Semesterbegleitende Projektaufgabe im
WPM-Projekt 2 „Machine Learning in Manufacturing“
an der Hochschule Albstadt-Sigmaringen

Vorhersagemodell für die Restlebensdauer von Kugellagern in Elektromotoren

Vorgelegt von Gruppe 3:

Christian Seidler	**Dominik Huber**	**Samuel Dittmann**	**Timmo Kupilas**
100499	100502	100506	100806
seidlech@hs-albsig.de	huberdom@hs-albsig.de	dittmasa@hs-albsig.de	kupilati@hs-albsig.de

Prüfer:

Felix Georg Müller

Plus10 GmbH

Werner-von-Siemens-Straße 6

86159 Augsburg

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die allgemeine Aufgabenstellung der semesterbegleitenden Projektaufgabe besteht darin, ein Vorhersagemodell zu erarbeiten. Hierbei wird davon ausgegangen, dass in einer Produktionslinie der Ausfall von stark belasteten Kugellagern in Elektromotoren die Hauptursache für ungeplante Stillstände ist. Als Resultat entstehen hohe Stillstandkosten und Terminverzüge, weshalb der Produktionsleiter darum bittet, ein Vorhersagemodell inklusive verständlicher Visualisierung zu erstellen. Dieses Modell soll zeigen, in welchem Zustand die jeweiligen Kugellager zu einem gegebenen Zeitpunkt sind und wie groß die verbleibende Zeit zum Ausfall des Bauteils ist. Als Hauptkennzahl soll hierbei die verbleibende Lebensdauer, auch Remaining useful lifetime (kurz: RUL) genannt, verwendet werden.

Die Problemstellung wird hierbei in zwei Teile gegliedert. Zunächst soll ein Verschleiß-Klassifikationsmodell mit drei Klassen aufgestellt werden. Hierdurch soll identifiziert werden, ob sich ein Problem anbahnen könnte. Der zweite Teil der Aufgabe besteht darin, ein Regressionsmodell zu entwickeln, mit dessen Hilfe die verbleibende Lebensdauer prognostiziert werden soll.

2 Verstehen der Problemstellung und der Daten

Der erste Schritt unserer Vorgehensweise besteht darin, einen groben Überblick über die Problemstellung zu bekommen. Hierfür wird zunächst nach bereits existierenden Modellen und Ansätzen gesucht. Grundlage dieser Recherche sind zum einen verschiedene wissenschaftliche Publikationen, zum anderen aber auch Lösungen von ähnlichen Datensätzen auf Kaggle. Auf Basis dieser Literaturrecherche können weitere sinnvolle Schritte geplant werden.

Beim Betrachten des Datensatzes fällt auf, dass eine große Anzahl an Datenpunkten pro Bearing vorliegt (2560 Messpunkte * 1000 Messungen = 2,5 Mio. Messpunkte). Aufgrund dieser großen Datenmenge werden im Nachfolgenden die einzelnen Messpunkte nicht direkt als Feature verwendet. Stattdessen wird jeweils eine Messung mit 0,1 Sek. Dauer (entspricht bei 25,6 kHz 2560 Messwerten) zu einem einzelnen Datenpunkt zusammengefasst. Hierfür werden Ansätze der Signalverarbeitung im Zeit- und Frequenzbereich verwendet. Diese sind im Nachfolgenden Kapitel näher erläutert.

3 Feature Extraction und Filterung

Die Feature Extraction wird batchweise durchgeführt. Hierfür wird schrittweise zunächst eine .csv-Datei eingelesen, die Features extrahiert und das Ergebnis in einem neuen Datensatz abgespeichert.

Dies wird für jede Messung jedes Bearing getrennt wiederholt. In Summe werden 36 Features pro .csv-Datei berechnet.

Zeitreihenanalyse

Im Rahmen der Zeitreihenanalyse werden 13 verschiedene Signalmerkmale berechnet. Da diese jeweils für die in horizontaler und vertikaler Richtung gemessen Beschleunigung berechnet werden, ergeben sich insgesamt 26 Merkmale. Diese umfassen: Mittelwert, absoluter Mittelwert, Standardabweichung, Schiefe (Skew), Kurtosis, RMS, absoluter maximaler Wert, Peak-to-Peak, Crest Faktor, Shape Faktor, Impuls, Clearance Faktor, Entropie, gleitenden Mittelwert mit Fensterlänge 640, absoluter gleitender Mittelwert mit Fensterlänge 640. Die genannten Merkmale sind aus Literaturrecherchen und Gesprächen mit dem Dozenten abgeleitet worden (siehe (Juodelyte, Cheplygina, Graversen, & Bonnet, 2022), (Huang, Farahat, & Gupta, 2020)).

