Marcel Wuwer, B.Sc.

Vorhersage des Energieverbrauchs von CNC-Maschinen mittels Machine Learning

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science

eingereicht an der
Technischen Universität Graz

Betreuer
Dipl.-Ing. Dr.techn. Markus Brillinger

Beurteiler
Univ.-Prof. Dipl-Ing. Dr.techn. Franz Haas

Institut für Fertigungstechnik in Kooperation mit Pro^{2}Future GmbH

Graz, im Oktober 2020

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum	Unterschrift

Kurzfassung

Aktuell steht die Reduktion der CO_2-Emissionen durch den Umstieg von fossilen auf erneuerbaren Energieträgern auf den Agenden vieler Regierungen. Daher wird auch die Senkung des Energieverbrauchs forciert. Es werden zunehmend auch kleine und mittelständische Unternehmens mit Produktionslosgröße Eins dazu verpflichtet, ihren Energiebedarf in der Produktion zu senken. Effiziente Anlagen und Maschinen liefern einen Beitrag dazu. In der zerspanenden Fertigung hat zudem die Bearbeitungsstrategie einen signifikanten Einfluss auf den Energiebedarf. Für die Produktion von Losgröße Eins spielt daher die Ermittlung des benötigten Energiebedarfs einer Bearbeitungsstrategie, bevor ein Bauteil gefertigt wird, eine entscheidende Rolle. In zahlreichen vorangegangenen Arbeiten wurden analytische Modelle zwischen dem Energiebedarf und der Bearbeitungsstrategie von zerspanenden Verfahren entwickelt. Deren Genauigkeit hängt jedoch wesentlich von der Parametrierung dieser Modelle durch definierte Versuche ab. In dieser Masterarbeit werden mehrere, verschiedene Machine-Learning Algorithmen, vor allem aus der Entscheidungsbaum-Familie untersucht (DecisionTree, RandomForest und boosted DesicionTree) auf ihre Tauglichkeit hin untersucht, den Energiebedarf von Zerspanungsoperationen basierend auf realen Produktionsdaten vorherzusagen, ohne dafür eigens Versuche durchzuführen. Die genauesten Vorhersagen wurden dabei mit dem RandomForest-Algorithmus erzielt.

Abstract

Nowadays, the reduction of CO_2 emissions by moving from fossil to renewable energy sources is on the agenda of many governments. At the same time, these governments are forcing the reduction of energy consumption. Today also small and medium enterprisess) with production lot size one are increasingly being obliged to reduce their energy requirements in production. Efficient plants and machines make a contribution to this. In machining processes, the machining strategy also has a significant influence on energy demand. For manufacturing of lot size one, the prediction of the energy demand of a machining strategy, before a component is manufactured plays a decisive role. In numerous previous studies, analytical models between the energy demand and the machining strategy have been developed. However, their accuracy depends largely on the parameterization of these models by dedicated experiments. In this thesis, different machine learning algorithms especially from the decision tree family (DecisionTree, RandomForest, boosted DesicionTree) are investigated for their ability to predict the energy demand of machining operations based on real production data, without the need for dedicated experiments. The most accurate energy demand predictions were achieved with the RandomForest algorithm.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CNC Computerized Numerical Control (dt. rechnergestützte numerische Steuerung)
UPS Uninterruptible Power System (dt. unterbrechungsfreie Stromversorgung)
KES Kognitives Energiemanagement System
CAM Computer-aided manufacturing (dt. rechnerunterstützte Fertigung)
CAD Computer-aided design (dt. rechnerunterstützte Konstruktion)
NC Numerical Control
K-NN k-Nearest-Neighbor (dt. Nächste-Nachbarn-Klassifikation)
SVM Support Vector Machine (dt. Stützvektormaschine)
SVR Support Vector Regression (dt. Stützvektorregression)
RF Random Forest
GPR Gauß-Prozess-Regression
PNN Probabilistic Neural Network (dt. neuronale Wahrscheinlichkeitsnetzwerke)
NSGA-II Nondominated Sorting Genetic Algorithm II
BpNN Back Propagation Neural Network
KMU kleine und mittelständische Unternehmen
SME small and medium enterprises
NaN Not a Number - Datentyp
Ges. Abweichung Gesamtabweichung
Mtl. abs. Abweichung Mittlere absolute Abweichung
EECI Electrical Energy Consumption Indicator

Symbolverzeichnis

Δs ... Wegdifferenz in x,y-Ebene
Δt ... Zeitdifferenz
Δz ... Wegdifferenz in z-Achsenrichtung
A ... Spanungsquerschnitt
c ... spezifisch Schnittkraft des Materials
c_S ... Schnittgeschwindigkeit
c_S ... Schnittgeschwindigkeit
$D_{bekannt}$ ... bekannter Werkzeugdurchmesser
E_i ... Energieverbrauch
$E(Y|X)$ ... Entropie der Datengruppe Y, gegeben X
$E(Y)$ ... Entropie der Datengruppe Y
F_c ... Schnittkraft
k ... Proportionalitätskonstante
k_{c1} ... spezifische Schnittkraft
k, b ... empirische Konstanten
L ... Kontaktlänge des Fräswerkzeugs
m ... Anzahl Messungen pro NC-Anweisung
n ... Anzahl NC-Anweisungen
P_{air} ... Leistungsbedarf bei Werkzeug in Luft
p_c ... Schnittleistung
P_{cut} ... Leistungsbedarf bei Fräsbearbeitung
p_{ges} ... Gesamtanzahl der Datenpunkte
p_{Klasse} ... Anzahl d. Datenpunkte in Klasse i
P_0 ... Grundenergiebedarf der Maschine
P_0 ... Grundenergiebedarf der Maschine
p_i ... Proportion der Datenpunkte in Klasse i
$P(x_{i-1} \mid y_{i-1})$ ... Startposition des Werkzeugs
$P(x_i \mid y_i)$ ... Endposition des Werkzeugs
Q ... Zeitspanvolumen
Q ... Zeitspanvolumen
R^2 ... erklärte Varianz
s_0 ... Start des Materialabtrags
s_1 ... Beginn des regulären Materialabtrags
t ... Schnitttiefe des Werkzeugs

t_{cs}, t_{ce} . . . Start-/Endzeit der Schnittbearbeitung
v_c . . . Schnittgeschwindigkeit
v_F . . . Vorschub
$W_{bekannt}$ . . . bekanntes Werkzeug
W_{neu} . . . unbekanntes Werkzeug
W_i . . . verwendetes Werkzeug
y_i . . . tatsächlicher Wert der Zielvariable
$\hat{y}_i$ . . . Vorhersagen
$\bar{y}$ . . . arithm. Mittel der Zielvariable

1 Einleitung

Motivation

Das hohe generelle Umweltbewusstsein in der heutigen Gesellschaft fordert von der Technik konkrete Lösungen zum Umweltschutz. Das ENERMAN-Projekt des Grazer Forschungsunternehmens Pro2Future GmbH sucht nach einer Möglichkeit durch das aktive Steuern des elektrischen Energieverbrauchs eines Computerized Numerical Control (dt. rechnergestützte numerische Steuerung) (CNC)-Maschinenparks monetäre Vorteile für den Betreiber zu erwirtschaften. Die strategische Relevanz zeigt sich im Vergleich der formgebenden Fertigungsverfahren: Während 70% der Werkzeugmaschinen das Werkstück spanend bearbeiten,[1] ist deren eindeutiger Nachteil gegenüber anderen Verfahren, wie in Abb. 1.1 ersichtlich, der hohe Energieaufwand. Bei einer CNC-Fräsmaschine wird dieser hauptsächlich von drei Verursachern getragen (s. Abb. 5.4):

- Kühlschmiermittelaufbereitung (meist extern)
- Nebenaggregate wie Hydraulik, Automatisierung
- Hauptspindel und Vorschubantriebe

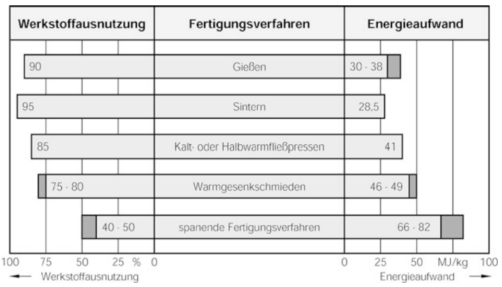


Abbildung 1.1: Werkstoffausnutzung und Energieaufwand verschiedener Fertigungsverfahren (Quelle: Fritz, 2018, S. 5)

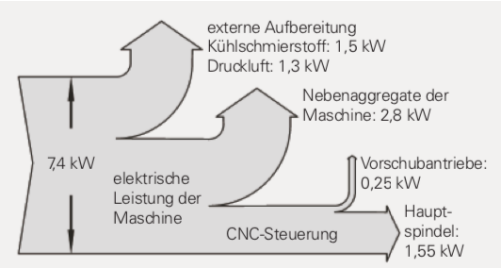


Abbildung 1.2: Mittlerer Leistungsbedarf für das Schlichten eines Gehäuseteils (Quelle: Dr. Johannes Heidenhain GmbH

[1] Fritz, 2018, S. 254.

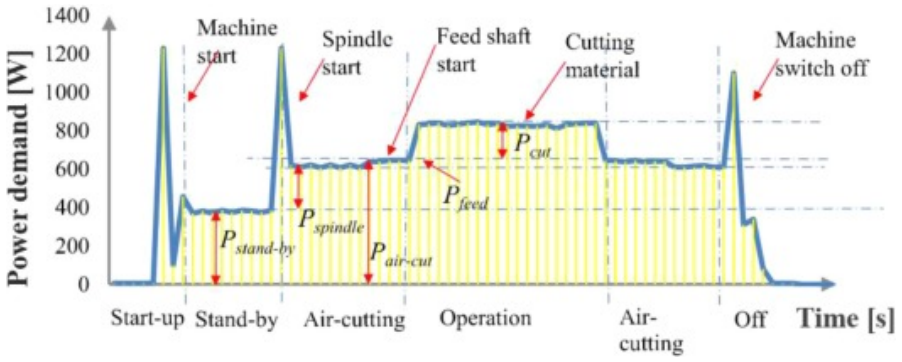


Abbildung 1.3: Arbeitszyklus der Spindeleinheit einer Werkzeugmaschine (Quelle: Abele u. a., 2011, S. 284)

Zeitlicher Verlauf des Energieverbrauchs von CNC - Bearbeitungsmaschinen

Über einen spanabhebenden Arbeitszyklus hinweg schwankt der Energieverbrauch. So bestehen Unterschiede im Energiebedarf zwischen Phasen des Beschleunigens oder Abbremsens und auch zwischen den einzelnen Bearbeitungsphasen wie Schruppen oder Feinschlichten (s. Abb. 2.4).

Um die Leistungsspitzen zu glätten ergeben sich zwei Möglichkeiten:

- Reduzierung der Maximalleistung und somit Verlängerung des Prozesses
- Ablaufplanung in einem Maschinennetzwerks[2]

Ersteres ist nicht zielführend, da ca. 85% des Leistungsbedarfs nicht im direkten Zusammenhang mit der Bearbeitung stehen[3] Eine Verlängerung des Prozesses zieht eine Verschlechterung des Wirkungsgrades mit sich und ist somit nicht zielführend. Der zweite Punkt impliziert eine aktive und smarte Zeitplanung um den Energieverbrauch zu beeinflussen, beispielsweise um die Leistungsspitze einer Maschine mit der negativen Leistungsspitze (Energieabgabe) einer anderen zusammen fallen zu lassen. Allerdings setzt dies voraus, dass man sehr genaue Kenntnisse über den zeitlichen Leistungsbedarf einer Bearbeitungsmaschine hat. Das mag in der Massenfertigung durch Messungen der ersten Teile bewerkstelligt werden, bei geringen Stückzahlen oder Einzelteilfertigung müsste aber die Leistungsaufnahme der Bearbeitungsmaschine im Vorfeld bestimmt bzw. vorausgesagt werden. Genau dies ist die Aufgabenstellung, welche in der vorliegenden Masterarbeit aufgegriffen wird. Diese Vorhersagen können anschließend nicht ein aktives und smartes Energiemanagement in einem Maschinennetzwerk verwendet werden: Auch in der Kette CAD - CAM - Fertigung können Vorhersagen der benötigten Energie bei der Fertigung, den Konstrukteur dabei unterstützen, Bauteile energieeffizient zu gestalten.

[2] Fang u. a., 2011.
[3] Fang u. a., 2011.

2 Stand der Technik

Für die Vorhersage des Energiebedarfs von CNC-Bearbeitungsmaschinen sind drei gegenwärtige Forschungsgebiete maßgeblich:

- Analytische Methoden zur Charakterisierung des Energiebedarfs von CNC-Maschinen
- Funktionsweise von Machine Learning-Algorithmen
- Anwendung von Machine Learning-Algorithmen in Produktionssystemen

Nachfolgend wird der Stand der Technik in diesen drei Gebieten umrissen.

2.1 Charakterisierung des Energiebedarfs von CNC-Maschinen

Der Energieverbrauch von CNC-Bearbeitungsmaschinen ist bereits umfassend untersucht und es existieren mehrere Modelle den Energieverbrauch zu beschreiben[1]. Alle Modelle haben gemein, dass der Energiebedarf der Maschine als Summe mehrerer Energiestadien ist:

$$\sum Energieverbrauch = Leistungsstadium_i * Zeit_i \tag{2.1}$$

Weiters unterscheiden sich die zahlreichen Modelle in der Aufschlüsselung der momentanen Leistungsstadien. So beschreibt beispielsweise Gutowski et al. den Energieverbrauch mit:[2]

$$E = (P_0 + k * Q) * t \tag{2.2}$$

[1] Pavanaskar, 2014, S. 7 f.
[2] Gutowski, Dahmus und Thiriez, 2006, S. 560 ff.

mit:

$$\begin{aligned} P_0 &\ldots \text{Grundenergiebedarf der Maschine} \\ k &\ldots \text{Proportionalitätskonstante} \\ Q &\ldots \text{Zeitspanvolumen} \end{aligned} \tag{2.3}$$

Spätere Ansätze sind grundsätzlich ähnlich aufgebaut und fokussieren auf eine weitere Aufschlüsselung und genauere Unterteilung der Arbeitszustände der Maschine. So führt beispielsweise Diaz et al. die Bewegung des Werkzeugs in der Luft, also ohne Materialabtrag hinzu:[3]

$$E = (P_{cut} + P_{air}) * t \tag{2.4}$$

wobei:

$$P_{cut} = (k * \frac{1}{Q} + b) * \Delta t \tag{2.5}$$

mit:

$$k, b \ldots \text{empirische Konstanten}$$

Zu bemerken ist hierbei, dass wiederum das Zeitspanvolumen Q ein entscheidender Parameter ist. Diaz et al. bestimmt für dieses Modell mit einem bestimmten Material und einer bestimmten Maschine empirische Werte für die genannten Konstanten. Allerdings befähigen diese Modelle nicht zur Vorhersage, da beispielsweise keine Bestimmungsmöglichkeit für die Zeitkomponente δt gegeben wird. Ziel dieser Modelle ist lediglich die Untersuchung und Aufschlüsselung des Energieverbrauchs von Bearbeitungsmaschinen. Ein analytischer Ansatz, mit diesen Charakterisierungen den Energieverbrauch vorherzusagen, wird im nachfolgenden Kapitel beschrieben.

2.2 Vorhersage des Energieverbrauchs von NC-Maschinen mittels analytischer Modelle

He u. a., 2012 beschreibt die Erstellung einer Methode zur Vorhersage des Energieverbrauchs von Numerical Control (NC)-Bearbeitungsmaschinen. Dazu

[3] Diaz-Elsayed, Redelsheimer und Dornfeld, 2011, S. 3 f.

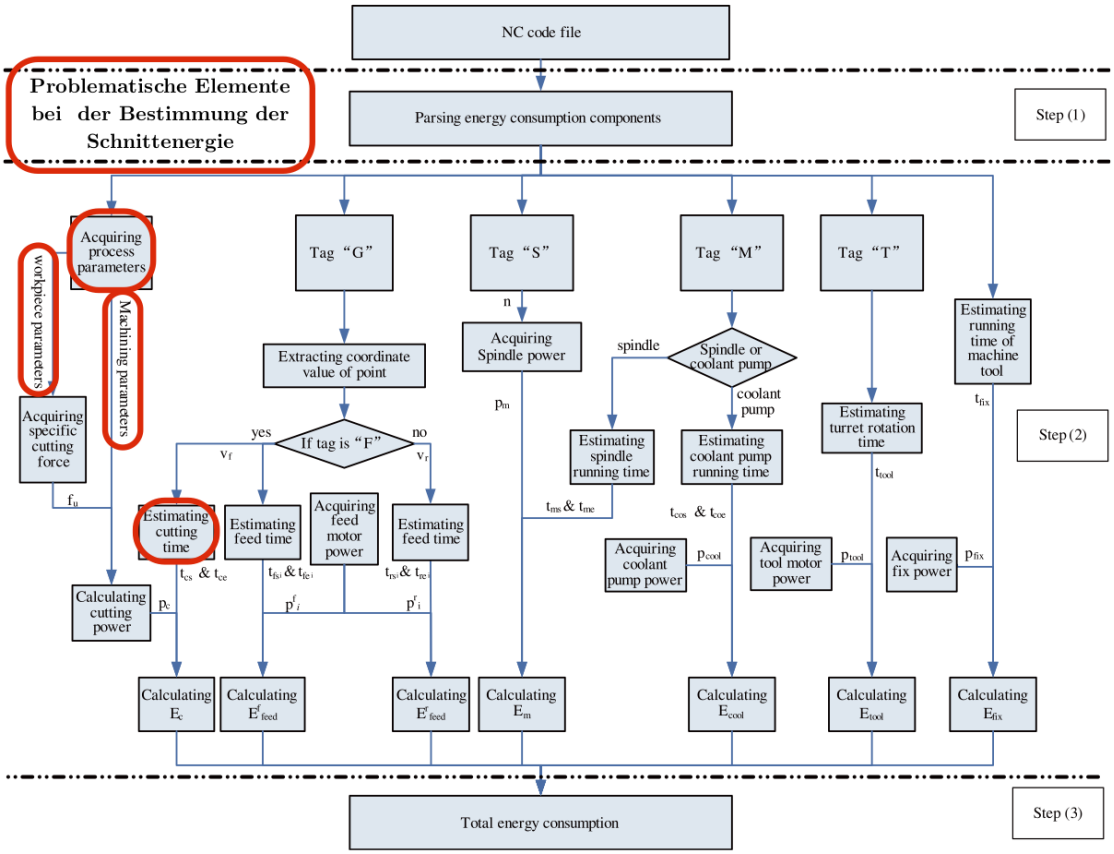


Abbildung 2.1: Analytisches Modell zur Vorhersage des Energieverbrauchs anhand des NC-Codes nach mit Markierungen der problematisch erscheinenden Elemente (Quelle: in Anlehnung an He u. a., 2012, S. 260)

werden die Befehle der NC-Datei in ihre Komponenten aufgesplittet und für jede Komponente eine analytische Methode entwickelt, den Energieverbrauch zu errechnen. Das gesamte analytische Modell ist in Abbildung 2.1 abgebildet. Die benötigte Schnittenergie E_c wird dabei beispielsweise vereinfacht zu:[4]

$$E_c = \int_{t_{cs}}^{t_{ce}} p_c \, \mathrm{d}t \tag{2.6}$$
$$p_c = F_c * v_c$$

wobei die Schnittkraft vereinfacht wird zu:

$$F_c = k_{c1} * L * t \tag{2.7}$$

mit:

t_{cs}, t_{ce} ... Start-/Endzeit der Schnittbearbeitung
p_c ... Schnittleistung
k_{c1} ... spezifische Schnittkraft
L ... Kontaktlänge des Fräswerkzeugs
t ... Schnitttiefe des Werkzeugs

Dieser Ansatz birgt allerdings einige Nachteile:

- Vernachlässigung von Beschleunigungen
- Bedarf von empirischen Leistungskonstanten (z.B. k_{c1}, s. Gleichung 2.7)
- Energieverbrauch abhängig von Zeitvariable, welche schwer aus dem NC-Code zu errechnen ist
- Genaue Kenntnis der Bearbeitungsstrategie nötig (z.B. Schnitttiefe, nicht im NC-Code auslesbar, s. Gleichung 2.7)

Wegen all der genannten Nachteile sind die Vorhersagen relativ ungenau: Für ein einfaches Test-Werkstück (s. Abb. 2.2) ist der Fehler der Vorhersage des Gesamtenergiebedarfs bei $9,3\%$.[5] Mit dem Einzug des Machine Learnings und Deep Learnings in die Produktionstechnik reibt sich die Frage auf, ob eine Vorhersage besser und effizienter mit diesen Mitteln erreicht werden kann. Daher werden in den nachfolgenden Kapiteln die Grundlagen des Machine Learnings abgehandelt (Kapitel 2.3) und anschließend Anwendungen in Fräsprozessen (Kapitel 2.4) und konkret bei der Vorhersage des Energieverbrauchs dieser Prozesse diskutiert (Kapitel 2.5).

[4] He u. a., 2012.
[5] He u. a., 2012.

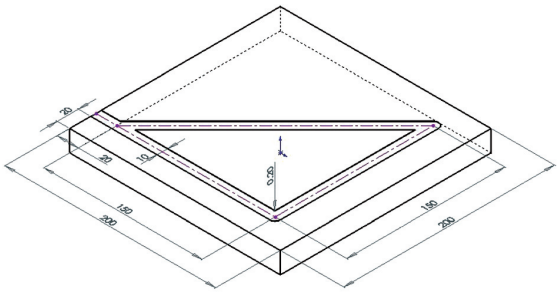


Abbildung 2.2: Test-Werkstück zum Vergleich der Vorhersage des analytischen Models mit Messdaten (Quelle: He u. a., 2012, S. 261)

2.3 Machine Learning-Algorithmen

Machine Learning beschreibt eine Klasse von Computerprogrammen, welche in der Lage sind aus Erfahrungen E zu lernen und somit für eine gegeben Aufgabe T, bezüglich einem Leistungsmaßstab P, mit mehr Erfahrung bessere Ergebnisse zu erzielen.[6] Die Entwicklung von Algorithmen ist ein reges Forschungsfeld der Informatik und so stehen heute eine Vielzahl von anwendbaren Algorithmen zur Verfügung. Die Funktionsweise der wichtigsten Algorithmen soll hier beleuchtet werden.

2.3.1 Entscheidungsbaum

Ein Entscheidungsbaum trifft Entscheidungen indem für einen zu klassifizierenden Datenpunkt eine Reihe einfacher Entscheidungen getroffen werden, bis das Ende einer Entscheidungskette erreicht wird. Im Beispiel in Abb. 2.3 beginnt die Klassifizierung in der Wurzel des Baumes (oberster Kasten) mit der Überprüfung ob die zu untersuchende Zahl die Bedingung $x \leqslant 4,95$ erfüllt. Trifft dies zu wandert die Variable den Stamm entlang zur nächsten Entscheidung $x \leqslant 0,8$. Trifft auch dies zu befindet sich die Variable bereits in einem Blatt und somit am Ende der Entscheidungskette. Die Variable kann somit zu einer Gruppe von bekannten Daten zugeordnet werden.

Erstellung von Entscheidungsbäumen

Bei der Erstellung von Entscheidungsbäumen werden zwei Methoden benötigt:

- Quantifizieren der Unordnung einer Datengruppe
- Optimierung der Einteilung einer Datengruppe

[6] Mitchell, 2010, S. 2.

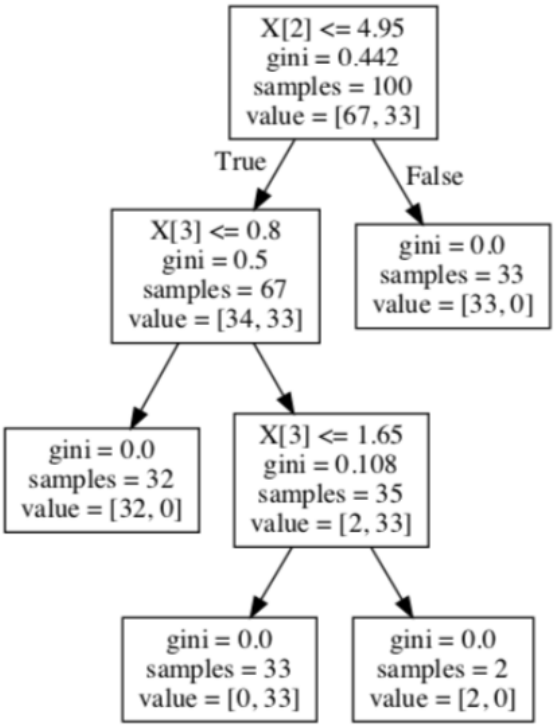


Abbildung 2.3: Aufbau eines Entscheidungsbaums (Quelle: Fenner, 2020, S. 246)

Quantifizieren der Unordnung einer Datengruppe

Um die Unordnung oder gegenteilig, die Reinheit einer Datengruppe zu ermitteln stehen mehrere Methoden zur Verfügung. Neben dem gini-Koeffizienten ist die Entropie eine häufig angewandte Methode:

$$Entropie(S) \equiv \sum_{i=1}^{c} -p_i \log_2(p_i) \tag{2.8}$$

$$p_i = \frac{p_{Klasse}}{p_{ges}} \tag{2.9}$$

mit:

p_i ... Proportion der Datenpunkte in Klasse i
p_{Klasse} ... Anzahl d. Datenpunkte in Klasse i
p_{ges} ... Gesamtanzahl der Datenpunkte
c ... Anzahl der Klassen

Informationsgewinn

Im vorangehenden Schritt wurde die Unordnung einer Datengruppe errechnet. Im nachfolgenden Schritt soll die Frage beantwortet werden, wie diese Unordnung S der Gruppe Y mithilfe einer weiteren Information X verringert werden kann. Der Informationsgewinn ist dann:

$$IG(Y, X) = E(Y) - E(Y|X)$$

mit:

$$E(Y) \ldots \text{Entropie der Datengruppe Y}$$
$$E(Y|X) \ldots \text{Entropie der Datengruppe Y, gegeben X}$$

Mit diesem beiden mathematisch-statistischen Werkzeugen klassifiziert der Algorithmus die gegebenen Daten, indem für jede Datengruppe jene Unterteilung unternommen wird, welche den maximalen Informationsgewinn liefert.

2.3.2 Weitere Algorithmen

Neben der Vielzahl von verfügbaren Algorithmen werden in der vorliegenden Arbeit zwei weitere verwendet:

k-Nearest-Neighbor

Dieser Algorithmus ordnet Datenpunkte aufgrund ihrer näheren geometrischen Umgebung (k nächste Nachbarn) ein und ordnet neue Datenpunkte entsprechend in Klassen ein (s. Abb. 2.4)

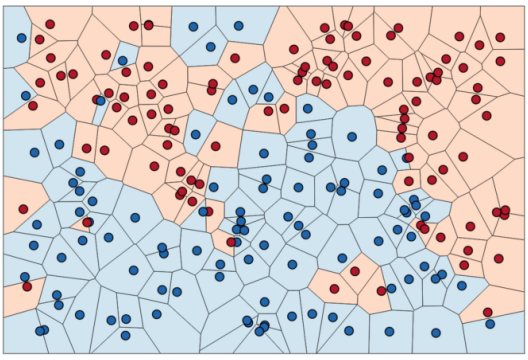


Abbildung 2.4: Klassifizierung aufgrund der k nächsten Nachbarn (Quelle: Subramanian, 2019)

Support Vector Machine

Die Stützvektormethode (Support Vector Machine) transferiert den Datensatz mithilfe von Stützvektoren in höhere Dimensionen. Anschließend wird versucht einen linearen Zusammenhang zu finden (Hyperebene) und Datenpunkte zu klassifizieren. Dadurch können nichtlineare Klassengrenzen in lineare Klassengrenzen umgewandelt werden (s. Abb. 2.5).

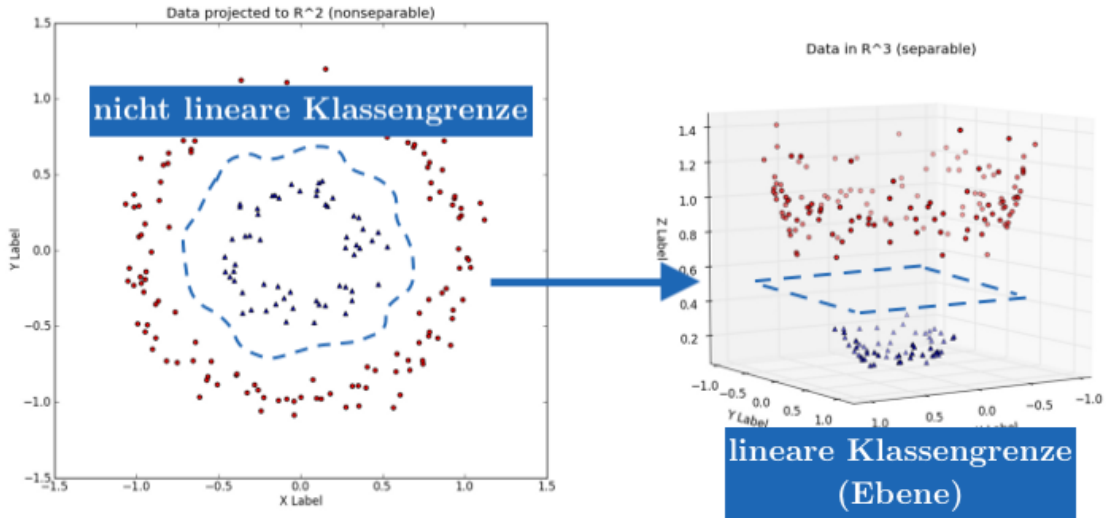


Abbildung 2.5: Transfer des Datensatzes durch Stützvektorenmotheode in höhere Dimensionen um linearen Zusammenhang zu finden (Quelle: in Anlehnung an Chakure, 2019)

2.3.3 Ensemble Methoden

Neben den bisher diskutieren klassischen Algorithmen kommen noch Ensemble Methoden zum Einsatz. Diese sind keine Algorithmen per se, sondern beschreiben lediglich Vorgehensweisen wie diese Algorithmen aufgebaut werden. Eine Ensemble Methode kann also auf verschiedene Algorithmen angewandt werden.

Bagging - Random Forest

Unter Bagging (Boostrap Aggregating) versteht man die zufällige Aufteilung der Testdaten in n Subdatensätze (Schritt 1) und das parallele Trainieren von n Algorithmen für jeden Subdatensatz (Schritt 2, s. Abb. 2.6). Jeder Algorithmus liefert dann eine Vorhersage, welche dann zu einer Gesamtvorhersage zusammen gefasst wird (Voting). Eine bekannte Implementierung dieser Methode mit dem Entscheidungsbaum wird Random Forest genannt.

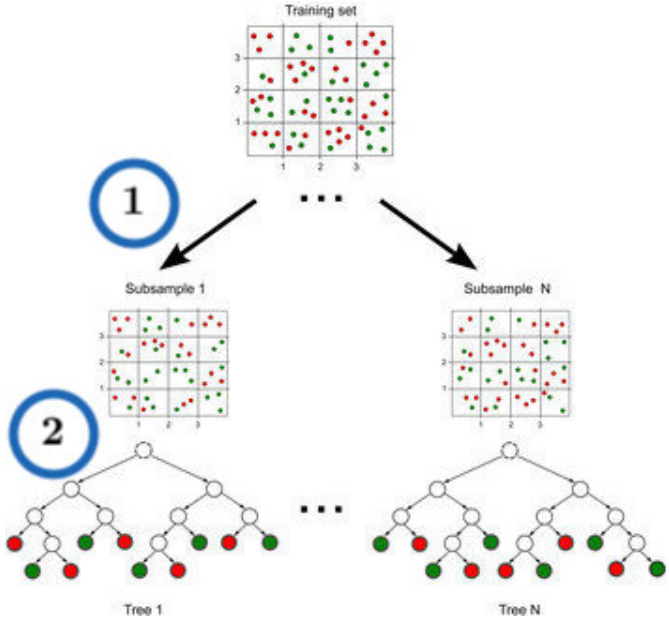


Abbildung 2.6: Funktionsprinzip des Bagging (Quelle: Machado, Recamonde-Mendoza und Corbellini, 2015)

Boosting - Adaptive Boosting

Boosting ist eine klassische Ensemble Methode die großen Einsatz in Klassifizierungsproblemen findet. Aber auch bei Regressionsproblemen kann Boosting zu deutlichen Verbesserungen der Vorhersagen führen.[7] Im Gegensatz zum Bagging werden beim Boosting die Algorithmen nicht parallel, sondern nachfolgend angelernt. Dabei wird bei jedem sequenziellen Schritt aufgrund der Fehler des vorhergehenden Algorithmus die Gewichtung der Datenpunkte angepasst. Unter allen verfügbaren Methoden liefert der AdaBoost-Algorithmus (Adaptive Boosting) dabei die besten Ergebnisse[8], weshalb dieser auch hier verwendet wird.

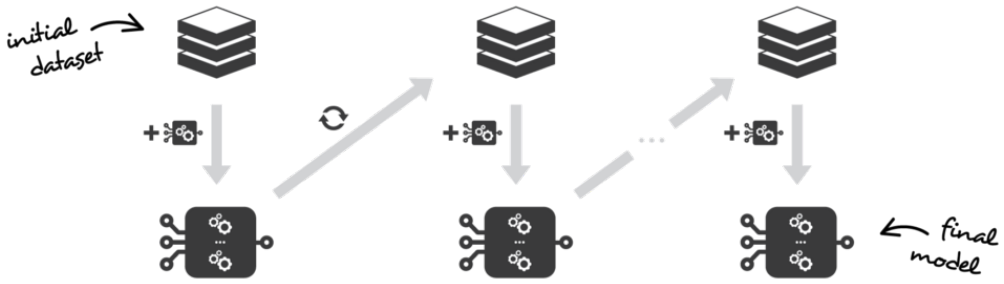


Abbildung 2.7: Boosting: Sequentielle Entwicklung des Algorithmus (Quelle: Rocca, 2019)

[7] Drucker, 1997.
[8] Duffy und Helmbold, 2002, S. 155 ff.

Stacking

Beim Stacking werden verschiedene Machine Learning-Algorithmen nacheinander trainiert, wobei der jeweils nachfolgende Algorithmus nicht auf die tatsächlichen Daten zugreifen kann, sondern nur aufgrund der Vorhersagen des jeweils vorhergehenden Algorithmus sein Vorhersagemodel erstellt: Dadurch kann auch bei Regressionsproblemen, wie dem hier vorliegenden, die Genauigkeit der Vorhersagen gegenüber den einzelnen Methoden gesteigert werden.[9] Allerdings leidet, vergleichbar wie auch bei neuronalen Netzwerken, bei dieser Methode die Nachvollziehbarkeit der Ergebnisse durch die Entscheidungsfindung in mehreren Schichten (Layers) stark. Auf eine eventuelle Verbesserung des Vorhersagefehlers wird daher bewusst zugunsten des Verständnisses des Modells verzichtet und die Methode des Stackings nicht weiter behandelt.

2.3.4 Bewertungsmethoden der Vorhersagen

Gesamtabweichung

Die Gesamtabweichung der Vorhersagen $\hat{y}_i$ von deren tatsächlichen Werten y_i wird berechnet:[10]

$$Gesamtabweichung = \sum(\hat{y}_i - y_i)$$

Diese Maßzahl ist intuitiv verständlich, bietet allerdings den Nachteil, dass sich Fehler gegenseitig aufheben können und somit keine Aussage über die Qualität der Vorhersagen getroffen werden kann.

Mittlere absolute Abweichung

Dieser Nachteil wird in dieser Methode durch die Verwendung von Absolutabweichungen, welche über die Probenanzahl n gemittelt werden, ausgebessert:[11]

$$MAA = \frac{1}{n}\sum(\mid \hat{y}_i - y_i \mid)$$

[9] Breiman, 1996.
[10] vgl. Fahrmeir u. a., 2016, S. 69 ff.
[11] vgl. Fahrmeir u. a., 2016, S. 370 ff.

RMSE – Root mean squared error

Eine weitere häufig angewendete Methode zur Bewertung der Übereinstimmung von zwei Datensätzen ist der RMSE (dt.: erwartete mittlere Abweichung).[12]

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Erklärte Varianz – R^2-Score

Die erklärte Varianz, oder auch R^2-Score gennant, ist ein bei Statistikern und Machine Learning-Anwendern häufig verwendetes Instrument zur Bewertung der Vorhersagen, da mit diesem Instrument ein Maß mit der Eigenschaft $0 \leqslant R^2 \leqslant 1$ und somit die Vergleichbarkeit verschiedener Datensätze gegeben ist:[13]

$$R^2 = 1 - \sum \frac{(y_i - \hat{y}_i)^2}{(y_i - \bar{y})^2}$$

mit:

y_i . . . tatsächlicher Wert der Zielvariable
$\hat{y}_i$ . . . Wert der Vorhersage
$\bar{y}$ . . . arithm. Mittel der Zielvariable

2.4 Machine Learning Anwendungen bei Fräsbearbeitungen

Im Zuge der regen Forschungstätigkeit im Bereich der smarten Produktion wurden bereits einige Anwendungen von Machine Learning in CNC-Fräsbearbeitungen untersucht. Die unterschiedlichen Anwendungen sind in Tabelle 2.1 aufgelistet. Es bestehen zahlreiche Untersuchungen zu Themen des Werkzeugverschleißes/versagen und dem Vibrations/-Ratterverhalten. Energiethemen, besonders die Vorhersage des Energieverbrauches, stehen bis dato nicht im Vordergrund. Nachfolgend soll der Stand der Technik für dieses Themengebiet beleuchtet werden.

[12] vgl. Fahrmeir u. a., 2016, S. 371 ff.
[13] Kvålseth, 2012.

Anwendung	Algorithmen	Input Parameter	Genauigkeit	Referenz
Überwachung Werkzeugverschleiß	K-NN, SVM	Werkzeugbilder	90,26%	Garcia-Ordas, 2017
Erkennung Werkzeugbruch	SVM, SVR	Schnittkraft, Energieverbrauch	99,38%	Cho u. a., 2005
Vorhersage Werkzeugverschleiß	RF	Schnittkraft, Vibration, akustische Emission	99,20%	Wu u. a., 2017
Vorhersage Energieverbrauch	GPR	Drehgeschw. Spindel, Vorschub, Schnitttiefe, aktive Werkzeugachse, Bearbeitungsstrategie	über 95%	Bhinge, Park u. a., 2016
Werkzeugverschleiß und Vorhersage verbleibende Nutzungsdauer	SVR	Vibration, Schnittkraft, akustische Emission	98,95%	Javed u. a., 2018
Vorhersage Energieverbrauch	GPR	Drehgeschw. Spindel, Vorschub, Schnittrichtung d. aktiven Werkzeugs, Schnitttiefe, Bearbeitungsstrategie, Länge des Werkzeugpfades	98,66%	Bhinge, Biswas u. a., 2014
Erkennung Werkzeugbruch	PNN	Drehgeschw. Spindel, Vorschub, Schnitttiefe, Maximalkraft, Varianz d. Maximalkraft	98,60%	Huang, Ma und Kuo, 2015
Optimierung Werkzeugpfad, Werkzeugwahl, Bearbeitungsparameter	NSGA-II	CAD Modell	N/A	Klancnik, Brezocnik und Balic, 2016
Vorhersagen Oberflächenrauheit	SVM	Drehgeschw. Spindel, Vorschub, Schnitttiefe	86,5%	Xiaohong u. a., 2016
Vorhersagen Ratterschwingungen	SVM	Vibrationsmessungen (Schnittgeschwindigkeit in x und y-Richtung)	98,33%	Peng, Wang und Liao, 2015
Überwachung Werkzeugzustand	J48 Entscheidungsbaum, Feedforward BpNN	Vibrationsmessungen (Accelerometer)	94.30% (J48), 95.40% (NN)	Krishnakumar, Krishnaswamy und K I, 2015
Bestimmung von spezifischen Schnittkräften	BpNN	Werkstoff, Schneidmaterial, Beschichtung, Werkzeugdurchmesser, Schnittgeschw., Vorschub, Schnitttiefe, Eintritts-/Austrittswinkel, durchschn. Spandicke, etc.	87,44%	Arnold u. a., 2017
Vorhersage von Verformungen bei Herstellung von dünnwandigen Werkstücken (Fräsen)	Bayessche Lernmethode	Versatzinformation	N/A	Yuan u. a., 2017

Tabelle 2.1: Übersicht der Anwendungsfälle von Machine Learning bei Fräsbearbeitungen (Quelle: Kim u. a., 2018)

2.5 Machine Learning zur Vorhersage des Energieverbrauches von CNC-Fräsmaschinen

Die Vorhersage des Energieverbrauchs von Fräsprozessen ist mithilfe von Machine Learning mit vielversprechenden Ergebnissen möglich. Der bei Bhinge, Park u. a., 2016 verwendete Versuchsaufbau führt zu einer Vorhersagegenauigkeit für den gesamten Prozess von über 95%. Dafür wurden während der Herstellung des in Abb. 2.9 abgelichteten Werkstückes einerseits Daten aufgezeichnet (NC-Answeisung, Vorschub, etc.) und anschließend Daten wie der inkrementell zurückgelegte Weg errechnet. Abschließend wurden weitere Daten durch eine rekonstruierte Simulation generiert, aufbauend auf einerseits den Messdaten, andererseits auf zusätzlichen Informationen wie der Schnitttiefe, etc. Eine umfassende Auflistung und Einteilung der Daten ist in Abbildung 2.8 referenziert.

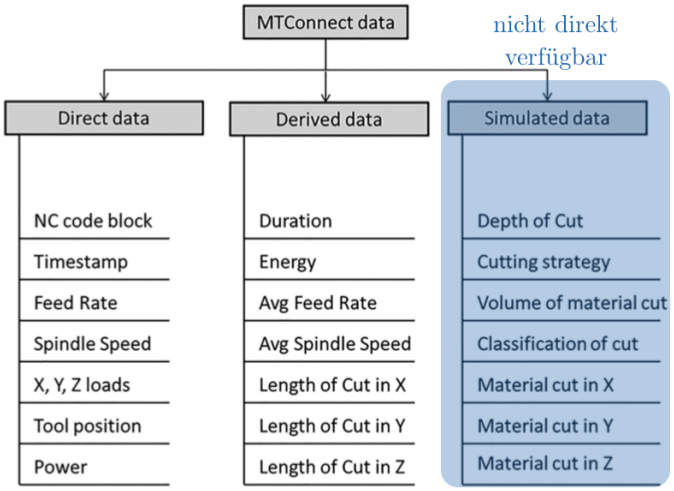


Abbildung 2.8: Verwendete Daten des Vorhersagemodells (Quelle: Bhinge, Park u. a., 2016)

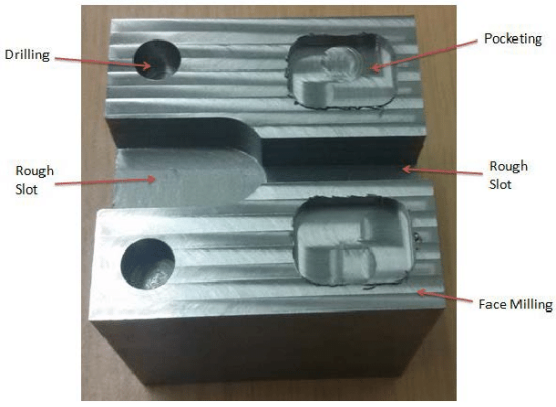


Abbildung 2.9: Werkstück mit ausgeführten Fräsbearbeitungen (Quelle: Bhinge, Park u. a., 2016)

3 Fokus dieser Arbeit

3.1 Forschungsfrage

Wie in Abschnit 2.2 beschrieben existieren Ansätze, den Energiebedarf von NC-Maschinen mittels analytischer Methoden aus dem NC-Code im Vorhinein zu bestimmen. Allerdings sind diese Ansätze trotz hoher Komplexität auf bestimmte Anwendungen limitiert und deren Vorhersagen sind mit hohen Fehlern behaftet. Daher wird versucht (s. Kapitel 2.5), den Energieverbrauch mittels Machine Learning vorherzusagen. Das vorgestellte Vorgehen nach Bhinge, Park u. a., 2016 liefert für ein einfaches Testwerkstück (s. Abb. 2.9) sehr gute Vorhersagen. Allerdings birgt dieses Vorgehen drei fundamentale Nachteile:

Laborwerkstücke Das vorgestellte Verfahren ist ausschließlich auf speziell gestaltete Werkstücke anwendbar, welche einfache Geometrien aufweisen und die Bearbeitungsgänge klar voneinander trennen (gekoppelte Bearbeitung). Ohne weiteres ist dieses Modell nicht auf alltägliche Werkstücke in der Produktion anwendbar.

Nachträgliche Datengenerierung (Simulation) Ein Großteil des Datensatzes wird nachträglich durch Simulation erstellt. Diese benötigt eine genaue Kenntnis des Bearbeitungsganges und ist nicht aus dem NC-Code ableitbar (bspw. Materialabtragsrate). Ob eine solche rekonstruierte Simulation bei realen, komplexeren Bauteilen mit vertretbarem Aufwand möglich ist, bleibt zu bezweifeln.

Fokussierung alleinig auf den Bearbeitungsgang Das Modell ist nur geeignet den losgelösten Bearbeitungsgang zu betrachten. Maschinenoperationen wie Werkzeugwechsel oder Anfahrwege werden außer Acht gelassen.

Die Frage stellt sich also, ob es mit Machine Learning möglich ist ein Vorhersagemodell zu entwickeln, welches auf reale Bauteile, wie sie in der Industrie auf CNC-Bearbeitungszentren hergestellt werden, anwendbar ist.

3.2 Aufgabenstellung der Masterarbeit

Bei der Fertigung auf CNC-Bearbeitungsmaschinen wird aus dem geometrischen Modell des Bauteils mit Hilfe von entsprechender Software eine Schritt-

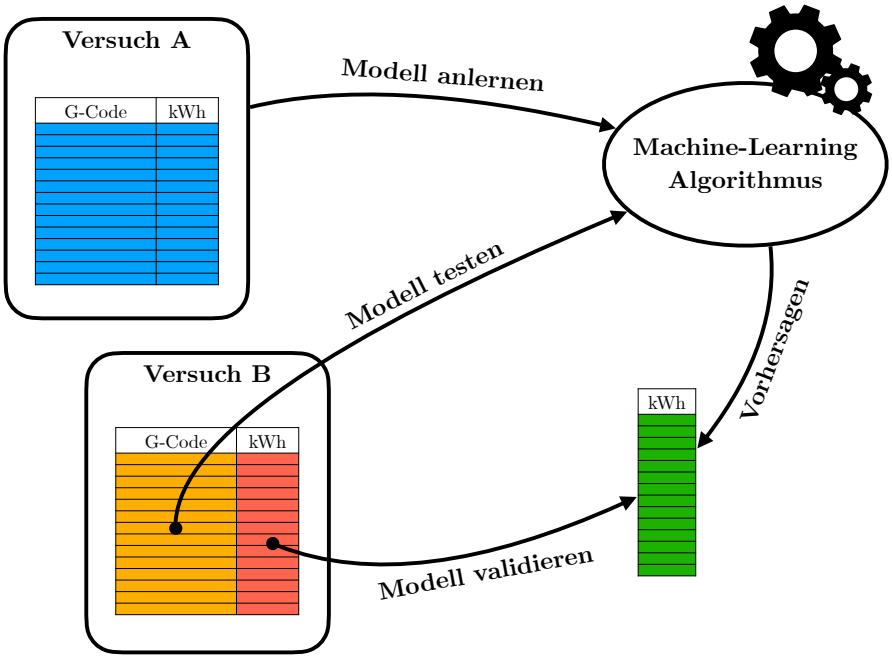


Abbildung 3.1: Ablauf: Anlernen und Validieren des Machine Learning Modells mit zwei unabhängigen Datensätzen (Quelle: Eigene Darstellung)

für-Schritt Anleitung für die Maschine erstellt, der sogenannte G-Code. Im Rahmen von zahlreichen Testreihen wurden bereits die Leistungsdaten einer Werkzeugmaschine für verschiedene Werkstückformen am Institut für Fertigungstechnik gemessen. Die Aufgabe der Masterarbeit ist ein Programm zu entwickeln, welches einen Zusammenhang herstellt zwischen den einzelnen Anweisungsblöcken des G-Codes und dem gemessenen Energiebedarf der Werkzeugmaschine während der tatsächlichen Fertigung des Bauteils. Anschließend soll die Möglichkeit untersucht werden, den Energiebedarf anhand des CNC-Codes mittels Machine Learning vorherzusagen. Kurz gefasst sind die beiden Hauptfelder:

- Herstellen eines Zusammenhangs zwischen NC-Anweisungen und gemessenen Leistungsdaten der Werkzeugmaschine
- Untersuchung der Möglichkeit den Energiebedarf mittels Machine Learning-Algorithmen vorherzusagen.

Erstellung von Machine Learning-Vorhersagen

Um die in Kapitel 3.2 geforderten Ziele zu erreichen sind folgende Aufgaben zu bewältigen (s. Abb. 3.2):

Parsing Der G-Code muss eingelesen und in eine verwertbare Datenstruktur gebracht werden. (s. Kapitel 5.1)

Feature extraction Aus den Daten müssen charakteristische Eigenschaften (Features) für die Anweisungsblöcke gebildet werden. (s. Kapitel 5.1)

Feature engineering Nicht alle Informationen sind ad hoc verfügbar. Durch Kenntnisse des Bearbeitungsverfahrens können weitere Charakteristika (Features) erstellt werden. (s. Kapitel 5.2)

Model training Der Machine Learning-Algorithmus muss mit den verfügbaren Daten trainiert werden.(s. Kapitel 5.3)

Prediction Als letzter Schritt können Vorhersagen (Predictions) für den G-Code des neuen Werkstücks erstellt werden. (s. Kapitel 5.4)

Abbildung 3.2: Ablauf der Erstellung von Machine Learning-Vorhersagen (Quelle: Eigene Darstellung)

3.3 Lösungsansatz

Die in Abschnitt 3.2 gestellten Aufgaben sollen mit Hilfe eines Machine Learning-Models bewältigt werden, welches im Vorhinein an dem CNC-Code eines Referenzbauteils (s. Abb. 4.1) und den Energiemessungen der tatsächlichen Fertigung dieses Referenzbauteils trainiert wurde. Zur Bewertung des trainierten Modells werden Vorhersagen für den Energieverbrauch eines zweiten, unabhängigen Fertigungsprozesses (Rückseite des Bauteils) getroffen und diese anschließend mit dem tatsächlichen Energieverbrauch validiert (s. Abb. 3.1).

Fünf Schritte-Prozess

Der Prozess zum Erreichen des Gesamtzieles, wie in Kapitel 3.3 dargestellt, wird in fünf Schritte unterteilt (s. Abb. 3.3): Beginnend mit einem prototypischen Modell, welches auf einem verringerten Datensatz aufbaut, werden anschließend iterativ jeweils der Algorithmus verbessert und der Datenumfang erweitert bis die angestrebte Modellreife erreicht wird.

Schritt 1: Aufbau eines prototypischen Modells mit vereinfachtem Datensatz In einem ersten Ansatz wird mit einem vereinfachten Datensatz ein Algorithmus trainiert und die Vorhersagen untersucht (s. Kap. 5).

Schritt 2: Verbesserung des prototypischen Modells Es werden die Schwächen des erstellten Modells untersucht und Möglichkeiten gesucht, die Vorhersagen zu verbessern.

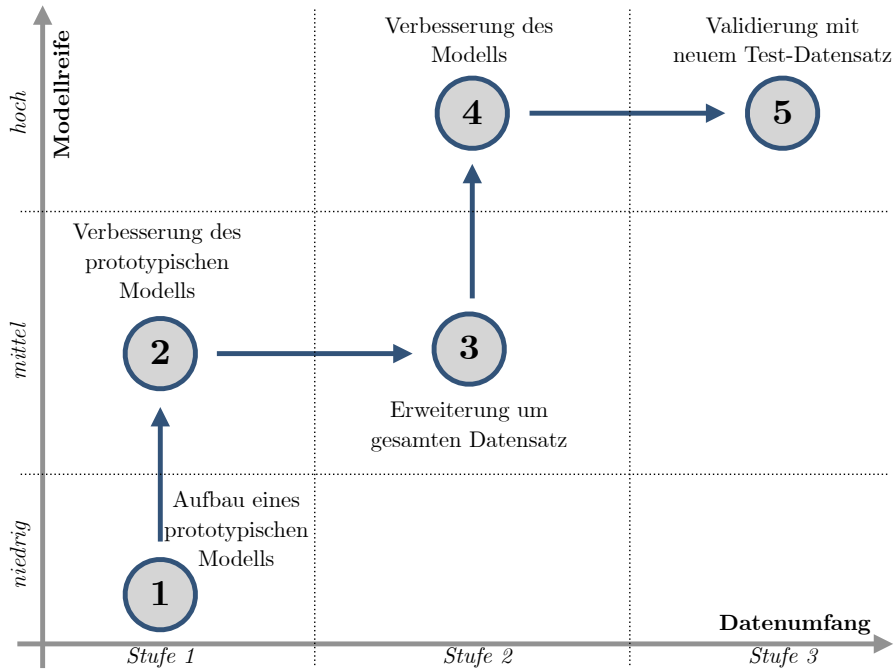


Abbildung 3.3: Unterteilung der gestellten Aufgabe in fünf Schritte (Quelle: Eigene Darstellung)

Schritt 3: Trainieren des erstellten Models mit dem gesamten Datensatz Das in Stufe 1 und 2 entwickelte und verbesserte Model wird derart angepasst, dass die Daten einer gesamten Werkstückbearbeitung verarbeitet werden können.

