechnet. Schließlich muss noch ein Grenzwert gewählt werden. Dieser gibt an, wie hoch die euklidische Distanz zu einem Zeitpunkt maximal sein darf, und wird anhand des finalen *Training Loss* des Modells ($loss$) berechnet. Der Grenzwert wird definiert als:

$$g = \frac{loss}{288} \cdot 2 = \frac{loss}{144}$$

Im vorliegenden Beispiel ergibt dies einen Grenzwert von 76. Die Distanzen der Verläufe im hier verwendeten Beispiel sind zu keinem Zeitpunkt größer als der Grenzwert. Somit würde diese Methode die vorliegende Anomalie im Beispiel nicht erkennen.

Oberer Grenzwert Als weitere Methode soll, wie in Abschnitt 3.1 beschrieben, ein obererer Grenzwert für den realen Verlauf festgelegt werden, der jedoch nicht konstant, sondern abhängig vom Zeitpunkt definiert sein soll. Mit D_P als den vorhergesagten Verlauf und D_P^i als den Wert zum Zeitpunkt i, soll die Funktion zur Berechnung des Grenzwertes wie folgt definiert werden:

$$g(i) = D_P^i \cdot 1,5 \text{ mit } i \in [0, 288[\quad (5)$$

Der Grenzwert wird anhand der in Gleichung 5 definierten Funktion errechnet und mit den Werten des realen Verlaufs aus dem vorliegenden Beispiel verglichen. Dabei stellt diese Methode eine Anomalie an zwei Zeitpunkten des Tages fest. Diese Zeitpunkte sind zwischen 02:05 und 02:10 Uhr und entsprechen dem Ausschlag in den frühen Morgenstunden, der in Abbildung 5a zu erkennen ist. Somit stellt diese Methode korrekt eine Anomalie im vorliegenden Beispiel fest.

3.3.2 Modell: LSTM

Als weitere Variante soll ein Modell auf Basis von Long Short-Term Memory (LSTM) (Ketkar und Santana 2017) erstellt werden. Wie in Abschnitt 3.3.1 geschildert werden auch hier die Werte und Funktionen verwendet, mit denen in verschiedenen Versuchen die besten Ergebnisse erzielt wurden.

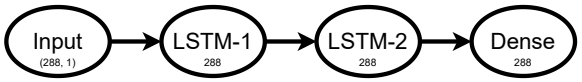


Abbildung 4: Architektur des erstellten LSTM Modells

Abbildung 4 stellt die Architektur des Modells dar. Es erhält als Eingabe 288 Datenpunkte, jedoch müssen hier alle Datenpunkte als Array übergeben werden. Somit ist die Eingabe ein zweidimensionales Array, welches 288 Einträge besitzt, die wiederum jeweils einen Eintrag beinhalten. Die Eingabe ist der Verlauf der *Error-Count Metrik* eines vollständigen vergangenen Tages. Als *Hidden Layer* werden zwei *LSTM Layer* verwendet, die jeweils aus 288 Neuronen bestehen. Die Ausgabeschicht besteht aus einem *Dense Layer* mit 288 Neuronen. Diese erzeugt damit die Vorhersage für den Tag in der folgenden Woche.

Das Trainieren des Modells erfolgt bei diesem Modell mit 100 Epochen. Dabei wird der *Adam Optimizer* verwendet, dessen *Learning Rate* abhängig von der Epoche festgelegt wird. Dafür wird die Funktion aus Gleichung 2 verwendet. Als *Loss Function* wird der *Mean Squared Error* (vgl. Gleichung 3) verwendet.

Vorhersage im Vergleich zum realen Verlauf In Abbildung 5b ist die Vorhersage des Modells im Vergleich zum realen Verlauf der *Error-Count Metrik* dargestellt. Der Tag ist dabei der gleiche wie in Abschnitt 3.2.

Vergleicht man die beiden Vorhersagen in Abbildung 5, so stellt man fest, dass die Vorhersage des autoregressiven Modells aus Abschnitt 3.3.1 fast identisch ist mit der Vorhersage des LSTM Modells. Somit treffen alle Aussagen zum Vergleich des vorhergesagten zum realen Verlauf, die für das autoregressive Modell zutreffen, auch für das LSTM Modell zu.

