



Technische Hochschule Nürnberg

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Elektronische und Mechatronische Systeme

Abschlussarbeit Master von

Maksim Skorobogatov

Vergleich und Optimierung der Algorithmen des maschinellen  
Lernens auf deren Eignung zur Berechnung des Volumenstroms  
einer Kreiselpumpe aus Frequenzumrichterdaten

Betreut von:

Prof. Dr.-Ing. Ronald Schmidt-Vollus

M.Sc. Tristan Strattner

Technische Hochschule Nürnberg

WiSe2022/2023

## Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Maksim

Vorname: Skorobogatov

Matrikel-Nr.: 3580170

Fakultät: Elektro-, Feinwerk-, Informationstechnik

Studiengang: Elektronische und Mechatronische Systeme

Semester: Wintersemester 2022/2023

### **Titel der Abschlussarbeit:**

Vergleich und Optimierung der Algorithmen des maschinellen Lernens auf deren Eignung zur Berechnung des Volumenstroms einer Kreiselpumpe aus Frequenzumrichterdaten

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

24.11.2022 Nürnberg

Ort, Datum, Unterschrift Studierende/Studierender

---

### **Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit**

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit  genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von 0 Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

24.11.2022 Nürnberg

Ort, Datum, Unterschrift Studierende/Studierender

**Datenschutz:** Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: <https://www.th-nuernberg.de/datenschutz/>

# Inhaltsverzeichnis

Eidesstattliche Erklärung.....	3
1. Einleitung .....	11
1.1. Motivation.....	11
1.2. Zielsetzung .....	11
1.3. Forschungsmethodik .....	12
2. Stand der Technik und die Grundlagen.....	12
2.1. Prozesstechnik .....	12
2.2. Maschinelles Lernen .....	12
2.3. Zukünftige Richtungen für Softsensorik .....	13
3. Stand der Wissenschaft .....	13
3.1. Maschinell lernender Wetter-Soft-Sensor zur Steuerung von Kläranlagen .....	13
3.2. Vorhersage von Sorptionsdampf-Methan-Reformierungsprodukten aus auf maschinellem Lernen basierenden Soft-Sensor-Modellen.....	14
3.3. Auf maschinellem Lernen basierendes Soft-Sensor-Modell für die BSB-Schätzung .....	15
4. Entwicklung der Lösung .....	18
4.1. Material und Methoden des maschinellen Lernens.....	18
4.1.1. Lineare Regression.....	18
4.1.2. Lasso Regression.....	19
4.1.3. Ridge Regression .....	20
4.1.4. ElasticNet Regression .....	20
4.1.5. Künstliche neuronale Netze (KNN) .....	20
4.1.6. K-nächste-Nachbarn-Algorithmus .....	21
4.1.7. Decision Tree .....	23
4.1.8. Random Forest .....	24
4.1.9. Support-Vektor-Maschine.....	25
4.1.10. Ergebnis der Recherche .....	25
4.2. Anforderungen/Bedarfsanalyse .....	26
4.3. Auswahl der geeigneten Daten.....	27
4.3.1. Der Prozess der Datenerstellung .....	27
4.3.2. Datenerstellung in TIA Portal-Programm .....	28
4.4. Entwurf und Implementierung .....	30
4.4.1. Pumpversuchstand und Simulation der Arbeit.....	30
4.4.2. Lineare Regression zur Vorhersage von Daten.....	32
4.4.3. Lasso Regression zur Vorhersage von Daten.....	36
4.4.4. Ridge Regression zur Vorhersage von Daten .....	36

4.4.5. ElasticNet Regression zur Vorhersage von Daten .....	37
4.4.6. Künstliches neuronales Netz (KNN) zur Vorhersage von Daten.....	38
4.4.7. K-nächste-Nachbarn- <i>Algorithmus</i> ( <i>KNNA</i> ) zur Vorhersage von Daten.....	39
4.4.8. Decision Tree Regression zur Vorhersage von Daten .....	40
4.4.9. Random Forest Regression zur Vorhersage von Daten .....	41
4.4.10. Support-Vektor-Maschine.....	42
4.5. Die Methodiken zu Optimierung von maschinellen Lernalgorithmen .....	43
4.5.1. Principal Component Regression (PCR) (Hauptkomponentenregression) .....	43
5. Systematischer Vergleich verschiedener Algorithmen zur Berechnung des Volumenstroms auf Frequenzumrichterdaten.....	45
5.1. Die Geschwindigkeit der Algorithmen.....	46
5.1.1. Ein Trainingsmodell, das innerhalb von 10 Minuten trainiert wurde .....	46
5.1.2. Ein Trainingsmodell, das innerhalb von 25 Minuten trainiert wurde .....	47
5.2. Beschreibung des Trainingsmodells, das in den Tests verwendet wird .....	48
5.3. Die Genauigkeit der Datenvorhersagung in Echtzeit .....	49
5.3.1. ElasticNet Regression zum Vorhersagen von Daten in Echtzeit .....	50
5.3.2. Künstliches neuronales Netz (KNN) zum Vorhersagen von Daten in Echtzeit.....	51
5.3.3. K-nächste-Nachbarn-Algorithmus zum Vorhersagen von Daten in Echtzeit .....	52
5.3.4. Decision Tree zum Vorhersagen von Daten in Echtzeit .....	53
5.3.5. Random Forest zum Vorhersagen von Daten in Echtzeit .....	55
5.3.6. Support-Vektor-Maschine (SVM) zum Vorhersagen von Daten in Echtzeit.....	56
5.3.7. Allgemeiner Vergleich der Genauigkeit von Algorithmen zum Vorhersagen bei Verwendung in Echtzeit.....	57
5.4. Die Genauigkeit der Datenvorhersagung unter gleichen Bedingungen .....	58
5.4.1. ElasticNet Regression zum Vorhersagen von Daten unter gleichen Bedingungen.....	59
5.4.2. Künstliches neuronales Netz (KNN) zum Vorhersagen von Daten unter gleichen Bedingungen .....	60
5.4.3. K-nächste-Nachbarn-Algorithmus zum Vorhersagen von Daten unter gleichen Bedingungen .....	61
5.4.4. Decision Tree zum Vorhersagen von Daten unter gleichen Bedingungen .....	62
5.4.5. Random Forest zum Vorhersagen von Daten unter gleichen Bedingungen .....	63
5.4.6. Support-Vektor-Maschine (SVM) zum Vorhersagen von Daten unter gleichen Bedingungen .....	64
5.4.7. Allgemeiner Vergleich der Genauigkeit von Algorithmen zum Vorhersagen von Daten unter gleichen Bedingungen .....	65
5.5. Vergleich verschiedener Parameter neuronaler Netze und deren Einfluss auf die Genauigkeit der Datenvorhersage.....	66
5.5.1. Units.....	67

5.5.2. Aktiviation .....	68
5.5.3. Optimizer .....	70
5.6. Praktische Vor- und Nachteile der betrachteten Algorithmen bei der Verwendung für den Pumpversuchstand.....	71
5.7. Aus der Literatur entnommenen Vor- und Nachteile der betrachteten Algorithmen bei der Verwendung für den Pumpversuchstand.....	75
6. User Interface zur Arbeit am Pumpversuchstand (GUI) .....	78
6.1. Warum wurde die Dear PyGUI-Bibliothek für dieses Projekt ausgewählt? .....	78
6.2. Wie funktioniert die Benutzeroberfläche? .....	79
6.3. Beschreibung der Bedienschritte der Benutzeroberfläche .....	80
6.3.1. Der Ausgangszustand der Benutzeroberfläche .....	80
6.3.2. Der Zustand der GUI-Interface, wenn dritte Schaltfläche gedrückt wird.....	81
6.4. Erläuterung des Codes, der beim Erstellen der GUI verwendet wurde.....	82
6.4.1. Erstellen des Hauptfensters und eines Arrays mit allen Daten .....	82
6.4.2. Diagramme aktualisieren und Daten abrufen.....	82
7. Zusammenfassung.....	84
Literaturverzeichnis.....	86

## Abkürzungsverzeichnis

BSB .....	<i>Biologischen Sauerstoffbedarf</i>
COD .....	<i>Chemischer Sauerstoffbedarf</i>
KNN .....	Künstliches neuronales Netz
<i>KNNA</i> .....	<i>K-nächste-Nachbarn-Algorithmus</i>
<i>LASSO</i> .....	<i>Least Absolute Shrinkage and Selection Operator</i>
ML.....	<i>Maschinelles Lernen</i>
N-Ammoniak.....	<i>Ammoniakkonzentration</i>
NaN .....	<i>Not-a-Number</i>
NCT.....	<i>Nuremberg Campus of Technology</i>
<i>PCR</i> .....	<i>Principal Component Regression</i>
Q <i>Zuflussrate</i>	
RF .....	<i>Random Forest</i>
<i>SVM</i> .....	<i>Support-Vektor-Maschine</i>
<i>TIA Portal</i> .....	<i>Totally Integrated Automation Portal</i>

# Abbildungsverzeichnis

Abbildung 1. Beziehung zwischen Q-, COD- und N-Ammoniakmaßen nach Skalierung und starker Filterung aus [3, S.8] .....	14
Abbildung 3. Tatsächliche und vorhergesagte Diagramme nach das Random-Forest- Algorithmus für: a) [H <sub>2</sub> O], b) [CO], c) [H <sub>2</sub> ] (aus [4. S. 7]) .....	15
Abbildung 2. Tatsächliche und vorhergesagte Diagramme nach den künstlichen neuronalen Netzen (KNN) für: a) [H <sub>2</sub> O], b) [CO], c) [H <sub>2</sub> ] (aus [4. S. 7]) .....	15
Abbildung 4. Cloud vs Edge Vorhersagezeit von Datensätzen von Experiment 1 aus [5, S. 971] .....	16
Abbildung 5. Cloud vs Edge Vorhersagezeit von Datensätzen von Experiment 2 aus [5, S. 973] .....	17
Abbildung 6. Beispiel für lineare Regression aus [11] .....	19
Abbildung 7. Die Architektur eines neuronalen 2-Schicht-Netzwerks aus [7] .....	21
Abbildung 8. Beispiel für den Algorithmus k-nächsten Nachbarn aus [8] .....	21
Abbildung 9. Berechnung der Euklidische Entfernung aus [8].....	22
Abbildung 10. Beispiel für einen einfachen Entscheidungsbaum aus [9].....	23
Abbildung 11. Optimale trennende Hyperebene aus [9] .....	25
Abbildung 12. Prozess der Datenerstellung .....	27
Abbildung 13. Datenerstellung in TIA Portal .....	29
Abbildung 14. Pumpversuchstand im Betrieb.....	30
Abbildung 15. Fließschema des Pumpversuchstandes.....	30
Abbildung 16. Handbetrieb des Motors A .....	31
Abbildung 17. Auswahl des Funktionstyps.....	31
Abbildung 18. Beispiel für die Genauigkeit der Datenvorhersage (Volumenstrom).....	36
Abbildung 19. Diagramm der RMSE gegen die Anzahl der Hauptkomponenten .....	44
Abbildung 20. RMSE-Ergebnisse der Regressionsmodelle. Je niedriger der RMSE, desto besser das Modell .....	44
Abbildung 21. Vorhersage des Volumenstromes am Pumpversuchstand.....	45
Abbildung 22. Vergleich von Trainingszeit und Vorhersagezeit mit einem 10-Minuten Trainingsmodell .....	47
Abbildung 23. Vergleich von Trainingszeit und Vorhersagezeit mit einem 25-Minuten Trainingsmodell .....	47
Abbildung 24. Erstellen eines Trainingsmodells zum Testen von Algorithmen.....	49
Abbildung 25. Vorhersage der Volumstromdaten in Echtzeit bei ElasticNet Regression .....	50
Abbildung 26. Vorhersage der Volumstromdaten in Echtzeit bei KNN.....	51
Abbildung 27. Vergleich von Modelltyp und Verlusten für KNN aus [17].....	52
Abbildung 28. Vorhersage des Volumstromes in Echtzeit bei KNNA (200 Nachbarn) .....	52
Abbildung 29. Vorhersage bei KNNA mit einem guten Trainingsmodell.....	53
Abbildung 30. Vorhersage des Volumstromes in Echtzeit bei Decision Tree (Baumtiefe ist 30) .....	54
Abbildung 31. Vorhersage des Volumstromes in Echtzeit bei Random Forest (Anzahl der Bäume ist 100) .....	55
Abbildung 32. Vorhersage des Volumstromes in Echtzeit bei SVM-Algorithmus .....	56
Abbildung 33. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen....	57
Abbildung 34. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen in Prozent.....	58
Abbildung 35. Vorhersage des Volumstromes unter gleichen Bedingungen bei ElasticNet-Regression .....	59
Abbildung 36. Vorhersage des Volumstromes unter gleichen Bedingungen bei Keras-Modell.....	60
Abbildung 37. Vorhersage des Volumstromes unter gleichen Bedingungen bei K-nächste-Nachbarn-Algorithmus.....	61

Abbildung 38. Vorhersage des Volumstromes unter gleichen Bedingungen bei Decision-Tree-Algorithmus.....	62
Abbildung 39. Vorhersage des Volumstromes unter gleichen Bedingungen bei Random-Forest-Algorithmus.....	63
Abbildung 40. Vorhersage des Volumstromes unter gleichen Bedingungen bei SVM-Algorithmus...	64
Abbildung 41. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen unter gleichen Bedingungen.....	65
Abbildung 42. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen unter gleichen Bedingungen in Prozent .....	66
Abbildung 43. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit 64, 128, 256 und 512 Units entsprechend .....	67
Abbildung 44. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit 64, 128, 256 und 512 Units entsprechend in Prozent.....	68
Abbildung 45. Eine Karikaturzeichnung eines biologischen Neurons (links) und seines mathematischen Modells (rechts) aus [27].....	68
Abbildung 46. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit relu-, sigmoid-, softmax- und tanh-Activation entsprechend.....	69
Abbildung 47. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit relu-, sigmoid-, softmax- und tanh-Activation in Prozent.....	69
Abbildung 48. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit RMSprop-, nadam, adam- und Ftrl-Optimizer entsprechend.....	70
Abbildung 49. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit RMSprop-, adadelta-, nadam- und Ftrl-Optimizer in Prozent.....	71
Abbildung 50. Schritte der Bedienung der Hauptfensterschnittstelle .....	79
Abbildung 51. Der Anfangszustand der GUI, wenn das Programm ausgeführt wird .....	80
Abbildung 52. Die dritte Schaltfläche auf der GUI wird gedrückt .....	81

## Tabellenverzeichnis

Tabelle 1. Die Geschwindigkeit der Algorithmen.....	46
Tabelle 2. Durchschnittliche Differenz zwischen vorhergesagten und realen Volumenströme bei Verwendung in Echtzeit.....	57
Tabelle 3. Durchschnittliche Differenz zwischen vorhergesagten und realen Volumenströme zum Vorhersagen von Daten unter gleichen Bedingungen.....	65
Tabelle 4. Praktische Vor- und Nachteile der betrachteten Algorithmen .....	74
Tabelle 5. Aus der Literatur entnommenen Vor- und Nachteile der betrachteten Algorithmen .....	77

# 1. Einleitung

## 1.1. Motivation

Soft-Sensoren, die auch als Softwaresensoren oder virtuelle Sensoren bezeichnet werden, können alternativ für Prozessgrößen eingesetzt werden, die aufgrund technologischer Einschränkungen, großer Messverzögerungen oder hoher Investitionskosten schwer zu messen sind [3]. Sie verwenden einen oder mehrere Rohmesswerte sowie ein empirisches oder theoretisches, dynamisches Modell [1], um aussagekräftige Informationen für Bioprozesse bereitzustellen.

Soft-Sensoren sind besonders nützlich bei der Datenfusion, bei der Messungen verschiedener Eigenschaften und Dynamiken kombiniert werden [3]. Es kann sowohl für die Fehlerdiagnose als auch für Steuerungsanwendungen verwendet werden. Soft-Sensoren können ungemessene, aber wichtige Daten aus anderen leicht zu messenden Variablen mit Computermodellen abschätzen [1]. Der Vorteil dieser Inferenzmessungen gegenüber der Labormessung besteht darin, dass die Schätzung in Echtzeit erfolgt. So sind Informationen kontinuierlich zur Verbesserung der Prozesssteuerung und -optimierung sowie zur Erhöhung der Prozesssicherheit verfügbar [4, S.1].

In vielen Anwendungen werden Softsensoren in der industriellen Prozesssteuerung eingesetzt, da sie die Produktqualität verbessern und die Sicherheit des Prozesses gewährleisten können [4, S. 2]. Es gibt viele Beispiele für den Einsatz von Techniken des maschinellen Lernens zur Modellierung von Soft-Sensoren.

Daher ist die *Hauptmotivation* in dieser Masterarbeit die Möglichkeit, virtuelle Sensoren zu verwenden, um echte physikalische Sensoren zu ersetzen. Dazu gehört auch der Auswahl und Prüfung verschiedener maschineller Lernalgorithmen zur Vorhersage des Volumenstromes.

## 1.2. Zielsetzung

Aber was soll erreicht werden? Die Ziele dieser Masterarbeit können nachstehend beschrieben werden:

1. Auswahl und Beschreibung von maschinellen *Lernalgorithmen* im theoretischen Teil
2. Auswahl der gewünschten *Eingabe-* und *Ausgabedaten* für das Lernen von Algorithmen
3. Entwicklung von Code und *Implementierung* von Algorithmen sowie deren *Test* und *Optimierung* für das beste Ergebnis.
4. Systematischer *Vergleich* verschiedener Algorithmen für die Eignung zur Berechnung des Volumenstroms auf Frequenzumrichterdaten
5. Entwicklung eines GUI-Interface für die Interaktion mit Algorithmen und für die Steuerung des Pumpversuchstandes

Das *Hauptziel* dieser Masterarbeit besteht also darin, am Ende einen Systemvergleich von Algorithmen zu haben, in dem die Vor- und Nachteile der jeweiligen Algorithmen angezeigt werden. Auch als wichtiges Ziel dieses Projekts ist es besonders nützlich, eine funktionierende Schnittstelle (oder ein GUI-Interface) zu entwickeln, die den gewünschten Algorithmus ausführen und Daten über den Volumenstrom vorhersagen kann.

### 1.3. Forschungsmethodik

Verschiedene maschinelle Lerntechniken werden verwendet, um einen virtuellen Sensor zur Vorhersage des Volumenstromes zu erstellen oder die Möglichkeit eines Softsensors zu untersuchen. Dazu können Regressionsalgorithmen wie lineare Regression, KNN-Algorithmus, Random Forest usw. gehören. Eine genauere Beschreibung aller verwendeten Algorithmen findet sich in Kapitel 4.

Es gibt auch einen Pumpversuchstand im NCT, der Wasser von einem Tank zum anderen pumpt. Damit kann die Funktionsfähigkeit von Algorithmen und die Genauigkeit der Flussvorhersage getestet werden. Eine ausführlichere Beschreibung des Pumpversuchstandes befindet sich in Kapitel 4.4.

## 2. Stand der Technik und die Grundlagen

### 2.1. Prozesstechnik

Softsensoren können effektiv verwendet werden, um die damit verbundenen Fehlererkennungsaufgaben zu automatisieren [1]. Das Soft-Sensor-Konzept wird mittlerweile in verschiedenen Anwendungsbereichen eingesetzt, z.B. in der biologischen Abwasserbehandlung, der Überwachung von Bioprozessen, biochemischen Systemen und vielen komplexen Prozessvorhersagen. Sie sind in der Kohlenwasserstoffindustrie und in der pharmazeutischen Industrie im Zusammenhang mit der Prozessanalysetechnik weit verbreitet.

### 2.2. Maschinelles Lernen

Softsensoren gehören zur Klasse der indirekten Messverfahren. Sie lassen sich in zwei Gruppen einteilen [1]:

- Softsensoren, die ein theoretisches Prozessmodell verwenden (auch als *White-Box*, *mechanistische* oder *Model-driven-Modelle* bezeichnet),
- und solche, die ein empirisches Modell zur Berechnung der Prozessvariablen verwenden (auch als *Black-Box*, *Data-driven-Modelle* bezeichnet).

Erste Prinzipmodelle basieren auf der Gleichgewichtsgleichung (Masse, Komponente, Energie) und enthalten ausführliche physikalisch-chemische Informationen über das System [2. S.2]. Leider sind die Prozesse in vielen praktischen Anwendungen oft zu komplex und unsicher, sodass solches Modell für eine detaillierte Modellentwicklung nicht ausreichend verstanden werden.

Und zweite Prinzipmodelle (*Data-driven-Modelle*) basieren auf Softsensoren, die gebaut werden, wenn kein detailliertes Wissen über den Prozess verfügbar ist. In diesem Fall werden Prozessdaten verwendet, um statistische Modelle zu erstellen, um die Beziehung zwischen Inputs und Outputs zu bestimmen [2. S.2].

## 2.3. Zukünftige Richtungen für Softsensorik

Viele Anwendungen verwenden Softsensoren mit verschiedenen Techniken des maschinellen Lernens, um einen Softsensor zu modellieren, z. B. Support-Vektor-Maschine, K-nächste Nachbarn, Decision Trees und weitere. In Kapitel 4 werden alle verwendeten Algorithmen detaillierter behandelt.

Derzeit werden Softsensoren in Unternehmen und ihren Programmen nicht sehr häufig eingesetzt. Sie müssen noch untersucht werden, damit wir verstehen, wie sie funktionieren und wie wir mit ihnen interagieren können.

Zukünftige Richtungen wären die experimentelle Implementierung der Softsensoren für einige Prozesse, um ihre Erfolgsrate mit physischen Sensoren zu vergleichen und damit diese Anwendung für maschinelles Lernen weiter zu validieren. Softsensor kann wirklich billiger sein als normale Hardwaresensoren [4, S.9]. Und diese virtuelle Sensoren sind die Modelle, die zur Berechnung schwer messbarer Prozessvariablen verwendet werden, um nützliche Informationen für die Überwachung, Steuerung und Optimierung industrieller Systeme bereitzustellen. Aus diesen Gründen ist dieses Thema aktuell und bedarf weiterer Forschung.

## 3. Stand der Wissenschaft

### 3.1. Maschinell lernender Wetter-Soft-Sensor zur Steuerung von Kläranlagen

Einige Artikel müssen berücksichtigt werden, um den aktuellen Stand des maschinellen Lernens mit Soft-Sensor zu demonstrieren. Zum Beispiel die Abwasserbehandlung ist seit Jahren eines der Hauptziele der Vereinten Nationen, um die Nachhaltigkeit der natürlichen Umwelt zu gewährleisten. Um eine effektive Wasseraufbereitung zu haben, wurden große Anstrengungen unternommen, um die Auswirkungen von Wasseraufbereitungsanlagen zu bewerten und zu reduzieren und einen autonomen Betrieb mit größtmöglicher Energieeinsparung zu garantieren. Dafür sind jedoch viele physikalische Sensoren teuer in der Anschaffung und Wartung. Darüber hinaus arbeiten nur wenige von ihnen hier online.

Ein Artikel entwarf den Softsensor für die Vorhersage der aktuellen Wetterbedingungen (Trocken, Regen oder Sturm). Um den Soft-Sensor zu konstruieren, wurden in dieser Studie die Datenvorverarbeitung, Modelldesign und Training der Daten abgeschlossen. Es wurden Algorithmen für maschinelles Lernen verwendet, die in Abschnitt 4 beschrieben wurden, wie Random Forest, Decision Trees, Support-Vektor-Maschine und weitere [3].

Diese Forschung wandte Techniken des maschinellen Lernens an, um die aktuellen Wetterbedingungen von drei weit verbreiteten Sensoren vorherzusagen: Q, COD (chemischer Sauerstoffbedarf) und N-Ammoniak (Ammoniakkonzentration). Und die Werte der drei Sensoren könnten leichter verwendet werden, um jede Wetterlage zu charakterisieren und zu unterscheiden:

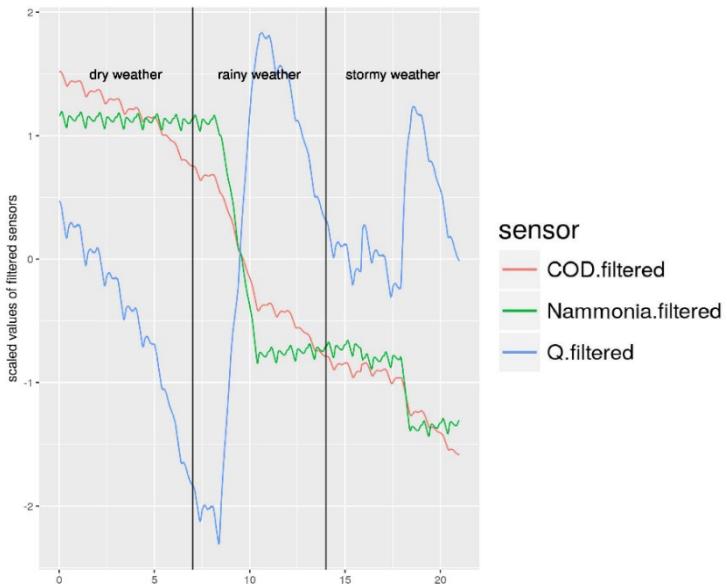


Abbildung 1. Beziehung zwischen Q-, COD- und N-Ammoniakmaßen nach Skalierung und starker Filterung aus [3, S.8]

Schließlich erzielt diese Forschung mit zwei Algorithmen für maschinelles Lernen (K-nächste Nachbarn und Random Forests) eine Genauigkeit des Wetter-Softsensors von ungefähr 85 % [3, S. 10]. Die Ergebnisse in diesem Artikel sind ermutigend und sollen daher die Leistung der genauereren Soft-Sensoren demonstrieren, um fortgeschrittene Steuerungsaufgaben in Kläranlagenprozessen zu bewältigen.

### 3.2. Vorhersage von Sorptionsdampf-Methan-Reformierungsprodukten aus auf maschinellem Lernen basierenden Soft-Sensor-Modellen

Ein weiteres praktisches Beispiel für die Anwendung von Softsensoren sollte berücksichtigt werden. Diese Forschung bezieht sich auf die Vorhersage von Sorptionsdampf-Methan-Reformierungsprodukten aus auf maschinellem Lernen basierenden Soft-Sensor-Modellen. Vorteile eines datengesteuerten Soft-Sensor-Modells gegenüber thermodynamischen Simulationen ist die Fähigkeit, Echtzeitinformationen abhängig von den tatsächlichen Prozessbedingungen zu erhalten. In dieser Studie wurden zwei Soft-Sensor-Modelle aufgezeigt und verwendet, um Variablen vorherzusagen und abzuschätzen, die sonst schwer zu messen wären. Sowohl künstliche neuronale Netze (benannt als KNN) als auch die Random-Forest-Modelle wurden als Soft-Sensor-Vorhersagemodelle entwickelt. Diese beiden Algorithmen für maschinelles Lernen werden in Kapitel 4 tiefer recherchiert.

Es wurde gezeigt, dass diese Soft-Sensor-Modelle gute Vorhersagen für Gaskonzentrationen in den *Reformer-* und *Regeneratorreaktoren* des Prozesses liefern, wobei Temperatur, Druck und Dampf-Kohlenstoff-Verhältnis als Eingabeprozessmerkmale verwendet werden. Beide Modelle waren sehr genau und alle über 98 % [4, S. 1]. Die Random-Forest-Modelle war jedoch in den Vorhersagen das Beste von allen.

Die Hauptbestandteile dieser Studie waren Input- und Output-Variablen. Und nachdem die fünf Input-Variablen und die gewünschten Output-Variablen ausgewählt wurden, wurde ein Algorithmus für das Training entwickelt. Von den beiden Arten von maschinellen

Lernkategorien - Klassifizierung und Regression - wurde die Regression verwendet, da die Daten kontinuierlich und numerisch waren.

Die Ergebnisse werden grafisch in Form von vorhergesagten und tatsächlichen Diagrammen dargestellt, was die Genauigkeit des Modells anleitet. Diese Diagramme zeigen an, wie genau die vorhergesagten Daten zu den tatsächlichen passen. Ein Modell, das eine gute Leistung erbringt, würde eine Streuung von Daten erzeugen, die nahe an der  $y = x$  Linie liegen:

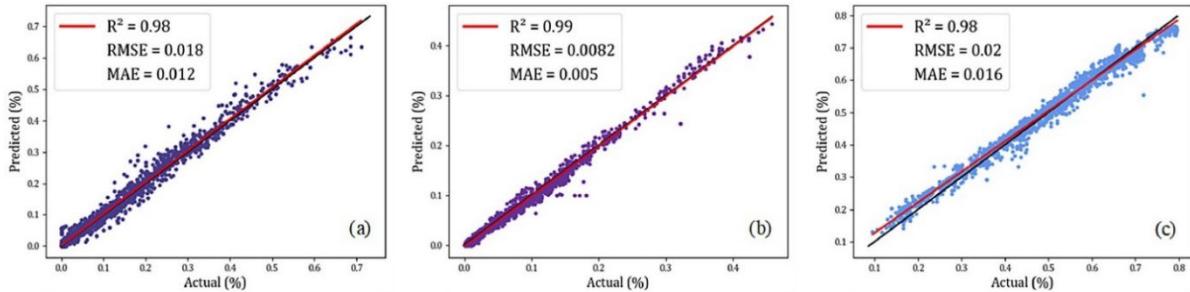


Abbildung 3. Tatsächliche und vorhergesagte Diagramme nach den künstlichen neuronalen Netzen (KNN) für: a) [H<sub>2</sub>O], b) [CO], c) [H<sub>2</sub>] (aus [4. S. 7])

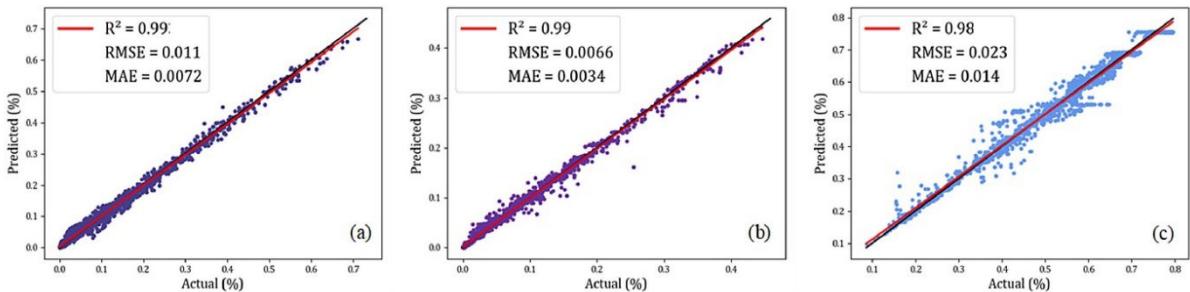


Abbildung 2. Tatsächliche und vorhergesagte Diagramme nach das Random-Forest-Algorithmus für: a) [H<sub>2</sub>O], b) [CO], c) [H<sub>2</sub>] (aus [4. S. 7])

Aus beiden Diagrammen und den unterstützenden Metriken geht hervor, dass der Random-Forest-Algorithmus bei der Vorhersage der Gaskonzentration in jedem Reaktor eine bessere Leistung erbrachte als das neuronale Netzwerk (KNN).

Auf diese Weise ist die Verwendung eines Softsensors im Vergleich zu einem Simulationsprogramm sehr vorteilhaft. Darüber hinaus kann ein Softsensor praktisch als Back-up-Sensor verwendet werden, wenn der Hardwaresensor defekt ist oder zur Wartung oder zum Austausch entfernt wird.

### 3.3. Auf maschinellem Lernen basierendes Soft-Sensor-Modell für die BSB-Schätzung

Und noch ein weiteres Beispiel für die Anwendung von Softsensoren sollte in Betracht gezogen werden. Die Überwachung der Wasserqualität ist neben der Luftqualität einer der kritischsten Aspekte der Umweltüberwachung. Der Zugang zu sauberem Trinkwasser ist für die Gesundheit und auch für eine gute Lebensqualität unerlässlich. Und um die sichere Trinkwasserversorgung zu gewährleisten, muss die Wasserqualität in Echtzeit überwacht werden.

Das ist ein komplexes System, da viele Qualitätsparameter überwacht werden müssen. Viele Wasserqualitätsparameter können aus verschiedenen Gründen nicht einfach online gemessen

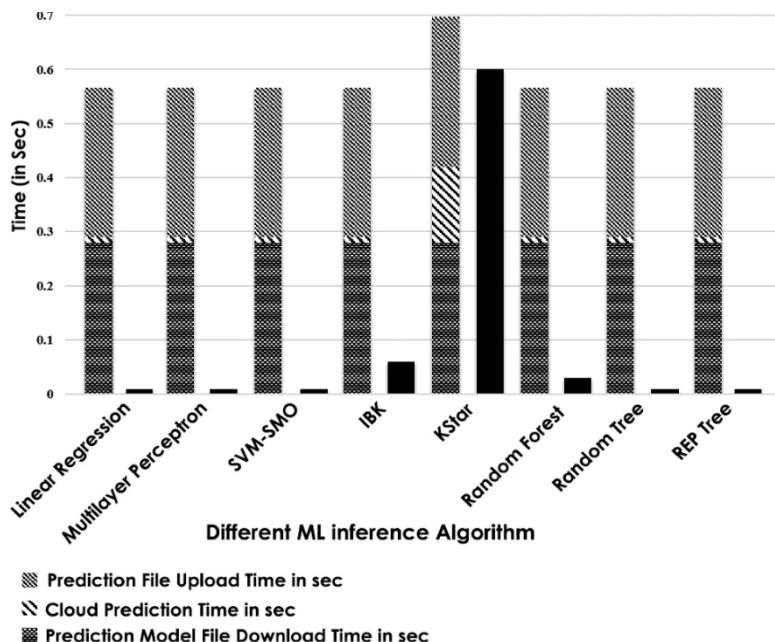
werden, wie z. B. teure Sensoren, niedrige Abtastrate, häufige Reinigung und Kalibrierung. Deshalb ist ein Soft-Sensor eine effiziente und komfortable Alternative zur Überwachung der Wasserqualität.

Dieser Artikel schlägt ein auf maschinellem Lernen basierendes Soft-Sensor-Modell zur Schätzung des biologischen Sauerstoffbedarfs (BSB) vor. BSB ist einer der lebenswichtigen Parameter zur Bestimmung der Wasserqualität. Und die Online-Messung von BSB durch Sensoren ist aufgrund wirtschaftlicher oder technischer Einschränkungen eine Herausforderung.

Nach der Vorverarbeitung sind die Echtzeitdaten bereit, Softsensoren mit dem maschinellen Lernalgorithmus zu modellieren. Hier wurden einige gängige Algorithmen zum Trainieren des Vorverarbeitungsdatensatzes wie lineare Regression, Random Forest, Random Tree und andere berücksichtigt. In Experiment 1 versuchen die Autoren, die Wasserqualität des Tanks zu überwachen. Und in Experiment 2 validieren sie das vorgeschlagene Modell mit den Echtzeitdaten zu dem Wasser des Flusses Ganga.

Um die Effizienz verschiedener ML-Algorithmen in Bezug auf Trainings- und Vorhersagezeit zu vergleichen, werden in Experiment 1 in einem Tank insgesamt 400 Stichproben für das Training und ein Datensatz für die Vorhersage entnommen [5, S. 968]. Für die Berechnung wurden zwei Techniken verwendet, die als Cloud Computing (online) und Edge Computing (offline) bezeichnet werden.