Schritt 4: Weitere Verbesserung des Modells Anschließend werden die Ergebnisse genau untersucht und Verbesserungen umgesetzt.

Schritt 5: Validierung des erstellten Models mit neuen Test-Daten Ein zweiter Datensatz, bestehend aus Daten der gesamten Bearbeitung eines zweiten Werkstückes, wird herangezogen um die Vorhersagegüte des Modells anhand unabhängiger Daten zu validieren und zu diskutieren.

Kontinuierlich steigender Datenumfang

Während des 5-Schritte-Prozesses kommen verschiedene Datenumfänge zum Einsatz, welche sich im Wesentlichen darin unterscheiden, welche Daten zur Vorhersage und zur Validierung herangezogen werden. So soll ermöglicht werden, dass der Algorithmus nicht als »blindes« Statistikinstrument bedatet, sondern dass der Algorithmus zusammen mit dem Verständnis der Ergebnisse wächst und in jeder Stufe iterativ verbessert werden kann.

Die drei Datenumfänge sind (s. Abb. 3.4):

Stufe 1 Aus den vorhanden Datensätzen der durchgeführten Versuche auf dem CNC-Bearbeitungszentrum wird händisch ein kleiner Datensatz (ca.

100 Datenpunkte) extrahiert, welcher einen einzelnen Arbeitsschritt der Maschine abgebildet.

Stufe 2 Als nächstes wird der gesamte Datensatz einer Werkstückbearbeitung herangezogen, sowohl zum Antrainieren wie auch zum Testen des Algorithmus.

Stufe 3 Abschließend werden unterschiedliche Werkstückbearbeitungen für das Antrainieren und das Testen verwendet. Im Vergleich zur vorangegangenen Stufe werden also Vorsagen für NC-Anweisungen erstellt, welche der Algorithmus noch nicht kennt.

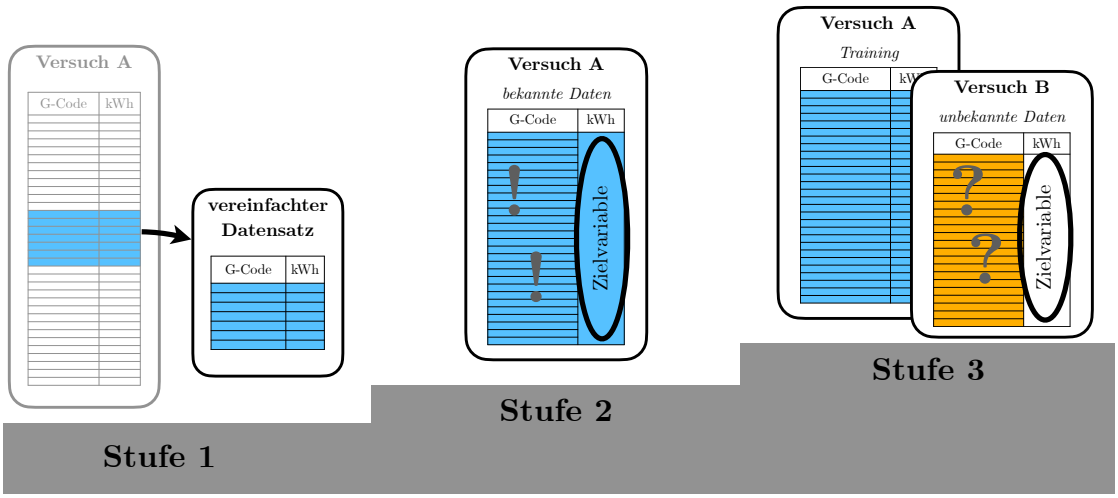


Abbildung 3.4: Drei Stufen des Datenumfangs (Quelle: Eigene Darstellung)

Da die drei Stufen des Datenumfangs stark von dem Datensatz abhängen, wird dessen Entstehung im nächsten Kapitel beleuchtet.

4 Daten

4.1 Datenerhebung

Der zugrundeliegende Datensatz stammt aus der Fertigung des Referenzbauteils (s. Abb. 4.1) auf dem 5-Achsen Bearbeitungszentrum U 5630 des Werkzeugmaschinenherstellers SPINNER. Abbildung 4.2 zeigt die im CAM-Programm simulierte Bearbeitung des Werkstücks mit dem Fräswerkzeug. Das Werkstück wird aus der Aluminiumlegierung AlCuMgPb (Material Nr. 3.1645) gefertigt.

Abbildung 4.1: gefertigtes Referenzbauteil mit Zwischenstufe (Quelle: Eigene Darstellung)

Ausgelesen wurden die Daten von der CNC-Steuereinheit SINUMERIK 840D SL (s. Abb. 4.3) im Anschluss der gesamten Fertigung. Im Datensatz sind aufgezeichnet: Die originalen NC-Anweisungen, zusätzliche von der Steuereinheit eingefügten Befehle und die gemessenen Momentanleistungen der verschiedenen Aggregate in kW. Die Messfrequenz der Ströme ist 500Hz ($\hat{=}$ alle 2ms Aufzeichnung des Messergebnisses). Der Datensatz wird von der Steuerung im .json-Dateiformat abgespeichert und muss vor der Verwendung noch bearbeitet werden, da einerseits die Daten nicht in tabellarischer Form zugrunde liegen und andererseits die NC-Anweisungen und Momentanleistungen getrennt voneinander abgespeichert werden und zuerst zugeordnet werden müssen. Die

Schritte zu einem verwendbaren Datensatz sind also wie folgt:

- Auslesen der .json-Datei aus der Steuereinheit SINUMERIK 840D SL
- Fließtext in tabellarische Form umwandeln (s. Abb. 5.3)
- Zusammenführen von NC-Anweisungen und Leistungsmessungen (s. Kapitel 4.2)

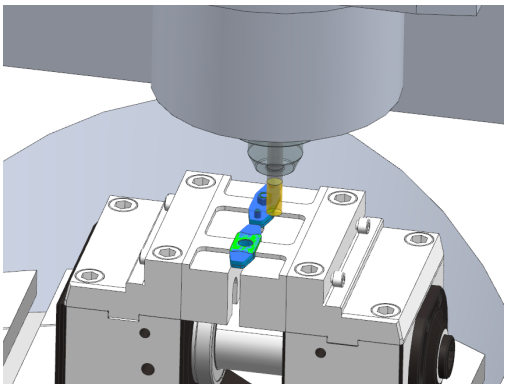


Abbildung 4.2: Simulation der Fertigung im CAM-Programm (Quelle: Eigene Darstellung)

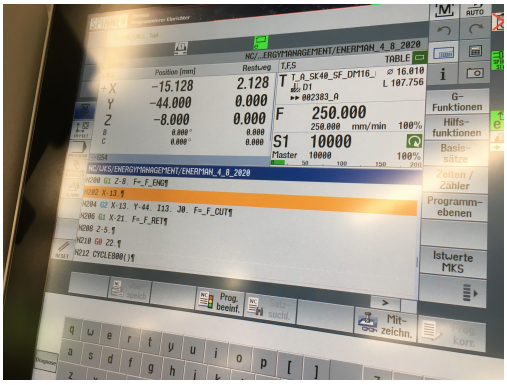


Abbildung 4.3: SINUMERIK Steuereinheit bei der Ausführung der G-Code Befehle (Quelle: Eigene Darstellung)

4.2 Zusammenführen von NC-Anweisungen und Leistungsmessungen

Die NC-Anweisungen und die Messergebnisse können über einen gemeinsamen Zeitstempel (HFPROBECOUNTER) zugeordnet werden. Allerdings fallen auf eine NC-Anweisung meist mehrere Messergebnisse (s. Abb. 4.4). Diese Situation kann auf mehrere Arten aufgelöst werden. Hier wurde derart verfahren, dass für jede NC-Anweisung die gemessenen Momentanleistungen $P_{i,j}$ mit der jeweiligen Zeitdifferenz $\Delta t_{i,j}$ zu einem Gesammtverbrauch W_i des NC-Anweisungsblocks zusammengefasst wird, wie in Abbildung 4.4 dargestellt:

$$
\begin{aligned}
W_{ges} &= \sum_{i=1}^{n} W_i \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} P_{i,j} \Delta t_{i,j}
\end{aligned}
$$

mit:

n . . . Anzahl NC-Anweisungen
m . . . Anzahl Messungen pro NC-Anweisung

Die Zeitdifferenz ist jeweils 2ms. Multipliziert mit der Momentanleistung in kW ergibt sich ein in Ws angegebener Energieverbrauch.

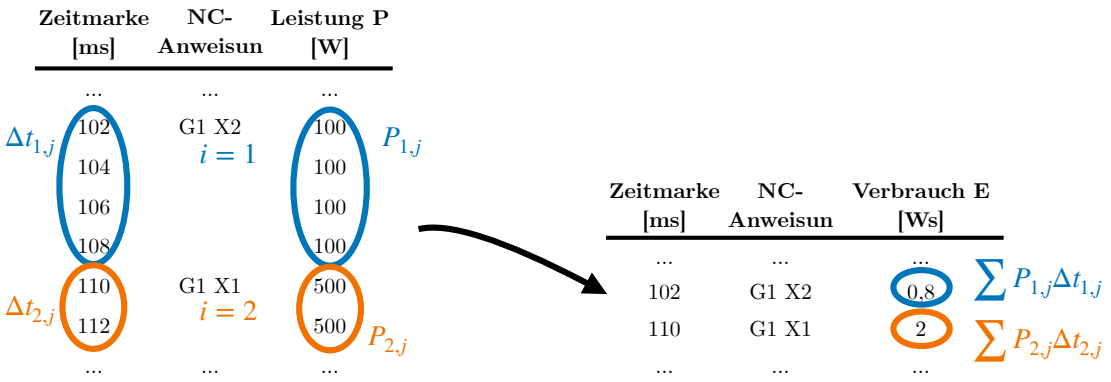


Abbildung 4.4: Zusammenführen von NC-Anweisungen und Leistungsmessungen (Quelle: Eigene Darstellung)

Nachteile der Zusammenführung

Wie bei jeder Vereinfachung gehen auch in diesem Schritt Informationen verloren:

Verlust der Momentanleistung Durch die Zusammenführung ist die maximale Momentanleistung einer NC-Anweisung nicht mehr verfügbar. Eine NC-Anweisung mit kurzer Dauer und hoher Leistungsanforderungen ist nicht zu unterscheiden von einer NC-Anweisung mit niedriger Leistungsanforderung aber hoher Dauer.

Verlust der Zeitinformation Konnten mit den bisherigen Daten Leistungsdiagramme unverzerrt über den Zeitstempel dargestellt werden, ist dies nicht mehr möglich, da durch die Zusammenführung jeder Datenpunkt (einzelne NC-Anweisung) eine unterschiedliche Dauer aufweist.

4.3 Informationsgehalt des erstellten Datensatzes

Nach der vorangegangen Umformatierung und Zusammenführung stehen die Daten zur weiteren Verwendung zur Verfügung. Neben einigen nicht relevanten

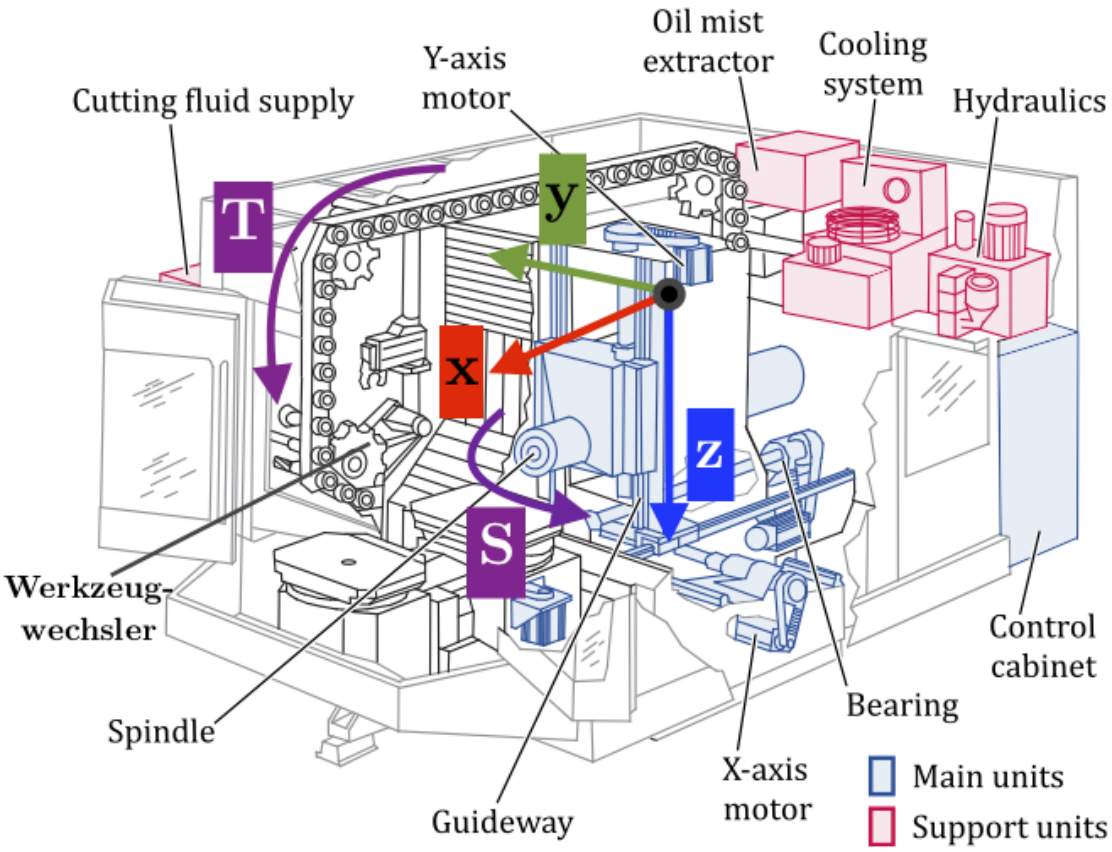


Abbildung 4.5: Betrachteten Verbraucher in einer Werkzeugmaschine (Quelle: angelehnt an Denkena u. a., 2020)

Datenspalten (z.B. ipoReadError, laBuf, IpoGC) stehen für jede NC-Anweisung folgende Daten zur Verfügung:

Bezeichnung	Datentyp	Einheit	Beschreibung
GCode	string	[-]	Inhalt der NC-Anweisung/internen Maschinenanweisung
ENERGY—1	float	[Ws]	Energieverbrauch des Aggregats zur Bewegung in x-Achse in Ws (später umbenannt zu ENERGY—x, s. Abb. 4.5)
ENERGY—2	float	[Ws]	Energieverbrauch des Aggregats zur Bewegung in y-Achse in Ws (später umbenannt zu ENERGY—y, s. Abb. 4.5)
ENERGY—3	float	[Ws]	Energieverbrauch des Aggregats zur Bewegung in z-Achse in Ws (später umbenannt zu ENERGY—z, s. Abb. 4.5)
ENERGY—4	float	[Ws]	Energieverbrauch des Aggregats zur Bewegung in c-Achse in Ws (hier keine Verwendung der c-Achse daher immer 0)
ENERGY—5	float	[Ws]	Energieverbrauch des Spindel-Aggregats in Ws (später umbenannt zu ENERGY—S, s. Abb. 4.5)
ENERGY—6	float	E[Ws]	nergieverbrauch des Aggregats zur Bewegung in b-Achse in Ws (hier keine Verwendung der b-Achse daher immer 0)
ENERGY—7	float	[Ws]	Energieverbrauch des Aggregats zur Bewegung des Werkzeugwechslers in Ws (später umbenannt zu ENERGY—T, s. Abb. 4.5)

Tabelle 4.1: Inhalt des erstellten Datensatzes (Quelle: Eigene Darstellung)

5 Stufe Eins: Aufbau eines prototypischen Modells

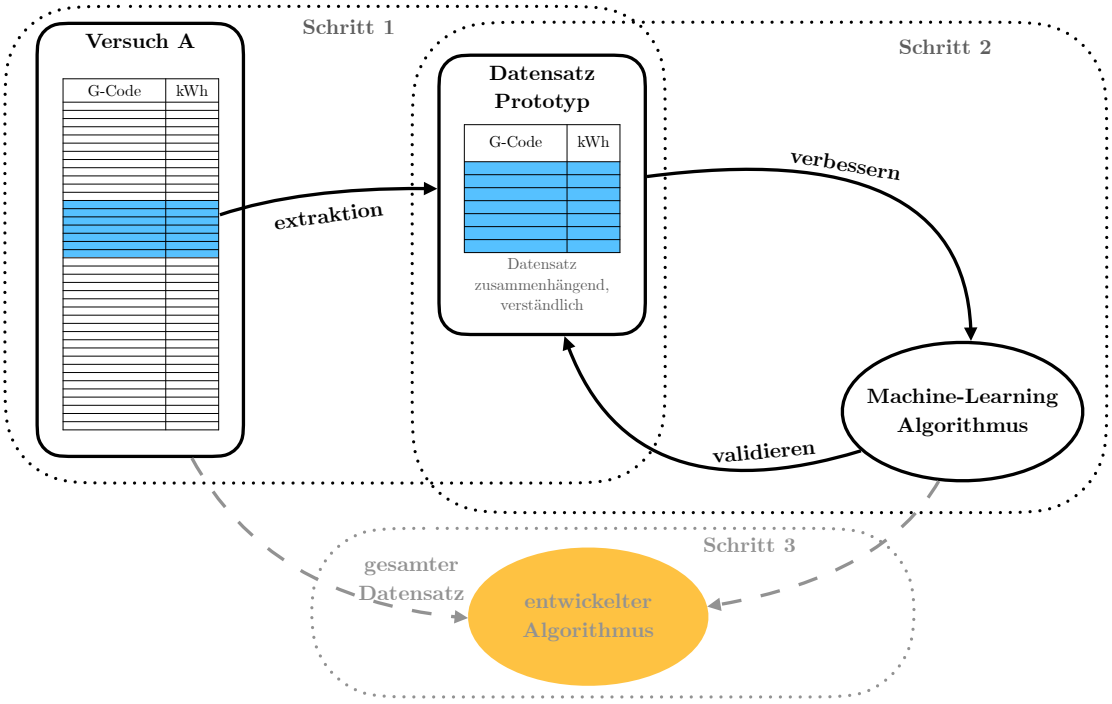


Abbildung 5.1: Prototyp-Algorithmus basierend auf einem Ausschnitt des Datensatzes (Quelle: Eigene Darstellung)

Um frühzeitig diskutierbare und erklärbare Ergebnisse zu erhalten und infolge dessen eine stringente Weiterentwicklung der Methode zu gewährleisten, wird vorerst ein Prototyp des Programmes gebaut (Bottom-up-Design, s. Kap. 3.3). Dieser basiert darauf, dass anstatt des automatischen Parsings (s. Kap. 3.2) ein Ausschnitt aus dem Datensatz (G-Code + Messungen) manuell in eine verwertbare Datenstruktur gebracht wird (s. Abb. 5.1). Dies ist um einiges schneller zu bewerkstelligen und lässt den Ansatz frühzeitig validieren. Außerdem liefert es ein besseres Verständnis des Modells und der vorhandenen Daten, wenn ein kleiner Datensatz verwendet wird, anstatt zu Beginn das Modell mit vielen Daten zu trainieren, bei denen die Zusammenhänge nicht ersichtlich sind.

5.1 Parsing und Feature Extraction

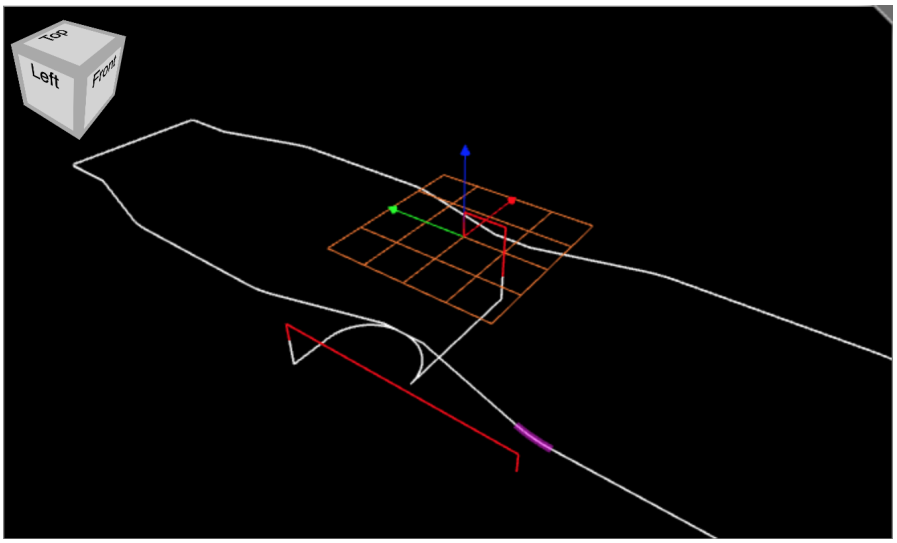


Abbildung 5.2: Visualisierung des Arbeitsganges der ausgewählten NC-Anweisungen für den Prototypen (Quelle: Eigene Darstellung)

5.1.1 Auswahl der Daten für Stufe Eins

Aus dem zur Verfügung stehenden Datensatz, den G-Code Befehlen und den dazugehörigen Messungen, die zur Fertigung eines bestimmten Bauteils auf einer CNC-Fräsmaschine gehören, wird eine zusammenhängende Auswahl an Befehlen und den dazugehörigen Energieverbrauch-Messungen ausgewählt (s. Abb. 5.1). Um eine stringente Auswahl zu treffen, wurden die Datenreihen zwischen zwei »G0«-Befehlsblöcken ausgewählt. Das bedeutet, dass sich zu Beginn der Auswahl das Werkzeug außerhalb des Materials befindet, anschließend das Werkzeug zugestellt wird, eine Kontur bearbeitet wird und das Werkzeug anschließend wieder aus dem Material herausfährt. Somit erhält man eine kondensierte Darstellung eines Bearbeitungsvorganges in einem kompakten Datensatz. Nicht berücksichtigt werden in diesem Datensatz folgende Elemente des Gesamtprozesses des CNC-Maschine:

- Die Aufwärmphase der Maschine mit allen vorbereitenden Prozessen
- Die Arbeitsschritte, die bei einem Werkzeugwechsel anfallen
- Der energierelevante Prozess des Anlaufens und Abbremsens der Werkzeugspindel
- Der Einfluss verschiedener Werkzeuge und Bearbeitungsmodi

Konkret wurden für den ersten Prototypen die NC-Anweisungen mit der Identifikationsnummer N290 bis N396 ausgewählt. Eine Visualisierung dieses

Bearbeitungsschrittes ist in Abbildung 5.2 zu sehen. Die roten Linien stellen dabei Verstellungen des Werkzeugs ohne Materialabtrag dar, beim Abfahren der weißen Linie findet hingegen eine Fräsbearbeitung statt.

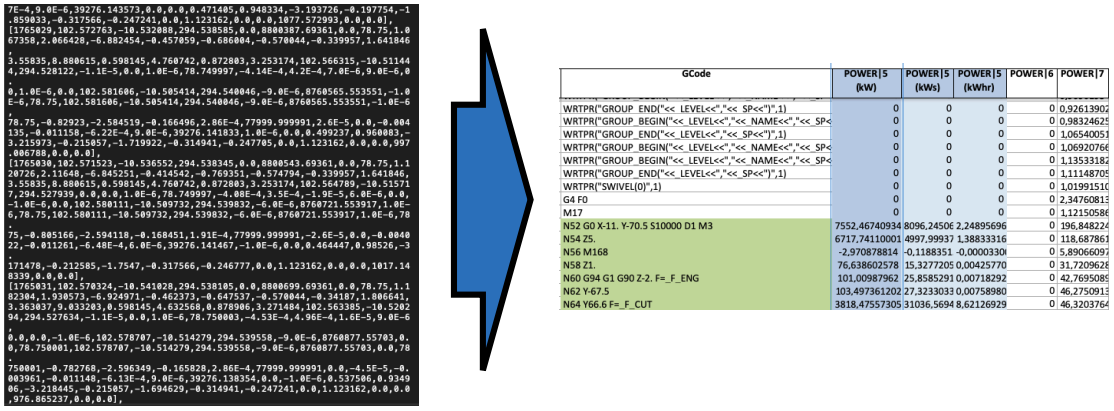


Abbildung 5.3: Überführen der rohen Daten (links) in eine Tabelle mit den NC-Anweisungen und dem dazugehörigen Energieverbrauch (Quelle: Eigene Darstellung)

5.1.2 Formatieren der Daten

Die Daten der NC-Anweisungen mit den zugehörigen Energieverbrauchsmessungen müssen aus dem rohen .json-Format in verarbeitbares Format umgewandelt werden (s. Abb. 5.3). Die NC-Anweisung ist in dieser Form noch nicht verwertbar und muss während des Parsing in seine einzelne Bestandteile unterteilt werden. Anschließend werden im so genannten Feature Extraction aus diesem Datensatz die Charakteristika herausgearbeitet werden. Dies geschieht für den kompakten Datensatz des Prototypen manuell in eine Excel-Tabelle, welche dann als .csv-Datei für die weitere Bearbeitung exportiert wird. Die ersten Einträge des Datensatzes sind in Tabelle 5.1 wiedergegeben.

index	G	X	Y	Z	I	J	F	E
	$[-]$	$[mm]$	$[mm]$	$[mm]$	$[mm]$	$[mm]$	$[\frac{mm}{min}]$	$[Ws]$
N288	0	0.0	0.0	0.0	NaN	NaN	0	0
N290	0	0.0	0.0	3.0	NaN	NaN	0	17
N292	0	0.0	-5.0	3.0	NaN	NaN	0	43
N294	0	0.0	-5.0	-3.0	NaN	NaN	0	16
N296	1	0.0	-5.0	-6.0	NaN	NaN	1000	25

Tabelle 5.1: Ersten Einträge des Datensatzes (Quelle: Eigene Darstellung)

5.2 Feature Engineering

Zu Beginn des Feature Engineerings steht nun ein formatierter Datensatz zur Verfügung (s. Tabelle 5.1). Würden wir das Machine Learning-Model mit diesen Daten trainieren, könnte kein vernünftiges Ergebnis erzielbar sein, da die bisherigen Charakteristika oder Features nicht im Zusammenhand mit der gesuchten Größe, dem Energieverbrauch stehen. Es ist beispielsweise trivial, dass der Energieverbrauch nicht von der X und Y-Position abhängen kann, also beispielsweise auf einer Seite der Maschine mehr Energie verbraucht wird als auf der anderen. Von der anderen Seite kann man aus dem Verständnis des Fräsprozesses schließen, dass der Energieverbrauch von der verrichteten Zerspanungsarbeit abhängig ist

5.2.1 Verrichtete Arbeit

Wie in Abschnitt 2.1 dargestellt werden in der Literatur meist die verschiedenen Leistungsniveaus der Maschine unterschieden und charakterisiert (z.B.: P_{cut}, P_0 usw.). Für unseren Fall ist eine andere Herangehensweise zielführender: Da bekanntlich gilt Arbeit $=$ Kraft $*$ Weg, kann der Energieverbrauch aus zwei Komponenten errechnet werden.

Kraft

Neben Reibungskräften in der Maschine und den zu überwindende Masseträgheiten ist der Hauptträger hier die auf das Werkzeug wirkende Schnittkraft. Diese ist errechnet sich mit

$$F_c = f(v_F, A, k_c, c_S) \tag{5.1}$$

wobei:

v_F . . . Vorschub
A . . . Spanungsquerschnitt
k_c . . . spezifisch Schnittkraft des Materials
c_S . . . Schnittgeschwindigkeit

Das heißt, die Kraft bleibt während eines Bearbeitungsvorganges konstant, wenn angenommen wird, dass:

- Die Vorschubgeschwindigkeit konstant bleibt. Da der Vorschub nur beim Werkzeugwechseln von der Steuerung geändert wird, ist diese Voraussetzung für jeweils ein Werkzeug gegeben.
- Der Spanungsquerschnitt konstant bleibt
- Das gleiche, homogene Material bearbeitet wird. Da nur ein Material bearbeitet wird, ist auch diese Voraussetzung erfüllt.
- Die Schnittgeschwindigkeit konstant bleibt. Sowohl Vorschub, wie Drehzahl der Spindel sind für ein Werkzeug und Bearbeitungsmodus konstant, somit ist auch diese Voraussetzung erfüllt.

Wenn man zusätzlich annimmt, dass der Spanungsquerschnitt für ein Werkzeug W_i konstant ist, gilt

$$F_c = f(W_i) \tag{5.2}$$

mit:

$$W_i \ldots \text{verwendetes Werkzeug}$$

und somit, dass die Schnittkraft lediglich eine Konstante des jeweiligen Werkzeugs ist.

Annahme konstanter Spanungsquerschnitt

Der Spanungsquerschnitt variiert während der Bearbeitung mit einem Werkzeug. Der Spanungsquerschnitt wird bei einer CNC-Bearbeitungsmaschine maßgeblich während der Erstellung der NC-Answeisungen festgelegt. Diese werden von einem CAM-Programm auf Basis der fertigen Geometrie und der Geometrie des Rohmaterials erstellt. Die Annahme wird getroffen, dass die CAM-Software den Werkzeugpfad in solcherweise plant, dass große Teile der Bearbeitung mit einem für das Werkzeug konstanten Spanungsquerschnitt ausgeführt werden.

Linearisierung des Wegs in der xy-Ebene

Mit den Erkenntnissen und Vereinfachungen aus Kapitel 5.2.1 folgt, dass die Schnittkraft für ein gegebenes Werkzeug konstant ist, bleibt zur Lösung der Gleichung Arbeit = Kraft $*$ Weg nur noch die Weg-Variable. Da diese als Feature nicht im Datensatz zur Auswertung zur Verfügung steht (s. Tbl. 5.1), muss dieses zuerst aus den verfügbaren Daten, sprich den absoluten Koordinaten, konstruiert werden. Linearisiert man die Kreisbewegungen des Werkzeugs, die

durch die G2 und G3-Befehle ausgelöst werden, gilt für den Weg Δs_i, der durch eine beliebige NC-Anweisung i verursacht wird:

$$\Delta s_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{5.3}$$

mit:

$$\begin{aligned} \Delta s_i &\ldots \text{Wegänderung in x,y-Ebene} \\ P(x_{i-1} \mid y_{i-1}) &\ldots \text{Startposition des Werkzeugs} \\ P(x_i \mid y_i) &\ldots \text{Endposition des Werkzeugs} \end{aligned}$$

5.2.2 Evaluierung des konstruierten Features

Ob die Konstruktion des zusätzlichen Features sinnvoll war, lässt sich noch vor dem Trainieren des Modells, grafisch überprüfen. Trägt man den Verfahrweg und die verbrauchte Energie einer jeden NC-Anweisung in einem Diagramm auf, lässt sich bereits eine starke Korrelation erkennen (s. Abb. 5.4). Weiter kann man im unteren, rechten Quadranten des Plots singuläre Datenpunkte erkennen, welche nicht dem globalen Trend folgen.

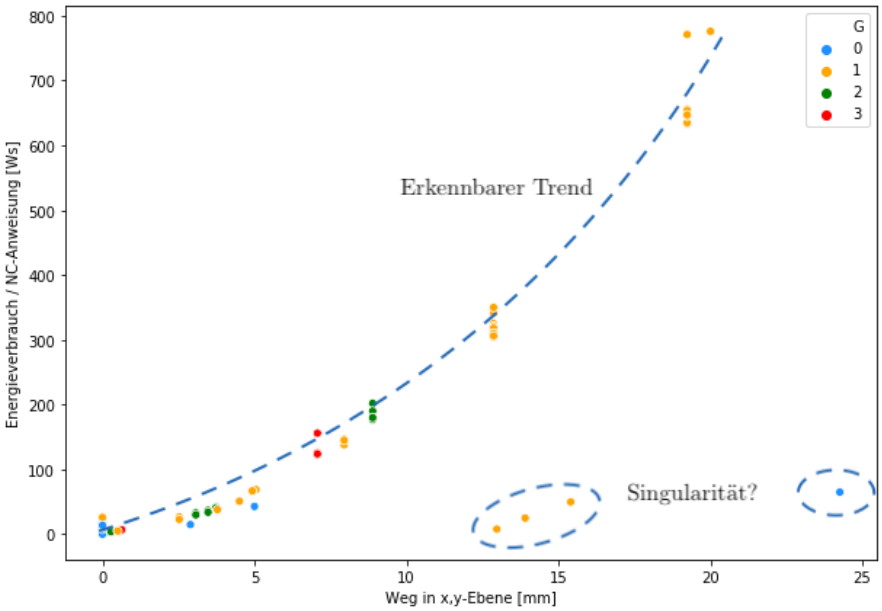


Abbildung 5.4: Plot der Datenpunkte nach dem Feature Engineering (Quelle: Eigene Darstellung)

Wegänderung in z-Richtung

Es ist aufgrund der Maschinenbauweise davon auszugehen, dass Wegänderungen Δz_i in z-Richtung einen fundamental differenzierten Energieverbrauch hervorrufen als Wegänderungen Δs_i in der xy-Ebene. Die mathematische Konstruktion und die Programmierung unterscheiden sich allerdings nicht wesentlich von Abschnitt 5.2.1.

5.3 Model Training

Mit dem geordneten, bereinigten und durch die konstruierten Features erweitertem Datensatz lässt sich schlussendlich das Machine Learning-Model trainieren.

Zielvariable und Charakteristika

Für das Trainieren des Models ist vorerst die Definition der Zielvariable, hier trivialerweise der Energieverbrauch des NC-Anweisungsblocks und außerdem eine Auswahl der einzubeziehenden Features zu erfolgen. Letzteres war Gegenstand mehrerer Tests, da eine höhere Anzahl an Features nicht unbedingt bessere Ergebnisse bedeuten. Ergebnis dieses Arbeitsschrittes ist ein Dataframe X, welches alle ausgewählten Features enthält, und ein separates Dataframe y mit der Zielvariablen.

Aufteilen in Trainings- und Testdaten

Weiterhin muss von den vorhandenen Daten ein kleiner Datensatz $(train_X, train_y)$ separiert. Dies dient dazu, das trainierte Model im Nachhinein verlässlich zu testen. Mit diesem Zwischenschritt stellt man sicher, dass das Model die Daten mit denen es getestet wird, nicht schon »kennt « und somit die Vorhersagen sehr genau getroffen werden können.

[7]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


td_cleaned = td.iloc[1:,]
y = td_cleaned.Energy_consumed_Ws
```

```
td_features = ['G', 'traveled_distance', 'F'] #  'delta_Z',␣
 ↪'delta_F', if included results get worse. F is is feed speed␣
 ↪and get's ramped up every time anyway
X = td_cleaned[td_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size␣
 ↪= 0.1, random_state = 2)
```

Im Anschluss wird das Model trainiert. Nach Validierung mehrerer Methoden hat sich herausgestellt, dass für die vorliegende Anwendung der populäre Random Forest-Algorithmus am besten geeignet ist und die besten Vorhersagen liefert. Nachdem der Datensatz ausreichend analysiert wurde, wird in Kapitel 6.6 genauer auf die verschiedenen Algorithmen und die Qualität deren Vorhersagen eingegangen.

[8]:
```
#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)
```

5.4 Prediction

Die Vorhersage (Prediction) kann dank der Implementiereung in das SCIKIT-LEARN-Paket denkbar einfach durch die .predict(X)-Methode ausgeführt werden. Beim Ausführen der Methode werden für alle NC-Anweisungen im Test-Datensatz (s. Kapitel 5.3) mit dem trainierten Model Vorhersagen errechnet und ausgegeben. Für eine bessere Vergleichbarkeit wird anschließend für jede NC-Anweisung manuell die Abweichung vom tatsächlichen Ergebnis errechnet und zu guter Letzt der gemittelte absolute Fehler in *Ws*, der Durchschnitt der prozentualen Abweichungen und die prozentuale Gesamtabweichung ausgerechnet und angezeigt.

[9]:
```
#make predictions and make a table to compare results to data

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
```

```
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.
 ↪meassured[i]) / comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)

mae = round(mean_absolute_error(val_y, val_predictions), 2)

sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n---------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'----------------------------------\n')
```

5.4.1 Bewertung der Vorhersagen

Die Qualität der Vorhersagen schwankt stark mit der Auswahl der Testdaten. In den meisten Testdatensätzen werden, wie nachfolgend zu sehen, sehr gute Vorhersagegenauigkeiten mit einer Gesamtabweichung von ca. 4 % erzielt:

index	G	traveled_distance	meassured	predictions	deviation
N370	1	12.867360	312	322.509571	0.033685
N352	2	3.732904	39	40.615353	0.041419
N382	1	2.524000	23	24.279770	0.055642
N408	3	0.632001	7	9.018373	0.288339

```
N446   2           3.471682         36    35.500884  -0.013864
N438   2           3.071627         33    29.190056  -0.115453
N420   1          12.867360        304   322.509571   0.060887
N378   1          12.867360        325   322.509571  -0.007663
N450   2           3.471682         36    35.500884  -0.013864
N460   1          12.867360        306   322.509571   0.053953
-------------------------------------
mean absolut error: 5.77 Ws.
mean of deviations:  6.85%
mae/mean_meassured:  4.06%
-------------------------------------
```

Eine Gesamtabweichung von ca. 4 % stellt bereits ein gutes Ergebnis dar. Bei einigen Testdatensätze fällt die Gesamtabweichung der Vorhersagen mit 16 % jedoch deutlich schlechter aus, wie zum Beispiel:

```
       G     distance    meassured  predictions  deviation
index
N362   1     20.006000         776   661.927980  -0.147000
N346   1     12.867360         315   320.398381   0.017138
N398   1     13.906000          25   127.055513   4.082221
N336   2      3.732904          42    40.093421  -0.045395
N322   1     20.006000         775   661.927980  -0.145899
N450   2      3.471682          36    35.582191  -0.011606
N294   0      0.000000          16    12.827417  -0.198286
N340   2      0.276921           4     4.167031   0.041758
N458   2      8.888127         180   187.508198   0.041712
N316   2      0.276921           4     4.167031   0.041758

-------------------------------------
mean absolut error: 34.79 Ws.
mean of deviations:  47.73%
mae/mean_meassured:  16.01%
-------------------------------------
```

Der Grund für diese erhöhte Gesamtabweichung liegt an der hohen Abweichung einzelner Vorhersagen (Ausreißer, s. N398). Es gilt nun im nachfolgenden Schritt die Vorhersagen, insbesondere dieser Ausreißer, zu verbessern.

6 Schritt Zwei: Verbesserung des prototypischen Models

Wie in Abschnitt 5.4.1 dargelegt, können mit dem Prototypen (Bottom-up-Design) für den ausgewählten Datensatz (s. Kapitel 4.1) bereits sehr gute Ergebnisse erzielt werden (Gesamtabweichung des Test G-Codes von $\sim$ 4%). Im Machine Learning werden üblicherweise derart geringe Ausreißer, wie sie während des Fräsprozesses zu finden sind, als Ungenauigkeit (Noise) hingenommen und nicht weiter behandelt. In Anbetracht des Projektziels, die Leistungsspitzen vorherzusagen, sind aber genau diese Ausreißer und eine zuverlässige Erkennung dieser von Interesse (s. Kapitel 1). Um dieses zu Erreichen, sind nachfolgende Schritte nötig:

Isolierung der Ausreißer Alle Ausreißer in der Vorhersage müssen ausfindig gemacht werden.

Untersuchung auf mögliche vorhandene Muster Die Ausreißer und deren unmittelbare Umgebung müssen genau untersucht werden um den Zusammenhang in Hinblick auf den Fräsprozesses zu verstehen.

Erarbeiten eines Lösungsansatzes Mit den Erkenntnissen aus der Untersuchung der Ausreißer können Hypothesen über deren Ursachen aufgestellt werden.

Implementierung einer Mustererkennung Sind die Gründe für die Ausreißer bekannt, können diese mit Hilfe von neuen Charakteristika (Features) für den Machine Learning-Algorithmus erkennbar gemacht werden. (s. Kapitel 5.2)

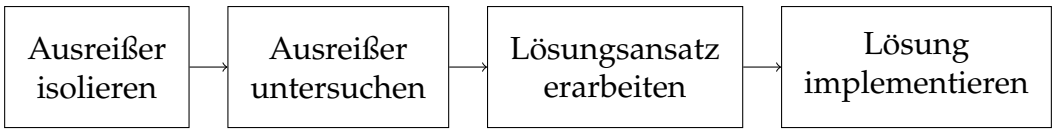


Abbildung 6.1: Vorgehen zur Implementierung einer Mustererkennung (Quelle: Eigene Darstellung)

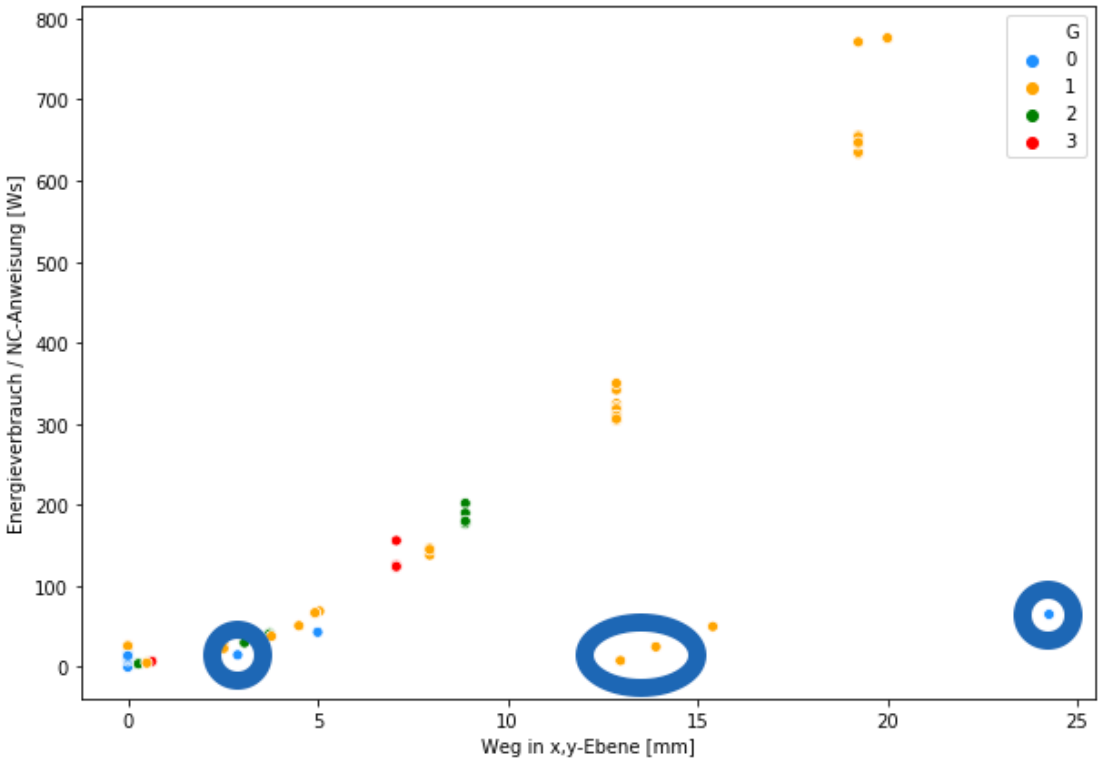


Abbildung 6.2: Plot der Datenpunkte mit markierten Ausreißern (Quelle: Eigene Darstellung)

6.1 Untersuchung der Ausreißer

Im dargestellten Testdatensatz in Abschnitt 5.4.1 weicht die Vorhersage der NC-Anweisung N398 vom tatsächlichen Energieverbrauch um über 400% ab. Im gesamten Datensatz gibt es vier weitere Vorhersagen, welche eine Abweichung von über 100% aufweisen:

- N394
- N398
- N400
- N478

Diese vier Datenpunkte decken sich teilweise mit den visuell-erfassbaren Ausreißern der sichtbaren Regression des Datensatzes. Die Ausreißer sind in Abbildung 6.2 rot markiert.

6.2 Häufigkeit und Muster der Ausreißer

Um nun den Grund für die Anomalie der in 6.1 gefunden NC-Blöcke herauszuarbeiten, ist eine Untersuchung der Datenpunkte in ihrem näheren Umfeld

nötig, sprich die Untersuchung des Fräsprozesses direkt vor und nach der Anomalie. Für drei dieser Datenpunkte ist die Datenumgebung in Tabelle 6.1 wiedergegeben.

N	G	ΔF	Δz	ΔWeg	E	$\frac{E}{\Delta \text{Weg}}$
	$[-]$	$[\frac{mm}{min}]$	$[mm]$	$[mm]$	$[Ws]$	$[\frac{Ws}{mm}]$
386	3	0,0	0,0	7,1	123,0	17,4
388	1	0,0	0,0	4,5	51,0	11,3
390	1	0,0	3,0	0,0	27,0	NaN
392	0	-1000,0	2,0	0,0	12,0	NaN
394	0	0,0	0,0	24,3	65,0	2,7
396	0	0,0	-2,0	0,0	8,0	NaN
398	1	1000,0	0,0	13,9	25,0	1,8
400	1	0,0	0,0	13,0	8,0	0,6
402	3	0,0	0,0	7,1	126,0	17,8
404	1	0,0	0,0	12,9	321,0	24,9
406	2	0,0	0,0	3,1	31,0	10,1
408	3	0,0	0,0	0,6	7,0	11,1

Tabelle 6.1: Unmittelbare Umgebung dreier Ausreißer (rot) (Quelle: Eigene Darstellung)

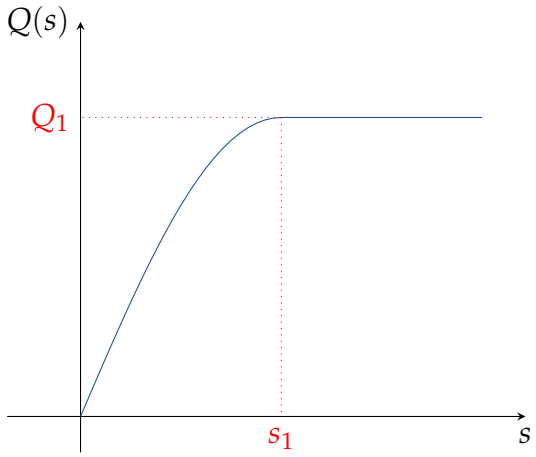
Aus den Daten in Tabelle 6.1 lassen sich zwei Gruppen erkennen:

- N394
- N398 und N400

Diese zwei Untergruppen werden im Weiteren untersucht um ein eventuelles Muster zu erkennen:

6.2.1 Untergruppe N394

Der NC-Block N394 benötigt weniger Energie als das Model vorhersagt. Hierbei handelt es sich um einen $G0$-Befehl, das Werkzeug führt eine Positionierbewegung in der x, y-Ebene aus, trägt also kein Material ab. In der vorangehenden NC-Anweisung findet noch eine Änderung der Vorschubgeschwindigkeit (N392: $\Delta F = -1000 \frac{mm}{min}$). Diese hat aber nur Einfluss auf materialabtragende Anweisungen ($G1, G2, G3$, usw.), dies kann daher nicht der Grund für die Anomalie sein.



Quelle: Eigene Darstellung

Abbildung 6.3: Zeitspanvolumen bei einem idealisierten Einfahrprozess (Quelle: Eigene Darstellung)

Muster

Der NC-Block N394 gehört nicht zu einem erkennbaren Muster. Vielmehr kann das Modell keine vernünftige Vorhersage treffen, da im Datensatz keine vergleichbare Positionierbewegung in der x, y-Ebene vorhanden ist. Befindet sich der Datenpunkt N394 also im Test-Datensatz, hat das Modell keinen annähernd vergleichbaren Datenpunkt im Trainingsdatensatz zur Verfügung. Die Lösung dieses Problems könnten mehr Daten sein, welche in Übereinstimmung mit dem besprochenen Vorgehen (s. Kapitel 3.3) später hinzugefügt werden.

6.2.2 Untergruppe N398 und N400

Auch die beiden NC-Anweisungen N398 und N400 benötigt weniger Energie als das Model vorhersagt. Hierbei handelt es sich um materialabtragende Bewegungen ($G1$). Außerdem wird die Vorschubgeschwindigkeit in N398 geändert.

Muster

Unmittelbar vor dieser Untergruppe befindet sich das Werkzeug in einer Positionierbewegung, also in der Luft und nicht im Material. Die betrachteten Befehle sind die ersten Anweisungen zum Materialabtrag. Anders als beim regulären Materialabtrag kann das Zeitspanvolumen Q nicht konstant sein sondern steigt mit dem eindringen des Werkzeugs bis zu einem als konstant angenommenen Zielwert an (s. Abschnitt 5.2.1 und Abb. 6.3 und 6.4). In einem idealisierten Einfahrprozess steigt das Zeitspanvolumen bis s_1 an und bleibt anschließend über den gesamten Bearbeitungsprozess konstant (s. Abb. 6.3).

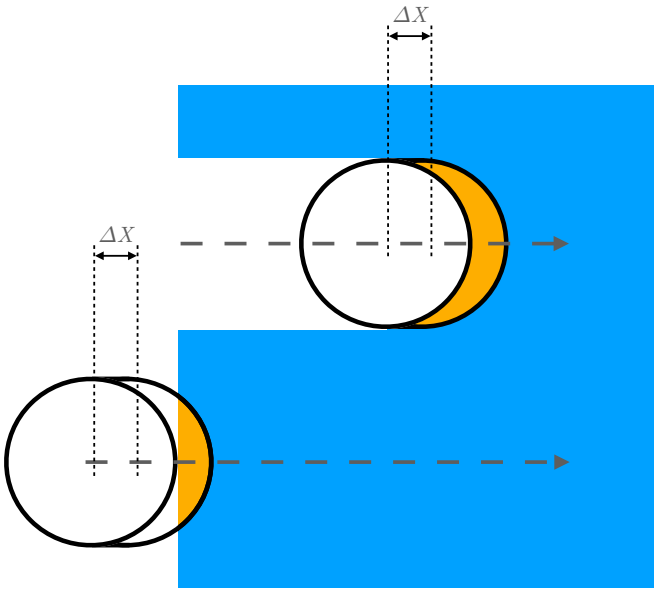


Abbildung 6.4: Gegenüberstellung: Regulärer Materialabtrag (oben) und Eintauchvorgang (unten) (Quelle: Eigene Darstellung)

$$E_{Einfahren} = \int_{s_0}^{s_1} Q(s)\, ds \tag{6.1}$$

$$\begin{aligned} E_{Materialabtrag} &= \int_{s_1}^{s} Q_1\, ds \\ &= Q_1 * \Delta s \end{aligned} \tag{6.2}$$

mit:

$$\begin{aligned} s_0 &\ldots \text{Start des Materialabtrags} \\ s_1 &\ldots \text{Beginn des regulären Materialabtrags} \end{aligned} \tag{6.3}$$

6.3 Implementierung einer Mustererkennung

Jene Datenpunkte, welche zu dem Eintauchprozess des Werkzeugs ins Werkstück gehören, müssen speziell hervorgehoben werden um so dem Machine Learning-Algorithmus eine Unterscheidung zu ermöglichen. Dazu wird ein neues Feature (Charakteristika) für alle Datenpunkte eingeführt (s. Abschnitt 5.2).

Umsetzung

Wie kann nun unterschieden werden, ob eine NC-Anweisung während eines Eintauchprozesses ausgeführt wird oder während der regulären Bearbeitung? Ein Lösungsansatz dazu ist, in einer Laufvariablen x_{lauf} alle Verfahrwege des Werkzeugs aufzusummieren und diese Laufvariable jeweils bei einer Go-Anweisung (nicht materialabtragend, Bewegung des Werkzeugs in der Luft) auf $x_{lauf} = 0$ zurück zu setzen. Für NC-Blöcke, für welche die Laufvariable kleiner ist, als ein werkzeugspezifischer Grenzwert $x_{lauf} < x_{grenz}$, kann angenommen werden, dass sie während eines Tauchvorgangs ausgeführt werden. Der Grenzwert wird anhand von empirischen Daten aus dem Datensatz geschätzt. Dem Feature, wird für diese Anweisungen der Wert 1 zugewiesen, in allen anderen Fällen der Wert 0.

[74]:
```
#neue Spalte für konstruiertes Feature
ramp_up = []
ramp_up.append(0)
total_distance = 0
grenz_distance = 15