Euklidische Distanz Zunächst soll mithilfe der euklidischen Distanz versucht werden, Anomalien zu erkennen. Dabei wird die euklidische Distanz anhand der in Gleichung 4 definierten Funktion berechnet.

Da die vorhergesagten Verläufe des autoregressiven und des LSTM Modells fast identisch sind, ergeben sich auch für die euklidischen Distanzen fast dieselben Werte. Da die Eingabedaten für beide Modelle gleich sind, ist auch der *Training Loss* bei beiden Modellen annähernd gleich, da diese fast identische Ausgaben erzeugen. Somit ist der Grenzwert identisch zum autoregressiven Modell. Dadurch erkennt auch das LSTM Modell mit dieser Methode keine Anomalie im vorliegenden Beispiel.

Oberer Grenzwert Als weitere Methode soll ein oberer Grenzwert für den realen Verlauf festgelegt werden. Dieser wird anhand der in Gleichung 5 definierten Funktion in Abhängigkeit zum Zeitpunkt berechnet.

Der errechnete Grenzwert ist aufgrund der fast identischen Vorhersage des autoregressiven und LSTM Modells ebenfalls annähernd gleich. Somit erkennt das LSTM Modell mit dieser Methode die Anomalie korrekt im vorliegenden Beispiel an den zwei Zeitpunkten am frühen Morgen.

3.4 Vergleich der Ansätze

Um entscheiden zu können, welcher der Ansätze für die Implementierung verwendet werden soll, werden die verschiedenen Ansätze miteinander verglichen. Dabei soll hinsichtlich folgender Aspekte der beste Ansatz ermittelt werden:

a. **Berechnungszeit**
   Die Implementierung soll den Ansatz schnell berechnen können, damit Anomalien sofort und zuverlässig erkannt werden. Der wöchentliche Trainingsprozess kann dabei auch mehrere Minuten benötigen, jedoch muss eine Vorhersage in maximal zehn Sekunden erfolgen.

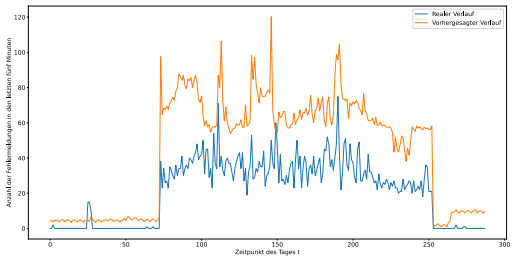
b. **Genauigkeit**
   Der Ansatz soll aufkommende Anomalien zuverlässig erkennen können, damit die Ausfälle aufgrund von Anomalien reduziert werden.

c. **Komplexität**
   Die Implementierung des Ansatzes soll leicht verständlich sein, damit der Quellcode von anderen Entwicklern ohne größere Vorkenntnisse betreut werden kann. Dadurch wird die Wartbarkeit des Systems verbessert.

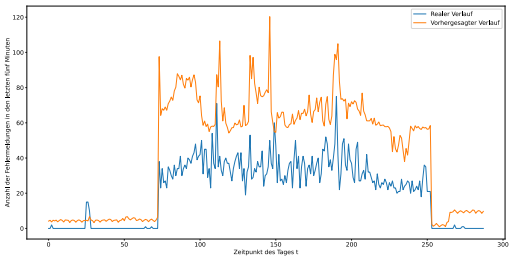
3.4.1 Einfacher Ansatz

Zunächst soll der einfache Ansatz (3.2) hinsichtlich der genannten Punkte analysiert werden.

Berechnungszeit Beim einfachen Ansatz wird ein Grenzwert wie in Gleichung 1 definiert berechnet. Zeitkritisch wäre dabei lediglich die Errechnung des Quantils, da jedoch die Anzahl der Datenpunkte, über der das Quantil errechnet wird, immer gleich ist, erfolgt die Berechnung des Grenzwertes in konstanter Zeit. Auch die Erkennung von Anomalien erfolgt in konstanter Zeit, da dies nur durch einen Vergleich des realen Wertes mit dem Grenzwert erfolgt.