Frequenzanalyse

Aus wissenschaftlichen Publikationen ist bekannt, dass Verschiebungen im Frequenzspektrum des gemessenen Signals für Bearings existieren, z.B. (Sutrisno, Oh, Vasan, & Pecht, 2012). Daher wird zusätzlich zur Zeitsignalanalyse die Frequenzanalyse verwendet. Die Idee dahinter besteht darin, das vorliegende Zeitsignal aus dem Zeitsignalraum in den Frequenzraum zu überführen. Hierdurch kann ein Signal in den einzelnen Frequenzanteilen dargestellt werden. Grundlage hierfür ist die sog. Fourier-Reihenentwicklung. Diese besagt, dass sich periodische Signale als Überlagerung von unendlich vielen Sinusfunktionen approximieren lassen (Hochschule Karlsruhe, kein Datum). Die verwendeten Sinusfunktionen unterscheiden sich hierbei in der Amplitude und der Phase. Mit Hilfe von Spektrogrammen kann dann zum Beispiel ermittelt werden, wie stark jeder Frequenzbereich im zeitlichen Verlauf des Signals ausgeprägt war. Die in Python gewählte FFT-Implementation liefert gemäß dem Nyquist-Kriterium im 50 Hz Abstand bis zur Hälfte der Abtastfrequenz (= 12800 Hz) jeweils die zugehörige Amplitude. Da sich das Zeitsignal entgegen der Annahme der Fourier-Transformation in unserem Fall nicht periodisch wiederholt, wird ein Kaiser-Fenster verwendet, um Verzerrungen bei der Transformation zu minimieren. Für dieses Projekt sind dies für jede Beschleunigungsrichtung somit $\frac{25600\ Hz}{2*50\ Hz} = 256$ Merkmale. Um die Komplexität zu verringern, werden nur die jeweils 5 am stärksten ausgeprägtesten Frequenzanteile verwendet. In Summe ergibt dies 10 Merkmale für die Frequenzanalyse.

Das Zusammenführen der Merkmale aus der Zeit- und Frequenzanalyse liefert die für die Klassifikation und Regression verwendete Datenbasis. Dieser Schritt wird beim Labeln der Daten durchgeführt.

4 Labeln der Daten

Startpunkt dieses Schrittes ist zunächst das Einlesen der im vorherigen Schritt aufbereiteten Datensätze. Die Labelung für die Klassifikation besteht aus den drei Klassen „Bearing zerstört“, „Verschleiß erkennbar“ und „kein Verschleiß“. Im Laufe der Projektarbeit wurden drei verschiedene Ansätze zur Labelung der Daten implementiert und bewertet.

Beim ersten Ansatz wird der maximale Wert betrachtet. Sobald das Merkmal „absoluter maximaler Wert“ den Grenzwert von 20 g überschreitet, gilt das Bearing als kaputt. Der Nachteil dieses Ansatzes ist, dass kurzzeitige Ausreißer im Signal die Klassifikation verzerren. Dies führt zur Überrepräsentierung der Klasse „Bearing zerstört“. Ein Beleg dafür ist Bearing 1_1, welches nach der Hälfte der Zeit einen Ausreißer in vertikaler-Richtung aufweist (siehe Anhang 1).

Der zweite Ansatz besteht darin, den gleitenden Mittelwert als Grundlage für die Labelung zu verwenden. Durch den gleitenden Mittelwert werden die gemessenen Signale geglättet und Ausreißer über 20 G, welche oftmals nur über einen kurzen Zeitraum bestehen, herausgefiltert. Eine Schwachstelle dieses Ansatzes ist, dass der gleitende Mittelwert mit Fensterlänge 640 nie 20 G überschreitet. Die im Anhang 2 dargestellten Kurven legen nahe, dass der gleitende Mittelwert für eine Glättung um den Faktor 10 sorgt. Somit ist der Grenzwert für die Klasse „Bearing zerstört“ anstatt bei 20 G nun bei 2 G.