#Klassifizieren der Datenpunkte: Eintauchen oder reg.␣
 ↪Materialabtrag
for i in range(1, len(td.index)):
    if total_distance <= grenz_distance:
        if td.G.iloc[i] == 0:
            ramp_up.append(0)
        else:
            ramp_up.append(1)
    else:
        ramp_up.append(0)

    if td.G.iloc[i] == 0:
        total_distance = 0
    else:
        total_distance += td.traveled_distance.iloc[i]

try:
    td['ramp_up'] = ramp_up
except:
    pass
```

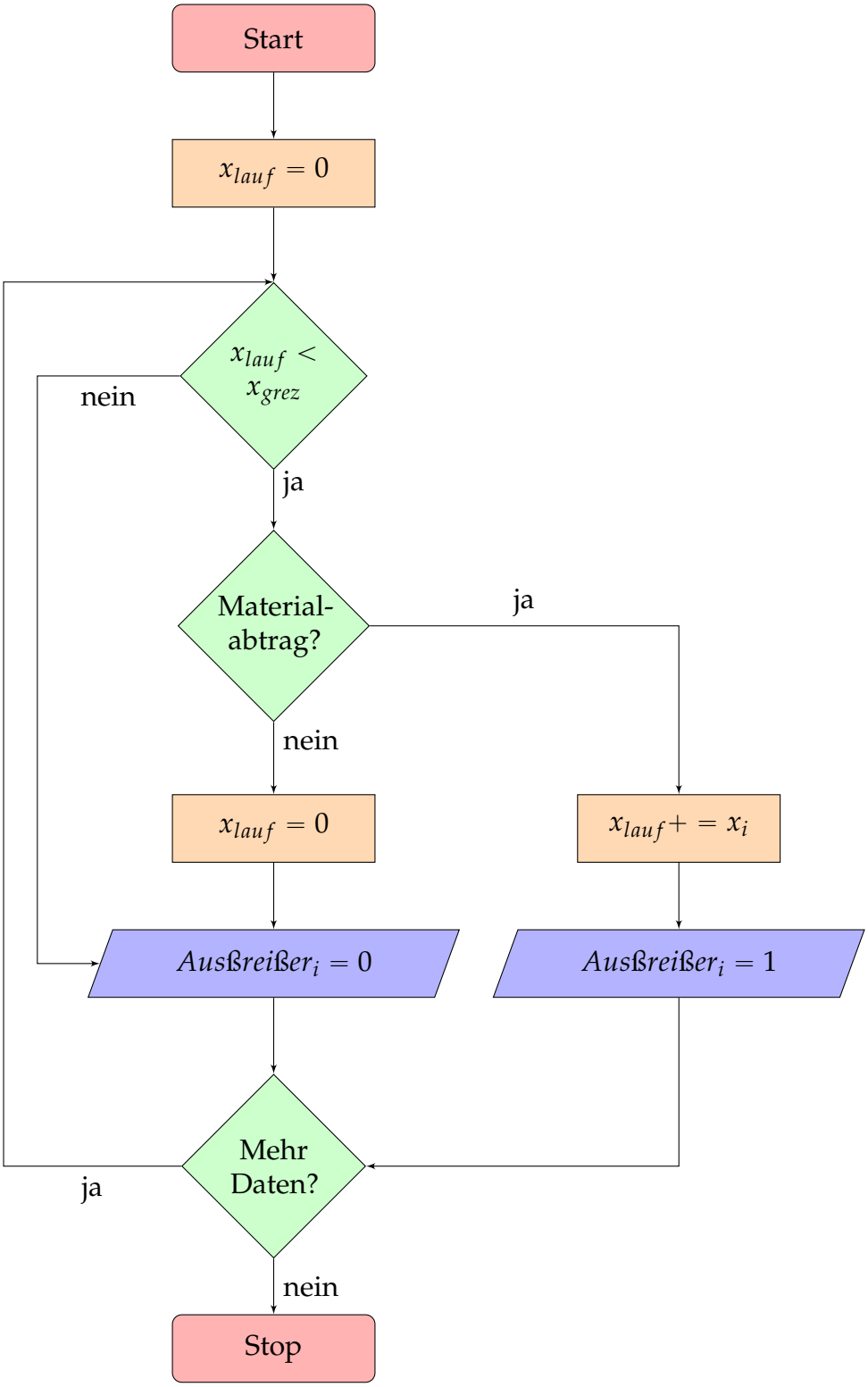


Abbildung 6.5: Grundmechanismus zum Markieren von Datenpunkten, welche in das Muster des Eintauchvorganges fallen (Quelle: Eigene Darstellung)

Ausreißer	Fehler ohne Mustererkennung	Fehler mit Mustererkennung
N394	1013,1%	1013,1%
N398	408,2%	300,1%
N478	128,9%	127,6%
N400	3901,9%	2307,1%

Tabelle 6.2: Vergleich der Vorhersagefehler mit und ohne Mustererkennung (Quelle: Eigene Darstellung)

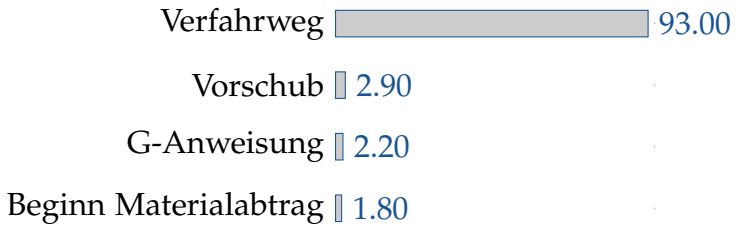


Abbildung 6.6: Feature-Gewichtungen (in %) (Quelle: Eigene Darstellung)

6.4 Bewertung der Mustererkennung

Ähnlich wie für das erste konstruierte Charakteristika (s. Kapitel 5.2) muss auch die Mustererkennung in ihrer Effektivität überprüft werden. Dazu wird der prozentuale Fehler der Vorhersage im Verhältnis zum tatsächlichen Energieverbrauch der NC-Anweisung für die Vorhersagen mit und ohne Mustererkennung überprüft. Der Vergleich ist in Tabelle 6.2 zusammengefasst.

Es zeigt sich, dass die Mustererkennung bei zwei der Ausreißer (N398, N400) zu signifikanten Verbesserungen der Vorhersage führt. Die beiden anderen Vorhersagen (N394, N478) verbessern sich dank der Mustererkennung nicht merklich bzw. überhaupt nicht.

6.5 Verbesserung der Mustererkennung

Da die korrekte Vorhersage des Energieverbrauchs der Ausreißer besondere Aufmerksamkeit geschenkt werden muss, sind die erzielten Ergebnisse der Mustererkennung nicht ausreichend. Bei der Suche nach Ursachen der mangelhaften Vorhersagen kann die Gewichtung der Charakteristika (Features) ausschlaggebend sein. Die Untersuchung, wie der Algorithmus die Charakteristika gewichtet, ist in Abbildung 6.6 dargestellt.

Aus den Gewichtungen schließt sich, dass die Mustermarkierung (»Beginn Materialabtrag«) von allen Features am geringsten gewichtet wird und dadurch auch bei Vorhersagen nicht genügend beachtet. Die Antwort, weshalb das Feature nicht stärker gewichtet wird, muss im Algorithmus selbst gesucht werden. Daher werden im nächsten Schritt, die Eigenschaften der verschiedenen Algorithmen beleuchtet und deren Umgang mit dem vorhanden Muster der Ausreißer vergleichen.

6.6 Auswahl des Algorithmus zur verbesserten Vorhersage für Ausreißer

Durch die Verschiedenartigkeit der Machine Learning-Algorithmen ergeben sich jeweils unterschiedliche Stärken und Schwächen der einzelnen Methoden in Hinblick auf einen spezifischen Datensatz. Die Gegenüberstellung der Algorithmen erfolgt an dieser Stelle, um die Auswirkung der Ausreißer und der Mustererkennung (s. Abschnitt 6.3 mit in die Betrachtung nehmen zu können.

In Kapitel 6.6.1 werden die klassische Machine Learning-Algorithmen verglichen:

- Decision Tree
- Support Vector Machine
- k-Nearest-Neighbor

Anschließend werden in 6.6.2 Ensemble-Methoden untersucht. Diese Methoden können mit jedem klassischen Algorithmus kombiniert werden und haben das Ziel deren Ergebnisse weiter zu steigern:

- Bagging - Random Forest
- Boosting - Adaptive Boosting
- Stacking

6.6.1 Klassische Machine Learning-Algorithmen

Bewertung Decision Tree

Abbildung 6.7 stellt die tatsächlichen Daten als Punkte und die durch den Algorithmus erstellte Regressionsfunktion (»Regressionstreppe «) für Vorhersagen als farbige Linien dar. Ersichtlich wird, dass eine eindeutige Unterscheidung stattfindet zwischen der Regression für die Materialabtrag im Werkstück (blaue Linie) und der Bewegungen beim Einfahren ins Material (grüne Linie). Als Nachteil ist zu werten, dass in Bereichen, in denen wenig Daten zur Verfügung

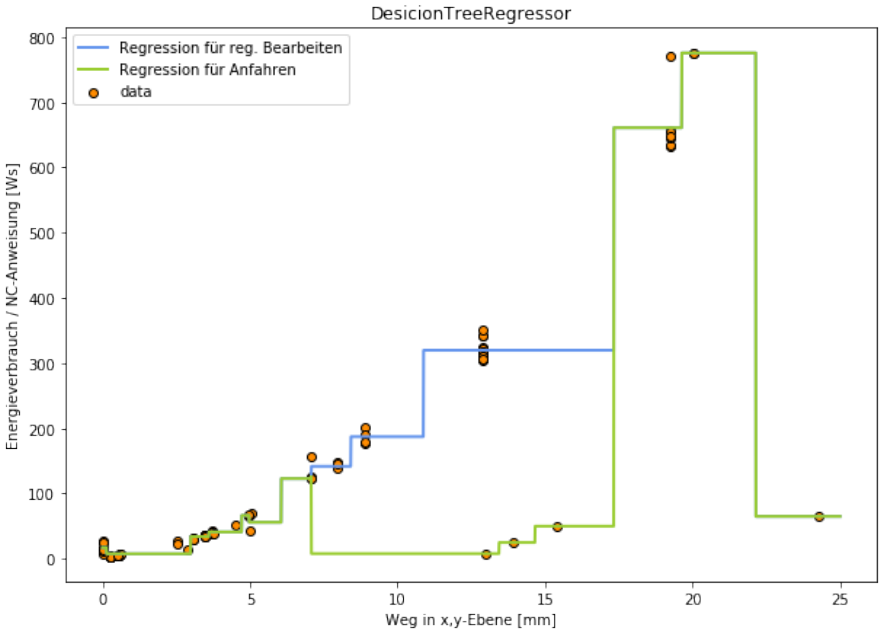


Abbildung 6.7: Regressionsfunktion Entscheidungsbaums (Quelle: Eigene Darstellung)

stehen, große Abstufungen in den Vorhersagen zu finden sind, somit auch mit großen Fehlern zu rechnen ist, falls Vorhersagen in diesem Bereich durchzuführen sind.

Bewertung k-Nearest-Neighbor

Gegenüber dem Entscheidungsbaum (Abb 6.7) führt der k-Nearest-Neighbor-Algorithmus zu einer Glättung der Vorhersagefunktionen (s. Abb. 6.8). Allerdings findet die Unterscheidung zwischen regulärem Materialabtrag (blaue Linie) und dem Einfahrprozess (grüne Linie) nicht mehr statt und beide Funktionen näher sich an.

Bewertung Support Vector Machine

In Kontrast zu den vorherigen Methoden ist das Ergebnis eine stetige Vorhersagefunktion (s. Abb. 6.9). Es findet aber keinerlei Unterscheidung zwischen den zwei untersuchten Bearbeitungsmodi statt (blaue und grüne Linie deckungsgleich). Im Bereich der Ausreißer ist eine sichtbare Ausbeulung der Funktion in deren Richtung zu beobachten. Dies ist dadurch begründet, dass der Algorithmus zum Ziel hat, die mittlere quadratische Abweichung zu minimieren.

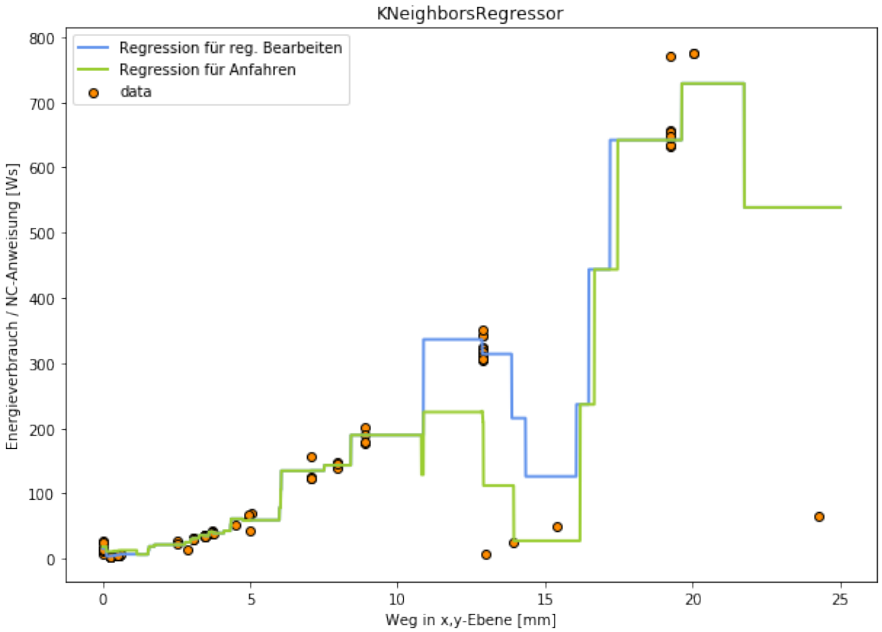


Abbildung 6.8: Regressionsfunktion k-Nearest-Neighbor (Quelle: Eigene Darstellung)

6.6.2 Ensemble Methoden

Bewertung Bagging - Random Forest

Gegenüber der Basismethode (Entscheidungsbaum, s. Abb. 6.7) ist die Entscheidungsfunktion des Random Forest (Abb. 6.10) feingliedriger. Jedoch wird auch die Vorhersage für den regulären Materialabtrag von den Ausreißern beeinflusst. In den zufällig ausgewählten Daten für den Teildatensatz gibt es eine Menge an Sätzen, welche keine Ausreißer enthalten. Diese Vorhersagen beeinflussen im Nachhinein die Gesamtvorhersage maßgeblich negativ.

Bewertung Boosting – Adaptive Boosting

Die Boosting-Methode wurde auf zwei Algorithmen angewandt: Dem Entscheidungsbaum und dem bagged Entscheidungsbaum (Random Forest). Für beide Algorithmen verbessert die Boosting-Methode die Ergebnisse: Beim einfachen Entscheidungsbaum (s. Abb. 6.12) wird der Bereich der korrekten Unterscheidung der beiden Fälle (regulärer Materialabtrag und Einfahren ins Material) gegenüber dem nicht geboosteten Entscheidungsbaum (s. Abb. 6.7) in Richtung kleinerer Wege vergrößert. Für den bagged Entscheidungsbaum (Random Forest, s. Abb. 6.10) verbessert die Boosting Methode (s. Abb 6.11) zwar die Ergebnisse auch, das grundlegende Problem dieser Methode, dass durch die Durchschnittsbildung keine klare Unterscheidung der Ausreißer stattfindet,

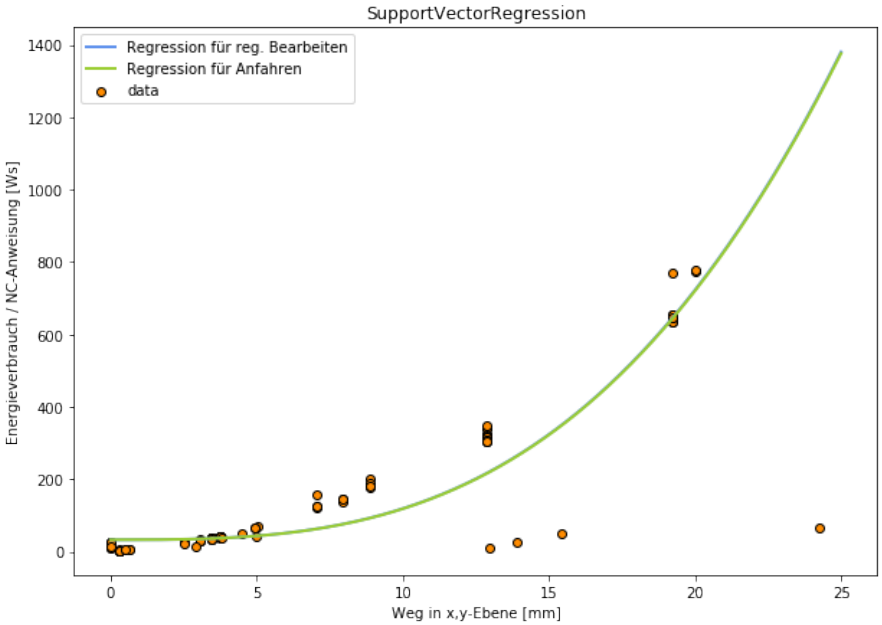


Abbildung 6.9: Regressionsfunktion Support Vector Machine (Quelle: Eigene Darstellung)

wird aber nicht gelöst (s. Kapitel 6.6.2)

6.7 Zusammenfassung der Bewertungen

Die Ergebnisse der vorhergehenden Abschnitte sind in Tabelle 6.3 zusammengefasst. Die besten Ergebnisse werden mit dem boosted Decision Tree erzielt.

Algorithmus	Regressionsfunktion	Unterscheidung der Ausreißer
DecisionTree	stark stufig	gut
k-Nearest-Neighbor	stufig	schlecht
SupportVectorMachine	stetige Polynomfunktion	keine
RandomForest	schwach stufig	schlecht
AdaBoost + DecisionTree	stark stufig	sehr gut
AdaBoost + RandomForest	stufig	schlecht

Tabelle 6.3: Bewertung der Algorithmen und Methoden (Quelle: Eigene Darstellung)

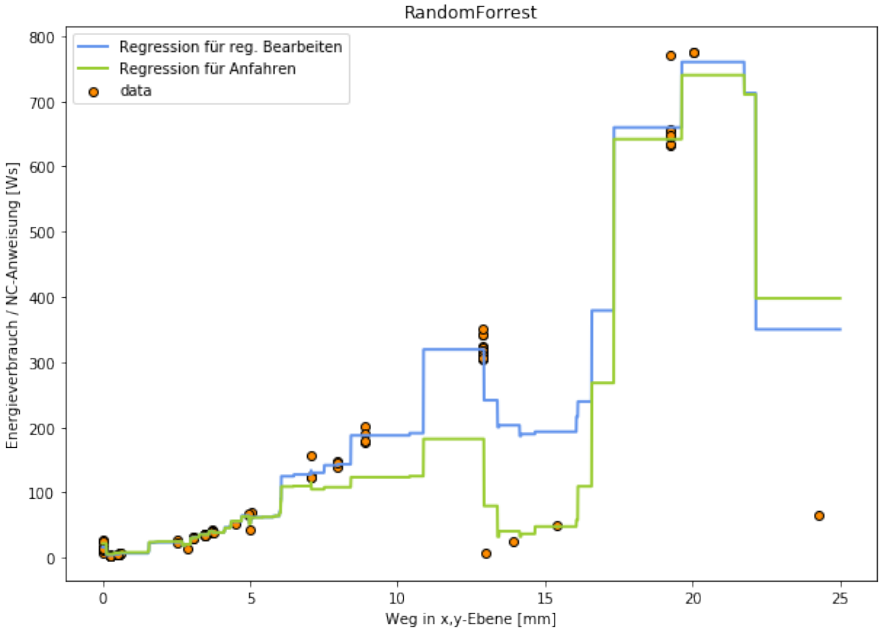


Abbildung 6.10: Regressionsfunktion Random Forest (Quelle: Eigene Darstellung)

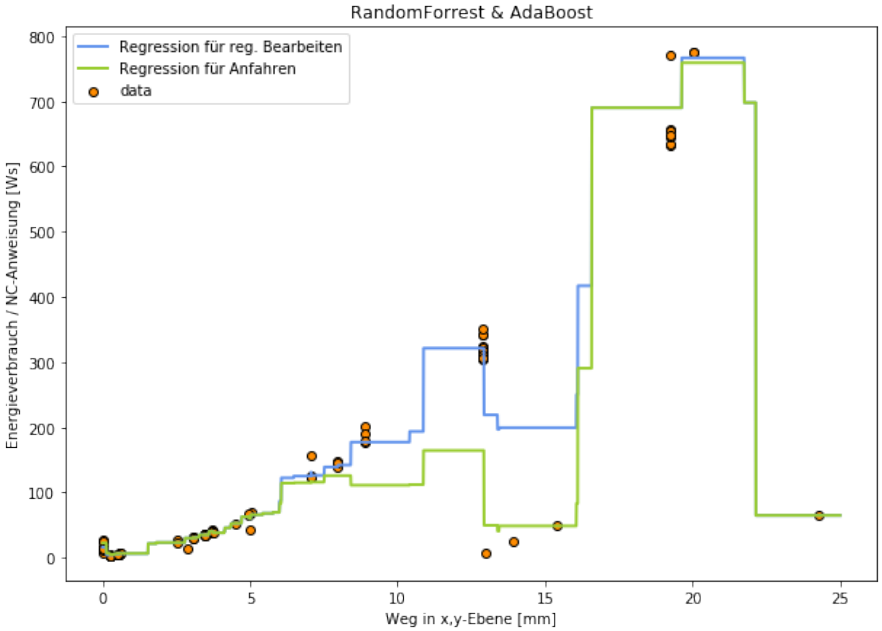


Abbildung 6.11: Regressionsfunktion Boosted Random Forest (Quelle: Eigene Darstellung)

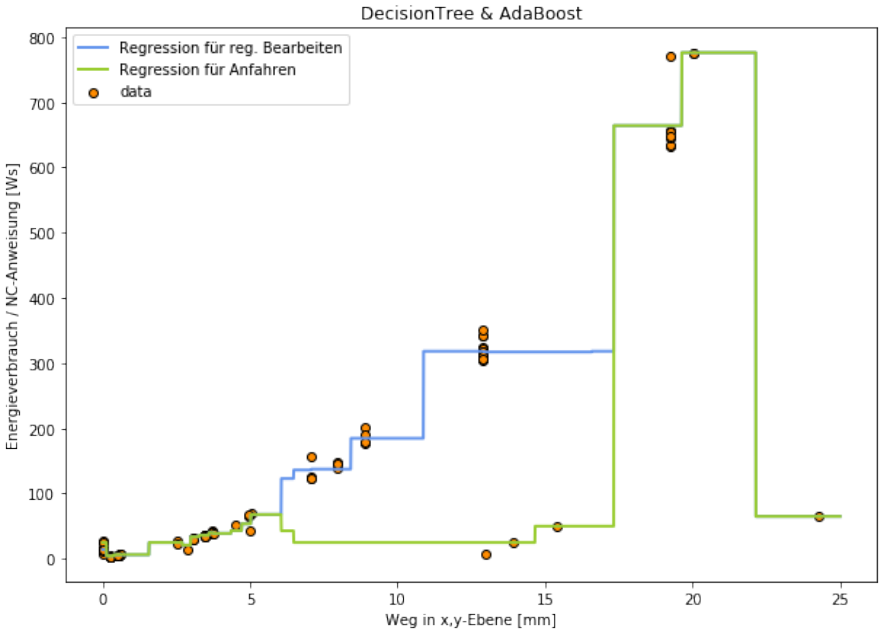


Abbildung 6.12: Regressionsfunktion Boosted Decision Tree (Quelle: Eigene Darstellung)

7 Schritt Drei: Erweiterung des Datensatzes

Die in Kapitel 5 getroffene Reduzierung des Datensatzes auf lediglich einen kleinen Ausschnitt hat in den vergangenen Kapiteln wertvolle Erkenntnisse ermöglicht. Eine weitere Verbesserung des Algorithmus kann erreicht werden, wenn nun der gesamte Datensatz zum Trainieren und Validieren des vorhandenen Models herangezogen wird (s. Kap. 3.3).

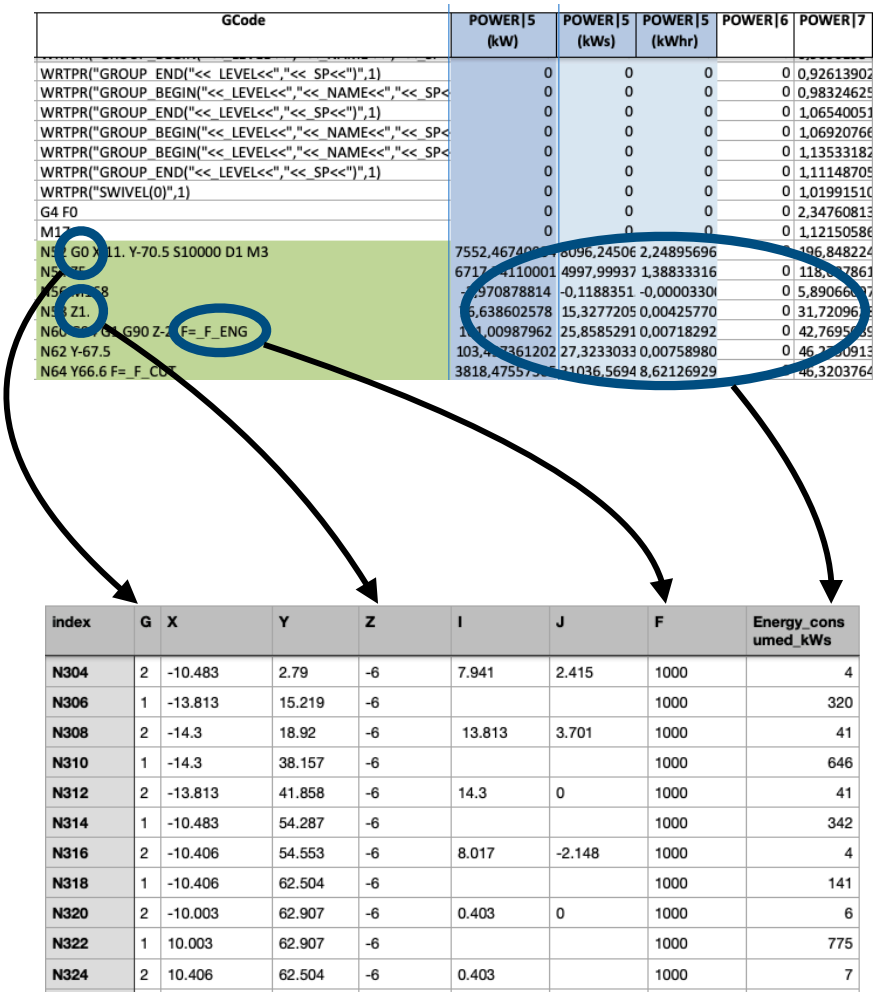
Das im vorangegangen Kapitel erstellte Model ist nicht direkt auf den neuen Datensatz anwendbar. Das Modell muss zuerst an den neuen, komplexeren Datensatz angepasst werden. Daher muss der gesamte, in Abbildung 7.1 dargestellte und in Kapitel 3.3 besprochene, Ablauf auch für das Machine Learning Programm dieses Schrittes durchlaufen werden. Mit dem Unterschied, dass nun keine Vereinfachungen mehr getroffen werden und keine händische Datenmanipulation außerhalb des Programmes vorgenommen wird.

7.1 Parsing

Die Aufbereitung und Formatierung der Daten, sodass sie von dem Machine Learning-Modell verarbeitet werden können, geschah im Prototypen für den kleinen Ausschnitt des Datensatzes manuell (s. Kap. 5.1) Diese Vorgehensweise ist für den gesamten Datensatz auf Grund der Datenmenge nicht realisierbar. Daher muss ein automatisches Computerprogramm (=Parser) entwickelt werden, welches die Zeichenketten der NC-Anweisung in seine Einzelteile untergliedert, die einzelnen Informationen sortiert und im auslesbaren Tabellenformat abspeichert (s. Abb. 7.2

Dabei bestehen mehrere Herausforderungen:

Abbildung 7.1: Einzelschritte zum Erstelen von Machine Learning-Vorhersagen (Quelle: Eigene Darstellung)



GCode	POWER|5 (kW)	POWER|5 (kWs)	POWER|5 (kWhr)	POWER|6	POWER|7
WRTPR("GROUP_END("<< LEVEL<<","<< SP<<")",1)	0	0	0	0	0,92613902
WRTPR("GROUP_BEGIN("<< LEVEL<<","<< NAME<<","<< SP<	0	0	0	0	0,98324625
WRTPR("GROUP_END("<< LEVEL<<","<< SP<<")",1)	0	0	0	0	1,06540051
WRTPR("GROUP_BEGIN("<< LEVEL<<","<< NAME<<","<< SP<	0	0	0	0	1,06920766
WRTPR("GROUP_BEGIN("<< LEVEL<<","<< NAME<<","<< SP<	0	0	0	0	1,13533182
WRTPR("GROUP_END("<< LEVEL<<","<< SP<<")",1)	0	0	0	0	1,11148705
WRTPR("SWIVEL(0)",1)	0	0	0	0	1,01991510
G4 F0	0	0	0	0	2,34760813
M17	0	0	0	0	1,12150586
N5 G0 X 11. Y-70.5 S10000 D1 M3	7552,467	8096,2450	2,24895696		196,848224
N5	6717	4997,99937	1,38833316	0	118,
N5	970878814	-0,1188351	-0,0000330	0	5,89066
N5 Z1.	6,638602578	15,3277205	0,00425770	0	31,72096
N60 G1 G90 Z-2 F=_F_ENG	1	1,00987962	25,8585291	0,00718292	0	42,76958
N62 Y-67.5	103,4	7361202	27,3233033	0,00758980	0	46,
N64 Y66.6 F=_F_CUT	3818,475	31036,5694	8,62126929			46,3203764

index	G	X	Y	Z	I	J	F	Energy_consumed_kWs
N304	2	-10.483	2.79	-6	7.941	2.415	1000	4
N306	1	-13.813	15.219	-6			1000	320
N308	2	-14.3	18.92	-6	13.813	3.701	1000	41
N310	1	-14.3	38.157	-6			1000	646
N312	2	-13.813	41.858	-6	14.3	0	1000	41
N314	1	-10.483	54.287	-6			1000	342
N316	2	-10.406	54.553	-6	8.017	-2.148	1000	4
N318	1	-10.406	62.504	-6			1000	141
N320	2	-10.003	62.907	-6	0.403	0	1000	6
N322	1	10.003	62.907	-6			1000	775
N324	2	10.406	62.504	-6	0.403		1000	7

Abbildung 7.2: Extraktion und Separation der benötigten Informationen aus dem G-Code (Quelle: Eigene Darstellung)

- Ein Großteil der Anweisungen sind interne Maschinenbefehle, deren Bedeutung weder bekannt noch für den Energieverbrauch relevant ist (s. Kap. 7.1.1).
- Es können in einer NC-Zeile mehrere unterschiedlich Befehle zusammengefasst werden (z.B. Materialabtrag und Einschalter der Kühlmittelzufuhr, s. Kap 7.1.2).
- Anweisungen können Auswirkungen auf die nachfolgenden Zeilen haben (z.B. Auswahl des Werkzeuges, s. Kap. 7.1.3).
- Der Vorschub ist nicht als numerische Information verfügbar und muss aus separaten Informationsteilen ermittelt werden (s. Kap. 7.2.1).
- Manche Befehle werden nicht direkt numerisch spezifiziert sondern enthalten Variablen, deren Wert nur maschinenintern bekannt ist (s. Kap. 7.2.2).

Diese Herausforderungen werden nachfolgend genauer betrachtet und Ansätze zur Bewältigung geboten.

7.1.1 NC-Anweisungen mit relevanten Auswirkungen

Das Parserprogramm untersucht die NC-Anweisungen auf ihren Inhalt und den verursachten Energieverbrauch und nimmt darauf hin eine Unterscheidung der NC-Anweisungen vor:

signifikante NC-Anweisung Befehl welcher Energieverbrauch verursacht oder wichtige Informationen enthält

Message Zusammenfassung aller nicht-relevanten NC-Anweisungen

In Tabelle 7.1 sind alle signifikanten NC-Anweisungen zusammengefasst.

Gleichzeitig stellt Tabelle 7.1 eine Liste aller NC-Anweisungstypen dar, welche vom Parser derart separiert und abgespeichert werden, dass der Algorithmus diese eindeutig unterschieden kann. Es sei an dieser Stelle schon auf die erheblichen Spannweiten einiger NC-Befehle hingewiesen: Dies bedeutet, dass bei identischen Code-Zeilen der Energieverbrauch beträchtlich schwankt. Der Algorithmus hat somit keine Chance, allein aus dem Informationsgehalt der NC-Zeile den Energieverbrauch richtig zu ermitteln.

7.1.2 Zusammenfassung von Befehlen

In der Logik der NC-Befehle können mehrere, voneinander unabhängige Anweisungen in einer Zeile zusammengefasst sein (s. Abb. 7.3):

N52 G0 X-11. Y-70.5 S10000 D1 M3

In dem obrigen NC-Befehl wird

Anweisung	Häufigkeit	Arithm. Mittel	Spannweite R	Anmerkung
G0 X-11. Y-70.5 S(variabler Wert) D1 M3	4	3500	7500	M3: Spindel ein
G0 Z5	5	2400	7800	-
G90 D0 SPOS=0 POS[X]=_X FA[X]=RED_RAPID_SPEED POS[Y]=_Y FA[Y]=RED_RAPID_SPEED POS[Z]=_Z FA[Z]	8	800	600	Anweisung erscheint jeweils doppelt: Das erste Mal keine Auswirkungen, beim zweiten dann signifikanter Energieverbrauch G90: Absolutbezugsmasseingabe D0: Reset Board
G1 G60 AX[_Z]=DP	6	500	1400	G60: Genauhalt fein
G54 G0 X0 Y0 Z5.	1	500	0	Einstellbare Nullpunktverschiebung
M168	3	150	430	M168: „Ignoring M-Tool Spindle Constant Speed Answer“
G94 G1 G90 Z-2. F=_F_ENG	1	140	0	G94: Vorschub mm/min, G90: Absolutbezugsmasseingabe
AX[_Z]=RFP+SDIS*(RFP-DP)/ABS(RFP-DP) G0 G90	7	110	1000	-
G0 AX[_Z]=RTP	6	105	60	-
G94 X0. Y-49.638 F24.	4	50	140	G94: Vorschub mm/min
MEAS=_M_N[0] AX[_MAV]=_M_SPE[10]*_M_SPE[29]	2	50	40	-
G0 G40 G60 G90 Z=$TC_CARR40[_TC1]-_TOOLL[2]*_FAK1	8	50	10	Anweisung wiederholt sich jeweils zwei Mal: Das erste Mal keine Auswirkungen, beim zweiten dann signifikanter Energieverbrauch
G90 AX[_YY]=_M_SPE[6]*_M_SPE[29]	1	32	0	Absolutbezugsmasseingabe
G91 AX[_ZZ]=-_ID	2	22	1	Kettenmaßeingabe
M169 M167	3	20	20	Deaktivierung Luft und Öl
D=$P_TOOL	7	13	9	-
MEAS=_M_N[0] AX[_ZZ]=_ID	2	13	1	-
G4 F0	7	5	4	Verweilzeit mit Adresse F
G4 F=_DTB	6	4	5	-
G09 AX[_MAV]=(_M_SPE[10]+_KF[2,0])*_M_SPE[29]	2	4	1	-
MEAS=_M_N[0] AX[_MAV]=_M_SPE[9]*_M_SPE[29]	2	3	1	-
M27 M28	4	2	4	-
M17	7	1	1	M17: Ende Unterprogramm Siemens
G0 Z=_Z_HOME D0	2	0	1	D0: Reset board

Tabelle 7.1: Signifikante NC-Anweisungen mit Klassifizierung des Energieverbrauchs (Quelle: Eigene Darstellung)

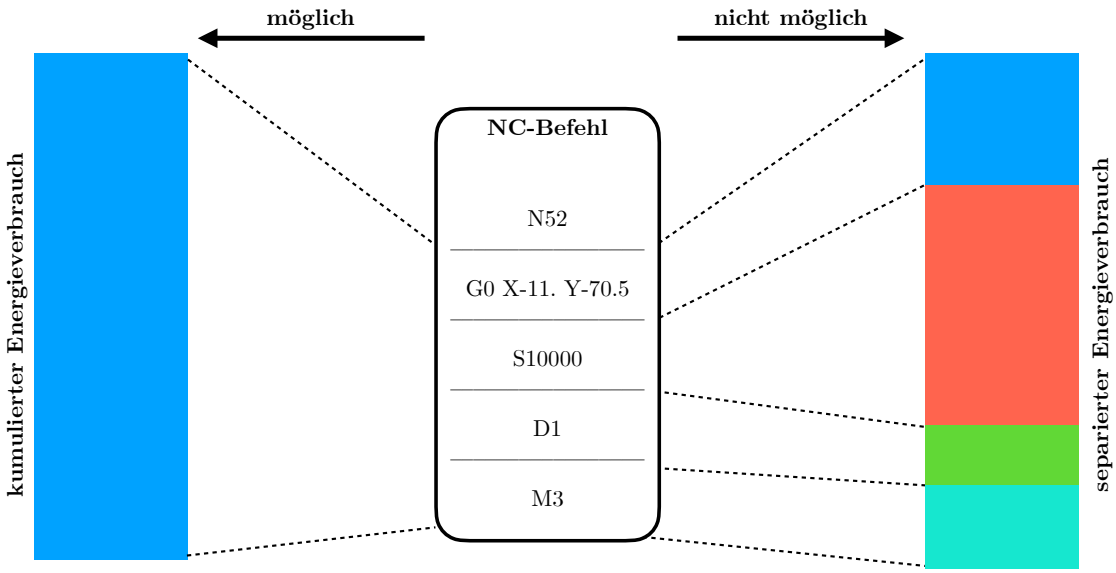


Abbildung 7.3: Aufteilung des Energieverbrauches bei zusammengesetzten NC-Befehlen (Quelle: Eigene Darstellung)

- eine Verfahrbewegung mit zugehöriger Zielposition (G0 X-11. Y-70.5),
- die Änderung der Spindelgeschwindigkeit (S10000),
- eine Speicheranweisung (D1) und
- das Festlegen der Spindeldrehrichtung (M3)

zusammengefasst. Prinzipiell bieten sich zwei Möglichkeiten zur Verarbeitung dieser zusammengefassten Befehle:

- Separierung der Information in unterschiedliche Datenpunkte
- Erstellung eines Datenpunktes mit kumulierten Informationen

Methodisch spricht einiges für den ersteren Ansatz: Einzelne Elemente können besser bearbeitet und vorhergesagt werden. So können für Verfahrbewegungen Anhang der Häufigkeit im Trainings-Datensatz sehr genaue Vorhersagen getroffen werden. Gleiches würde auch für die Änderung der Spindelgeschwindigkeit gelten, wenn entsprechende Trainingsdaten verfügbar wären. Dies ist genau der Nachteil der Separierung. NC-Befehle, wie S, D oder M, treten im verfügbaren Datensatz nur in Kombination mit anderen Befehlen auf. Eine Zuordnung des gemessenen Energieverbrauches auf die einzelnen Elemente des Blockbefehles (s. Darstellung 7.3) ist nicht möglich.

Daher muss auf die zweite, nachteilige Methode zurückgegriffen und alle Informationen in einem Datenpunkt abgelegt werden. Durch die Kumulierung der Informationen entstehen singuläre Datenpunkte, die nicht einfach mit anderen in Zusammenhang gebracht werden können. Dieser Nachteil wird sich später beim Anlernen des Algorithmus und bei den Vorhersagen bemerkbar machen.

7.1.3 Auswirkungen auf nachfolgende Zeilen

Der erstellte Machine Learning-Algorithmus betrachtet jeden Datenpunkt separat ohne Beachtung der vorangegangenen und nachfolgenden Ereignisse / Datenpunkte. NC-Befehle haben aber häufig Auswirkungen auf nachfolgende Code-Zeilen:

G-Befehle

Ein G-Befehl bleibt so lange aktiv, bis er von einem anderen abgelöst wird. In den nachfolgenden Zeilen ist ein Ausschnitt aus dem Datensatz dargestellt in welchem zu sehen ist, dass in der zweiten Zeile, keine Angabe des G-Befehles mehr notwendig ist um die lineare Bearbeitung in z-Richtung fortzuführen.

N574 G1 X-19.907
N576 Z-5.

Werkzeugauswahl

Die Auswahl eines Werkzeugs wird durch folgenden, beispielhaften NC-Befehl initiiert:

N40 T="002383_A"

Das daraufhin aufgenommene Werkzeug bleibt aber relevant für alle nachfolgenden Bearbeitungen. Der Parser muss also beim Erstellen des Datensatzes solche Informationen in die nächsten Zeilen übertragen.

Mechanismus zum Übertragen der Informationen

Zur Übertragung der Informationen, wie die oben genannten G-Befehle und die Werkzeugauswahl, in die nachfolgenden Datenpunkte erfolgt in einer Schleife durch alle Zeilen des Datensatzes. Die grundsätzliche Logik ist nachfolgend dargestellt (s. Abb. 7.4). Zwar erfordert jede Anwendung spezielle Funktionen und Abwandlungen, der im Flussdiagramm abgebildete Mechanismus stellt aber den Grundmechanismus dar, der allen Übertragungen zugrunde liegt.

Energierückeinspeisung bei Spindel-Stop

Bei den vorangegangenen Beispielen beeinflusst ein NC-Befehl die Definitionen bzw. den grundsätzlichen Informationsgehalt der darauf folgenden Datenpunkte. Daneben bestehen noch Situationen in denen ein NC-Befehl allein auf den

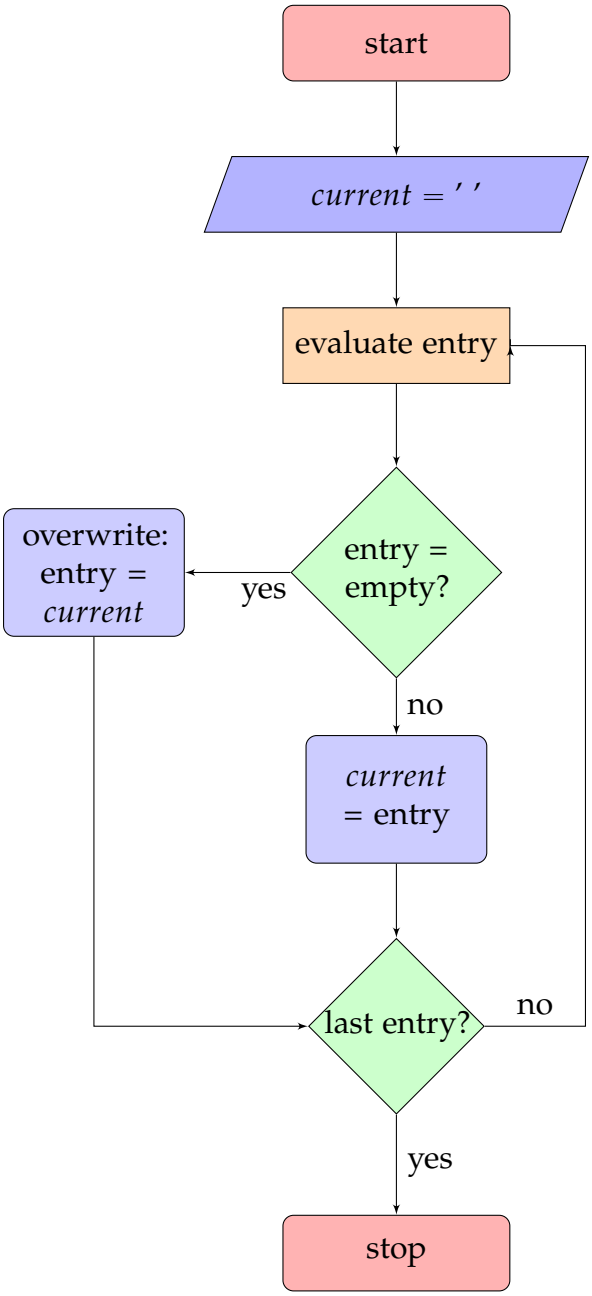


Abbildung 7.4: Grundmechanismus beim Übertragen von Informationen in nachfolgenden Datenzeilen (Quelle: Eigene Darstellung)

GCode	ENERGY—S
G4 F0	5,60
N666 M169 M167	21,80
N668 M5	-453,34
N670 SUPA Z=_Z_HOME D0	-219,62
N670 SUPA Z=_Z_HOME D0	-7801,42
"MSG(ZENTRIERBOHREN , Tool : 002383_A)"	-60,46
STOPRE	-218,93
STOPRE	-212,06
STOPRE	-202,03
N021 _AUTOGEAR_2=($P_SAUTOGEAR[1] AND ($AC_SGEAR[1]¿=2))	-192,09
N390 IF NOT M_ENABLE_TOOLCHANGE AND NOT _SIM AND NOT $P_ISTEST AND NOT TESTRACK AND _POS GOTOF FEHLER4	-182,28
N394 STOPRE	-171,98
N396 _MODE=STEP_OLD FAST=0 _TEST=_SIM OR $P_SEARCH OR $P_ISTEST OR TESTRACK	-162,57
N396 _MODE=STEP_OLD FAST=0 _TEST=_SIM OR $P_SEARCH OR $P_ISTEST OR TESTRACK	-42,50
N464 IF($AA_IM[_AX4]¡$MN_USER_DATA_INT[83]-0.5)OR($AA_IM[_AX4]¿$MN_USER_DATA_INT[84]+0.5)	-149,36
N480 GETSELT(T_PR)	-38,89
N542 M27 M28	-317,54
N562 IF NOT M_ENABLE_TOOLCHANGE AND NOT _SIM AND NOT $P_ISTEST AND NOT TESTRACK AND NOT $P_SEARCH GOTOF FEHLER	-113,24
N572 IF ($AA_IM[_AX4]¡$MN_USER_DATA_INT[83]-0.5) AND ($MA_POS_LIMIT_MINUS[AX4]¿-360) GOTOF FEHLER3	-102,11
N574 IF ($AA_IM[_AX4]¿$MN_USER_DATA_INT[84]+0.5) AND ($MA_POS_LIMIT_PLUS[AX4]¡360) GOTOF FEHLER3	-92,70
N626 STOPRE	-83,39
N638 STOPRE	-74,30
N640 _H_EIN=($AC_MARKER[3]==9999)	-65,70
N796 STEP=1	-16,29
N838 FAST=1	-15,45
N841 STOPRE	-51,30
N864 IF SPINDLE_GEAR AND NOT SPINDLE_REF	-42,72
N912 SUPA G90 D0 SPOS=0 POS[X]=_X FA[X]=RED_RAPID_SPEED POS[Y]=_Y FA[Y]=RED_RAPID_SPEED POS[Z]=_Z FA[Z	-34,32
N912 SUPA G90 D0 SPOS=0 POS[X]=_X FA[X]=RED_RAPID_SPEED POS[Y]=_Y FA[Y]=RED_RAPID_SPEED POS[Z]=_Z FA[Z	-50,37
N1000 STOPRE	0
N1020 IF FUSE_PERIPHERAL	0

Tabelle 7.2: Energiebedarf vor und nach dem Stoppen der Spindel (M5, dritte Zeile) (Quelle: Eigene Darstellung)

gemessenen Energiebedarf der nachfolgenden Code-Zeilen Einfluss nimmt. Tabelle 7.2 gibt die NC-Befehlszeilen vor und nach dem Ausschalten der Spindel wieder. Es ist zu sehen, dass nach dem Stoppen der Spindel mit dem »M5«-Befehl in Zeile drei, der Energieverbrauch noch über mehrere Zeilen negativ bleibt ergo Bremsenergie rückgeführt wird. Für die Vorhersage ist dies fatal, da die Höhe der zurückgewonnenen Arbeit keiner Gesetzmäßigkeit zu folgen scheint: In Abbildung 7.5 sind die Tabellenwerte graphisch dargestellt. Dabei ist deutlich zu erkennen, dass die rückgespeiste Energie kurz nach dem Halt-Befehl (M5) einen Peak erreicht und anschließend über einen längeren Zeitraum abklingt.

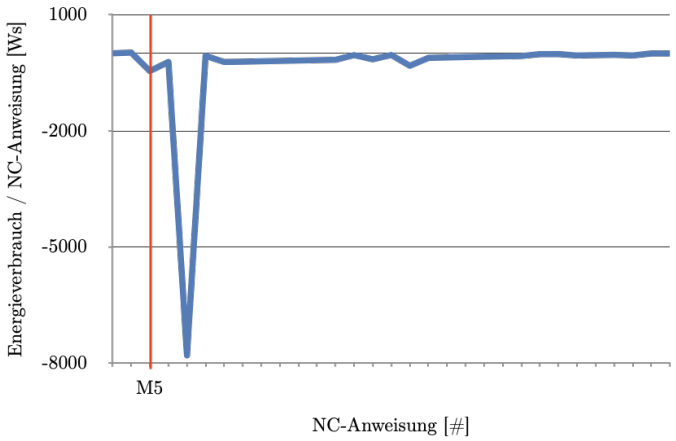


Abbildung 7.5: Verlauf der Rückspeisung von Energie nach dem Stoppen der Spindel (M5) (Quelle: Eigene Darstellung)

7.2 Feature Extraction

7.2.1 Ermittlung der Vorschubgeschwindigkeit

Der Vorschub ist nur in Ausnahmefällen direkt als numerischer Wert aus der NC-Anweisung auslesbar. Im Regelfall muss der Wert des Vorschubes über Informationen ermittelt werden, welche in vier voneinander unabhängigen und nicht aufeinander folgenden NC-Befehlen erfolgen:

```
MSG("BOHRFRAESEN_SCHRUPPEN , Tool : 002384_A")
N782 T="002380_A"
N788 _F_RET = 680 ;Retract Move
N808 X0. Y-28.538 I1.346 J0.107 F=_F_RET
```

NC-Anweisung N782

Die Werkzeugauswahl für die Bearbeitung geschieht in der Nachrichtenzeile MSG, der darauffolgende Befehl »T="002380_A"« bezieht sich auf den Werkzeugnamen im Werkzeugspeicher. Der Parser muss diese Informationen voneinander trennen um später den Vorschub für das Werkzeug in der Spindel abzufragen und nicht für das Werkzeug, welches in dem Werkzeuglager für den nächsten Bearbeitungsgang bereit steht.

Der Parser hat die Aufgabe die Vorschubgeschwindigkeit für jede NC-Zeile als nummerischen Wert abzuspeichern, damit diese Information später vom Algorithmus sinnvoll genutzt werden kann. Neben der Herausforderung, dass die Vorschubgeschwindigkeit von vier NC-Befehlen festgesetzt wird, kommt erschwerend hinzu, dass die Zuordnung der numerischen Werte zu den Va-

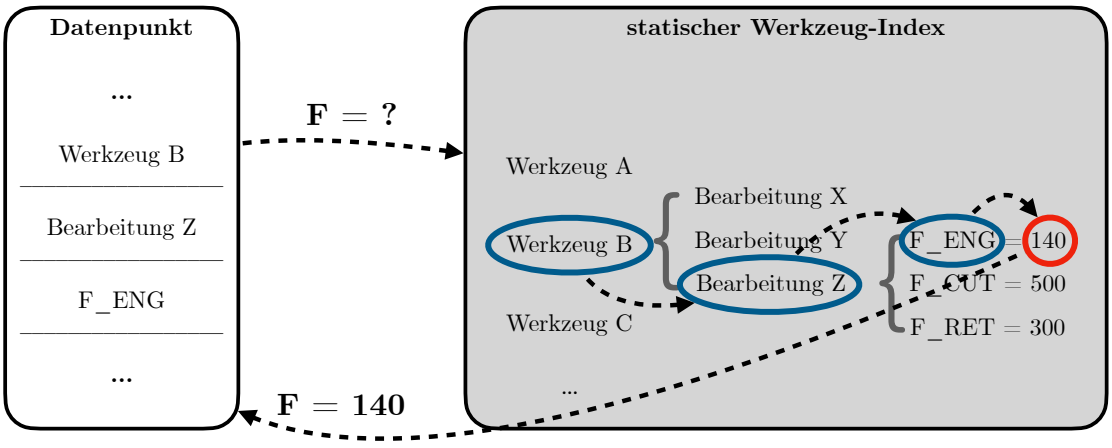


Abbildung 7.6: Arbeitsprinzip zur Ermittlung der Vorschubgeschwindigkeit (Quelle: Eigene Darstellung)

riablen F_{eng}, F_{cut} und F_{ret} nur im G-Code der Orginaldatei zur Verfügung steht und nicht in der hier verarbeiteten .json-Datei mit abgespeichert wird. Die Definitionen des Vorschubes müssen somit händisch aus der originalen .MPF-Datei ausgelesen werden (s. Tabelle 7.3).

Anschließend kann aus diesen Informationen ein statischer Werkzeugindex aufgebaut werden (s. Abb. 7.6). Dieser kann, sofern die benötigten Informationen in die Datenpunkte übertragen wurden (s. Abschnitt 7.1.3), für jeden Datenpunkt die aktive Vorschubgeschwindigkeit ermitteln und ausgeben.

7.2.2 Variablen mit unbekanntem Wert

In den NC-Befehlen können Positionen nicht nur direkt numerisch angegeben werden (z.B.: X3.42) sondern auch in Form von Variablen:

- X=IC(0), Y=IC(0), Z=IC(0)
- X_HOME, Y_HOME, Z_HOME
- AX[_Z]=DP
- AX[_Z]=RTP
- F=_DTB

Der Wert der meisten Variablen ist nur maschinenintern bekannt und steht nach der Bearbeitung nicht mehr zur Verfügung. Die Ausgangsposition (X,Y,Z_HOME) ist für die verwendete CNC-Bearbeitungsmaschine (SPINNER U5630) bekannt und in Tabelle 7.4 angegeben.

Andere Variablen bleiben unbekannt und werden vom Parser entweder ignoriert oder er setzt für sie willkürlichen den Wert Null ein. Das diese Annahmen einen erheblichen, negativen Einfluss auf die Vorhersagen ausüben, ist in Tabelle 7.5 deutlich zu erkennen: Hier wird die Vorhersagegüte des Energieverbrauchs

Separierter Teil der NC-Anweisung	SelectedTool / ActiveTool	Vorschub (F_ENG / F_CUT / F_RET)
MSG(PLANEN , Tool : 001691_A)	5	-
T=3D_TASTER	-	-
N36 T=001691_A und WKZBG_SK_40-10-50_SF10_32	2	-
MSG(SCHRUPPEN , Tool : 001691_A)	1 / 2	-
N40 T=002383_A	1/2	1000 / 1000 / 1000
MSG(ZENTRIERBOHREN , Tool : 002383_A)	1/2	500/ - / 500
N674 T=002293_A	1/2	-
MSG(VORBOHREN_4.7 , Tool : 002293_A)	12	-
N712 T=001728_A	15	500 / - / 0
MSG(VORBOHREN_2.8 , Tool : 001728_A)	-	-
N748 T=002384_A	6	100 / - / 0
MSG(BOHRFRAESEN_SCHRUPPEN , Tool : 002384_A)	6	-
N782 T=002380_A	9	24 / 680 / 680
MSG(KREUZ_SCHRUPPEN , Tool : 002384_A)	9 / 6	750 / 750 / 750
MSG(KREUZ_SCHLICHTEN , Tool : 002384_A)	9 / 6	500 / 500 / 500
MSG(RESTFRAESEN_SCHRUPPEN , Tool : 002384_A)	9 / 6	750 / 750 / 750
MSG(FASENFRAESEN_1 , Tool : 002380_A)	9 / 6	-
MSG(FASENFRAESEN_2 , Tool : 002380_A)	21 / 9	-
N1436 T=T_A_SK40_SF_DM3_SL_A	21 / 9	550 / 550 / 550
MSG(SCHLICHTEN_AUSSEN , Tool : 002384_A)	-	-
N1662 T=001691_A	2 / 21	550 / 550 / 550
MSG(SCHLICHTEN_INNEN , Tool : 002384_A)	2 / 21	-

Tabelle 7.3: NC-Anweisungen mit elementaren Informationen bzgl. des Vorschubes (Quelle: Eigene Darstellung)

Variable	Wert (mm)
X_HOME	0
Y_HOME	0
Z_HOME	610

Tabelle 7.4: Definition der Basisposition bezogen auf der Maschinenkoordinatensystem (Quelle: Eigene Darstellung)

für die Z-Achse verglichen: Für eine unbekannte Basisposition in Z-Richtung und für die tatsächliche Basisposition in Z-Richtung. In Abbildung 7.7 sind die Verbesserungen der Vorhersagequalität quantifiziert.

Maß	Z_HOME (unbekannt)	Z_HOME (bekannt)
Ges. Abweichung [%]	4,29	-6,29
Mtl. abs. Abw. [Ws]	4,28	3,13
RMSE	41,77	32,81
Erkl. Varianz	0,37	0,61

Tabelle 7.5: Auswirkungen von Positionsvariablen (Quelle: Eigene Darstellung)

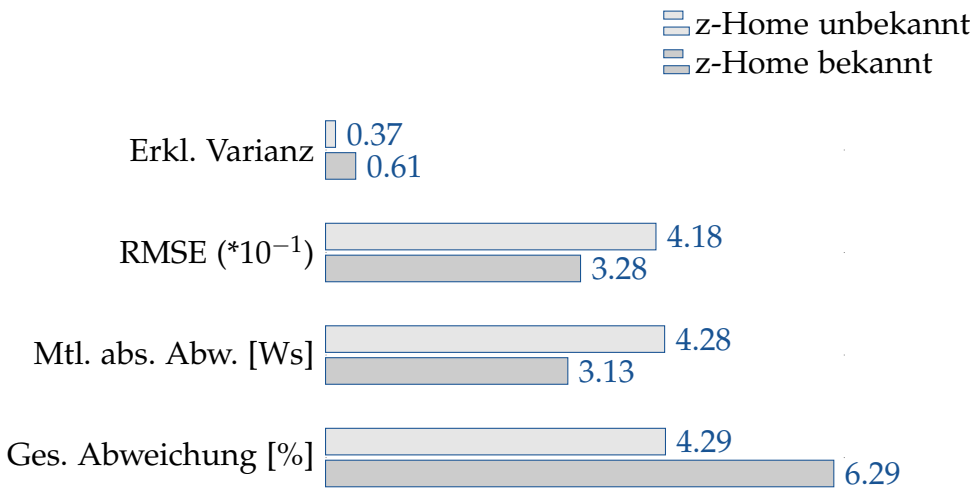


Abbildung 7.7: Verbesserung der Vorhersagen durch bekannte z-Position (Quelle: Eigene Darstellung)

8 Schritt Vier: Verbesserung des Modells

8.1 Feature Engineering

Wie bereits bei der Entwicklung des prototypischen Modells (s. Kap. 5.2) erläutert, können nun mit dem erweiterten Datensatz auch weitere Charakteristika konstruiert werden (Feature Engineering).

8.1.1 Auswirkungen des Werkzeugdurchmessers

Der Werkzeugdurchmesser steht nicht als Charakteristika zur Verfügung, es ist aber trivial, dass ein Zusammenhang zwischen Werkzeuggröße und Energieverbrauch besteht. Man kann davon ausgehen, dass große Werkzeuge für Bearbeitungen mit großen Zeitspanungsvolumen eingesetzt werden und umgekehrt. Die eingesetzten Werkzeuge mit ihren geometrischen Eigenschaften sind in Tabelle 8.1 aufgelistet. Die Verwendung eines numerischen Werkzeugdurchmessers $D_{bekannt}$ bietet außerdem den Vorteil, dass in Zukunft unbekannte Werkzeuge $W_{unbekannt}$ besser eingeordnet werden können. Trainiert das Modell nur auf die spezifischen Werkzeugnamen kann kein Zusammenhang für neue Werkzeuge hergestellt werden. Ein numerischer Durchmesser lässt sich allerdings einordnen, wenn die Bearbeitungslänge $s_{Bearbeitung}$ für beide Werkzeuge gleich ist:

$$
\begin{aligned}
D_{bekannt} &< D_{Neu} \\
\text{wenn: } s_{Bearbeitung} &= \text{konstant} \\
\longrightarrow W_{bekannt} &< W_{Neu}
\end{aligned}
$$

Es kann nun auf die in Kapitel 7.2.1 Methodik zurückgegriffen und der entwickelten Werkzeugindex (s. Abb. 7.6) um die Werkzeugdurchmesser erweitert werden. Jedem Datenpunkt kann dadurch der Durchmesser des verwendeten

Werkzeug	G-Code-Bezeichnung
End Mill DM16	T_A_SK40_DM16_A
End Mill DM10	001691_A
End Mill DM6	001688_A
End Mill DM3	002384_A und T_A_SK40_SF_DM3_SL_A
Center Drill DM8 90Grad	00233_A
Twist Drill DM4,7	002293_A
Chamfer Mill DM8 90Grad	002380_A
Chamfer Mill DM10 60Grad	002387_A
Ball Mill DM4	002397_A
Drill DM2,8	001728_A
Reibahle DM5	002411_A

Tabelle 8.1: Verwendete Werkzeuge und deren Bezeichnung im G-Code (Quelle: Eigene Darstellung)

Werkzeuges zugefügt werden. In Tabelle 8.2 ist die Vorhersagegenauigkeit mit und ohne diesem konstruierten Charakteristikum dargestellt. Wie vermutet, kann eine deutliche Verbesserung der Vorhersagequalität erzielt werden.

Modus	Maß	X	Y	Z	S	W
ohne Werkzeug-∅	Ges. Abweichung [%]	8,96	3,87	4,54	1,71	6,49
	Mtl. abs. Abw. [Ws]	2,63	0,61	4,9	58,76	1,2
	RMSE	25,63	5,05	45,09	359,7	7,37
	Erkl. Varianz	0,69	0,66	0,26	0,82	0,68
mit Werkzeug-∅	Ges. Abweichung [%]	7,29	4,03	4,29	3,76	7,04
	Mtl. abs. Abw. [Ws]	2,4	0,6	4,28	52,74	1,02
	RMSE	24,39	4,87	41,77	332,86	6,19
	Erkl. Varianz	0,72	0,68	0,37	0,85	0,77

Tabelle 8.2: Vergleich der Vorhersagegüte bei Miteinbeziehen des Werkzeugdurchmessers (Quelle: Eigene Darstellung)

8.1.2 Einbeziehen von D-Codes

D-Befehle sind interne Maschinenbefehle, welche im Zusammenhang mit der Speicherverwaltung der Werkzeugkorrekturen stehen[1]. Insofern macht es auf

[1] Grote und Feldhusen, 2014, S. 1483.

den ersten Blick keinen Sinn dieses Feature mit einzubeziehen, da eine Speicheroperation per se keinen signifikanten Energieverbrauch hervorruft. Interessanterweise korrelieren diese Speicheroperationen teilweise mit Datenpunkten mit erhöhtem Energiebedarf.

Markiert man für den Algorithmus alle Datenpunkte während deren eine Speicheroperation ausgeführt wird, ist eine deutliche Verbesserung der Vorhersagen insbesondere für die Z-Achse und die Spindel zu verzeichnen, wie die Gegenüberstellung in Tabelle 8.3 zeigt.

Modus	Maß	X	Y	Z	S	W
ohne D-Befehle	Ges. Abweichung [%]	5,09	3,96	-6,29	5,39	4,19
	Mtl. abs. Abw. [Ws]	22,37	0,59	3,13	46,18	0,97
	RMSE	24,3	4,83	32,81	297,92	5,85
	Erkl. Varianz	0,72	0,69	0,61	0,88	0,8
mit D-Befehlen	Ges. Abweichung [%]	3,47	3,15	3,39	3,27	5,03
	Mtl. abs. Abw. [Ws]	2,25	0,59	1,1	37,05	0,88
	RMSE	23,69	4,74	11,5	230,16	5,51
	Erkl. Varianz	0,74	0,7	0,95	0,93	0,82

Tabelle 8.3: Vergleich der Vorhersagegüte bei Miteinbeziehen von Speicheranweisungen (Quelle: Eigene Darstellung)

8.2 Liste der verwendeten Features

Nach den Verbesserungen des Algorithmus in Kapitel 7 und 8.1 werden folgende Features, welche vom Algorithmus genutzt werden:

delta_X Differenz des Weges in x-Achsenrichtung im Bezug zur vorhergehenden NC-Anweisung
delta_Y Differenz des Weges in y-Achsenrichtung im Bezug zur vorhergehenden NC-Anweisung
delta_Z Differenz des Weges in z-Achsenrichtung im Bezug zur vorhergehenden NC-Anweisung
delta_S Differenz der Spindeldrehgeschwindigkeit im Bezug zur vorhergehenden NC-Anweisung
F_val Vorschubgeschwindigkeit
S Spindeldrehgeschwindigkeit
D_W Durchmesser des verwendeten Werkzeuges
Toolchange Werkzeugwechsel
TurnOp Schraubenlinien-Interpolation
Commands_G0 G-Funktion, Eilgang
Commands_G0 G40 G60 G-Funktion

Commands_G0 M106 G-Funktion
Commands_G0 M3 G-Funktion, M3: Spindel ein (rechtslaufend)
Commands_G09 G-Funktion, Genauhalt an Ecken
Commands_G1 G-Funktion, Geradeninterpolation mit Vorschub
Commands_G1 G60 G-Funktion, G60: Genauhalt fein
Commands_G2 G-Funktion, Kreisinterpolation im Uhrzeigersinn
Commands_G4 G-Funktion, Verweilzeit mit Adresse F oder S
Commands_G40 G-Funktion, Werzeugradiuskorrektur löschen
Commands_G41 G1 G-Funktion, G41: Werzeugradiuskorrektur links von der Kontur
Commands_G4F1 G-Funktion
Commands_G54 G0 G-Funktion, G54: Nullpunktverschiebung
Commands_G90 G-Funktion, Absolutbezugsmaßeingabe
Commands_G91 G-Funktion ‚Kettenmaßeingabe
Commands_G94 G-Funktion, Vorschub Millimeter pro Minute
Commands_G94 G1 G90 G-Funktion
Commands_M168 M-Funktion, aktiviert die Minimalmengenschmierung (Luft und Ã–l)
Commands_M169 M167 M-Funktion, Deaktivierung von Luft (M167) und Öl (M169)
Commands_M17 M-Funktion, Ende Unterprogramm Siemens
Commands_M27 M28 M-Funktion
Commands_M5 M-Funktion, Spindel stopp
Commands_M58 M-Funktion
Commands_M59 M-Funktion
Commands_MSG Zusammenfassung aller übrigen NC-Codelinien
D_D0 Befehl zur Deaktivierung des Korrekturdatensatzes für das aktive Werkzeug
D_D1 D-Funktion
D_D=P_TOOL D-Funktion

8.3 Model Training und Prediction

Multiple Zielvariablen

Der Algorithmus der ersten beiden Stufen war darauf ausgelegt, Vorhersagen für eine Zielvariable zu treffen. In dem erweiterten Datensatz ist nun der Energieverbrauch von fünf separaten Aggregaten vorherzusagen. Für die Implementierungen im SCIKIT-LEARN-Paket ist dies allerdings nur für den DecisionTree und RandomForest möglich. Für den gebooteten Entscheidungsbaum (AdaBoost + DecisionTree) bestehen auch theoretische Ansätze, mehrere Zielva-

riablen gleichzeitig vorherzusagen, welche genauere Vorhersagen produzieren[2], allerdings ist keine verwendbare Implementierung im SCIKIT-LEARN-Paket vorhanden. Daher muss auf eine Technik zurückgegriffen werden, welche einen separaten Algorithmus für jede Zielvariable trainiert und die Ergebnisse nach der Vorhersage wieder zusammenführt (s. Abb. 8.1).

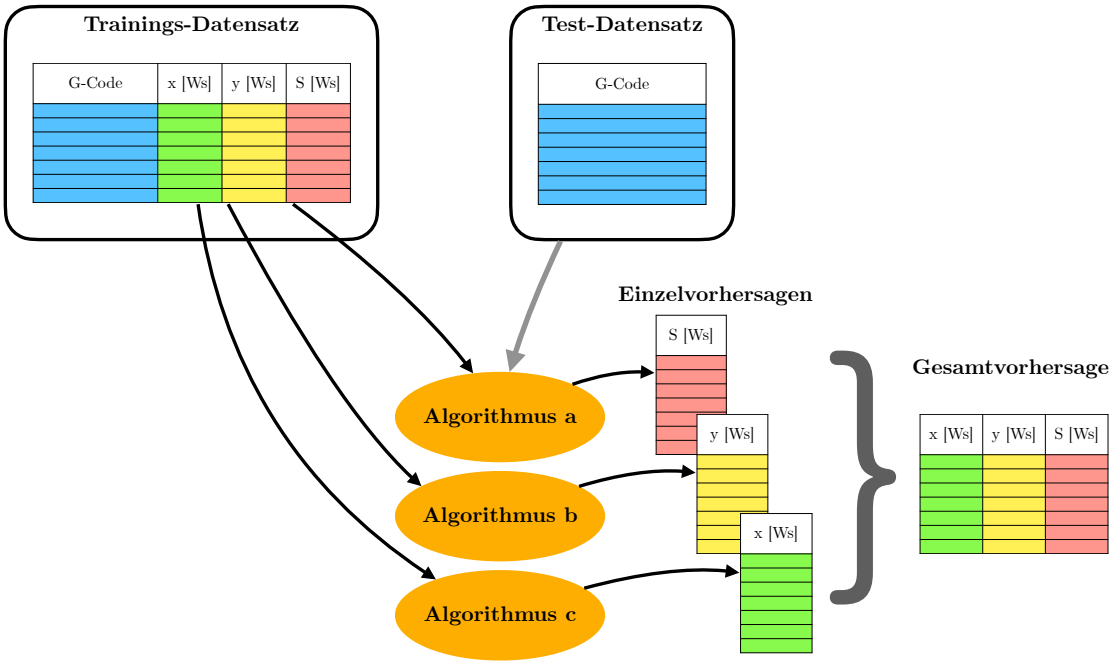


Abbildung 8.1: Erstellen von multiplen Vorhersagen für single-Output Algorithmen (Quelle: Eigene Darstellung)

8.3.1 Vergleich verschiedener Algorithmen

Für die Vorhersagen des prototypischen Modells wurden bereits mehrere Algorithmen verglichen um eine fundierte Auswahl in Hinblick der besten Vorhersagen treffen zu können, insbesondere in Betracht auf die Ausreißerunterscheidung (s. Kapitel 6.6). Jedoch wurden mit dem vollständigen Datensatz die Rahmenbedingungen insoweit verändert, dass eine erneute Analyse von Nöten ist. In Tabelle 8.4 werden der Algorithmus mit den besten Ergebnissen aus dem Prototypen (Boosted DecisionTree), der ihm zugrundeliegende einfache Entscheidungsbaum (DecisionTree) und die häufig verwendete Bagging-Methode des Entscheidungsbaums (RandomForest) vergleichen.

[2]Kummer und Najjaran, 2014.

Algorithmus	Maß	X	Y	Z	S	W
RandomForest	Ges. Abweichung [%]	3,47	3,15	3,39	3,27	5,03
	Mtl. abs. Abw. [Ws]	2,25	0,59	1,1	37,05	0,88
	RMSE	23,69	4,74	11,5	230,16	5,51
	Erkl. Varianz	0,74	0,7	0,95	0,93	0,82
DecisionTree	Ges. Abweichung [%]	0,0	0,0	0,0	0,0	0,0
	Mtl. abs. Abw. [Ws]	2,02	0,48	0,31	14,43	0,63
	RMSE	23,08	4,55	5,69	145,89	4,95
	Erkl. Varianz	0,75	0,72	0,99	0,97	0,86
AdaBoost	Ges. Abweichung [%]	-2,55	-18,37	-1,09	8,8	5,96
	Mtl. abs. Abw. [Ws]	2,06	0,79	0,37	27,33	0,89
	RMSE	24,33	4,7	5,79	196,22	7,14
	Erkl. Varianz	0,72	0,71	0,99	0,95	0,7

Tabelle 8.4: Vergleich der Vorhersagegüte verschiedener Algorithmen bei Vorhersage bekannter Daten (Quelle: Eigene Darstellung)

8.3.2 Null Prozent Abweichung beim Entscheidungsbaum

In Tabelle 8.4 wird die Gesamtabweichung für den Entscheidungsbaum jeweils mit 0% angegeben. Dies ist keinesfalls ein Fehler sondern auf den Versuchsaufbau zurückzuführen. Ist die Gesamtabweichung null, muss gelten:

$$\sum E_{Vorhersage} = \sum E_{Messungen} \tag{8.1}$$

Nehmen wir an, dass der Entscheidungsbaum aus zwei Blättern (leafs) mit m, n Datenpunkten (samples) besteht deren Durchschnitt $\varnothing_i$ aus den Teilmengen A, B mit jeweils n, m Messergebnissen mit $A \cup B = C_{Messergebnisse}$ gebildet wird, so gilt:

$$n * \varnothing_1 + m * \varnothing_2 = \sum_{i=1}^{n+m} E_i \tag{8.2}$$

$$n * \frac{1}{k} \sum_{i=1}^{k} E_i + m * \frac{1}{j} \sum_{i=k+1}^{k+j} E_i = \sum_{i=1}^{n+m} E_i \tag{8.3}$$

Da im einfachen Entscheidungsbaum die gleichen Daten zum Testen herangezogen werden, wie zum Erstellen, werden bei der Vorhersage jedem Blatt des Baumes die gleichen Datenpunkte zugeordnet wie bei der Erstellung des

Baumes. Dadurch ist $n = k$ und $m = j$ und somit schließlich:

$$n * \frac{1}{n} \sum_{i=1}^{n} E_i + m * \frac{1}{m} \sum_{i=n+1}^{n+m} E_i = \sum_{i=1}^{n+m} E_i \tag{8.4}$$

$$\sum_{i=1}^{n+m} E_i = \sum_{i=1}^{n+m} E_i \tag{8.5}$$

Dieser Effekt ist nur für diesen Schritt des Entwicklungsprozesses tragend, da im nächsten Schritt ein Datensatz zum Trainieren und ein anderer zur Generierung von Vorhersagen herangezogen wird.

8.3.3 Bewertung der Vorhersagen von Schritt Vier

In Tabelle 8.4 sind die Kennzahlen der Vorhersagegüte bei bekannten Daten für verschiedene Algorithmen dargestellt. Daraus lässt sich schließen, dass der einfache Entscheidungsbaum wieder bessere Ergebnisse liefert als seine gebaggte Variante (RandomForest). Allerdings liefert die geboostete Variante des Entscheidungsbaums (AdaBoost) wider Erwarten nicht die besten Ergebnisse, was auf die Problemstellung der multiplen Variablen zurückzuführen ist (s. Kap. 8.3).

Eine genauere Bewertung und Diskussion der Ergebnisse soll in der nächsten Stufe mit unbekannten Daten erfolgen.

9 Schritt Fünf: Validierung des erstellten Models mit neuen Test-Daten

Das in Schritt Vier erstellte Modell soll im letzten Schritt mit Daten getestet werden, welche nicht bei dem Trainieren des Algorithmus verwendet wurden (unbekannte Daten). Die Generierung von Vorhersagen und deren Gegenüberstellung ist kongruent mit den Abläufen in den vorangegangen Schritten des Prozesses (s. Kap. 3.2). Bis auf die in Kapitel 9.1 beschriebenen Änderungen ist das Programm deckungsgleich mit der zweiten Stufe und wird daher nicht ein weiteres Mal besprochen.

Die fünfte Stufe des Fünf-Schritte-Prozesses (s. Kap. 3.3) hat nicht zum Ziel, weitere Verbesserungen des Modells herbeizuführen, sondern die Ergebnisse mit unabhängigen Daten zu verifizieren und zu diskutieren. Um dies zu ermöglichen ist jedoch eine Anpassung des Programmes notwendig.

9.1 Anpassung des Modells an unbekannte Daten

Unterscheidung von Trainingsdaten und Testdaten

Wurde in den vorhergehenden Schritten mit jeweils nur einem Datensatz gearbeitet, muss das Programm nun in der Lage sein zwei verschiedene Datensätze zu handhaben. Der erste Datensatz stammt aus der Fertigung der Vorderseite des Werkstückes (s. Kap. 4.1) und wird zur Erstellung bzw. zum Antrainieren des Algorithmus verwendet. Der zweite Datensatz wurde bei der Fertigung der Rückseite des Werkstückes angelegt und dient zur Verifikation des erstellten Algorithmus: Dabei werden die Messergebnisse vorerst vom Datensatz getrennt und anschließend gegen die Vorhersagen des Machine Learning-Algorithmus verglichen.

Angleichung der Charakteristika

Das Modell, insbesondere das Parsing der NC-Anweisungen (s. Kap. 7.1) sind bereits mit der Kompatibilität an die zweite Seite erstellt worden. Allerdings bleibt eine Anpassung des Machine Learning-Programms aus: Die beiden Datensätze beinhalten nicht kongruente Sätze von NC-Anweisungen, sprich ein Datensatz beinhaltet NC-Anweisungen, welche im anderen Datensatz nicht enthalten sind. Da jede individuelle NC-Anweisung durch das entwickelte Programm als Charakteristika (Feature) zum Trainieren des Models herangezogen wird, führt es zu Fehlern falls diese Features beim Trainings- und Test-Datensatz nicht deckungsgleich sind. Um nicht zu viel Informationen zu verlieren wird, wo möglich, jede zu ersetzender NC-Anweisung, durch jene Anweisung ersetzt, welche ihr am Ähnlichsten ist. Namentlich werden in den Datensätzen folgende Ersetzungen durchgeführt:

$$
\begin{aligned}
\text{G0 G90} &\longrightarrow \text{G0} \\
\text{G41 G94 G1 G90} &\longrightarrow \text{G1} \\
\text{M58;} &\longrightarrow \text{M58,} \\
\text{G41} &\longrightarrow \text{MSG} \\
\text{G94 G3 G90} &\longrightarrow \text{G2}
\end{aligned}
$$

9.2 Ergebnissdiskussion

9.2.1 Quantitative Bewertung

Wie auch in den vorangegangenen Kapiteln bei der Bewertung neuer Features soll auch hier eine quantitative Bewertung der Vorhersageergebnisse geschehen: Wie in Tabelle 9.1 dargestellt, liefert der RandomForest-Algorithmus bereits Vorhersagen, die über den gesamten Bearbeitungsprozess gesehen, beispielsweise für die Achsaggregate lediglich 2-6% von den tatsächlichen Messergebnissen abweichen. Auch wird eine erklärte Varianz von bis zu 79% erreicht.

Die für die Vorhersage verwendete Charakteristika (Features) und deren Gewichtung bei der Erstellung der Vorhersage ist in Abb. 9.1 dargestellt.

Diese Kennzahlen allein erlauben jedoch nicht eine abschließende Bewertung und Beurteilung. Solange es keine alternativen Vorhersagemodelle gibt, gegen welche man diese Kennzahlen abgleichen kann, bleibt die qualitative Bewertung unumgänglich.

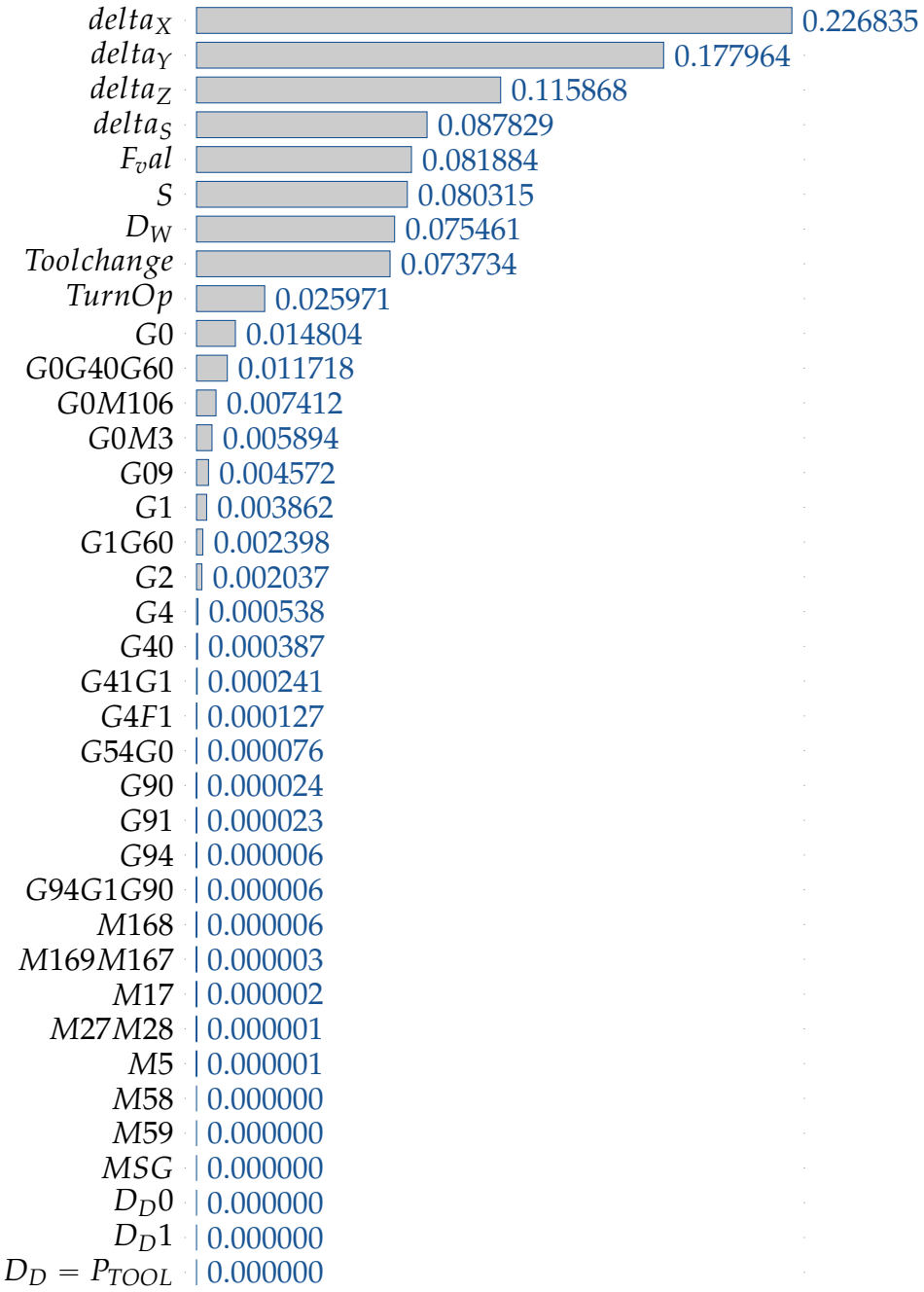


Abbildung 9.1: Feature-Gewichtungen bei der Erstellung der Vorhersagen (Quelle: Eigene Darstellung)

Algorithmus	Maß	X	Y	Z	S	W
RandomForest	Ges. Abweichung [%]	2,34	-3,01	-5,8	7,27	39,39
	Mtl. abs. Abw. [Ws]	2,55	0,83	1,85	98,39	1,16
	RMSE	24,75	5,93	23,95	651,96	8,83
	Erkl. Varianz	0,71	0,53	0,79	0,42	0,54
DecisionTree	Ges. Abweichung [%]	-6,83	-17,98	-4,04	9,08	21,2
	Mtl. abs. Abw. [Ws]	3,35	1,83	3,14	129,75	1,57
	RMSE	27,44	6,81	30,11	685,3	9,42
	Erkl. Varianz	0,64	0,38	0,67	0,36	0,48
AdaBoost	Ges. Abweichung [%]	-4,86	-14,92	-11,78	-35,82	24,26
	Mtl. abs. Abw. [Ws]	2,24	1,22	1,57	191,67	1,3
	RMSE	23,69	6,51	28,21	698,24	9,69
	Erkl. Varianz	0,74	0,44	0,71	0,34	0,45

Tabelle 9.1: Vergleich der Vorhersagegüte verschiedener Algorithmen bei unbekannten Daten (Quelle: Eigene Darstellung)

9.2.2 Qualitative Bewertung

x,y,z-Aggregate

Beispielhaft für die verwendeten Achsaggregate soll hier das Aggregat der y-Achse besprochen werden. In Abbildung 9.2 sind die tatsächlichen Messergebnisse und die Werte der Vorhersagen gegenübergestellt. Es ist zu sehen, dass der Verlauf des Energieverbrauches grundsätzlich richtig dargestellt wird: Der Energieverbrauch während der Bearbeitungen ist sowohl im Verlauf wie in der Skala richtig vorhergesagt. Bei den Verbrauchsspitzen lässt sich allerdings eine starke Divergenz feststellen und deren Höhen sind häufig nicht korrekt vorhergesagt. Es muss jedoch angemerkt werden, dass die Position der Verbrauchsspitzen im Verlauf richtig vorhergesagt wurden.

Spindelaggregat

Als Aggregat, welches die höchsten positiven wie auch negativen Energiespitzen verantwortet, ist der Verbrauch des Spindelantriebs von besonderem Interesse. Wie schon bei der y-Achse ähneln sich die beiden Graphen für tatsächliche Messergebnisse und Vorhersagen in Abbildung 9.3. Wiederum sind die Verbrauchsspitzen zur richtigen Zeit im Verlauf vorhergesagt unterscheiden sich allerdings merklich im Betrag.

Bemerkenswert ist bei den Vorhersagen die treffende Vorhersage der negativen Spannungsspitzen. Dies zeigt, dass die in Kapitel 7.2 konstruierten Charakteris-

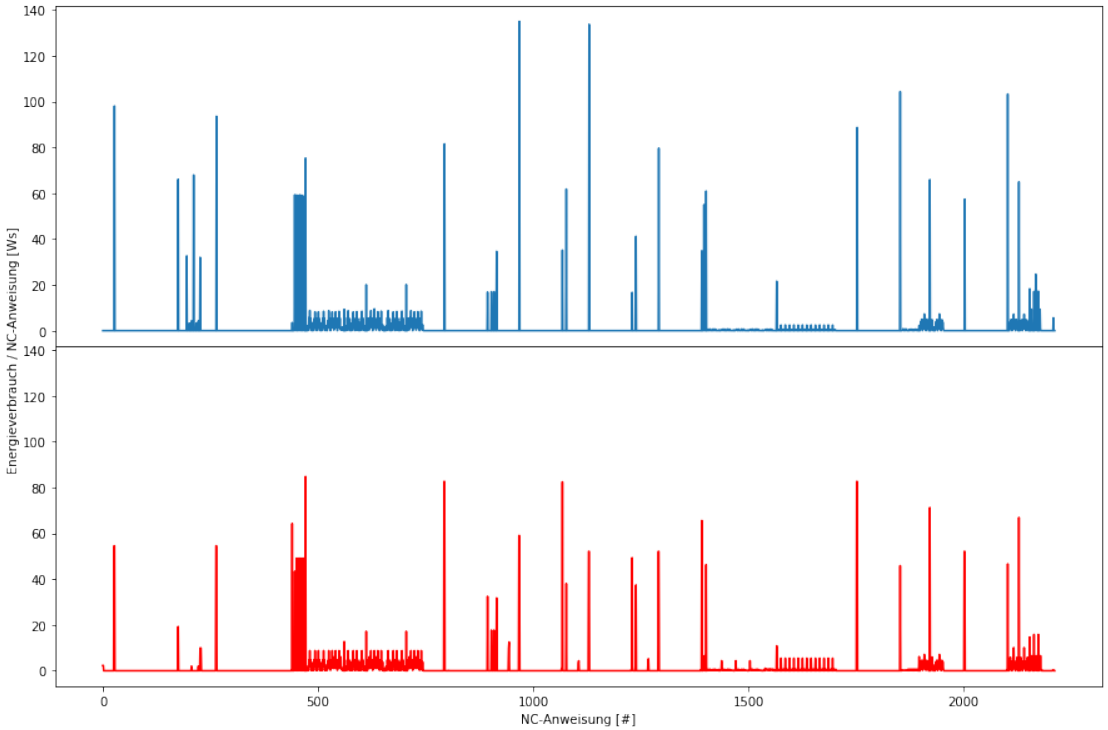


Abbildung 9.2: Gegenüberstellung der tatsächlichen Messergebnisse (oben) und der vorhergesagten Werte (unten) für die y-Achse (Quelle: Eigene Darstellung)

tika (Feature Extraction) für den Algorithmus ausreichend gewesen sind, um ein Stoppen der Spindel richtig zu interpretieren.

Werkzeugwechsler

Der Werkzeugwechsel ist an sich nicht von großem Interesse, da das verantwortliche Aggregat nicht zu den großen Verbrauchern gehört. Allerdings entstehen auch hier Spannungsspitzen. Außerdem ist beim Werkzeugwechsel der verantwortliche Maschinenbefehl (M6) nicht im Datensatz enthalten. Wie Abbildung 9.4 zeigt, ist die Vorhersage trotz dieser Einschränkung dank des durchgeführten Feature Engineering erstaunlich gut. Alleinig im Hochfahrprozess der Maschine (zwei Verbrauchsspitzen im oberen Graphen) werden Spitzen nicht richtig vorhergesagt.

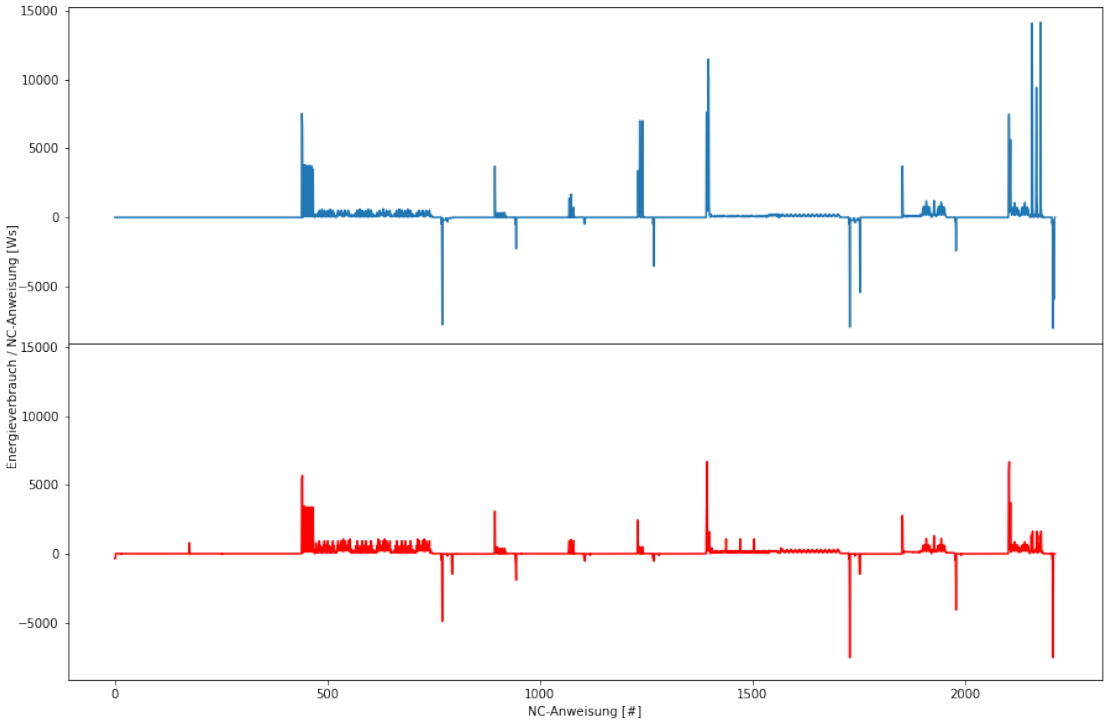


Abbildung 9.3: Gegenüberstellung der tatsächlichen Messergebnisse (oben) und der vorhergesagten Werte (unten) für das Spindelaggregat (Quelle: Eigene Darstellung)

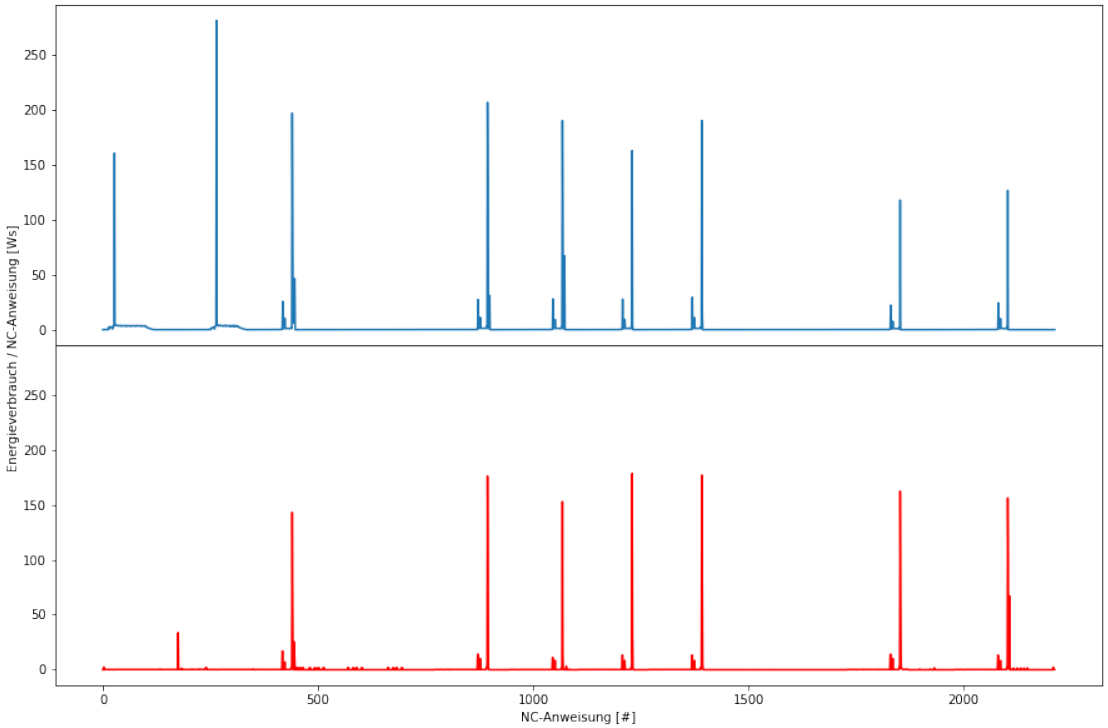


Abbildung 9.4: Gegenüberstellung der tatsächlichen Messergebnisse (oben) und der vorhergesagten Werte (unten) für den Werkzeugwechsler (Quelle: Eigene Darstellung)

10 Ergebnisse

Kongruent mit der Aufgabendefinition in Kapitel 3.2 sollen auch die Ergebnisse hinsichtlich der zwei Hauptaufgaben besprochen werden:

- Herstellen eines Zusammenhangs zwischen NC-Anweisungen und gemessenen Leistungsdaten der Werkzeugmaschine
- Untersuchung der Möglichkeit den Energiebedarf mittels Machine Learning-Algorithmen vorherzusagen.

10.1 Herstellen eines Zusammenhangs zwischen NC-Anweisung und Verbrauch

Die Herstellung eines Zusammenhangs zwischen NC-Anweisung und gemessenen Leistungsdaten der Werkzeugmaschine stellte den größten Teil der Arbeit dar.

Schritt Eins und Zwei

In den ersten beiden Schritten des Entwicklungsprozesses (s. Kap. 6) wurde im Rahmen eines dezimierten Datensatzes ein erster Zusammenhang zwischen NC-Anweisung und Energieverbrauch hergestellt und anschließend das Verständnis des prototypischen Modells auf die Erkennung von Ausreißern hin verbessert.

Schritt Drei

In dritten Schritt wurde der gesamte Datensatz in Hinblick auf die Verständlichkeit für den Algorithmus geparst, sprich aus dem Fließtext der NC-Anweisungen eine tabellarische Darstellung der wichtigen Daten erstellt (s. Kap 7.1). Die Summe der anschließenden, inkrementellen Verbesserungen der Datenaufarbeitung ist in Abbildung 10.1 abgebildet. Zu sehen ist, dass der Zusammenhang der NC-Anweisungen und der Messergebnisse in so weit verbessert werden konnte, sodass die Abweichungen drastisch verringert werden konnten und der Anteil der erklärten Varianz bis fast auf den Maximalwert von eins gesteigert werden konnte.

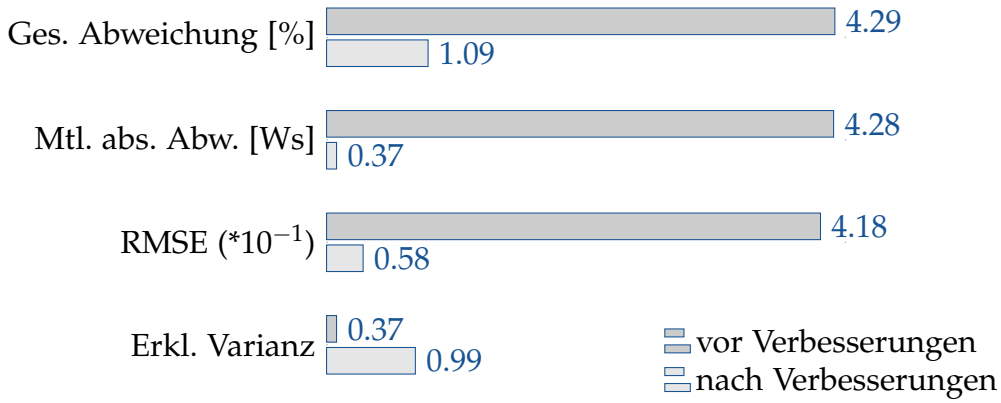


Abbildung 10.1: Vergleich der von den ersten Vorhersagen (z-Achse) zu Vorhersagen nach Verbesserungen des Parsings (s. Kap. 7.1) für bekannte Daten (Quelle: Eigene Darstellung)

10.2 Untersuchung der Möglichkeit den Energiebedarf mittels Machine Learning-Algorithmen vorherzusagen

Anschließend an die ersten Schritte des Entwicklungsprozess (s. Kap. 3.3) wird in der letzten Stufe die generelle Möglichkeit untersucht, ob Machine Learning-Algorithmen zu verwertbaren Vorhersagen des Energieverbrauchs von Werkzeugmaschinen führen. Dafür wurden in Schritt Fünf (s. Kap. 9) unbekannte Daten herangezogen um das in Schritt Vier erstellte Modell zu validieren. Wie in Darstellung 10.2 zu sehen, liefert der Algorithmus trotz zahlreicher Annahmen und Vereinfachungen sehr gute Ergebnisse:

- Der Energieverbrauch wird über den Bearbeitungsverlauf generell richtig dargestellt.
- Der Level der verschiedenen Bearbeitungsmodi wird richtig abgebildet.
- Bei den Verbrauchsspitzen gibt es Betragsabweichungen, der Zeitpunkt in der Bearbeitung wird aber richtig erfasst.

Die erzielten Ergebnisse rechtfertigen die im Prozess getätigten Annahmen und Vereinfachungen:

- Linearisierung des Weges von Kreisinterpolationen (s. Kap. 5.2.1)
- Annahme eines konstanten Spanungsquerschnitt während der Bearbeitung (s. Kap. 5.2.1)
- Geschätzter Grenzwert bei Mustererkennung (s. Kap. 6.3)
- Unbekannte Positionen mit Null ersetzt (s. Kap. 7.2.2)
- Rückschluss von Werkzeugdurchmesser auf Energieverbrauch (s. Kap. 8.1.1)

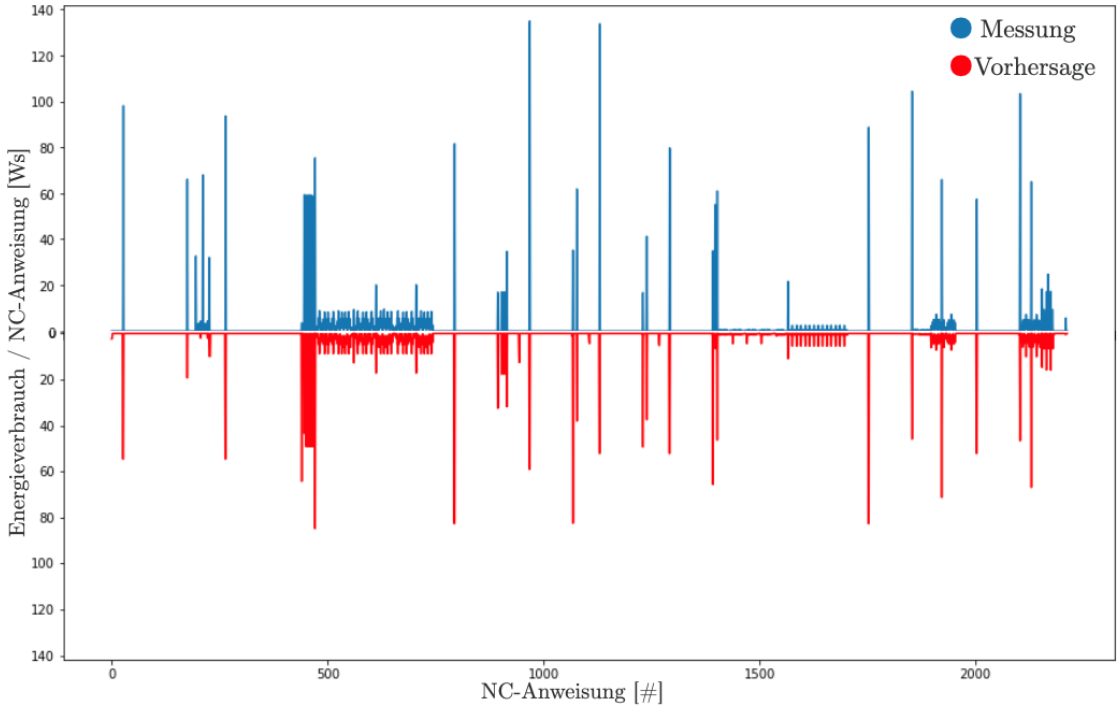


Abbildung 10.2: Messungen und Vorhersagen des Energieverbrauchs des y-Achsen-Aggregats (Quelle: Eigene Darstellung)

Beim Vergleich verschiedener Algorithmen fällt auf, dass selbst in einer Algorithmenfamilie (hier: Entscheidungsbaum) sich die Vorhersagegenauigkeiten stark unterscheiden. Zwar gibt es Algorithmen, welche punktuell, bspw. für ein spezielles Aggregat, eine bessere Vorhersage liefern, global gesehen liefert aber der »RandomForest« die besten Ergebnisse (s. Abb. 10.3).

Wie in Abbildung 10.2 und 10.3 zu sehen, führt der in dieser Arbeit entwickelte Machine Learning-Algorithmus bereits zu sehr guten Ergebnissen. Die in der Aufgabenstellung gestellte Frage nach der Möglichkeit, den Energiebedarf von CNC-Maschinen mithilfe von Machine Learning vorherzusagen, muss also klar bejaht werden. Allerdings ist das Potential bei weitem noch nicht ausgeschöpft: Felder in welchen die Entwicklung fortgesetzt werden könnte, sind im folgenden Kapitel beschrieben.

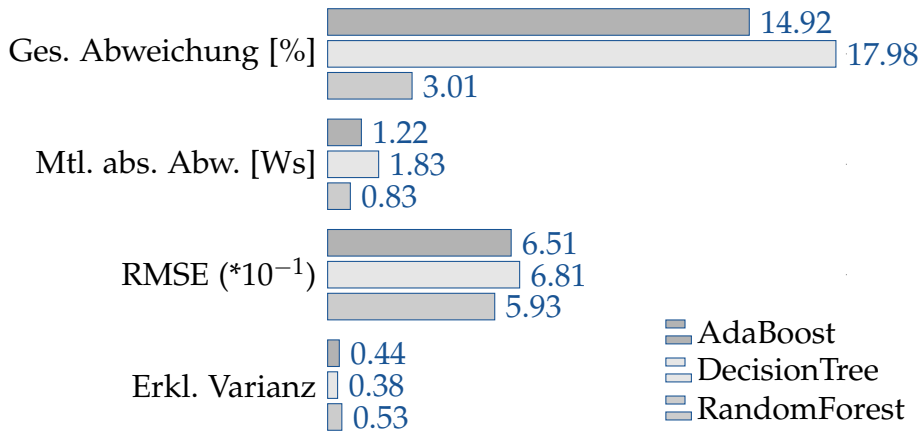


Abbildung 10.3: Vergleich der Vorhersagegüte (y-Achse) verschiedener ML-Algorithmen (Quelle: Eigene Darstellung)

11 Ausblick

Der entwickelte Algorithmus zur Vorhersage des Energiebedarfs von CNC-Maschinen beantwortet die Frage nach der generellen Umsetzbarkeit der Idee, und stellt gleichzeitig den Grundstein dar für weitere Entwicklungen. Während der Arbeit haben sich mehrere Themenfelder aufgezeigt, die für eine weitere Untersuchung interessant erscheinen. Diese lassen sich in die physische und virtuelle Sphäre einteilen (s. Abb. 11.1).

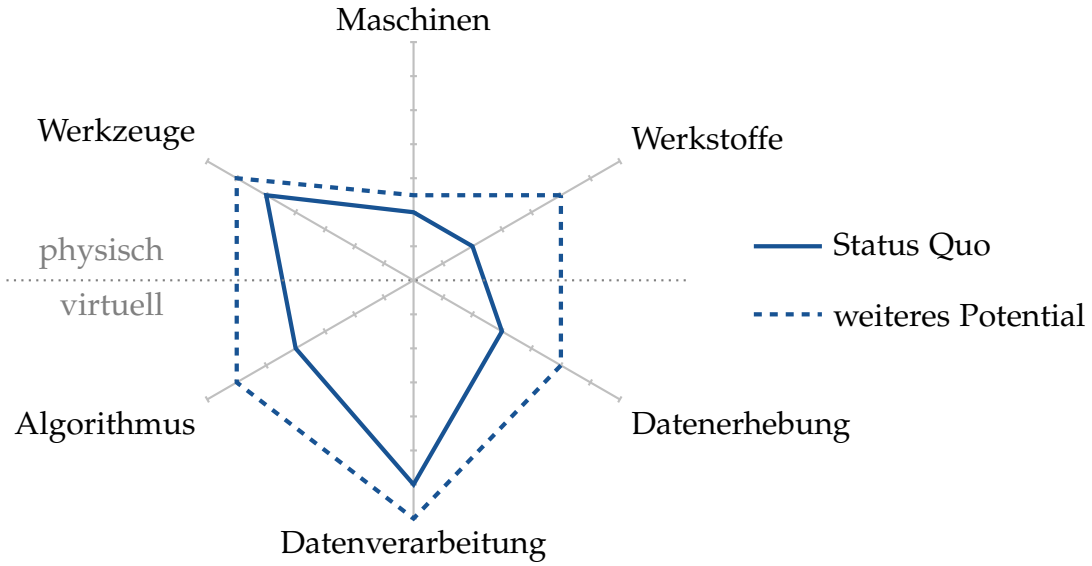


Abbildung 11.1: Beurteilung des Status Quo und des Potentials von ML-Vorhersagen für den Energieverbrauch von CNC-Bearbeitungsmaschinen (Quelle: Eigene Darstellung)

11.1 Physische Sphäre

Im Bereich der physischen Themengebiete bieten sich folgende weitere Möglichkeiten:

Werkstoffe In der vorliegenden Arbeit wurden nur Daten verwendet, welche aus der Bearbeitung eines Werkstoffes stammen. Es bleibt offen, inwieweit es möglich ist, das Modell mit anderen Werkstoffen zu trainieren und diese dann vorherzusagen. Als weiteren Schritt ist denkbar, das Modell mit

Werkstoffdaten zu verknüpfen, sodass aus Trainingsdaten mit mehreren Werkstoffen und einer umfangreichen Werkstoffdatenbank ein Algorithmus trainiert werden kann, welcher auch für nicht getestete Werkstoffe befriedigende Vorhersagen liefert.

Maschinen Alle hier verwendeten Daten stammen von einer einzigen Maschine. Es bietet sich hier an, den Algorithmus derart zu abstrahieren, dass dieser maschinenunabhängig wird. Die maschinenspezifischen Daten könnten dann extern gespeichert und durch weitere Maschinen erweitert werden, sodass der Algorithmus in der Lage wäre Vorhersagen für verschiedene Maschinen zu generieren, auch wenn keine Testdaten für diese zur Verfügung stehen.

Werkzeuge Die entwickelten Features bieten bereits gute Ergebnisse für Werkzeuge, welche der Algorithmus noch nicht kennt (s. Kap. 8.1.1). Falls mehr Daten zur Verfügung stehen würden, könnte eine weitere Verfeinerung beispielsweise durch eine Gruppierung nach Bearbeitungsmodus geschehen (Bohren, Planfräsen, Reiben, etc.).

11.2 Virtuelle Sphäre

Im Bereich der virtuellen Themengebiete bieten sich folgende weitere Möglichkeiten:

Algorithmus

Weitere Algorithmen Es wurden nur Standardalgorithmen untersucht und verglichen. Eine Ausweitung der Untersuchung auf weitere Algorithmen könnte weitere Verbesserungen bringen.

Neuronale Netze Der Einsatz von neuronalen Netzen könnte bessere Vorhersagen liefern. Dieses Thema wurde hier gänzlich außer Acht gelassen.

Eigener Algorithmus Denkbar wäre auch die Entwicklung eines eigenen Algorithmus, welcher beispielsweise erst eine Kategorisierung mit Hilfe eines Entscheidungsbaums vornimmt und anschließend den Energieverbrauch für jede Kategorie mit einer Support Vector Maschine bestimmt (s. Kap. 6.6.1).

Datenverarbeitung

Separierung von zusammengefassten Befehlen In einer NC-Anweisung können mehrere Befehle und Befehlstypen zusammengefasst werden (s. Kap. 7.1.2). Gelänge es diese Befehle voneinander zu trennen könnten für jede Anweisung der Energiebedarf getrennt errechnet und anschließend zusammengefügt werden. Eine Verbesserung der Vorhersagen wäre dabei zu erwarten.

Dimensionslose Features Durch weitere konstruierte Features könnte die Vorhersagequalität weiter verbessert werden. Ein interessanter Ansatz dabei wären dimensionslose Kenngrößen wie den Electrical Energy

Consumption Indicator (EECI) zu verwenden. Dieser bringt zerspantes Volumen V_s, Energieverbrauch E und spezifische Schnittkraft k_{c1} in den Zusammenhang $EECI = \frac{E}{V_s k_{c1}}$.[1]

Datenerhebung

Doppelte Codes Wie in Kapitel 8.1.2 besprochen, werden einige NC-Befehle doppelt aufgezeichnet wobei nur die zweite Aufzeichnung einen signifikanten Energiebedarf aufweist. Einen Behebung dieses Problems bei der Datenerfassung würde die Vorhersage für Leistungsspitzen verbessern.

Momentanleistung statt Energieverbrauch Eine Vereinfachung, welche zu Beginn getroffen wurde (s. Kap. 4.1) hat zum Verlust der zeitlichen Komponente geführt. Möchte man die tatsächliche Momentanleistung vorhersagen, muss über Alternativen nachgedacht werden, die Messergebnisse mit dem NC-Code zu verbinden.

Vorhersage für NC-Code ohne interne Maschinenanweisungen Im verwendeten Datensatz waren die ausgeführten internen Maschinenbefehle mit enthalten. Möchte man im Weiteren den Energieverbrauch allein anhand der NC-Codes, wie sie ein CAM-Programm ausgibt, vorhersagen, muss vorerst die Frage geklärt werden, wie mit dem Energieverbrauch verfahren wird, welcher durch eben diese internen Anweisungen verursacht wird.

[1] Haas u. a., 2019.

Literatur

Abele, E. u. a. (2011). »Analyzing Energy Consumption of Machine Tool Spindle Units and Identification of Potential for Improvements of Efficiency«. In: *Glocalized Solutions for Sustainability in Manufacturing: Proceedings of the 18th CIRP International 280 Conference on Life Cycle Engineering, Technische Universität Braunschweig, Braunschweig, Germany, May 2nd - 4th, 2011*. Hrsg. von J. Hesselbach und C. Herrmann. DOI: 10.1007/978-3-642-19692-8_49 (siehe S. 2).

Arnold, F. u. a. (Dez. 2017). »New Approaches for the Determination of Specific Values for Process Models in Machining Using Artificial Neural Networks«. In: *Procedia Manufacturing* 11, S. 1463–1470. DOI: 10.1016/j.promfg.2017.07.277 (siehe S. 14).

Bhinge, Raunak, Nishant Biswas u. a. (Okt. 2014). *An Intelligent Machine Monitoring System for Energy Prediction Using a Gaussian Process Regression* (siehe S. 14).

Bhinge, Raunak, Jinkyoo Park u. a. (Nov. 2016). »Toward a Generalized Energy Prediction Model for Machine Tools«. In: *Journal of Manufacturing Science and Engineering* 139.4. 041013. ISSN: 1087-1357. DOI: 10.1115/1.4034933. URL: https://doi.org/10.1115/1.4034933 (siehe S. 14–16).

Breiman, Leo (1996). »Stacked regressions«. In: *Machine Learning* 24.1, S. 49–64. DOI: 10.1007/BF00117832. URL: https://doi.org/10.1007/BF00117832 (siehe S. 12).

Chakure, Afroz (2019). *Support Vector Machines (SVMs)*. URL: https://medium.com/@aaaanchakure/support-vector-machines-svms-4bcccbd78369 (besucht am 05. 10. 2020) (siehe S. 10).

Cho, Sohyung u. a. (März 2005). »Tool breakage detection using support vector machine learning in a milling process«. In: *International Journal of Machine Tools and Manufacture* 45, S. 241–249. DOI: 10.1016/j.ijmachtools.2004.08.016 (siehe S. 14).

Denkena, Berend u. a. (Juli 2020). »Energy efficient machine tools«. In: *CIRP Annals*. DOI: 10.1016/j.cirp.2020.05.008 (siehe S. 24).

Diaz-Elsayed, Nancy, Elena Redelsheimer und David Dornfeld (Jan. 2011). »Energy Consumption Characterization and Reduction Strategies for Milling Machine Tool Use«. In: DOI: 10.1007/978-3-642-19692-8_46 (siehe S. 4).

Drucker, Harris (Aug. 1997). »Improving Regressors Using Boosting Techniques«. In: *Proceedings of the 14th International Conference on Machine Learning* (siehe S. 11).

Duffy, Nigel und David Helmbold (Mai 2002). »Boosting Methods for Regression«. In: *Machine Learning* 47, S. 153–200. DOI: 10.1023/A:1013685603443 (siehe S. 11).

Fahrmeir, Ludwig u. a. (2016). *Statistik: Der Weg zur Datenanalyse*. ger. 8. Aufl. 2016. Springer-Lehrbuch. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 9783662503713 (siehe S. 12, 13).

Fang, Kan u. a. (Okt. 2011). »A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction«. In: *Journal of Manufacturing Systems - J MANUF SYST* 30, S. 234–240. DOI: 10.1016/j.jmsy.2011.08.004 (siehe S. 2).

Fenner, Mark E. (2020). *Machine Learning With Python For Everyone*. 1. Aufl. Addison Wesley Data: Analytics Series. Addison-Wesley Professional, Pearson education. ISBN: 9780134845647 (siehe S. 8).

Fritz, Alfred Herbert, Hrsg. (2018). *Fertigungstechnik*. ger. 12., neu bearbeitete und ergänzte Auflage. Springer-Lehrbuch. ISBN: 9783662565346 (siehe S. 1).

Garcia-Ordas, M. (2017). »Wear characterization of the cutting tool in milling processes using shape and texture descriptors«. Ph.D. thesis. Universidad de Leon (siehe S. 14).

Grote, Karl-Heinrich und Jörg Feldhusen, Hrsg. (2014). *Dubbel: Taschenbuch für den Maschinenbau*. 24., aktualisierte Aufl. Springer. ISBN: 9783642388910 (siehe S. 63).

Gutowski, Timothy, Jeffrey Dahmus und Alex Thiriez (2006). »Electrical energy requirements for manufacturing processes«. In: (Siehe S. 3).

Haas, Franz u. a. (Juli 2019). »Platform for Monitoring and Comparing Machining Processes in Terms of Energy Efficiency«. English. In: *Transactions of FAMENA* 43.2, S. 31–47. ISSN: 1333-1124. DOI: 10.21278/TOF.43203 (siehe S. 81).

He, Yan u. a. (Mai 2012). »Analysis and estimation of energy consumption for numerical control machining«. In: *Proceedings of the Institution of Mechanical Engineers Part B Journal of Engineering Manufacture* 226, S. 255–266. DOI: 10.1177/0954405411417673 (siehe S. 4–7).

Huang, Potsang, C.-C Ma und C.-H Kuo (Dez. 2015). »A PNN self-learning tool breakage detection system in end milling operations«. In: *Applied Soft Computing Journal* 37, S. 114–124. DOI: 10.1016/j.asoc.2015.08.019 (siehe S. 14).

Javed, Kamran u. a. (Dez. 2018). »Tool Wear Monitoring and Prognostics Challenges: A Comparison of Connectionist Methods toward an Adaptive Ensemble Model«. In: *J. Intell. Manuf.* 29.8, S. 1873–1890. ISSN: 0956-5515. DOI: 10.1007/s10845-016-1221-2. URL: https://doi.org/10.1007/s10845-016-1221-2 (siehe S. 14).

Kim, Dong-Hyeon u. a. (Aug. 2018). »Smart Machining Process Using Machine Learning: A Review and Perspective on Machining Industry«. In: *International Journal of Precision Engineering and Manufacturing-Green Technology* 5, S. 555–568. DOI: 10.1007/s40684-018-0057-y (siehe S. 14).

Klancnik, Simon, M. Brezocnik und J. Balic (März 2016). »Intelligent CAD/CAM system for programming of CNC machine tools«. In: *International Journal of Simulation Modelling* 15, S. 109–120. DOI: 10.2507/IJSIMM15(1)9.330 (siehe S. 14).

Krishnakumar, P., Rameshkumar Krishnaswamy und Ramachandran K I (Dez. 2015). »Tool Wear Condition Prediction Using Vibration Signals in High Speed Machining (HSM) of Titanium (Ti-6Al-4 V) Alloy«. In: *Procedia Computer Science* 50. DOI: 10.1016/j.procs.2015.04.049 (siehe S. 14).

Kummer, Nikolai und H. Najjaran (Okt. 2014). »Adaboost.MRT: Boosting regression for multivariate estimation«. In: *Artificial Intelligence Research* 3. DOI: 10.5430/air.v3n4p64 (siehe S. 66).

Kvålseth, Tarald (März 2012). »Cautionary Note About R2«. In: *The American Statistician* 39, S. 279–285. DOI: 10.1080/00031305.1985.10479448 (siehe S. 13).

Machado, Gustavo, Mariana Recamonde-Mendoza und Luís Corbellini (Juli 2015). »What variables are important in predicting bovine viral diarrhea virus? A random forest approach«. In: *Veterinary Research* 46, S. 85. DOI: 10.1186/s13567-015-0219-7 (siehe S. 11).

Mitchell, Tom M (2010). *Machine learning*. eng. Internat. ed., 24. [print.]. McGraw-Hill series in computer science : Artificial intelligence. ISBN: 9780071154673 (siehe S. 7).

Pavanaskar, Sushrut S. (2014). »Improving Energy Efficiency in CNC Machining«. Dissertation. University of California, Berkeley (siehe S. 3).

Peng, Chong, Lun Wang und T. Liao (Okt. 2015). »A new method for the prediction of chatter stability lobes based on dynamic cutting force simulation model and support vector machine«. In: *Journal of Sound and Vibration* 354, S. 118–131. DOI: 10.1016/j.jsv.2015.06.011 (siehe S. 14).

Rocca, Joseph (2019). *Ensemble methods: bagging, boosting and stacking*. URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (siehe S. 11).

Subramanian, Dhilip (2019). *A Simple Introduction to K-Nearest Neighbors Algorithm*. URL: https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e (besucht am 05. 10. 2020) (siehe S. 9).

Wu, Dazhong u. a. (Apr. 2017). »A Comparative Study on Machine Learning Algorithms for Smart Manufacturing: Tool Wear Prediction Using Random Forests«. In: *Journal of Manufacturing Science and Engineering* 139 (siehe S. 14).

Xiaohong, Lu u. a. (2020/10/18 2016). »Research on the prediction model of micro-milling surface roughness of Inconel718 based on SVM«. In: 68.2, S. 206–211. DOI: 10.1108/ILT-06-2015-0079. URL: https://doi.org/10.1108/ILT-06-2015-0079 (siehe S. 14).

Yuan, Ye u. a. (Feb. 2017). »Bayesian Learning-Based Model-Predictive Vibration Control for Thin-Walled Workpiece Machining Processes«. In: *IEEE/ASME*

Transactions on Mechatronics 22, S. 509–520. DOI: 10.1109/TMECH.2016.2620987 (siehe S. 14).

Appendix

Im Anhang enthalten

- Code des prototypischen Modells
- Code des automatischen Parsers
- Code zur Erstellung des Machine-Learning Modells
- Code zur Validierung des erstellten Models mit neuen Test-Daten

prototype_verfahrweg-Copy1

October 22, 2020

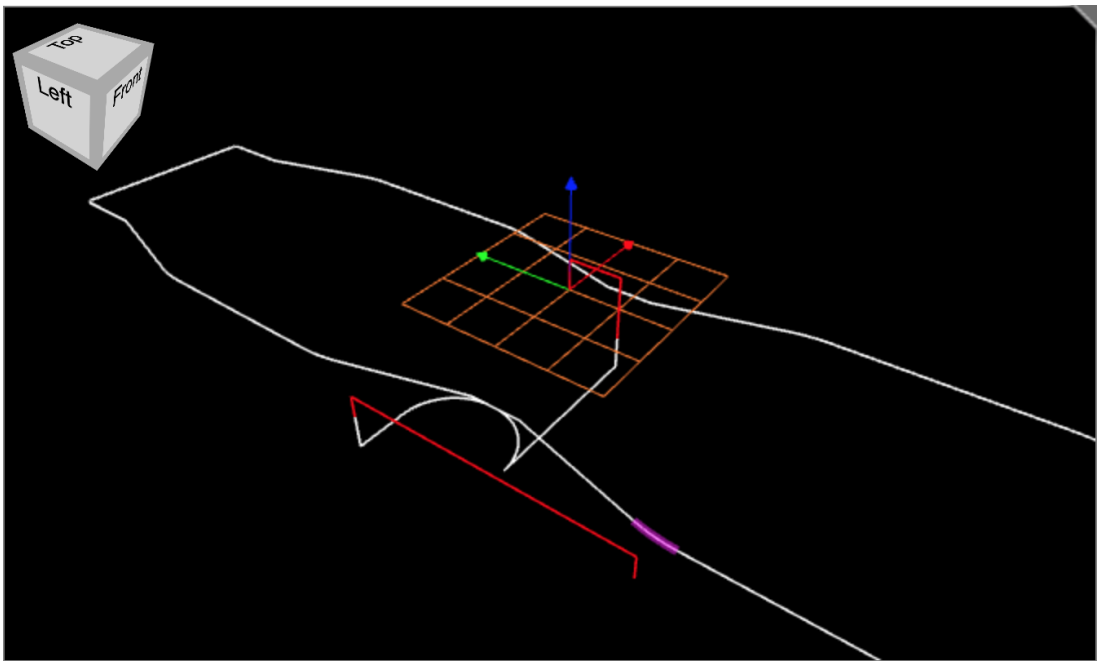
1 Prototyp

1.1 Importing gcode in csv-file

[1]:
```
#import data from csv file for further processing
import pandas as pd
import numpy as np