(a) Vorhersage: Autoregressives Modell



(b) Vorhersage: LSTM Modell

Abbildung 5: Vorhersagen der Modelle im Vergleich zum realen Verlauf

Genauigkeit Überschreitet der *Error-Count* den üblichen Peak der letzten Wochen, so erkennt dies das Modell, was während des Arbeitstages die meisten Anomalien abdeckt. Tritt jedoch eine kleinere Anomalie auf, vor allem zu Zeiten, in denen das System nicht aktiv verwendet wird, wie zum Beispiel früh morgens, so wird dies vom einfachen Ansatz nicht erkannt, wie im Beispiel in Abschnitt 3.2 zu erkennen ist. Da außerhalb der regulären Arbeitszeiten solche kleineren Anomalien oft auftreten können, ist dies ein Nachteil dieses Ansatzes.

Komplexität Der einfache Ansatz beruht lediglich auf mathematischen Berechnungen, die in jeder Programmiersprache umzusetzen sind. In *Python* (vanRossum 1995) existiert zum Beispiel eine Bibliothek *NumPy* (Oliphant 2006), in der viele mathematische Operationen bereits implementiert sind. Dadurch kann dieser Ansatz einfach und verständlich implementiert werden.

3.4.2 Autoregressiver Ansatz

Als nächstes sollen die genannten Punkte beim autoregressiven Ansatz (3.3.1) analysiert werden.

Berechnungszeit Der autoregressive Ansatz basiert auf dem wöchentlichen Training eines autoregressiven Modells und dessen Vorhersagen. Für das Beispiel in Abschnitt 3.3.1 hat das Modell pro Epoche 0,2 Sekunden benötigt, für den gesamten Trainingsprozess 19,5 Sekunden. Für eine Vorhersage benötigt das Modell 0,1 Sekunden. Da die Rechenleistung im Produktivsystem, in dem die Implementierung später umgesetzt wird, stärker ist als die Leistung des Rechners, auf dem die Beispiele trainiert wurden, kann davon ausgegangen werden, dass diese Zeiten in der späteren Umsetzung tendenziell geringer sind.

Die Errechnung der euklidischen Distanz zum realen Wert mit der Funktion aus Gleichung 4 erfolgt in konstanter Zeit, ebenso wie die Errechnung des Grenzwertes. Die Erkennung von Anomalien geschieht in konstanter Zeit mit einem Vergleich.

Beim oberen Grenzwert erfolgt die Berechnung dieses ebenfalls in konstanter Zeit mit der Funktion aus Gleichung 5, da der Grenzwert immer aus 288 Datenpunkten besteht. Auch hier ist die Erkennung der Anomalie lediglich ein Vergleich und erfolgt somit in konstanter Zeit.

Sowohl die Prüfung mithilfe der euklidischen Distanz als auch mit dem oberen Grenzwert haben auf dem Entwicklungsrechner weniger als eine hundertstel Sekunde benötigt. Somit kann für beide Varianten angenommen werden, dass die Prüfung einer Anomalie ungefähr der Vorhersagezeit des Modells entspricht.

Genauigkeit Mit der euklidischen Distanz können höhere Abweichungen des *Error-Count* erkannt werden. Jedoch muss hier die Abweichung um einen gewissen absoluten Wert höher sein als der vorhergesagte Wert, um als Anomalie eingestuft zu werden. Dies führt dazu, dass kleinere Anomalien wie früh morgens im Beispiel nicht erkannt werden.

Beim oberen Grenzwert wird ein relativer Toleranzbereich errechnet, der überschritten werden darf, womit größere Abweichungen gut erkannt werden. Aber auch kleinere Abweichungen können so wie im Beispiel erkannt werden, da bei einer geringeren Anzahl zu erwartender Fehlermeldungen der Toleranzbereich niedriger ausfällt.