Abbildung 4 zeigt die vergleichende Analyse einschließlich der benötigten Vorhersagezeit in der Cloud (immer linke Spalte) und in dem Edge-Knoten (rechte Spalte):

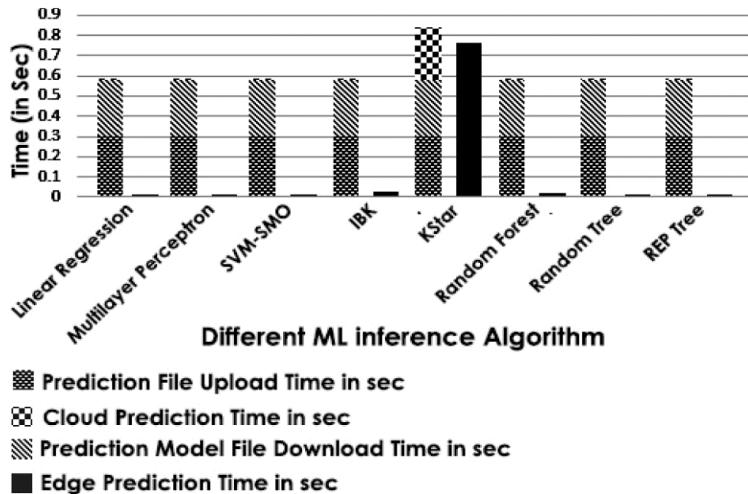


*Abbildung 4. Cloud vs Edge Vorhersagezeit von Datensätzen von Experiment 1 aus [5, S. 971]*

Aus dieser Abbildung ist ersichtlich, dass die Edge-Vorhersagezeit (offline) ungefähr 39-mal schneller ist als Cloud-Vorhersagezeit (online) [5, S. 971]. Wenn die Vorhersage in der Cloud

durchgeführt wird, sind die Zeit für das Hochladen und für das Herunterladen von Daten nach der Vorhersage erheblich. Und ein Vergleich zwischen Cloud-Level- und Edge-Level-Training wird ausgeführt, um die Werte in Echtzeit zu schätzen.

In Experiment 2 kann dasselbe mit Echtzeitdaten des Ganges (ein Fluss) getestet werden. Soft Sensor verwendete Datensätze mit vier Variablen als Eingabe, um die Ausgangsvariable BSB für den Ganga-Fluss vorherzusagen. Insgesamt werden 500 Proben für das Training und ein einziger Datensatz für die Vorhersage entnommen [5, S.973]. Abbildung 5 zeigt das Ergebnis des Experiments unten:



*Abbildung 5. Cloud vs Edge Vorhersagezeit von Datensätzen von Experiment 2 aus [5, S. 973]*

Aus dieser Abbildung ist ersichtlich, dass die Edge-Vorhersagezeit ungefähr 31-mal schneller ist als Cloud-Vorhersagezeit [5, S. 973]. Damit zeigen beide Experimente, dass Offline-Geräte besser und zuverlässiger sind als online.

In diesem Artikel wird ein BSB-Soft-Sensor-Modell vorgestellt, das datengesteuerte ML-Techniken verwendet, um den Wert von BSB in Echtzeit zu schätzen. Eine Vergleichsstudie zwischen verschiedenen ML-Algorithmen wurde durchgeführt, um eine geeignete Regressionstechnik für das vorgeschlagene System auszuwählen. Und es wurde festgestellt, dass die KNN und Random-Tree-Algorithmus gut passen, weil sie feinen guten Korrelationskoeffizienten und eine Edge-Antwortzeit (offline) haben.

Softsensoren haben einen praktischen Einfluss auf das Design und die Entwicklung von Wasserqualitätsüberwachungssystemen. Basierend auf dieser Studie können wir Entscheidungen treffen und notwendige Maßnahmen ergreifen sowie das Wasserqualitätsüberwachungssystem in Echtzeit mit Soft-Sensoren steuern. Deshalb sind weitere Forschungen mit Soft-Sensoren für andere Projekte in der Zukunft wichtig.

Viele physikalische Sensoren sind teuer in der Anschaffung. Und einige Attribute können nicht online mit ihrer Hilfe überwacht werden. Und in diesen Fällen können Softsensoren Online-Informationen bereitstellen, die von physischen Sensoren nicht direkt abgerufen werden können. Tatsächlich wird ein Softsensor als ein Modell definiert, das in der Lage ist, Variablen

vorherzusagen, die schwer zu messen sind. Und das Modell basiert auf früheren Daten, sogenannten Trainingsdaten, die von physikalischen Sensoren stammen.

In diesem Kapitel wurden verschiedene Forschungsprojekte behandelt und die praktische Anwendung der virtuellen Sensoren veranschaulicht. Wie man sehen kann, ist die Entwicklung von Softsensoren ein aktuelles und für viele Bereiche notwendiges Thema. Ihr Einsatz ist nicht neu und wurde in den letzten drei Jahrzehnten in der Prozessindustrie implementiert. Maschinelles Lernen wurde in vielen Technologien eingesetzt. Aber Softsensoren wurden jedoch nicht in vielen Bereichen benutzt. Deshalb hat diese Technologie ein gutes Potenzial.

## 4. Entwicklung der Lösung

### 4.1. Material und Methoden des maschinellen Lernens

Es gibt so etwas wie das „*No Free Lunch*“-Theorem. Sein Wesen liegt in der Tatsache, dass es keinen solchen Algorithmus existiert, der für jede Aufgabe die beste Wahl wäre.

Beispielsweise kann nicht gesagt werden, dass neuronale Netze immer genauer funktionieren als Entscheidungsbäume und umgekehrt. Die Effektivität der Datenvorhersage wird von vielen Faktoren beeinflusst wie von der Größe und Struktur des Datensatzes.

Aus diesem Grund sollten viele verschiedene Algorithmen getestet werden, indem die Wirksamkeit jedes einzelnen auf dem Testdatensatz überprüft und die beste Option ausgewählt wird. Wenn wir eine Analogie ziehen, dann verwendet eine Person beim Reinigen eines Hauses einen Staubsauger, einen Besen oder einen Mopp, aber keine Schaufel.

In diesem Abschnitt sollten die Methoden des maschinellen Lernens beschrieben werden, die zur Berechnung des Volumenstroms einer Kreiselpumpe verwendet werden, wie z. B. *K-nächste Nachbarn*, *Random Forest* und andere. Diese Algorithmen wurden gewählt, um Softsensoren zu erstellen, weil zu diesem Zweck am häufigsten überwachte Lernmethoden (*supervised learning* in english) benutzt werden [6, S. 20598]. In den nächsten Abschnitten werden diese Techniken an dem Pumpversuchstand gezeigt, aus dem der gesamte Datensatz gesammelt wurde

#### 4.1.1. Lineare Regression

Lineare Regression ist einer der bekanntesten und verständlichsten Algorithmen in Statistik und maschinellem Lernen. Eine lineare Regression kann als Verhältnisgleichung dargestellt werden, die eine gerade Linie beschreibt, die die Beziehung zwischen den Eingabeveriablen X und den Ausgabeveriablen Y am genauesten zeigt [6, S. 20600].

Zum Beispiel, wir haben eine Gleichung  $Y = B_0 + B_1 * X$ :

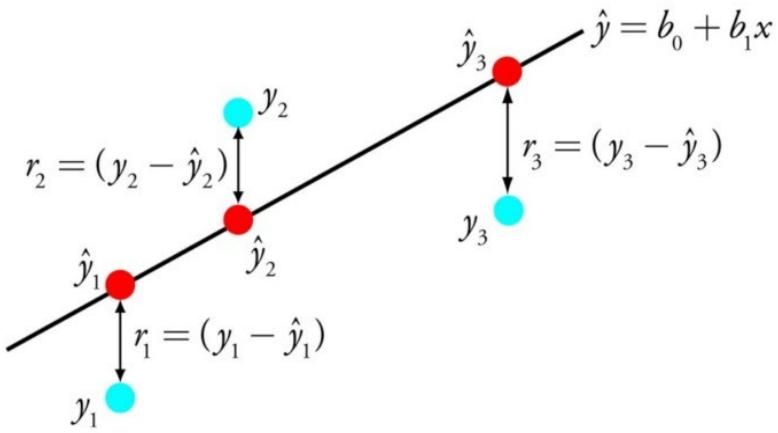


Abbildung 6. Beispiel für lineare Regression aus [11]

Wenn wir X kennen, müssen wir Y finden, und der Zweck der linearen Regression besteht darin, die Werte der Koeffizienten B0 und B1 zu bekommen. Verschiedene Techniken wie die lineare Algebra oder die Methode der kleinsten Quadrate werden verwendet, um das Regressionsmodell zu bewerten.

Lineare Regression ist ein schneller und einfacher überwachter Lernalgorithmus, der als erster Algorithmus in der Masterarbeit gut geeignet ist. Und die Regression ist nützlich, um die Reaktion auf neue Bedingungen vorherzusagen. Zum Beispiel aus Temperatur, Tageszeit und Anzahl der Bewohner lässt sich der Stromverbrauch eines Wohngebäudes abschätzen.

#### 4.1.2. Lasso Regression

Der *LASSO Regression* ist eine Variante der linearen Regression, die speziell für Daten angepasst ist, die eine starke Korrelation von Merkmalen untereinander aufweisen[12]. Es automatisiert Teile der Modellauswahl, wie z. B. die Variablenauswahl oder das Löschen von Parametern. LASSO verwendet Schrumpfung, ein Prozess, bei dem sich Datenwerte einem zentralen Punkt (z. B. einem Mittelwert) annähern.

Der Schrumpfungsprozess bietet Regressionsmodellen mehrere Vorteile:

- Genauere und stabilere Schätzungen der wahren Parameter.
- Reduzierung von Stichprobenfehlern und fehlender Stichprobe.

Das Lasso-Regressionsverfahren versucht, die Komplexität der Daten so zu reduzieren, dass sie durch einfache Regressionsverfahren verarbeitet werden können. In diesem Prozess hilft das Lasso automatisch, stark korrelierte und redundante Merkmale in der Low-Varianz-Methode zu eliminieren oder zu zerzerren.

Die Lasso-Regression verwendet die *L1-Regularisierung*, bei der die Fehler nach ihrem absoluten Wert gewichtet werden. Eine solche Regularisierung führt oft zu dünneren Modellen mit weniger Koeffizienten, da einige von ihnen null werden können und daher aus dem Modell ausgeschlossen werden. Dies ermöglicht es, das System zu interpretieren.

#### 4.1.3. Ridge Regression

Die Ridge-Regression ist der LASSO-Regression sehr ähnlich, da sie die Schrumpfung anwendet. Beide Algorithmen eignen sich gut für Datensätze mit vielen Merkmalen, die nicht unabhängig voneinander sind.

Der größte Unterschied zwischen den beiden besteht jedoch darin, dass die Ridge-Regression die *L<sub>2</sub>-Regularisierung* verwendet, was bedeutet, dass keiner der Koeffizienten Null wird, wie es bei der LASSO-Regression der Fall ist. Stattdessen nähern sich die Koeffizienten immer mehr Null, haben aber aufgrund der Natur des Algorithmus wenig Anreiz, sie zu erreichen[13].

Und die Ridge-Regression ist gut geeignet, wenn eine große Anzahl von Variablen priorisiert werden soll, von denen jede einen kleinen Effekt hat. Wenn mehr Daten im Modell berücksichtigt werden müssen, ist ein Lasso die beste Wahl.

#### 4.1.4. ElasticNet Regression

*ElasticNet* zielt darauf ab, das Beste aus *Ridge-Regression* und *Lasso-Regression* zu nehmen, indem es die Regularisierung von *L<sub>1</sub>* und *L<sub>2</sub>* kombiniert. Lasso und Ridge Regression sind zwei Regularisierungsmethoden. In beiden Fällen ist  $\lambda$  der Schlüsselfaktor, der das Gewicht bestimmt, das verschiedenen Teilen eines Objekts gegeben wird [14].

Die *ElasticNet-Regression* fügt dem  $\lambda$ -Parameter einen zusätzlichen  $\alpha$ -Parameter hinzu, der misst, wie «gemischt» die Regularisierungen *L<sub>1</sub>* und *L<sub>2</sub>* sein müssen. Wenn der  $\alpha$ -Parameter 0 ist, ist das Modell eine reine Ridge-Regression, und wenn es 1 ist, ist es eine reine Lasso-Regression.

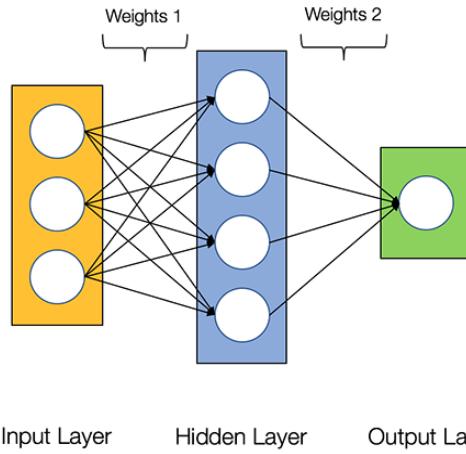
Der "Mischkoeffizient"  $\alpha$  bestimmt einfach, wie viel Regularisierung *L<sub>1</sub>* und *L<sub>2</sub>* in der Verlustfunktion berücksichtigt werden müssen. Alle drei populären Regressionsmodelle - *Ridge*, *Lasso* und *ElasticNet* — zielen darauf ab, die Größe ihrer Koeffizienten zu reduzieren, aber jedes wirkt auf seine eigene Weise.

#### 4.1.5. Künstliche neuronale Netze (KNN)

*Künstliche neuronale Netze* (*Artificial neural network* auf Englisch) sind eine Familie von Modellen, die von biologischen neuronalen Netzen inspiriert sind. Das wird als Systeme miteinander verbundener „Neuronen“ dargestellt, die Nachrichten untereinander austauschen. Die Verbindungen haben numerische Gewichte, die basierend auf Erfahrung abgestimmt werden können, wodurch sie an Eingaben (*Inputs*) anpassbar und lernfähig sind [6, S. 20601].

Die Arbeit mit neuronalen Netzen besteht darin, Operationen mit Vektoren durchzuführen, die als mehrdimensionale Arrays dargestellt werden. Die Hauptvektoren innerhalb eines Systems sind die Gewichtungs- (*Weights*) und Bias-Vektoren. Grob gesagt soll unser neuronales Netzwerk überprüfen, ob eine Eingabe anderen Eingaben ähnelt, die es bereits gesehen hat. Wenn die neuen Eingaben den zuvor gesehenen Eingaben ähnlich sind, dann werden auch die Ausgaben ähnlich sein. So erhält man das Ergebnis einer Vorhersage.

Das folgende Bild zeigt ein Beispiel für die Architektur eines neuronalen 2-Schicht-Netzes:



*Abbildung 7. Die Architektur eines neuronalen 2-Schicht-Netzwerks aus [7]*

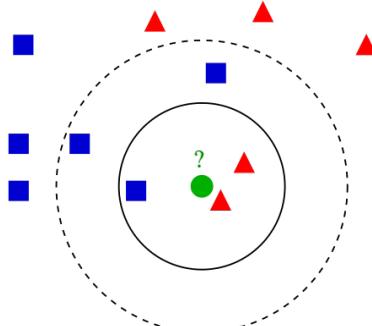
Wie andere maschinelle Lernverfahren werden neuronale Netze zur Lösung einer Vielzahl von Aufgaben in verschiedenen Anwendungsbereichen eingesetzt. Theoretisch sind sie in der Lage, sich jeder linearen/nichtlinearen Funktion anzunähern, indem sie aus beobachteten Daten lernen. In den letzten Jahrzehnten wurden verschiedene Arten von künstlichen neuronalen Netzen entwickelt. Die meisten von ihnen wurden für überwachtes Lernen gebildet, wie z. B. *Datenklassifizierung* und *Regressionsmodellierung* [6, S. 20601].

#### 4.1.6. K-nächste-Nachbarn-Algorithmus

*K-nächste-Nachbarn-Algorithmus (KNNA)* ist ein überwachter Algorithmus, der für die Klassifizierung und Regression verwendet wird [6, S. 20603]. Es ist eine einfache Methode, um einige Daten vorherzusagen, indem nur die ähnlichsten Datenpunkte (nach Nähe) betrachtet werden, die in der Trainingsphase gelernt wurden. Wenn dann ein neuer Datensatz klassifiziert wird, wird er dem häufigsten Datensatz unter seinen  $k$  nächsten Nachbarn zugewiesen (wobei  $k$  eine kleine positive ganze Zahl ist). Diese Technik hat viele Anwendungen mit Softsensoren.

Auf einer intuitiven Ebene ist die Essenz der Methode einfach: Schauen Sie sich die Nachbarn an, welche von ihnen dominieren, das sind Sie. Formal basiert die Methode auf der Kompaktheitshypothese: wird die Abstandsmetrik zwischen Beispielen erfolgreich eingeführt, dann liegen ähnliche Beispiele viel eher in derselben Klasse als in unterschiedlichen.

Ein Beispiel für die Klassifizierung von k-nächsten Nachbarn:



*Abbildung 8. Beispiel für den Algorithmus k-nächsten Nachbarn aus [8]*

- Wir haben eine Testprobe in Form eines *grünen Kreises*. Wir bezeichnen die blauen *Quadrat*e als *Klasse 1*, die roten *Dreiecke* als *Klasse 2*.
- Der grüne Kreis sollte als Klasse 1 oder Klasse 2 eingestuft werden. Wenn der von uns betrachtete Bereich ein *kleiner Kreis* ist, wird das Objekt als 2. *Klasse* klassifiziert, da innerhalb des gegebenen Kreises 2 Dreiecke und nur 1 Quadrat sind.
- Wenn wir einen *großen Kreis* (gestrichelt) betrachten, wird der Kreis als 1. *Klasse* klassifiziert, da innerhalb des Kreises 3 Quadrate im Gegensatz zu 2 Dreiecken sind.

Neben einer einfachen Erklärung ist es notwendig, die grundlegenden mathematischen Komponenten eines Algorithmus zu verstehen.

*Die euklidische Metrik* (Euclidean distance) ist eine Metrik, der Abstand zwischen zwei Punkten des euklidischen Raums nach dem Satz des Pythagoras berechnet. Einfach gesagt, es ist die kleinste mögliche Distanz zwischen A und B. Obwohl die euklidische Entfernung für kleine Messungen nützlich ist, funktioniert sie nicht für große Messungen und für kategorische Variablen. Der Nachteil der euklidischen Entfernung ist, dass sie die Ähnlichkeiten zwischen den Attributen ignoriert. Jeder wird als völlig anders angesehen als jeder andere:

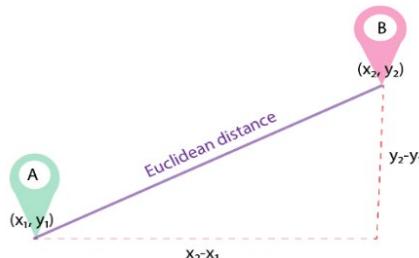


Abbildung 9. Berechnung der Euklidische Entfernung aus [8]

Ein weiterer wichtiger Bestandteil der Methode ist die **Normalisierung**. Verschiedene Attribute haben normalerweise einen anderen Bereich der angezeigten Werte in einer Stichprobe. Zum Beispiel wird das Merkmal A im Bereich von 0,01 bis 0,05 und das Merkmal B im Bereich von 500 bis 1000 dargestellt. Und hier können die Abstandswerte stark von Attributen mit großen Bereichen abhängen. Daher durchlaufen die Daten in den meisten Fällen eine *Normalisierung*. Bei der Clusteranalyse gibt es *zwei* grundlegende Möglichkeiten, Daten zu normalisieren: *MinMax-Normalisierung* und *Z-Normalisierung*.

Die *MinMax-Normierung* wird wie folgt durchgeführt:

$$x' = (x - \min[X]) / (\max[X] - \min[X])$$

In diesem Fall liegen alle Werte im Bereich von 0 bis 1; Diskrete Binärwerte werden als 0 und 1 definiert.

*Z-Normalisierung*:

$$x' = (x - M[X]) / \sigma[X]$$

wobei  $\sigma$  die Standardabweichung ist. In diesem Fall fallen die meisten Werte in den Bereich.

Und es sollte auch erwähnt werden, dass es keine spezifische Möglichkeit gibt, den besten Wert für  $k$  (das ist Anzahl der Nachbarn) zu bestimmen, daher sollten mehrere Werte ausprobieren werden, um den besten zu finden. Aber die am meisten bevorzugte Zahl für  $k$  ist 5 [8]:

- Ein niedriger  $k$ -Wert, z. B. 1 oder 2, kann dazu führen, dass das Modell nicht vertraut wird.
- Ein sehr hoher  $k$ -Wert erscheint auf den ersten Blick akzeptabel, es kann jedoch zu Leistungsschwierigkeiten des Modells kommen und das Risiko einer Umschulung steigt

*Vorteile des  $k$ -nahen Nachbarn-Algorithmus:*

- Der Algorithmus ist einfach und leicht zu implementieren.
- Es ist nicht notwendig, ein Modell zu erstellen, mehrere Parameter zu konfigurieren oder zusätzliche Annahmen zu treffen.
- Der Algorithmus ist universell. Es kann für beide Arten von Aufgaben verwendet werden: Klassifikation und Regression.

*Nachteile des  $k$ -nahen Nachbarn-Algorithmus:*

- Der Algorithmus arbeitet deutlich langsamer, wenn die Menge an Stichproben, Prädiktoren oder unabhängigen Variablen zunimmt.
- Aus dem obigen Argument folgt ein hoher Rechenaufwand zur Laufzeit.
- Es ist immer notwendig, den optimalen  $k$ -Wert zu bestimmen.

#### 4.1.7. Decision Tree

Der *Entscheidungsbaum* (Decision Tree) ist ein beliebtes und effektives Werkzeug für Data Mining und prädiktive Analysen. Dieser Algorithmus hilft bei der Lösung von Klassifizierungs- und Regressionsaufgaben und es gibt viele Beispiele für die Verwendung von Entscheidungsbäumen mit Softsensoren.

*Decision Tree* ist ein überwachter Klassifizierungs- und Regressionsalgorithmus, der einen Datensatz rekursiv in kleinere Sätze unterteilt, basierend auf einer Reihe von Tests, die in jedem Knoten des Baums definiert sind. Der Baum hat einen *Wurzelknoten*, der aus allen Anfangsdaten gebildet wird. Eine Reihe von *Zwischenknoten*, die sich aus den Unterteilungen ergeben. Und eine Reihe von *Endknoten*, die *Blätter* genannt werden. *Decision Trees* erfordern keine Annahmen bezüglich der Verteilungen der Eingabedaten. Es ist viel einfacher zu verstehen, wie ein *Decision Tree* anhand eines Beispiels funktioniert:

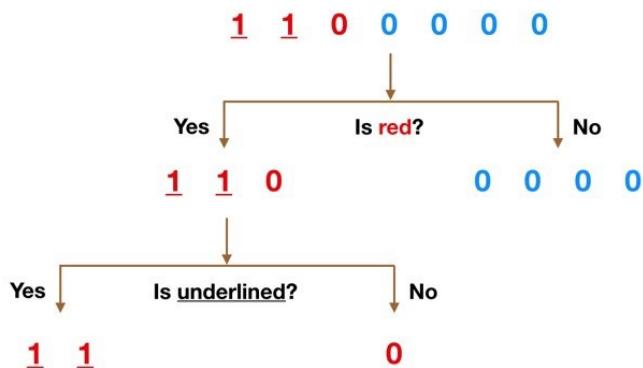


Abbildung 10. Beispiel für einen einfachen Entscheidungsbaum aus [9]

#### 4.1.8. Random Forest

*Random Forest* (RF) ist auch ein überwachter Algorithmus, der eine Reihe von Klassifizierungs- oder Regressionsbäumen auf andere Weise als ein herkömmlicher Entscheidungsbaumalgorithmus (*Decision Tree*) generiert [6, S. 20604]. Daher ändert der RF-Algorithmus nicht nur jeden Baum mit einer anderen Stichprobe der Daten, sondern auch die Art und Methode, wie Bäume konstruiert werden. Mit RF wird jeder Knoten des Baums in eine Teilmenge von Merkmalen unterteilt, die randomisch ausgewählt werden. Deswegen werden die Suchvorgänge des Wurzelknotens und die Aufteilung der Merkmalsknoten zufällig ausgeführt.

Im Gegensatz zu den meisten Techniken, die im maschinellen Lernen verwendet werden, erfordert der Random-Forest-Algorithmus keine komplexe Theorie. Der Random Forest-Algorithmus ist ein universeller Lernalgorithmus [6, S. 20604], dessen Essenz darin besteht, ein Ensemble von entscheidenden Bäumen (Random Forest) zu verwenden. Und der entscheidende Baum selbst bietet eine niedrige Klassifizierungsqualität, aber aufgrund seiner großen Anzahl verbessert sich das Ergebnis erheblich. Es ist auch einer der wenigen Algorithmen, der in den absoluten meisten Aufgaben verwendet werden kann.

Der Algorithmus besteht aus vier Stufen:

- Erstellung einer zufälligen Stichprobe aus einem angegebenen Dataset.
- Für jede Stichprobe wird ein Entscheidungsbaum erstellt und das Ergebnis der Vorhersage mit diesem Baum erhalten.
- Abstimmung für jede erhaltene Prognose.
- Auswahl der Vorhersagen mit den meisten Stimmen als Endergebnis

Jeder Baum in einer zufälligen Gesamtstruktur gibt eine Prognose der Klasse zurück. Und die Klasse mit den meisten Stimmen wird die Vorhersage des Systems sein. Jeder einzelne Entscheidungsbaum wird mithilfe von Metriken für die Metrikauswahl generiert, wie zum Beispiel das Informationsgewinnkriterium, das Gewinnverhältnis und der Gini-Index für jedes Merkmal. Und jeder solchen Baum wird auf der Grundlage einer unabhängigen Zufallsabtastung erstellt.

*Vorteile von Random Forest:*

- Der zufällige Wald gilt als eine hochpräzise und zuverlässige Methode, da viele Entscheidungsbäume am Vorhersageprozess beteiligt sind [9].
- Random Forest leidet nicht unter dem Umschulungsproblem. Der Hauptgrund dafür ist, dass eine zufällige Gesamtstruktur den Durchschnitt aller Vorhersagen verwendet, wodurch Offsets eliminiert werden.
- Eine zufällige Gesamtstruktur kann auch mit fehlenden Werten arbeiten.
- Random Forest berechnet auch die relative Bedeutung von Metriken, die bei der Auswahl der wichtigsten Merkmale für den Klassifikator hilft.

*Nachteile von Random Forest:*

- Der zufällige Wald ist langsam, da der Algorithmus viele Bäume verwendet: Jeder Baum erhält die gleichen Eingaben, auf deren Grundlage er seine Vorhersage zurückgeben muss.

Es gibt auch eine Abstimmung über die erhaltenen Prognosen. Die Implementierung eines zufälligen Baumalgorithmus erfordert eine beträchtliche Menge an Rechenressourcen.

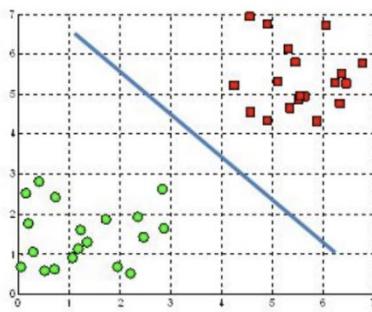
- Im Gegensatz zu einfacheren Algorithmen sind die Ergebnisse einer zufälligen Gesamtstruktur schwieriger zu interpretieren.

#### 4.1.9. Support-Vektor-Maschine

*Support Vector Machine* (SVM) ist ein binär überwachter Regressionsalgorithmus. Das SVM-Modell stellt die Daten in zwei Räume, die so breit wie möglich sind, dar und trennt die Klassen durch eine Hyperebene, die als Unterstützungsvektor bezeichnet wird [6, S. 20602]. Die Erfolgsquote von SVM ist besonders hoch, wenn der Trainingsdatensatz gut genug ist.

Das Ziel der SVM ist es, die Hyperebene zu definieren, die die Punkte in zwei Klassen unterteilt wie unten:

A hyperplane in  $\mathbb{R}^2$  is a line



A hyperplane in  $\mathbb{R}^3$  is a plane

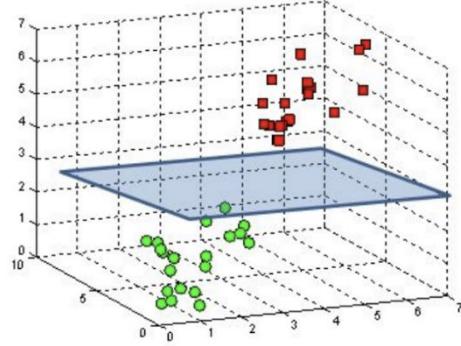


Abbildung 11. Optimale trennende Hyperebene aus [9]

Die Erklärung der SVM-Theorie kann sehr technisch und schwer zu verstehen sein. Es wird besser sein, die Arbeit des Algorithmus anhand praktischer Beispiele zu demonstrieren, die in den nachfolgenden Kapiteln zusammen mit dem Pumpversuchstand behandelt werden.

#### 4.1.10. Ergebnis der Recherche

Diese Algorithmen des maschinellen Lernens, die oben beschrieben und untersucht wurden, können verwendet werden, um verschiedene Daten vorherzusagen. In diesen Kapiteln wurde eher die theoretische Grundlage dieser Methoden analysiert und erklärt, nämlich wie sie funktionieren. Die detaillierte Tabelle 3 und 4, die sich in Kapitel 5.5 und 5.6 befindet, enthalten jedoch alle Vor- und Nachteile der betrachteten Algorithmen zusammen.

Es sollte auch darauf hingewiesen werden, dass die betrachteten maschinellen Lernalgorithmen universell sind und verwendet werden können, um jede Ausgabe vorherzusagen. Auf diese Weise können sie für alle Aufgaben und Anwendungen benutzt werden, nicht nur für die Arbeit am Pumpversuchstand.

Welcher der Algorithmen besser für den Pumpversuchstand geeignet ist, ist zum jetzigen Zeitpunkt schwer zu sagen, da sich alle in der praktischen Anwendung unterschiedlich funktionieren können. Ein detaillierter Vergleich findet sich in Kapitel 5.

## 4.2. Anforderungen/Bedarfsanalyse

Die Bedingungen und gewünschten Ziele der Entwicklung sollten im Abschnitt "Anforderungen" festgelegt werden. Auch verwendete Software und Frameworks sollten hier erwähnt werden.

Um mit dem Pumpversuchstand zu arbeiten, wird ein Programm von Siemens namens *WinCC Runtime Advanced* verwendet. Das ist eine leistungsstarke Software zur Lösung einfacher Visualisierungsaufgaben auf Produktionsmaschinenebene. Und es kann für den Aufbau von computerbasierten Bildgebungsstationen in allen Bereichen der industriellen Produktion sowie in Automatisierungssystemen verwendet werden. Eine ausführlichere Beschreibung der Verwendung dieser Anwendung findet sich in Kapitel 4.4.1.

Das Programm von Siemens namens *TIA Portal* wird verwendet, um eine Anwendung zur Visualisierung des Pumpversuchstandes zu entwickeln, die in der oben beschriebenen „WinCC Runtime Advanced“-Software angezeigt wird. Das TIA Portal integriert die zugrunde liegende Software in eine Schnittstelle: STEP 7, WinCC, SINAMICS Start Drive und andere.

Die zentrale Komponente des Systems ist *ibaPDA*, das Programm, das zum Sammeln, Anzeigen und Speichern von Messdaten verwendet wird (die Prozessdatenerfassung). Diese Daten können dann für maschinelles Lernen benutzt werden, wie in den nachfolgenden Kapiteln beschrieben.

Der Controller *SIMATIC S7-300* wird verwendet, um den Pumpversuchstandsbetrieb zu steuern. Und dieser Controller wird in vielen Anwendungen angewandt und hat sich gut bewährt. Die Steuerung bietet eine kompakte und modulare Konstruktion. Alle diese Programme sind für die Masterarbeit notwendig, da sie den Standbetrieb visualisieren und Daten von Sensoren sammeln können, die in maschinellen Lernprogrammen verwendet werden.

Die Programmiersprache Python wird zum Schreiben und Optimieren von Softsensorerstellungsprogrammen verwendet, da Python eine beliebte und einfache Computersprache ist und für einen Raspberry Pi-Controller benutzt werden kann.

Eines der wichtigsten Ziele der Arbeit ist es, einen Softsensor zu erstellen, der den physikalischen Drucksensor ersetzt, mit dem der Volumenstrom einer Kreiselpumpe berechnet wird. Um die Werte eines realen Sensors vorherzusagen, werden Daten von anderen Sensoren in einem maschinellen Lernprogramm verwendet. Und der Softsensor muss schnell arbeiten und die Daten echter Sensoren genau vorhersagen.

Die Optimierung und das Testen des Programms sind ebenfalls wichtige Teile der Masterarbeit. Bei der Ausführung einer Anwendung sollten keine Fehler auftreten, die den Benutzer daran hindern, das Endergebnis zu erhalten. Und die Programmoptimierung bedeutet, einen kurzen und präzisen Programmcode zu schreiben, der nicht viel Speicher beansprucht und nicht viele Ressourcen benötigt. Weitere Informationen befindet sich in den folgenden Kapiteln.

## 4.3. Auswahl der geeigneten Daten

### 4.3.1. Der Prozess der Datenerstellung

Maschinelle Lernalgorithmen lernen aus Daten. Es ist sehr wichtig, ihnen die richtigen Werten zur Verfügung zu stellen. Selbst wenn gute Eingaben und Ausgaben vorhanden sind, man soll auch sicherstellen, dass die Daten den passenden Maßstab und das korrekte Format haben. Die Genauigkeit der Ergebnisse des maschinellen Lernens hängt direkt von der Qualität der Daten für das Training des Modells ab.

Der Prozess der Datenerstellung für den maschinellen Lernalgorithmus kann in drei Phasen zusammengefasst werden:

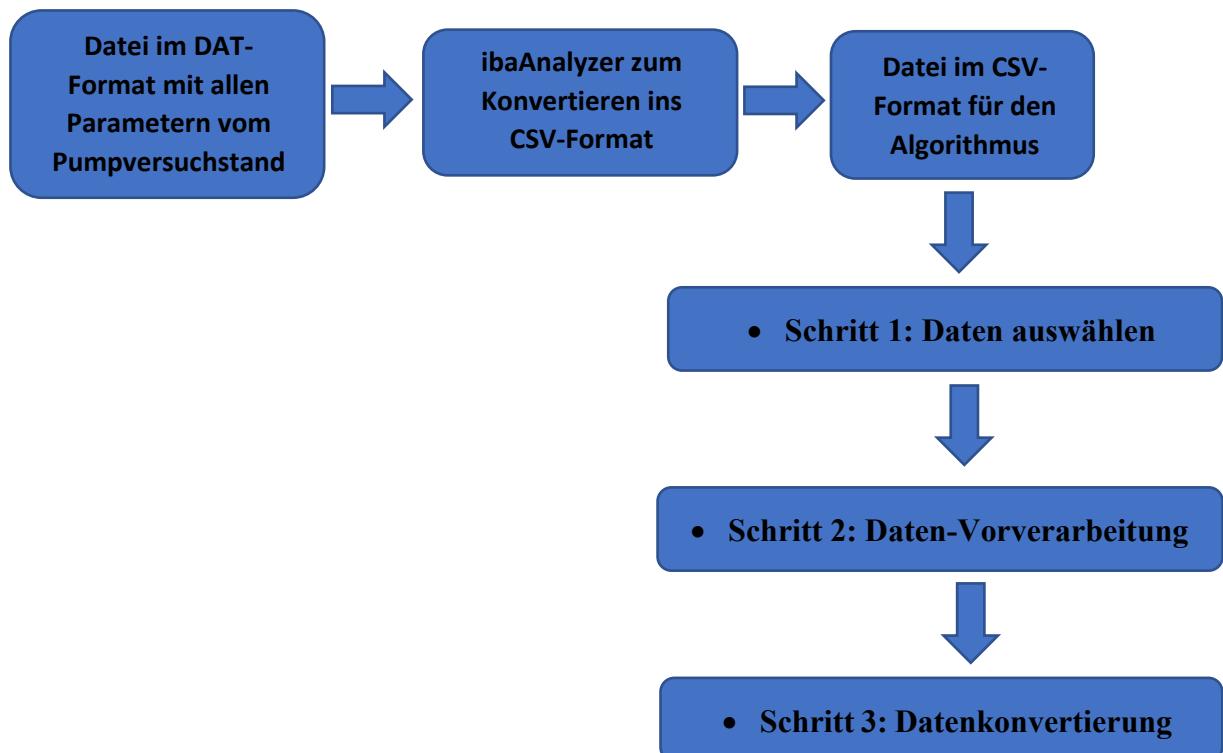


Abbildung 12. Prozess der Datenerstellung

**Schritt 1: Daten auswählen.** In diesem Schritt geht es darum, eine Teilmenge von Daten auszuwählen, die in der Arbeit verwendet werden. Es besteht immerhin der starke Wunsch, alle verfügbaren Werte einzubeziehen, damit das „Mehr ist besser“-Prinzip eingehalten wird[29]. Aber in Wirklichkeit hilft das nicht immer und macht das Programm noch schwieriger.