Der dritte Ansatz unterscheidet sich nur geringfügig vom zweiten. Der Unterschied besteht darin, dass beim ersten und beim zweiten Ansatz die Annahme gilt, dass das Bearing zum letzten Messzeitpunkt in jeden Fall der Klasse „Bearing zerstört“ zugeordnet werden muss – selbst, wenn das Bearing nie die gewählten Grenzwerte während der Messung überschritten hat. Beim dritten Ansatz wird diese Annahme nicht getroffen. Überschreitet das Bearing die Grenzwerte während der Messung nicht, dann besteht die Klasse „Bearing zerstört“ aus 0 Datenpunkten.

Die Unterscheidung der beiden Klassen „Verschleiß erkennbar“ und „kein Verschleiß“ erfolgt bei allen Ansätzen über die verbleibende Zeit bis zum Ausfall bzw. bis zum letzten Messzeitpunkt. Ein Bearing wird in die Klasse „Verschleiß erkennbar“ eingeteilt, wenn weniger als 1,225 h bis zum Ausfall bzw. zum Messende verbleiben. Diese Einteilung wurde mit Hilfe des Medians der gemessenen Zeiten bis zum tatsächlichen Ende für alle Bearings ermittelt. Die nachfolgende

Tabelle 1 stellt die gemessen Zeiten dar. Der Median von 2,45 h wird halbiert, so dass jedes Bearing über die zwei Klassen „Verschleiß erkennbar“ und „kein Verschleiß“ verfügt. Die im Nachfolgenden dargestellten Ergebnisse für die Klassifikation basieren auf der Labelung mit dem dritten Ansatz.

Tabelle 1: Exemplarische Darstellung der Ermittlung der Klassengrenzen für die Klassen "Verschleiß erkennbar" und "Kein Verschleiß" für Ansatz 3

Bearing	Anzahl Datenwerte	Zeit bis zum Ausfall [h]
1_1	2673	7,4
1_2	851	2,4
2_1	900	2,5
2_2	786	2,2
3_1	493	1,4
3_2	1628	4,5
Mittelwert	1222	3,40
Median	876	2,45

5 Verwendete Algorithmen und Methodik

Der ursprüngliche Train-Test-Split des Datensets wird beibehalten. Zusätzlich werden alle Conditions zusammen als ein einzelner Datensatz betrachtet.

5.1 Klassifikation

Algorithmen

Im Rahmen des Projekts wurden zahlreiche Klassifikationsalgorithmen aus der Python-Bibliothek „Scikit-Learn“ verwendet und deren Ergebnisse miteinander verglichen. Die nachfolgende Tabelle 2 zeigt die angewandten Algorithmen und deren gewählten Parameter. Für die 5 Algorithmen mit den besten Ergebnissen für die Testdaten wurde zusätzlich ein Hyperparametertuning durchgeführt (siehe nächster Abschnitt). Zur Gewährleistung der Reproduzierbarkeit wurde, sofern möglich, der Parameter „random_state“ der Algorithmen auf „14“ gesetzt.

Tabelle 2: Übersicht der verwendeten Algorithmen zur Klassifikation

Algorithmus	Gewählte Parameter	Hyperparameter-Tuning?
Linearer SVC	Kernel=“linear“, C=0,025	Nein
Nichtlinearer SVC	kernel="rbf", gamma=2, C=1	Nein
Random Forest	max_depth=10, n_estimators=20, max_features=1	Ja
AdaBoost	-	Ja
k-NN	n_neighbors=100	Nein
Gaussian Process	kernel=1.0 * RBF(1.0)	Nein

Algorithmus	Gewählte Parameter	Hyperparameter-Tuning?
Decision Tree	criterion='gini', max_depth=4, min_samples_leaf=0.1, min_samples_split=0.1	Ja
MLPC	alpha=1, max_iter=1000	Ja
Gaussian NB	-	Nein
Quadratic Discriminant Analysis	-	Nein
Gradient Boosting	-	Ja