#small dataset
#td_filepath = './dataset4_sorted.csv'
#extended dataset
td_filepath = './dataset4 extended/dataset4_sorted_extended.csv'
td = pd.read_csv(td_filepath, index_col=['index'])
td.rename(columns={'Energy_consumed_kWs':'Energy_consumed_Ws'}, inplace=True)
td.head()


#td.tail()
#td.columns
```

[1]:

index	G	X	Y	Z	I	J	F	Energy_consumed_Ws
N288	0	0.0	0.0	0.0	NaN	NaN	0	0
N290	0	0.0	0.0	3.0	NaN	NaN	0	17
N292	0	0.0	-5.0	3.0	NaN	NaN	0	43
N294	0	0.0	-5.0	-3.0	NaN	NaN	0	16
N296	1	0.0	-5.0	-6.0	NaN	NaN	1000	25



1.2 Make information in data visible for ML-model

[2]:
```
# inspect data here

td.describe()
```

[2]:

	G	X	Y	Z	I	J	\
count	97.000000	97.000000	97.000000	97.000000	42.000000	41.000000	
mean	1.391753	-3.380144	-1.503289	-4.237113	-0.193643	0.281146	
std	0.757693	12.131840	34.396031	1.977863	9.589312	2.977698	
min	0.000000	-19.907000	-62.907000	-6.000000	-14.300000	-7.300000	
25%	1.000000	-13.300000	-18.920000	-6.000000	-7.941000	-1.889000	
50%	1.000000	-9.517000	-2.524000	-3.000000	0.000000	0.000000	
75%	2.000000	10.406000	15.478000	-3.000000	7.941000	2.148000	
max	3.000000	14.300000	62.907000	3.000000	14.300000	7.300000	

	F	Energy_consumed_Ws
count	97.000000	97.000000
mean	927.835052	157.938144
std	260.105054	211.185040
min	0.000000	0.000000
25%	1000.000000	17.000000
50%	1000.000000	41.000000
75%	1000.000000	306.000000
max	1000.000000	776.000000

[3]:
```
#processing data in table to add features
from math import sqrt


#calculating travelled distance
distances = []
distances.append(0)
for i in range(1, len(td.index)):
    distances.append( sqrt(pow((td.X.iloc[i]-td.X.iloc[i-1]), 2) + pow((td.Y.
 ↪iloc[i]-td.Y.iloc[i-1]), 2))) #+ pow((td.Z.iloc[i]-td.Z.iloc[i-1]), 2)
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['traveled_distance'] = distances
except:
    pass

#create delta_z for differentiated calculations
delta_z = []
delta_z.append(0)
for i in range(1, len(td.index)):
    delta_z.append(td.Z.iloc[i] - td.Z.iloc[i-1])
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['delta_z'] = delta_z
except:
    pass

#calculating delta F
delta_F = []
delta_F.append(0)
for i in range(1, len(td.index)):
    delta_F.append( td.F.iloc[i] - td.F.iloc[i-1] )
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['delta_F'] = delta_F
except:
    pass

td.head(7)
#td.tail()
```

[3]:

index	G	X	Y	Z	I	J	F	Energy_consumed_Ws	\
N288	0	0.000	0.0	0.0	NaN	NaN	0	0	
N290	0	0.000	0.0	3.0	NaN	NaN	0	17	
N292	0	0.000	-5.0	3.0	NaN	NaN	0	43	
N294	0	0.000	-5.0	-3.0	NaN	NaN	0	16	

```
N296   1   0.000 -5.0 -6.0  NaN  NaN  1000                25
N298   1 -15.406 -5.0 -6.0  NaN  NaN  1000                50
N300   3 -10.406  0.0 -6.0  0.0  5.0  1000               156
```

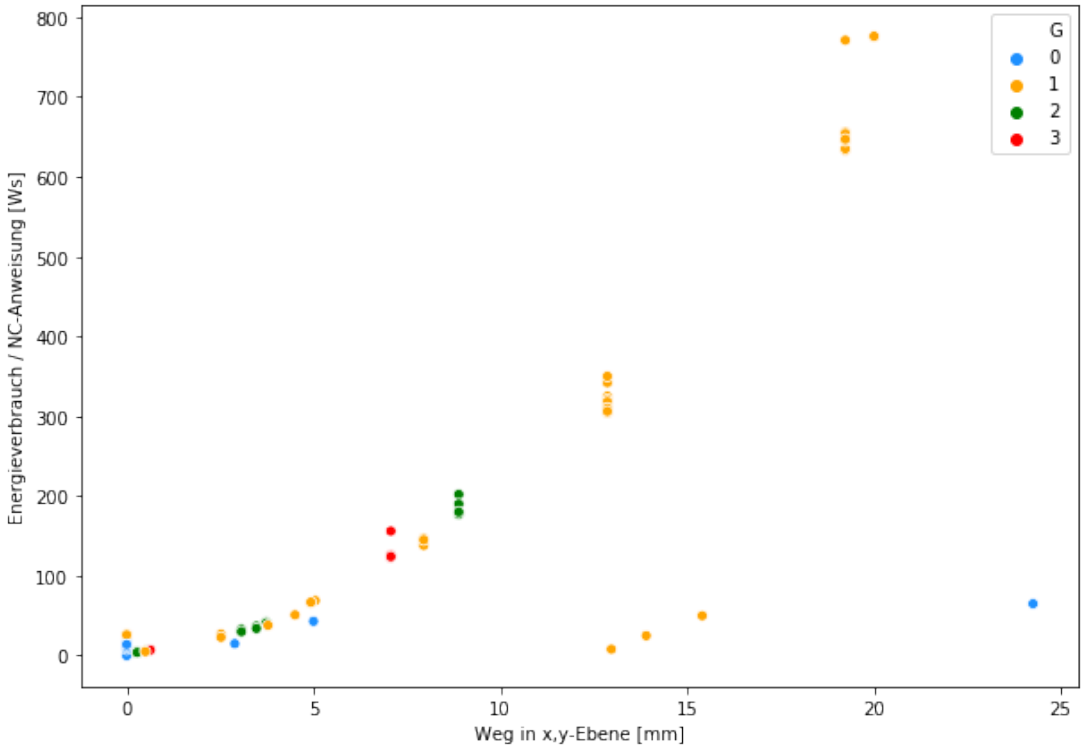
index	traveled_distance	delta_z	delta_F
N288	0.000000	0.0	0
N290	0.000000	3.0	0
N292	5.000000	0.0	0
N294	0.000000	-6.0	0
N296	0.000000	-3.0	1000
N298	15.406000	0.0	0
N300	7.071068	0.0	0

[4]:
```
#safe csv for external inspection
td.to_csv (r'./dataset4_export_from_pandas.csv', index = False, header=True)
```

1.3 Visualize Data and show dependencies

[5]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Examine the data by sight

#td.plot(figsize=(30,10))

plt.figure(figsize = (10, 7))
sns.scatterplot(x = td['traveled_distance'],
                y = td['Energy_consumed_Ws'],
                hue = td['G'],
                palette=['dodgerblue','orange', 'green','red'],)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel('Energieverbrauch / NC-Anweisung [Ws]')
```

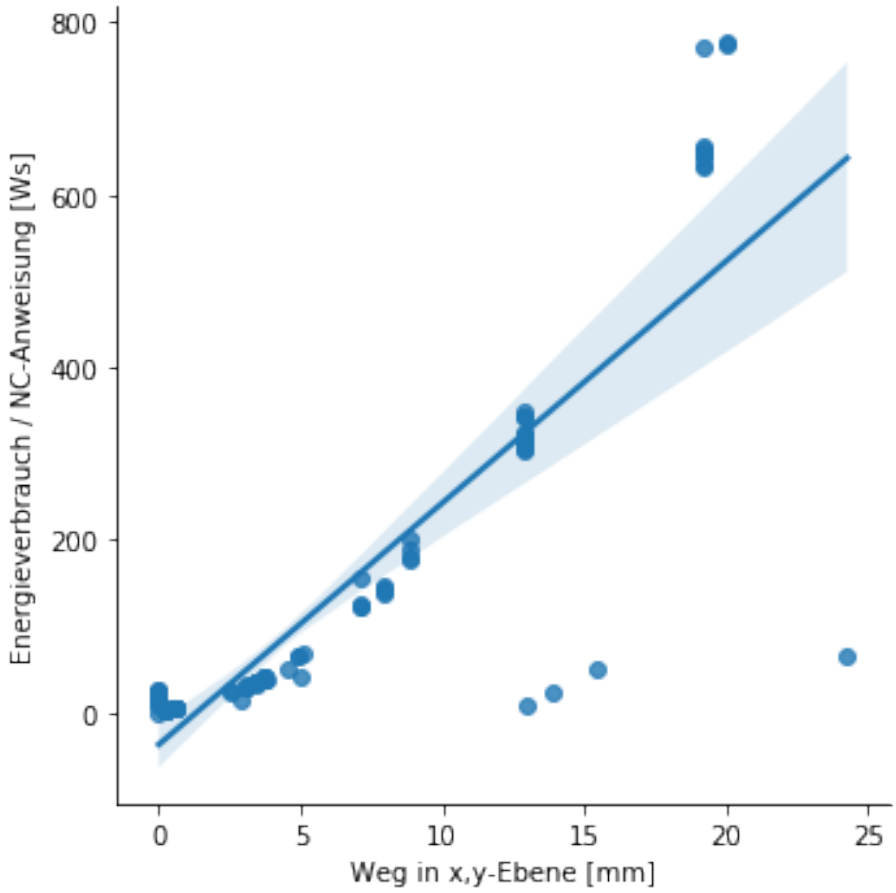
[5]:
```
Text(0, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]')
```



[6]:
```
plt.figure(figsize = (10, 7))

sns.lmplot(x = 'traveled_distance',
           y = 'Energy_consumed_Ws',
           #hue = 'G',
          data = td)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel('Energieverbrauch / NC-Anweisung [Ws]')
```

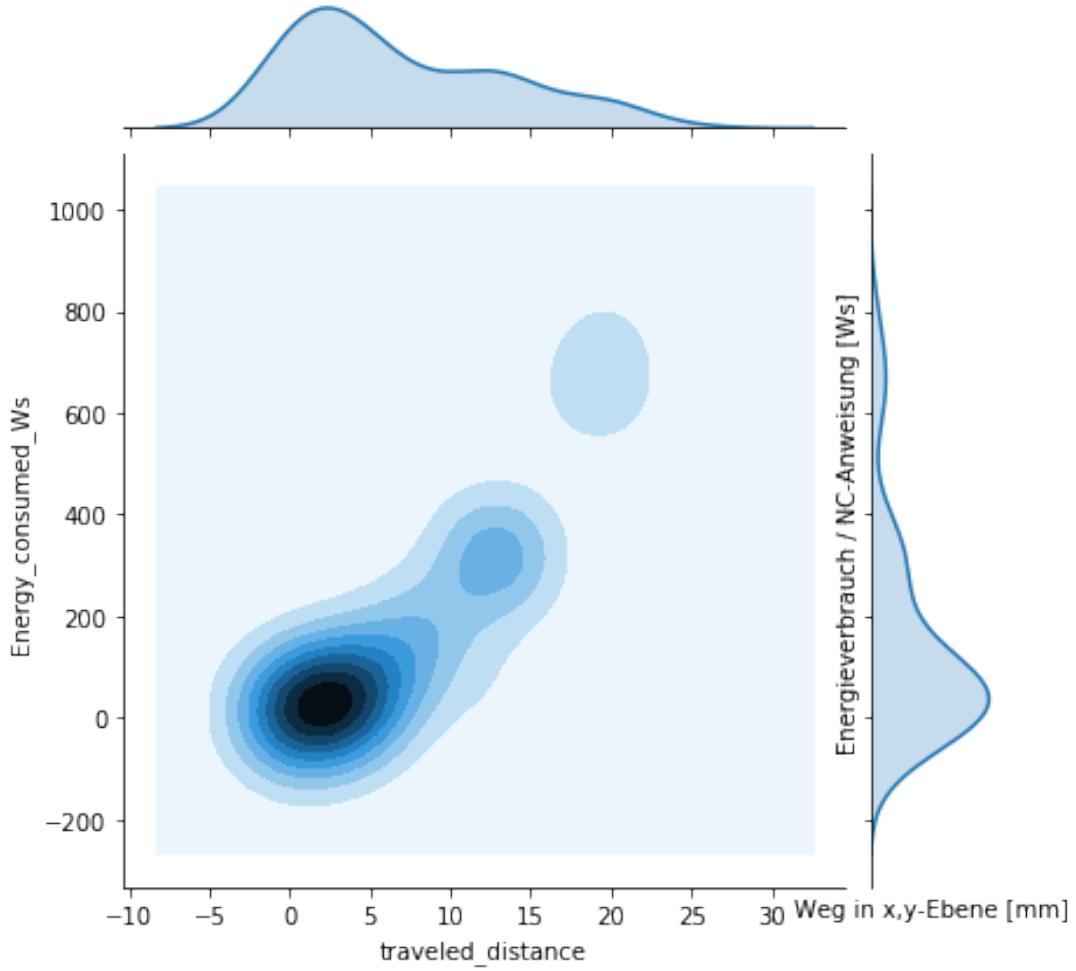
[6]: Text(3.799999999999997, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]')

<Figure size 720x504 with 0 Axes>



[7]:
```
sns.jointplot(x = td['traveled_distance'],
              y = td['Energy_consumed_Ws'],
              kind="kde")
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel('Energieverbrauch / NC-Anweisung [Ws]')
# Fazit: Problem bei kleinen und großen Wegen
```

[7]:
```
Text(336.9714285714286, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]')
```



1.4 Split dataset and train model

[8]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


td_cleaned = td.iloc[1:,]
y = td_cleaned.Energy_consumed_Ws
td_features = ['G', 'traveled_distance', 'delta_z'] #  , 'delta_F', if included␣
 ↪results get worse. F is is feed speed and get's ramped up every time anyway
X = td_cleaned[td_features]
```

```
train_X, val_X, train_y, val_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 2)
```

[9]:
```
#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)
```

[9]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=1, verbose=0, warm_start=False)
```

1.5 Make predictions and compare to acual values

[10]:
```
#make predictions and make a table to compare results to data

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.meassured[i]) /␣
 ↪comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)

mae = round(mean_absolute_error(val_y, val_predictions), 2)
```

```
sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n----------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'-----------------------------------\n')

#print(td.loc['N396']) #very bad prediciton
#print(td.loc['N294']) #very bad prediction
#print(td.loc['N392'])

#Ausreiser über 100%: N400, N398, N478, N394
#über 20%: N408, N294, N432, N396, N386, N324, N290, N356, N292, N354, N324,␣
 ↪N302, N476, N356
```

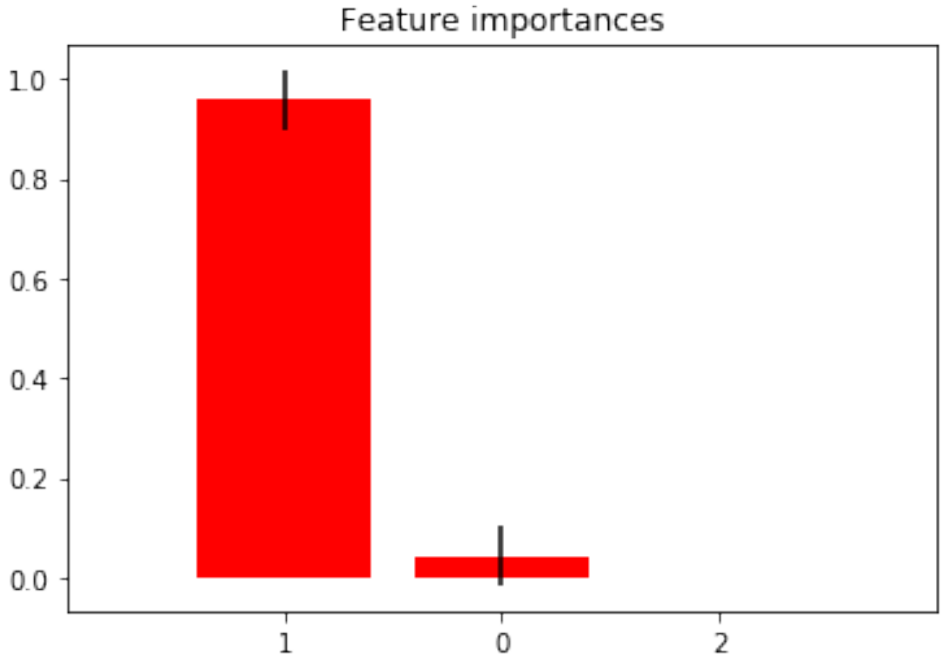
index	G	traveled_distance	delta_z	meassured	predictions	deviation
N362	1	20.006000	0.0	776	661.927980	-0.147000
N346	1	12.867360	0.0	315	320.398381	0.017138
N398	1	13.906000	0.0	25	127.055513	4.082221
N336	2	3.732904	0.0	42	40.093421	-0.045395
N322	1	20.006000	0.0	775	661.927980	-0.145899
N450	2	3.471682	0.0	36	35.582191	-0.011606
N294	0	0.000000	-6.0	16	13.209333	-0.174417
N340	2	0.276921	0.0	4	4.167031	0.041758
N458	2	8.888127	0.0	180	187.508198	0.041712
N316	2	0.276921	0.0	4	4.167031	0.041758

```
-----------------------------------
mean absolut error: 34.76 Ws.
mean of deviations: ± 47.49%
mae/mean_meassured: ± 16.0%
-----------------------------------
```

[11]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier
```

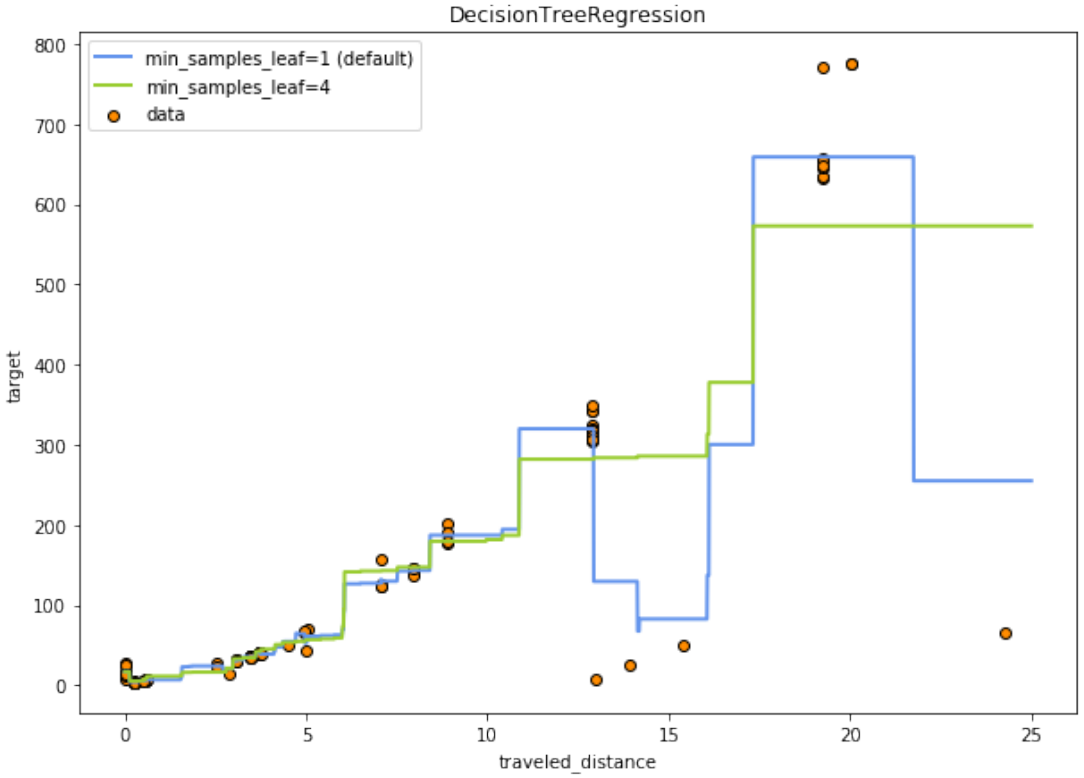
```
# # Build a classification task using 3 informative features
# X, y = make_classification(n_samples=1000,
#                            n_features=10,
#                            n_informative=3,
#                            n_redundant=0,
#                            n_repeated=0,
#                            n_classes=2,
#                            random_state=0,
#                            shuffle=False)

# # Build a forest and compute the impurity-based feature importances
# forest = ExtraTreesClassifier(n_estimators=250,
#                               random_state=0)

# forest.fit(X, y)

importances = test_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in test_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
        color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.956623)
2. feature 0 (0.043346)
3. feature 2 (0.000031)
```



[12]:
```
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# # Create a random dataset
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = train_X[['traveled_distance']]

# Fit regression model
regr_1 = RandomForestRegressor(min_samples_leaf=1)
regr_2 = RandomForestRegressor(min_samples_leaf=4)
regr_1.fit(red_train_X, train_y)
regr_2.fit(red_train_X, train_y)


# Predict
X_range = np.arange(0.0, 25.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_range)
y_2 = regr_2.predict(X_range)
```

```
#inter_X = val_X[['traveled_distance']].copy()
#inter_X.sort_values(by=['traveled_distance'], inplace=True)
#y_1 = regr_1.predict(inter_X)
#y_2 = regr_2.predict(inter_X)

# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="min_samples_leaf=1 (default)", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="min_samples_leaf=4",␣
 ↪linewidth=2)
plt.xlabel("traveled_distance")
plt.ylabel("target")
plt.title("DecisionTreeRegression")
plt.legend()
plt.show()
```



[13]:
```
#X_grid = np.arange(min(td_cleaned.traveled_distance), max(td_cleaned.
 ↪traveled_distance), 0.01)
#X_grid = X_grid.reshape((len(X_grid), 1))
#plt.scatter(X, y, color = 'red')
#plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
#plt.title('Visualize Regressor')
#plt.xlabel('distance')
#plt.ylabel('energy_consumption')
#plt.show()
```

1.6 Trying to catch outlier

[14]:
```
#create Anfahren column
ramp_up = []
ramp_up.append(0)
total_distance = 0
grenz_distance = 15

#G   trav   total
#G0   5       0
#G1   10      10
#G1   10      20

#reset total distance if G = 0, and set ramp up to true if total is smaller than␣
 ↪grenz
for i in range(1, len(td.index)):
    if total_distance <= grenz_distance:
        if td.G.iloc[i] == 0:
            ramp_up.append(0)
        else:
            ramp_up.append(1)
    else:
        ramp_up.append(0)

    if td.G.iloc[i] == 0:
        total_distance = 0
    else:
        total_distance += td.traveled_distance.iloc[i]



try:
    #td.insert(9, 'delta_Z', delta_z)
    td['ramp_up'] = ramp_up
except:
```

```
    pass

td.head(7)
```

[14]:

index	G	X	Y	Z	I	J	F	Energy_consumed_Ws	traveled_distance	delta_z	delta_F	ramp_up
N288	0	0.000	0.0	0.0	NaN	NaN	0	0	0.000000	0.0	0	0
N290	0	0.000	0.0	3.0	NaN	NaN	0	17	0.000000	3.0	0	0
N292	0	0.000	-5.0	3.0	NaN	NaN	0	43	5.000000	0.0	0	0
N294	0	0.000	-5.0	-3.0	NaN	NaN	0	16	0.000000	-6.0	0	0
N296	1	0.000	-5.0	-6.0	NaN	NaN	1000	25	0.000000	-3.0	1000	1
N298	1	-15.406	-5.0	-6.0	NaN	NaN	1000	50	15.406000	0.0	0	1
N300	3	-10.406	0.0	-6.0	0.0	5.0	1000	156	7.071068	0.0	0	0

[15]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


td_cleaned = td.iloc[1:,]
y = td_cleaned.Energy_consumed_Ws
td_features = ['G', 'traveled_distance', 'F', 'ramp_up'] #  'delta_Z',␣
 ↪'delta_F', if included results get worse. F is is feed speed and get's ramped␣
 ↪up every time anyway
X = td_cleaned[td_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 6)

#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)
```

[15]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=1, verbose=0, warm_start=False)
```

[16]:
```
#make predictions and make a table to compare results to data

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.meassured[i]) /␣
 ↪comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)

mae = round(mean_absolute_error(val_y, val_predictions), 2)

sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n---------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'----------------------------------\n')
```

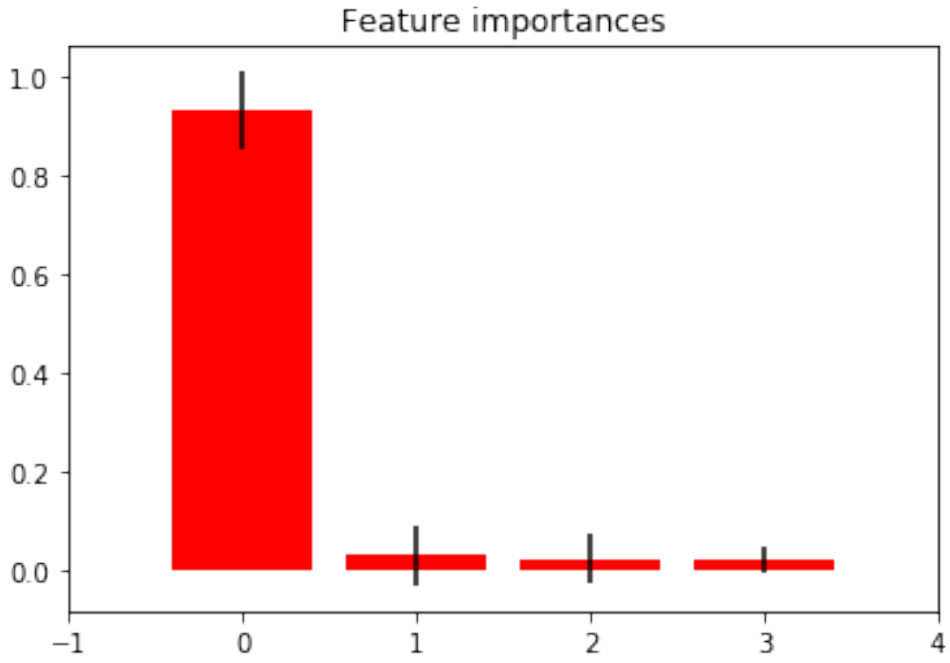
```
      G  traveled_distance     F  ramp_up  meassured  predictions  deviation
```

```
index
N400   1          12.973428  1000        1          8   192.570591  23.071324
N330   1          12.867360  1000        0        317   320.158571   0.009964
N336   2           3.732904  1000        0         42    40.174737  -0.043459
N350   1          19.237000  1000        0        635   665.823453   0.048541
N290   0           0.000000     0        0         17    11.733968  -0.309767
N366   1           7.951000  1000        0        145   142.008917  -0.020628
N364   2           0.569928  1000        0          6     6.271302   0.045217
N418   2           3.471682  1000        0         36    35.540172  -0.012773
N368   2           0.276921  1000        0          4     4.163490   0.040873
N408   3           0.632001  1000        0          7     6.698357  -0.043092

-------------------------------------
mean absolut error: 22.98 Ws.
mean of deviations: ± 236.46%
mae/mean_meassured: ± 18.88%
-------------------------------------
```

[17]:
```python
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier

importances = test_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in test_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
        color="r", yerr=std[indices], align="center")
#plt.xticks(range(X.columns[f]), indices) #X.shape[1]
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.930117)
2. feature 2 (0.029427)
```

```
3. feature 0 (0.021750)
4. feature 3 (0.018706)
```



1.7 Comparing Models

This compares mayn models and gives back overall accuracy.

[18]:
```
from sklearn.impute import SimpleImputer
#from sklearn_pandas import CategoricalImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
 ↪GradientBoostingRegressor
from IPython.display import FileLink



models = [AdaBoostRegressor(learning_rate=2),
          SVR(kernel='linear'),
          RandomForestRegressor(n_estimators=200, random_state=1)]
```

```
errors = []
for model in models:
    model_name = model.__class__.__name__
    error = np.sqrt(abs(cross_val_score(model, train_X, train_y, cv=5,␣
 ↪scoring='neg_mean_squared_error'))).mean()
    errors.append([model_name, error])

result_df = pd.DataFrame(errors, columns=['Model name', 'Average error'])
result_df
```

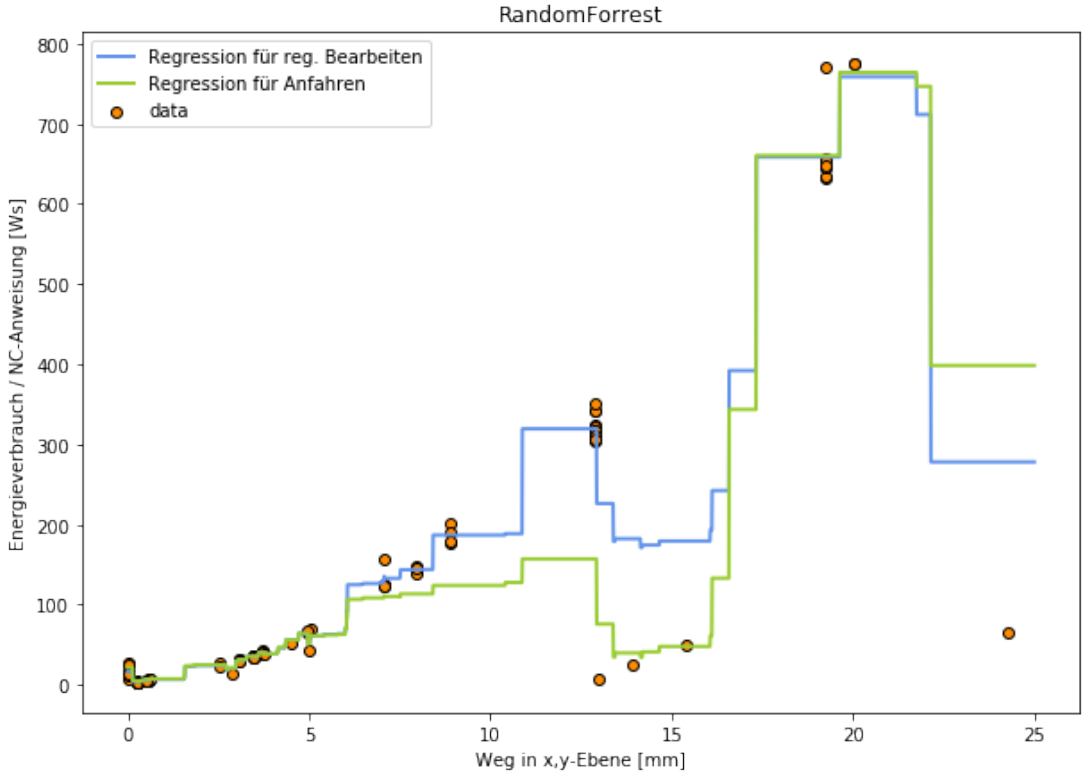
[18]:

	Model name	Average error
0	AdaBoostRegressor	53.844478
1	SVR	101.816042
2	RandomForestRegressor	48.999219

[19]:

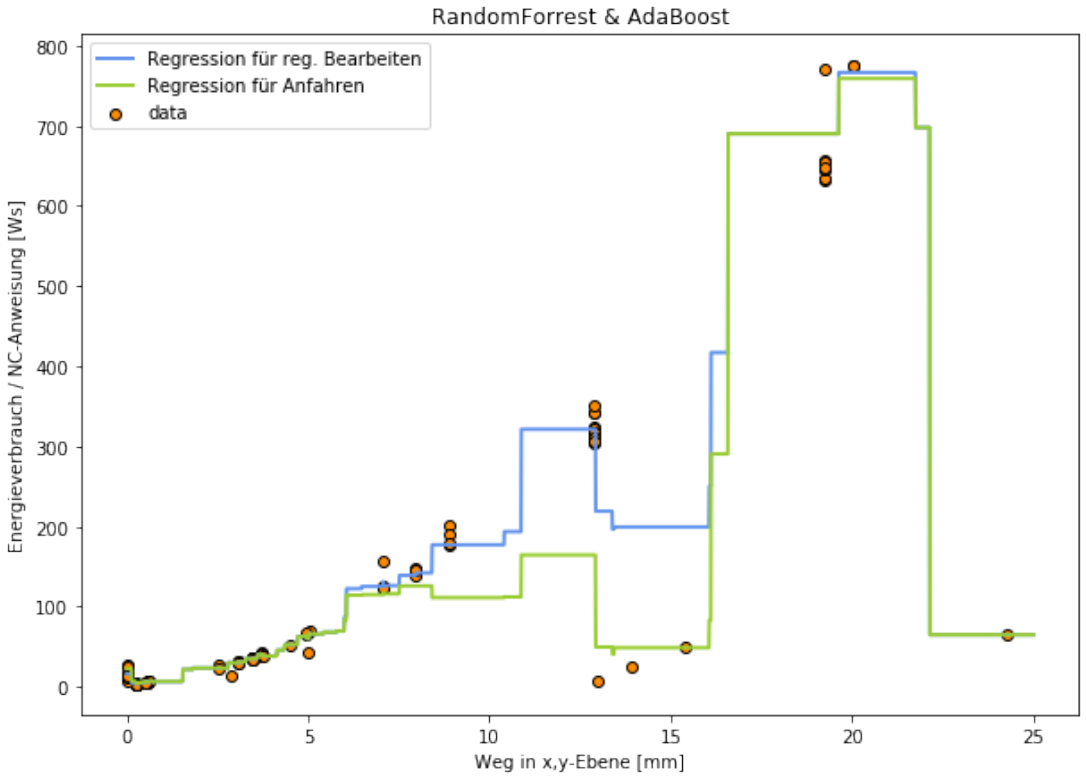
```
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# # Create a random dataset
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = RandomForestRegressor(min_samples_leaf=1)
regr_2 = RandomForestRegressor(min_samples_leaf=1)
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)


# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)
```

```
# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("RandomForrest")
plt.legend()
plt.show()
```



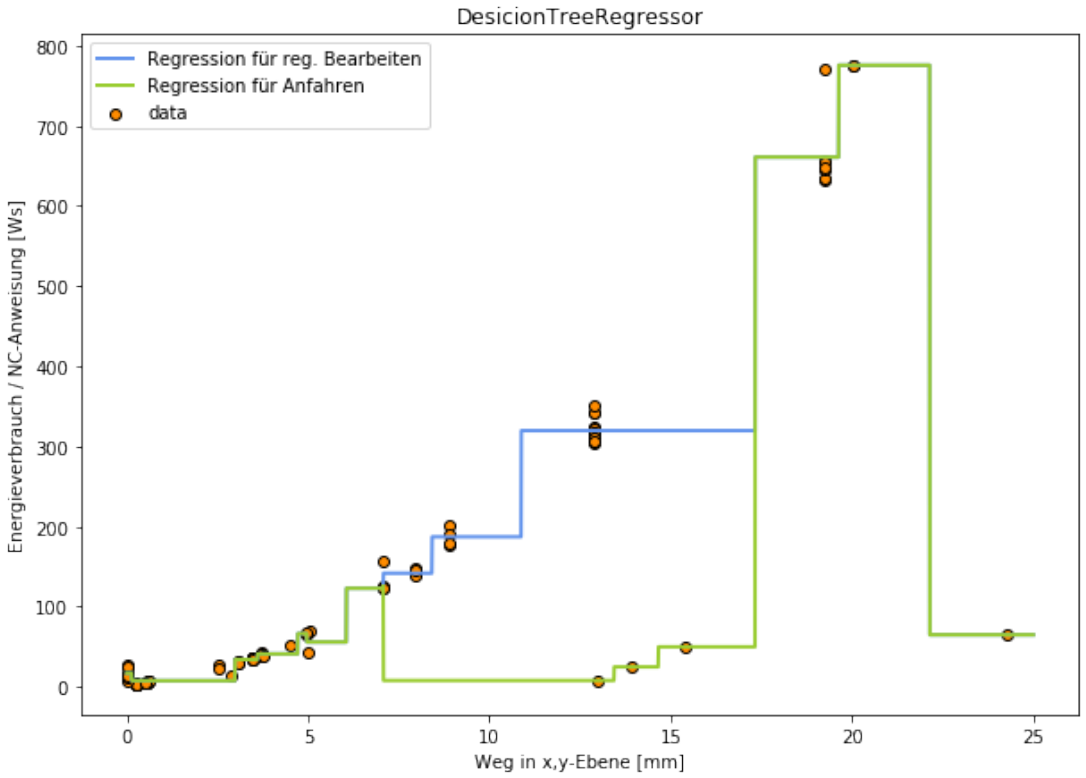
[20]:
```
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# # Create a random dataset
```

```
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = AdaBoostRegressor(RandomForestRegressor(min_samples_leaf=1),
                           n_estimators=300, random_state = 2)
regr_2 = AdaBoostRegressor(RandomForestRegressor(min_samples_leaf=1),
                           n_estimators=300, random_state = 2)
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)


# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("RandomForrest & AdaBoost")
plt.legend()
plt.show()
```



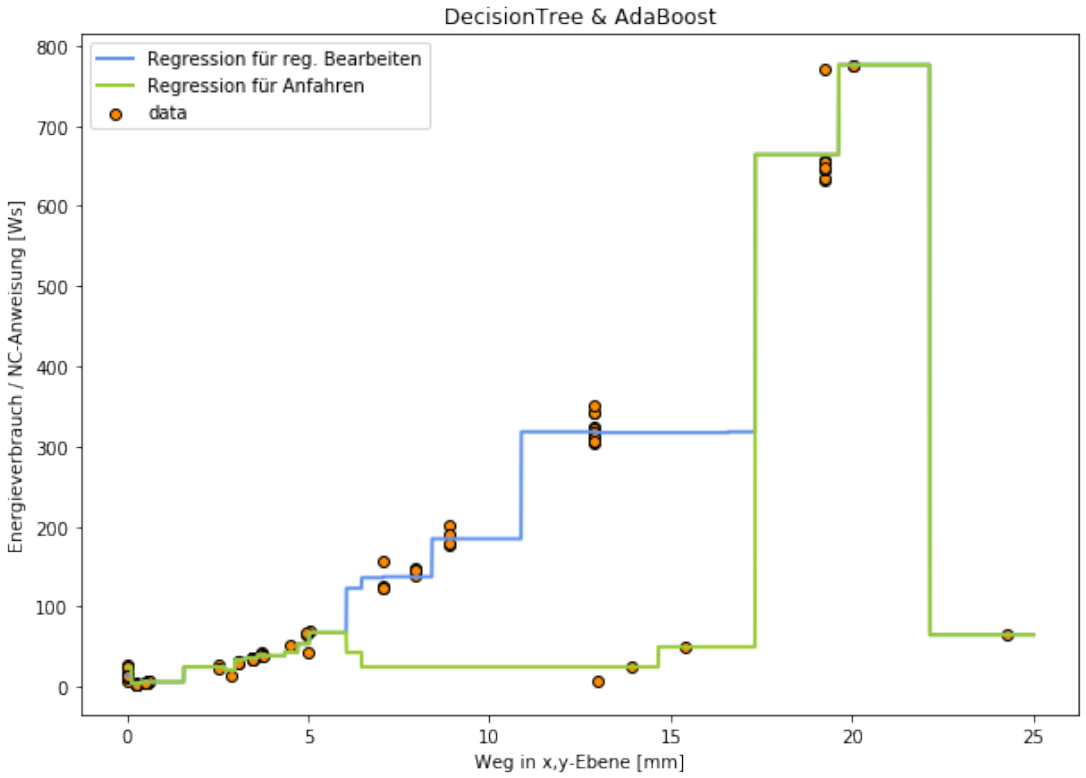
[21]:
```
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostRegressor

# # Create a random dataset
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = DecisionTreeRegressor(min_samples_leaf=1, max_depth=5, random_state =␣
 ↪11)
regr_2 = DecisionTreeRegressor(min_samples_leaf=1, max_depth=5, random_state =␣
 ↪11)
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)
```

```
# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("DesicionTreeRegressor")
plt.legend()
plt.show()
```



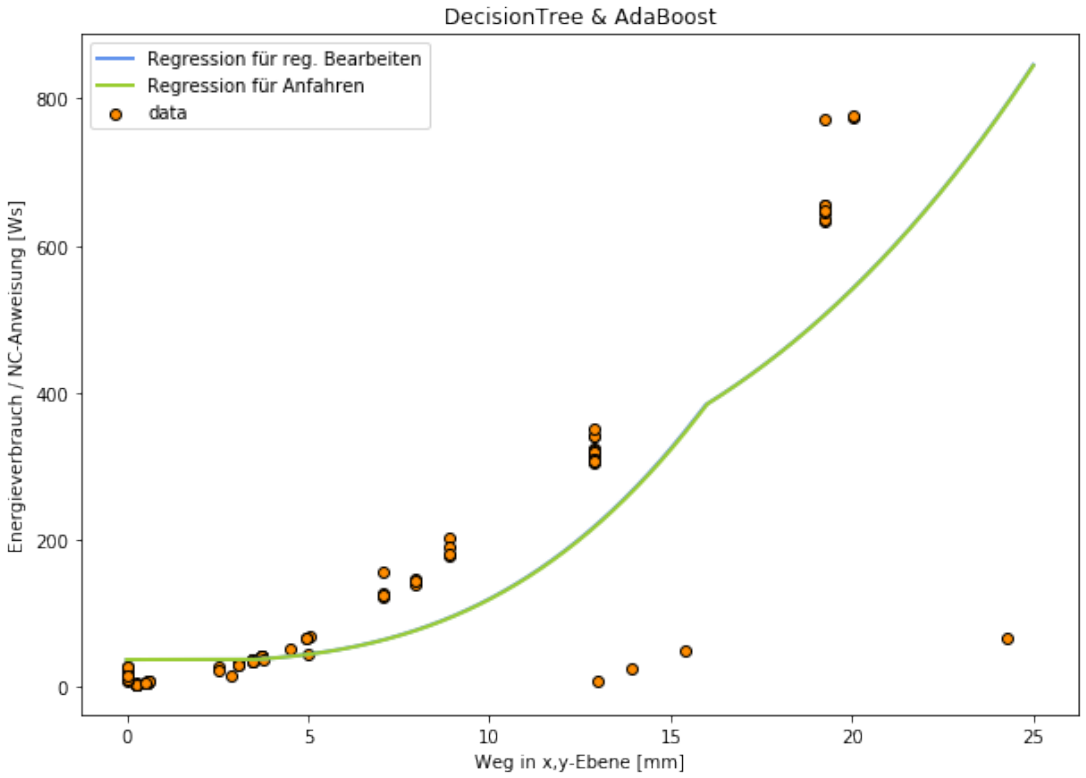
[22]:
```
##  Adaboost

# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostRegressor

# # Create a random dataset
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=5),
                           n_estimators=300, random_state = 1)
regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=5),
                           n_estimators=300, random_state = 1)
```

```
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)


# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("DecisionTree & AdaBoost")
plt.legend()
plt.show()
```



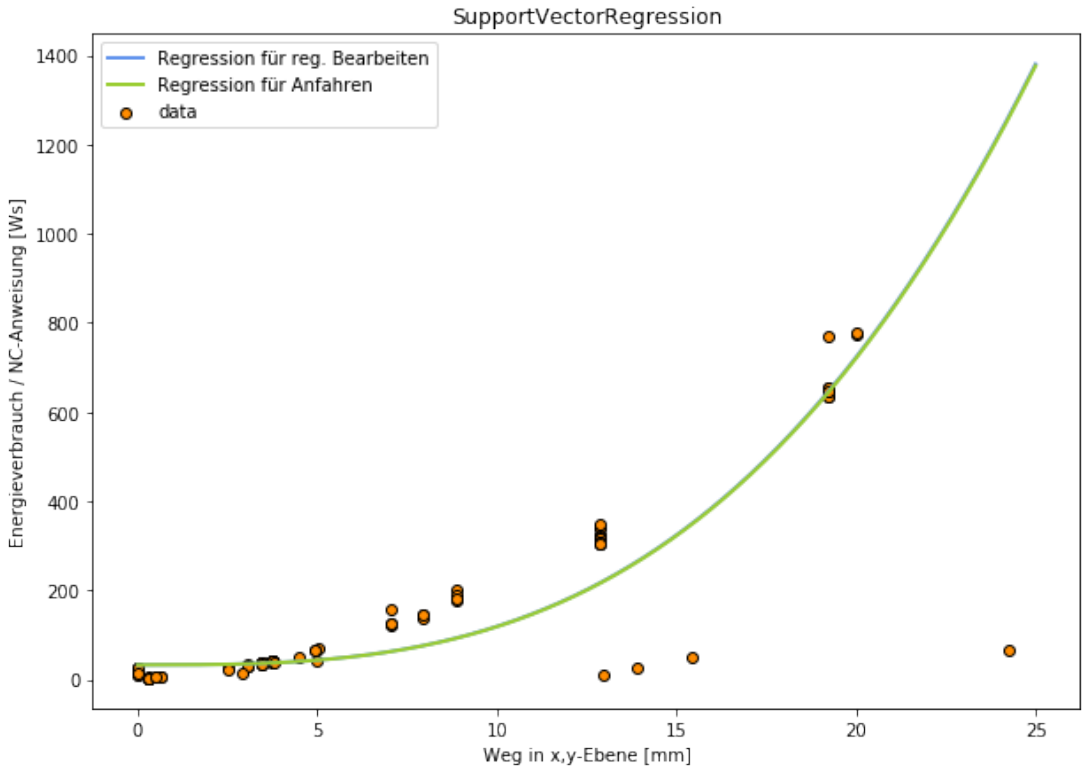
[23]:
```
##  Adaboost & SupportVectorMachine

# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostRegressor
from sklearn import svm

# # Create a random dataset
# rng = np.random.RandomState(1)
# X = np.sort(5 * rng.rand(80, 1), axis=0)
# y = np.sin(X).ravel()
# y[::5] += 3 * (0.5 - rng.rand(16))

red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = AdaBoostRegressor(svm.SVR(kernel='poly'),
                           n_estimators=300, random_state = 1)
regr_2 = AdaBoostRegressor(svm.SVR(kernel='poly'),
```

```
                          n_estimators=300, random_state = 1)
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)


# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("DecisionTree & AdaBoost")
plt.legend()
plt.show()
```



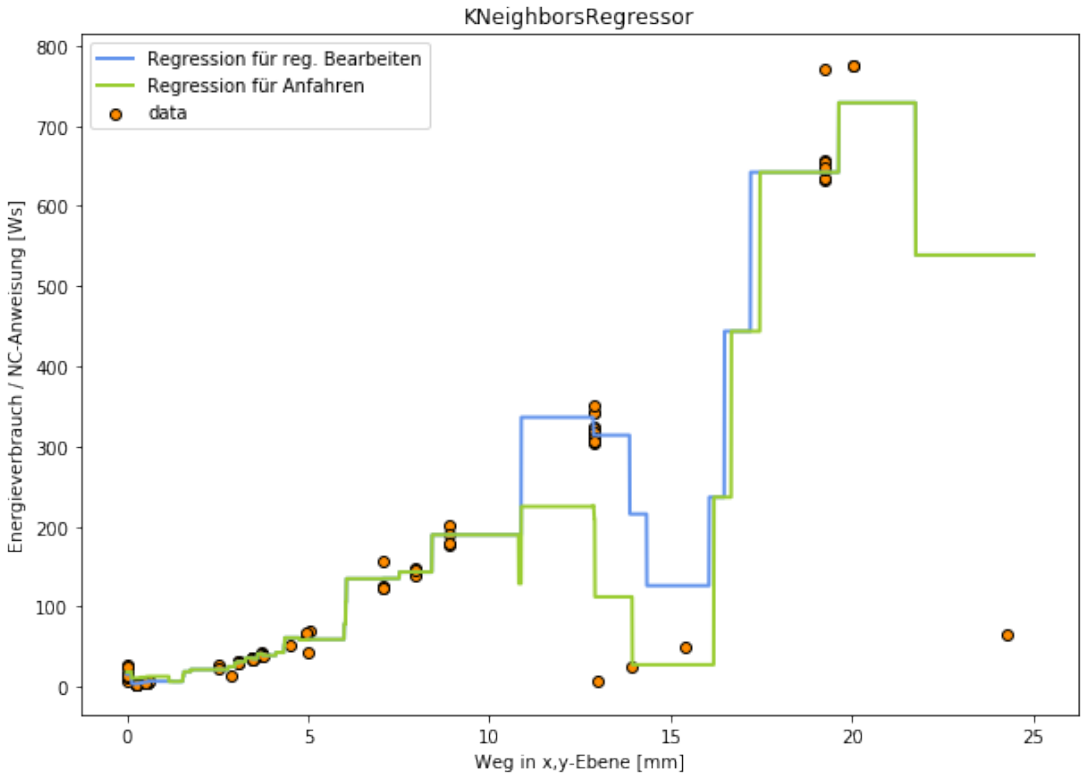
[24]:
```
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn import svm



red_train_X = X[['traveled_distance', 'ramp_up']]



# Fit regression model
regr_1 = svm.SVR(kernel='poly')
regr_2 = svm.SVR(kernel='poly')
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)
```

```
# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("SupportVectorRegression")
plt.legend()
plt.show()
```



[25]:
```
# y_pred = gnb.fit(X_train, y_train).predict(X_test)


##  k neighbors (and Gaussian in Comment)

# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostRegressor
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsRegressor


red_train_X = X[['traveled_distance', 'ramp_up']]

# Fit regression model
regr_1 = KNeighborsRegressor(n_neighbors=3)
regr_2 = KNeighborsRegressor(n_neighbors=3)
regr_1.fit(red_train_X, y)
regr_2.fit(red_train_X, y)
```

```
# Predict
X_range = pd.DataFrame()
X_range['traveled_distance'] = np.arange(0.0, 25.0, 0.01)
X_range0 = X_range.copy()
X_range1 = X_range.copy()

X_range0['ramp_up'] = 0
X_range1['ramp_up'] = 1

y_1 = regr_1.predict(X_range0)
y_2 = regr_2.predict(X_range1)


# Plot the results
plt.figure(figsize = (10, 7))
plt.scatter(X['traveled_distance'], y, edgecolor="black", #, s=20
            c="darkorange", label="data")
plt.plot(X_range, y_1, color="cornflowerblue",
         label="Regression für reg. Bearbeiten", linewidth=2)
plt.plot(X_range, y_2, color="yellowgreen", label="Regression für Anfahren",␣
 ↪linewidth=2)
plt.xlabel("Weg in x,y-Ebene [mm]")
plt.ylabel("Energieverbrauch / NC-Anweisung [Ws]")
plt.title("KNeighborsRegressor")
plt.legend()
plt.show()
```



1.8 Boosting Random Forrest

[ ]:

1.9 Dealing with imbalanced data - Oversampling

[26]:
```
ramp_up_df = td_cleaned[td_cleaned['ramp_up'] == 1]
oversampled_td = td_cleaned.append(ramp_up_df)

oversamplingGrade = 1
for i in range(oversamplingGrade):
    oversampled_td = oversampled_td.append(ramp_up_df)

oversampled_td.tail(10)
```

[26]:

index	G	X	Y	Z	I	J	F	Energy_consumed_Ws	\
N478	0	-19.907	-5.000	-1.0	NaN	NaN	1000	15	

```
N480   0 -19.907  -5.000 -5.0 NaN NaN  1000                  14
N296   1   0.000  -5.000 -6.0 NaN NaN  1000                  25
N298   1 -15.406  -5.000 -6.0 NaN NaN  1000                  50
N398   1  -6.000 -18.762 -3.0 NaN NaN  1000                  25
N400   1 -18.971 -19.013 -3.0 NaN NaN  1000                   8
N296   1   0.000  -5.000 -6.0 NaN NaN  1000                  25
N298   1 -15.406  -5.000 -6.0 NaN NaN  1000                  50
N398   1  -6.000 -18.762 -3.0 NaN NaN  1000                  25
N400   1 -18.971 -19.013 -3.0 NaN NaN  1000                   8
```

index	traveled_distance	delta_z	delta_F	ramp_up
N478	2.892000	0.0	0	0
N480	0.000000	-4.0	0	0
N296	0.000000	-3.0	1000	1
N298	15.406000	0.0	0	1
N398	13.906000	0.0	1000	1
N400	12.973428	0.0	0	1
N296	0.000000	-3.0	1000	1
N298	15.406000	0.0	0	1
N398	13.906000	0.0	1000	1
N400	12.973428	0.0	0	1

[27]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


y = oversampled_td.Energy_consumed_Ws
td_features = ['G', 'traveled_distance', 'F', 'ramp_up'] #  'delta_Z',␣
 ↪'delta_F', if included results get worse. F is is feed speed and get's ramped␣
 ↪up every time anyway
X = oversampled_td[td_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 1)

#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)

#make predictions and make a table to compare results to data
```

```
from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.meassured[i]) /␣
 ↪comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)

mae = round(mean_absolute_error(val_y, val_predictions), 2)

sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n---------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'-----------------------------------\n')
```

index	G	traveled_distance	F	ramp_up	meassured	predictions	deviation
N296	1	0.000000	1000	1	25	24.753929	-0.009843
N360	2	0.569928	1000	0	6	6.247845	0.041308
N452	1	12.867360	1000	0	310	322.147419	0.039185
N408	3	0.632001	1000	0	7	6.729167	-0.038690
N366	1	7.951000	1000	0	145	143.840779	-0.007995
N368	2	0.276921	1000	0	4	4.169487	0.042372
N396	0	0.000000	0	0	8	14.711821	0.838978
N460	1	12.867360	1000	0	306	322.147419	0.052769
N352	2	3.732904	1000	0	39	40.627900	0.041741

```
N402   3           7.071040  1000        0        126   146.818333   0.165225
N398   1          13.906000  1000        1         25    23.710000  -0.051600

-----------------------------------
mean absolut error: 5.53 Ws.
mean of deviations: ± 12.09%
mae/mean_meassured: ± 6.08%
-----------------------------------
```

1.10 2nd Try: Oversampling without testing the same data

[28]:
```
ramp_up_df = td_cleaned[td_cleaned['ramp_up'] == 1]
ramp_up_df = ramp_up_df.drop(index = 'N398')

oversampled_td = td_cleaned.append(ramp_up_df)

oversamplingGrade = 1
for i in range(oversamplingGrade):
    oversampled_td = oversampled_td.append(ramp_up_df)

oversampled_td.tail(10)
```

[28]:

index	G	X	Y	Z	I	J	F	Energy_consumed_Ws	traveled_distance	delta_z	delta_F	ramp_up
N474	1	-19.906	-7.892	-3.0	NaN	NaN	1000	26	0.000000	0.0	0	0
N476	0	-19.906	-7.892	-1.0	NaN	NaN	1000	11	0.000000	2.0	0	0
N478	0	-19.907	-5.000	-1.0	NaN	NaN	1000	15	2.892000	0.0	0	0
N480	0	-19.907	-5.000	-5.0	NaN	NaN	1000	14	0.000000	-4.0	0	0
N296	1	0.000	-5.000	-6.0	NaN	NaN	1000	25	0.000000	-3.0	1000	1
N298	1	-15.406	-5.000	-6.0	NaN	NaN	1000	50	15.406000	0.0	0	1
N400	1	-18.971	-19.013	-3.0	NaN	NaN	1000	8	12.973428	0.0	0	1
N296	1	0.000	-5.000	-6.0	NaN	NaN	1000	25	0.000000	-3.0	1000	1
N298	1	-15.406	-5.000	-6.0	NaN	NaN	1000	50				
N400	1	-18.971	-19.013	-3.0	NaN	NaN	1000	8				

```
N298            15.406000       0.0         0         1
N400            12.973428       0.0         0         1
```

[29]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


y = oversampled_td.Energy_consumed_Ws
td_features = ['G', 'traveled_distance', 'F', 'ramp_up'] #  'delta_Z',␣
 ↪'delta_F', if included results get worse. F is is feed speed and get's ramped␣
 ↪up every time anyway
X = oversampled_td[td_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 0)

#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)

#make predictions and make a table to compare results to data

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.meassured[i]) /␣
 ↪comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)
```

```
mae = round(mean_absolute_error(val_y, val_predictions), 2)

sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n---------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'----------------------------------\n')
```

index	G	traveled_distance	F	ramp_up	meassured	predictions	deviation
N342	1	5.048000	1000	0	69	60.560750	-0.122308
N474	1	0.000000	1000	0	26	22.178000	-0.147000
N294	0	0.000000	0	0	16	14.369286	-0.101920
N400	1	12.973428	1000	1	8	26.532176	2.316522
N472	1	3.781911	1000	0	38	45.771484	0.204513
N458	2	8.888127	1000	0	180	171.143357	-0.049204
N322	1	20.006000	1000	0	775	732.729895	-0.054542
N422	2	8.888127	1000	0	202	171.143357	-0.152756
N398	1	13.906000	1000	1	25	26.532176	0.061287
N426	2	8.888127	1000	0	177	171.143357	-0.033088
N396	0	0.000000	0	0	8	14.369286	0.796161

```
----------------------------------
mean absolut error: 12.36 Ws.
mean of deviations: ± 36.72%
mae/mean_meassured: ± 8.92%
----------------------------------
```

1.11 Adding 2nd tool

[30]:
```
td_filepath = './dataset4 2nd_tool/Dataset4 2nd_tool.csv'
td = pd.read_csv(td_filepath, index_col=['index'])
td.rename(columns={'Energy_consumed_kWs':'Energy_consumed_Ws'}, inplace=True)
td.head()
```

[30]:

index	G	X	Y	Z	I	J	F	tool A	tool B	Energy_consumed_Ws

```
NaN    NaN  0.0  0.0  0.0  0.0  0.0     0       0       0                  0
N290   0.0  0.0  0.0  3.0  NaN  NaN     0       1       0                 17
N292   0.0  0.0 -5.0  3.0  NaN  NaN     0       1       0                 43
N294   0.0  0.0 -5.0 -3.0  NaN  NaN     0       1       0                 16
N296   1.0  0.0 -5.0 -6.0  NaN  NaN  1000       1       0                 25
```

[31]:
```
#processing data in table to add features
from math import sqrt


#calculating travelled distance
distances = []
distances.append(0)
for i in range(1, len(td.index)):
    distances.append( sqrt(pow((td.X.iloc[i]-td.X.iloc[i-1]), 2) + pow((td.Y.
 ↪iloc[i]-td.Y.iloc[i-1]), 2))) #+ pow((td.Z.iloc[i]-td.Z.iloc[i-1]), 2)
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['traveled_distance'] = distances
except:
    pass

#create delta_z for differentiated calculations
delta_z = []
delta_z.append(0)
for i in range(1, len(td.index)):
    delta_z.append(td.Z.iloc[i] - td.Z.iloc[i-1])
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['delta_z'] = delta_z
except:
    pass

#calculating delta F
delta_F = []
delta_F.append(0)
for i in range(1, len(td.index)):
    delta_F.append( td.F.iloc[i] - td.F.iloc[i-1] )
try:
    #td.insert(9, 'delta_Z', delta_z)
    td['delta_F'] = delta_F
except:
    pass

td.head(7)
#td.tail()
```

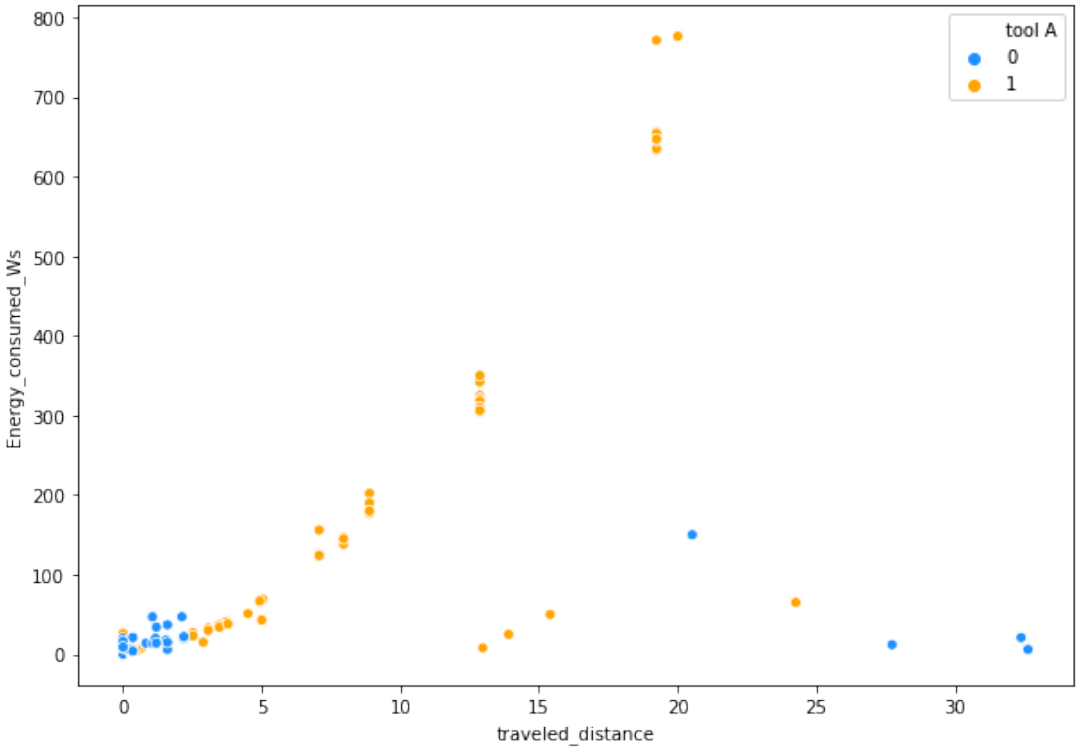
[31]:

index	G	X	Y	Z	I	J	F	tool A	tool B	\
NaN	NaN	0.000	0.0	0.0	0.0	0.0	0	0	0	
N290	0.0	0.000	0.0	3.0	NaN	NaN	0	1	0	
N292	0.0	0.000	-5.0	3.0	NaN	NaN	0	1	0	
N294	0.0	0.000	-5.0	-3.0	NaN	NaN	0	1	0	
N296	1.0	0.000	-5.0	-6.0	NaN	NaN	1000	1	0	
N298	1.0	-15.406	-5.0	-6.0	NaN	NaN	1000	1	0	
N300	3.0	-10.406	0.0	-6.0	0.0	5.0	1000	1	0	

index	Energy_consumed_Ws	traveled_distance	delta_z	delta_F
NaN	0	0.000000	0.0	0
N290	17	0.000000	3.0	0
N292	43	5.000000	0.0	0
N294	16	0.000000	-6.0	0
N296	25	0.000000	-3.0	1000
N298	50	15.406000	0.0	0
N300	156	7.071068	0.0	0

[32]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Examine the data by sight

#td.plot(figsize=(30,10))

plt.figure(figsize = (10, 7))
sns.scatterplot(x = td['traveled_distance'],
                y = td['Energy_consumed_Ws'],
                hue = td['tool A'],
                palette=['dodgerblue','orange'],)   #, 'green','red'],)
```

[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a20ceef90>
```



[33]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split


td_cleaned = td.iloc[1:,]
y = td_cleaned.Energy_consumed_Ws
td_features = ['G', 'traveled_distance', 'F', 'tool A', 'tool B'] #  'delta_Z',␣
 ↪'delta_F', if included results get worse. F is is feed speed and get's ramped␣
 ↪up every time anyway
X = td_cleaned[td_features]

train_X, val_X, train_y, val_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 2)

#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor


test_model = RandomForestRegressor(random_state = 1)
test_model.fit(train_X, train_y)
```

```
#make predictions and make a table to compare results to data

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow


#print(val_X)
val_predictions = test_model.predict(val_X)

comparison = val_X.copy()
comparison['meassured'] = val_y
comparison['predictions'] = val_predictions

deviation = []
for i in comparison.index:
    deviation.append( (comparison.predictions[i] - comparison.meassured[i]) /␣
 ↪comparison.meassured[i])
comparison['deviation'] = deviation


print(comparison)

mae = round(mean_absolute_error(val_y, val_predictions), 2)

sum_deviation = 0
for i in comparison.index:
    sum_deviation += abs(comparison.deviation[i])
mean_deviation = sum_deviation / len(comparison.index)
mre = round(mean_deviation * 100, 2)
mre2 = round(mae/comparison.meassured.mean()*100, 2)


print(f'\n---------------------------------- \n'
      f'mean absolut error: {mae} Ws. \n'
      f'mean of deviations: \u00B1 {mre}% \n'
      f'mae/mean_meassured: \u00B1 {mre2}%\n'
      f'----------------------------------\n')

#print(td.loc['N396']) #very bad prediciton
#print(td.loc['N294']) #very bad prediction
#print(td.loc['N392'])
```

```
         G  traveled_distance     F  tool A  tool B  meassured  predictions  \
index
N314   1.0          12.867360  1000       1       0        342   317.114564
N296   1.0           0.000000  1000       1       0         25    23.571254
```

N900	3.0	0.353553	750	0	1	21	11.356576
N302	1.0	2.524000	1000	1	0	27	21.862143
N898	1.0	0.200000	750	0	1	6	6.090000
N458	2.0	8.888127	1000	1	0	180	187.948167
N464	1.0	19.237000	1000	1	0	647	663.290671
N868	1.0	0.200000	750	0	1	6	6.090000
N348	2.0	3.732904	1000	1	0	39	40.659122
N902	1.0	1.600000	750	0	1	37	18.349371
N378	1.0	12.867360	1000	1	0	325	317.114564
N372	2.0	3.732904	1000	1	0	40	40.659122
N904	2.0	2.192031	750	0	1	22	21.162903
N818	0.0	27.734339	0	0	1	12	102.540000
N380	2.0	0.276921	1000	1	0	4	4.157650
N930	1.0	1.600000	750	0	1	6	18.349371
N300	3.0	7.071068	1000	1	0	156	126.736429

index	deviation
N314	-0.072764
N296	-0.057150
N900	-0.459211
N302	-0.190291
N898	0.015000
N458	0.044156
N464	0.025179
N868	0.015000
N348	0.042542
N902	-0.504071
N378	-0.024263
N372	0.016478
N904	-0.038050
N818	7.545000
N380	0.039412
N930	2.058229
N300	-0.187587

---

mean absolut error: 13.38 Ws.
mean of deviations: ± 66.67%
mae/mean_meassured: ± 12.0%

1.12 Detect Outliers

[34]:
```
td_outliers = td.drop(['I', 'J'], axis=1)
td_outliers.head()
```

[34]:
```
          G    X    Y    Z     F  tool A  tool B  Energy_consumed_Ws  \
index
NaN     NaN  0.0  0.0  0.0     0       0       0                   0
N290    0.0  0.0  0.0  3.0     0       1       0                  17
N292    0.0  0.0 -5.0  3.0     0       1       0                  43
N294    0.0  0.0 -5.0 -3.0     0       1       0                  16
N296    1.0  0.0 -5.0 -6.0  1000       1       0                  25

       traveled_distance  delta_z  delta_F
index
NaN                  0.0      0.0        0
N290                 0.0      3.0        0
N292                 5.0      0.0        0
N294                 0.0     -6.0        0
N296                 0.0     -3.0     1000
```

[35]:
```
#from sklearn.ensemble import IsolationForest
#import pandas as pd
#import seaborn as sns

# Predict and visualize outliers
#clf = IsolationForest(contamination=0.1, behaviour='new')
#td_outliers = td.drop(['I', 'J'], axis=1)
#outliers = clf.fit_predict(td_outliers)
#sns.scatterplot(td_outliers.traveled_distance, td_outliers.Energy_consumed_Ws,␣
 ↪outliers, palette='Set1', legend=False)

#plt.figure(figsize = (10, 7))
#sns.scatterplot(x = td_outliers['traveled_distance'],
#                y = td_outliers['Energy_consumed_Ws'],
#                hue = td_outliers['outliers'],
#                palette=['dodgerblue','orange', 'green','red'],)
```

Parsing2

October 22, 2020

1 Parser

[32]:
```
import pandas as pd
import numpy as np
```

[33]:
```
## Loading the raw data

df_filepath = './df2.csv'
saveAs = 'df2_parsed.csv'

df = pd.read_csv(df_filepath)
#df.head(20)
```

[34]:
```
## Formatting Data
## Delete unneeded columns & rename Columns

unneededColumns = ['ActiveTool', 'Unnamed: 0', 'HFProbeCounter', 'Channel',␣
 ↪'SeekOffset', 'SelectedTool', 'IpoGC', 'ipoReadError', 'laBuf',␣
 ↪'HFProbeCounterNext', 'POWER|4', 'POWER|6']
for column in unneededColumns:
    del df[str(column)]

df.rename(columns={'POWER|1':'ENERGY|x',
                   'POWER|2':'ENERGY|y',
                   'POWER|3':'ENERGY|z',
                   'POWER|5':'ENERGY|S',
                   'POWER|7':'ENERGY|T'},
          inplace=True)
#pd.set_option("display.max_rows", None, "display.max_columns", None)
#df
#df.head(10)
```

[35]:
```
## Seperate "N..." index

idents = ['N', 'G', 'M', 'M', 'G', 'G', 'G', 'X', 'Y', 'Z', 'I', 'J', 'F', 'S',␣
 ↪'D', 'M', 'T=', 'TURN', 'F']
index = 0
```

```
for ident in idents:
    array = []
    for i, message in enumerate(df['GCode']):
        text = str(message).strip()
        if text.startswith('SUPA'):
            split = text.split(' ', 1)
            df.loc[i,'GCode'] = split[1].strip()

        if text.startswith(ident) \
        and text[0:2] != 'IF' \
        and text[0:2] != 'ST' \
        and text[0:2] != 'SE' \
        and text[0:2] != 'SP' \
        and text[0:2] != 'FA' \
        and text[0:2] != 'GE' \
        and text[0:2] != 'ME' \
        and text[0:2] != 'MS':

            split = text.split(' ', 1)
            array.append(split[0])
            if len(split) > 1:
                df.loc[i,'GCode'] = split[1]
            else:
                df.loc[i,'GCode'] = ''

        else:
            array.append('')

    if ident in df.columns:
        for i, existing in enumerate(df[ident]):
            df.loc[i, ident] = existing + ' ' + array[i]
    else:
        df.insert(index, ident, array, True)
        index += 1


#pd.set_option("display.max_rows", None, "display.max_columns", None)

#         and text[0:2] != 'X=' \
#         and text[0:2] != 'Y=' \
#         and text[0:2] != 'Z=' \

df
#display(df)
#df.head(20)
```

[35]:

```
          N      G   M          X          Y          Z I  J   F S    D T=  \
0
1       N32   G0                                 Z=_Z_HOME              D0
2       N32   G0                                 Z=_Z_HOME              D0
3       N34             X=_X_HOME  Y=_Y_HOME
4       N34             X=_X_HOME  Y=_Y_HOME
…       …    …  ..         …          …          … .. .. .. ..  .. ..
2627  N1878             X=_X_HOME  Y=_Y_HOME
2628  N1878             X=_X_HOME  Y=_Y_HOME
2629  N1880
2630  N1880
2631
```

```
     TURN                                                GCode   ENERGY|x  \
0          MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")   0.000000
1                                                              0.000000
2                                                              0.000000
3                                                              0.000000
4                                                              0.000000
…   …                                                  …        …
2627                                                           0.000000
2628                                                          97.641451
2629                                  B=_B_HOME C=_C_HOME      0.000000
2630                                  B=_B_HOME C=_C_HOME      0.000000
2631                                              1772089      0.000000
```

```
      ENERGY|y  ENERGY|z  ENERGY|S  ENERGY|T
0     0.000000       0.0       0.0       0.0
1     0.000000       0.0       0.0       0.0
2     0.000000       0.0       0.0       0.0
3     0.000000       0.0       0.0       0.0
4     0.000000       0.0       0.0       0.0
…        …       …        …        …
2627  0.000000       0.0       0.0       0.0
2628  8.555121       0.0       0.0       0.0
2629  0.000000       0.0       0.0       0.0
2630  0.000000       0.0       0.0       0.0
2631  0.000000       0.0       0.0       0.0

[2632 rows x 19 columns]
```

[36]:
```
np.unique(df['TURN'])
```

[36]:
```
array([''], dtype=object)
```

[37]:
```
for i in range(len(df)):
    if str(df.loc[i, 'N']).strip() == 'N804':
```

```
        print(i)
        print(df.loc[i])
```

```
964
N                 N804
G                G2
M
X              X-5.998
Y              Y-7.482
Z
I               I4.097
J               J4.383
F            F=_F_CUT
S
D
T=
TURN
GCode
ENERGY|x      0.393357
ENERGY|y       1.80492
ENERGY|z             0
ENERGY|S       145.734
ENERGY|T             0
Name: 964, dtype: object
```

[38]:
```
## Singuläre Events in Nummerische Information umwandeln

ToolChange = []

for i in range(len(df)):
    if str(df.loc[i, 'T=']).strip() == '':
        ToolChange.append(0)
    else:
        ToolChange.append(1)

df.insert(12, 'Toolchange', ToolChange, True)
df.Toolchange

Turn = []

for i in range(len(df)):
    if str(df.loc[i, 'TURN']).strip() == '':
        Turn.append(0)
    else:
        Turn.append(1)

df.insert(12, 'TurnOp', Turn, True)
```

```
df.TurnOp
np.unique(df['TurnOp'])
```

[38]:
```
array([0])
```

[39]:
```
# Complete G-Codes

currentG = 'G0'

for i in range(len(df)) :
    if str(df.loc[i, "G"]).strip() != '':
        currentG = df.loc[i, "G"]
    elif str(df.loc[i, "X"]).strip() != '' \
    or str(df.loc[i, "Y"]).strip() != '' \
    or str(df.loc[i, "Z"]).strip() != '':
        df.at[i,'G'] = currentG

df.head()
```

[39]:

	N	G	M	X	Y	Z	I	J	F	S	…	T=	TurnOp	\
0											…		0	
1	N32	G0				Z=_Z_HOME					…		0	
2	N32	G0				Z=_Z_HOME					…		0	
3	N34	G0		X=_X_HOME	Y=_Y_HOME						…		0	
4	N34	G0		X=_X_HOME	Y=_Y_HOME						…		0	

	Toolchange	TURN		GCode	\
0	0		MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")		
1	0				
2	0				
3	0				
4	0				

	ENERGY|x	ENERGY|y	ENERGY|z	ENERGY|S	ENERGY|T
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

[5 rows x 21 columns]

[40]:
```
commands = []

for i in range(len(df)) :
    if str(df.loc[i, "G"]).strip() != '':
        if str(df.loc[i, "G"]).strip() == 'G3':
```

```
            df.loc[i, "G"] = 'G2'
        if str(df.loc[i, "M"]).strip() != '':
            commands.append(df.loc[i, "G"].strip() + ' ' + df.loc[i, "M"].
↪strip())
        else:
            commands.append(df.loc[i, "G"].strip())
    elif str(df.loc[i, 'M']).strip() != '':
        commands.append(df.loc[i, "M"].strip())
    else:
        commands.append('MSG')

df.insert(1, 'Commands', commands, True)

#df
```

[41]:
```
#df[df.isnull().any(axis=1)]
np.unique(df['Commands'])
```

[41]:
```
array(['G0', 'G0 G40 G60', 'G0 G90', 'G0 M106', 'G0 M3', 'G09', 'G1',
       'G1 G60', 'G2', 'G4', 'G40', 'G41 G1', 'G41 G94 G1 G90', 'G4F1',
       'G54 G0', 'G90', 'G91', 'G94', 'G94 G1 G90', 'M168', 'M169 M167',
       'M17', 'M27 M28', 'M5', 'M58;', 'M59', 'MSG'], dtype=object)
```

[42]:
```
np.unique(df['Commands'])
```

[42]:
```
array(['G0', 'G0 G40 G60', 'G0 G90', 'G0 M106', 'G0 M3', 'G09', 'G1',
       'G1 G60', 'G2', 'G4', 'G40', 'G41 G1', 'G41 G94 G1 G90', 'G4F1',
       'G54 G0', 'G90', 'G91', 'G94', 'G94 G1 G90', 'M168', 'M169 M167',
       'M17', 'M27 M28', 'M5', 'M58;', 'M59', 'MSG'], dtype=object)
```

[43]:
```
# Commands and Tool will be Category Type

categories = ['Commands', 'D']

for category in categories:
    df[category] = df[category].astype('category')
df.dtypes
```

[43]:
```
N              object
Commands     category
G              object
M              object
X              object
Y              object
Z              object
I              object
J              object
```

```
F               object
S               object
D             category
T=              object
TurnOp           int64
Toolchange       int64
TURN            object
GCode           object
ENERGY|x       float64
ENERGY|y       float64
ENERGY|z       float64
ENERGY|S       float64
ENERGY|T       float64
dtype: object
```

[44]:
```
# Check if worked

#df['Commands'] = df['Commands'].cat.codes
#df.head(20)
```

[45]:
```
columns = ['X', 'Y', 'Z', 'I', 'J', 'F', 'S'] # F Missing!!

for column in columns:
    for i, value in enumerate(df[column]):
        df.loc[i, column] = str(df.loc[i, column])[1:]

#df
df.head(16)
```

[45]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	\
0		MSG									…	
1	N32	G0	G0				=_Z_HOME				…	
2	N32	G0	G0				=_Z_HOME				…	
3	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
4	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
5		MSG									…	
6		MSG									…	
7		MSG									…	
8		MSG									…	
9	N021	MSG									…	
10	N390	MSG									…	
11	N394	MSG									…	
12	N396	MSG									…	
13	N396	MSG									…	
14	N464	MSG									…	
15	N480	MSG									…	

	T=	TurnOp	Toolchange	TURN	\
0		0	0		
1		0	0		
2		0	0		
3		0	0		
4		0	0		
5	T="3D_TASTER"	0	1		
6		0	0		
7		0	0		
8		0	0		
9		0	0		
10		0	0		
11		0	0		
12		0	0		
13		0	0		
14		0	0		
15		0	0		

		GCode	ENERGY|x	ENERGY|y	\
0	MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")		0.0	0.0	
1			0.0	0.0	
2			0.0	0.0	
3			0.0	0.0	
4			0.0	0.0	
5			0.0	0.0	
6		STOPRE	0.0	0.0	
7		STOPRE	0.0	0.0	
8		STOPRE	0.0	0.0	
9	_AUTOGEAR_2=($P_SAUTOGEAR[1] AND ($AC_SGEAR[1]…		0.0	0.0	
10	IF NOT M_ENABLE_TOOLCHANGE AND NOT _SIM AND NO…		0.0	0.0	
11		STOPRE	0.0	0.0	
12	_MODE=STEP_OLD FAST=0 _TEST=_SIM OR $P_SEARCH …		0.0	0.0	
13	_MODE=STEP_OLD FAST=0 _TEST=_SIM OR $P_SEARCH …		0.0	0.0	
14	IF($AA_IM[_AX4]<$MN_USER_DATA_INT[83]-0.5)…		0.0	0.0	
15		GETSELT(T_PR)	0.0	0.0	

	ENERGY|z	ENERGY|S	ENERGY|T
0	0.0	0.0	0.000000
1	0.0	0.0	0.000000
2	0.0	0.0	0.000000
3	0.0	0.0	0.000000
4	0.0	0.0	0.000000
5	0.0	0.0	0.000000
6	0.0	0.0	0.000000
7	0.0	0.0	0.000000
8	0.0	0.0	0.000000
9	0.0	0.0	0.000000

```
10       0.0       0.0  0.054302
11       0.0       0.0  0.238047
12       0.0       0.0  0.232837
13       0.0       0.0  0.115216
14       0.0       0.0  0.591310
15       0.0       0.0  0.161503

[16 rows x 22 columns]
```

[46]:
```
tool = []
bearbeitung = []

for i, message in enumerate(df['GCode']):
    text = str(message).strip()
    if text.startswith('MSG('):
        split = text.split('"')
        split2 = split[1].strip().split()
#         print(split2)
        bearbeitung.append(split2[0].strip())
#         df.loc[i,'GCode'] = split2[2] + split2[3] + ' ' + split2[4]
        tool.append(split2[4])
#     elif text.startswith('T='):
#         split = text.split('"')
#         tool.append(split[1]) ### NOPE!
#         bearbeitung.append('')
    else:
        tool.append('')
        bearbeitung.append('')

df.insert(10, 'Werkzeug', tool, True)
df.insert(11, 'Bearbeitung', bearbeitung, True)

#df
```

[47]:
```
np.unique(df['Werkzeug'])
np.unique(df['Bearbeitung'])
```

[47]:
```
array(['', 'BOSS_MILLING', 'BOSS_MILLING_COPY', 'FASENFRAESEN_1',
       'FASENFRAESEN_2', 'FASENFRAESEN_2_COPY', 'FLOOR_WALL',
       'FLOOR_WALL_1', 'FLOOR_WALL_1_COPY', 'FLOOR_WALL_1_COPY_COPY',
       'FLOOR_WALL_COPY', 'FLOOR_WALL_COPY_1', 'FLOOR_WALL_IPW',
       'FLOOR_WALL_IPW_COPY', 'FLOOR_WALL_IPW_COPY_COPY',
       'FLOOR_WALL_IPW_COPY_COPY_COPY', 'RAEUMEN', 'VORBOHREN_4.7',
       'ZENTRIERBOHREN'], dtype=object)
```

[48]:
```
toolList = {
  "T_A_SK40_DM16_A" : {
```

```
    "name" : "End Mill DM16",
    "DM" : 16,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL_IPW" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
  },
  "T_A_SK40_SF_DM16_A" : {
    "name" : "End Mill DM16",
    "DM" : 16,
    "FLOOR_WALL_IPW" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL_COPY" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
  },
  "001691_A" : {
    "name" : "End Mill DM10",
    "DM" : 10,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "PLANEN" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
    "SCHRUPPEN" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL_COPY_1" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
    "FLOOR_WALL_1" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550},
    "FLOOR_WALL_IPW_COPY" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
  },
  "001688_A" : {
    "name" : "End Mill DM6",
    "DM" : 6,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL_IPW_COPY_COPY" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' :␣
↪1000},
    "FLOOR_WALL_IPW_COPY_COPY_COPY" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET'␣
↪: None},
  },
  "002384_A" : {
    "name" : "End Mill DM3",
    "DM" : 3,
    "STANDARD" : {'F_ENG' :  100, 'F_CUT' : None, 'F_RET' : 0},
    "BOHRFRAESEN_SCHRUPPEN" : {'F_ENG' :  24, 'F_CUT' : 680, 'F_RET' : 680},
    "KREUZ_SCHRUPPEN" : {'F_ENG' :  750, 'F_CUT' : 750, 'F_RET' : 750},
    "KREUZ_SCHLICHTEN" : {'F_ENG' :  500, 'F_CUT' : 500, 'F_RET' : 500},
    "RESTFRAESEN_SCHRUPPEN" : {'F_ENG' :  750, 'F_CUT' : 750, 'F_RET' : 750},
    "SCHLICHTEN_AUSSEN" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550},
    "SCHLICHTEN_INNEN" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FLOOR_WALL_1_COPY" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550},
    "FLOOR_WALL_1_COPY_COPY" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' :␣
↪None},
  },
  "T_A_SK40_SF_DM3_SL_A" : {
```

```
    "name" : "End Mill DM3",
    "DM" : 3,
    "STANDARD" : {'F_ENG' :  550, 'F_CUT' : None, 'F_RET' : 550}
  },
  "002383_A" : {
    "name" : "Center Drill",
    "DM" : 8,
    "STANDARD" : {'F_ENG' :  1000, 'F_CUT' : 1000, 'F_RET' : 1000},
    "ZENTRIERBOHREN" : {'F_ENG' :  500, 'F_CUT' : None, 'F_RET' : 500},
    "SCHRUPPEN" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},          #␣
↪Werte???!!
  },
  "002293_A" : {
    "name" : "Twist Drill",
    "DM" : 4.7,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "VORBOHREN_4.7" : {'F_ENG' :  500, 'F_CUT' : None, 'F_RET' : 0}
  },
  "002380_A" : {
    "name" : "Chamfer Mill DM8 90Grad",
    "DM" : 8,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FASENFRAESEN_1" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550},
    "FASENFRAESEN_2" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550}
  },
  "002387_A" : {
    "name" : "Chamfer Mill DM10 60Grad",
    "DM" : 10,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FASENFRAESEN_1" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550},

  },
  "002397_A" : {
    "name" : "Ball Mill DM4",
    "DM" : 4,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "FASENFRAESEN_2_COPY" : {'F_ENG' :  550, 'F_CUT' : 550, 'F_RET' : 550}
  },
  "001728_A" : {
    "name" : "Drill DM2,8",
    "DM" : 2.8,
    "VORBOHREN_2.8" : {'F_ENG' :  100, 'F_CUT' : None, 'F_RET' : 0},
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None}
  },
  "002411_A" : {
    "name" : "Reibahle DM5",
    "DM" : 5,
```

```
    "RAEUMEN" : {'F_ENG' :  100, 'F_CUT' : None, 'F_RET' : 0},
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None}
  },
  "T_A_SK40_SF_DM3_SL_A" : {
    "name" : "End Mill DM3",
    "DM" : 3,
    "STANDARD" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
    "BOSS_MILLING_COPY" : {'F_ENG' :  220, 'F_CUT' : 550, 'F_RET' : 550},
    "BOSS_MILLING" : {'F_ENG' :  None, 'F_CUT' : None, 'F_RET' : None},
  },
}

print(toolList['002384_A']['STANDARD']['F_ENG'])
```

```
100
```

[49]:
```
# Insert Spindel Speeds

currentS = 0

for i in range(len(df)) :
    if not pd.isnull(df.loc[i, "S"]):
        currentS = df.loc[i, "S"]
    elif str(df.loc[i, "Commands"]).strip() == 'M5':
        currentS = 0
        df.at[i,'S'] = currentS
    else:
        df.at[i,'S'] = currentS
```

[50]:
```
#Werkzeugdurchmesser aufnehmen

D_W = []
unbekannteWerkzeuge = []

for i in range(len(df)):
    if str(df.loc[i, 'Werkzeug']).strip() != '':
        try:
            D_W.append(toolList[df.loc[i, 'Werkzeug'].strip()]['DM'])
        except:
            D_W.append(0)
            unbekannteWerkzeuge.append(df.loc[i, 'Werkzeug'].strip())
            print()
            print(i)
            print(df.loc[i])
    else:
        D_W.append('')
```

```
df.insert(10, 'D_W', D_W, True)

#weiter schreiben
current = 0
for i in range(len(df)) :
    if str(df.loc[i, 'D_W']).strip() != '':
        current = df.loc[i, 'D_W']
    else:
        df.at[i,'D_W'] = current

df
```

[50]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	\
0		MSG									…	
1	N32	G0	G0				=_Z_HOME				…	
2	N32	G0	G0				=_Z_HOME				…	
3	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
4	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
…	…	…	…	..	…	…	…	..	..	..	…	
2627	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2628	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2629	N1880	MSG									…	
2630	N1880	MSG									…	
2631		MSG									…	

	T=	TurnOp	Toolchange	TURN	\
0		0	0		
1		0	0		
2		0	0		
3		0	0		
4		0	0		
…	..	…	…	…	
2627		0	0		
2628		0	0		
2629		0	0		
2630		0	0		
2631		0	0		

	GCode	ENERGY|x	ENERGY|y	\
0	MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")	0.000000	0.000000	
1		0.000000	0.000000	
2		0.000000	0.000000	
3		0.000000	0.000000	
4		0.000000	0.000000	
…	…	…	…	
2627		0.000000	0.000000	
2628		97.641451	8.555121	

```
2629                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2630                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2631                                            1772089   0.000000  0.000000

      ENERGY|z ENERGY|S ENERGY|T
0          0.0      0.0      0.0
1          0.0      0.0      0.0
2          0.0      0.0      0.0
3          0.0      0.0      0.0
4          0.0      0.0      0.0
…          …        …        …
2627       0.0      0.0      0.0
2628       0.0      0.0      0.0
2629       0.0      0.0      0.0
2630       0.0      0.0      0.0
2631       0.0      0.0      0.0

[2632 rows x 25 columns]
```