**Schritt 2: Daten-Vorverarbeitung.** Nach der Auswahl der Daten muss überlegt werden, wie sie verwendet werden sollen. Dieser Vorverarbeitungsschritt dient dazu, die ausgewählten Daten in eine weiterverarbeitbare Form zu bringen[29].

Die drei Hauptschritte der Vorverarbeitung sind *Formatierung*, *Bereinigung* und *Auswahl*:

- *Formatierung*: Die ausgewählten Daten sind möglicherweise nicht in dem für die Arbeit geeigneten Format. Zum Beispiel könnten sich die Eingaben in einer relationalen Datenbank befinden, und wir möchten, dass sie sich in einer flachen Datenbank befinden.

- *Bereinigung:* Unter Datenbereinigung versteht man das Entfernen oder Korrigieren fehlender Daten. Es kann vorkommen, dass die Eingaben unvollständig sind und nicht die Werte enthalten, die zur Lösung des Problems erforderlich sind. Darüber hinaus können einige Attribute vertrauliche Informationen beinhalten, die möglicherweise eine Anonymisierung oder vollständige Entfernung aus den Daten erfordern.
- *Auswahl:* Es können viel mehr ausgewählte Daten vorhanden sein, als für den Algorithmus erforderlich sind. Mehr Daten können zu viel längeren Algorithmuslaufzeiten und höheren Rechen- und Speicheranforderungen führen. Es kann erforderlich sein, eine kleinere repräsentative Stichprobe der ausgewählten Eingaben auszuwählen, was viel schneller zu untersuchen und zu analysieren ist, bevor der gesamte Datensatz berücksichtigt wird.

**Schritt 3: Datenkonvertierung.** Die drei allgemeinen Datentransformationen sind Skalierung, Attributzerlegung und Attributaggregation[29]:

*Skalierung:* Die vorverarbeiteten Daten können Attribute mit einer Mischung aus Gewichten für verschiedene Mengen enthalten, z. B. Grad, Kilogramm und Meter. Viele maschinelle Lernmethoden haben die gleiche Skala, z. B. 0 bis 1 für den kleinsten und größten Wert für eine bestimmte Funktion. Daher ist es notwendig, alle Daten zu skalieren.

*Attributzerlegung:* Es kann Funktionen geben, die ein komplexes Konzept darstellen, das für die maschinelle Lernmethode bei der Aufteilung in Komponenten nützlicher sein kann. Ein Beispiel ist ein Datum, das Tages- und Zeitkomponenten haben kann. Vielleicht hat nur die Stunde des Tages etwas mit dem gelösten Problem zu tun.

*Attributaggregation:* Möglicherweise gibt es Funktionen, die zu einer einzigen Funktion kombiniert werden können, die für den Betrieb des Algorithmus sinnvoller ist. Beispielsweise kann es Dateninstanzen für jede Benutzeranmeldung sein, die zu einem Zähler für die Anzahl der Anmeldungen gesammelt werden können, wodurch zusätzliche Instanzen verworfen werden können.

#### 4.3.2. Datenerstellung in TIA Portal-Programm

Um einen Sensor mit maschinellen Lernalgorithmen weiterzuentwickeln, muss man genau verstehen, welche Daten ausgewählt werden müssen. Insgesamt befinden sich im Pumpversuchstand 6 wichtige Drucksensoren: 2 neben der Pumpe, 2 neben dem Regelventil und 2 an der Messblende. Sie werden jedoch nicht in einem maschinellen Lernprogramm benutzt, weil das Ziel der Masterarbeit darin besteht, den Volumenstrom ohne Verwendung von Drucksensoren vorherzusagen. Zur Berechnung des Volumenstroms werden anderen Daten und Messwerte verwendet.

Mit dem Programm TIA Portal von Siemens können Sensordaten und andere Messwerte gewählt werden. So werden die Daten im TIA-Programm gefunden und ausgewählt:

The screenshot shows the TIA Portal Data Catalog titled "Daten\_Sensorik". It lists 22 data items with the following columns:

	Name	Datentyp	Offset	Startwert	Remanenz	Sichtbar i...	Einstellwert	K
1	Static							
2	Drucksensor_A1	"UDT_Daten_Analog..."	0.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
3	Drucksensor_A2	"UDT_Daten_Analog..."	20.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
4	Drucksensor_B1	"UDT_Daten_Analog..."	40.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
5	Drucksensor_B2	"UDT_Daten_Analog..."	60.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
6	Drucksensor_Q1	"UDT_Daten_Analog..."	80.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
7	Drucksensor_Q2	"UDT_Daten_Analog..."	100.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
8	Drucksensor_R1	"UDT_Daten_Analog..."	120.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
9	Drucksensor_R2	"UDT_Daten_Analog..."	140.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
10	Differenzdruck_untere...	"UDT_Daten_Analog..."	160.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
11	Differenzdruck_oberer...	"UDT_Daten_Analog..."	180.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
12	Temperatur	"UDT_Daten_Analog..."	200.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
13	Regelventil_Stellung	"UDT_Daten_Analog..."	220.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
14	Volumenstrom	Struct	240.0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
15	Füllstand_oberer_Tank	Real	248.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
16	Füllstand_unterer_Tank	Real	252.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
17	Motor_A_Elektrische_...	Real	256.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
18	Motor_A_Elektrische_...	Real	260.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
19	Motor_A_Wirkleistung...	Real	264.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
20	Motor_A_Wirkleistung	Real	268.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
21	Motor_A_Wirkleistung...	Real	272.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
22	Motor_A_Leistungsfa...	Real	276.0	0.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Abbildung 13. Datenerstellung in TIA Portal

Nachdem die Messdaten gesammelt wurden, kann ibaPDA-Programm sie in eine DAT-Datei konvertieren. Und ibaAnalyzer-Programm konvertiert es in eine CSV-Datei, die bereits für das maschinelle Lernprogramm verwendet wird. So funktioniert in dieser Masterarbeit der Prozess der Datenerfassung zum Erstellen eines Softsensors.

Aus allen Sensordaten und verschiedenen Messwerten, die für die Erstellung eines maschinellen Lernprogramms verfügbar sind, werden die folgenden Daten verwendet:

- *Volumenstrom*. Dieser Messwert muss auch zum Trainieren des Softsensors verwendet werden. Ohne ihn kann das Programm die Metrik in Zukunft nicht vorhersagen, da es nicht weiß, was am Ausgang sein sollte.
- Zustand des *Regelventil*. Dieser Wert beeinflusst den Volumenstrom und den Druck der Sensoren und muss daher berücksichtigt werden.
- *Elektrische Leistung* der Motoren. Aus dem gleichen Grund können diese Messwerte beim Lernen des Softsensors hilfreich sein.
- Die *Drehzahl* des Motors A. Dieser Indikator ist wichtig, da er hauptsächlich den Volumenstrom steuert.

Diese Werte und Indikatoren sollten ausreichen, um eine erste Version des Sensors zu erstellen. Später können jedoch möglicherweise mehr Daten verwendet werden.

## 4.4. Entwurf und Implementierung

### 4.4.1. Pumpversuchstand und Simulation der Arbeit

In dieser Masterarbeit wird ein Pumpversuchstand verwendet, um einen Softsensor zu erstellen, der den physikalischen Drucksensor ersetzt, mit dem der Volumenstrom einer Kreiselpumpe berechnet wird. Dieses System ist einfach: das Wasser wird mithilfe eines *Motors* von einem *Tank* in einen anderen *Tank* gepumpt. Ein wichtiger Teil hier ist das Regelventil, das den Wasserfluss durch mehr Öffnen und Schließen regulieren kann. Viele Parameter und Werte in diesem Stand dienen zur Berechnung des Volumenstroms und können angezeigt werden. In diesem System befinden sich auch die Wasserstandssensoren für zwei Tanks.

Der Pumpversuchstand wird unten auf dem Foto gezeigt:

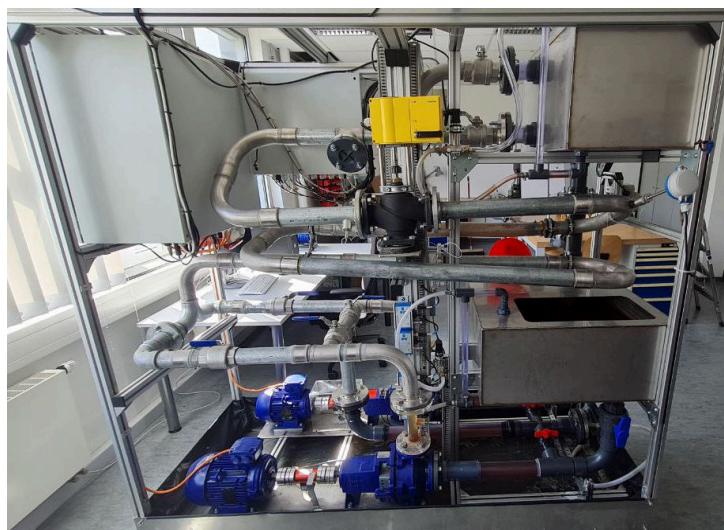


Abbildung 14. Pumpversuchstand im Betrieb

Darüber hinaus ist unten ein RI-Fließschema des Pumpversuchstands im Emulationsprogramm (SIMATIC WinCC Runtime Advanced) dargestellt. Hier werden die Daten aller Sensoren angezeigt, der aktuelle Wasserstand in den Tanks sowie der Öffnungsgrad des Ventils werden abgebildet:

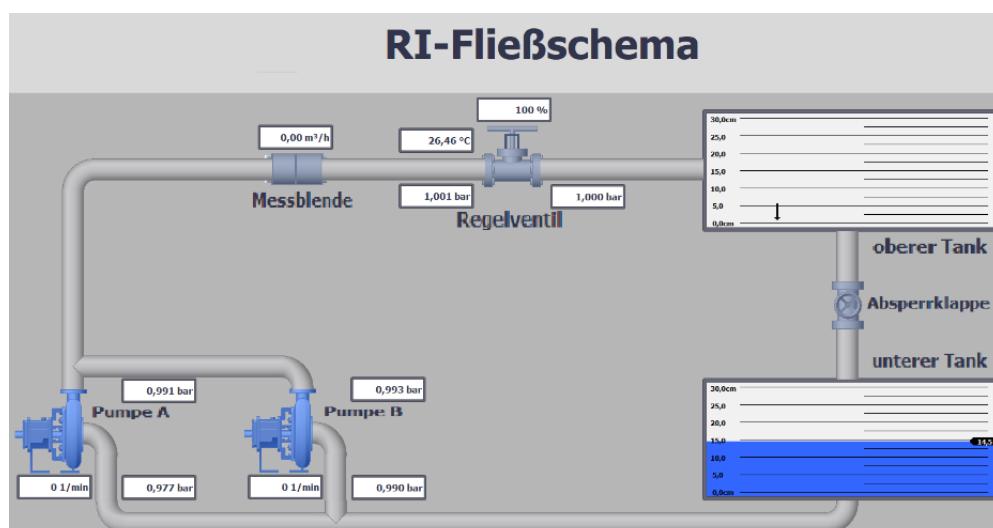


Abbildung 15. Fließschema des Pumpversuchstandes

Für die Regelung des Systems ist der Niederspannungs-Asynchronmotor verantwortlich, an den Signale mit einer bestimmten Modulation oder Verschiebung gesendet werden können. So kann eine Rechteck-Funktion ausgewählt und eine gewünschte Amplitude mit Dauer der Periode festgelegt werden. Auf diese Weise kann der Pumpversuchstand vielfältig gesteuert werden und hat verschiedene Funktionsarten mit unterschiedlichen Indikatoren:

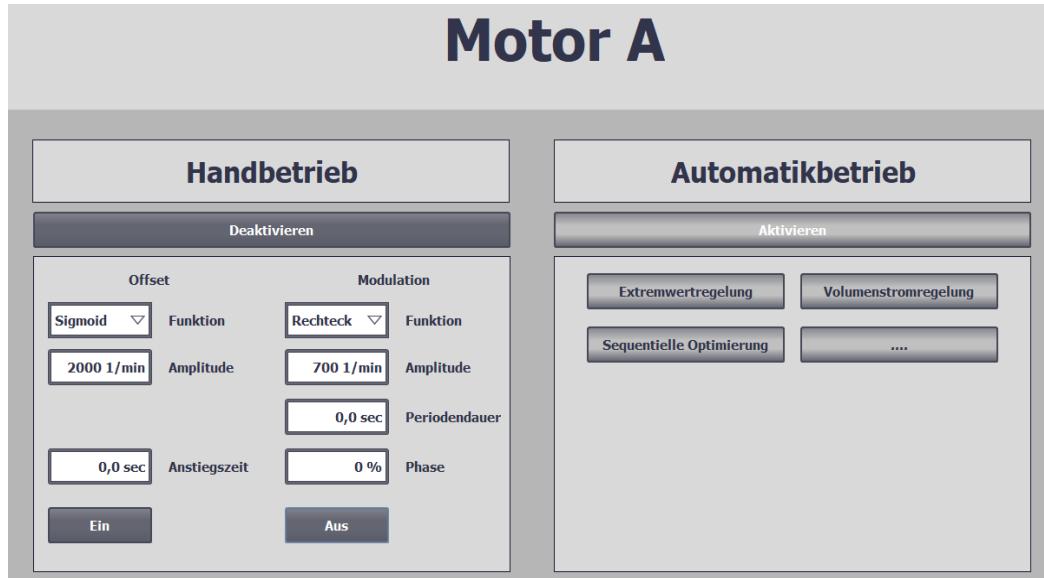


Abbildung 16. Handbetrieb des Motors A

Die folgende Abbildung unten zeigt, wie die Funktionsarten des Motors ausgewählt werden können. Hier sind drei grundlegende Modulationsregime verfügbar, mit denen der Pumpversuchstand funktionieren kann: Sinus, Sinus<sup>2</sup> und Rechteck:



Abbildung 17. Auswahl des Funktionstyps

Darüber hinaus kann der Offset auch ausgewählt werden, wenn es für die Arbeit benötigt wird. Um die anfängliche Entwicklung des Softsensors zu vereinfachen, wird der Funktionstyp *Rechteck* gewählt. Andere Arten von Funktionen können jedoch später betrachtet werden.

#### 4.4.2. Lineare Regression zur Vorhersage von Daten

Aber was ist überhaupt eine *Regression*? Regression sucht nach Beziehungen zwischen Variablen, wie es bereits im theoretischen Teil erwähnt wurde.

Zum Beispiel hängt der *Gehaltswert* von anderen unabhängigen Daten wie Berufserfahrung, Bildungsniveau, Rolle, Arbeitsstadt und so weiter ab. Die *Regression* löst das Problem der *Darstellung* von Analysedaten für jeden Mitarbeiter. Auf die gleiche Weise kann auch eine mathematische Beziehung zwischen anderen Daten hergestellt werden.

Regression betrachtet ein bestimmtes Phänomen und eine Reihe von Beobachtungen, die verschiedene Variablen haben. Unter der Annahme, dass eine Variable von anderen abhängt, versuchen wir, eine Beziehung zwischen ihnen aufzubauen. Oder mit anderen Worten, die *lineare Regression* findet eine Funktion, die die Abhängigkeit einiger *lineare* Daten von anderen zeigt [6, S. 20592]. Es kann verwendet werden, um die Antwort auf neue Bedingungen vorherzusagen. Der Stromverbrauch eines Wohnhauses kann aus den Daten der Temperatur, der Tageszeit und der Anzahl der Bewohner errechnet werden.

Unten wird der lineare Regressionsalgorithmus ausführlich beschrieben. Aber in den folgenden Kapiteln für andere Algorithmen wird nur ein Teil des Codes erklärt, der sich ändert. Der *linearen Regression* in Python kann selbst in fünf Schritten dargestellt werden:

1. Importieren der erforderlichen Pakete und Klassen
2. Bereitstellung von Daten für Arbeit und Konvertierung
3. Erstellung eines Regressionsmodells und die Anpassung der vorhandenen Daten
4. Anwenden eines Modells auf Prognosen
5. Überprüfung der Prognoseergebnisse

Dies sind allgemeine Schritte für die meisten Regressionsansätze.

##### **Schritt 1: Importieren der erforderlichen Pakete und Klassen**

Im ersten Schritt werden das *Pandas-Paket*, die *Linear-Regression-Klasse* und den *mean\_squared\_error* aus der *sklearn-Bibliothek* importiert:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Das *sklearn-Paket* ist eine Bibliothek, die im maschinellen Lernen weit verbreitet ist. Es stellt Werte für Vorverarbeitungsdaten bereit, reduziert die Dimension, implementiert Regression, Klassifikation, Clustering usw. Und der *mittlere quadratische Fehler* (Mean Squared Error auf Englisch) misst den Durchschnitt der Fehlerquadrate oder die durchschnittliche quadratische Differenz zwischen den geschätzten Werten und dem wahren Wert.

Zu Beginn jeder Analyse müssen Daten bereinigt, organisiert und aufgeräumt werden. Und die *Pandas-Bibliothek* kann uns helfen, Zeilen und Spalten zu löschen und die richtige Auswahl an Daten zu erhalten. Alle diese Bibliotheken befinden sich in Open-source und jeder kann sie nutzen.

## Schritt 2: Bereitstellung von Daten für Arbeit und Konvertierung

Bevor der Code für diesen Teil erklärt wird, sollte beachtet werden, dass Dateien für maschinelle Lernprogramme ohne „NaN“-Formatwerte verwendet werden sollten. Andernfalls kann der Algorithmus möglicherweise nicht verstehen, was dieser Wert ist und wie er gelesen werden sollte. Auch die Daten von 0 (Null) sollten vermieden werden, da maschinelle Lernalgorithmen besser mit Zahlen ungleich Null funktionieren.

Die Pandas-Bibliothek wird zur Bereitstellung von Daten verwendet. Der Code kann unten dargestellt werden:

```
16     data_set = pd.read_csv("Data for Learning New.txt",
17                         delimiter = ';', low_memory=False, encoding='latin-1')
18
19     data_set.columns = ["RotationMotorA", 'Volumenstrom',
20                         'Regelventil', 'Motor_A_Elektrische Leistung']
21
22     data_set.to_csv('Data for Learning.csv',
23                      index = None)
24
25     columns = ["RotationMotorA", 'Volumenstrom',
26                'Regelventil', 'Motor_A_Elektrische Leistung']
27
28     data_set = data_set.loc[:, columns]
29
30     fullstand2 = ['Volumenstrom']
31     X = data_set.drop(fullstand2, axis=1)
32     y = data_set[fullstand2]
```

In Zeile 16 wird der Befehl "read\_csv" verwendet, mit der eine *TXT-Datei* geöffnet wird. Der Parameter *delimiter* bedeutet, dass die Spalten mit dem Trennzeichen ";" geteilt werden. Aber im Allgemeinen können andere Trennzeichen wie Punkt, Komma usw. benutzt werden. Die Codierung wird 'latin-1' verwendet, um sicherzustellen, dass es keine Probleme beim Anzeigen von Buchstaben in der Konsole gibt.

In Zeile 19 wird für dieses Datumsset der Befehl "columns" verwendet, um jeder Spalte einen Namen zu geben. Dies ist erforderlich, damit der Benutzer anhand des Namens auf die Spalte zugreifen kann.

Danach wird in Zeile 22 das Dateiformat von *TXT* in *CSV* konvertiert (die Funktion *to\_csv*), da es für die *Pandas-Bibliothek* einfacher ist, direkt mit dem *CSV-Format* zu arbeiten.

Die Zeile 25 setzt die "columns"-Variable, die alle verfügbaren Daten enthält. Schließlich wird diese Variable weiter in Zeile 28 benutzt, um alle Daten anzugeben, die vom maschinellen Lernalgorithmus verwendet werden.

Darüber hinaus wird in Zeile 28 für unseren Datensatz der Befehl "loc" verwendet. Wenn wir die *loc-Methode* auf einige Daten anwenden, geben wir an, welche Zeilen und welche Spalten wir nutzen wollen. Auf diese Weise, wenn geschrieben wird "data\_set.loc[:, columns]", dann bedeutet das, dass alle Zeilen und alle Spalten, die in der Variablen "columns" sind, verwendet werden.

Die Zeile 30 ist wichtig, da hier der Name des Parameters angegeben wird, der vorhergesagt werden soll. Es kann auch als Ausgabe (*Output*) bezeichnet werden. In unserem Beispiel wird der Volumenstromwert vorhergesagt, daher wurde *Volumenstrom* in Klammern geschrieben. Aber der Benutzer kann, wenn er das Programm verwendet, jede andere Variable angeben, die er vorhersagen möchte. Der Algorithmus wird auch diese neue Ausgabe abschätzen.

Die Zeilen 31 und 32 geben die Variablen  $X$  und  $Y$  an, die jeweils Eingabe- und Ausgabedaten sind und für den maschinellen Lernalgorithmus verwendet werden. Beim Festlegen der Eingaben  $X$  wird der Befehl "drop" benutzt. Die Pandas-Drop-Funktion ist eine hilfreiche Funktion, um Spalten und Zeilen zu löschen. In Klammern wird die Spalte angegeben, die gelöscht werden soll (in unserem Beispiel ist das der *Volumenstrom*). Der Parameter "axis" bedeutet: Standardwert 0 für Zeilen und 1 für Spalten.

Auf diese Weise können die Daten für den maschinellen Lernalgorithmus bereitgestellt werden. Die Erklärung kann schwer zu verstehen sein, aber wenn man den Code Zeile für Zeile betrachtet, sieht es nicht so kompliziert aus.

### *Schritt 3: Erstellung eines Regressionsmodells und die Anpassung der Daten*

Nachdem die Daten in das Programm hochgeladen wurden, müssen sie in einer für den maschinellen Lernalgorithmus geeigneten Form dargestellt werden. Der Code dieses Teils kann unten gezeigt werden:

```
34      # Split our data
35      X_train, X_test, y_train, y_test = train_test_split(X, y,
36                                test_size=0.2, random_state=0)
37      # Run standardization on X variables
38      X_train_scaled = scale(X_train)
39      X_test_scaled = scale(X_test)
40
41      # Train model on training set
42      lin_reg = LinearRegression().fit(X_train_scaled,
43                                      y_train.values.ravel())
```

Um ein Regressionsmodell zu erstellen, müssen die Eingaben in Trainings- und Testdaten aufgeteilt werden. Dadurch werden wir zuerst das Modell auf dem Trainingsset trainieren (" $X_{train}$ " und " $y_{train}$ ") und dann auf dem Testset testen (" $X_{test}$ " und " $y_{test}$ "). So funktioniert die Regression: zuerst wird der Algorithmus auf Trainingsdaten trainiert, dann werden die Ergebnisse auf Testdaten überprüft.

In Zeile 35 wird die wichtige Funktion "train\_test\_split" verwendet, die die Aufteilung der Eingaben in Trainings- und Testdaten erleichtert. Natürlich können die Daten manuell geteilt werden, aber das Verfahren kann leicht auch so durchgeführt werden.

Im folgenden Code teilt *train\_test\_split* die Daten auf und gibt eine Liste zurück, die vier NumPy Arrays enthält. Der Parameter *train\_size* = 0.2 fügt 80% der Daten in ein *Trainingsset* und die restlichen 20% in ein *Testset* ein.

Der letzte Parameter `random_state` ist ein Zufallszahlenparameter, mit dem wir bei jeder Ausführung des Codes die gleiche exakte Aufteilung reproduzieren können (wenn dieser Wert 0 ist, sind die geteilten Datenwerte nicht zufällig, sondern exakt gleich). Somit kann die Randomisierung während der Datenaufteilung gesteuert werden.

Bevor der Algorithmus für maschinelles Lernen ausgeführt wird, ist es wichtig, jede Eingabe so zu standardisieren, dass es im gleichen Maßstab ist, bevor die Hauptkomponenten generiert werden. Wenn keine Standardisierung angewendet wird, werden die Merkmale mit größeren Reichweiten (und damit höherer Varianz) dominieren und eine große Rolle bei den Daten spielen. Die Zeilen 38 und 39 benutzt die `scale-Funktion` aus der `sklearn-Bibliothek`, um alle Eingabewerte zu standardisieren. Eine andere Version des Codes verwendet die Datennormalisierung, die nach den Formeln aus Kapitel 3.1.6 berechnet wird.

In Zeile 42 wird eine Variable aus der linearen Regressionsklasse erstellt, die das Regressionsmodell darstellt. Mit der `fit-Funktion` wird die optimale Beziehung zwischen Trainingsdaten berechnet, wobei die vorhandenen Eingabe- und Ausgabedaten als Argumente verwendet werden (`X_train` und `y_train`). Oder mit anderen Worten, `fit-Funktion` kombiniert das Regressionsmodell.

#### **Schritt 4:** Anwenden eines Modells auf Prognosen

Jetzt kann das lineare Regressionsmodell für die Vorhersagen von Testdaten verwendet werden. Dieser Teil enthält nur eine Codezeile, die unten dargestellt werden kann:

```
44      # Predicting multiple observations  
45      predictions = lin_reg.predict(X_train_scaled)
```

Das Modell verwendet die Informationen, die während des Modelltrainingsprozesses gelernt wurden (war in Schritt 3). Die vorhergesagte Antwort kann mit der "`predict-Funktion`" aus der `sklearn-Bibliothek` erhalten werden, wobei die Eingabetestdaten in Klammern angegeben werden.

#### **Schritt 5:** Überprüfung der Prognoseergebnisse

Nach der Vorhersage der Daten muss man sicherstellen, dass die Ergebnisse zufriedenstellend sind oder wie genau die Prognosen sind. Der Code dieses Teils nimmt auch nicht viele Zeilen ein und befindet sich unten:

```
47      score = lin_reg.score(X_test_scaled, y_test)  
48      print("\n How accurate it works: (1 - the best)")  
49      print(score)
```

Die `score-Funktion` aus der gleichen `sklearn-Bibliothek` kann verwendet werden, um die Genauigkeit der Vorhersagen eines linearen Regressionsmodells zu bestimmen. Hier ist die Genauigkeit definiert als:  $\frac{\text{korrekte Vorhersagen}}{\text{Gesamtzahl der Datenwerte}}$ . Je höher dieser Wert, desto besser ist es (aber der Höchstwert ist 1). Normalerweise beträgt dieser Parameter 0,97 oder mehr. Hier ist beispielsweise die Genauigkeit der Datenvorhersage bei Verwendung eines linearen Regressionsmodells mit einer einfachen Vorhersage mit Rechteckmodus:

```
How accurate it works: (1 - the best)  
0.9800684067070101
```

Abbildung 18. Beispiel für die Genauigkeit der Datenvorhersage (Volumenstrom)

Dies ist ein Beispiel dafür, wie die Vorhersagegenauigkeit von Daten getestet werden kann und ob der Algorithmus erfolgreich funktioniert. Im nächsten Kapitel (im Kapitel 4) wird eine andere und übersichtlichere Methode zur Bestimmung der Genauigkeit der Datenvorhersage erläutert. Und danach wird erwähnt, wie der Code optimiert und die Genauigkeit der Vorhersage verbessert werden kann.

#### 4.4.3. Lasso Regression zur Vorhersage von Daten

Die Beschreibung des Prinzips der *Lasso-Regression* wurde im theoretischen Teil (Kapitel 3.1.2) gegeben, sodass es keinen Sinn macht, noch einmal im Detail über Lasso-Regression zu schreiben. Es ist besser, sich sofort auf den Python-Code für diesen Algorithmus zu beziehen. Das Programm hier unterscheidet sich fast nicht von dem Programm für die lineare Regression. Allerdings muss das *LassoCV-Modul*, das für die Lasso-Regression zuständig ist, am Anfang aus der *sklearn-Bibliothek* importiert werden:

```
<<from sklearn.linear_model import LassoCV>>
```

Der Teil des Codes, der anders sein wird, sieht folgendermaßen aus:

```
41      # Train model on training set  
42      lasso_reg = LassoCV().fit(X_train_scaled,  
43                                y_train.values.ravel())  
44  
45      # Predicting multiple observations  
46      predictions = lasso_reg.predict(X_train_scaled)  
47  
48      score = lasso_reg.score(X_test_scaled, y_test)  
49      print("\nHow accurate it works: (1 - the best)")  
49      print(score)
```

Der einzige Unterschied besteht darin, dass das *LassoCV-Modul* beim Lernen und Vorhersagen verwendet wird, das für die Lasso-Regression verantwortlich ist. Der Rest des Codes blieb unverändert. Die Genauigkeit mit einer einfachen Vorhersage liegt ebenfalls bei etwa 0.98, genau wie bei der linearen Regression.

#### 4.4.4. Ridge Regression zur Vorhersage von Daten

*Ridge-Regression* braucht auch keine Einführung, da es bereits im theoretischen Teil erwähnt wurde. Der Code für diesen Algorithmus unterscheidet sich ebenfalls nicht sehr, jedoch muss das *RidgeCV-Modul* aus der importiert werden, um mit der Ridge-Regression arbeiten zu können:

```
«from sklearn.linear_model import RidgeCV»
```

Der geänderte Teil des Codes kann unten dargestellt werden:

```
41  # Train model on training set
42  ridge_reg = RidgeCV().fit(X_train_scaled,
43                           y_train.values.ravel())
44
45  # Predicting multiple observations
46  predictions = ridge_reg.predict(X_train_scaled)
47
48  score = ridge_reg.score(X_test_scaled, y_test)
49  print("\nHow accurate it works: (1 - the best)")
50  print(score)
```

Wie man sehen kann, ist der einzige Unterschied im Code die Verwendung des *RidgeCV-Moduls*. Dadurch kann der Benutzer die Ridge Regression nicht im Voraus mathematisch programmieren, sondern einfach die fertige Bibliothek verwenden. Die Genauigkeit der Volumenstromvorhersage bei der Ridge-Regression mit einer einfachen Vorhersage beträgt ebenfalls 0.98, was ein hervorragendes Ergebnis ist.

#### 4.4.5. ElasticNet Regression zur Vorhersage von Daten

*ElasticNet-Regression* hat das Beste von Lasso- und Ridge-Regression. Die Unterschiede im Programm sind wie folgt:

```
41  # Train model on training set
42  elastic_net = ElasticNetCV().fit(X_train_scaled,
43                                   y_train.values.ravel())
44
45  # Predicting multiple observations
46  predictions = elastic_net.predict(X_train_scaled)
47
48  score = elastic_net.score(X_test_scaled, y_test)
49  print("\nHow accurate it works: (1 - the best)")
50  print(score)
```

Um diesen Algorithmus verwenden zu können, muss das *ElasticNetCV-Modul* aus der *sklearn-Bibliothek* importiert werden:

```
«from sklearn.linear_model import ElasticNetCV»
```

Der grundlegende Unterschied besteht darin, dass das *ElasticNetCV-Modul* beim Lernen und Vorhersagen von Daten verwendet wird. Der gesamte Rest des Programms ähnelt den vorherigen Regressionsprogrammen. Und die Genauigkeit des Volumenstroms mit einer einfachen Vorhersage ist auch 0.98.

Daher wurden in diesen Kapiteln die grundlegenden Regressionsarten in Python beschrieben, die leicht benutzt werden können, um unterschiedliche Arten von Daten vorherzusagen. Und

dafür ist es nicht notwendig, den mathematischen Teil dieser Algorithmen zu kennen, der Benutzer kann die fertige und open-source sklearn-Bibliothek mit verschiedenen Regressionsmodulen verwenden.

#### 4.4.6. Künstliches neuronales Netz (KNN) zur Vorhersage von Daten

Um künstliche neuronale Netze zu erstellen, müssen wir die *Keras-Bibliothek* verwenden:

```
«from tensorflow import keras»
```

Und um mit einer anderen Keras-Bibliothek für maschinelles Lernen zu arbeiten, müssen verschiedene Datenschichten erstellt werden. Daher wird dieser Teil des Codes etwas anders aussehen. Die Anfangsphase beim Exportieren von Daten bleibt jedoch gleich. Nur der Teil, der mit dem Lernen des Modells und der Datenvorhersage verbunden ist, wird sich ändern:

```
32     x_train = X_train.astype("float32")
33     x_test = X_test.astype("float32")
34
35     inputs = keras.Input(shape= (1, 3))
36     x = layers.Dense(128, activation="relu", name="dense_1")(inputs)
37     x = layers.Dense(128, activation="relu", name="dense_2")(x)
38     outputs = layers.Dense(1)(x)
39
40     model = keras.Model(inputs=inputs, outputs=outputs)
41
42     model.compile(loss="mean_squared_error", optimizer='adam',
43                     metrics=['accuracy'])
44
45     model.fit(X_train, y_train, epochs=10, batch_size=1)
```

Die Zeilen 32 und 33 verwenden die *astype-Funktion*, um die Eingabe im "float32"-Format darzustellen. Dies ist erforderlich, damit die Schichten in Keras korrekt funktionieren. In Zeile 35 wird ein Eingabedaten-Layer erstellt, wobei Parameter (*shape=(1, 3)*) im Formular angegeben wird, da wir 3 Eingabedaten haben.

Normalerweise reichen zwei Schichten aus, um künstliche neuronale Netze zu funktionieren, die für maschinelles Lernen verwendet werden. Aus diesem Grund wurden in den Zeilen 36 und 37 zwei neue Schichten erstellt, die mit Eingaben (mit Inputs) verknüpft sind. Der erste Parameter 128 repräsentiert die Ausgabegröße der Ebene. Es spielt eine wichtige Rolle bei der Größe der Gewichtsmatrix. Und 128 sollte für unser Beispiel ausreichen. Der zweite Parameter *Aktivierung* wird benutzt beim Anwenden der Aktivierungsfunktion in einer Schicht. Standardmäßig wird die *relu-Aktivierung* verwendet, aber wir können eine der vielen Optionen, die Keras dafür bereitstellt, ändern und zu einer anderen wechseln. Und der dritte Parameter gibt der entsprechenden Schicht einen Namen.

Es sollte auch erwähnt werden, dass für schwerere neuronale Netzwerke eine größere Anzahl von Schichten erforderlich ist. Für unser Beispiel reicht extra zwei jedoch aus.

In Zeile 38 wird der *Ausgabe-Schicht* angegeben, in dem sich die Ausgabe befindet. Und in Zeile 40 wurden die Eingabe- und Ausgabedaten für unser System festgelegt.

In Zeile 42 wurden die Lernkonfigurationen des Modells angegeben. Das Ziel beim maschinellen Lernen ist es, die Verlustfunktion zu optimieren und ideale Gewichte von Datenkomponenten zu erhalten. Die zur Optimierung verwendete Methode wird als *Optimizer* bezeichnet. Deshalb wird ein *nadam-Optimizer* verwendet. Es ist effizient zu bedienen und verbraucht sehr wenig Speicher. Das ist in Fällen angebracht, in denen eine große Menge an Daten und Parametern zur Verfügung steht.

Auch in dieser Zeile ist in Klammern die Verlustfunktion (*loss*) angegeben. Der Zweck von Verlustfunktionen besteht darin, die Menge zu berechnen, die ein Modell während des Trainings minimieren soll. Und der letzte Parameter (*metrics*) gibt die zu überwachenden Metriken an. In unserem Beispiel ist dies die Genauigkeit (oder *accuracy*)

In der letzten Zeile 44 wird mit der *fit-Funktion* die optimale Beziehung zwischen Trainingsdaten berechnet, wobei die Eingabe- und Ausgabedaten als Argumente verwendet werden (*X\_train* und *y\_train*). Um das System zu trainieren, wird auch "*batch\_size*" angegeben, das die Daten mit der angegebenen Batchgröße teilt. Bei einem Wert von 1 werden alle Daten für das Training verwendet. Das Modell durchläuft den gesamten Datensatz eine bestimmte Anzahl von Malen (*Epochen*). Durch Erhöhen diesen Wert wird neuronales Netz genauer, aber gleichzeitig wird es mehr Zeit in Anspruch nehmen.