Hyperparametertuning

Für die in der vorher dargestellten Tabelle 2 markierten Algorithmen wurde ein Hyperparametertuning durchgeführt. Für das Tuning wird „GridSearchCV“ von „scikit-learn“ verwendet. Als Bewertungsmetrik dient die Accuracy. Zur Kreuzvalidierung wird die Klasse „RepeatedStratifiedKFold“ verwendet, mit den Parametern n_splits=5, und n_repeats=2, d.h. die Trainingsdaten werden 5-mal gesplittet und jeder Split wird 2-mal durch das Modell bewertet. Somit wird für jede Parameterkonfiguration 10-mal die Modellperfomance beurteilt. Der jeweils verwendete Suchbereich ist in der Datei: „Hyperaparameter Tuning.txt“ dargestellt. Die unten dargestellte Tabelle 3 zeigt die getunten Modelle, für welche im Kapitel 6.1 die Ergebnisse der Testdaten zu sehen sind. Die getunten Modelle erzielen im Vergleich zu den ursprünglichen Modellen im Mittel eine Verbesserung um 10 Prozentpunkte auf den Testdatensatz (siehe Dokumente unter „Dokumente und Ergebnisse“ → „Hyperparametertuning Klassifikation“).

Tabelle 3: Parameter der getunten Algorithmen zur Klassifikation

Algorithmus	Parameter nach Tuning
Random Forest	max_depth=15, n_estimators=25, max_features=4, random_state=14
MLPC	alpha=0.001, max_iter=900, activation='logistic', random_state=14
AdaBoost	n_estimators=100, learning_rate=0.1, ='SAMME', random_state=14
Gradient Boosting	learning_rate=0.1, max_depth=5, max_features='sqrt', n_estimators=300, random_state=12
Decision Tree	random_state=14, criterion='gini', max_depth=4, min_samples_leaf=0.1, min_samples_split=0.1

5.2 Regression

Regressoren von scikit-learn

Zur Regression wurden folgende Algorithmen verwendet: Lineare Regression, Ridge Regression, Lasso Regression, Elastic Net, Random Forest Regressor, XGBoost (mit den Parametern objective="reg:squarederror", colsample_bytree=0.3, learning_rate=0.1, max_depth=5, alpha=10, n_estimators=10), sowie ein Support Vector Regressor (mit den Paramtern kernel="rbf", C=100, gamma=0.1, epsilon=0.1). Sofern möglich, wurde der Parameter „random_state“ auf „11“ gesetzt.

Deep-Learning

Neben den oben genannten Algorithmen werden zusätzlich zwei Architekturen von neuronalen Netzen zur Regression verwendet: Feed-Forward Netzwerke, sowie Recurrent Neural Networks (LSTM). Der genaue Aufbau der verwendeten Netzwerke ist im Code unter dem Skript „deep_learning.py“ dargestellt.

Automatisierte Architekturermittlung mittels autokeras

Abschließend wurde die Python-Bibliothek „autokeras“ verwendet, um automatisiert die Netzwerk-Architektur zu ermitteln. Hierfür wird ein StructuredDataRegressor verwendet mit der Loss-Vorgabe „Mean Absolute Error“. In den vorgeschlagenen Architekturen von autokeras ist ein CategoryEncodingLayer enthalten. Falls diese Layer verwendet wird, ist das Netzwerk jedoch nicht mehr in der Lage, den Abwärtstrend der RUL bis zum Ausfall gut widerzuspiegeln (siehe Anhang 3). Daher werden die vorgeschlagenen Modelle im Nachfolgenden ohne diesen Layer verwendet.

6 Ergebnisse

6.1 Klassifikation

Die nachfolgenden Abbildungen zeigen exemplarisch die Ergebnisse für die beiden getunten Modelle „Random Forest“ und „MLPC“. Die Ergebnisse der anderen getunten Modelle sind in der Abgabe unter „Dokumente → Ergebnisse getunte Klassifikationsmodelle“ zu finden.