Komplexität Das autoregressive Modell kann nicht in jeder Programmiersprache implementiert werden. Für die Beispiele wurde dafür die Bibliothek *Keras* (Ketkar und Santana 2017) verwendet, welche zum Beispiel in *Python* verwendet werden kann. Die Erstellung des Modells gestaltet sich dabei recht simpel, da die Architektur des Modells aus Abbildung 3 übernommen werden kann. Die Berechnungen zur Erkennung einer Anomalie können ebenfalls einfach in *Python* umgesetzt werden. Somit kann dieser Ansatz verständlich implementiert werden, jedoch sollte ein Entwickler Vorkenntnisse im Bereich *Machine Learning* (Ketkar und Santana 2017) haben.

3.4.3 LSTM Ansatz

Nun soll der LSTM Ansatz anhand der genannten Punkte (3.3.2) analysiert werden.

Berechnungszeit Für diesen Ansatz wird wöchentlich ein LSTM Modell trainiert und dessen Vorhersage analysiert. Für das Beispiel in Abschnitt 3.3.2 hat das Modell 3,3 Sekunden pro Epoche und insgesamt 328,2 Sekunden benötigt. Eine Vorhersage dauert 1,0 Sekunden.

Für die Erkennung der Anomalie, sowohl mit der euklidischen Distanz als auch mit dem oberen Grenzwert, benötigt dieser Ansatz dieselbe Zeit wie der autoregressive Ansatz. Auch die Zeiten für die Prüfung mit den verschiedenen Varianten verhalten sich gleich.

Genauigkeit Da, wie in Abbildung 5 zu erkennen ist, die Vorhersagen des autoregressiven und des LSTM Modells fast identisch sind, ist die Genauigkeit sowohl für die euklidische Distanz als auch den oberen Grenzwert wie beim autoregressiven Ansatz.

Komplexität Wie auch das autoregressive Modell, kann das LSTM Modell mit *Keras* (Ketkar und Santana 2017) implementiert werden. Die Erstellung des Modells kann dabei anhand der Abbildung 4 erfolgen, jedoch gestaltet sich dies etwas komplexer als beim autoregressiven Modell. Die Berechnungen zur Erkennung von Anomalien sind gleich komplex wie beim autoregressiven Ansatz.

3.4.4 Wahl des besten Ansatzes

Abschließend soll anhand der analysierten Kriterien gewählt werden, welches der beste Ansatz ist, der dann realisiert werden soll. Um die Ansätze vergleichen zu können werden jedem Ansatz für jedes analysierte Kriterium Punkte von null bis fünf vergeben. Für die Realisierung ist es am wichtigsten, dass die Genauigkeit des Ansatzes möglichst gut ist.

Die Bewertungen der verschiedenen Ansätze sind in Tabelle 1 dargestellt. Anhand dieser soll nun das beste Modell ermittelt werden.

Der einfache Ansatz hat sowohl eine sehr schnelle Berechnungszeit als auch eine sehr geringe Komplexität, weshalb er in beiden Aspekten fünf Punkte erhält. Jedoch ist die Genauigkeit im Vergleich zu den anderen Ansätzen recht gering, weshalb er für diesen Aspekt nur zwei Punkte erhält.

Die Berechnungszeit und die Komplexität des autoregressiven Ansatzes sind ebenfalls recht schnell und einfach, jedoch etwas langsamer und komplexer als der einfache Ansatz, weshalb dieser jeweils vier Punkte für diese Aspekte erhält. Dafür ist die Genauigkeit mit diesem Ansatz höher als mit dem einfachen. Der Ansatz mit dem oberen Grenzwert erzielt dabei bessere Ergebnisse als derjenige mit der euklidischen Distanz, jedoch sind beide nicht optimal. Darum erhält der Ansatz mit dem oberen Grenzwert vier, mit der euklidischen Distanz drei Punkte für die Genauigkeit.