[51]:
```
unbekannteWerkzeuge
```

[51]:
```
[]
```

[52]:
```
#  Werkzeug weiterschreiben
current = ''
for i in range(len(df)) :
    if str(df.loc[i, 'Werkzeug']).strip() != '':
        current = df.loc[i, 'Werkzeug']
    else:
        df.at[i,'Werkzeug'] = current

#Bearbeitung weiterschreiben mit Löschung bei Werkzeugwechsel!
current = 'STANDARD'
currentTool = ''
for i in range(len(df)) :
    if str(df.loc[i, 'Werkzeug']).strip() != currentTool:
        current = 'STANDARD'
        if str(df.loc[i, 'Bearbeitung']).strip() == '':
            df.at[i,'Bearbeitung'] = current
    elif str(df.loc[i, 'Bearbeitung']).strip() != '':
        current = df.loc[i, 'Bearbeitung']
    else:
        df.at[i,'Bearbeitung'] = current
    currentTool = str(df.loc[i, 'Werkzeug']).strip()

df
```

[52]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	\
0		MSG									…	
1	N32	G0	G0				=_Z_HOME				…	
2	N32	G0	G0				=_Z_HOME				…	
3	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
4	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
…	…	…	…	..	…	…	…	…				
2627	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2628	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2629	N1880	MSG									…	
2630	N1880	MSG									…	
2631		MSG									…	

	T=	TurnOp	Toolchange	TURN	\
0		0	0		
1		0	0		
2		0	0		
3		0	0		
4		0	0		
…	..	…	…	…	
2627		0	0		
2628		0	0		
2629		0	0		
2630		0	0		
2631		0	0		

	GCode	ENERGY|x	ENERGY|y	\
0	MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")	0.000000	0.000000	
1		0.000000	0.000000	
2		0.000000	0.000000	
3		0.000000	0.000000	
4		0.000000	0.000000	
…	…	…	…	
2627		0.000000	0.000000	
2628		97.641451	8.555121	
2629	B=_B_HOME C=_C_HOME	0.000000	0.000000	
2630	B=_B_HOME C=_C_HOME	0.000000	0.000000	
2631	1772089	0.000000	0.000000	

	ENERGY|z	ENERGY|S	ENERGY|T
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
…	…	…	…
2627	0.0	0.0	0.0

```
2628       0.0       0.0       0.0
2629       0.0       0.0       0.0
2630       0.0       0.0       0.0
2631       0.0       0.0       0.0

[2632 rows x 25 columns]
```

[53]:
```
for i, message in enumerate(df['F']):
    #text = str(message).strip()
    if message.startswith('=_DTB'):
        df.at[i,'F'] = 0
    elif message.startswith('=_'):
        df.at[i,'F'] = message[2:]
#         print(df.at[i,'F'])
```

[54]:
```
F_val = []
F = ''

for i in range(len(df)) :
    if str(df.loc[i, 'F']).strip() != '' \
    and not any(map(str.isdigit, str(df.loc[i, "F"]))) \
    and df.loc[i, 'F'] != None:
        try:
            F_val.append(toolList[df.loc[i, 'Werkzeug'].strip()][df.loc[i,␣
 ↪'Bearbeitung'].strip()][df.loc[i, 'F'].strip()])
        except:
            F_val.append(None)
            print(i)
            print(df.loc[i])
    elif any(map(str.isdigit, str(df.loc[i, "F"]))):
        F_val.append(df.loc[i, "F"])
    else:
        F_val.append('')

df.insert(10, 'F_val', F_val, True)

df
```

```
1830
N                     N2
Commands              G1
G                   G1
M
X
Y
Z
I
```

```
J
F                  =FFR
D_W                   5
Werkzeug         002411_A
Bearbeitung      STANDARD
S
D
T=
TurnOp                0
Toolchange            0
TURN
GCode           AX[_Z]=DP
ENERGY|x              0
ENERGY|y              0
ENERGY|z        -15.5594
ENERGY|S         247.489
ENERGY|T         2.16026
Name: 1830, dtype: object
1831
N                    N3
Commands             G4
G                  G4
M
X
Y
Z
I
J
F                  =DTB
D_W                   5
Werkzeug         002411_A
Bearbeitung      STANDARD
S
D
T=
TurnOp                0
Toolchange            0
TURN
GCode
ENERGY|x              0
ENERGY|y              0
ENERGY|z              0
ENERGY|S       0.986936
ENERGY|T              0
Name: 1831, dtype: object
1832
N                                          N4
Commands                                  MSG
```

```
G
M
X
Y
Z
I
J
F                                                =RFF
D_W                                                 5
Werkzeug                                       002411_A
Bearbeitung                                    STANDARD
S
D
T=
TurnOp                                              0
Toolchange                                          0
TURN
GCode          AX[_Z]=RFP+SDIS*(RFP-DP)/ABS(RFP-DP)
ENERGY|x                                            0
ENERGY|y                                            0
ENERGY|z                                      24.0161
ENERGY|S                                      232.252
ENERGY|T                                            0
Name: 1832, dtype: object
```

[54]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	\
0		MSG									…	
1	N32	G0	G0				=_Z_HOME				…	
2	N32	G0	G0				=_Z_HOME				…	
3	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
4	N34	G0	G0		=_X_HOME	=_Y_HOME					…	
…	…	…	…	..	…	…	…	..	..	..	…	
2627	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2628	N1878	G4	G4		=_X_HOME	=_Y_HOME					…	
2629	N1880	MSG									…	
2630	N1880	MSG									…	
2631		MSG									…	

	T=	TurnOp	Toolchange	TURN	\
0		0	0		
1		0	0		
2		0	0		
3		0	0		
4		0	0		
…	..	…	…	…	
2627		0	0		
2628		0	0		

```
2629         0          0
2630         0          0
2631         0          0

                                                    GCode   ENERGY|x  ENERGY|y  \
0     MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")   0.000000  0.000000
1                                                        0.000000  0.000000
2                                                        0.000000  0.000000
3                                                        0.000000  0.000000
4                                                        0.000000  0.000000
…                                                   …        …       …
2627                                                     0.000000  0.000000
2628                                                    97.641451  8.555121
2629                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2630                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2631                                             1772089   0.000000  0.000000

      ENERGY|z  ENERGY|S ENERGY|T
0          0.0       0.0      0.0
1          0.0       0.0      0.0
2          0.0       0.0      0.0
3          0.0       0.0      0.0
4          0.0       0.0      0.0
…        …       …       …
2627       0.0       0.0      0.0
2628       0.0       0.0      0.0
2629       0.0       0.0      0.0
2630       0.0       0.0      0.0
2631       0.0       0.0      0.0

[2632 rows x 26 columns]
```

[55]:
```
#  F_val weiterschreiben
current = 0
for i in range(len(df)) :
    if str(df.loc[i, 'F_val']).strip() != '' \
    and df.loc[i, 'F_val'] != None:
        current = df.loc[i, 'F_val']
    else:
        df.at[i,'F_val'] = current

df
```

[55]:
```
          N Commands      G   M         X         Y         Z I  J  F   …  \
0                MSG                                                  …
1       N32       G0   G0                          =_Z_HOME            …
2       N32       G0   G0                          =_Z_HOME            …
```

```
3       N34       G0   G0        =_X_HOME  =_Y_HOME                    …
4       N34       G0   G0        =_X_HOME  =_Y_HOME                    …
…     …       …    …  ..       …       …       … .. .. ..  …
2627  N1878       G4  G4         =_X_HOME  =_Y_HOME                    …
2628  N1878       G4  G4         =_X_HOME  =_Y_HOME                    …
2629  N1880      MSG                                                    …
2630  N1880      MSG                                                    …
2631             MSG                                                    …

     T= TurnOp Toolchange TURN  \
0            0          0
1            0          0
2            0          0
3            0          0
4            0          0
…  ..    …        …  …
2627         0          0
2628         0          0
2629         0          0
2630         0          0
2631         0          0

                                                      GCode   ENERGY|x  ENERGY|y  \
0     MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")   0.000000  0.000000
1                                                        0.000000  0.000000
2                                                        0.000000  0.000000
3                                                        0.000000  0.000000
4                                                        0.000000  0.000000
…                                                      …        …       …
2627                                                     0.000000  0.000000
2628                                                    97.641451  8.555121
2629                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2630                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2631                                            1772089   0.000000  0.000000

      ENERGY|z  ENERGY|S ENERGY|T
0          0.0       0.0      0.0
1          0.0       0.0      0.0
2          0.0       0.0      0.0
3          0.0       0.0      0.0
4          0.0       0.0      0.0
…        …       …      …
2627       0.0       0.0      0.0
2628       0.0       0.0      0.0
2629       0.0       0.0      0.0
2630       0.0       0.0      0.0
2631       0.0       0.0      0.0
```

```
[2632 rows x 26 columns]
```

[56]:
```
#Checkpoint

#df2 = df.copy()
#df = df2.copy()
```

[57]:
```
df
```

[57]:
```
          N Commands      G   M         X         Y         Z I  J  F   …  \
0                MSG                                                    …
1       N32       G0   G0                             =_Z_HOME           …
2       N32       G0   G0                             =_Z_HOME           …
3       N34       G0   G0        =_X_HOME  =_Y_HOME                      …
4       N34       G0   G0        =_X_HOME  =_Y_HOME                      …
…     …      …    …   ..       …        …       … .. .. ..  …
2627  N1878       G4  G4         =_X_HOME  =_Y_HOME                      …
2628  N1878       G4  G4         =_X_HOME  =_Y_HOME                      …
2629  N1880      MSG                                                     …
2630  N1880      MSG                                                     …
2631               MSG                                                    …

     T= TurnOp Toolchange TURN  \
0            0          0
1            0          0
2            0          0
3            0          0
4            0          0
…   ..     …        …   …
2627         0          0
2628         0          0
2629         0          0
2630         0          0
2631         0          0

                                                    GCode   ENERGY|x  ENERGY|y  \
0     MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")   0.000000  0.000000
1                                                        0.000000  0.000000
2                                                        0.000000  0.000000
3                                                        0.000000  0.000000
4                                                        0.000000  0.000000
…                                                   …        …       …
2627                                                     0.000000  0.000000
2628                                                    97.641451  8.555121
2629                                B=_B_HOME C=_C_HOME   0.000000  0.000000
2630                                B=_B_HOME C=_C_HOME   0.000000  0.000000
```

```
2631                                                1772089    0.000000   0.000000
```

```
      ENERGY|z  ENERGY|S ENERGY|T
0          0.0       0.0      0.0
1          0.0       0.0      0.0
2          0.0       0.0      0.0
3          0.0       0.0      0.0
4          0.0       0.0      0.0
…          …         …        …
2627       0.0       0.0      0.0
2628       0.0       0.0      0.0
2629       0.0       0.0      0.0
2630       0.0       0.0      0.0
2631       0.0       0.0      0.0

[2632 rows x 26 columns]
```

[58]:
```
# X, Y, Z weiterschreiben
coordinates = ['X', 'Y']

for coord in coordinates:
    current = 0
    for i in range(len(df)) :
        if str(df.loc[i, coord]).strip() != '' \
        and str(df.loc[i, coord]).strip()[0] != '=' :
            current = df.loc[i, coord]
        elif str(df.loc[i, coord]).strip() != '' \
        and str(df.loc[i, coord]).strip()[0] == '=' :
            current = 0
            df.at[i, coord] = current
        else:
            df.at[i, coord] = current

# Z weiterschreiben
coordinates = ['Z']

for coord in coordinates:
    current = 610
    for i in range(len(df)) :
        if str(df.loc[i, coord]).strip() != '' \
        and str(df.loc[i, coord]).strip()[0] != '=' :
            current = df.loc[i, coord]
        elif str(df.loc[i, coord]).strip() != '' \
        and str(df.loc[i, coord]).strip()[0] == '=' :
            current = 610
            df.at[i, coord] = current
        else:
```

```
            df.at[i, coord] = current

# I, J mit "0" auffüllen

coordinates = ['I', 'J']

for coord in coordinates:
    for i in range(len(df)) :
        if str(df.loc[i, coord]).strip() == '' :
            df.at[i, coord] = 0
```

[59]:
```
# Spindel bei M5 stoppen

for i, message in enumerate(df['S']):
    if str(df.loc[i, 'M']).strip().startswith('M5'):
        df.at[i,'S'] = 0

# S weiterschreiben
coordinates = ['S']

for coord in coordinates:
    current = 0
    for i in range(len(df)) :
        if str(df.loc[i, coord]).strip() != '':
            current = df.loc[i, coord]
        else:
            df.at[i, coord] = current

df
```

[59]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	T=	TurnOp	Toolchange	\
0		MSG			0	0	610	0	0		…		0	0	
1	N32	G0	G0		0	0	610	0	0		…		0	0	
2	N32	G0	G0		0	0	610	0	0		…		0	0	
3	N34	G0	G0		0	0	610	0	0		…		0	0	
4	N34	G0	G0		0	0	610	0	0		…		0	0	
…	…	…	…	…	…	…	…	…	…	…	…	…	…		
2627	N1878	G4	G4		0	0	610	0	0		…		0	0	
2628	N1878	G4	G4		0	0	610	0	0		…		0	0	
2629	N1880	MSG			0	0	610	0	0		…		0	0	
2630	N1880	MSG			0	0	610	0	0		…		0	0	
2631		MSG			0	0	610	0	0		…		0	0	

	TURN	GCode	ENERGY|x	\
0	MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")		0.000000	
1			0.000000	
2			0.000000	

```
3                                                          0.000000
4                                                          0.000000
…   …                                                  …         …
2627                                                       0.000000
2628                                                      97.641451
2629                                  B=_B_HOME C=_C_HOME   0.000000
2630                                  B=_B_HOME C=_C_HOME   0.000000
2631                                               1772089  0.000000

      ENERGY|y  ENERGY|z  ENERGY|S ENERGY|T
0     0.000000       0.0       0.0      0.0
1     0.000000       0.0       0.0      0.0
2     0.000000       0.0       0.0      0.0
3     0.000000       0.0       0.0      0.0
4     0.000000       0.0       0.0      0.0
…        …       …        …       …
2627  0.000000       0.0       0.0      0.0
2628  8.555121       0.0       0.0      0.0
2629  0.000000       0.0       0.0      0.0
2630  0.000000       0.0       0.0      0.0
2631  0.000000       0.0       0.0      0.0

[2632 rows x 26 columns]
```

[60]:
```
# Change object data type to float

categories = ['X', 'Y', 'Z', 'I', 'J', 'S', 'F_val', 'D_W']

for category in categories:
    df[category] = pd.to_numeric(df[category],errors='coerce')
df.dtypes
```

[60]:
```
N                object
Commands       category
G                object
M                object
X               float64
Y               float64
Z               float64
I               float64
J               float64
F                object
F_val           float64
D_W             float64
Werkzeug         object
Bearbeitung      object
S                 int64
```

```
D               category
T=                object
TurnOp             int64
Toolchange         int64
TURN              object
GCode             object
ENERGY|x         float64
ENERGY|y         float64
ENERGY|z         float64
ENERGY|S         float64
ENERGY|T         float64
dtype: object
```

[61]:
```
#processing data in table to add features
from math import sqrt

#calculating travelled distance in xy-Plane
distances = []
distances.append(0)
for i in range(1, len(df.index)):
    distances.append( sqrt(pow((df.X.iloc[i]-df.X.iloc[i-1]), 2) + pow((df.Y.
 ↪iloc[i]-df.Y.iloc[i-1]), 2))) #+ pow((df.Z.iloc[i]-df.Z.iloc[i-1]), 2)
df.insert(10, 'delta_xy', distances, True)


#calculating Deltas
deltas = ['S', 'Z', 'Y', 'X']
for delta in deltas:
    listD = []
    listD.append(0)
    for i in range(1, len(df.index)):
        j = i - 1
        listD.append(df.loc[i, delta] - df.loc[j, delta])
    df.insert(11, ('delta_' + str(delta)), listD, True)

#Betrag bei X und Y delta
deltas = ['delta_Y', 'delta_X']
for delta in deltas:
    for i in range(len(df)):
        df.loc[i, delta] = abs(df.loc[i, delta])


df.head(7)
#df.tail()
```

[61]:
```
     N Commands      G   M    X    Y      Z    I    J F  …            T=  \
0           MSG            0.0  0.0  610.0  0.0  0.0    …
```

```
1  N32       G0   G0        0.0  0.0  610.0  0.0  0.0    …
2  N32       G0   G0        0.0  0.0  610.0  0.0  0.0    …
3  N34       G0   G0        0.0  0.0  610.0  0.0  0.0    …
4  N34       G0   G0        0.0  0.0  610.0  0.0  0.0    …
5           MSG             0.0  0.0  610.0  0.0  0.0    …  T="3D_TASTER"
6           MSG             0.0  0.0  610.0  0.0  0.0    …

   TurnOp  Toolchange  TURN  \
0       0           0
1       0           0
2       0           0
3       0           0
4       0           0
5       0           1
6       0           0

                                                 GCode  ENERGY|x  ENERGY|y  \
0  MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")       0.0       0.0
1                                                          0.0       0.0
2                                                          0.0       0.0
3                                                          0.0       0.0
4                                                          0.0       0.0
5                                                          0.0       0.0
6                                               STOPRE     0.0       0.0

  ENERGY|z ENERGY|S  ENERGY|T
0      0.0      0.0       0.0
1      0.0      0.0       0.0
2      0.0      0.0       0.0
3      0.0      0.0       0.0
4      0.0      0.0       0.0
5      0.0      0.0       0.0
6      0.0      0.0       0.0

[7 rows x 31 columns]
```

[62]:
```
df.to_csv(saveAs)
```

Hauptprogramm

October 22, 2020

1 Machine-Learning Programm

1.1 (Train & Test = gleiche Daten)

1.2 Importing gcode in csv-file

[1]:
```
#import data from csv file for further processing
import pandas as pd
import numpy as np

df_filepath = './df_parsed.csv'
df = pd.read_csv(df_filepath, index_col=['Unnamed: 0'])


#pd.set_option("display.max_rows", None, "display.max_columns", None)
df.describe()
#df.head()
#df.tail()
#df.columns
```

[1]:

	X	Y	Z	I	J
count	2213.000000	2213.000000	2213.000000	2213.000000	2213.000000
mean	-1.300142	1.552118	285.452850	-0.008438	0.016631
std	6.403329	21.329063	303.116148	2.433507	0.865189
min	-19.907000	-70.500000	-8.000000	-14.300000	-7.300000
25%	-0.100000	-2.262000	-3.200000	0.000000	0.000000
50%	0.000000	0.000000	50.000000	0.000000	0.000000
75%	0.000000	2.262000	610.000000	0.000000	0.000000
max	14.300000	67.500000	610.000000	14.300000	7.300000

	delta_xy	delta_X	delta_Y	delta_Z	delta_S
count	2213.000000	2213.000000	2213.000000	2.213000e+03	2213.000000
mean	2.128933	0.553685	1.810044	2.054891e-16	0.000000
std	8.751811	1.789431	8.622186	5.131558e+01	694.202289
min	0.000000	0.000000	0.000000	-6.070000e+02	-13000.000000
25%	0.000000	0.000000	0.000000	0.000000e+00	0.000000
50%	0.000000	0.000000	0.000000	0.000000e+00	0.000000

```
75%       0.632001     0.000000     0.085000  0.000000e+00      0.000000
max     134.100000    25.906000   134.100000  6.070000e+02  13000.000000

              F_val          D_W             S       TurnOp   Toolchange  \
count   2213.000000  2213.000000   2213.000000  2213.000000  2213.000000
mean     120.208766     4.382738   4584.274740     0.001808     0.004067
std      264.405105     3.416697   5465.059687     0.042486     0.063657
min        0.000000     0.000000      0.000000     0.000000     0.000000
25%        0.000000     2.800000      0.000000     0.000000     0.000000
50%        0.000000     3.000000      0.000000     0.000000     0.000000
75%        0.000000     8.000000  10000.000000     0.000000     0.000000
max      750.000000    10.000000  13000.000000     1.000000     1.000000

          ENERGY|x     ENERGY|y     ENERGY|z      ENERGY|S     ENERGY|T
count   2213.000000  2213.000000  2213.000000   2213.000000  2213.000000
mean       4.129043     1.478090     2.903118    118.339919     1.492249
std       46.044159     8.667296    52.573514    857.416625    13.030186
min        0.000000     0.000000  -129.385580  -8043.649349     0.000000
25%        0.000000     0.000000     0.000000      0.000000     0.000000
50%        0.000000     0.000000     0.000000      0.000000     0.000000
75%        0.000000     0.083114     0.000000    101.593985     0.000000
max      570.195688   134.946734   940.387548  14158.866014   281.044521
```

1.3 One-Hot categorical encoding

[2]:
```
dfCompare = df.copy()
df = pd.get_dummies(df, columns=["Commands", 'D'])
list(df.columns.values)
```

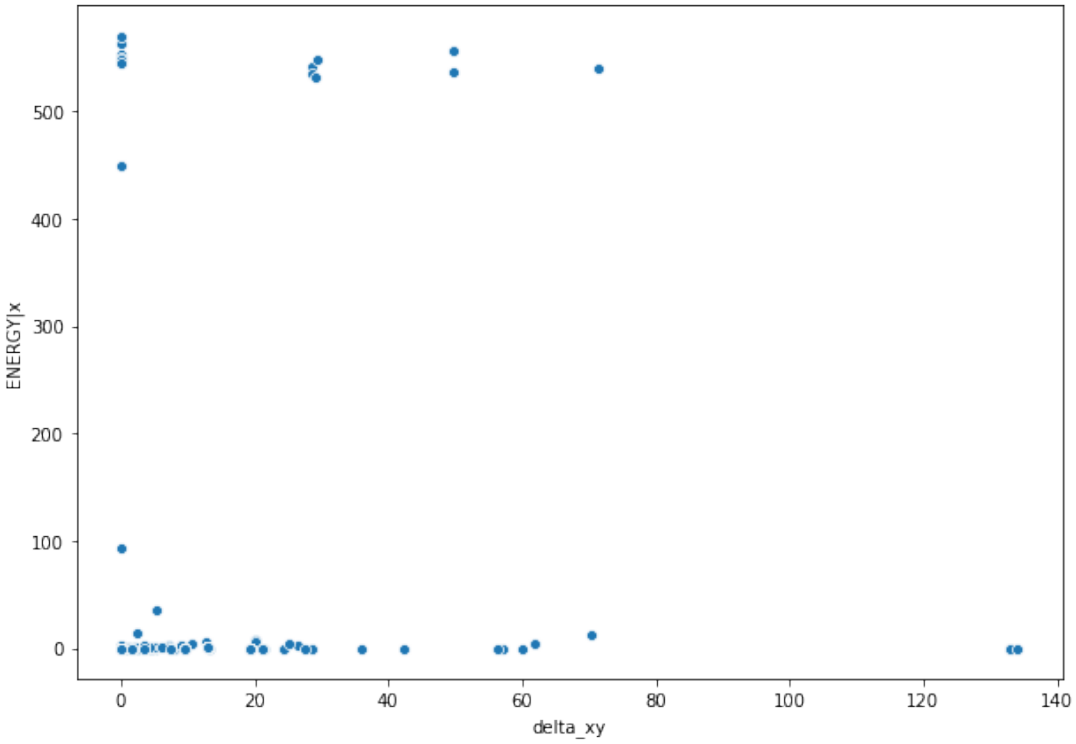
[2]:
```
['N',
 'G',
 'M',
 'X',
 'Y',
 'Z',
 'I',
 'J',
 'F',
 'delta_xy',
 'delta_X',
 'delta_Y',
 'delta_Z',
 'delta_S',
 'F_val',
 'D_W',
 'Werkzeug',
```

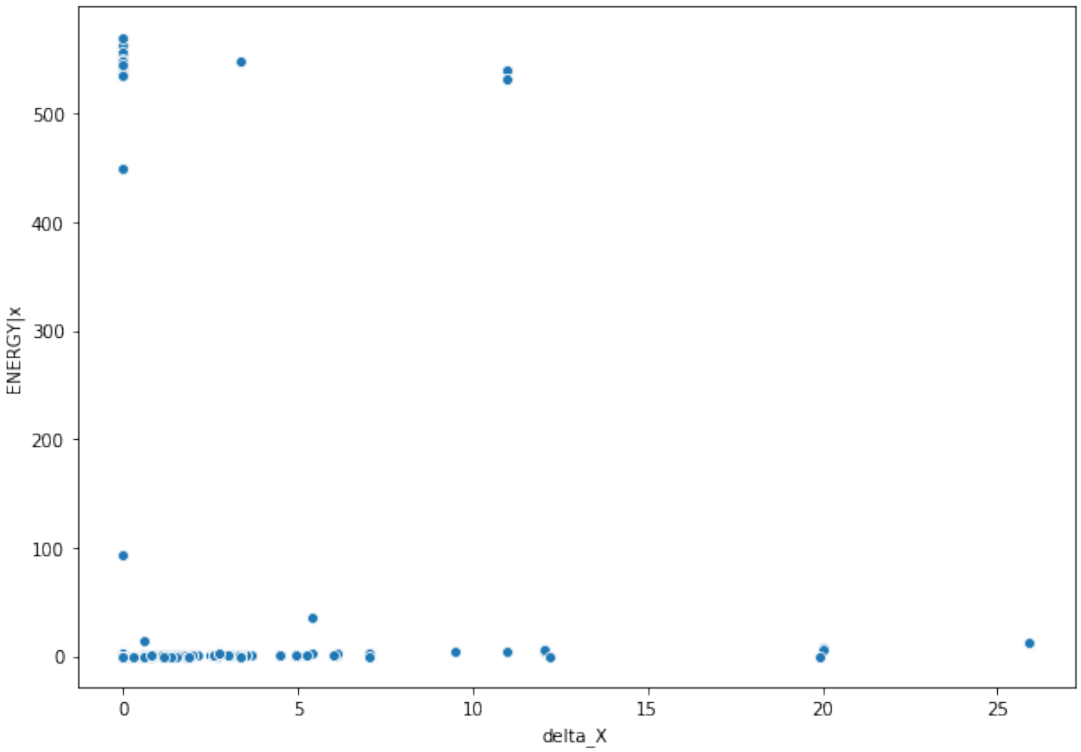
```
'Bearbeitung',
'S',
'T=',
'TurnOp',
'Toolchange',
'TURN',
'GCode',
'ENERGY|x',
'ENERGY|y',
'ENERGY|z',
'ENERGY|S',
'ENERGY|T',
'Commands_G0',
'Commands_G0 G40 G60',
'Commands_G0 M106',
'Commands_G0 M3',
'Commands_G09',
'Commands_G1',
'Commands_G1 G60',
'Commands_G2',
'Commands_G4',
'Commands_G40',
'Commands_G41',
'Commands_G41 G1',
'Commands_G4F1',
'Commands_G54 G0',
'Commands_G90',
'Commands_G91',
'Commands_G94',
'Commands_G94 G1 G90',
'Commands_G94 G3 G90',
'Commands_M168',
'Commands_M169 M167',
'Commands_M17',
'Commands_M27 M28',
'Commands_M5',
'Commands_M58,',
'Commands_M59',
'Commands_MSG',
'D_D0',
'D_D1',
'D_D=$P_TOOL']
```

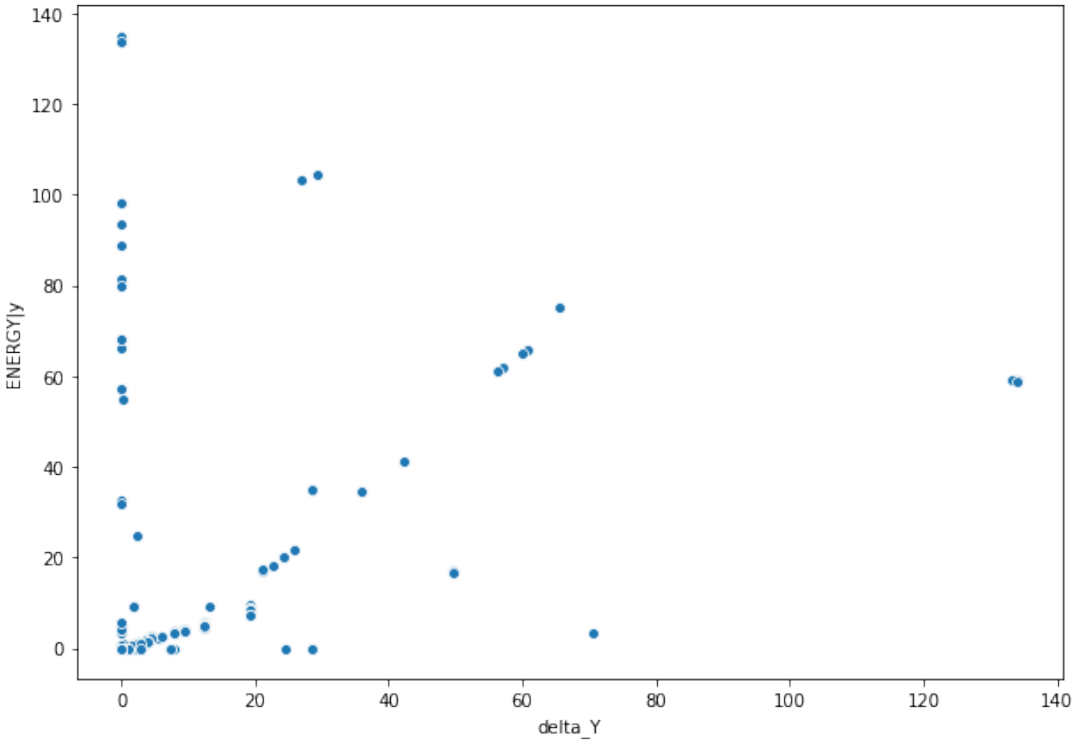
1.4 Visualize Data and show dependencies

[3]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Examine the data by sight

#df.plot(figsize=(30,10))

plt.figure(figsize = (10, 7))
sns.scatterplot(x = df['delta_xy'],
                y = df['ENERGY|x'])

plt.figure(figsize = (10, 7))
sns.scatterplot(x = df['delta_X'],
                y = df['ENERGY|x'])
                #hue = df['Commands'])
                #palette=['dodgerblue','orange', 'green','red'],)

plt.figure(figsize = (10, 7))
sns.scatterplot(x = df['delta_Y'],
                y = df['ENERGY|y'])
                #hue = df['Commands'])
                #palette=['dodgerblue','orange', 'green','red'],)
```

[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1b4e1050>
```


ENERGY|x
delta_xy
0
100
200
300
400
500
0
20
40
60
80
100
120
140


ENERGY|x
delta_X
0
100
200
300
400
500
0
5
10
15
20
25



1.5 Split dataset and train model

[4]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split

#df_cleaned = df.iloc[1:,]
toPredict = ['ENERGY|x', 'ENERGY|y', 'ENERGY|z', 'ENERGY|S', 'ENERGY|T']
y = df[toPredict]
features = ['delta_X', 'delta_Y', 'delta_Z', 'delta_S', 'F_val', 'S', 'D_W',␣
 ↪'Toolchange', 'TurnOp',
 'Commands_G0',
 'Commands_G0 G40 G60',
 'Commands_G0 M106',
 'Commands_G0 M3',
 'Commands_G09',
 'Commands_G1',
 'Commands_G1 G60',
 'Commands_G2',
 'Commands_G4',
```

```
 'Commands_G40',
 'Commands_G41',
 'Commands_G41 G1',
 'Commands_G4F1',
 'Commands_G54 G0',
 'Commands_G90',
 'Commands_G91',
 'Commands_G94',
 'Commands_G94 G1 G90',
 'Commands_G94 G3 G90',
 'Commands_M168',
 'Commands_M169 M167',
 'Commands_M17',
 'Commands_M27 M28',
 'Commands_M5',
 'Commands_M58,',
 'Commands_M59',
 'Commands_MSG',
 'D_D0',
 'D_D1',
 'D_D=$P_TOOL']
X = df[features]
#X = df_cleaned[df_features]

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.15,␣
 ↪random_state = 1)
```

[5]:
```
#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.multioutput import MultiOutputRegressor


test_model = RandomForestRegressor(random_state = 1)
#test_model = MultiOutputRegressor(AdaBoostRegressor(DecisionTreeRegressor(␣
 ↪)))#n_estimators = 5000,
#                                                                          ␣
 ↪max_depth=5,
#                                                                          ␣
 ↪random_state=1,
#                                                                          ␣
 ↪min_samples_leaf=1,
#                                                                          ␣
 ↪min_samples_split = 2)))
```

```
#MODEL TRAINING

#test_model.fit(train_X, train_y)
test_model.fit(X, y)

#PREDICTIONS

predictions = test_model.predict(X)
#predictions = test_model.predict(test_X)

df_pred = pd.DataFrame(data=predictions) #.flatten()
df_pred
#df_test = test_y.set_index(df_pred.index)
#df_test
```

[5]:

	0	1	2	3	4
0	2.813236	0.467707	101.740464	-329.080923	1.423255
1	2.813236	0.467707	101.740464	-329.080923	1.423255
2	0.000000	0.000000	6.714769	4.305998	0.000000
3	0.000000	0.000000	6.714769	4.305998	0.000000
4	0.000000	0.014813	-0.002333	0.000000	0.092763
…	…	…	…	…	…
2208	8.638172	1.360249	0.000000	-685.828604	0.503554
2209	27.084200	1.629831	0.000000	-1735.975893	1.781366
2210	0.000000	0.000000	0.000000	-1.212235	0.188384
2211	0.000000	0.000000	0.000000	-1.212235	0.188384
2212	0.000000	0.000000	0.000000	-1.212235	0.188384

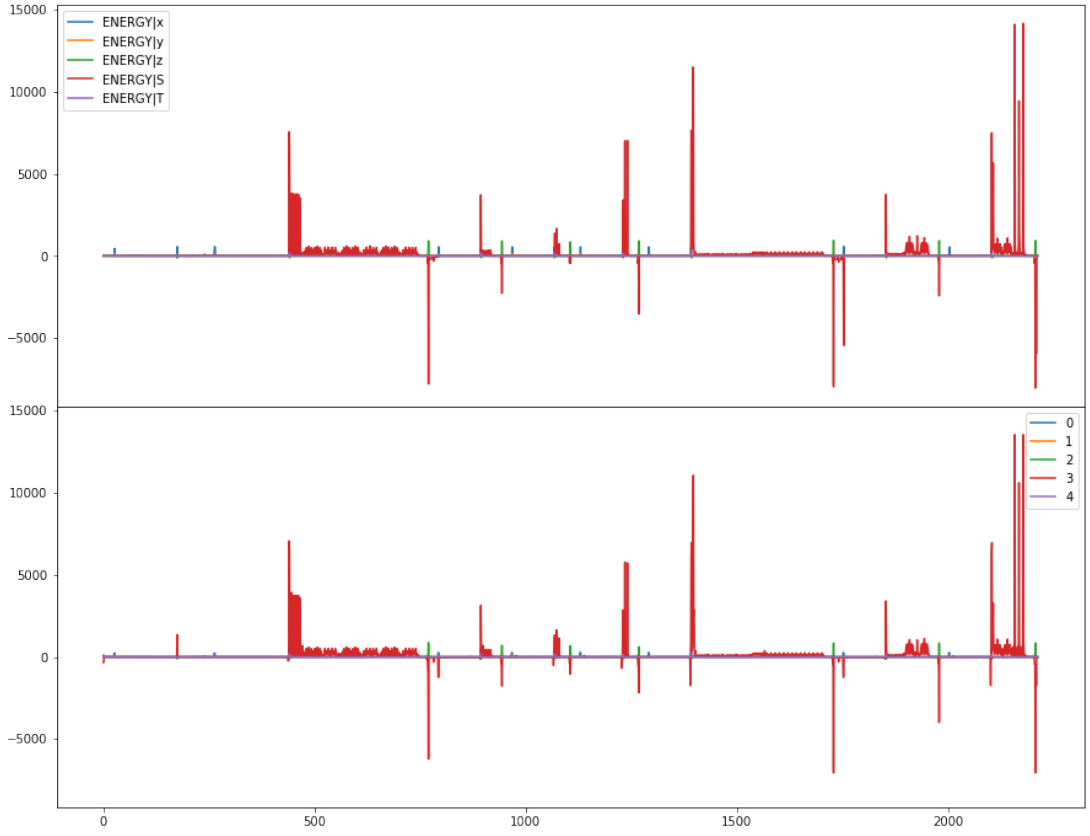
[2213 rows x 5 columns]

1.6 Vergleich der Vorhersagen

1.6.1 Überblick

[6]:
```
from sklearn.metrics import mean_absolute_error
from math import sqrt, pow

fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
 ↪figsize=(15,12))

#y = df_pred
y.plot(ax=axs[0])
df_pred.plot(ax=axs[1])
```

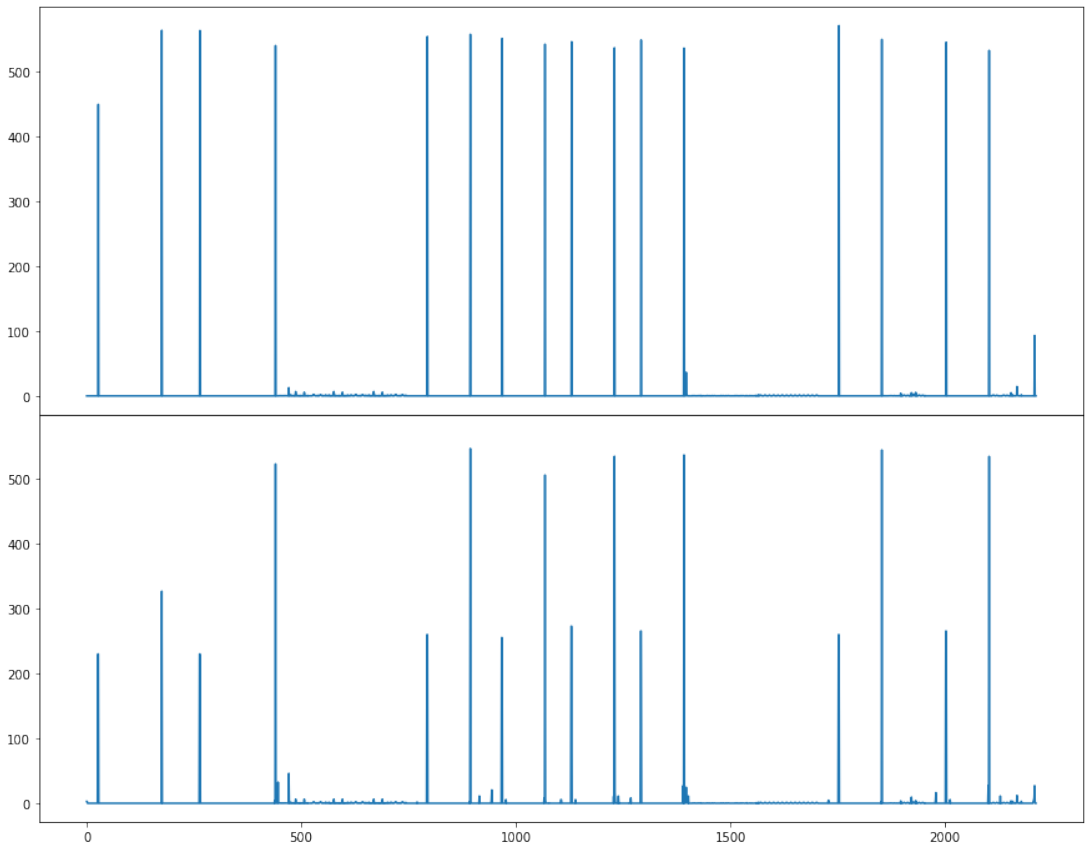
[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ba52290>



1.6.2 X-Achse

[7]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
↪figsize=(15,12))

y['ENERGY|x'].plot(ax=axs[0])
df_pred[0].plot(ax=axs[1])
```

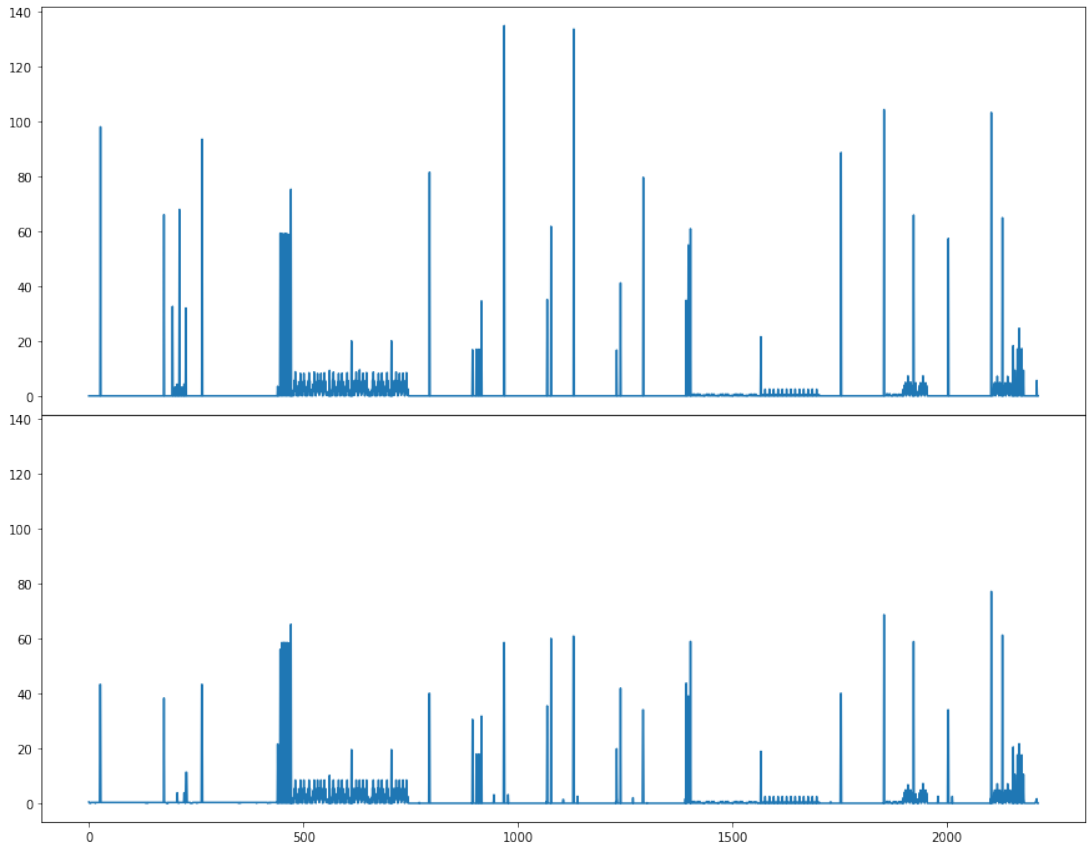
[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c14e8d0>
```



1.6.3 y - Achse

[8]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
↪figsize=(15,12))

y['ENERGY|y'].plot(ax=axs[0])
df_pred[1].plot(ax=axs[1])
```

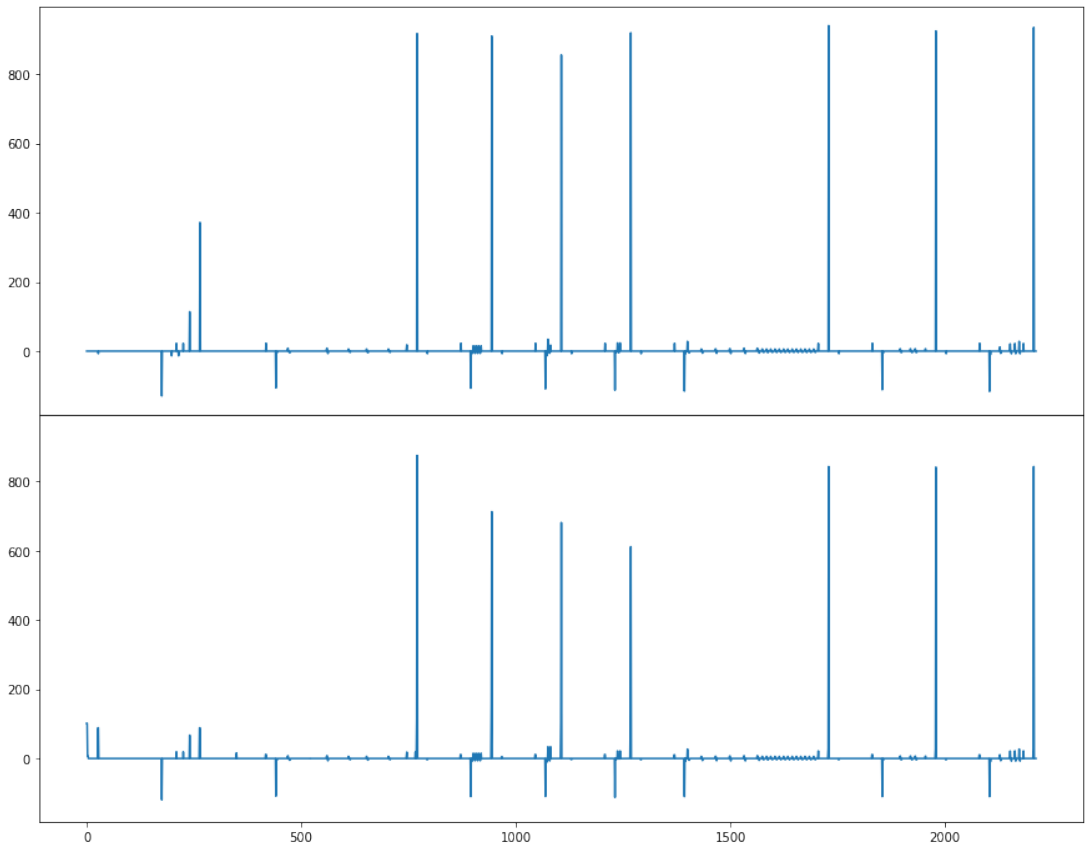
[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1b4c7890>
```



1.6.4 z Achse

[9]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
↪figsize=(15,12))

y['ENERGY|z'].plot(ax=axs[0])
df_pred[2].plot(ax=axs[1])
```

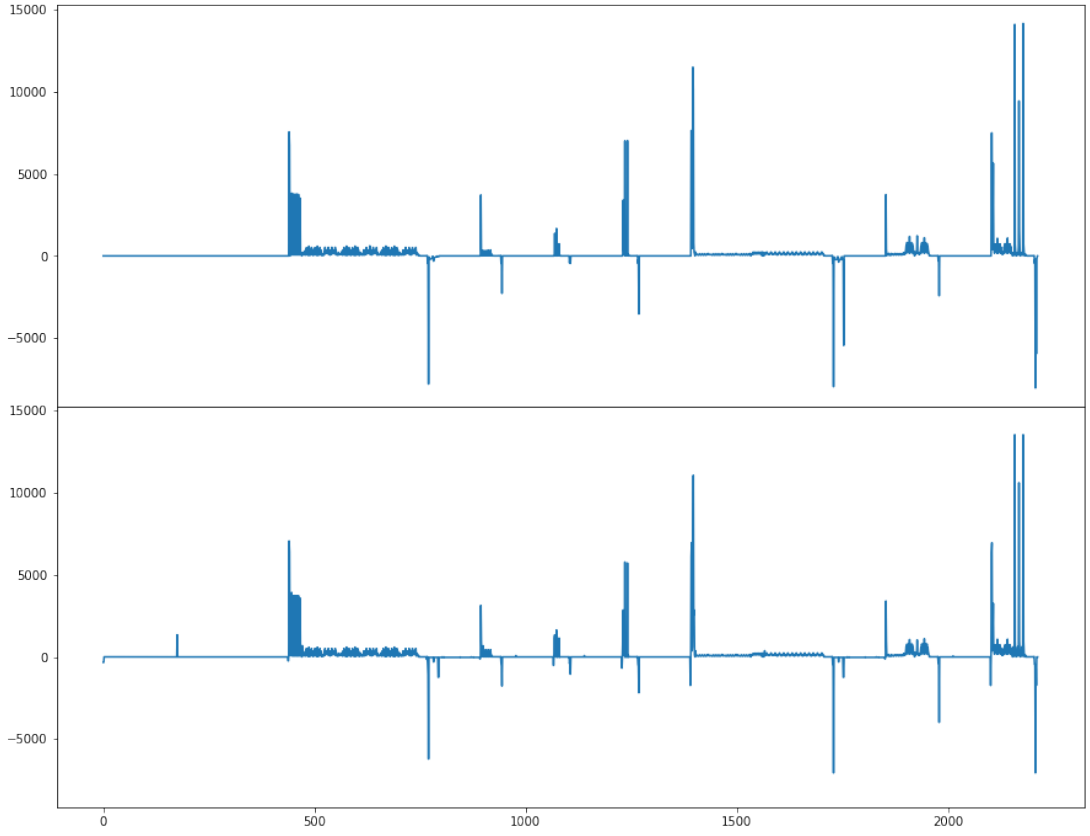
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b49d9d0>



1.6.5 Spindel

[10]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
↪figsize=(15,12))

y['ENERGY|S'].plot(ax=axs[0])
df_pred[3].plot(ax=axs[1])
```

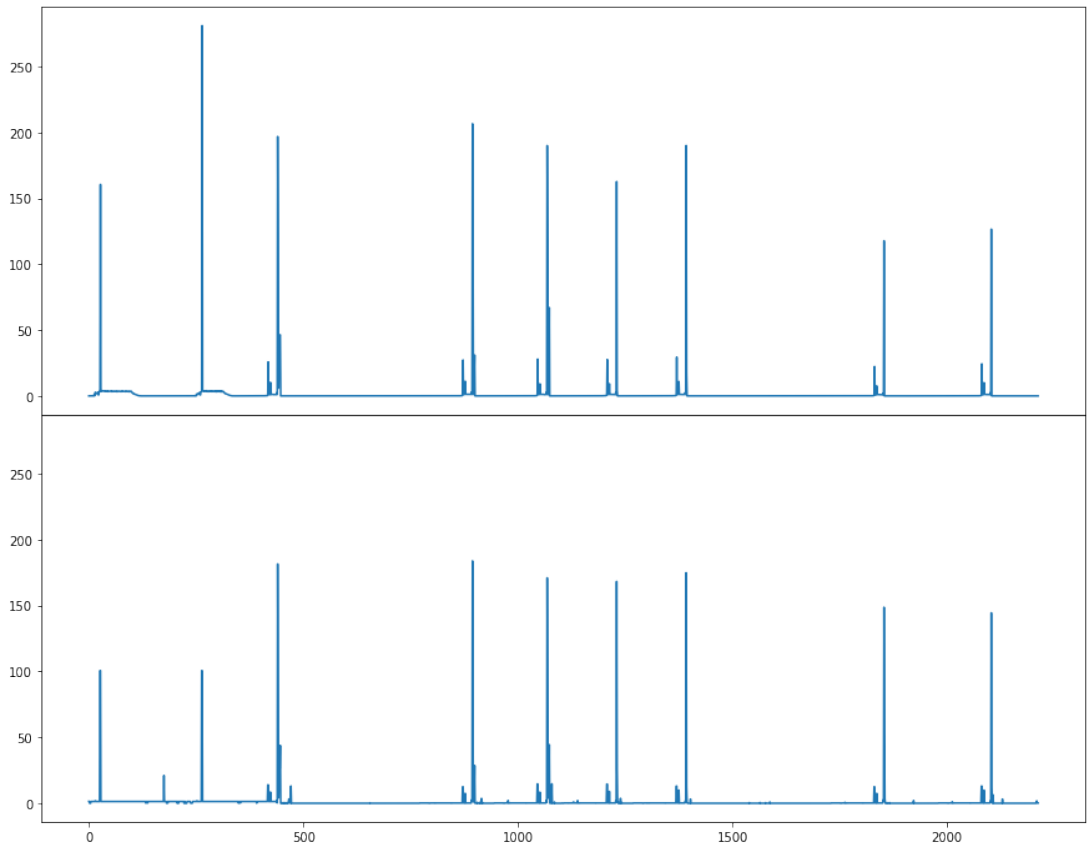
[10]: <matplotlib.axes._subplots.AxesSubplot at 0x105e3a310>



1.6.6 Toolchange

[11]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, gridspec_kw={'hspace': 0},␣
↪figsize=(15,12))

y['ENERGY|T'].plot(ax=axs[0])
df_pred[4].plot(ax=axs[1])
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x116ae78d0>
```