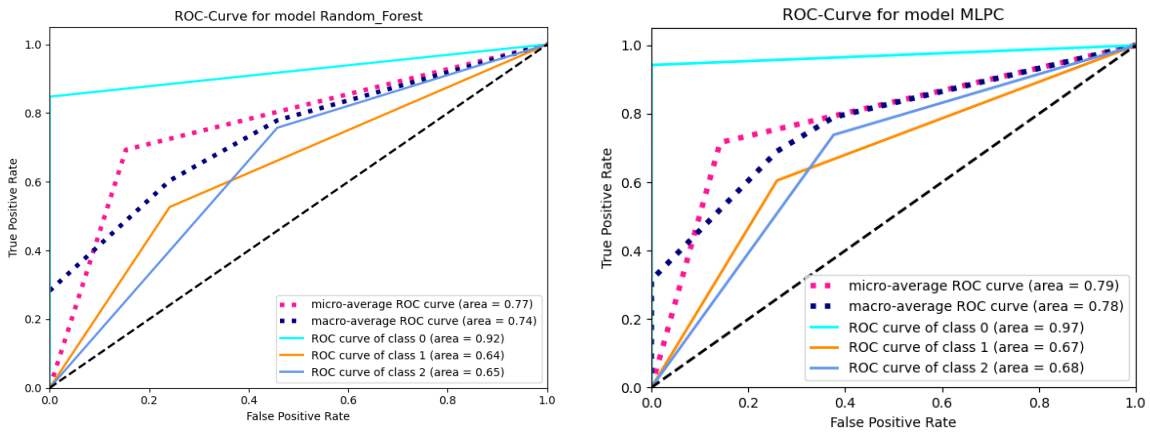


Abbildung 1: ROC-Kurven für die Modelle „Random Forest“ und „MLPC“

Aus den ROC-Kurven geht hervor, dass die Klasse „Bearing zerstört“ (Klassenlabel 0) von beiden Algorithmen mit sehr hoher Genauigkeit korrekt erkannt wird. Da die Area Under Curve (AUC) für jede Klasse größer als 0,5 ist, sind die Algorithmen besser als raten. Im Mittel liegt die AUC für alle Klassen knapp um die 0,75 (siehe Macro-Average ROC) über alle Conditions. In der nachfolgenden Tabelle 4 sind die Ergebnisse für die verschiedenen Conditions gegenübergestellt.

Tabelle 4: Vergleich von Accuracy und Precision für die beiden Modelle "Random Forest" und "MLPC", getrennt nach Conditions

	Alle Conditions zusammen	Durchschnitt von Condition 1	Durchschnitt von Condition 2	Durchschnitt von Condition 3
Random Forest	Recall: 0.6933 Precision: 0.6838 Accuracy: 0.6933 F1-Score: 0.6874	Recall: 0.6711 Precision: 0.7311 Accuracy: 0. 6711 F1-Score: 0. 6748	Recall: 0.4995 Precision: 0. 7119 Accuracy: 0. 4995 F1-Score: 0. 5446	Recall: 0.2840 Precision: 0.9971 Accuracy: 0.2840 F1-Score: 0.4387
MLPC	Recall: 0.7161 Precision: 0.6972 Accuracy: 0.7161 F1-Score: 0.6896	Recall: 0.6392 Precision: 0.6723 Accuracy: 0.6392 F1-Score: 0.6299	Recall: 0.4743 Precision: 0.7472 Accuracy: 0.4743 F1-Score: 0.4888	Recall: 0.2386 Precision: 0.9971 Accuracy: 0.2386 F1-Score: 0.3814

Aus der oben gezeigten Tabelle geht hervor, dass die Modelle je nach Condition unterschiedliche Ergebnisse erzielen. Auffallend ist, dass die Modelle für Condition 1 am besten performen. Bei Condition 3 legen die Metriken „Accuracy“ und „F1-Score“ nahe, dass die Modelle für diese Rahmenbedingungen wenig aus den Trainingsdaten gelernt haben. Die Tabelle zeigt, dass in einer zukünftigen Untersuchung geprüft werden sollte, ob ein getrenntes Betrachten der Conditions, d.h. das Erstellen von drei verschiedenen Modellen für die Conditions bessere Ergebnisse liefert.

6.2 Regression

„Normale“ Algorithmen

Die nachfolgende Abbildung 2 stellt die vorhergesagten Werte für die RUL von Bearing 1_3 dem tatsächlichen Verlauf für verschiedene Regressionsmodelle gegenüber. Daraus ist ersichtlich, dass bis auf den RandomForestRegressor und den XGBoost-Algorithmus keiner der anderen Algorithmen den Abwärtstrend der RUL wiedergeben. Diese Erkenntnis wird durch die schlechten Scores für das gesamte Testset (und Bearing 1_3!) unterstützt. Die Scores sind in der Abgabe unter „Dokumente und Ergebnisse -> Ergebnisse Regressionsmodelle“ dargestellt. Die schlechte Performance der „normalen“ Algorithmen kann verschiedene Ursachen haben. Eine Erklärung ist, dass weitere Features notwendig sind, um den Trend wiederzugeben. Eine andere Erklärung ist, dass die 36 verwendeten Merkmale, d.h. 36 Dimensionen, zu komplex für die „normalen“ Algorithmen sind. Daher werden anschließend neuronale Netzwerke angewandt, um zu überprüfen, ob diese bessere Performance erzielen. Im nachfolgenden Abschnitt sind die Ergebnisse für ein Feed-Forward-Netzwerk und ein LSTM dargestellt.