Der letzte geänderte Teil des Codes wird unten gezeigt:

```
50     print('\n# Predicting....')
51     predictions = model.predict(X_test)
52     print('predicted data:\n', predictions)
53
54     #Evaluation
55     print("Evaluate on test data")
56     results = model.evaluate(X_test, y_test, batch_size=8)
57     print("test loss, test acc:", results)
```

Die Zeile 51 wird benutzt, um Testdaten vorherzusagen. In Zeile 56 wird jedoch die Funktion *evaluate* verwendet, die die Genauigkeit der Vorhersage von Volumenstromdaten auswertet. Die Batchgröße wie im Lernprozess wird benötigt, um die gesamte Datenmenge zur Auswertung in kleine Teile zu unterteilen. Und am Ende wird die Genauigkeit um etwa 0.97 erhältet. Der Fehler oder die Abweichung der Datenvorhersage beträgt ungefähr 0.03 Werte.

#### 4.4.7. K-nächste-Nachbarn-Algorithmus (*KNNA*) zur Vorhersage von Daten

Bei Regressionsproblemen sagt der *KNNA* einen neuen Wert voraus, indem er den Durchschnitt der  $k$  Nachbarn zurückgibt. Zum Beispiel, wenn die fünf nächsten Eingaben von [95, 100, 105, 110, 100] hatten, dann würde der Algorithmus einen Wert von 102 zurückgeben, der der Durchschnitt ist.

Um den *KNNA* zu verwenden, muss ein Modul namens *KNeighborsRegressor* aus der *sklearn-Bibliothek* importiert werden:

```
«from sklearn.neighbors import KNeighborsRegressor»
```

Das ist der einfachste Weg, um den K-nächste-Nachbarn-Algorithmus zu verwenden, der keine mathematische Konstruktion des Modells erfordert und für den normalen Benutzer verständlich ist. Der Rest des Codes bleibt unverändert, wie es im Kapitel mit *linearer Regression* der Fall war. Nur dieser Teil wird geändert:

```
41      # Train model on training set
42      knn = KNeighborsRegressor(n_neighbors=50, weights =
43                                'distance').fit(X_train_scaled, y_train.values.ravel())
44
45      # Predicting multiple observations
46      predictions = knn.predict(X_train_scaled)
47
48      score = knn.score(X_test_scaled, y_test)
49      print("\nHow accurate it works: (1 - the best)")
50      print(score)
```

Bei der Arbeit mit dem Algorithmus wird das *KNeighborsRegressor-Modul* verwendet. Werfen wir einen Blick auf ersten Parameter in Klammern:  $k$  (*n\_neighbors*). Der Wert von  $k$  bestimmt die Anzahl der zu betrachtenden Nachbarn. Wenn  $k = 50$  verwendet wird, bedeutet das, dass wir uns die fünfzehn nächsten Nachbarn des neuen Datenpunkts ansehen.

Der zweite Parameter in Klammern zeigt, wie die Nachbarn selbst gewichtet werden. Standardmäßig die *sklearn-Bibliothek* nutzt *weights = 'uniform'* [15]. Das bedeutet, dass unabhängig davon, wie weit jeder Nachbar vom neuen Datenpunkt entfernt ist, er das gleiche Gewicht hat.

Wenn man das ändern will, um stattdessen jede Eingabe basierend auf seiner Entfernung von der neuen Eingabe zu wiegen, kann man das Argument auf *weights = 'distance'* setzen. In diesem Fall ist das Gewicht jedes Nachbarn das Gegenteil seiner Entfernung zum Datenpunkt. Auf diese Weise werden genauere Werte für die Volumenstromvorhersagen erzielt, die gerade in meiner Masterarbeit erforderlich sind.

Wenn das Programm mit diesem Algorithmus auf dem Pumpversuchstand ausgeführt wird, wird ein Vorhersagewert (Score-Parameter) im Bereich von 0.992 erzeugt. Das ist ein sehr gutes Ergebnis und noch genauer ist als bei der Verwendung einer *linearen Regression*.

#### 4.4.8. Decision Tree Regression zur Vorhersage von Daten

Die Entscheidungsbaumregression (*Decision Trees Regression*) beobachtet Merkmale eines Objekts und trainiert ein Modell in der Struktur eines Baums, um Daten in der Zukunft vorherzusagen, um eine aussagekräftige kontinuierliche Ausgabe zu erzeugen. Um dieses Modul *DecisionTreeRegressor* in Python verwenden zu können, muss es aus der *sklearn-Bibliothek* importiert werden:

```
«from sklearn.tree import DecisionTreeRegressor»
```

Somit kann der Benutzer das vorgefertigte *DecisionTreeRegressor-Modul* für das Modelltraining und die Datenvorhersage verwenden. Der geänderte Teil des Codes sieht folgendermaßen aus:

```
41 # Train model on training set
42 reg = DecisionTreeRegressor(max_depth=3).fit(X_train_scaled,
y_train.values.ravel())
43
44 # Predicting multiple observations
45 predictions = reg.predict(X_train_scaled)
46
47 score = reg.score(X_test_scaled, y_test)
48 print("\nHow accurate it works: (1 - the best)")
49 print(score)
```

Der Parameter *max\_depth* in Klammern gibt die maximale Tiefe des Baums an. Wenn es zu hoch eingestellt ist, lernen die Entscheidungsbäume zu feinen Details der Trainingsdaten und trainieren aus dem Rauschen, was für das Programm nicht gut ist. Im nächsten Kapitel zur Codeoptimierung wird erläutert, wie die optimale Baumtiefe gewählt werden kann. Für unser Beispiel wird die Tiefe des Baumes gleich drei genommen.

Und bei einer einfachen Vorhersage dieses Algorithmus beträgt der Metrik-Wert (Score-Parameter) der Volumenstromvorhersage jedoch fast 0,99, was ebenfalls ein hervorragendes Ergebnis ist.

#### 4.4.9. Random Forest Regression zur Vorhersage von Daten

*Random Forest* und ein *Decision Tree* unterscheiden sich nicht sehr. Zuerst werden zufällige Stichproben aus den Daten erstellt, danach wird für jede Stichprobe einen Entscheidungsbaum verwendet. Eine detailliertere Beschreibung der Theorie finden Sie im vorherigen Kapitel. Und die *sklearn-Bibliothek* hat auch ein *Random-Forest-Modul*, das importiert werden muss, bevor wir mit diesem Algorithmus arbeiten können:

```
«from sklearn.tree import DecisionTreeRegressor»
```

Dies ist das Merkmal der *sklearn-Bibliothek*, da sie es ermöglicht, Programme für maschinelles Lernen einfach zu schreiben, ohne Kenntnisse in Mathematik oder Theorie zu haben. Der geänderte Teil des Codes ist wie folgt:

```
41 # Train model on training set
42 randomForest = RandomForestRegressor(n_estimators = 50,
random_state = 0).fit(X_train_scaled, y_train.values.ravel())
43
44 # Predicting multiple observations
45 predictions = randomForest.predict(X_train_scaled)
46
47 score = randomForest.score(X_test_scaled, y_test)
```

```

48     print("\nHow accurate it works: (1 - the best)")
49     print(score)

```

Der Parameter *n\_estimators* in Klammern definiert die Anzahl der Bäume in der Zufallsstruktur. Man kann einen beliebigen numerischen Wert hier verwenden. Es wird jedoch empfohlen, mit einem niedrigen Wert zu beginnen (wie im Beispiel mit 50 Bäumen). Der Parameter *random\_state* ist der Startwert, der von der Zufallszahl verwendet wird. Wenn wird es mit *random\_state = 0* (oder einem anderen Wert) geschrieben, erhält der Benutzer jedes Mal den gleichen Ergebnisgenerator.

Hier bei einer einfachen Vorhersage des Algorithmus beträgt der Metrik-Wert (Score-Parameter) der Volumenstromvorhersage jedoch fast 0,99, was sehr gut ist.

#### 4.4.10. Support-Vektor-Maschine

Wie im theoretischen Teil erwähnt das Ziel der Support-Vektor-Maschine (SVM) ist es, die Hyperebene zu definieren, die die Punkte in zwei Klassen unterteilt wie unten. Dazu gibt es in der *sklearn-Bibliothek* ein Modul namens *NuSVR*. *R* bedeutet, dass diese Aufgabe die Regression bezieht. Das NuSVR-Modul funktioniert etwas besser als das normale SVR-Modul, da NuSVR eine andere Parametrisierung verwendet. Für die weitere Arbeit mit diesem Modul muss es importiert werden:

```
<<from sklearn.svm import NuSVR>>
```

Unten kann ein Codeausschnitt dargestellt werden, der sich von früheren Algorithmen unterscheidet. Der Rest des Codes bleibt genau gleich:

```

44     # Train model on training set
45     svr_regressor = NuSVR(cache_size=1000).fit(X_train_scaled,
46                                         y_train.values.ravel())
47
48     # Predicting multiple observations
49     predictions = svr_regressor.predict(X_train_scaled)
50
51     score = svr_regressor.score(X_test_scaled, y_test)
52     print("\n How accurate it works: (1 - the best)")
53     print(score)

```

Dieser Parameter *cache\_size* bezeichnet nur die Größe des Modulkerns. Es ist nicht notwendig, ihn genau anzugeben, wenn es keine Anforderungen gibt. Und der Metrik-Wert (Score-Parameter) bei einer einfachen Vorhersage dieses Algorithmus ist 0,994, was ebenfalls ein gutes Ergebnis ist und bedeutet, dass dieser Algorithmus die Daten des Volumenstromes gut vorhersagt.

Auf diese Weise wurden in diesem Abschnitt verschiedene gängige Algorithmen für maschinelles Lernen untersucht und genau erklärt, wie sie in Python implementiert werden müssen. Aber wie werden sie unter realen Bedingungen am Pumpversuchstand funktionieren? Dies wird in den folgenden Kapiteln erläutert.

## 4.5. Die Methodiken zu Optimierung von maschinellen Lernalgorithmen

### 4.5.1. Principal Component Regression (PCR) (Hauptkomponentenregression)

Die Hauptkomponentenregression (*PCR*) ist eine *Regressionstechnik*, die dem gleichen Ziel dient wie die lineare Standardregression — die Modellierung der Beziehung zwischen Ausgaben und Eingaben [16]. Der Unterschied besteht darin, dass die PCR die Hauptkomponenten als Eingaben für die Regressionsanalyse verwendet.

Der *Hauptvorteil* der *PCR* besteht darin, dass Multikollinearität in den Daten eliminiert wird, indem Hauptkomponenten entfernt werden, die mit kleinen Eigenwerten assoziiert sind.

Der erste Schritt besteht darin, das *PCA-Modul* zu importieren, das sich in der *sklearn*-Bibliothek befindet:

```
«from sklearn.decomposition import PCA»
```

Als Nächstes müssen die Hauptkomponenten (*PCA-Komponente*) generiert werden, um dies in Python zu implementieren:

```
73     pca = PCA()  
74     X_train_pc = pca.fit_transform(X_train_scaled)
```

Auf diese Weise wird die Methode zur Analyse der Trainingsdaten (Hauptkomponentenanalyse) auf das gesamte *X\_train-Array* angewendet.

Der nächste Schritt besteht darin, zu bestimmen, wie viele Eingaben für den Algorithmus benötigt werden. Dazu zeichnen wir den Regressionsverlust des mittleren quadratischen Fehlers für eine bestimmte Datenmenge auf:

```
83     rmse_list = []  
84  
85     for i in range(1, X_train_pc.shape[1] + 1):  
86         rmse_score = -1 * cross_val_score(lin_reg,  
87             X_train_pc[:, :i], y_train,  
88             scoring='neg_root_mean_squared_error').mean()  
89  
90         rmse_list.append(rmse_score)  
91  
92 # Plot RMSE vs count of principal components used  
93 plt.plot(rmse_list, '-o')  
94 plt.xlabel('Number of principal components in regression')  
95 plt.ylabel('RMSE')  
96 plt.title('How many components of data we need')  
97 plt.xlim(xmin=-1);  
98 plt.xticks(np.arange(X_train_pc.shape[1]),  
99             np.arange(1, X_train_pc.shape[1] + 1))  
100 plt.show()
```

Auf diese Weise wird eine Schleife erstellt, die den linearen Regressionsalgorithmus verwendet, aber gleichzeitig die Anzahl der Eingaben kontinuierlich erhöht.

Zum Beispiel wird am Anfang nur eine Eingabe verwendet, um die Daten vorherzusagen: die Daten von Sensor A. Danach werden bereits 2 Eingaben verwendet: die Daten von Sensor A und B. Dann bereits 3: die Sensoren A und B sowie die Motordrehzahl. Auf diese Weise wird die Anzahl der Eingaben ständig zunehmen und der Fehler für jeden von ihnen wird berücksichtigt.

Als Ergebnis des Programms wird die folgende Grafik erhalten:

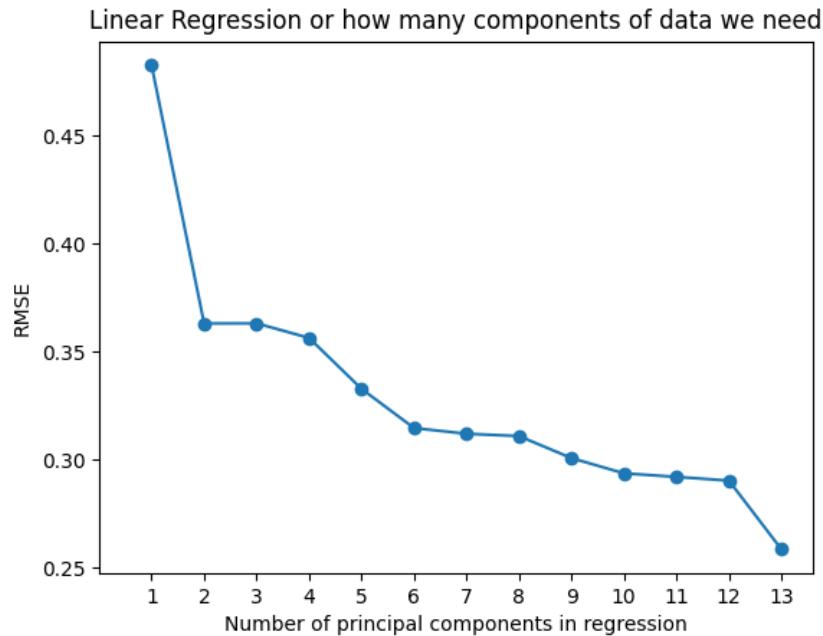


Abbildung 19. Diagramm der RMSE gegen die Anzahl der Hauptkomponenten

Deswegen kann man daraus schließen, dass je mehr Eingaben während des Vorhersageprozesses verwendet werden, desto geringer der Fehler am Ende ist. Es muss jedoch daran erinnert werden, dass, wenn die Eingaben für das Lernen des Algorithmus zu viel sind, ein Umschulungseffekt auftreten kann. Daher lohnt es sich, ein solches Diagramm zu erstellen, um zu verstehen, bei welcher Anzahl von Komponenten (Eingaben) ein geringerer Fehler auftritt.

Die folgende Abbildung zeigt die RMSE-Ergebnisse der verschiedenen Algorithmen für maschinelles Lernen, einschließlich der beiden PCR-Modelle, die mit einer unterschiedlichen Anzahl von Hauptkomponenten trainiert wurden:

	RMSE (Train Set)	RMSE (Test Set)
<b>Linear Regression</b>	0.660	0.625
<b>Lasso Regression</b>	0.659	0.627
<b>Ridge Regression</b>	0.660	0.625
<b>PCR (9 components)</b>	0.659	0.627
<b>PCR (7 components)</b>	0.664	0.635

Abbildung 20. RMSE-Ergebnisse der Regressionsmodelle.  
Je niedriger der RMSE, desto besser das Modell

Auf diese Weise kann die Analyse der Hauptkomponenten verwendet werden, um die Algorithmen des maschinellen Lernens zu optimieren, wodurch die Laufzeit des Algorithmus und der Fehler beim Training reduziert werden.

## 5. Systematischer Vergleich verschiedener Algorithmen zur Berechnung des Volumenstroms auf Frequenzumrichterdaten

Im obigen theoretischen Teil wurden 9 populäre Algorithmen zur Vorhersage von Daten untersucht, aber welcher ist besser oder schlechter für den Pumpversuchstand?

Es sollte darauf hingewiesen werden, dass für das Beispiel das *Training* jedes Algorithmus bei 3 *Eingaben* stattfand, wobei die Ergebnisse von den Indikatoren alle 0.01 Sekunden aufgezeichnet wurden. Das *Trainingsmodell* wurde 20 Minuten lang gespeichert. Diese Daten wurden in Algorithmen maschinelles Lernen verwendet. Der Code kann auf der Github-Website meiner Masterarbeit in einem Programm namens "gui\_Keras\_KNN.py" genauer betrachtet werden. Die Ein- und Ausgangsdaten befinden sich wie unten im Diagramm:

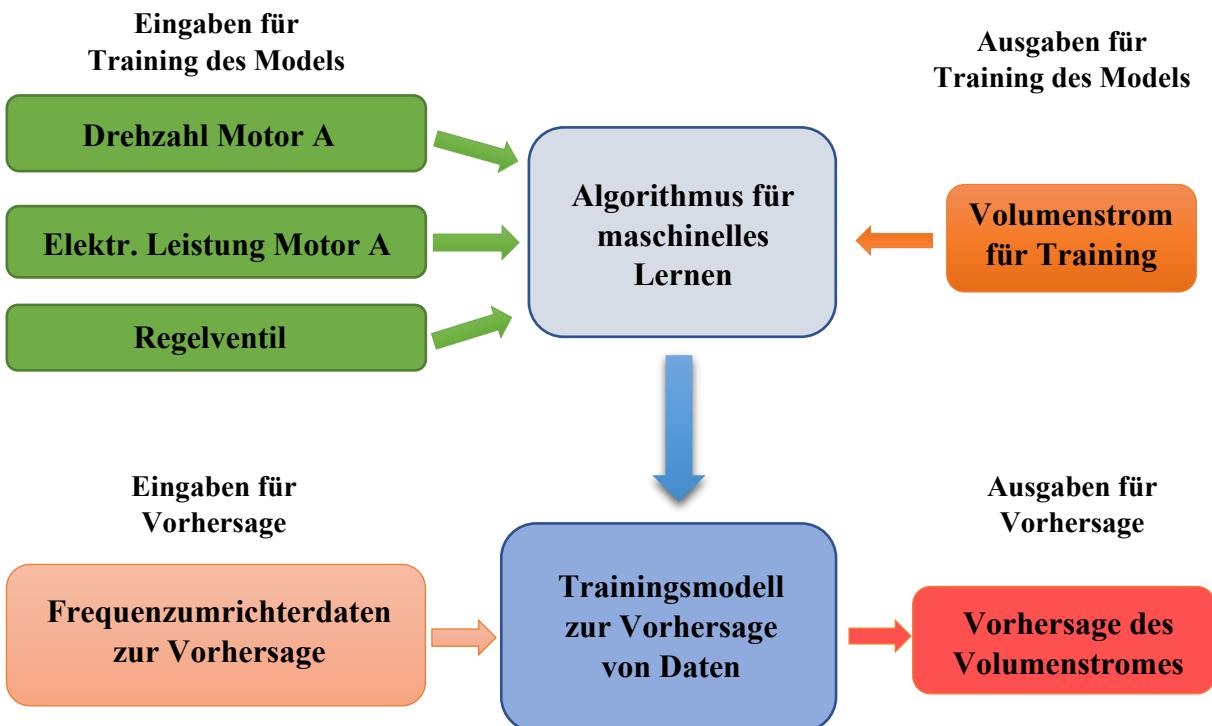


Abbildung 21. Vorhersage des Volumenstromes am Pumpversuchstand

Auf diese Weise wird ein maschinelles Lernalgorithmus trainiert, der dann zur Vorhersage des Volumenstromes verwendet wird. Das Training erfolgt mit Hilfe von drei Indikatoren: *Drehzahl*, *elektrische Leistung* des Motors A und *Regelventil*. Und die gleichen Daten werden aus dem Frequenzumrichter gelesen, die dann vom Trainingsmodell verwendet werden, um den Volumenstrom vorherzusagen.

Für den Vergleich von Volumenstromvorhersage der Algorithmen sollten zwei Metriken berücksichtigt werden: die *Geschwindigkeit* des Algorithmus und die *Genauigkeit* der

Datenvorhersagung. Deswegen wird jede dieser Metriken für alle verfügbaren Algorithmen beachtet.

## 5.1. Die Geschwindigkeit der Algorithmen

Aus dem Namen ist klar, dass diese Metrik für die Geschwindigkeit der Ausführung des Algorithmus verantwortlich ist oder wie schnell die Ergebnisse der Datenvorhersage erhalten werden. Um die Ausführungszeit zu berechnen, werden nur drei Zeilen in Code benötigt:

```
import time
start_time = time.time()
...
print("---- %s seconds ---" % (time.time() - start_time))
```

Um die Ausführungszeit des Algorithmus zu berechnen, reicht es aus, die Zeitbibliothek zu verwenden und die Startzeit von der Endzeit abzuziehen. Und das Ergebnis sollte am Ende gezeigt werden, nachdem die Daten vorhergesagt wurden.

### 5.1.1. Ein Trainingsmodell, das innerhalb von 10 Minuten trainiert wurde

Im folgenden Vergleich wird die Geschwindigkeit der Algorithmen für ein Modell veranschaulicht, das nur 10 Minuten trainiert wurde. Dies bedeutet, dass es nicht viele Eingaben gibt und ihre Größe relativ klein ist, da die Eingaben 10 Minuten lang gezeichnet wurden. Auf diese Weise kann eine Tabelle für die verfügbaren 9 Algorithmen und für dieses Trainingsmodell erstellt werden, die zeigt, wie schnell jeder von ihnen die Daten vorhersagt:

Algorithmus	Trainingszeit, s	Vorhersagezeit, s
Lineare Regression	0.216	0.0006
Lasso Regression	0.589	0.005
Ridge Regression	0.229	0.004
ElasticNet Regression	0.547	0.0045
Künstliche neuronale Netze	7.884	0.125
K-nächste Nachbarn	0.228	0.059
Decision Tree	0.225	0.003
Random Forest	1.334	0.012
Support-Vektor-Maschine	10.516	3.353

Tabelle 1. Die Geschwindigkeit der Algorithmen

Aber es ist schwer zu verstehen, was diese Zahlen bedeuten. Zum besseren Verständnis sollte die Geschwindigkeit der Algorithmen in Diagrammen angezeigt werden, die sich unten befinden:

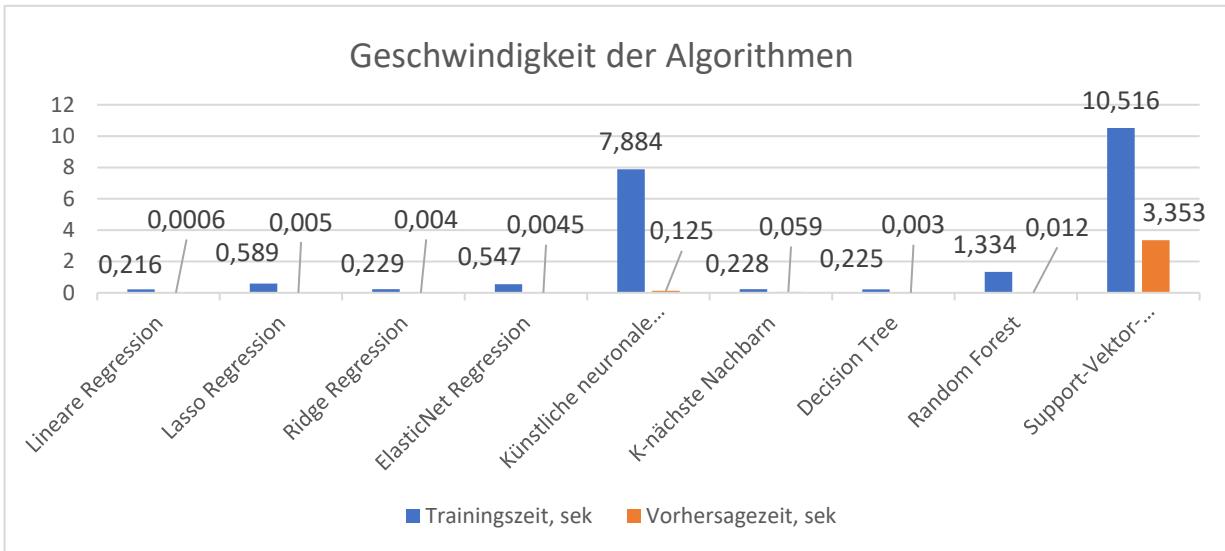


Abbildung 22. Vergleich von Trainingszeit und Vorhersagezeit mit einem 10-Minuten Trainingsmodell

Anhand der obigen Abbildung lässt sich schlussfolgern, dass das schlechteste Ergebnis von *Trainingszeit* und *Vorhersagezeit* tatsächlich von zwei Algorithmen angezeigt wird: *Künstliches neuronales Netz (KNN)* und *Support-Vektor-Maschine (SVM)*. Bei anderen Algorithmen sind die Werte der Trainingszeit und der Vorhersagezeit so klein, dass sie in der Abbildung sogar schlecht sichtbar sind.

### 5.1.2. Ein Trainingsmodell, das innerhalb von 25 Minuten trainiert wurde

Das folgende Diagramm zeigt die Geschwindigkeit der Algorithmen jedoch bereits für eine Situation, in der das Trainingsmodell länger als 25 Minuten aufgezeichnet wurde. Dadurch werden die Größe und das Volumen der Eingaben deutlich erhöht, was sich direkt auf die Geschwindigkeit der Datenvorhersage auswirkt:

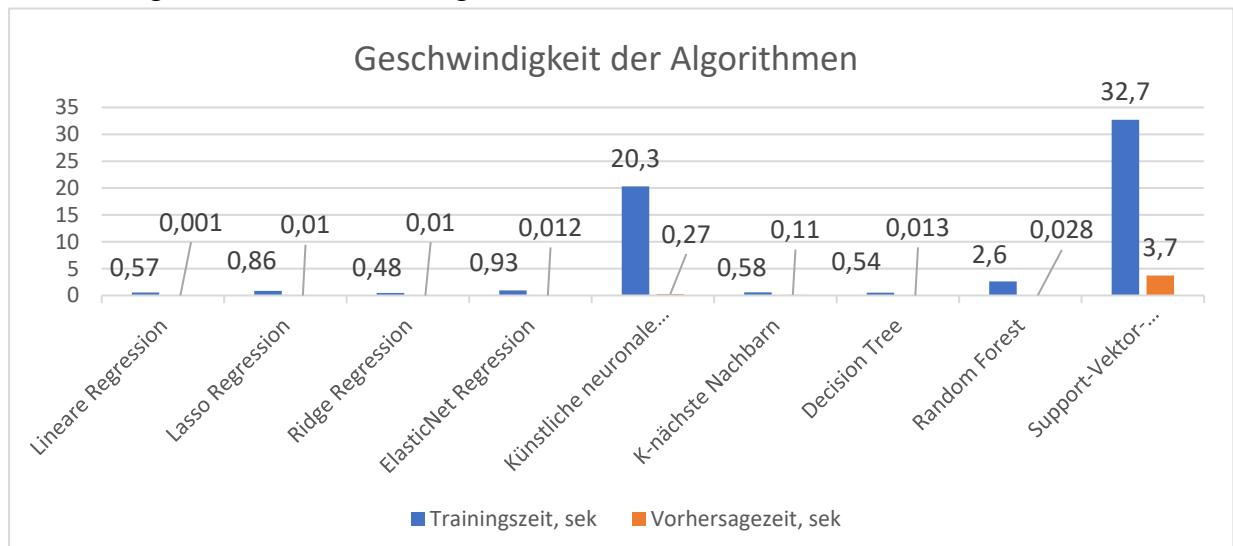


Abbildung 23. Trainingszeit und Vorhersagezeit mit einem 25-Minuten Trainingsmodell

Wie man sehen kann, hat sich die *Trainingsgeschwindigkeit* erheblich erhöht (um mehr als das 2-Fache), da die Menge der Eingaben innerhalb von 25 Minuten aufgezeichnet wurde. Aus diesem Grund ist die *Trainingsgeschwindigkeit* deutlich gestiegen. Die *Vorhersagegeschwindigkeit* auf der anderen Seite hat nicht viel zugenommen, weil es durch das Datenvolumen nicht mehr so stark beeinflusst wird. Dieses Beispiel hat gezeigt, dass es für die Geschwindigkeit sehr wichtig ist, wie groß die Eingabe- und Ausgabedaten sind.

Es ist wichtig zu verstehen, dass die *Vorhersagezeit* für den Pumpversuchstand von größerer Bedeutung ist, da das *Training* nur einmal zu Beginn des Programmstarts stattfindet. Aber die *Datenvorhersage* wird jede Sekunde geschehen, sobald das Programm gestartet wird. Daher hat Spalte 1 mit *Trainingszeit* einen geringeren Wert als Spalte 2 mit *Vorhersagezeit*. Die Vorhersage wird während des Betriebs des Pumpversuchstandes in Echtzeit erfolgen und hier ist es wichtig, die optimalen Vorhersagezeit zu wählen.

Und nach den Ergebnissen zu urteilen, sind die besten zwei Algorithmen in Bezug auf Trainingszeit und Vorhersagezeit: *Lineare Regression* und *Decision Tree*. Und das schlechteste Resultat wurde von *Künstliche neuronale Netze* und *Support-Vektor-Maschine* gezeigt.

## 5.2. Beschreibung des Trainingsmodells, das in den Tests verwendet wird

In diesem Kapitel wird das Trainingsmodell beschrieben, das in den nachfolgenden Kapiteln 5.3 und 5.4 für die Tests von Algorithmen verwendet wird. Diese Beschreibung wird erstellt, um besser zu verstehen, auf welche Daten das Modell trainiert wurde, um einige Werte vorherzusagen. Und insgesamt wurden 3 *Eingaben* und 1 *Ausgabe* verwendet. Das Diagramm mit den Eingangs- und Ausgangsdaten wurde in Abbildung 21 dargestellt.

Der Trainingsprozess des Modells besteht aus drei Stufen und unterscheidet sich hauptsächlich durch den Öffnungsgrad des Regelventils. In diesem Beispiel dauert es etwa 5-7 Sekunden zwischen den einzelnen Schritten. Deswegen bedeutet der Pfeil “→” 5-7 Sekunden. Unten werden alle drei Schritte des Modelltrainings beschrieben:

*Erste Stufe:* 30% ist der Öffnungsgrad des Regelventils → Drehzahl des Motors A ist 800 1/s → Drehzahl des Motors A ist 1300 1/s → Drehzahl des Motors A ist 1800 1/s → Drehzahl des Motors A ist 2300 1/s

*Zweite Stufe:* 60% ist der Öffnungsgrad des Regelventils → Drehzahl des Motors A ist 800 1/s → Drehzahl des Motors A ist 1300 1/s → Drehzahl des Motors A ist 1800 1/s → Drehzahl des Motors A ist 2300 1/s

*Dritte Stufe:* 90% ist der Öffnungsgrad des Regelventils → Drehzahl des Motors A ist 800 1/s → Drehzahl des Motors A ist 1300 1/s → Drehzahl des Motors A ist 1800 1/s → Drehzahl des Motors A ist 2300 1/s

Um die Trainingsschritte des Modells besser zu verstehen, wird unten auch eine Abbildung mit allen Stufen für den Öffnungsgrad des Regelventils und die Motordrehzahl dargestellt:

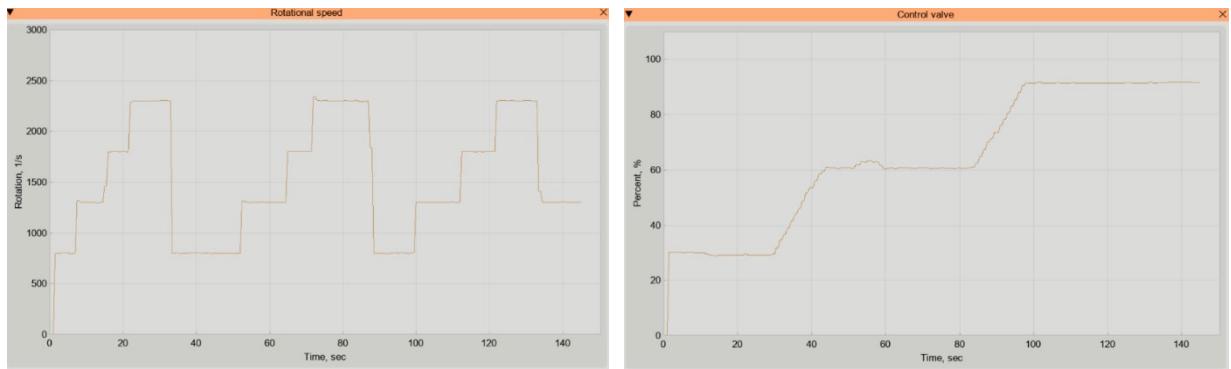


Abbildung 24. Erstellen eines Trainingsmodells zum Testen von Algorithmen

So wird nach drei Stufen ein Trainingsmodell erstellt und trainiert. Diese Kombinationen von Motordrehzahl und Öffnungsgrad des Regellventils reichen aus, um Algorithmen unter den gleichen Bedingungen (Kapitel 5.3) und unter Bedingungen mit zufälligen Werten zu testen (Kapitel 5.4). Dieses Trainingsmodell kann jetzt zur Vorhersage von Volumenstromwerten verwendet werden.

### 5.3. Die Genauigkeit der Datenvorhersagung in Echtzeit

Die Genauigkeit ist genauso wichtig wie die Ausführungsgeschwindigkeit. Wenn der Algorithmus schnell ist, aber die Vorhersage der Daten schlecht oder ungenau ist, macht es keinen Sinn, diesen Algorithmus zu verwenden. Der folgende Teil in Code wird benutzt, um diesen Indikator zu berechnen. Ein Beispiel zeigt den SVR-Algorithmus:

```
predictions = svr_regressor.predict(array1)
MainDifference = float (predictions1) - float (readedValuesfromPLC[2])
Differenz.append(float(abs(MainDifference)))
MeanDifferenz = mean (Differenz)
```

Auf diese Weise werden die *Volumstromdaten* vorhergesagt. Und danach wird die Differenz zwischen dem tatsächlichen Wert des Volumenstromes und dem vorhergesagten Wert berücksichtigt. Am Ende wird der Durchschnitt dieser Abweichung für die gesamte Dauer des Pumpversuchstandsbetriebs ermittelt.

In den folgenden Kapiteln wird jeder der Algorithmen untersucht, um zu zeigen, wie genau jeder die Volumstromdaten vorhersagt. Aus allen linearen Algorithmen (Lineare, Lasso, Ridge und ElasticNet Regression) wird ElasticNet zum Testen ausgewählt, weil es das Beste aus anderen Regressionen nimmt. Die *blaue* Linie zeigt den tatsächlichen Volumenstrom und die *orange gefarbene* Linie gibt den vorhergesagten Volumenstrom an.

Hier erfolgt der Vergleich von Algorithmen in *Echtzeit*, da es interessant ist, die Genauigkeit der Datenvorhersage bei der Verwendung am Pumpversuchstand zu vergleichen. Das passiert genau dann, wenn die Daten vom Frequenzumrichter zufällig und ohne Reihenfolge kommen. So wird die Arbeit des Pumpversuchstandes unter realen Bedingungen simuliert, was sehr gut ist.