Für den LSTM Ansatz werden zwei Punkte für die Berechnungszeit vergeben, da diese deutlich länger ist als die der beiden andern Ansätze. Da auch die Komplexität höher ist als beim autoregressiven Ansatz, werden hierfür drei Punkte vergeben. Die Genauigkeit ist aufgrund der fast identischen Ergebnisse gleich zu bewerten wie beim autoregressiven Ansatz.

Betrachtet man die Bewertungen der Ansätze, so kann festgestellt werden, dass die Genauigkeit beim einfachen Ansatz niedriger ist als bei den anderen Ansätzen. Da die Genauigkeit das wichtigste Kriterium für die Realisierung ist, wird dieser Ansatz nicht verwendet. Bei den Machine Learning Ansätzen ist die Genauigkeit bei den Ansätzen mit der euklidischen Distanz niedriger als bei den Ansätzen mit dem oberen Grenzwert. Da sowohl die Berechnungszeit als auch die Komplexität beim autoregressiven Ansatz besser sind als beim Ansatz mit LSTM, wird im Folgenden der autoregressive Ansatz mit dem oberen Grenzwert für die Realisierung verwendet.

4 REALISIERUNG

Nach der Konzipierung mehrere Ansätze soll nun ein System erstellt werden, das mithilfe des besten Ansatzes die Anomalien im Produktivbetrieb erkennt. Dafür soll zunächst die Architektur des Systems sowie die Umsetzung der verwendeten Skripte erklärt werden. Anschließend wird dargestellt, wie das System im Produktivsystem bereitgestellt wird.

4.1 Architektur des Systems

Zunächst soll die Architektur des zu implementierenden *Anomaly Detection* Systems erklärt werden. Diese ist in Abbildung 6 dargestellt.

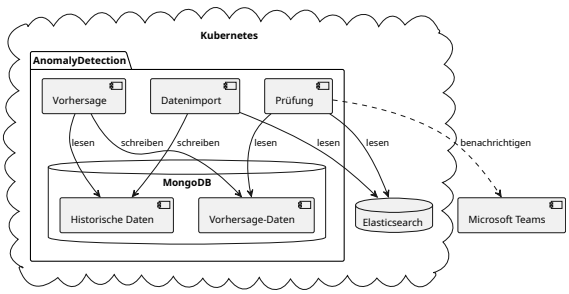


Abbildung 6: Architektur des *Anomaly Detection* Systems

Das *Anomaly Detection* System selbst besteht aus vier Komponenten und benutzt zusätzlich zwei externe Komponenten. Die erste externe Komponente

Ansatz	Genauigkeit	Berechnungszeit	Komplexität
Einfacher	2	5	5
Autoregressiver / euklidische Distanz	3	4	4
Autoregressiver / oberer Grenzwert	*4*	*4*	*4*
LSTM / euklidische Distanz	3	2	3
LSTM / oberer Grenzwert	4	2	3

Tabelle 1: Bewertungen der verschiedenen Ansätze

ist dabei der Elasticsearch Server (Elasticsearch B.V. 2010), der als Datenbank für alle vergangenen Fehlermeldungen agiert. Die zweite ist der Microsoft Teams Server (Ilag und Sabale 2022) der Firma.

Alle Daten im System selbst werden in einer MongoDB (Bradshaw, Brazil und Chodorow 2019) gespeichert, die hauptsächlich aus zwei Komponenten, in diesem Fall Datensammlungen, besteht. Die erste enthält die aufbereiteten Daten aus dem Elasticsearch Server, die zweite die vorhergesagten Verläufe für jeden Tag.

Die Funktionalität des Systems ist auf drei Komponenten aufgeteilt. Dabei existiert zunächst die Datenimport-Komponente, die historische Daten aus dem Elasticsearch Server liest, aufbereitet und in die MongoDB schreibt. Des Weiteren ist die Vorhersage-Komponente vorhanden, die die aufbereiteten historischen Daten liest, damit eine Vorhersage für einen bestimmten Tag erstellt und diese in die Datenbank schreibt. Schließlich existiert noch die Prüfungs-Komponente, die die Vorhersage für den aktuellen Tag sowie die realen Daten des aktuellen Tages liest und anhand dieser Daten entscheidet, ob eine Anomalie vorliegt oder nicht. Falls eine Anomalie erkannt wird, wird ein Verantwortlicher via Microsoft Teams benachrichtigt.