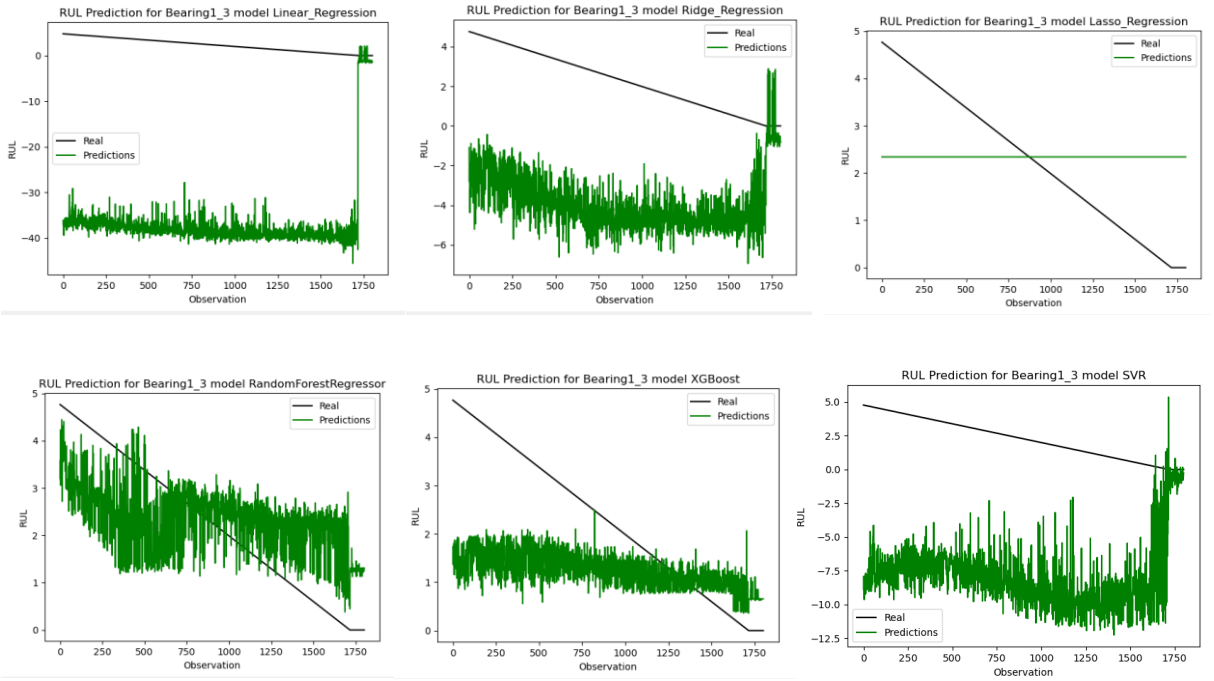


Abbildung 2: Vergleich der tatsächlichen RUL mit den vorhergesagten Werten für verschiedene Modelle für Bearing 1_3

Deep-Learning

Für das Feed-Forward-Netzwerk wurden die Trainingsdaten in eine zufällige Reihenfolge gebracht, um zu vermeiden, dass das Netz ein Bias zu den Werten am Anfang der Messung aufweist. Es wurden 50

Epochen zum Trainieren der Modelle verwendet. Die Batch-Size beträgt beim LSTM 128 und bei den Feed-Forward-Netzwerken 16.