### 5.3.1. ElasticNet Regression zum Vorhersagen von Daten in Echtzeit

Für erstes Beispiel wurde ein solcher ElasticNet-Algorithmus getestet, um die Volumstromdaten vorherzusagen, deren Ergebnis unten zu sehen ist:

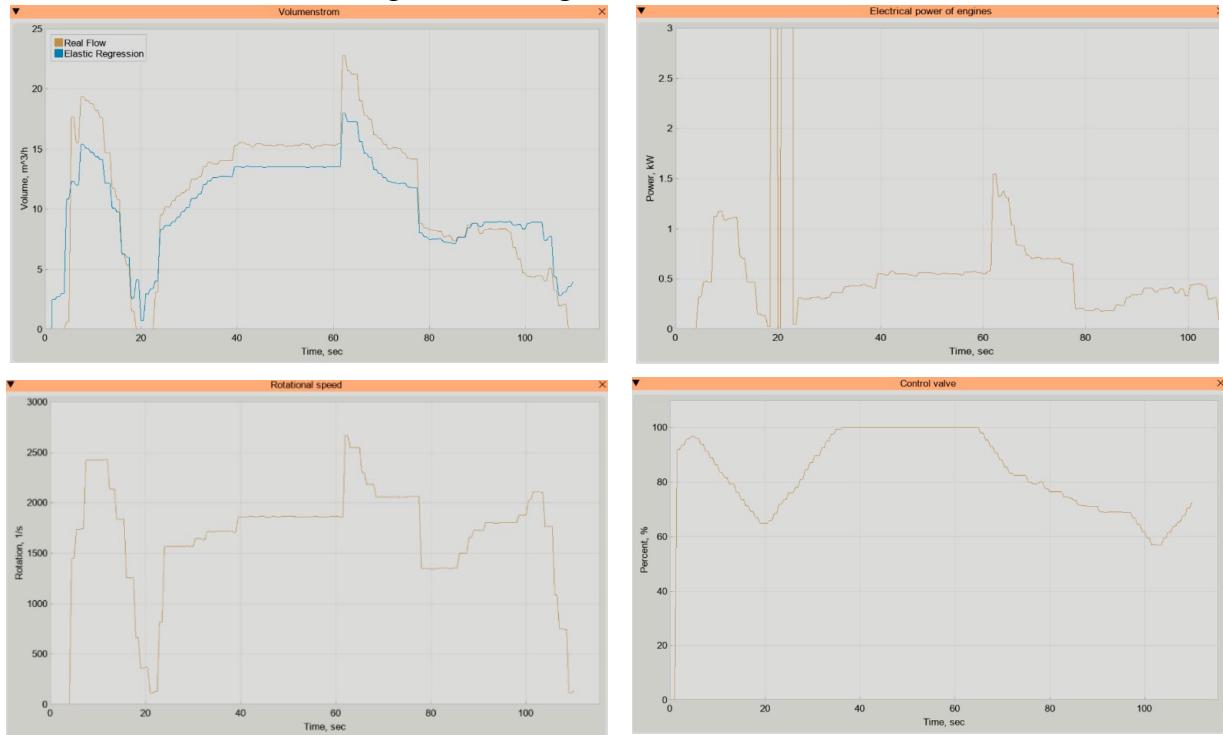


Abbildung 25. Vorhersage der Volumstromdaten in Echtzeit bei ElasticNet

Hier zeigen verschiedene Farben den *tatsächlichen* Volumstrom und die *Vorhersagen* mit dem ElasticNet-Algorithmus an. Darüber hinaus werden auch alle Eingaben angezeigt: *Drehzahl* und elektrische *Leistung* des Motors und *Regelventil*. Wie man sehen kann, funktioniert der Algorithmus im Allgemeinen nicht so schlecht. Es sagt voraus, in welchem Bereich sich der tatsächliche Volumenstrom befindet. Dies ist bereits gut und zeigt, dass der Algorithmus seine Aufgabe bewältigt und für unseren Zweck verwendet werden kann.

Wenn man sich andere Diagramme ansieht, kann man bestätigen, dass der Algorithmus funktioniert nicht so genau, wenn die Werte von *Drehzahl* und elektrische *Leistung* des Motors zu hoch sind. Zum Beispiel steigen die elektrische *Leistung* und die *Motordrehzahl* bei etwa 10 Sekunden dramatisch an. Und an diesem Punkt ist die Vorhersage des Volumenstromes nicht mehr so genau. Daraus kann man eine Schlussfolgerung ziehen: als das *Trainingsmodell* erstellt wurde, gab es keine solche Kombination von solchen hohen Werten für elektrische *Leistung* und *Motordrehzahl*. Deswegen wurde das Trainingsmodell nicht für diese Mischung von Eingaben trainiert und weiß nicht, was sie in einer gegebenen Situation vorhersagen soll. Für ein besseres Ergebnis sollten genauere Modelle für das Training verwendet werden.

Zum Beispiel kann ein neues Trainingsmodell erstellt werden, das länger als eine Stunde trainiert wird. Dann wird es eher diese unvorhersehbare Kombination von Werten haben. Es sollte jedoch daran erinnert werden, dass je schwieriger das Trainingsmodell ist, desto länger wird es trainieren und vorhersagen (die Geschwindigkeit der Vorhersage wird abnehmen, wie in Kapitel 5.1 beschrieben).

Plötzliche elektrische Spannungsspitzen bei 20 Sekunden können ignoriert werden, da sie innerhalb von nur einer halben Sekunde 10 kV erreichen können und durch die Besonderheiten des Motorbetriebs verursacht werden.

Als Ergebnis der Berechnungen wurde festgestellt, dass der Durchschnitt der Abweichung des vorhergesagten Volumenstromes vom realen Volumenstrom bei ElasticNet-Algorithmus etwa 12.1% beträgt. Insgesamt ist das ein gutes Resultat, es zeigt, dass der Algorithmus funktioniert. Der Unterschied kann jedoch deutlich reduziert werden, wenn das Trainingsmodell verbessert wird. Dies könnte für zukünftige Forschung nützlich sein.

### 5.3.2. Künstliches neuronales Netz (KNN) zum Vorhersagen von Daten in Echtzeit

In diesem Kapitel wird der KNN-Algorithmus getestet. Wenn ein gutes neuronales Netz mit 10 Trainingsphasen und einer 100%-Analyse der Trainingsdaten erstellt wurde, wurden die folgenden Ergebnisse erzielt:

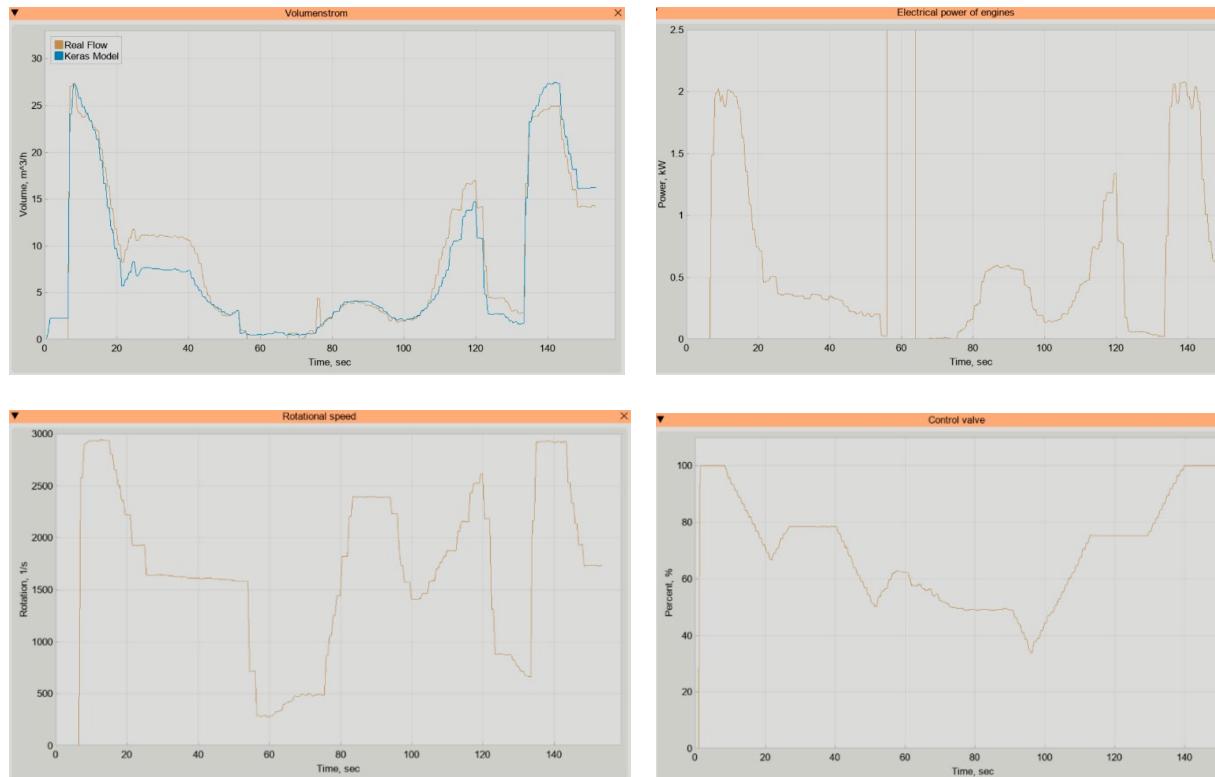


Abbildung 26. Vorhersage der Volumstromdaten in Echtzeit bei KNN

Wie man sehen kann, ist das neuronale Netz zwar gut trainiert. Und der Algorithmus funktioniert sehr fein und kann Werte genau vorhersagen. Und bei 30 Sekunden gibt es die größte Abweichung bei der Vorhersage von Volumenstrom. Dies liegt daran, dass ein weniger genaues Trainingsmodell verwendet wird, wie war es bei der *Elastic Regression*.

Es ist auch notwendig, einen Kompromiss zwischen Einfachheit und der Schwierigkeit des Trainings für KNN-Modelle zu finden. Je besser ein neuronales Netz trainiert ist, desto schwieriger ist es, mit Echtzeitdaten zu arbeiten. Eine Zusammenfassung der Anpassungstypen beim Vergleich von Modelltyp und Verlusten ist in der Abbildung unten zu sehen:

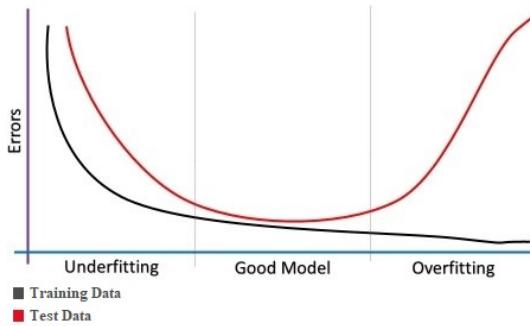


Abbildung 27. Vergleich von Modelltyp und Verlusten für KNN aus [17]

Deswegen man muss verstehen, dass für die optimale Leistung des Karas-Modells das optimalen Trainingsmodell für den Algorithmus ausgewählt werden muss.

Und der durchschnittliche Abweichungswert des vorhergesagten und tatsächlichen Volumenstromes bei KNN beträgt 7.3%. Das ist ein sehr gutes Ergebnis und künstliche neuronale Netze können weiter verwendet werden.

### 5.3.3. K-nächste-Nachbarn-Algorithmus zum Vorhersagen von Daten in Echtzeit

Hier für unser Beispiel wurde ein K-nächste-Nachbarn-Algorithmus (KNNA) Algorithmus getestet, um die Volumstromdaten vorherzusagen, deren Ergebnis unten zu sehen ist:

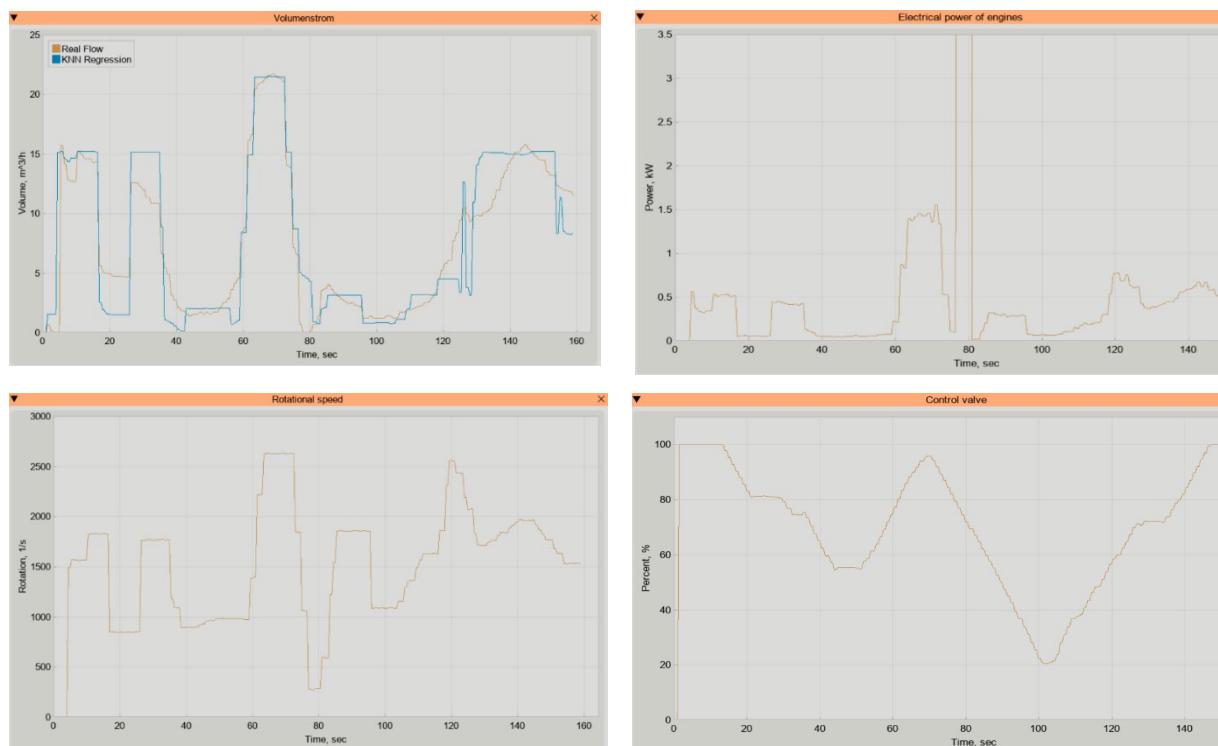


Abbildung 28. Vorhersage des Volumstromes in Echtzeit bei KNNA (200 Nachbarn)

Nach den Testergebnissen können wir sagen, dass sich der *KNNA* ziemlich gut zeigt. Aber manchmal gibt es selten einige scharfe Spitzenwerte, was mit der Arbeit des Algorithmus selbst zusammenhängt.

Und manchmal gibt es bei einigen Vorhersagebereichen Abweichungen vom realen Volumenstrom. Der Grund dafür ist die Funktionsweise des *K-nächste-Nachbarn-Algorithmus* sowie das Trainingsmodell, mit dem die Daten vorhergesagt wurden. Weitere Details zu dem von uns in den Tests verwendeten Trainingsmodell wurden in Kapitel 5.2 beschrieben.

Das folgende Bild zeigt, wenn das Trainingsmodell gut trainiert ist und eine bessere Vorhersage für den Volumenstrom aufweist. Wie man sehen kann, sind die Vorhersagewerte sehr genau und die Abweichung beträgt nur ein paar Prozent:



Abbildung 29. Vorhersage bei KNNA mit einem guten Trainingsmodell

Aus diesem Grund ist es sehr wichtig, ein Trainingsmodell zu haben, das in jedem Wertebereich gut trainiert wird.

Die durchschnittliche Abweichung des prognostizierten und tatsächlichen Volumenstromes bei *KNNA-Algorithmus* beträgt 9.6%. Somit können wir sagen, dass der Algorithmus auch recht gut funktioniert und die Daten mit geringerer Differenz vorhersagt.

### 5.3.4. Decision Tree zum Vorhersagen von Daten in Echtzeit

Dieses Kapitel zeigt die Arbeit des *Decision Tree-Algorithmus* bei der Vorhersage von Daten. Die Abbildung für den Decision-Tree-Algorithmus befindet sich auf der nächsten Seite, um es anschaulicher zu machen:

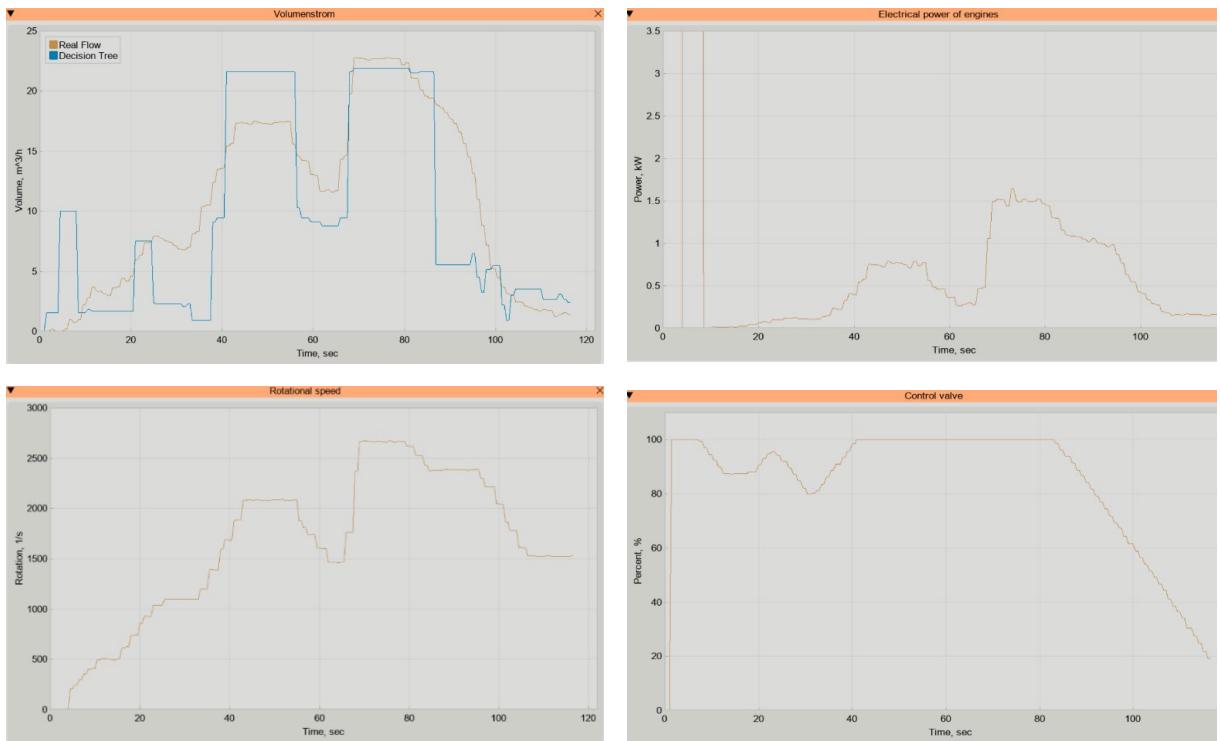


Abbildung 30. Vorhersage des Volumstromes in Echtzeit bei Decision Tree (Baumtiefe ist 30)

Wenn man sich das Bild oben anschaut, kann man daraus schließen, dass dieser Algorithmus die Daten nicht so genau vorhersagt. Das *Hauptproblem* besteht darin, dass Decision Tree die Volumenstromswerte zu drastisch und schnell ändert, was zu starken Sprüngen führt. Und es kann links im Volumenstrom-Bild zu sehen sein, dass der Algorithmus niedrige Werte vorhergesagt hat, die nicht der Realität entsprachen.

Es sollte auch davon ausgegangen werden, dass ein hoher Öffnungsgrad des Ventils die Genauigkeit der Vorhersage des Volumenstromes beeinflusst. Wenn der Ventilstand um 100% geöffnet ist, fällt die Genauigkeit der Strömungsvorhersage merklich ab. Dieser Parameter wirkt sich offenbar auf das maschinelle Lernen dieses Algorithmus aus. Und daher ist es für zukünftige Forschungen notwendig, ein Trainingsmodell zu haben, das für Decision-Tree-Algorithmus auf das Öffnungsniveau des Ventils trainiert wird.

Und die durchschnittliche Abweichung des prognostizierten und tatsächlichen Volumenstromes hier beträgt 22,8%. Das ist bisher die *größte* Abweichung des realen und vorhergesagten Werts unter allen Algorithmen. Leider prognostiziert *Decision Tree* Daten schlechter als *andere maschinelle Lerntechniken*. Vielleicht kann es immer noch für unseren Zweck verwendet werden, wenn dieser Algorithmus optimiert und ein verbessertes Trainingsmodell erstellt werden kann. Es erfüllt die Funktion der Datenvorhersage ganz normal, aber nicht so genau und mit Sprungübergängen.

### 5.3.5. Random Forest zum Vorhersagen von Daten in Echtzeit

Das Ergebnis der Anwendung des Random-Forest-Algorithmus kann unten dargestellt werden:

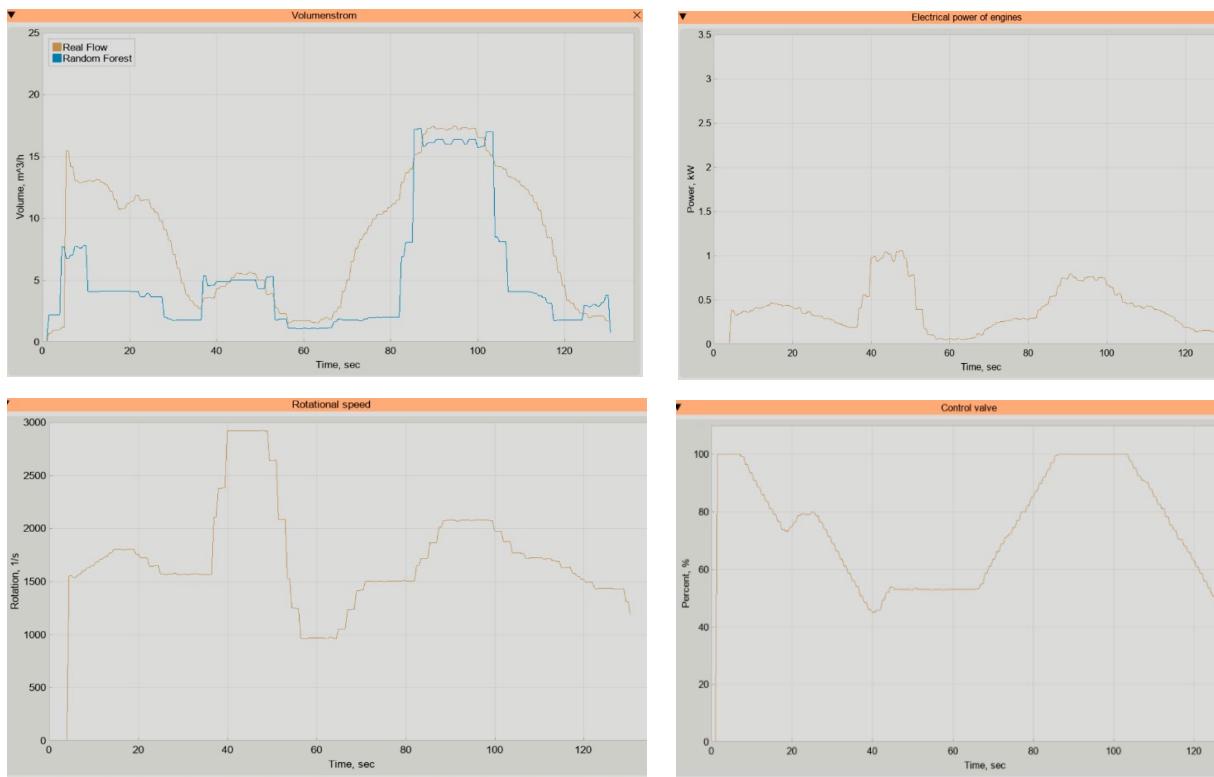


Abbildung 31. Vorhersage des Volumenstromes in Echtzeit bei Random Forest  
(Anzahl der Bäume ist 100)

Wie man das sehen kann, funktioniert dieser Algorithmus nicht so korrekt, obwohl er versucht, die tatsächlichen Werte des Volumenstromes vorherzusagen. Aber oft zeigt der *Random-Forest-Algorithmus* verschiedene Fehler und seine Vorhersagen springen auch (wie bei *Decision Tree*), wenn sich die Volumenstrom-Metriken ändern.

Hier fällt (im Gegensatz zu anderen Algorithmen) bei hoher elektrischer Spannung und Motordrehzahl die Genauigkeit der Vorhersage ab. Dies ist bei 40 und 90 Sekunden deutlich sichtbar. Daher ist es für einen solchen Fall notwendig, ein separates Trainingsmodell zu trainieren, das diese Indikatoren bei niedrigen Werten berücksichtigt. Dadurch wird sich die Vorhersage verbessert und kann in weiteren Forschungen verwendet werden.

Der durchschnittliche Abweichungswert des vorhergesagten und tatsächlichen Volumenstromes bei Random-Forest-Algorithmus beträgt 21.2%. Es ist schwer zu verstehen, was der Grund für solche Fehler bei der Anwendung dieses Algorithmus ist. Hier werden die gleichen Eingaben und Ausgaben benutzt. Vielleicht sollte ein besser trainiertes Modell verwendet werden, um diesen Algorithmus zu trainieren.

Oder es ist auch wahrscheinlich, dass der *Random-Forest-Algorithmus* selbst nicht für Echtzeit-Datenvorhersageprobleme geeignet ist, da er den Volumenstrom nicht genau vorhersagt.

### 5.3.6. Support-Vektor-Maschine (SVM) zum Vorhersagen von Daten in Echtzeit

Der letzte der untersuchten Algorithmen ist der *SVM-Algorithmus*. Die vorhergesagten Werte können unten im Bild sichtbar sein:

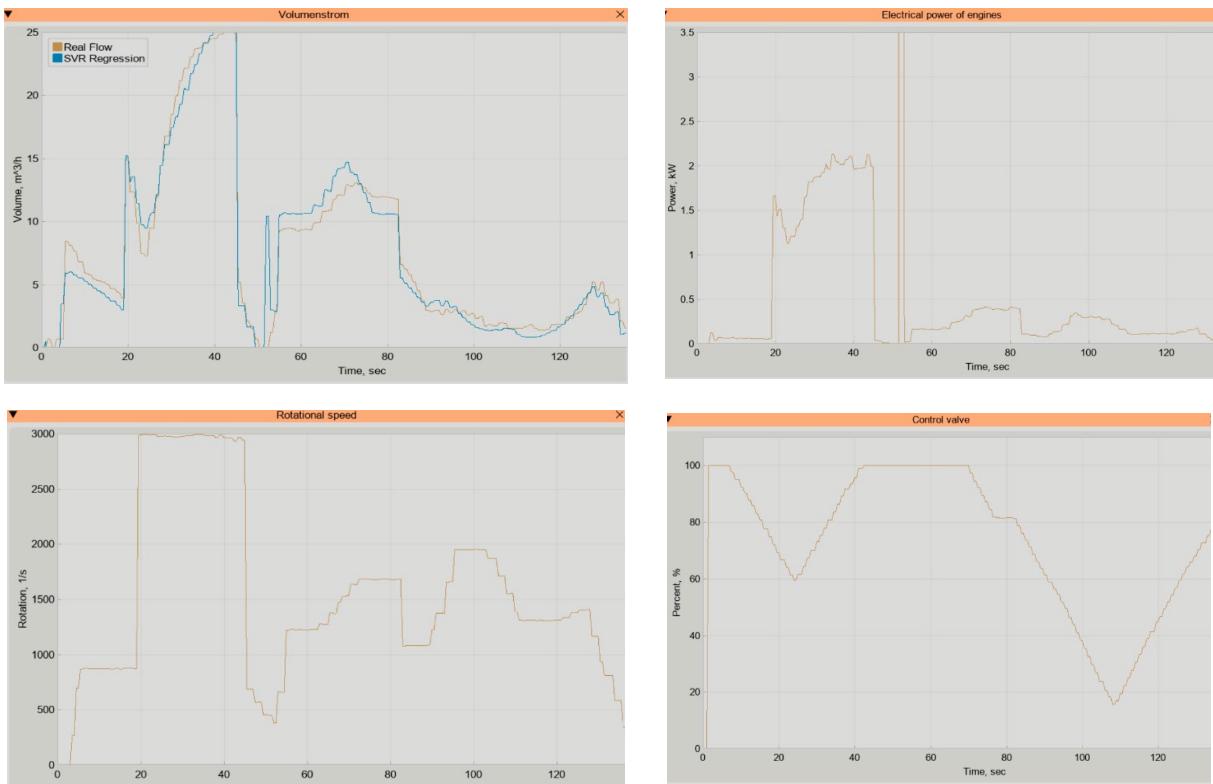


Abbildung 32. Vorhersage des Volumstromes in Echtzeit bei SVM-Algorithmus

Daraus kann geschlossen werden, dass *SVM-Algorithmus* sehr gut und fast perfekt funktioniert. Es gibt einige Abweichungen bei der Vorhersage des Volumenstromes. Aber der SVM-Algorithmus konnte die Werte genau vorhersagen, wenn sich der Motor um 3000 1/s drehte, obwohl der Algorithmus nicht auf diese Eingaben trainiert wurde.

Die maximalen Trainingswerte für die Drehung des Motors waren 2300 1/s. Auf diese Weise kann der *SVM-Algorithmus* Werte unter ungewöhnlichen Bedingungen wirklich gut vorhersagen.

Und der durchschnittliche Abweichungswert beträgt 4.8%, wenn man im Durchschnitt die Arbeit des Algorithmus schätzt. Das ist das beste Ergebnis der Vorhersage des Volumenstromes unter allen Algorithmen.

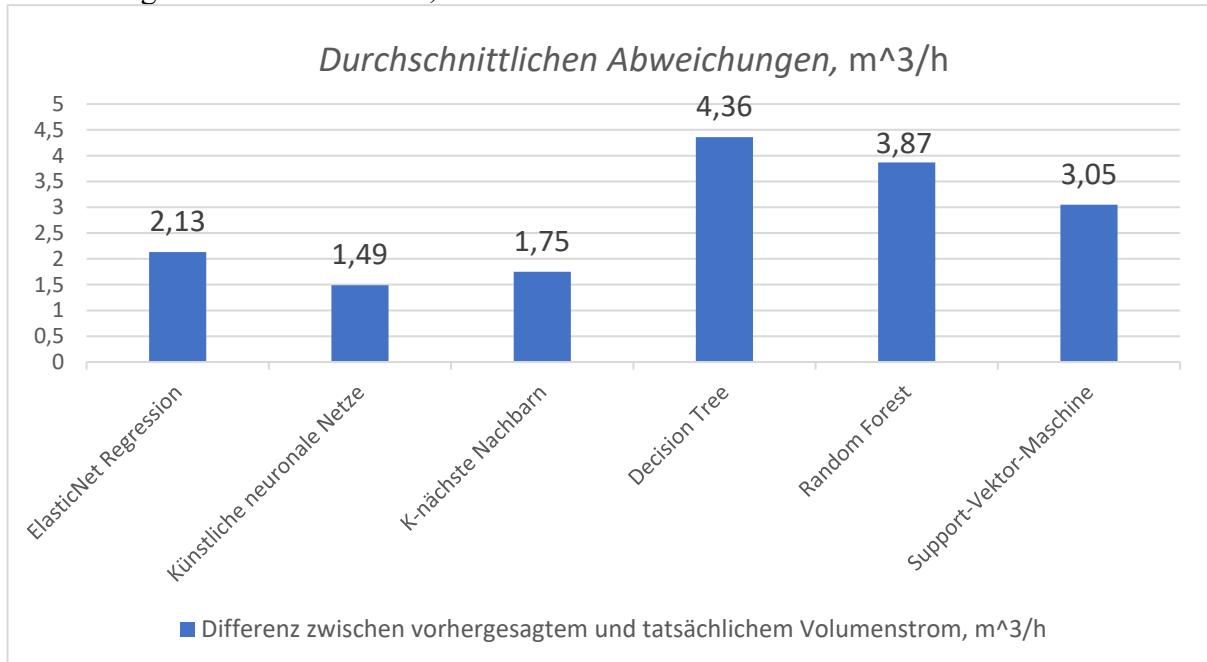
### 5.3.7. Allgemeiner Vergleich der Genauigkeit von Algorithmen zum Vorhersagen bei Verwendung in Echtzeit

Die folgende Tabelle enthält die Ergebnisse des durchschnittlichen Unterschieds zwischen dem realen und dem vorhergesagten Volumenstrom für verschiedene betrachtete Algorithmen:

<i>Algorithmus</i>	<i>Durchschnittlichen Abweichungen, m^3/h</i>
ElasticNet Regression	2.13
Künstliche neuronale Netze	1.49
K-nächste-Nachbarn	1.75
Decision Tree	4.36
Random Forest	3.87
Support-Vektor-Maschine	1.17

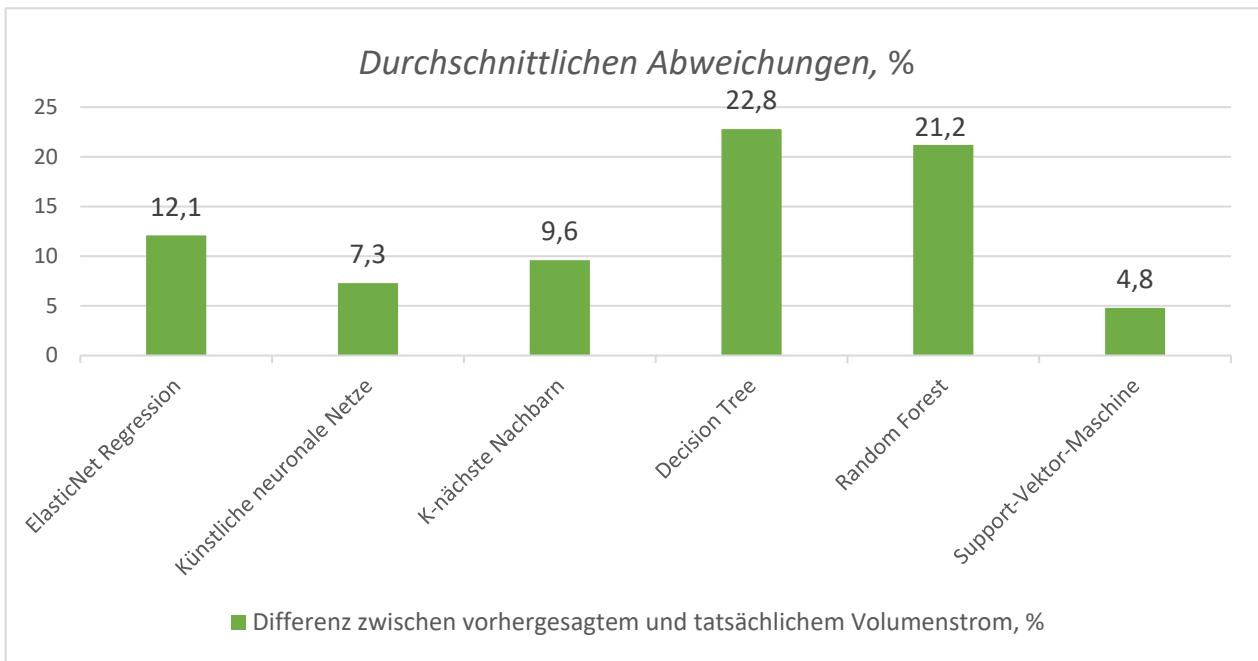
*Tabelle 2. Durchschnittliche Differenz zwischen vorhergesagten und realen Volumenströme bei Verwendung in Echtzeit*

Um besser zu verstehen, welcher Algorithmus die Daten am genauesten vorhersagt, sollte ein weiteres Diagramm erstellt werden, das sich unten befindet:



*Abbildung 33. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen*

Und zur besseren Visualisierung wird unten auch ein Diagramm der Differenz zwischen dem tatsächlichen und dem vorhergesagten Volumenstrom in Prozent dargestellt:



*Abbildung 34. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen in Prozent*

Je kleiner der Abweichungswert ist, desto besser ist er, weil er dem tatsächlichen Volumenstrom am nächsten kommt. Daraus kann geschlossen werden, dass es am besten ist, *Künstliche neuronale Netze* und *SVM-Algorithmus* zu verwenden, um den Volumenstromdaten vorherzusagen. Es sind diese Algorithmen, die einen höheren Vorhersagegrad haben.

Die Algorithmen *Random Forest* und *Decision Tree* sollten ebenfalls beiseitegelassen werden, da hier die Differenz zwischen dem tatsächlichen und dem vorhergesagten Volumenstrom in Prozent ziemlich hoch ist, was im Vergleich zu anderen ungenau ist.

Daher kann die Schlussfolgerung gezogen werden, dass die Algorithmen *Künstliche neuronale Netze* und *SVM-Algorithmus* hervorragende Werte aufweisen. Und sie sind besser für den Pumpversuchstand geeignet, da die Abweichungen hier niedrig sind.