4.2 Umsetzung der Skripte

Die drei funktionalen Komponenten des *Anomaly Detection* Systems sind als *Python* (vanRossum 1995) Skripte realisiert. Im Folgenden soll kurz die Funktionsweise dieser drei Skript-Komponenten erläutert werden.

Datenimport Die Datenimport-Komponente stellt zunächst eine Anfrage an den Elasticsearch Server, wie viele Fehlermeldungen innerhalb der letzten fünf Minuten verzeichnet wurden. Die Anzahl, die der Server als Antwort liefert, wird anschließend zusammen mit dem aktuellen Zeitpunkt, an dem die Anfrage gestellt wurde, in die MongoDB geschrieben. Dabei werden zusätzlich zum normalen Zeitstempel Jahr, Monat, Tag, Stunde, Minute und Wochentag zusätzlich als Attribute gespeichert.

Um einen lückenlosen Datenbestand zu erhalten, läuft das Skript alle fünf Minuten.

Vorhersage Die Vorhersage-Komponente verwendet den erarbeiteten Lösungsansatz aus Abschnitt 3, um eine Vorhersage für den nächsten Tag zu treffen. Dabei werden die Daten der gleichen Wochentage aus der MongoDB gelesen. Diese werden anschließend so aufbereitet, dass sie für das Training des Modells verwendet werden können. Die Implementierung des autoregressiven Modells mit Keras (Ketkar und Santana 2017) ist in Listing 1 dargestellt.

```
model = Sequential([
    Input(shape=(288)),
    Dense(512, activation="tanh"),
    Dense(1024, activation="tanh"),
    Dense(512, activation="tanh"),
    Dense(288),
])
```

Listing 1: Implementierung des autoregressiven Modells in Keras

Ist das Training des Modells abgeschlossen, so wird anhand der Daten des letzten gleichen Wochentages eine Vorhersage erstellt. Diese wird schließlich in die Datenbank geschrieben.

Um immer eine aktuelle Vorhersage zu haben, läuft das Skript jeden Abend kurz vor Mitternacht und erstellt dabei eine Vorhersage für den kommenden Tag.

Prüfung Die Prüfungs-Komponente holt sich zunächst die Vorhersage für den aktuellen Tag aus der MongoDB. Anhand des aktuellen Zeitpunktes wird der vorhergesagte Wert für diesen ermittelt und daraus der obere Grenzwert für den aktuellen *Error-Count* berechnet. Anschließend wird die Anzahl der Fehlermeldungen innerhalb der letzten fünf Minuten, dem in Abschnitt 2.1 definierten Zeitintervall, am Elasticsearch Server abgefragt. Jetzt wird dieser Wert mit dem errechneten Grenzwert verglichen. Wird der Grenzwert überschritten, so wird ein verantwortlicher Mitarbeiter via Microsoft Teams benachrichtigt.

Um keine Anomalien zu übersehen, läuft das Skript alle fünf Minuten.