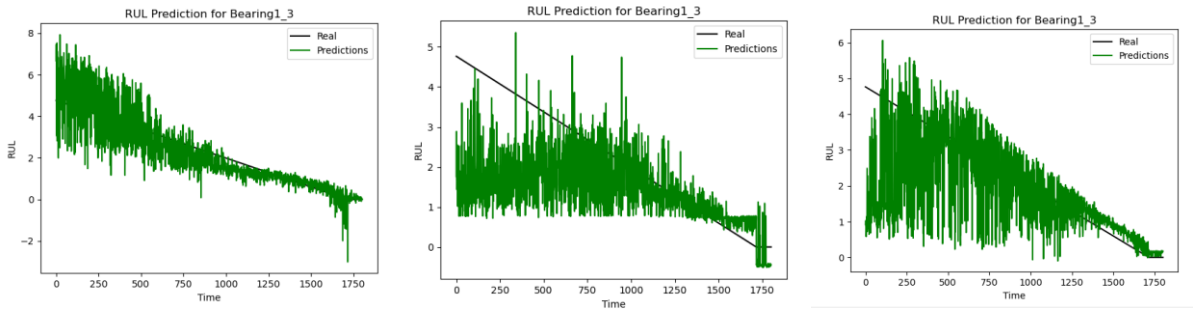


Abbildung 3: Tatsächliche Werte vs. Vorhersage der RUL für Feed-Forward-Netzwerk (links) und LSTM-Modell (Mitte) und Feed-Forward-Architektur nach Autokeras-Vorschlag

Aus der obigen Abbildung geht hervor, dass das Feed-Forward-Netzwerk für Bearing 1_3 die beste Vorhersage von allen untersuchten Deep-Learning Modellen liefert. Der R^2-Score liegt hier bei 64 % und der mittlere absolute Fehler bei 0,6 Stunden. Dies ist auch in den Metriken (siehe Datei „Regression_Ergebnisse_Feed-Forward.txt“) ersichtlich. Über alle Bearings schneiden die neuronalen Netzwerke jedoch alle ähnlich gut ab (siehe Dateien „Regression_Ergebnisse_LSTM.txt“ und „Regression_Ergebnisse_Autokeras.txt“). Auffallend ist, dass das Modell mit der von Autokeras vorgeschlagene Architektur als einziges keine negative RUL-Werte aufweist.

7 Fazit

Die Vorgehensweise des Projekts entspricht bei näherer Betrachtung der Methodik nach CRISP-DM. Es wurden alle Schritte bis auf „Deployment“ durchlaufen.

Im Rahmen des Projektes wurde gezeigt, dass mit Hilfe von ML-Ansätzen die RUL für einzelne Bearings mit hoher Genauigkeit vorhergesagt werden kann. Bei näherer Betrachtung der Scores zeigt sich, dass der Random Forest ähnliche Ergebnisse wie die ausprobierten Deep-Learning Ansätze liefert. Das größte Problem stellt die Generalisierung für alle Bearings dar. Dies ist daran ersichtlich, dass sich die Modellperformance je nach Bearing stark unterscheidet. Die Modelle liefern für Condition 1 die besten Ergebnisse – bei Bearings in den anderen Conditions sind die Modelle nicht in der Lage, den Abwärtstrend der RUL abzubilden.

Folgende Punkte könnten an unserer Lösung in Zukunft verbessert werden:

- Die neuronalen Netze wurden nur einmal trainiert. Da die Gewichte am Anfang des Trainings zufällig initialisiert werden, ist das Ergebnis nicht reproduzierbar. Eine Verbesserung liegt darin, die Modelle mehrfach zu trainieren und den Mittelwert der Ergebnisse zu bilden.

- Außer dem LSTM wurde kein weiterer Algorithmus für Zeitreihen untersucht.
- Die Ergebnisse der FFT wurden reduziert. Es sollte untersucht werden, ob bessere Ergebnisse erzielt werden, wenn die möglichen 256 Merkmale der FFT in den Modellen verwendet werden.
- Es sollte überprüft werden, ob bessere Ergebnisse erzielt werden, wenn Modelle getrennt für die einzelnen Conditions erstellt werden.
- Es sollte untersucht werden, ob die Extraktion weiterer Merkmale aus dem Zeitsignal, wie beispielsweise die Energie, mit Bibliothek tsfresh, bessere Ergebnisse liefert.
- Die Labelung der Daten für die Klassifikation wurde über die verbleibende Zeit bis zum ersten Ausfall durchgeführt. Hier sollte untersucht werden, ob nicht über andere Metriken ein besseres Klassifikationsergebnis erzielt werden kann.