#### 5.4. Die Genauigkeit der Datenvorhersagung unter gleichen Bedingungen

Im vorherigen Kapitel 5.3 wurde ein Vergleich von Algorithmen in Echtzeit gezeigt, wenn es keine Datenreihenfolge gab. Es war ein Chaos, als die Eingaben zufällig generiert wurden.

Das ist interessant, da es uns den realen Nutzungsbedingungen des Pumpversuchstandes näher bringt. Aber es wird viel nützlicher sein, Algorithmen unter *gleichen Bedingungen* zu vergleichen. Zum Beispiel, wenn es eine bestimmte Eingabesequenz gibt. Für jeden Algorithmus wurden hier die identischen Eingaben verwendet, also alle Algorithmen hatten die ähnlichen Bedingungen, z. B. die gleichen Motordrehzahlen und die gleichen Öffnungsstufen des Regelventils.

#### 5.4.1. ElasticNet Regression zum Vorhersagen von Daten unter gleichen Bedingungen

*ElasticNet-Regression* wurde hier als erstes Beispiel getestet, um die Volumstromdaten unter gleichen Bedingungen vorherzusagen, deren Ergebnis unten zu sehen ist:

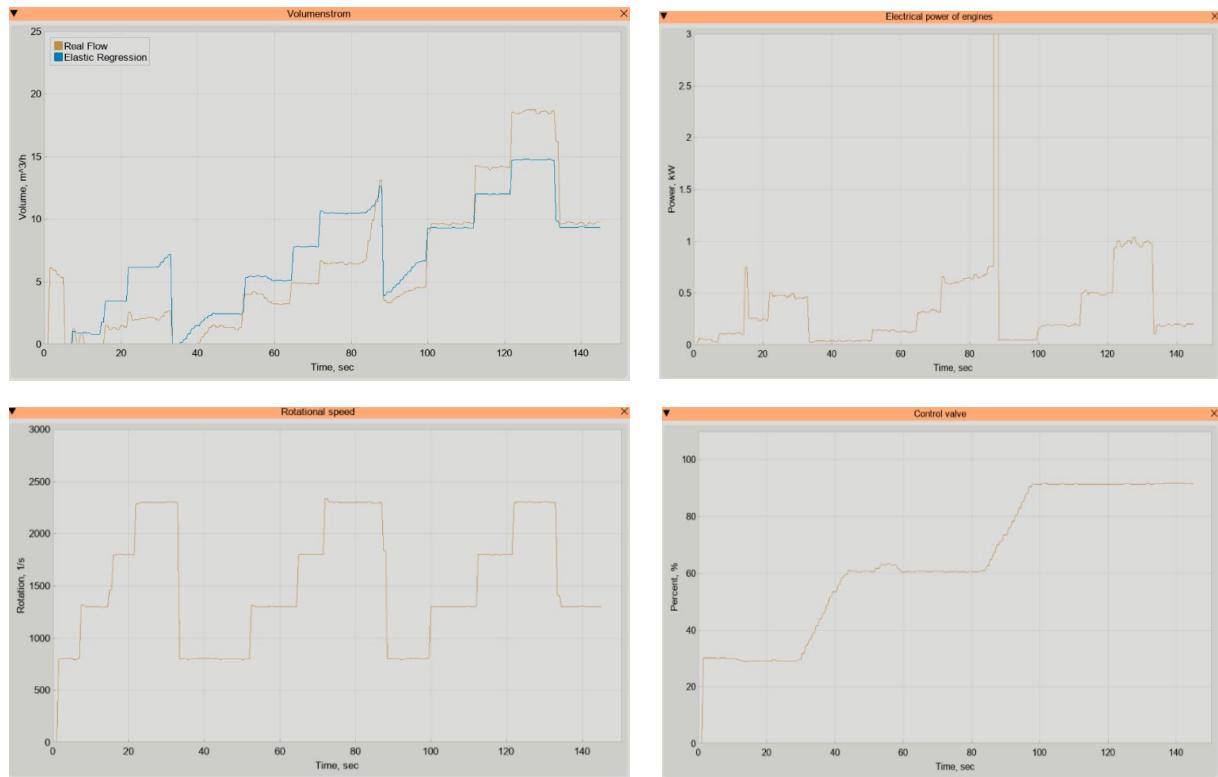


Abbildung 35. Vorhersage des Volumstromes unter gleichen Bedingungen bei ElasticNet-Regression

Wie man sehen kann, ist die Vorhersage des Volumenstromes im Allgemeinen genau. Hier ist es schwierig zu sagen, in welchem Wertebereich der *ElasticNet-Algorithmus* nicht so gut funktioniert, da er insgesamt in jedem Wertebereich identische Ergebnisse hat und überall Abweichungen vorhanden sind.

Und der durchschnittliche Abweichungswert beträgt 11.2%, wenn man im Durchschnitt die Arbeit des Algorithmus schätzt. Das ist wesentlich besser, wenn man die Leistung der ElasticNet-Regression mit dem vorherigen Kapitel 5.3.1 vergleicht, in dem Echtzeitdaten vorhergesagt wurden.

#### 5.4.2. Künstliches neuronales Netz (KNN) zum Vorhersagen von Daten unter gleichen Bedingungen

In diesem Kapitel wird künstliches neuronales Netz betrachtet und getestet, um die Volumstromdaten vorherzusagen. Das Vorhersageergebnis des Algorithmus ist unten zu sehen:

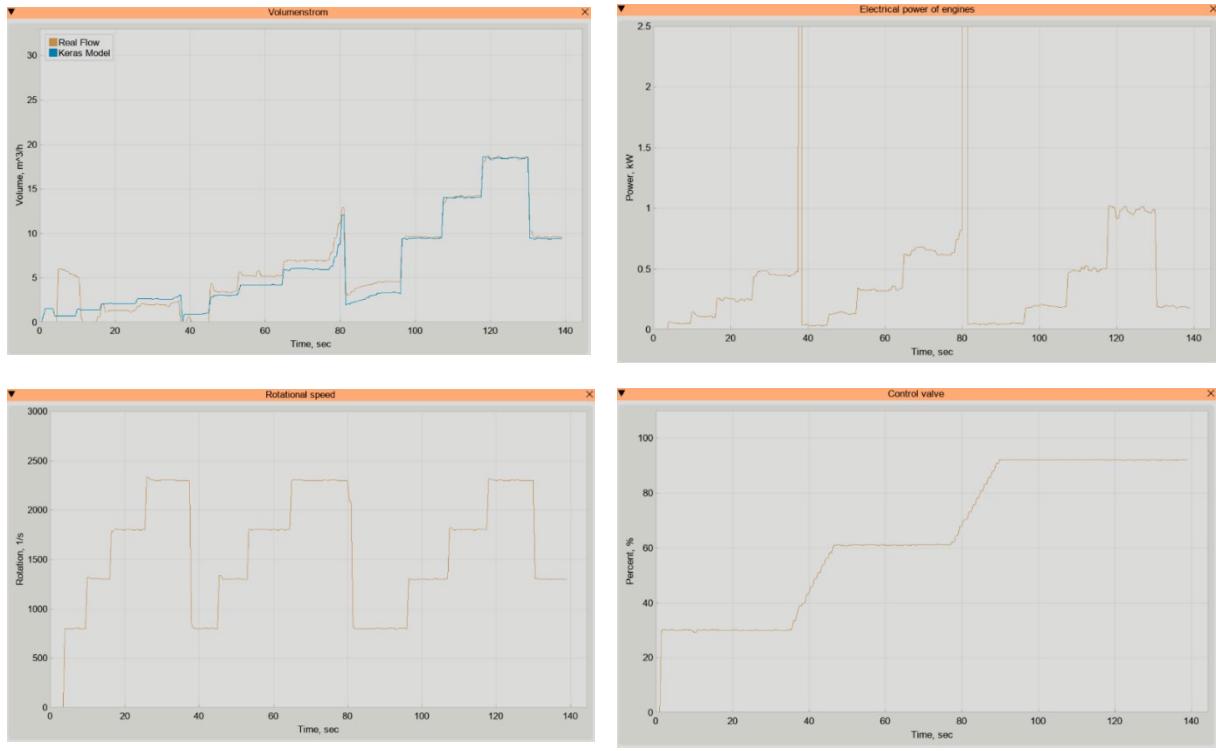


Abbildung 36. Vorhersage des Volumstromes unter gleichen Bedingungen bei Keras-Modell

Hier kann man sofort bemerken, dass die Vorhersagen der Daten im Allgemeinen sehr genau sind. Tatsächlich gibt es während der gesamten Arbeit bei Keras-Modell (KNN) keine signifikanten Abweichungen. Die genauesten Werte sind vorhanden, wenn der Öffnungsgrad des Ventils mit 90 % hoch ist.

Der durchschnittliche Abweichungswert des vorhergesagten und tatsächlichen Volumenstromes beträgt 4.5%. Das sind sehr genaue Werte, wenn sie mit anderen Algorithmen wie Random Forest oder Decision Tree verglichen werden. Natürlich können die neuronalen Netze noch verbessert werden, aber die 4.5% Abweichung ist bereits ein gutes Ergebnis.

### 5.4.3. K-nächste-Nachbarn-Algorithmus zum Vorhersagen von Daten unter gleichen Bedingungen

Unten werden die Testergebnisse des K-nächste-Nachbarn-Algorithmus für die Vorhersage des Volumenstromes angezeigt. Die Testbedingungen für alle Algorithmen sind gleich, deswegen interessiert uns nur die Genauigkeit der Datenvorhersage:

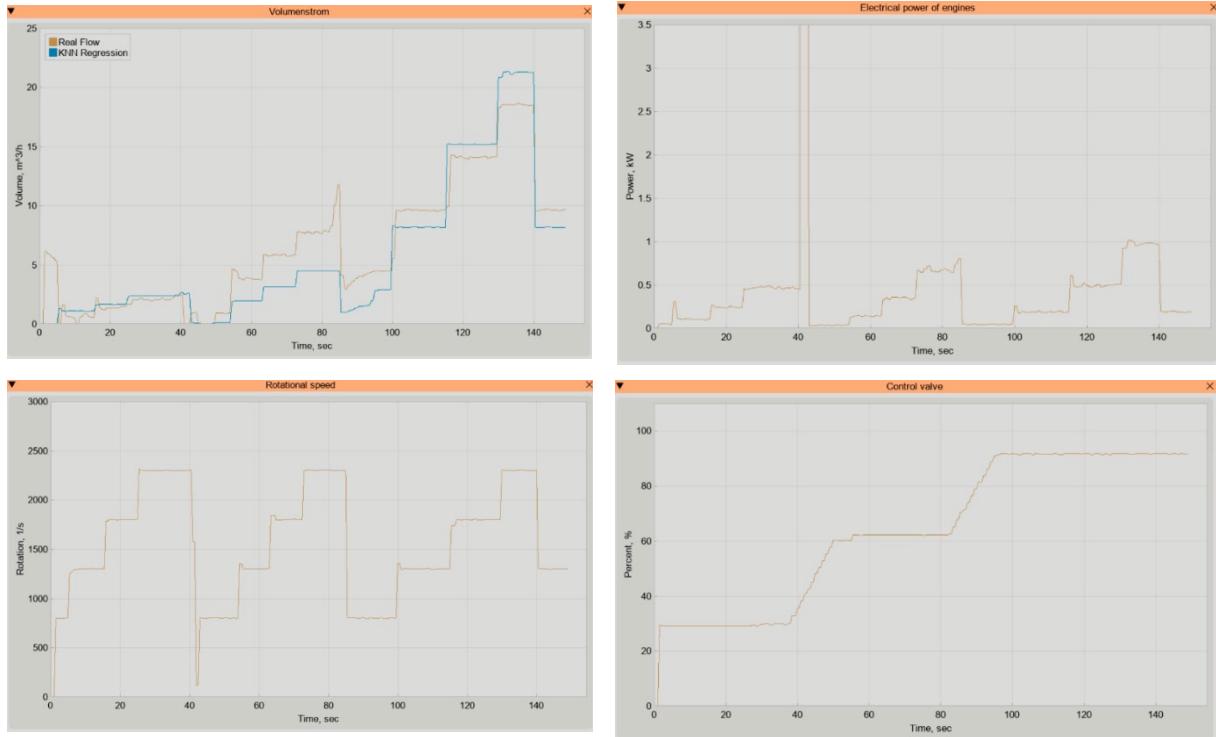


Abbildung 37. Vorhersage des Volumenstromes unter gleichen Bedingungen bei K-nächste-Nachbarn-Algorithmus

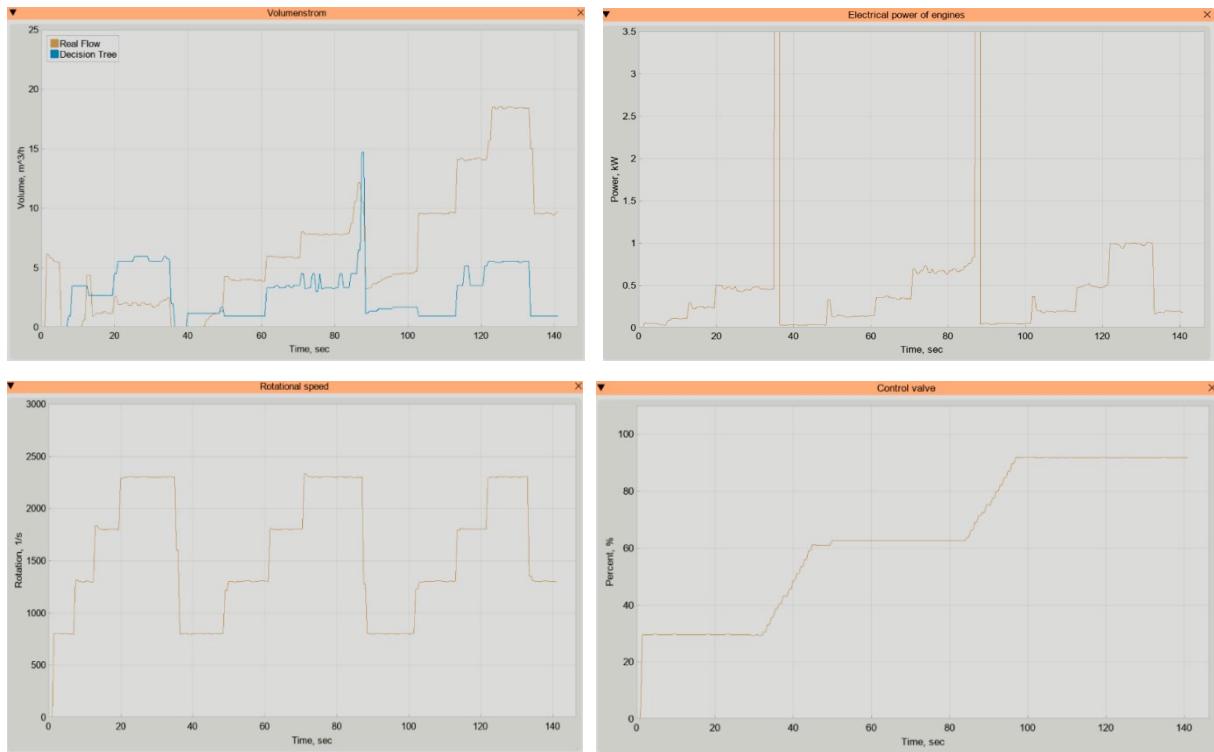
Es muss darauf hingewiesen werden, dass der *K-nächste-Nachbarn-Algorithmus* im Allgemeinen seine Aufgabe erfüllt und den Volumenstrom vorhersagt, obwohl er im Vergleich zu neuronalen Netzen nicht so genau ist.

Bei dem niedrigen Öffnungsstand des Regelventils (bei 30%) ist die Genauigkeit der Volumenstromvorhersage hoch (erste 40 Sekunden auf dem Bild mit dem Volumenstrom). Und es werden kaum Abweichungen vom tatsächlichen Volumenstrom beobachtet. Wenn jedoch der Öffnungsgrad des Regelventils erhöht wird (z.B. bei 60%), nimmt auch die Differenz vom realen Volumenstrom deutlich zu.

Und die durchschnittliche Abweichung des prognostizierten und tatsächlichen Volumenstromes bei *K-nächste-Nachbarn-Algorithmus* beträgt 8.3%. Das ist auch weniger als 10%, deswegen kann gesagt werden, dass der Algorithmus die Aufgabe perfekt bewältigt hat

#### 5.4.4. Decision Tree zum Vorhersagen von Daten unter gleichen Bedingungen

Das Ergebnis des *Decision-Tree-Algorithmus* kann in der folgenden Abbildung dargestellt werden:



*Abbildung 38. Vorhersage des Volumstromes unter gleichen Bedingungen bei Decision-Tree-Algorithmus*

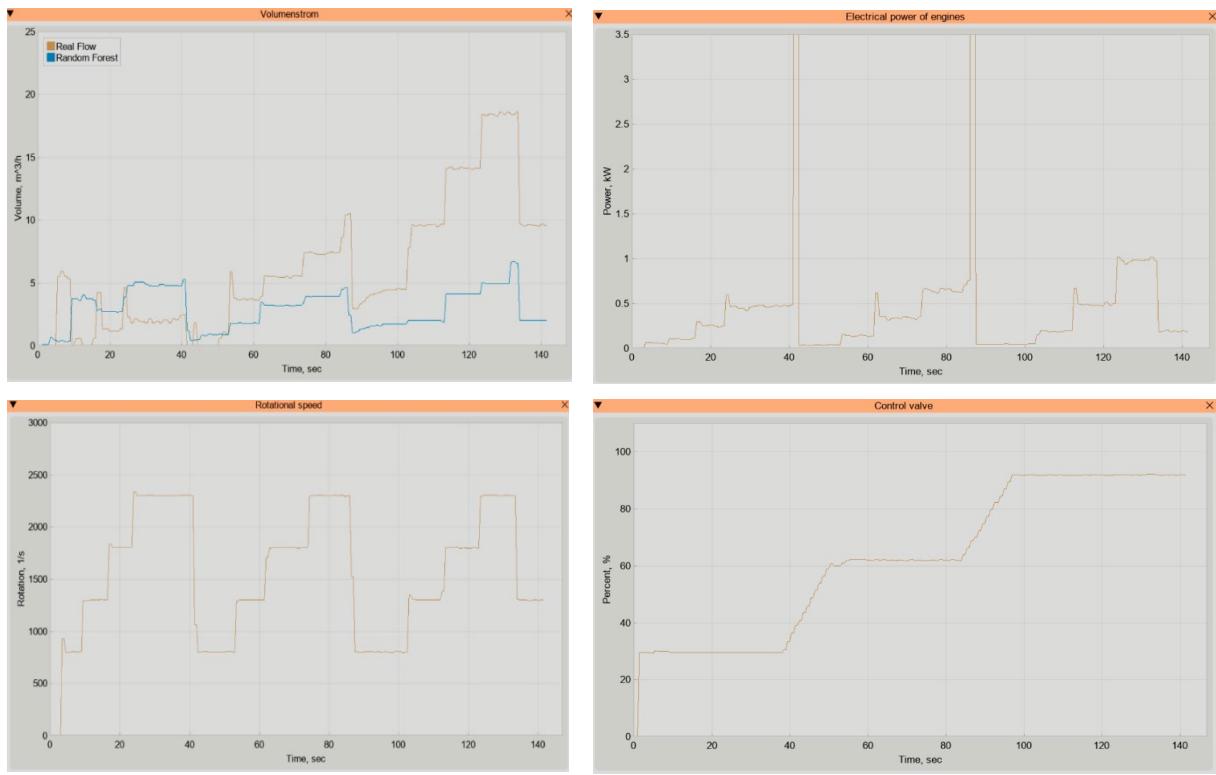
Hier ist alles nicht so eindeutig. Im Allgemeinen versucht der Decision-Tree-Algorithmus, die Form des realen Volumenstromes zu replizieren, aber während des gesamten Programmablaufs war die Genauigkeit der Vorhersage des Volumenstromes gering.

Deswegen ist es schwierig zu vergleichen, bei welchen Werten dieser Algorithmus der Volumenstrom nicht so gut vorhersagt, da er im Allgemeinen während des gesamten Diagramms nicht genau ist. Darüber hinaus sind die vorhergesagten Daten bei Decision-Tree-Algorithmus oft deutlich niedriger, als sie in Wirklichkeit sein sollten.

Und die durchschnittliche Abweichung vom realen Volumenstrom ist hoch und beträgt 24,8%. Das ist viel größer als 10%, daher wird der Decision Tree-Algorithmus möglicherweise nicht weiter verwendet, um die Daten vorherzusagen.

#### 5.4.5. Random Forest zum Vorhersagen von Daten unter gleichen Bedingungen

Die Algorithmen *Random Forest* und *Decision Tree* unterscheiden sich nicht sehr voneinander, aber beide sollten dennoch separat getestet werden. In diesem Kapitel wird der *Random-Forest-Algorithmus* untersucht und getestet, und sein Ergebnis kann unten gezeigt werden:



*Abbildung 39. Vorhersage des Volumstromes unter gleichen Bedingungen bei Random-Forest-Algorithmus*

Wie man sehen kann, hat der *Random-Forest-Algorithmus* (wie auch bei *Decision-Tree-Algorithmus*) in der gesamten Abbildung erhebliche Abweichungen. Deswegen kann dieser Algorithmus nicht analysiert werden und es kann nicht gesagt werden, welche Abhängigkeit zwischen Eingabe und Ausgabe vorhanden ist.

Random Forest (wie auch der Decision Tree) ist nicht für den Pumpversuchstand geeignet, da er sich instabil verhält und große Abweichungen aufweist. Und deswegen sollten diese beiden Algorithmen beiseitegelassen werden.

Und diese Abweichungen vom realen Volumenstrom betragen durchschnittlich 22,7%. Es ist auch größer als 10%, aus diesem Grund kann *Random Forest* nicht verwendet werden, um die Daten an diesem Stand vorherzusagen.

Dieser Algorithmus hat jedoch eine höhere Genauigkeit bei der Vorhersage von Daten (22.7%) als der Decision Tree (24.8%), der im vorherigen Kapitel 5.4.4 behandelt wurde.

#### 5.4.6. Support-Vektor-Maschine (SVM) zum Vorhersagen von Daten unter gleichen Bedingungen

Die Letzte der untersuchten Algorithmen ist der *Support Vektor Maschine*, dessen Testergebnisse sich unten befinden:

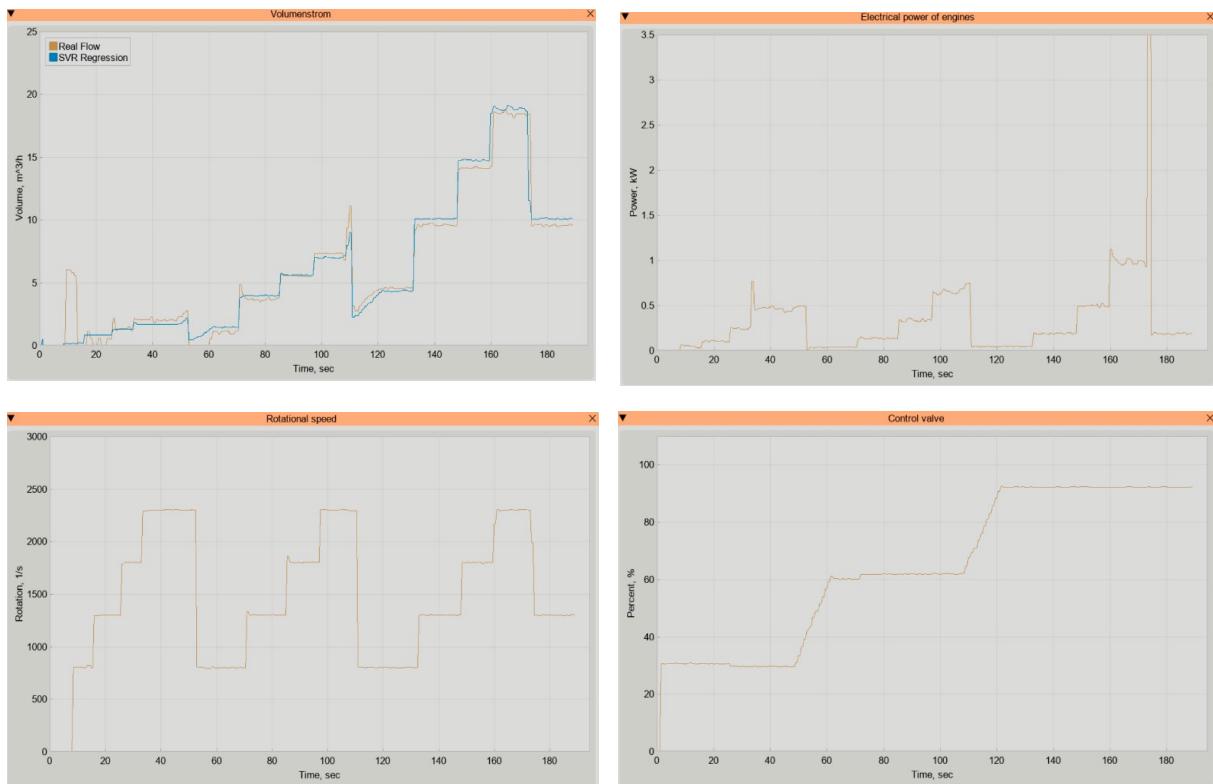


Abbildung 40. Vorhersage des Volumstromes unter gleichen Bedingungen bei SVM-Algorithmus

Was sofort ins Auge fällt, ist, dass der SVM-Algorithmus sehr genau funktioniert, wenn ein gut trainiertes Lernmodell vorhanden ist. Es ist nicht erforderlich, die Ein- und Ausgabedaten zu analysieren, da es klar ist, dass während des gesamten Programmablaufs die Genauigkeit hoch und die Abweichung von den Werten gering waren.

Und es kann außerdem darauf hingewiesen werden, dass der *SVM-Algorithmus* stabil und ohne sichtbare Fehler funktioniert. In diesem Fall ähnelt es den *neuronalen Netzen*, die zuvor untersucht und getestet wurden, da sie auch stabil und ohne kritische Fehler vorhersagen.

Die durchschnittliche Differenz zwischen dem tatsächlichen und dem prognostizierten Volumenstrom bei *SVM-Algorithmus* beträgt 3,1 %. Das ist das beste Ergebnis unter allen betrachteten Algorithmen.

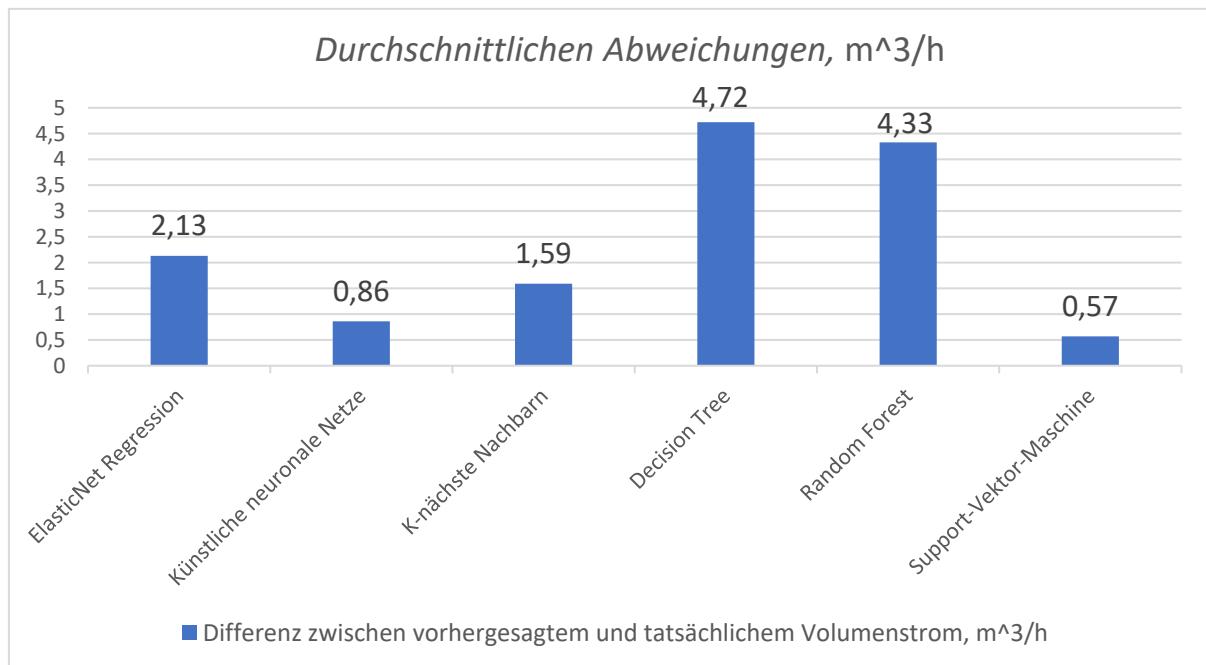
#### 5.4.7. Allgemeiner Vergleich der Genauigkeit von Algorithmen zum Vorhersagen von Daten unter gleichen Bedingungen

Und die folgende Tabelle enthält die Ergebnisse des durchschnittlichen Unterschieds zwischen dem realen und dem vorhergesagten Volumenstrom für verschiedene betrachtete Algorithmen:

<i>Algorithmus</i>	<i>Durchschnittlichen Abweichungen, m^3/h</i>
ElasticNet Regression	2.13
Künstliche neuronale Netze	0.86
K-nächste-Nachbarn	1.59
Decision Tree	4.72
Random Forest	4.33
Support-Vektor-Maschine	0.57

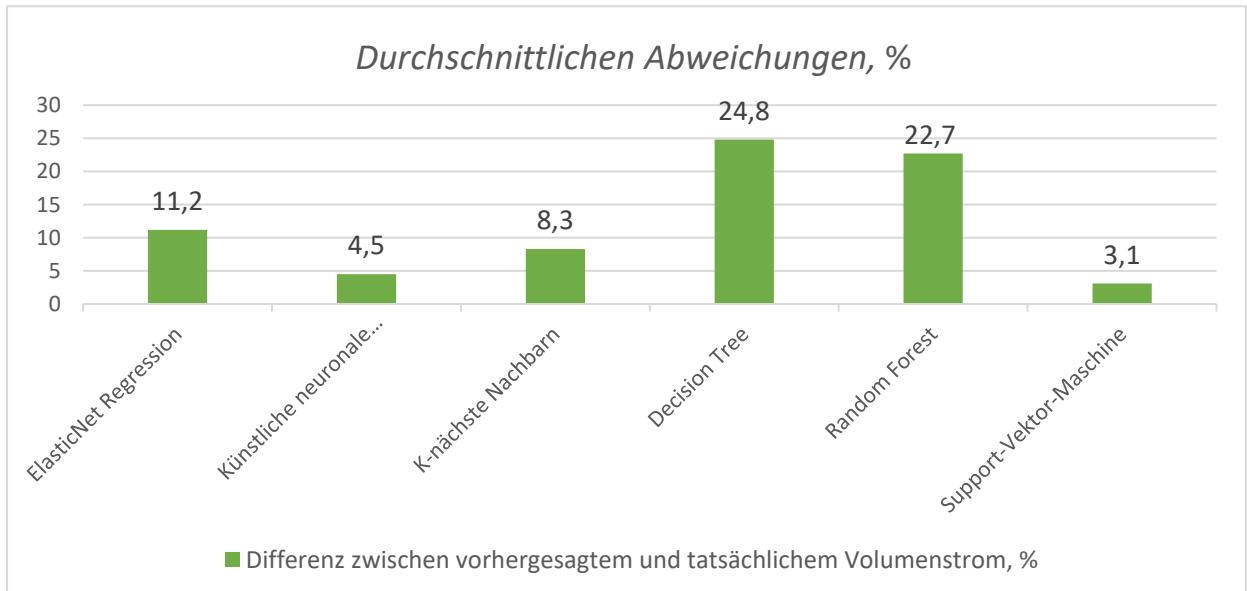
*Tabelle 3. Durchschnittliche Differenz zwischen vorhergesagten und realen Volumenströme zum Vorhersagen von Daten unter gleichen Bedingungen*

Für ein besseres visuelles Verständnis können diese Daten als Diagramm unten dargestellt werden:



*Abbildung 41. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen unter gleichen Bedingungen*

Darüber hinaus ist unten eine Abbildung der durchschnittlichen Differenz der Volumenströme in Prozent dargestellt. So kann es besser und klarer sein, um zu verstehen, welcher der Algorithmen einen geringeren Prozentsatz an Abweichungen von realen Daten aufweist:



*Abbildung 42. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen unter gleichen Bedingungen in Prozent*

Im vorherigen Kapitel 5.3.7 wurde erwähnt, dass je kleiner der Abweichungswert ist, desto besser ist er, weil er dem tatsächlichen Volumenstrom am nächsten kommt. Und hier kann auch geschlossen werden, dass *künstliche neuronale Netze* und der *SVM-Algorithmus* am besten funktionieren und die geringste Abweichung vom realen Wert des Volumenstromes aufweisen. Diese Algorithmen haben einen höheren Vorhersagegrad.

Und wie im vorherigen Kapitel 5.3.7 sollten *Random Forest* und *Decision Tree* Algorithmen beiseitegelassen werden und nicht für die Zwecke der Vorhersage von Daten an dem Pumpversuchstand verwendet werden. Hier gibt es eine Abweichung von mehr als 10% vom tatsächlichen Wert des Volumenstromes. Aus diesem Grund kann man davon ausgehen, dass diese Algorithmen nicht genau genug sind, um die Daten vorherzusagen.

In vorherigen Kapiteln wurden verschiedene maschinelle Lerntechniken unter zufälligen und gleichen Bedingungen getestet. Als allgemeine Schlussfolgerung kann man sagen, dass *künstliche neuronale Netze* und der *SVM-Algorithmus* die besten Werte für Vorhersagen aufweisen. Und sie sind die am besten geeigneten Algorithmen für den Pumpversuchstand, weil die Abweichungen hier niedrig und weniger als 5% sind.

## 5.5. Vergleich verschiedener Parameter neuronaler Netze und deren Einfluss auf die Genauigkeit der Datenvorhersage

Im Gegensatz zu anderen maschinellen Lernalgorithmen sind neuronale Netze schwierig zu konfigurieren und zu programmieren. Sie haben viele verschiedene Parameter, die geändert werden können. Und aus diesem Grund verändert sich auch das Modell der neuronalen Netze, was die Genauigkeit der Datenvorhersagen beeinflusst.

Deswegen werden in diesem Kapitel die 3 grundlegenden Parameter betrachtet, die das Ergebnis der Vorhersage von neuronalen Netzen beeinflussen können. Hier wird ein

Trainingsmodell aus Kapitel 5.2 benutzt, das auch in den vorherigen Kapiteln für die Tests von Algorithmen verwendet wurde.