4.3 Bereitstellung im Kubernetes

Um die Skripte periodisch ausführen zu können, werden diese im Kubernetes (The Kubernetes Authors 2014) als *CronJob* bereitgestellt. Ein Beispiel, wie diese erstellt werden können, ist in Listing 2 dargestellt.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: anomaly-collect
  namespace: anomaly
spec:
  jobTemplate:
    metadata:
      name: anomaly-collect
    spec:
      template:
        spec:
          containers:
            - name: anomaly-collect
              image: anomaly-collect:1.0.0
          restartPolicy: OnFailure
  schedule: "*/5 * * * *"
```

Listing 2: Beispiel *CronJob* im Kubernetes

Das Beispiel in Listing 2 stellt den *CronJob* für das Datenimport-Skript dar. Dabei wird zunächst mit **apiVersion: batch/v1beta1** und **kind: CronJob** spezifiziert, dass ein *CronJob* erstellt werden soll. Anschließend wird in den Metadaten definiert, wie der *CronJob* benannt und in welchem *Namespace* er erstellt werden soll. Jetzt wird festgelegt, welche *Container* ausgeführt werden sollen. Dabei wird für jeden der Name und das *Container Image* definiert. In diesem Beispiel wird ein einzelner *Container* spezifiziert, der das Datenimport-Skript ausführt. Schließlich wird noch spezifiziert, wann der *CronJob* ausgeführt werden soll. Hier wird dieser mit **schedule: "*/5 * * * *"** alle fünf Minuten ausgeführt.

Für alle Skripte des Systems wird ein entsprechender *CronJob* erstellt. Die Ausführungszeiten werden dabei wie in Abschnitt 4.2 erläutert definiert.

5 EVALUATION UND FAZIT

Im Folgenden soll evaluiert werden, inwiefern das entwickelte *Anomaly Detection* System die Anomalien im Produktivbetrieb der Firma erkennt und dies einen Nutzen erbringt.

Als Evaluationsumgebung wird die Produktivumgebung der Firma verwendet. Dabei werden zunächst die Daten der *Error-Count Metrik* der letzten sechs Monate ausgelesen und gespeichert. Anschließend wird über einen Zeitraum von vier Wochen für jeden Wochentag eine Vorhersage berechnet und anhand dieser versucht, Anomalien zu erkennen.

Nach diesen vier Wochen konnte festgestellt werden, dass, wie auch im Beispiel aus Abschnitt 3, Beginn und Ende der regulären Arbeitszeiten sehr gut vom System erkannt werden. Dies ist auch darin zu erkennen, dass Anomalien außerhalb der Arbeitszeiten immer erkannt werden. Problematisch während des Evaluationszeitraumes war jedoch, dass durch einen Hardwareausfall im Produktivsystem sehr viele Fehlermeldungen erzeugt wurden und dadurch das System fast durchgehend eine Anomalie gemeldet hat. Mehrere Mitarbeiter waren mit der Behebung dieses Ausfalls beschäftigt, sodass die Meldung einer Anomalie in diesem Zeitraum als korrekt angesehen werden kann.

Jedoch musste festgestellt werden, dass die Vorhersagen der Tage zwar oftmals nahe dem realen Verlauf gekommen sind, allerdings an einigen Tagen auch etwas weiter entfernt waren von diesem. Dadurch sind einige Anomalien gemeldet worden, die keine sind. Da im Laufe der Entwicklung des Systems die Vorhersagen durch mehr Trainingsdaten stetig besser wurden, kann davon ausgegangen werden, dass die vorhandene Datenmenge noch nicht ausreichend ist für zuverlässige Vorhersagen.

Auch wenn die Ergebnisse des Systems noch nicht optimal sind, erbringt es bereits jetzt einen Nutzen zur Überwachung des Systems, sodass dieses immer für die Kunden zur Verfügung steht. Dieser ist dahingehend vorhanden, dass Ausfallzeiten im Produktivsystem verkürzt werden, was somit auch lange Ausfallzeiten für die Kunden verhindert.

6 ZUSAMMENFASSUNG