8 Literaturverzeichnis

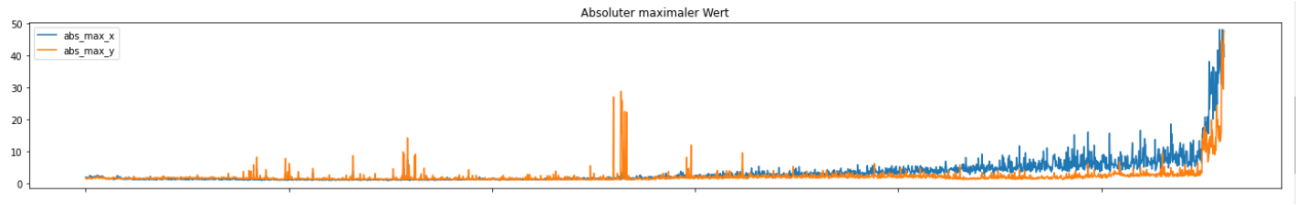
Hochschule Karlsruhe. (kein Datum). *Systemtheorie Online*. Von Fourier-Reihe: https://www.eit.hs-karlsruhe.de/mesysto/teil-a-zeitkontinuierliche-signale-und-systeme/spektrum-eines-signals/fourier-reihe.html abgerufen

Huang, W., Farahat, A., & Gupta, C. (2020). *Similarity-based Feature Extraction from Vibration Data for Prognostics.* Santa Clara, USA: ANNUAL CONFERENCE OF THE PROGNOSTICS AND HEALTH MANAGEMENT SOCIETY.

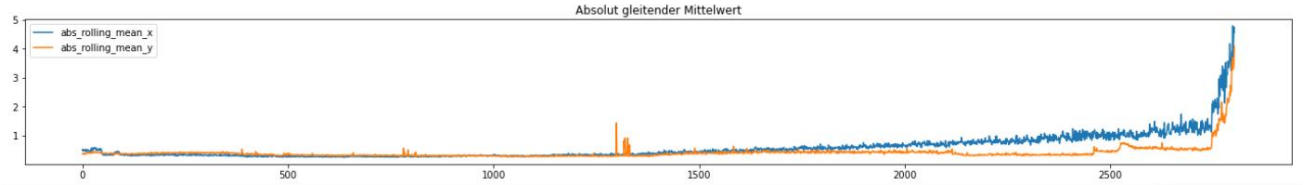
Juodelyte, D., Cheplygina, V., Graversen, T., & Bonnet, P. (2022). *Predicting Bearings' Degradation Stages for Predictive Maintenance in the Pharmaceutical Industry.* Copenhagen, Denmark: IT University of Copenhagen.

Sutrisno, E., Oh, H., Vasan, A. S., & Pecht, M. (2012). *Estimation of Remaining Useful Life of Ball Bearings using Data Driven Methodologies.* University of Maryland, College Park: Center for Advanced Life Cycle Engineering (CALCE).

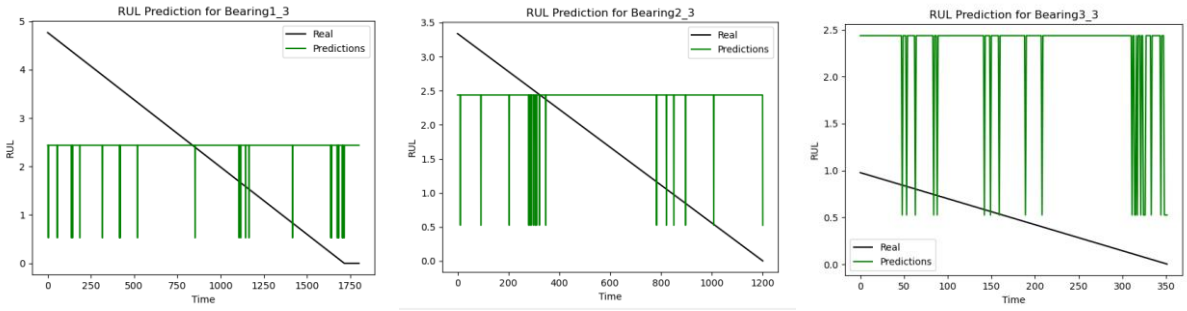
9 Anhang



Anhang 1: Darstellung des Verlaufs des absoluten maximalen Wertes für Bearing 1_1 in horizontaler (blau) und vertikaler (orange) Richtung



Anhang 2: Darstellung des Verlaufs des absolut gleitenden Mittelwerts für Bearing 1_1 in horizontaler (blau) und vertikale (orange) Richtung



Anhang 3: Darstellung der RUL-Vorhersage für ein Feed-Forward-Netzwerk, unter Verwendung des Layer "CategoryEncoding", für je ein Bearing pro Condition. Die Kurven für die anderen Bearing weisen denselben Trend auf.