### 5.5.1. Units

*Units* ist der erste Hauptparameter, der die Layer-Parameter des neuronalen Netzwerks angibt:

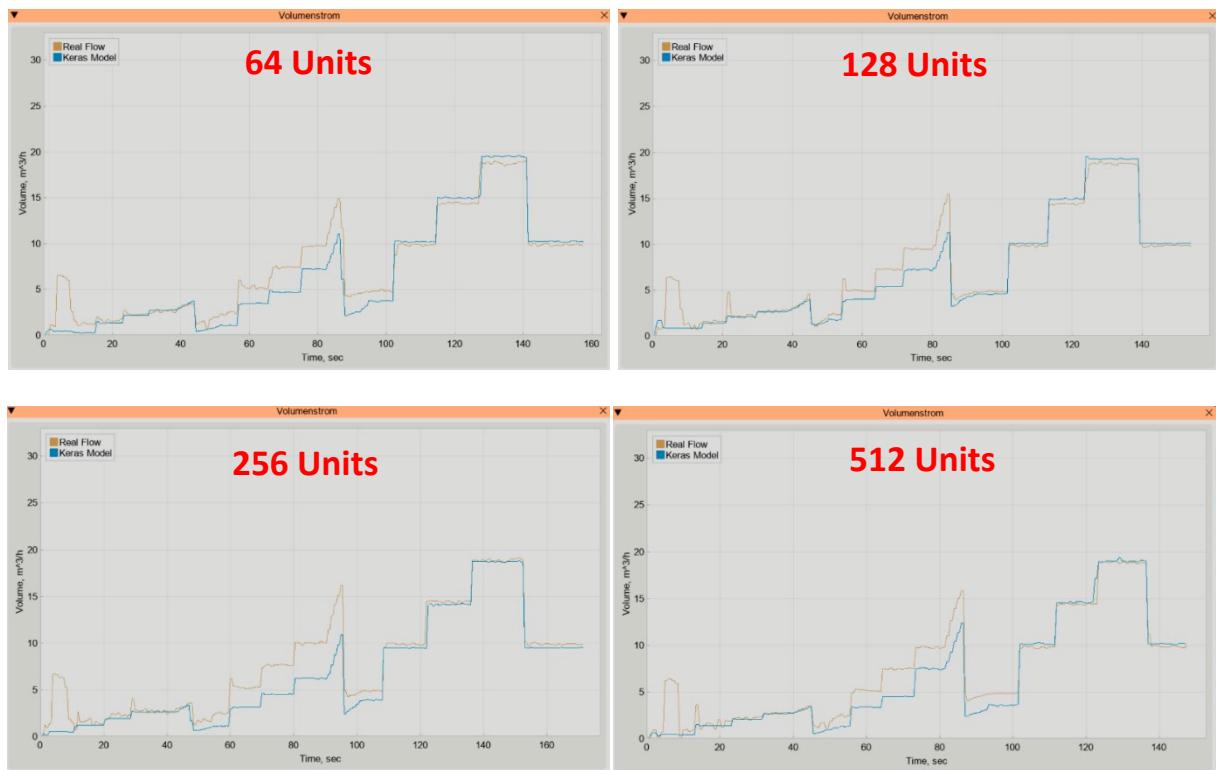
```
keras.layers.LSTM(units, activation=None, name=None)
```

Dies ist das grundlegendste Merkmal aller Parameter, er verwendet eine positive Ganzzahl als Wert und repräsentiert die Ausgabegröße der Ebene. Es ist der Einheitenparameter, der eine wichtige Rolle bei der Größe der Gewichtsmatrix spielt[26].

Um die Auswirkungen dieses Parameters auf die Genauigkeit der Vorhersage zu vergleichen, werden 4 verschiedene neuronale Netze mit den folgenden Units-Parametern untersucht:

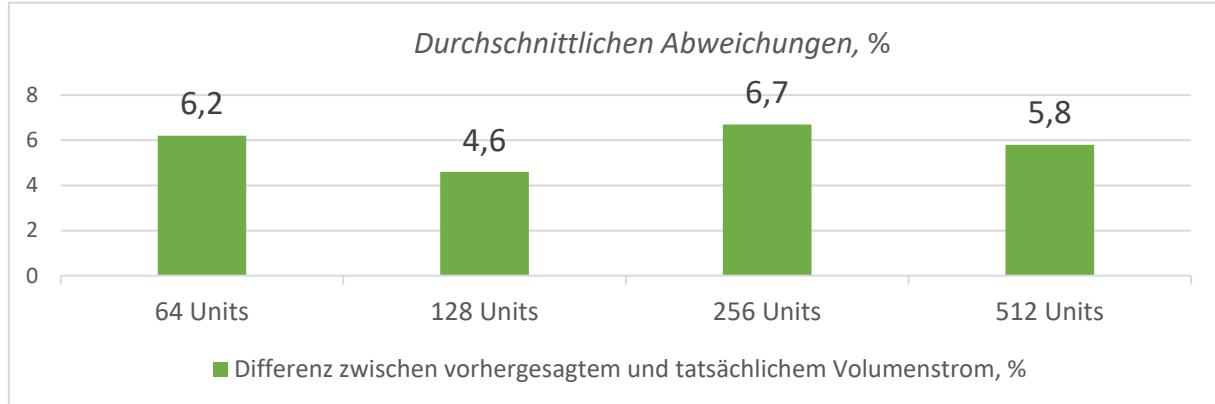
```
x = layers.LSTM(64, activation="relu", name="dense_1")
x = layers.LSTM(128, activation="relu", name="dense_1")
x = layers.LSTM(256, activation="relu", name="dense_1")
x = layers.LSTM(512, activation="relu", name="dense_1")
```

Und unten ist das Ergebnis der Vorhersage des Volumenstromes für Modelle mit **64, 128, 256** und **512** Units entsprechend:



*Abbildung 43. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit 64, 128, 256 und 512 Units entsprechend*

Und für ein besseres visuelles Verständnis können diese Daten als Diagramm unten dargestellt werden:



*Abbildung 44. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit 64, 128, 256 und 512 Units entsprechend in Prozent*

Wie man sehen kann, ist es am besten, ein neuronales Netzmodell mit 128 Units zu verwenden, weil sie die höchste Genauigkeit bei der Vorhersage von dem Volumenstrom hat. Wenn Units (bei 256 und 512) vergrößert werden, macht das Trainingsmodell eine Umschulung, sodass die durchschnittlichen Abweichungen der vorhergesagten Daten zunehmen und die Genauigkeit sinkt. Außerdem zeigt die Abbildung 43 bei 100 Sekunden deutlich, dass das Trainingsmodell mit 128 Units besser funktioniert und den Volumenstrom genauer vorhersagt.

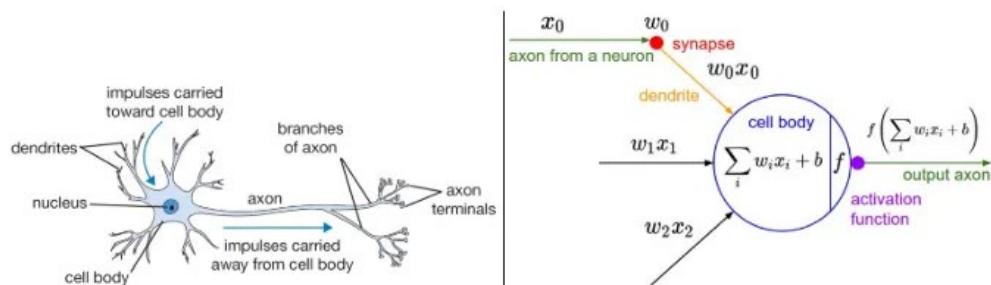
### 5.5.2. Aktivierung

*Aktivierung* ist der zweite Hauptparameter, der die Layer-Parameter eines neuronalen Netzwerks angibt:

```
keras.layers.LSTM(units, activation=None, name=None)
```

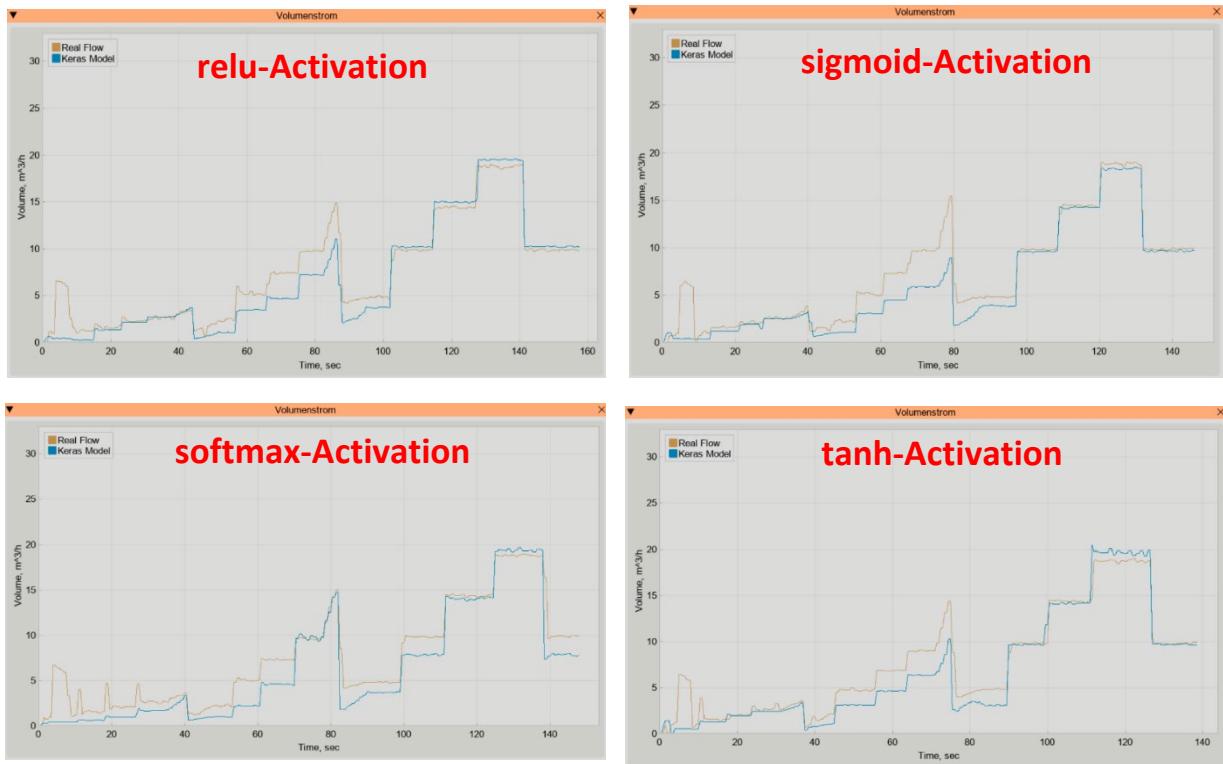
Die Idee der Aktivierungsfunktionen leitet sich aus dem neuronenbasierten Modell des menschlichen Gehirns ab. Gehirne bestehen aus einem komplexen Netzwerk biologischer Neuronen, in denen ein Neuron basierend auf bestimmten Eingaben des vorherigen Neurons aktiviert wird[27].

Im künstlichen neuronalen Netzwerk haben wir künstliche Neuronen, die nichts anderes sind als eine mathematische Einheit, die aus der Aktivierungsfunktion besteht [27]. Es löst auch die Ausgabe basierend auf bestimmten Eingaben aus, die vom vorherigen Neuron empfangen wurden. Das folgende Diagramm erklärt die Analogie zwischen dem biologischen und dem künstlichen Neuron.



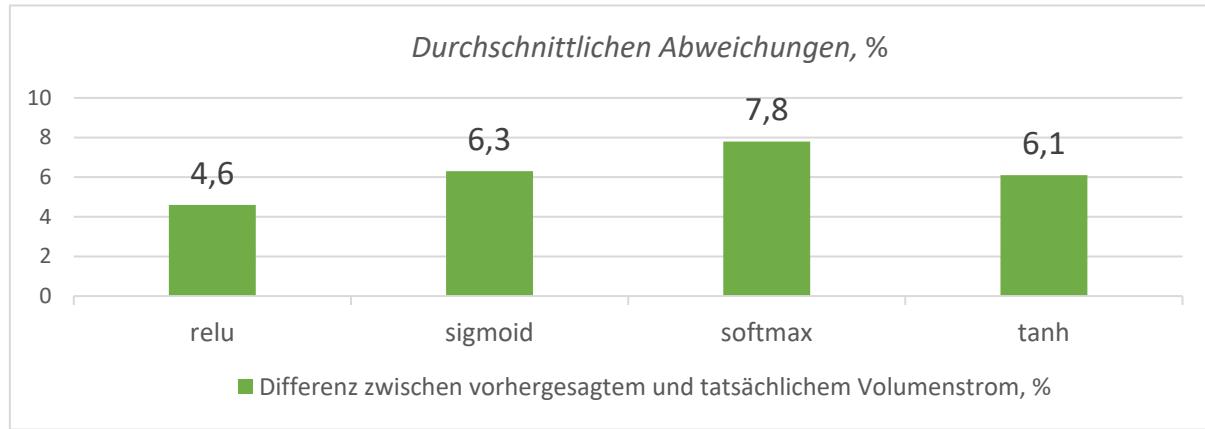
*Abbildung 45. Eine Karikaturzeichnung eines biologischen Neurons (links) und seines mathematischen Modells (rechts) aus [27]*

Um die Auswirkungen dieses Parameters auf die Genauigkeit der Vorhersage zu vergleichen, werden 4 verschiedene neuronale Netze mit den folgenden Aktivationsparametern untersucht: *relu*, *sigmoid*, *softmax* und *tanh*. Unten kann man das Ergebnis der Vorhersage des Volumenstromes für Modelle mit diesen verschiedenen Aktivations entsprechend untersuchen:



*Abbildung 46. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit *relu*-, *sigmoid*-, *softmax*- und *tanh*-Activation entsprechend*

Unten kann ein Diagramm für jede der Aktivierungsmethoden dargestellt werden:



*Abbildung 47. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit *relu*-, *sigmoid*-, *softmax*- und *tanh*-Activation in Prozent*

Daraus kann man schließen, dass die Relu-Aktivierung die genaueste ist, um die Daten vorherzusagen. Andere Arten von Aktivierungen, die in Trainingsmodellen benutzt werden, können auch den Volumenstrom gut vorhersagen. Aber wenn wir mit dem Pumpversuchstand arbeiten, ist es besser, Relu-Aktivierung in neuronalen Netzwerkparametern zu verwenden.

### 5.5.3. Optimizer

*Optimierer* ist eines der beiden Argumente, die zum Kompilieren eines Keras-Modells erforderlich sind:

```
model.compile(loss="mean_squared_error", optimizer='rmsprop')
```

Jetzt ist es für uns wichtiger, den zweiten *Optimizer-Parameter* zu berücksichtigen, weil der *Loss-Parameter* die Kompilierung des Modells nicht so stark beeinflusst.

Und beim Training eines neuronalen Netzes werden seine Gewichte zunächst zufällig initialisiert und dann in jeder Epoche so aktualisiert, dass sie die Gesamtgenauigkeit des Netzwerks erhöhen. In jeder Epoche wird die Ausgabe der Trainingsdaten mithilfe der Verlustfunktion mit tatsächlichen Daten verglichen, um den Fehler zu berechnen. Und dann wird das Gewicht entsprechend aktualisiert[28].

Aber woher wissen wir, wie man das Gewicht so aktualisiert, dass es die Genauigkeit erhöht? Dies ist im Wesentlichen ein *Optimierungsproblem*, bei dem das Ziel darin besteht, die Verlustfunktion zu optimieren und ideale Gewichte zu erhalten[28].

Um die Auswirkungen verschiedener Optimierer auf die Genauigkeit zu vergleichen, werden 4 neuronale Netze mit den folgenden Optimierer-Parametern untersucht: *RMSprop*, *nadam*, *adam* und *Ftrl*. Unten kann man das Ergebnis der Vorhersage des Volumenstromes für Modelle mit diesen Optimierern entsprechend analysieren:

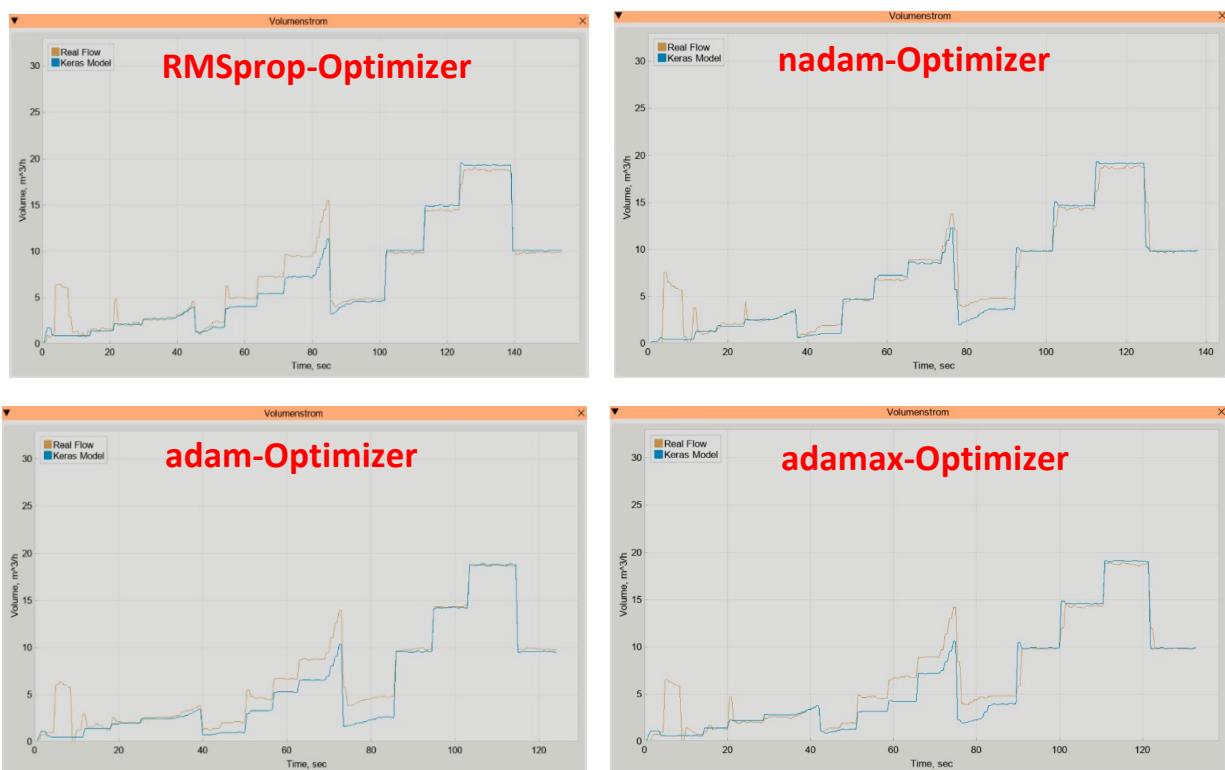


Abbildung 48. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit RMSprop-, nadam, adam- und Ftrl-Optimizer entsprechend

Das Ergebnis der Genauigkeit der Vorhersage kann auch als Diagramm dargestellt werden:

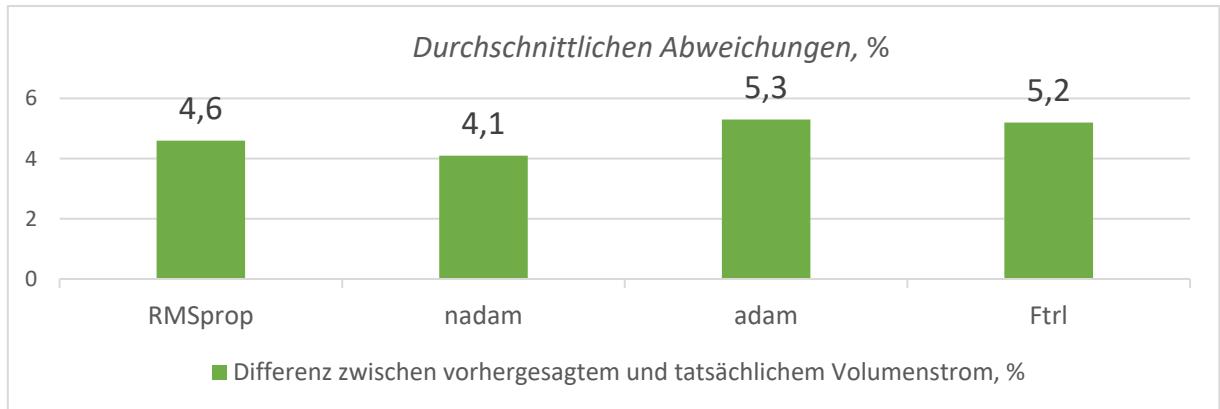


Abbildung 49. Vergleich von Abweichungen der Datenvorhersage für verschiedene Algorithmen mit RMSprop-, adadelta-, nadam- und Ftrl-Optimizer in Prozent

Und wie man sehen kann, zeigt der *nadam-Optimierer* von allen Modelloptimierern die besten Ergebnisse mit nur 4,1 % Abweichung von den tatsächlichen Volumenstrom.

Wenn man sich die Abbildung 48 mit einem Zeitraum von 50 bis 70 Sekunden ansieht, ist es klar, dass der *nadam-Optimierer* ziemlich genau funktioniert. In diesem Zeitbereich weichen die Werte fast nicht von den realen ab. Während andere Algorithmen an dieser Stelle immer Abweichungen hatten und Vorhersagefehler machten. Deswegen ist der *nadam-Optimierer* unter allen Modelloptimierern das Beste und sollte an dem Pumpversuchstand verwendet werden, um Daten vorherzusagen.

Auf diese Weise wurden in diesem Kapitel die grundlegenden Merkmale behandelt, die beim Erstellen neuronaler Netze angepasst werden können. Für eine genauere Vorhersage sollte ein Modell mit den folgenden Parametern verwendet werden: *128 Units, relu-Aktivierung* und *nadam-Optimierer*. In diesem Fall sollte die Abweichung vom tatsächlichen Volumenstrom minimal sein.

## 5.6. Praktische Vor- und Nachteile der betrachteten Algorithmen bei der Verwendung für den Pumpversuchstand

Um alle Angaben über die betrachteten Algorithmen zusammenzufassen, zeigt dieses Kapitel eine Tabelle mit einer detaillierten Beschreibung der Vor- und Nachteile jedes der Algorithmen. Diese Information wurde durch die *praktische* Anwendung des Pumpversuchstandes beobachtet. Das sind keine aus der Literatur entnommenen Vor- und Nachteile, sie werden im nächsten Kapitel 5.6 diskutiert.

Zur besseren Darstellung der Tabelle befindet sie sich auf der nächsten Seite:

	Vorteile (+)	Nachteile (-)
Elastic Regression und andere lineare Regressionen	<ul style="list-style-type: none"> <li>+ Die schnellste Trainingszeit und Vorhersagezeit unter allen Algorithmen;</li> <li>+ Der Algorithmus ist einfach und benötigt daher nicht viel Speicherplatz;</li> <li>+ Die durchschnittliche Differenz zwischen vorhergesagten und realen Volumenströme beträgt 11-13%. Daher kann man daraus schließen, dass der Algorithmus die Daten ziemlich genau vorhersagt;</li> </ul>	<ul style="list-style-type: none"> <li>- Funktioniert bei schnellen Volumenstromänderungen nicht gut;</li> <li>- Im Allgemeinen funktioniert der Algorithmus nicht so genau, wenn er mit neuronalen Netzen oder einem SVM-Algorithmus verglichen wird. Die durchschnittliche Abweichung vom tatsächlichen Volumenstrom beträgt mehr als 10%;</li> <li>- Der Algorithmus funktioniert am Pumpversuchstand nicht so genau, wenn die Werte von <i>Drehzahl</i> und elektrische <i>Leistung</i> des Motors zu hoch sind;</li> </ul>
Künstliches neuronales Netz (TensorFlow mit Keras-Modell)	<ul style="list-style-type: none"> <li>+ Das trainierte Modell kann separat erstellt und später verwendet werden. Das beschleunigt den Datenvorhersageprozess;</li> <li>+ Das Keras-Modell funktioniert sehr gut für die Vorhersage von Daten. Die durchschnittliche Abweichung vom tatsächlichen Volumenstrom beträgt nicht mehr als 7-8%;</li> <li>+ Für eine gute Vorhersage reichen nur 3 Eingaben aus, wie in unserem Beispiel;</li> <li>+ Der vorhergesagte Volumenstrom ändert sich fast sofort. Kann in den Diagrammen gesehen werden, wie sich der vorhergesagte Wert des Volumenstromes schnell anpasst;</li> <li>+ Der Algorithmus ist während der gesamten Arbeit stabil und zeigt im Allgemeinen ungefähr das gleiche Vorhersagenergebnis an. Obwohl es gibt einige Abweichungen;</li> </ul>	<ul style="list-style-type: none"> <li>- Wenn viele Eingaben zum Trainieren des Modells vorhanden sind, kann es bis 30 Min. dauern, bis das Trainingsmodell gut trainiert ist. Der Trainingsprozess ist sehr lang;</li> <li>- Man muss einen Kompromiss zwischen der Einfachheit und der Schwierigkeit des Modelllernens finden. Sonst funktioniert das Trainingsmodell nicht richtig oder nicht zu genau (mehr dazu in Kapitel 5.3.2.);</li> <li>- Neuronale Netze haben viele Parameter, die ausgewählt werden müssen. Daher ist es nicht immer klar, wie man sie wählt. Oft werden viele Merkmale zufällig ausgewählt, um das Modell zu testen. Und dann werden diese Parameter angepasst;</li> </ul>
K-nächste Nachbarn (KNNA)	<ul style="list-style-type: none"> <li>+ Ziemlich schnelle Trainingszeit und Vorhersagezeit, die durchschnittlich 0,6 Sek. bzw. 0,11 Sek. dauern;</li> <li>+ Kann vorhergesagte Werte schnell ändern, anpassen und vorhersagen;</li> <li>+ Es genügt, 2 Parameter zu konfigurieren, damit der Algorithmus funktioniert: einen K-Wert und eine Entfernungsfunktion;</li> <li>+ Die Genauigkeit der Datenvorhersage ist sogar höher als bei der ElasticNet-Regression und erreicht ungefähr 9-10%;</li> </ul>	<ul style="list-style-type: none"> <li>- Manchmal gibt es schlechte Vorhersage, wenn die Eingabedaten nicht mit dem Trainingsmodell und den Trainingseingaben übereinstimmen. Dies ist in Kapitel 5.3.3 deutlich sichtbar, als KNNA unter realen Bedingungen getestet wurde;</li> <li>- Es gibt auch einige scharfe Spitzenwerte des vorhergesagten Volumenstromes, die aufgrund der Funktionsweise des Algorithmus auftreten;</li> </ul>

		<ul style="list-style-type: none"> <li>- Funktioniert bei schnellen Volumenstromänderungen nicht so gut;</li> </ul>
Decision Tree	<ul style="list-style-type: none"> <li>+ Schnelle Trainingszeit und Vorhersagezeit, die durchschnittlich 0,5 Sek. bzw. 0,013 Sek. dauern;</li> <li>+ Dieser Algorithmus ist einfach zu verwenden und es nimmt wenig Platz im Speicher;</li> <li>+ Nicht für Echtzeit-Anwendungen geeignet, funktioniert aber gut für eine einzelne Datenvorhersage, dann erhöht sich die Genauigkeit. Zum Beispiel, wenn man einmal einen Wert vorhersagen muss;</li> </ul>	<ul style="list-style-type: none"> <li>- Bei schneller Änderung des Volumstromes zu starken Sprüngen der Vorhersagedaten führt. Und manchmal werden Daten deutlich niedriger vorhergesagt, als sie in Wirklichkeit sein sollten;</li> <li>- Der Algorithmus sagt die Daten am Pumpversuchstand nicht so genau vorher. Und manchmal kann <i>Decision Tree</i> hohe Werte vorhersagen, die nicht der Realität entsprachen;</li> <li>- Der Algorithmus funktioniert in Echtzeit am Pumpversuchstand nicht so gut und kann sich nicht schnell an die Eingaben anpassen. Deswegen beträgt die Genauigkeit der Vorhersage des Volumenstromes durchschnittlich 23-25%, was unter anderen Algorithmen das schlechteste Ergebnis ist;</li> <li>- Manchmal gibt es eine Umschulung des Algorithmus, wodurch die Genauigkeit der Datenvorhersagen verloren geht;</li> </ul>
Random Forest (basierend auf den gleichen Algorithmen wie bei Decision Tree)	<ul style="list-style-type: none"> <li>+ Hier gibt es auch schnelle Vorhersagezeit und es dauert durchschnittlich 0,02 Sekunden;</li> <li>+ <i>Random Forest</i> funktioniert besser als der <i>Decision-Tree-Algorithmus</i>. Er ist eine verbesserte Version des Letzteren. Deswegen ist seine Genauigkeit um etwa 2-3% besser als bei <i>Decision Tree</i>;</li> <li>+ <i>Random Forest</i> funktioniert gut für eine einzelne Datenvorhersage;</li> <li>+ Normalerweise gibt es keine Umschulung des Algorithmus. Und <i>Random Forest</i> ist einfach zu bedienen und er nimmt nicht viel Speicherplatz in dem Speicher;</li> </ul>	<ul style="list-style-type: none"> <li>- <i>Random Forest</i> funktioniert mit echten Daten am Pumpversuchstand nicht so korrekt, obwohl er versucht, den tatsächlichen Volumenstrom vorherzusagen;</li> <li>- <i>Random Forest</i> funktioniert in Echtzeit am Pumpversuchstand nicht immer gut oder korrekt und kann sich nicht schnell an die Eingaben anpassen. Auf diesem Grund erreicht die Genauigkeit der Datenvorhersagen 21-22%, was ein ungenaues Ergebnis ist;</li> <li>- Manchmal werden Daten deutlich niedriger vorhergesagt, als sie in Wirklichkeit sein sollten. Das gleiche Problem wurde beim <i>Decision-Tree-Algorithmus</i> beobachtet;</li> </ul>
Support-Vektor-Maschine (SVM)	<ul style="list-style-type: none"> <li>+ Die Genauigkeit des Algorithmus ist der größte unter allen anderen Algorithmen. Hier ist der Unterschied zwischen dem vorhergesagten und dem realen Volumenstrom am wenigsten und beträgt etwa 3-5%;</li> </ul>	<ul style="list-style-type: none"> <li>- Die Vorhersagezeit kann bis zu 3-5 Sekunden betragen, was deutlich höher ist als bei anderen Algorithmen.</li> </ul>

	<ul style="list-style-type: none"> <li>+ SVM-Algorithmus kann sich schnell und fast sofort an Änderungen der Eingabewerte anpassen (wie bei neuronalen Netzen);</li> <li>+ Bei Verwendung dieses Algorithmus gibt es keine plötzlichen Wertesprünge.</li> <li>+ Hier auch für eine gute Vorhersage reichen es nur 3 Eingaben aus.</li> <li>+ Wie bei neuronalen Netzen ist der SVM-Algorithmus während der gesamten Arbeit stabil und zeigt im Allgemeinen ungefähr das gleiche Vorhersagenergebnis an. Das ist gut für den realen Einsatz am Pumpversuchstand.</li> <li>+ Es ist nicht notwendig, viele Parameter wie bei neuronalen Netzen zu konfigurieren. Der Algorithmus ist einfach zu bedienen;</li> </ul>	<ul style="list-style-type: none"> <li>- Sehr lange Modelltrainingszeit und bei großen Datensätzen kann es 1-2 Minuten dauern. Aber es lohnt sich auch zu bemerken, dass neuronale Netze das gleiche Problem haben;</li> <li>- Der Algorithmus reagiert sehr empfindlich auf unnötige Daten und Geräusche. Daher kann die Genauigkeit der Vorhersage des Volumenstromes bei Vorhandensein von Fremddaten erheblich reduziert werden;</li> </ul>
--	--	---

*Tabelle 4. Praktische Vor- und Nachteile der betrachteten Algorithmen*

## 5.7. Aus der Literatur entnommenen Vor- und Nachteile der betrachteten Algorithmen bei der Verwendung für den Pumpversuchstand

Dieses Kapitel beschreibt die Vor- und Nachteile der betrachteten Algorithmen, die aus der Literatur stammen, wenn sie für den Pumpversuchstand verwendet werden. Deswegen sind sie eher theoretisch als praktisch. Die Tabelle befindet sich unten:

	Vorteile (+)	Nachteile (-)
Elastic Regression und andere lineare Regressionen	<ul style="list-style-type: none"> <li>+ Einfaches mathematisches Prinzip, bedienungsfreundlicher Algorithmus [18];</li> <li>+ Praktisch außer Konkurrenz, wenn es viele Parameter gibt (von Hunderttausenden oder mehr) [19];</li> <li>+ Schnelle und einfache Modellierung, insbesondere wenn das Trainingsmodell über keine großen Datenmengen verfügt [18];</li> <li>+ Logistische Regression kann auch Wahrscheinlichkeiten für die Zuordnung zu verschiedenen Klassen ergeben (dies wird zum Beispiel im Kredit-Scoring sehr geschätzt) [19];</li> </ul>	<ul style="list-style-type: none"> <li>– Funktioniert nicht gut in Aufgaben, bei denen die Abhängigkeit von Antworten von Merkmalen komplex und nicht linear ist [19];</li> <li>– Bei nichtlinearen Daten ist eine lineare Regression schwer zu konstruieren. Es muss Informationen über die Datenstruktur und die Beziehung zwischen den Variablen geben [18];</li> <li>– Lineare und andere Arten von Regression sind nicht so effizient, wenn es um sehr komplexe Daten geht [18];</li> </ul>
Künstliches neuronales Netz (TensorFlow mit Keras-Modell)	<ul style="list-style-type: none"> <li>+ Da neuronale Netze vielschichtig sein können, sind sie bei der Modellierung komplexer nichtlinearer Beziehungen wirksam [18];</li> <li>+ Wir müssen uns keine Gedanken über die Datenstruktur in neuronalen Netzen machen. Sie sind sehr flexibel, um fast jede Art von Merkmalvariablen zu lernen [18];</li> <li>+ Die Forschung zeigt ständig: Je mehr Lerndaten ein neuronales Netz hat, desto produktiver wird es [18];</li> <li>+ Es besteht die Möglichkeit, bei Verlust einzelner Elemente funktionsfähig zu bleiben. Die Beschädigung bestimmter Komponenten hindert neuronale Netze nicht daran, ein korrektes Ergebnis zu erzielen [25].</li> </ul>	<ul style="list-style-type: none"> <li>– Aufgrund der Komplexität des Keras-Modells ist es nicht einfach zu verstehen und zu implementieren [18];</li> <li>– Neuronale Netze benötigen enorme Datenmengen, um eine hohe Leistung zu erzielen. Und infolgedessen sind sie anderen ML-Algorithmen unterlegen, wenn nur wenige Daten vorhanden sind [18];</li> <li>– Neuronale Netze erfordern eine sorgfältige Einstellung der Parameter und der Lerngeschwindigkeit [18];</li> <li>– Es ist nicht vollständig klar, wie das neuronale Netzwerk funktioniert und welche Daten zur Entscheidungsfindung verwendet werden [25];</li> </ul>
K-nächste-Nachbarn (KNNA)	<ul style="list-style-type: none"> <li>+ Der Algorithmus ist einfach und leicht zu implementieren, er ist nicht empfindlich auf Ausreißer in Daten [20];</li> </ul>	<ul style="list-style-type: none"> <li>– Der K-nächste-Nachbarn-Algorithmus funktioniert nicht gut mit großen Daten, da es für den Algorithmus bei einer vielen Anzahl von</li> </ul>