In diesem Paper wurde gezeigt, welche Probleme im Produktivsystem der ODAV AG (ODAV AG, Gesellschaft für Informatik und Telekommunikation 1969) noch vorhanden sind (Abschnitt 2). Für diese Probleme wurde anschließend ein Lösungskonzept erstellt, indem mehrere Ansätze konzipiert und anschließend miteinander verglichen wurden (Abschnitt 3). Mit dem Konzept, das am besten abgeschnitten hat, wurde dann ein System erstellt, dass mithilfe dieses Konzeptes Anomalien im Produktivbetrieb erkennen kann (Abschnitt 4). Dieses System wurde schließlich in einem Testzeitraum getestet und anhand der dort gesammelten Ergebnisse evaluiert (Abschnitt 5).

Es zeigt sich, dass mit einem System zur *Anomaly Detection* Ausfallzeiten in einem Produktivsystem, wie es zum Beispiel in der ODAV AG vorhanden ist, verhindert beziehungsweise verkürzt werden können. Jedoch hat sich auch gezeigt, dass für ein solches System, wenn es mit einem *Machine Learning* Modell aufgebaut ist, eine große Menge an Trainingsdaten benötigt wird, um zuverlässige Vorhersagen zu treffen, wobei dies noch nicht verlässlich evaluiert wer-

den konnte.

Als Erweiterung zu diesem Paper könnte man somit evaluieren, wie sich ein solches *Anomaly Detection* System entwickelt, wenn Daten ausgehend von wenigen Monaten bis hin zu mehreren Jahren verfügbar sind. Auch könnte man vergleichen, wie sich das dargestellte System im Vergleich zu anderen generischen System verhält.

LITERATUR

Bradshaw, Shannon, Eoin Brazil und Kristina Chodorow (2019). *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media.

Chandola, Varun, Arindam Banerjee und Vipin Kumar (2009). „Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3, S. 1–58.

Dave, Konark Truptiben (2013). „Brute-force Attack 'Seeking but Distressing'". In: *Int. J. Innov. Eng. Technol. Brute-force* 2.3, S. 75–78.

Day, J.D. und H. Zimmermann (1983). „The OSI reference model". In: *Proceedings of the IEEE* 71.12, S. 1334–1340. DOI: 10.1109/PROC.1983.12775.

Elasticsearch B.V. (2010). *Free and Open Search: The Creators of Elasticsearch, ELK & Kibana | Elastic*. URL: https://www.elastic.co/ (besucht am 28. 09. 2022).

Ilag, Balu N und Arun M Sabale (2022). „Microsoft teams overview". In: *Troubleshooting Microsoft Teams*. Springer, S. 17–74.

Ketkar, Nikhil und Eder Santana (2017). *Deep learning with Python*. Bd. 1. Springer.

ODAV AG, Gesellschaft für Informatik und Telekommunikation (1969). *Homepage - ODAV AG*. URL: https://www.odav.de/ (besucht am 14. 09. 2022).

Oliphant, Travis E (2006). *A guide to NumPy*. Bd. 1. Trelgol Publishing USA.

The Kubernetes Authors (2014). *Kubernetes*. URL: https://kubernetes.io/ (besucht am 28. 09. 2022).

vanRossum, Guido (1995). „Python reference manual". In: *Department of Computer Science [CS]* R 9525.

AUTOREN

Benjamin Eder (M.Sc.) schloss 2023 sein Masterstudium Informatik mit Schwerpunkt *Software Engineering* an der OTH Regensburg ab. Seit 2018 ist er bei der ODAV AG als Werkstudent tätig und ist dort seit 2023 als Softwareentwickler festangestellt.

Benjamin Neumann (Dipl.-Inf. (FH)) beendete 2009 sein Informatikstudium an der Fachhochschule Regensburg. Danach war er bis 2012 bei Continental Regensburg beschäftigt. Seitdem ist er bei der Firma ODAV AG in Straubing, anfangs als Softwareentwickler im DevOps Bereich, seit 2022 als Teamleiter Basissysteme.

Prof. Dr. Frank Herrmann wurde in Münster geboren und studierte Informatik an der Rheinisch-Westfälischen Technischen Hochschule in Aachen. Nach seinem Diplom 1989 arbeitete er bei dem Fraunhofer Institut IITB in Karlsruhe. Während dieser Zeit promovierte er 1996 über Ressourcenbelegungsplanungsprobleme. Von 1996 bis 2003 arbeitete er für die SAP AG in verschiedenen Funktionen, zuletzt als Direktor. Im Jahr 2003 wurde er Professor für Produktionslogistik an der Ostbayerischen Technischen Hochschule in Regensburg. Seine Forschungsthemen sind Planungsalgorithmen, Optimierung und Simulation für die operative Produktionsplanung und -steuerung. Er ist Leiter des Innovations- und Kompetenzzentrums für Produktionslogistik und Fabrikplanung (IPF).