	<ul style="list-style-type: none"> <li>+ Es ist nicht notwendig, ein Modell zu erstellen, mehrere Parameter zu konfigurieren oder zusätzliche Annahmen zu treffen [20];</li> <li>+ Dieser Algorithmus ist faul zu lernen, deswegen erfordert er fast kein Training, bevor er in Echtzeit vorhersagt. Dies macht den K-nächste-Nachbarn-Algorithmus viel schneller als andere Algorithmen, die Lernen brauchen, z. B. SVM, Keras-Modell usw. [20];</li> <li>+ Die Implementierung von diesem Algorithmus erfordert nur zwei Parameter: einen K-Wert und eine Entfernungsfunktion (z. B. euklidisch oder Manhattan usw.) [20];</li> <li>+ Der Algorithmus ist universell. Es kann für beide Arten von Aufgaben verwendet werden: Klassifikation und Regression [21];</li> </ul>	<ul style="list-style-type: none"> <li>Messungen schwierig wird, die Entfernung in jeder Dimension zu berechnen [21];</li> <li>- Wenn ein Datensatz viele Parameter enthält, ist es schwierig, geeignete Gewichte zu finden und festzustellen, welche Merkmale für die Klassifizierung/Regression nicht wichtig sind [21];</li> <li>- Es gibt keine theoretische Grundlage für die Auswahl von Nachbarn - nur eine Suche nach dieser bestimmten Anzahl. Bei wenigen Nachbarn ist das Verfahren empfindlich gegenüber Ausreißern, d. h. es neigt zur Umschulung [21];</li> </ul>
Decision Tree	<ul style="list-style-type: none"> <li>+ Einfacher Algorithmus zu verwenden und nimmt wenig Platz in Anspruch [21];</li> <li>+ Gut geeignet für das Studium komplexer nichtlinearer Beziehungen [18];</li> <li>+ Das endgültige Modell kann mithilfe eines «Baumdiagramms» ordentlich visualisiert und interpretiert werden [21];</li> <li>+ Die Schaffung von klaren Klassifizierungsregeln, die dem Menschen klar sind, zum Beispiel: "Wenn das Alter &lt; 25 ist und das Interesse an Motorrädern besteht, dann verweigern Sie den Kredit." [21];</li> <li>+ Unterstützung für numerische und kategorische Merkmale [21];</li> <li>+ Wenige Modellparameter [21];</li> </ul>	<ul style="list-style-type: none"> <li>- Entscheidungsbäume können anfällig für ernsthafte Umschulungen sein. Ein abgeschlossenes Modell des Entscheidungsbaums kann übermäßig komplex sein und eine unnötige Struktur enthalten [18];</li> <li>- Bäume reagieren sehr empfindlich auf Geräusche in den Eingaben, und das gesamte Modell kann sich grundlegend ändern, wenn sich die Lernprobe etwas verändert. Daher können sich auch die Klassifizierungsregeln stark ändern, was die Interpretationsfähigkeit des Systems beeinträchtigt [21];</li> <li>- Umschulung des Algorithmus ist möglich. Aber es kann durch Festlegen einer Mindestanzahl von Proben in jedem Teil des Baumes oder Definieren einer maximalen Tiefe vermieden werden [21];</li> <li>- Kleine Änderungen an den Daten können den konstruierten Entscheidungsbäumen erheblich verändern [21];</li> </ul>
Random Forest (basierend auf den gleichen Algorithmen wie bei Decision Tree)	<ul style="list-style-type: none"> <li>+ Gut geeignet für das Studium komplexer nichtlinearer Beziehungen [22];</li> <li>+ Reduziertes Risiko der <i>Umschulung</i>: Decision Tree (zuvor betrachteter Algorithmus) besteht die Gefahr der Umschulung, da sie dazu neigen, alle Stichproben innerhalb der Trainingsdaten eng anzupassen. Wenn jedoch eine robuste Anzahl von Entscheidungsbäumen in einer zufälligen Gesamtstruktur vorhanden ist, macht der Klassifikator keine</li> </ul>	<ul style="list-style-type: none"> <li>- Da zufällige Gesamtstrukturen größere Datensätze verarbeiten, benötigen sie mehr Ressourcen, um diese Daten zu speichern [22];</li> <li>- Es ist schwierig, das endgültige Modell zu visualisieren. Und die Erstellung des Modells kann zeitaufwendig sein, wenn nicht genügend Rechenleistung zur Verfügung steht oder wenn der Datensatz extrem groß ist [22];</li> </ul>

	<p>Umschulung, weil die Mittelung unkorrelierter Bäume die Gesamtvarianz und den Vorhersagefehler verringert [22];</p> <ul style="list-style-type: none"> <li>+ Bietet <i>Flexibilität</i>: Da Random Forest sowohl Regressions- als auch Klassifizierungsaufgaben mit hoher Genauigkeit bewältigen kann, ist es eine beliebte Methode unter Datenwissenschaftlern. Dieser Algorithmus ist auch ein effektives Werkzeug zum Schätzen fehlender Werte, da er die Genauigkeit beibehält, wenn ein Teil der Daten fehlt [22];</li> <li>+ Random Forest macht es einfach, die Wichtigkeit von Variablen oder ihren Beitrag zum Modell zu bewerten [22];</li> </ul>	<ul style="list-style-type: none"> <li>- Komplexer: Die Vorhersage eines einzelnen Entscheidungsbaums ist im Vergleich zu einer zufälligen Gesamtstruktur einfacher zu interpretieren [22];</li> <li>- Um eine höhere Leistung zu erzielen, werden der Speicher und die Zeit, in der große zufällige Wälder verwendet werden, gespendet [18];</li> </ul>
Support-Vektor-Maschine (SVM)	<ul style="list-style-type: none"> <li>+ Sehr effiziente Methode zur Klassifizierung von Objekten [23];</li> <li>+ Der Algorithmus funktioniert gut mit großen Merkmalen und kleinen Datenmengen [10];</li> <li>+ Auch für große Merkmalsräume mit vielen Trainingsdaten und Trainingsmustern anwendbar [23];</li> <li>+ SVM verwendet eine Teilmenge der Lernpunkte in der Entscheidungsfindungsfunktion, sodass sie Speicher effizient nutzt [23];</li> <li>+ Der SVM-Algorithmus unterstützt der linearen und nicht-linearen Klassifizierung [6];</li> <li>+ Relativ geringer Aufwand zur Implementierung eines SVM-Modells [23];</li> <li>+ Geometrisch sehr anschauliche Arbeitsweise des Algorithmus [23];</li> </ul>	<ul style="list-style-type: none"> <li>- Bei größeren Datensätzen ist die Verarbeitung sehr zeitaufwendig [24];</li> <li>- Nicht resistent gegen Rauschen: Ausreißer in den Trainingsdaten werden störend und wirken sich direkt auf die Konstruktion der trennenden Hyperebene aus [24];</li> <li>- SVM funktioniert nicht gut mit überlappenden Klassen von Daten [24];</li> <li>- Es kann schwierig sein, Hyperparameter für den SVM-Algorithmus auszuwählen, die eine ausreichende Leistung ermöglichen [24];</li> </ul>

*Tabelle 5. Aus der Literatur entnommenen Vor- und Nachteile der betrachteten Algorithmen*

Auf diese Weise hat jeder Algorithmus seine Vor- und Nachteile und kann für seine eigenen Zwecke verwendet werden. Das *Keras-Modell* (künstliche neuronale Netze) zum Beispiel funktioniert gut und sind flexibel für verschiedene Eingaben. Die *ElasticNet-Regression* ist schnell und einfach, sodass sie dort benutzt werden kann, wo wenig Speicher vorhanden ist. Der *SVM-Algorithmus* kann als Alternative zu den künstlichen neuronalen Netzen verwendet werden, da sie ähnliche Ergebnisse aufweisen. *Decision Tree* und *Random Forest* sollten vielleicht ausgeschlossen werden, um Werte in Echtzeit vorherzusagen, weil diese Algorithmen nicht so genau am Pumpversuchstand funktionieren.

Es sollte jedoch beachtet werden, dass sich für diese Masterarbeit und für den Pumpversuchstand der *SVM-Algorithmus* und *künstliche neuronale Netze* am besten bewährt haben. Diese maschinellen Lerntechniken können die Daten stabil, genauer und korrekt vorhersagen.

## 6. User Interface zur Arbeit am Pumpversuchstand (GUI)

Um die untersuchten Algorithmen zu verwenden, muss eine Benutzeroberfläche erstellt werden. Daher wurde für diese Aufgabe ein Framework namens *Dear PyGUI* ausgewählt. Mit seiner Hilfe kann eine Schnittstelle erstellt werden, auf die der Benutzer mit dem Pumpversuchstand interagieren kann. Oder dieses Interface kann auch benutzt werden, um zufällige Werte dorthin zu senden, Daten zu lesen und sie im Diagrammformat anzuzeigen.

Es ist notwendig, eine Schnittstelle mit der Programmiersprache *Python* zu erstellen, da die Algorithmen für maschinelles Lernen auch in Python geschrieben wurden.

### 6.1. Warum wurde die Dear PyGUI-Bibliothek für dieses Projekt ausgewählt?

*Dear PyGUI* unterscheidet sich grundlegend von anderen Python-GUI-Frameworks. Mehr als 70 Widgets sind hier verfügbar. Diese Bibliothek ermöglicht es uns, dynamische Schnittstellen zu erstellen. Und sie verwendet auch keine nativen Widgets, sondern zeichnet sie mithilfe einer Computergrafikkarte, was gut für die Leistung ist.

Außerdem ist zu beachten, dass auch einige gute Frameworks zum Zeichnen der Oberfläche in Python existieren. Zum Beispiel gibt es Alternativen wie *Tkinter*, *Flexx*, *PyGUI* usw. Aber sie sind primitiv und haben nicht die Fähigkeit, ein normales Anwendungsdesign zu erstellen, sodass sie einfach und ausdruckslos aussehen.

Wir können das *Matlab*-Modul auch in Python verwenden, aber es zeigt nur Grafiken und erstellt keine funktionierende Schnittstelle, mit der der Benutzer interagieren kann. Aus diesem Grund wurde entschieden, das *Dear PyGUI-Framework* (oder kurz *DPG*) zu benutzen, um eine einfache und schöne Oberfläche zu entwickeln.

#### **Installieren der Dear PyGUI-Bibliothek**

Zuerst muss man sicherstellen, dass mindestens Python 3.7 (64 Bit) verwendet wird. Danach sollte der Benutzer einen Befehl in die Konsole schreiben, um das Framework zu installieren:

```
«pip install dearpygui»
```

## 6.2. Wie funktioniert die Benutzeroberfläche?

Um genau zu verstehen, wie die Benutzeroberfläche funktioniert, muss man sich auf das Diagramm beziehen, das diesen Prozess unten erklärt:

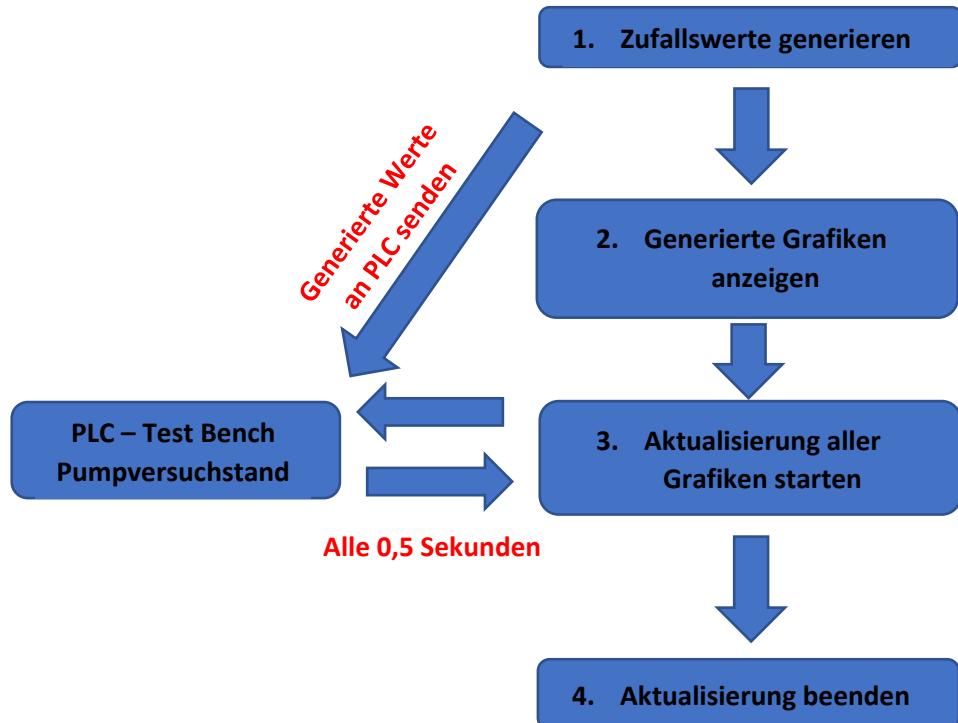


Abbildung 50. Schritte der Bedienung der Hauptfensterschnittstelle

Damit das System ordnungsgemäß funktioniert und die richtigen Werte in den Diagrammen angezeigt werden, müssen 4 Schritte ausgeführt werden. Der *Erste* besteht darin, die Anfangsparameter für das *Regelventil* (das Öffnungs niveau, wobei das Maximum 100% beträgt) sowie die *Anfangsdrehzahl* des Motors A zu erzeugen. Somit werden diese beiden Parameter zufällig generiert.

Wenn der Benutzer auf erste Schaltfläche klickt, werden diese beiden Zufallswerte an *Pumpversuchstand* gesendet. Dieser Vorgang des Initialisierens und Sendens der Daten erfolgt nur *1 Mal*, um die Anfangsparameter des Systems zu erhalten. Dann werden diese beiden Parameter nach einer bestimmten Zeit automatisch generiert.

Wenn der Benutzer im nächsten *zweiten* Schritt auf die zweite Schaltfläche klickt, werden die Diagramme angezeigt, die diese beiden Zufallswerte generiert haben. Dieser Teil hätte im ersten Schritt enthalten sein können. Aber um den Programmcode zu vereinfachen, wurde beschlossen, ihn zu einer separaten Schaltfläche zu machen.

Als Nächstes beginnt im *dritten* Schritt der Prozess des Sendens und Empfangens von Daten alle 0,5 Sekunden für das *Pumpversuchstand-System*. Hier sendet das Programm die generierten Werte für das *Ventil* und die *Motordrehzahl* und empfängt die Parameter des gesamten Systems. Zusätzlich werden alle Diagramme alle 0,5 Sekunden mit den Eingaben aktualisiert, die von dem Pumpversuchstand kommen.

Der letzte **vierte** Schritt ist formal, aber wenn der Prozess des Generierens, Empfangens von Daten und Aktualisierens von Diagrammen abgeschlossen werden muss, kann der Benutzer auf diese Schaltfläche klicken. Oder man kann das Programm beenden, es wird auch nicht mehr funktionieren. So wurde der Ablauf der Programmabläufe Schritt für Schritt beschrieben und die Benutzeroberfläche des Hauptbenutzeroberfensters erläutert.

### 6.3. Beschreibung der Bedienschritte der Benutzeroberfläche

#### 6.3.1. Der Ausgangszustand der Benutzeroberfläche

Um den Aufbau der GUI zu verstehen, es ist besser, ihren Anfangszustand zu demonstrieren:

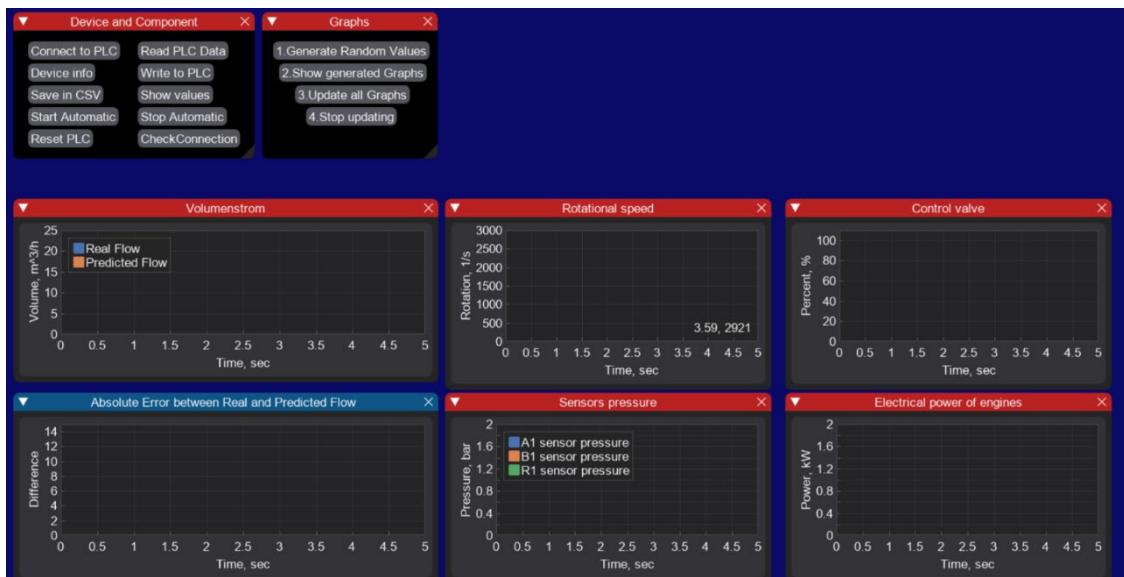


Abbildung 51. Der Anfangszustand der GUI, wenn das Programm ausgeführt wird

So sieht die Benutzeroberfläche aus, sobald das Programm gestartet wird. Das Design wurde nur erstellt, um die *Funktionalität* des Prototyps zu demonstrieren. Das nächste Kapitel zeigt, wie der Benutzer das Interface-Design ändern oder verbessern kann.

In der Mitte der Benutzeroberfläche werden 6 Diagramme mit den Hauptparametern *Pumpversuchstand-System* angezeigt. Mehr darüber wird später geschrieben.

Hier können ein paar Fragen auftreten, da oben links zwei Fenster mit den Namen "*Device and Components*" und "*Graphs*" zu sehen sind. Die Abbildung 50 mit einer Beschreibung der Arbeitsschritte bezieht sich speziell auf das GUI-Interface (Abbildung 51).

Das Fenster "*Gerät und Komponenten*" mit vielen Eigenschaften wurde erstellt, damit der Benutzer sie bei Bedarf verwenden kann. Beispielsweise braucht man erneut mit dem Pumpversuchstand verbinden, dann kann man eine Funktion "Mit PLC verbinden" nutzen. Oder man kann die aktuellen Parameter in einer CSV-Datei speichern. Hier gibt es auch die Möglichkeit, die Informationen über alle Eingaben zu zeigen. Im Allgemeinen ist dies ein universelles Fenster. Aber es sollte gesagt werden, dass diese ersten Fenster eher für PRO-Benutzer geeignet ist, die sich mit dem System auskennen oder es irgendwie testen möchten.

Im Fenster "Graphs" werden alle Schritte, die auch in Abbildung 50 gezeigt wurden, in der Reihenfolge und in Form der Schaltflächen angezeigt. Der Benutzer klickt also zuerst auf die *erste* Schaltfläche, wo er Zufallswerte generiert. Nach dem Drücken der *zweiten* Taste werden die generierten Diagramme angezeigt. Die *dritte* und *vierte* Schaltfläche werden verwendet, um die Anzeige und Aktualisierung von Diagrammen zu starten und zu stoppen.

### 6.3.2. Der Zustand der GUI-Inferface, wenn dritte Schaltfläche gedrückt wird

Wenn dritte Schaltfläche gedrückt wird, beginnt der Aktualisierungsprozess der Diagramme, wobei Daten vom Pumpversuchstand gesendet und empfangen werden. Das kann unten gezeigt werden:



Abbildung 52. Die dritte Schaltfläche auf der GUI wird gedrückt

Außerdem ist hier bereits genau sichtbar, wie die restlichen Grafiken aktualisiert werden. Alle Informationen über den Status der Sensoren, den Öffnungsgrad des Ventils und andere Anzeigen werden alle 0,5 Sekunden angezeigt. Auf der linken Seite befindet sich auch ein Volumenstromdiagramm. Und hier werden zwei Volumenströme angezeigt: einer ist real und der andere wird mithilfe eines maschinellen Lernprogramms vorhergesagt. Unten links befindet sich ein Diagramm der Differenz zwischen dem tatsächlichen und dem vorhergesagten Volumenstrom. Somit kann der Benutzer deutlich sehen, wie genau der Algorithmus für maschinelles Lernen funktioniert.

Es wurde möglicherweise bemerkt, dass, wenn der Benutzer auf die erste und zweite Schaltfläche klickt, werden diese Tasten im Fenster "Grafiken" verschwinden. Aber warum passiert das? Man kann nur einmal auf die erste und zweite Schaltfläche klicken, um den Datengenerierungsprozess zu initialisieren. Diese Tasten werden in dem Moment entfernt, in dem der Benutzer daraufklickt, damit man sie nicht versehentlich erneut klickt und das Programm nicht unterbricht.

## 6.4. Erläuterung des Codes, der beim Erstellen der GUI verwendet wurde

Um die Schnittstelle zu erstellen, wurde Code in der Programmiersprache Python unter Verwendung des oben beschriebenen Dear PyGUI-Frameworks geschrieben.

### 6.4.1. Erstellen des Hauptfensters und eines Arrays mit allen Daten

Gleich zu Beginn der Schnittstellenkonstruktion wird das größte *Hauptfenster* erstellt, das sich auf dem gesamten Bildschirm befindet. Zu diesem Zweck wird der folgende Teil des Codes verwendet:

```
488     dpg.create_viewport(title='GUI Interface', width=1280,
489                           height=720, clear_color=(10, 10, 104, 255))
490     ConnectPLC()
491
492     readedValuesfromPLC = [MotorA_Drehzahl.Value, Temp.Value,
493                           Volumenstrom.Value, Ventil.Value, EngineA_Power.Value,
494                           EngineA_Wirkleistung.Value, EngineA_Leistungsfaktor.Value]
```

Die Funktion *dpg.create\_viewport* ermöglicht es uns, das Hauptfenster zu erstellen. Sein Name ist in Klammern angegeben, ebenso wie seine Breite und Höhe. Darüber hinaus kann der Benutzer eine bestimmte Hintergrundfarbe feststellen, wenn es dafür einen Grund gibt.

Mit der Funktion *ConnectPLC()* in Zeile 490 wird eine Verbindung zum Pumpversuchstand aufgebaut. Wenn das Programm also startet, wird versucht, eine Verbindung herzustellen, was sehr praktisch ist und automatisch geschieht. Von 492 bis 494 Zeilen wird ein Array *readedValuesfromPLC* festgestellt, in dem die Werte von Sensoren und anderen Parametern vom Pumpversuchstand aufgezeichnet und empfangen werden. Auf diese Weise wird von Anfang ein globales Array mit allen benötigten Eingaben und Ausgaben erstellt, das dann aktualisiert wird.

### 6.4.2. Diagramme aktualisieren und Daten abrufen

Zur Vereinfachung der Erklärung wird die Aktualisierung des Graphs, der den Wert des Zustands des *Regelventils* anzeigt, betrachtet. Wenn der Benutzer auf die dritte Schaltfläche klickt, um den Aktualisierungsprozess von Diagrammen zu starten, wird die Funktion "*Onactualisationgrafics*" abgerufen:

```
265     def Onactualisationgrafics():
266         print("Thread actualisationgrafics started")
267
268         while stop == False:
269
270             readedValuesfromPLC = [MotorA_Drehzahl.Value, Temp.Value,
271                                   Volumenstrom.Value, Ventil.Value, EngineA_Power.Value,
272                                   EngineA_Wirkleistung.Value, EngineA_Leistungsfaktor.Value]
273
274         global i
```

```

273     XdataGraphs = list(np.arange(0, i, 0.5))
274
275     VEN_DataG.append(float(readedValuesfromPLC[4]))
276
277     # Set part for Ventil
278     dpg.set_value('rev_plotREG', [XdataGraphs, VEN_DataG])
279     dpg.set_axis_limits("y_axisREG", 0, 110)
280     dpg.set_axis_limits("x_axisREG", 0, i + 5)
281
282     i += 0.5
283     sleep(0.5)
284
285     print("Thread actualisationgrafics endet")
286
287 ThreadActualisation=threading.Thread(target=Onactualisationgrafics)

```

Diese Funktion wird ausgeführt, solange der Wert der Variable *stop==False* ist. Es wird nur wahr sein, wenn der Benutzer auf die vierte Schaltfläche klickt, die für das *Stoppen* der *Aktualisierung* von Diagrammen verantwortlich ist. In Zeile 270 erhalten wir die tatsächlichen Werte von Sensoren und anderen Parametern vom Pumpversuchstand, wodurch jedes Mal ein Array *readedValuesfromPLC* mit allen Eingaben und Ausgaben erstellt wird.

Außerdem wird jedes Mal die *XdataGraphs-Variable* erstellt, die für die Erstellung eines Bereichs von 0 bis zum Wert *i* verantwortlich ist. Durch Erhöhen von *i* jedes Mal um 0,5 werden die Diagramme und Achsen alle 0,5 Sekunden aktualisiert.

Ein neuer Wert in Zeile 275 vom Regelventil wird der *VEN\_DataG-Liste* hinzugefügt, die während der anfänglichen Konstruktion der Grafiken erstellt wurde.

Und schließlich findet der Vorgang zum Aktualisieren des Diagramms und der X- und Y-Achse statt. Die Funktion *dpg.set\_value* ermöglicht es, neue Werte für eine Grafik mit einem bestimmten Tag festzulegen. Und die Funktion *dpg.set\_axis\_limits* setzt andere Grenzwerte für die X- und Y-Achse. Außerdem wird Zeile 287 benötigt, um den *Threading-Prozess* zu starten, der jedes Mal nur die Diagrammaktualisierungsfunktion abruft.

Als Ergebnis dieses Kapitels, eine einfache und benutzerfreundliche Oberfläche wurde entwickelt. Der Benutzer kann jetzt mit dieser Schnittstelle interagieren und den Prozess zum Abrufen und Vorhersagen von Daten steuern. Der gesamte Code und das Programm selbst befinden sich in einer Python-Datei namens "GUI Interface".

## 7. Zusammenfassung

Eines der wichtigsten Ziele dieser Masterarbeit war es, zu zeigen, dass Softwaresensoren (oder virtuelle Sensoren) als Alternative zu herkömmlichen physikalischen Sensoren verwendet werden können. Sie können auch alternativ für Prozessgrößen eingesetzt werden, die aufgrund technologischer Einschränkungen, großer Messverzögerungen oder hoher Investitionskosten schwer zu messen sind [3].

In dieser Masterarbeit wurde gezeigt, dass es möglich ist, ein Programm zu erstellen, das auf maschinellen Lernalgorithmen basiert und den Volumenstrom vorhersagen kann. Demonstrierte virtuelle Sensoren können ungemessene, aber wichtige Variablen aus anderen leicht zu messenden Variablen mit Computermodellen abschätzen. So kann mit nur 3 Eingabeparametern (Drehzahl der Motor, elektrische Leistung und Regelventil) der Wert des Volumenstromes vorhergesagt werden, ohne die mathematischen Formeln zu verwenden, um ihn zu berechnen. Der Vorteil dieser Vorhersage gegenüber der Labormessung besteht darin, dass die Schätzung in Echtzeit erfolgt. Auf diese Weise passt sich das erstellte Programm kontinuierlich an die Eingaben und Ausgaben an.

Die erfüllten Ziele dieser Masterarbeit bei der Arbeit mit einem Pumpversuchstand können unten beschrieben werden:

1. Im theoretischen Teil (Kapitel 4.1) wurden alle 9 maschinellen Lernalgorithmen *ausgewählt und beschrieben*
2. Der Prozess zur *Auswahl* von Eingabe- und Ausgabedaten wurde ebenfalls beschrieben und das Datenvorhersageschema wurde gezeigt (Abbildung 21)
3. Der Code für jeden der Algorithmen wurde erfolgreich entwickelt und implementiert. Mehr dazu in Kapitel 4.4
4. Systematischer Vergleich verschiedener Algorithmen für die Eignung zur Berechnung des Volumenstroms auf Frequenzumrichterdaten wurde durchgeführt (mehr dazu in Kapitel 5). Darüber hinaus wurde in diesem Kapitel für jeden Algorithmus beschrieben, wie die Genauigkeit der Datenvorhersage verbessert werden kann und warum es wichtig ist, ein großes Trainingsmodell zu haben, das für jede Kombination von Eingabedaten geeignet ist.
5. Die Normalisierung von Daten wurde auf alle Algorithmen angewendet, was die Vorhersage des Volumenstromes um etwa 15-20 % verbesserte.
6. Es wurde auch ein Vergleich verschiedener Parameter neuronaler Netze und ihre Auswirkungen auf die Genauigkeit der Datenvorhersage entwickelt. Dies ist ein wichtiger Teil, da es erklärt, wie die Merkmale eines neuronalen Netzes angepasst werden können und die beste Option zur Vorhersage von Daten ausgewählt werden kann. Mehr dazu in Kapitel 5.5.
7. Darüber hinaus wurde das GUI-Interface für die Interaktion mit Algorithmen und für die Steuerung des Pumpversuchstandes entwickelt und verbessert (letztes Kapitel 6)

Aber von all diesen Zielen der Masterarbeit besteht das Hauptziel darin, am Ende einen Systemvergleich von Algorithmen zu haben, um den Volumenstrom vorherzusagen. Dazu wurde in Kapitel 5 demonstriert, wie genau jeder von ihnen die Werte vorhersagt. Und basierend auf den Ergebnissen sind *künstliche neuronale Netze* und *SVM-Algorithmus* am

besten für den Pumpversuchstand geeignet, da sie die genaueste und stabilste Leistung zeigen. Darüber hinaus kann in die Kapitel 5.6 und 5.7 einige Tabellen mit dem Vergleich der Vor- und Nachteile aller untersuchten Algorithmen gefunden werden.

Die erstellten Programme zur Vorhersage des Volumenstromes können in Zukunft für weitere Forschung oder praktische Anwendungen verwendet und verbessert werden. Eine Liste möglicher Verbesserungen der vorgestellten Algorithmen kann auch hier vorgeschlagen werden:

1. Um die Leistung der Algorithmen zu verbessern, kann ein Trainingsmodell erstellt werden, das 1 Stunde oder länger trainiert wird. Es wird mehr Speicher in Anspruch nehmen und wesentlich mehr Zeit zum Trainieren und Vorhersagen benötigen. Aber die Genauigkeit kann verbessert werden.
2. Wenn mehr Eingabedaten benutzt werden, verbessert sich auch die Genauigkeit der Vorhersage. Daher könnten statt 3 Eingaben 5 oder 6 verwendet werden, was die Vorhersagegenauigkeit der Algorithmen deutlich erhöhen würde.
3. Es ist auch wichtig, sich daran zu erinnern, wie bedeutungsvoll die Verwendung der Datennormalisierung ist. Sie verbessert die Datenvorhersage erheblich, weil es für Algorithmen einfacher ist, mit normalisierten Eingaben und Ausgaben zu arbeiten. Deswegen kann man versuchen, die Normalisierung der Daten besser zu machen, um ein noch genaueres Ergebnis für die Vorhersage des Volumenstromes zu erzielen.

Auf diese Weise wurde in dieser Masterarbeit ein anschauliches praktisches Beispiel für die Erstellung eines virtuellen Sensors mit dem Pumpversuchstand gezeigt. Darüber hinaus wurde ein detaillierter systematischer Vergleich der Algorithmen durchgeführt und ihre Vor- und Nachteile wurden festgestellt. Diese maschinellen Lernalgorithmen können auch verwendet werden, um andere Daten vorherzusagen, wobei die Ausgabedaten im Programmcode geändert werden sollen und ein neues Trainingsmodell erstellt werden soll.

## Literaturverzeichnis

- [1] S. Assawajaruwan und B. Hitzmann, "Process Analysis | Bioprocess Analysis", Academic Press, S. 377-383, 2019. DOI: 10.1016/B978-0-12-409547-2.11324-1.
- [2] J. Abonyi, B. Farsang und T. Kulcsar, "Data-driven development and maintenance of soft-sensors," 2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI), S. 239-244, 2014. DOI: 10.1109/SAMI.2014.6822414.
- [3] F. Hernández-del-Olmo, E. Gaudies', und Duro, N. "Machine Learning Weather Soft-Sensor for Advanced Control of Wastewater Treatment Plants". Sensors, S. 14, 2019. DOI: 10.3390/s19143139
- [4] P. Nkulikiyinka, Y. Yan und F. Güleç. „Prediction of sorption enhanced steam methane reforming products from machine learning based soft-sensor models“, Energy and AI, S. 10, 2020. DOI: 10.1016/j.egyai.2020.100037
- [5] B. Pattnaik, A. Pattanayak und S. Udgata, "Machine learning based soft sensor model for BOD estimation using intelligence at edge". Complex Intell, S. 961–976, 2021. DOI: 10.1007/s40747-020-00259-9
- [6] Z. Ge, Z. Song und S. X. Ding, "Data Mining and Analytics in the Process Industry: The Role of Machine Learning," IEEE Access, vol. 5, S. 20590-20616, 2017, DOI: 10.1109/ACCESS.2017.2756872.
- [7] R. Hewage, "Artificial Neural Network in Python", 24 Mai, 2020. [Online]. Verfügbar unter: <https://medium.datadriveninvestor.com/artificial-neural-network-in-python-704fae2e23>
- [8] A. Maszanski, "K-nearest neighbor method", 19 Juli, 2021. [Online]. Verfügbar unter: <https://proglab.io/p/metod-k-blizhayshih-sosedey-k-nearest-neighbour-2021-07-19>
- [9] T. Yiu, "Understanding Random Forest?", 12 Jan, 2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [10] R. Gandhi, "Support Vector Machine — Introduction to Machine Learning Algorithms", 7 Jan, 2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [11] A. Joaquin C. Figueroa, "Starting in the world of Machine Learning", 16 Mai, 2019. [Online]. Verfügbar unter: <https://medium.com/@joaquinfigueroa/starting-in-the-world-of-machine-learning-884171f9e711>
- [12] M. Baliyan, "Implementation of Lasso Regression From Scratch using Python", 5 Sep, 2020. [Online]. Verfügbar unter: <https://www.geeksforgeeks.org/implementation-of-lasso-regression-from-scratch-using-python/>
- [13] K. Qshick, "Ridge Regression for Better Usage", 3 Jan, 2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db>
- [14] J. Brownlee, "How to Develop Elastic Net Regression Models in Python", 7 Oct, 2020. [Online]. Verfügbar unter: <https://machinelearningmastery.com/elastic-net-regression-in-python/>

- [15] Scikit-learn developers, “1.6. Nearest Neighbors – sklearn-learn 1.1.1 documentation”, 2022. [Online]. Verfügbar unter: <https://scikit-learn.org/stable/modules/neighbors.html>
- [16] K. Leung, “Principal Component Regression — Clearly Explained and Implemented”, 7 Apr, 2022. [Online]. Verfügbar unter: <https://towardsdatascience.com/principal-component-regression-clearly-explained-and-implemented-608471530a2f>
- [17] Jason LZP, “LSTM For Bitcoin Prediction In Python”, 25 Dec, 2021. [Online]. Verfügbar unter: <https://medium.com/geekculture/lstm-for-bitcoin-prediction-in-python-6e2ea7b1e4e4>
- [18] G. Seif, “Selecting the best Machine Learning algorithm for your regression problem”, 5 März, 2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/selecting-the-best-machine-learning-algorithm-for-your-regression-problem-20c330bad4ef>
- [19] P.Nesterov, “Topic 4. Linear Classification and Regression”, 9 December, 2021. [Online]. Verfügbar unter: [https://github.com/Yorko/mlcourse.ai/tree/main/jupyter\\_english/topic04\\_linear\\_models](https://github.com/Yorko/mlcourse.ai/tree/main/jupyter_english/topic04_linear_models)
- [20] C.Maklin, “K Nearest Neighbor Algorithm In Python”, 2 July, 2019. [Online]. Verfügbar unter: <https://towardsdatascience.com/k-nearest-neighbor-python-2fccc47d2a55>
- [21] Y. Kashnitsky, “Open Machine Learning Course. Topic 3. Classification, Decision Trees and k Nearest Neighbors”, 19 Februar, 2018. [Online]. Verfügbar unter: <https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-3-classification-decision-trees-and-k-nearest-neighbors-8613c6b6d2cd>
- [22] IBM, “What is Random Forest?”, 7 December, 2020. [Online]. Verfügbar unter: <https://www.ibm.com/cloud/learn/random-forest>
- [23] S. Luber, N. Litzel, “Was ist eine Support Vector Machine?”, 6 November, 2019. [Online]. Verfügbar unter: <https://www.bigdata-insider.de/was-ist-eine-support-vector-machine-a-880134/>
- [24] A. Yadav, “ SUPPORT VECTOR MACHINES(SVM)”, 20 Oktober, 2018. [Online]. Verfügbar unter: <https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589>
- [25] E. Burns, J. Burke “ Künstliches neuronales Netz (KNN)”, Juni, 2020. [Online]. Verfügbar unter: <https://www.computerweekly.com/de/definition/Kuenstliches-neuronales-Netz-KNN>
- [26] P. Sharma, “Keras Dense Layer Explained for Beginners”, 20 Oktober, 2020. [Online]. Verfügbar unter: <https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/>
- [27] P. Sharma, “Keras Activation Layers – Ultimate Guide for Beginners”, 7 December, 2020. [Online]. Verfügbar unter: <https://machinelearningknowledge.ai/keras-activation-layers-ultimate-guide-for-beginners/>
- [28] P. Sharma, “Keras Optimizers Explained with Examples for Beginners”, 2 December, 2020. [Online]. Verfügbar unter: <https://machinelearningknowledge.ai/keras-optimizers-explained-with-examples-for-beginners/>
- [29] J. Brownlee, “How to Prepare Data For Machine Learning”, 15 December, 2013. [Online]. Verfügbar unter: <https://machinelearningmastery.com/how-to-prepare-data-for-machine-learning/>