1.7 Vorhersagegüte Quantifizieren

[12]:
```
from sklearn.metrics import mean_absolute_error
from math import sqrt, pow
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import r2_score
from math import sqrt

#activate for Train-Test-Split
#y_true = df_test
#y = y_true

#activate for complete dataset
y_true = y
y_pred = df_pred

gesAbwArr = []
```

```
maeArr = []
rmsArr = []
exVarArr = []

scoresFrame = pd.DataFrame()

for j in range(len(y.columns)):
    deviation = []
    for i in range(len(y.index)):
        if y.iloc[i, j] == 0 and df_pred.iloc[i, j] == 0:
            deviation.append(0)
        elif y.iloc[i, j] == 0:
            deviation.append(0)
        else:
            deviation.append( ( y.iloc[i, j] - df_pred.iloc[i, j]) / y.iloc[i,␣
 ↪j])
    scoresFrame.insert(j, str(j), deviation)
    #scoresFrame[str(j)] = deviation

    mae = round(mean_absolute_error(y.iloc[:,j], df_pred.iloc[:,j]), 2)

    exVar = round(explained_variance_score(y.iloc[:,j], df_pred.iloc[:,j]), 2)
    r2 = round(r2_score(y.iloc[:,j], df_pred.iloc[:,j]), 2)


    rms = round(sqrt(mean_squared_error(y_true.iloc[:,j], y_pred.iloc[:,j])), 2)

    sum_deviation = 0
    for i in range(len(scoresFrame)):
        sum_deviation += abs(scoresFrame.iloc[i, j])  #Fehler?
    mean_deviation = sum_deviation / len(scoresFrame)
    mre = round(mean_deviation * 100, 2)
    mre2 = round(mae/(y.iloc[:,j].mean())*100, 2)
    gesAbw = round((((y.iloc[:, j].sum() / df_pred.iloc[:, j].sum())-1)*100), 2)

    gesAbwArr.append(gesAbw)
    maeArr.append(mae)
    rmsArr.append(rms)
    exVarArr.append(exVar)

    print(f'\n---------------------------------- \n'
          f'{y.columns[j]}\n'
          f'Gesammtabweichung {gesAbw}%\n'
          f'mean absolut error: {mae} Ws. \n'
          f'RMSE: {rms}\n'
          f'Explained Variance: {exVar}\n'
          f'----------------------------------\n')
```

```
#           f'R2: {r2}\n'
#           f'mean of deviations: \u00B1 {mre}% \n'
#           f'mae/mean_meassured: \u00B1 {mre2}%\n'

print(gesAbwArr)
print(maeArr)
print(rmsArr)
print(exVarArr)
print()

row = 10
print(y.iloc[[row]])
print(df_pred.iloc[[row]])
print(scoresFrame.iloc[[row]])
```

```
-----------------------------------
ENERGY|x
Gesammtabweichung 3.33%
mean absolut error: 2.3 Ws.
RMSE: 23.76
Explained Variance: 0.73
-----------------------------------


-----------------------------------
ENERGY|y
Gesammtabweichung 3.6%
mean absolut error: 0.59 Ws.
RMSE: 4.75
Explained Variance: 0.7
-----------------------------------


-----------------------------------
ENERGY|z
Gesammtabweichung 2.89%
mean absolut error: 1.27 Ws.
RMSE: 13.23
Explained Variance: 0.94
-----------------------------------


-----------------------------------
ENERGY|S
Gesammtabweichung 1.58%
```

```
mean absolut error: 36.96 Ws.
RMSE: 231.08
Explained Variance: 0.93
-----------------------------------


-----------------------------------
ENERGY|T
Gesammtabweichung 3.52%
mean absolut error: 0.89 Ws.
RMSE: 5.81
Explained Variance: 0.8
-----------------------------------

[3.33, 3.6, 2.89, 1.58, 3.52]
[2.3, 0.59, 1.27, 36.96, 0.89]
[23.76, 4.75, 13.23, 231.08, 5.81]
[0.73, 0.7, 0.94, 0.93, 0.8]

    ENERGY|x  ENERGY|y  ENERGY|z  ENERGY|S  ENERGY|T
10       0.0       0.0       0.0       0.0  0.085961
      0         1         2    3         4
10  0.0  0.293551 -0.058887  0.0  1.332531
      0    1    2    3        4
10  0.0  0.0  0.0  0.0 -14.5015
```

1.8 Feature Importance

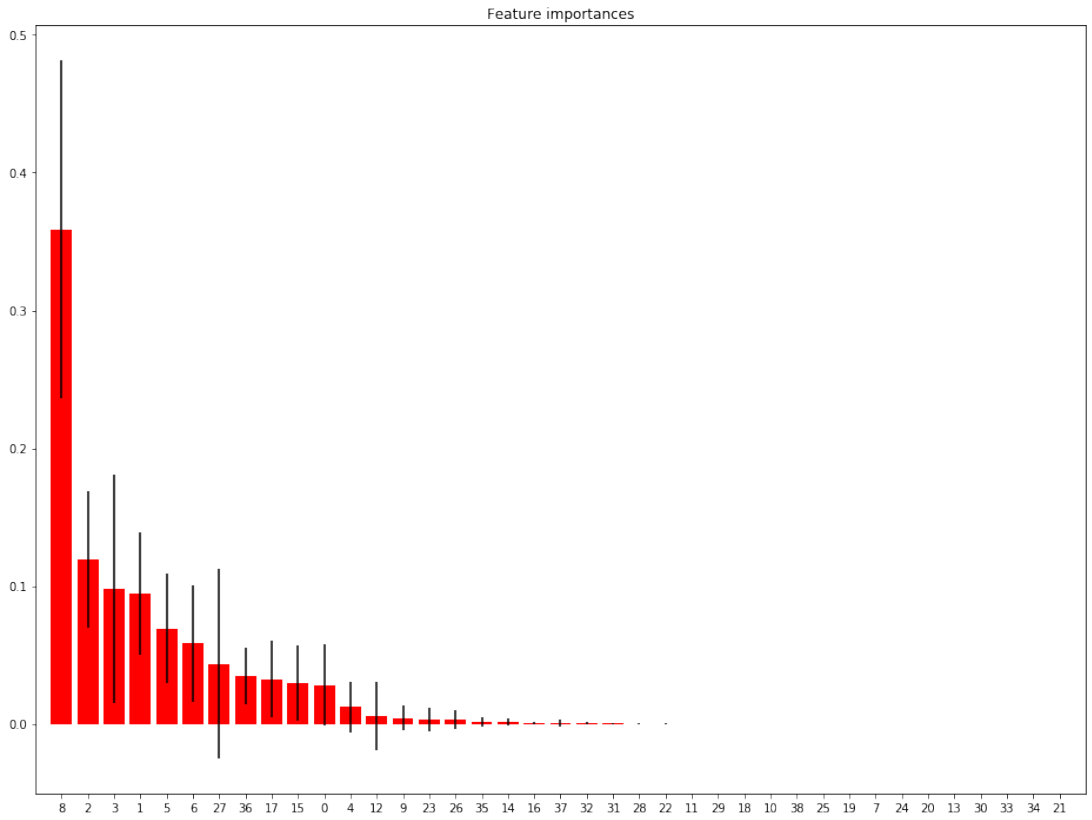
[13]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier

# # Build a classification task using 3 informative features
# X, y = make_classification(n_samples=1000,
#                            n_features=10,
#                            n_informative=3,
#                            n_redundant=0,
#                            n_repeated=0,
#                            n_classes=2,
#                            random_state=0,
#                            shuffle=False)

# # Build a forest and compute the impurity-based feature importances
# forest = ExtraTreesClassifier(n_estimators=250,
```

```
#                                 random_state=0)

# forest.fit(X, y)

importances = test_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in test_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking:
1. feature 8 (0.358939)
2. feature 2 (0.119085)
3. feature 3 (0.098041)
4. feature 1 (0.094621)
5. feature 5 (0.069303)
6. feature 6 (0.058254)
7. feature 27 (0.043544)
8. feature 36 (0.034922)
9. feature 17 (0.032524)
10. feature 15 (0.029309)
11. feature 0 (0.027932)
12. feature 4 (0.012087)
13. feature 12 (0.005758)
14. feature 9 (0.004064)
15. feature 23 (0.003268)
16. feature 26 (0.003158)
17. feature 35 (0.001720)
18. feature 14 (0.001414)
19. feature 16 (0.000642)
20. feature 37 (0.000571)
21. feature 32 (0.000508)
22. feature 31 (0.000131)
23. feature 28 (0.000105)
24. feature 22 (0.000059)
25. feature 11 (0.000017)
26. feature 29 (0.000008)
27. feature 18 (0.000007)
28. feature 10 (0.000003)
29. feature 38 (0.000002)
30. feature 25 (0.000002)
31. feature 19 (0.000002)
```

```
32. feature 7 (0.000001)
33. feature 24 (0.000001)
34. feature 20 (0.000000)
35. feature 13 (0.000000)
36. feature 30 (0.000000)
37. feature 33 (0.000000)
38. feature 34 (0.000000)
39. feature 21 (0.000000)
```

[14]:
```
# Plot the impurity-based feature importances of the forest
plt.figure(figsize = (16,12))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
        color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```



1.9 Show datapoints with highest errors

[15]:
```
absErrors = pd.DataFrame()

for j in range(len(y.columns)):
    deviation = []
    for i in range(len(y.index)):
        deviation.append(abs(y.iloc[i, j] - df_pred.iloc[i, j]))
    absErrors.insert(j, str(j), deviation)

#absErrors
dfErrors = pd.concat([dfCompare, absErrors], axis = 1)
dfErrors
```

[15]:

	N	Commands	G	M	X	Y	Z	I	J	F	…	\
0	N32		G0	G0	0.0	0.0	610.0	0.0	0.0	NaN	…	
1	N32		G0	G0	0.0	0.0	610.0	0.0	0.0	NaN	…	
2	N34		G0	G0	0.0	0.0	610.0	0.0	0.0	NaN	…	
3	N34		G0	G0	0.0	0.0	610.0	0.0	0.0	NaN	…	
4	NaN		MSG		0.0	0.0	610.0	0.0	0.0	NaN	…	
…	…	…	…	..	…	…	…	…	…	…	…	
2208	N1842		G4	G4	0.0	0.0	610.0	0.0	0.0	NaN	…	
2209	N1842		G4	G4	0.0	0.0	610.0	0.0	0.0	NaN	…	
2210	N1844		MSG		0.0	0.0	610.0	0.0	0.0	NaN	…	
2211	N1844		MSG		0.0	0.0	610.0	0.0	0.0	NaN	…	
2212	NaN		MSG		0.0	0.0	610.0	0.0	0.0	NaN	…	

	ENERGY|x	ENERGY|y	ENERGY|z	ENERGY|S	ENERGY|T	0	\
0	0.000000	0.00000	0.0	0.000000	0.0	2.813236	
1	0.000000	0.00000	0.0	0.000000	0.0	2.813236	
2	0.000000	0.00000	0.0	0.000000	0.0	0.000000	
3	0.000000	0.00000	0.0	0.000000	0.0	0.000000	
4	0.000000	0.00000	0.0	0.000000	0.0	0.000000	
…	…	…	…	…	…	…	
2208	0.000000	0.00000	0.0	-219.734254	0.0	8.638172	
2209	93.042402	5.59896	0.0	-5963.601137	0.0	65.958202	
2210	0.000000	0.00000	0.0	-219.693970	0.0	0.000000	
2211	0.000000	0.00000	0.0	-59.709629	0.0	0.000000	
2212	0.000000	0.00000	0.0	0.000000	0.0	0.000000	

	1	2	3	4
0	0.467707	101.740464	329.080923	1.423255
1	0.467707	101.740464	329.080923	1.423255
2	0.000000	6.714769	4.305998	0.000000
3	0.000000	6.714769	4.305998	0.000000
4	0.014813	0.002333	0.000000	0.092763
…	…	…	…	…

```
2208  1.360249    0.000000   466.094350  0.503554
2209  3.969130    0.000000  4227.625244  1.781366
2210  0.000000    0.000000   218.481735  0.188384
2211  0.000000    0.000000    58.497393  0.188384
2212  0.000000    0.000000     1.212235  0.188384

[2213 rows x 36 columns]
```

[16]:
```
dfErrors
```

[16]:
```
          N Commands      G   M    X    Y      Z    I    J    F  …  \
0       N32       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …
1       N32       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …
2       N34       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …
3       N34       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …
4       NaN      MSG             0.0  0.0  610.0  0.0  0.0  NaN  …
…     …       …    …   ..  …  …    …  …  …  …  …
2208  N1842       G4  G4         0.0  0.0  610.0  0.0  0.0  NaN  …
2209  N1842       G4  G4         0.0  0.0  610.0  0.0  0.0  NaN  …
2210  N1844      MSG             0.0  0.0  610.0  0.0  0.0  NaN  …
2211  N1844      MSG             0.0  0.0  610.0  0.0  0.0  NaN  …
2212    NaN      MSG             0.0  0.0  610.0  0.0  0.0  NaN  …

       ENERGY|x  ENERGY|y  ENERGY|z     ENERGY|S  ENERGY|T         0  \
0      0.000000   0.00000       0.0     0.000000       0.0  2.813236
1      0.000000   0.00000       0.0     0.000000       0.0  2.813236
2      0.000000   0.00000       0.0     0.000000       0.0  0.000000
3      0.000000   0.00000       0.0     0.000000       0.0  0.000000
4      0.000000   0.00000       0.0     0.000000       0.0  0.000000
…         …       …        …           …       …        …
2208   0.000000   0.00000       0.0  -219.734254       0.0  8.638172
2209  93.042402   5.59896       0.0 -5963.601137       0.0 65.958202
2210   0.000000   0.00000       0.0  -219.693970       0.0  0.000000
2211   0.000000   0.00000       0.0   -59.709629       0.0  0.000000
2212   0.000000   0.00000       0.0     0.000000       0.0  0.000000

             1           2           3         4
0     0.467707  101.740464   329.080923  1.423255
1     0.467707  101.740464   329.080923  1.423255
2     0.000000    6.714769     4.305998  0.000000
3     0.000000    6.714769     4.305998  0.000000
4     0.014813    0.002333     0.000000  0.092763
…        …          …          …       …
2208  1.360249    0.000000   466.094350  0.503554
2209  3.969130    0.000000  4227.625244  1.781366
2210  0.000000    0.000000   218.481735  0.188384
2211  0.000000    0.000000    58.497393  0.188384
```

```
2212  0.000000    0.000000     1.212235  0.188384

[2213 rows x 36 columns]
```

Hauptprogramm-zweiteSeite

October 22, 2020

1 Machine-Learning Programm

1.1 (Train & Test = verschiedene Daten)

1.2 Importing gcode in csv-file

[1]:
```
#import data from csv file for further processing
import pandas as pd
import numpy as np

#extended dataset
df_filepath = './df_parsed.csv'
df2_filepath = './df2_parsed.csv'
#df = pd.read_csv(df_filepath, index_col=['Unnamed: 0'])
#df2 = pd.read_csv(df2_filepath, index_col=['Unnamed: 0'])


#anders herum
df = pd.read_csv(df2_filepath, index_col=['Unnamed: 0'])
df2 = pd.read_csv(df_filepath, index_col=['Unnamed: 0'])

df.head()
#df.tail()
#df.columns
#pd.set_option("display.max_rows", None, "display.max_columns", None)
```

[1]:
```
     N Commands      G   M    X    Y      Z    I    J    F  …   T=  TurnOp  \
0  NaN      MSG            0.0  0.0  610.0  0.0  0.0  NaN  …  NaN       0
1  N32       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …  NaN       0
2  N32       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …  NaN       0
3  N34       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …  NaN       0
4  N34       G0   G0        0.0  0.0  610.0  0.0  0.0  NaN  …  NaN       0

   Toolchange  TURN                                                GCode  \
0           0   NaN  MSG("FLOOR_WALL_IPW , Tool : T_A_SK40_SF_DM16_A")
1           0   NaN                                                NaN
2           0   NaN                                                NaN
```

```
3           0   NaN                                                 NaN
4           0   NaN                                                 NaN

   ENERGY|x  ENERGY|y ENERGY|z ENERGY|S  ENERGY|T
0       0.0       0.0      0.0      0.0       0.0
1       0.0       0.0      0.0      0.0       0.0
2       0.0       0.0      0.0      0.0       0.0
3       0.0       0.0      0.0      0.0       0.0
4       0.0       0.0      0.0      0.0       0.0

[5 rows x 31 columns]
```

1.3 Make information in data visible for ML-model

[2]:
```
# inspect data here

df.describe()
```

[2]:
```
                X            Y            Z            I            J  \
count  2632.000000  2632.000000  2632.000000  2632.000000  2632.000000
mean     -0.346558     5.114300   339.129900     0.005070     0.005068
std       5.712738    24.588699   304.748957     1.438973     1.177767
min     -21.000000   -65.748000   -11.200000   -20.861000   -11.855000
25%       0.000000     0.000000    -2.700000     0.000000     0.000000
50%       0.000000     0.000000   610.000000     0.000000     0.000000
75%       0.000000    16.340000   610.000000     0.000000     0.000000
max      21.000000    65.652000   610.000000    20.861000    22.247000

          delta_xy      delta_X      delta_Y      delta_Z       delta_S  …  \
count  2632.000000  2632.000000  2632.000000  2632.000000   2632.000000  …
mean      1.951557     0.793742     1.410078     0.000000      0.000000  …
std       8.015243     2.802926     7.588044    57.969187    699.030115  …
min       0.000000     0.000000     0.000000  -608.000000 -13000.000000  …
25%       0.000000     0.000000     0.000000     0.000000      0.000000  …
50%       0.000000     0.000000     0.000000     0.000000      0.000000  …
75%       0.000000     0.000000     0.000000     0.000000      0.000000  …
max     131.400000    30.811000   131.400000   608.000000  13000.000000  …

               D_W            S  TurnOp  Toolchange  TURN     ENERGY|x  \
count  2632.000000   2632.000000  2632.0  2632.000000   0.0  2632.000000
mean      7.587804   3505.661094     0.0     0.004939   NaN     5.235004
std       4.879583   4779.267816     0.0     0.070119   NaN    51.274160
min       0.000000      0.000000     0.0     0.000000   NaN     0.000000
25%       3.000000      0.000000     0.0     0.000000   NaN     0.000000
50%       6.000000      0.000000     0.0     0.000000   NaN     0.000000
75%      10.000000   8000.000000     0.0     0.000000   NaN     0.000000
```

```
max      16.000000  13000.000000     0.0     1.000000   NaN   581.742320

          ENERGY|y     ENERGY|z     ENERGY|S     ENERGY|T
count  2632.000000  2632.000000  2632.000000  2632.000000
mean      1.700706     3.545730    84.817385     1.392565
std      10.831427    58.581012   612.857259    13.152782
min       0.000000  -148.005430 -8041.625124     0.000000
25%       0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000    84.672564     0.000000
max     172.307578   928.606553  7846.936889   247.118818

[8 rows x 21 columns]
```

[3]:
```
df2.describe()
```

[3]:
```
                X            Y            Z            I            J  \
count  2213.000000  2213.000000  2213.000000  2213.000000  2213.000000
mean     -1.300142     1.552118   285.452850    -0.008438     0.016631
std       6.403329    21.329063   303.116148     2.433507     0.865189
min     -19.907000   -70.500000    -8.000000   -14.300000    -7.300000
25%      -0.100000    -2.262000    -3.200000     0.000000     0.000000
50%       0.000000     0.000000    50.000000     0.000000     0.000000
75%       0.000000     2.262000   610.000000     0.000000     0.000000
max      14.300000    67.500000   610.000000    14.300000     7.300000

          delta_xy      delta_X      delta_Y       delta_Z       delta_S  \
count  2213.000000  2213.000000  2213.000000  2.213000e+03   2213.000000
mean      2.128933     0.553685     1.810044  2.054891e-16      0.000000
std       8.751811     1.789431     8.622186  5.131558e+01    694.202289
min       0.000000     0.000000     0.000000 -6.070000e+02 -13000.000000
25%       0.000000     0.000000     0.000000  0.000000e+00      0.000000
50%       0.000000     0.000000     0.000000  0.000000e+00      0.000000
75%       0.632001     0.000000     0.085000  0.000000e+00      0.000000
max     134.100000    25.906000   134.100000  6.070000e+02  13000.000000

             F_val          D_W             S       TurnOp   Toolchange  \
count  2213.000000  2213.000000   2213.000000  2213.000000  2213.000000
mean    120.208766     4.382738   4584.274740     0.001808     0.004067
std     264.405105     3.416697   5465.059687     0.042486     0.063657
min       0.000000     0.000000      0.000000     0.000000     0.000000
25%       0.000000     2.800000      0.000000     0.000000     0.000000
50%       0.000000     3.000000      0.000000     0.000000     0.000000
75%       0.000000     8.000000  10000.000000     0.000000     0.000000
max     750.000000    10.000000  13000.000000     1.000000     1.000000

          ENERGY|x     ENERGY|y     ENERGY|z     ENERGY|S     ENERGY|T
```

```
count  2213.000000  2213.000000  2213.000000   2213.000000  2213.000000
mean      4.129043     1.478090     2.903118    118.339919     1.492249
std      46.044159     8.667296    52.573514    857.416625    13.030186
min       0.000000     0.000000  -129.385580  -8043.649349     0.000000
25%       0.000000     0.000000     0.000000      0.000000     0.000000
50%       0.000000     0.000000     0.000000      0.000000     0.000000
75%       0.000000     0.083114     0.000000    101.593985     0.000000
max     570.195688   134.946734   940.387548  14158.866014   281.044521
```

[4]:
```
np.unique(df['Commands'])
```

[4]:
```
array(['G0', 'G0 G40 G60', 'G0 G90', 'G0 M106', 'G0 M3', 'G09', 'G1',
       'G1 G60', 'G2', 'G4', 'G40', 'G41 G1', 'G41 G94 G1 G90', 'G4F1',
       'G54 G0', 'G90', 'G91', 'G94', 'G94 G1 G90', 'M168', 'M169 M167',
       'M17', 'M27 M28', 'M5', 'M58;', 'M59', 'MSG'], dtype=object)
```

[5]:
```
list(np.unique(df['Commands']))
list(np.unique(df2['Commands']))

for i in np.unique(df['Commands']):
    if not i in list(np.unique(df2['Commands'])):
        print(i)
```

```
G0 G90
G41 G94 G1 G90
M58;
```

[6]:
```
# neue Befehle in zweitem Frame ersetzen, damit Frames gleich

for i in range(len(df2)) :
    if str(df2.loc[i, "Commands"]).strip() == 'G0 G90':
        df2.at[i,'Commands'] = 'G0'
    elif str(df2.loc[i, "Commands"]).strip() == 'G41 G94 G1 G90':
        df2.at[i,'Commands'] = 'G1'
    elif str(df2.loc[i, "Commands"]).strip() == 'M58;':
        df2.at[i,'Commands'] = 'M58,'

    elif str(df2.loc[i, "Commands"]).strip() == 'G41':
        df2.at[i,'Commands'] = 'MSG'
    elif str(df2.loc[i, "Commands"]).strip() == 'G94 G3 G90':
        df2.at[i,'Commands'] = 'G2'
```

[7]:
```
# Befehle im alten Frame ersetzen damit Frames gleich
for i in range(len(df)) :
    if str(df.loc[i, "Commands"]).strip() == 'G41':
        df.at[i,'Commands'] = 'MSG'
    elif str(df.loc[i, "Commands"]).strip() == 'G94 G3 G90':
```

```
        df.at[i,'Commands'] = 'G2'

    elif str(df.loc[i, "Commands"]).strip() == 'G0 G90':
        df.at[i,'Commands'] = 'G0'
    elif str(df.loc[i, "Commands"]).strip() == 'G41 G94 G1 G90':
        df.at[i,'Commands'] = 'G1'
    elif str(df.loc[i, "Commands"]).strip() == 'M58;':
        df.at[i,'Commands'] = 'M58,'
```

[8]:
```
list(np.unique(df['Commands']))
```

[8]:
```
['G0',
 'G0 G40 G60',
 'G0 M106',
 'G0 M3',
 'G09',
 'G1',
 'G1 G60',
 'G2',
 'G4',
 'G40',
 'G41 G1',
 'G4F1',
 'G54 G0',
 'G90',
 'G91',
 'G94',
 'G94 G1 G90',
 'M168',
 'M169 M167',
 'M17',
 'M27 M28',
 'M5',
 'M58,',
 'M59',
 'MSG']
```

[9]:
```
df = pd.get_dummies(df, columns=['Commands', 'D'])
df2 = pd.get_dummies(df2, columns=['Commands', 'D'])
```

[10]:
```
list(df.columns.values)
```

[10]:
```
['N',
 'G',
 'M',
 'X',
 'Y',
```

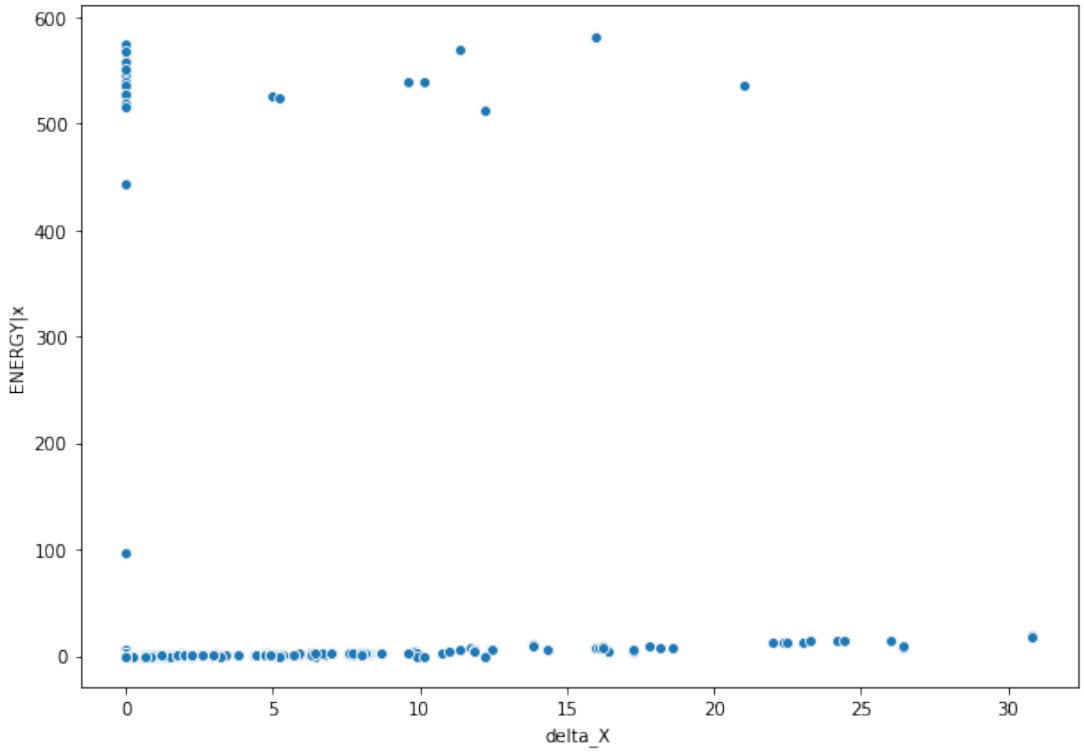
```
'Z',
'I',
'J',
'F',
'delta_xy',
'delta_X',
'delta_Y',
'delta_Z',
'delta_S',
'F_val',
'D_W',
'Werkzeug',
'Bearbeitung',
'S',
'T=',
'TurnOp',
'Toolchange',
'TURN',
'GCode',
'ENERGY|x',
'ENERGY|y',
'ENERGY|z',
'ENERGY|S',
'ENERGY|T',
'Commands_G0',
'Commands_G0 G40 G60',
'Commands_G0 M106',
'Commands_G0 M3',
'Commands_G09',
'Commands_G1',
'Commands_G1 G60',
'Commands_G2',
'Commands_G4',
'Commands_G40',
'Commands_G41 G1',
'Commands_G4F1',
'Commands_G54 G0',
'Commands_G90',
'Commands_G91',
'Commands_G94',
'Commands_G94 G1 G90',
'Commands_M168',
'Commands_M169 M167',
'Commands_M17',
'Commands_M27 M28',
'Commands_M5',
'Commands_M58,',
```

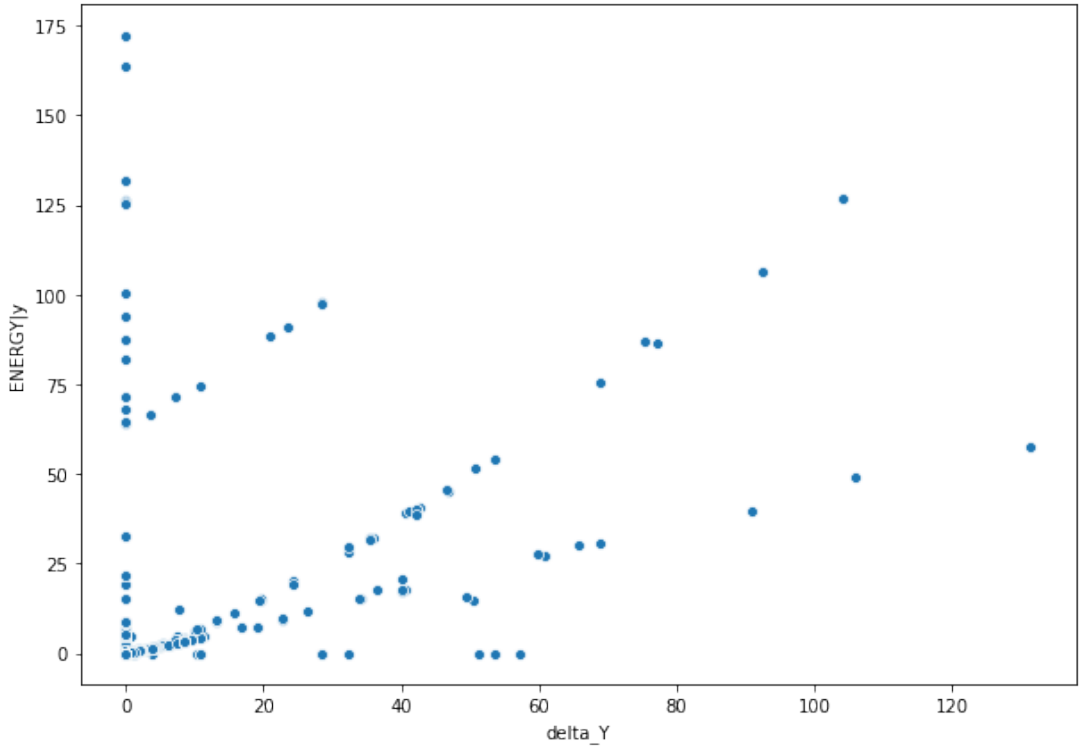
```
 'Commands_M59',
 'Commands_MSG',
 'D_D0',
 'D_D1',
 'D_D=$P_TOOL']
```

1.4 Visualize Data and show dependencies

[11]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Examine the data by sight

#df.plot(figsize=(30,10))

plt.figure(figsize = (10, 7))
sns.scatterplot(x = df['delta_X'],
                y = df['ENERGY|x'])
                #hue = df['Commands'])
                #palette=['dodgerblue','orange', 'green','red'],)

plt.figure(figsize = (10, 7))
sns.scatterplot(x = df['delta_Y'],
                y = df['ENERGY|y'])
                #hue = df['Commands'])
                #palette=['dodgerblue','orange', 'green','red'],)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1a7dff10>
```


ENERGY|x
delta_X
600
500
400
300
200
100
0
0
5
10
15
20
25
30


ENERGY|y
delta_Y
175
150
125
100
75
50
25
0
0
20
40
60
80
100
120

1.5 Split dataset and train model

[12]:
```
#split dataset into train and test dataset

from sklearn.model_selection import train_test_split

#df_cleaned = df.iloc[1:,]
toPredict = ['ENERGY|x', 'ENERGY|y', 'ENERGY|z', 'ENERGY|S', 'ENERGY|T']
y = df[toPredict]
features = ['delta_X', 'delta_Y', 'delta_Z', 'delta_S', 'F_val', 'S', 'D_W',␣
 ↪'Toolchange', 'TurnOp',
 'Commands_G0',
 'Commands_G0 G40 G60',
 'Commands_G0 M106',
 'Commands_G0 M3',
 'Commands_G09',
 'Commands_G1',
 'Commands_G1 G60',
 'Commands_G2',
 'Commands_G4',
 'Commands_G40',
 'Commands_G41 G1',
 'Commands_G4F1',
 'Commands_G54 G0',
 'Commands_G90',
 'Commands_G91',
 'Commands_G94',
 'Commands_G94 G1 G90',
 'Commands_M168',
 'Commands_M169 M167',
 'Commands_M17',
 'Commands_M27 M28',
 'Commands_M5',
 'Commands_M58,',
 'Commands_M59',
 'Commands_MSG',
 'D_D0',
 'D_D1',
 'D_D=$P_TOOL']

X = df[features]

#train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 2)
```

[13]:
```
y2 = df2[toPredict]
X2 = df2[features]
```

[14]:
```
#build ML-model an train it

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.multioutput import MultiOutputRegressor

#AdaBoostRegressor(DecisionTreeRegressor(max_depth=5),
#                          n_estimators=300, random_state = 1)

test_model = RandomForestRegressor(random_state = 1)
#test_model =␣
 ↪MultiOutputRegressor(AdaBoostRegressor(DecisionTreeRegressor(#n_estimators =␣
 ↪5000,
#                                                                  ␣
 ↪max_depth=5,
#                                                                  ␣
 ↪random_state=1,
#                                                                  ␣
 ↪min_samples_leaf=1,
#                                                                  ␣
 ↪min_samples_split = 2)))

#test_model = DecisionTreeRegressor(min_samples_leaf=1, max_depth=5,␣
 ↪random_state = 11)

#test_model.fit(train_X, train_y)
#test_model.fit(X, y.iloc[:,3])
test_model.fit(X, y)
```

[14]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=1, verbose=0, warm_start=False)
```

1.6 Vergleich der Vorhersagen

[15]:
```
from sklearn.metrics import mean_absolute_error
from math import sqrt, pow
```

[16]:
```
y
```

[16]:

	ENERGY|x	ENERGY|y	ENERGY|z	ENERGY|S	ENERGY|T
0	0.000000	0.000000	0.0	0.0	0.0
1	0.000000	0.000000	0.0	0.0	0.0
2	0.000000	0.000000	0.0	0.0	0.0
3	0.000000	0.000000	0.0	0.0	0.0
4	0.000000	0.000000	0.0	0.0	0.0
…	…	…	…	…	…
2627	0.000000	0.000000	0.0	0.0	0.0
2628	97.641451	8.555121	0.0	0.0	0.0
2629	0.000000	0.000000	0.0	0.0	0.0
2630	0.000000	0.000000	0.0	0.0	0.0
2631	0.000000	0.000000	0.0	0.0	0.0

[2632 rows x 5 columns]

[17]:
```
predictions = test_model.predict(X2)

df_pred = pd.DataFrame(data=predictions)
df_pred
```

[17]:

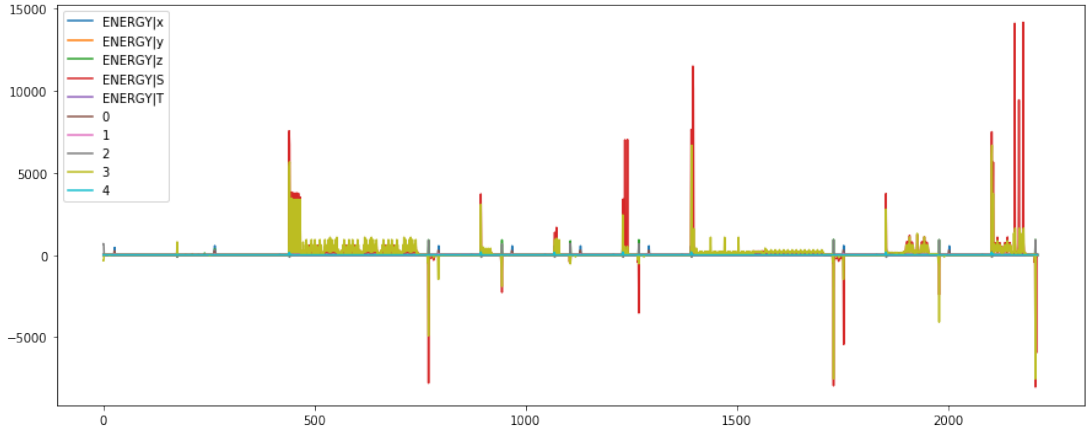
	0	1	2	3	4
0	9.707158	2.302436	673.792795	-351.060894	0.002094
1	9.707158	2.302436	673.792795	-351.060894	0.002094
2	0.000000	0.000000	3.030461	7.215761	2.224146
3	0.000000	0.000000	3.030461	7.215761	2.224146
4	0.000000	0.000000	0.000000	0.000000	0.071490
…	…	…	…	…	…
2208	3.296344	0.332590	0.000000	3.883694	1.883532
2209	4.979714	0.436311	0.000000	0.008459	2.271359
2210	0.000000	0.000000	0.000000	-9.186817	0.183654
2211	0.000000	0.000000	0.000000	-9.186817	0.183654
2212	0.000000	0.000000	0.000000	-9.186817	0.183654

[2213 rows x 5 columns]

1.6.1 Überblick

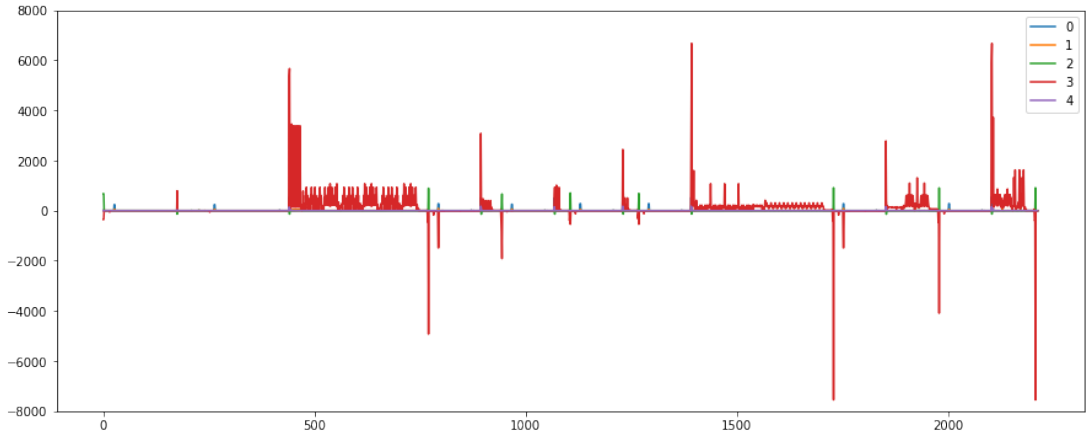
[18]:
```
ax1 = y2.plot(figsize=(15,6))
df_pred.plot(figsize=(15,6), ax = ax1)
```

[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b88f690>



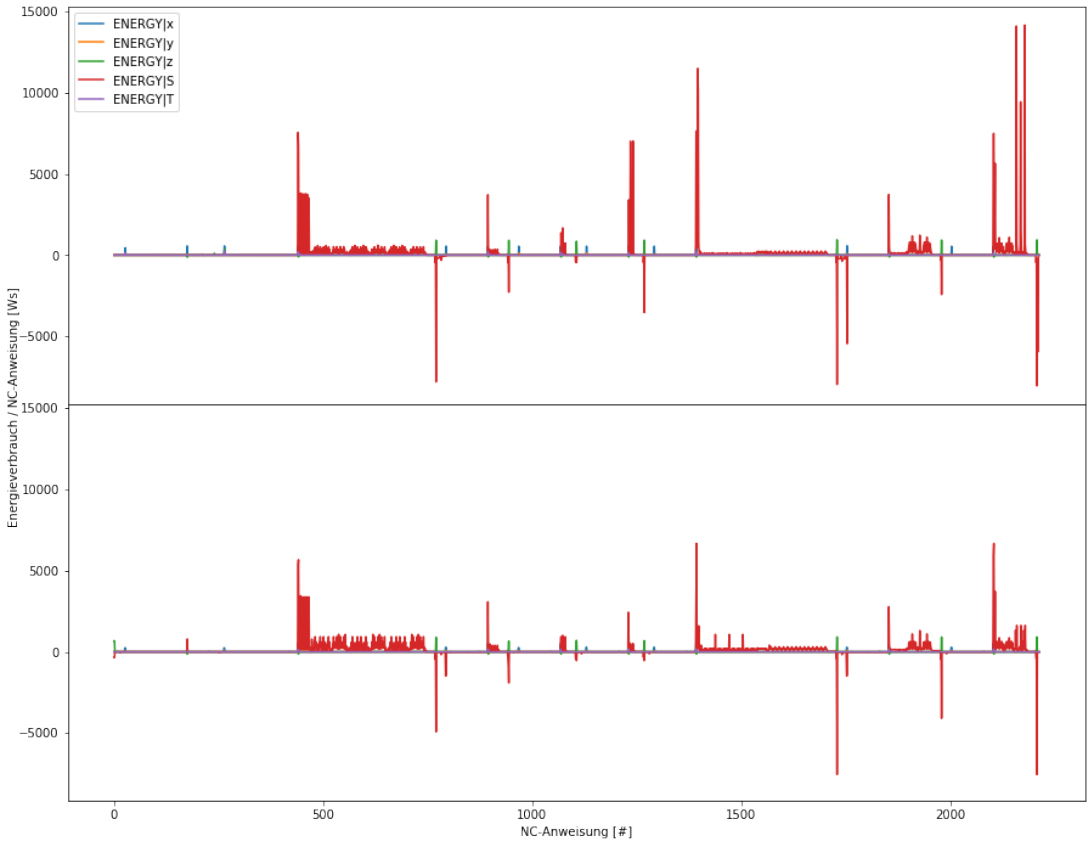
[19]:
```
df_pred.plot(figsize=(15,6), ylim = (-8000, 8000))
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b97ead0>



[20]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,12))

y2.plot(ax=axs[0])
df_pred.plot(ax=axs[1], legend=None)
```

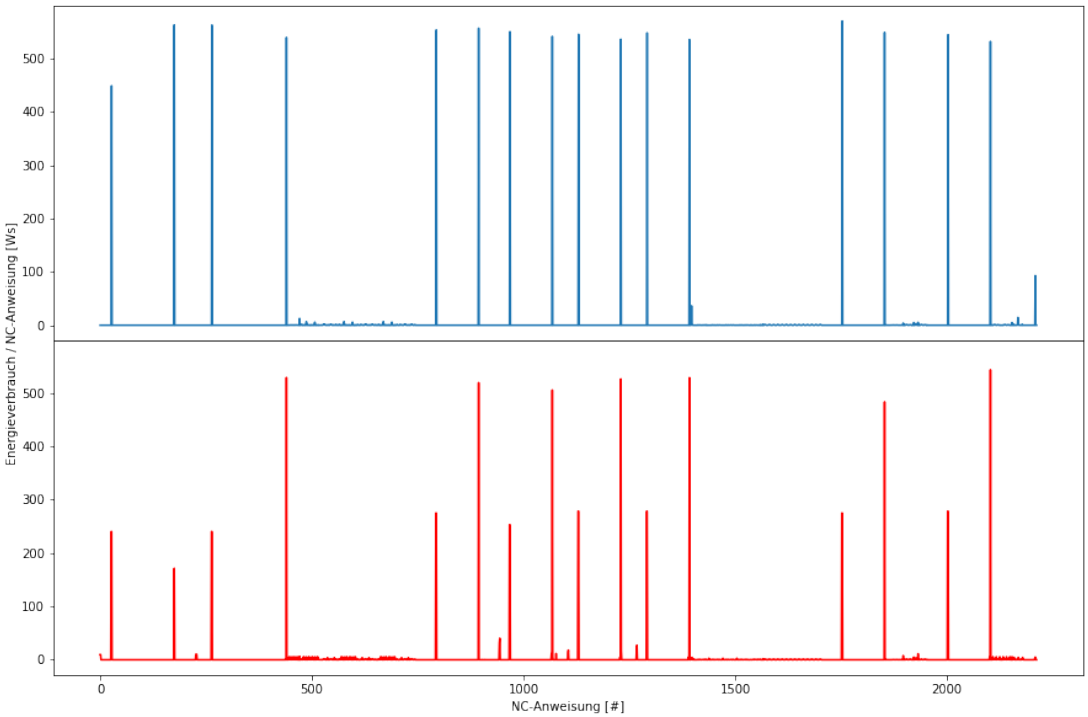
```
plt.xlabel("NC-Anweisung [#]")
fig.text(0.08, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')

plt.subplots_adjust(hspace=0)
```



1.6.2 X-Achse

[21]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,10))

line1 = y2['ENERGY|x'].plot(ax=axs[0])
line2 = df_pred[0].plot(ax=axs[1], color='r')

plt.xlabel("NC-Anweisung [#]")
fig.text(0.09, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')

plt.subplots_adjust(hspace=0)
```

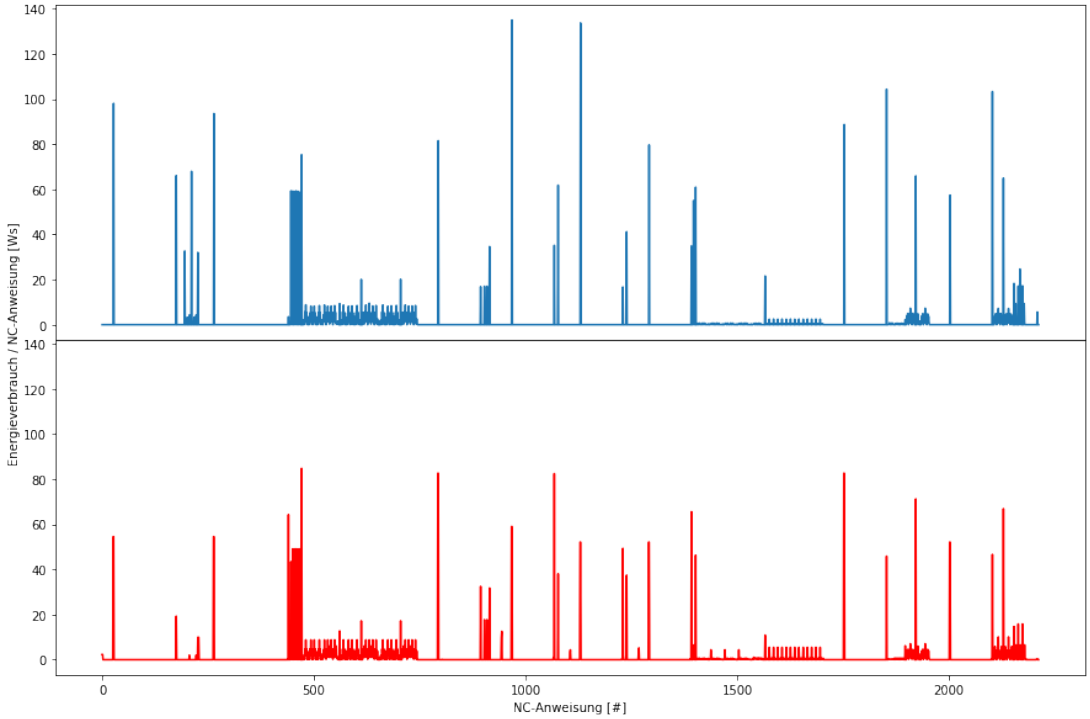
```
#y2['POWER|x'].plot(figsize=(15,6))
#df_pred[0].plot(figsize=(15,6)) #, ax = ax1)
```



[22]:
```
#df_pred[0].plot(figsize=(15,6))
```

1.6.3 y - Achse

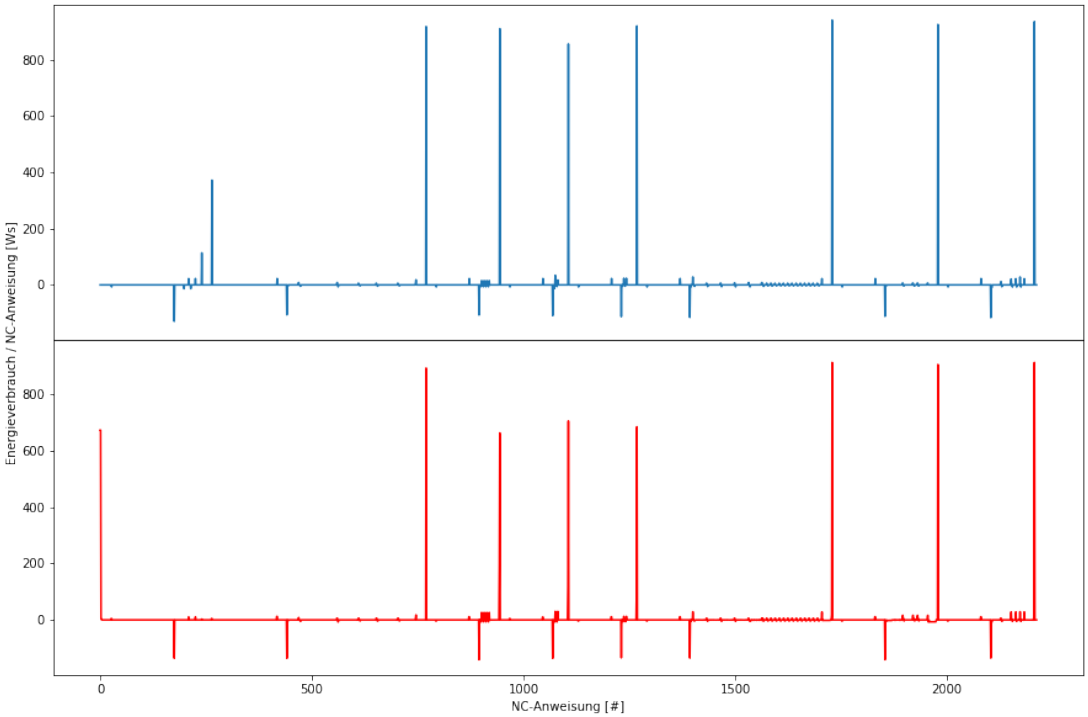
[23]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,10))

y2['ENERGY|y'].plot(ax=axs[0])
df_pred[1].plot(ax=axs[1], color='r')

plt.xlabel("NC-Anweisung [#]")
fig.text(0.09, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')

plt.subplots_adjust(hspace=0)
#axs[1].invert_yaxis()

#y2['POWER|y'].plot(figsize=(15,6))
```



[24]:
```
#df_pred[1].plot(figsize=(15,6))
```

1.6.4 z Achse

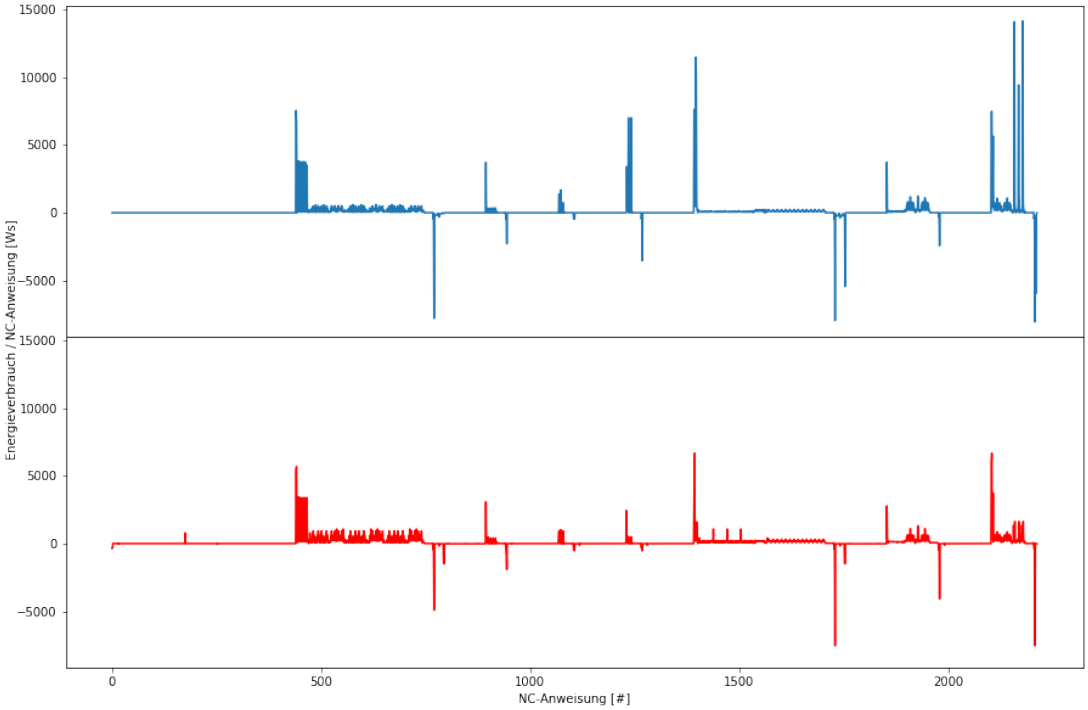
[25]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,10))

y2['ENERGY|z'].plot(ax=axs[0])
df_pred[2].plot(ax=axs[1], color='r')

plt.xlabel("NC-Anweisung [#]")
fig.text(0.09, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')

plt.subplots_adjust(hspace=0)

#y['POWER|z'].plot(figsize=(15,6))
```



[26]:
```
#df_pred[2].plot(figsize=(15,6))
```

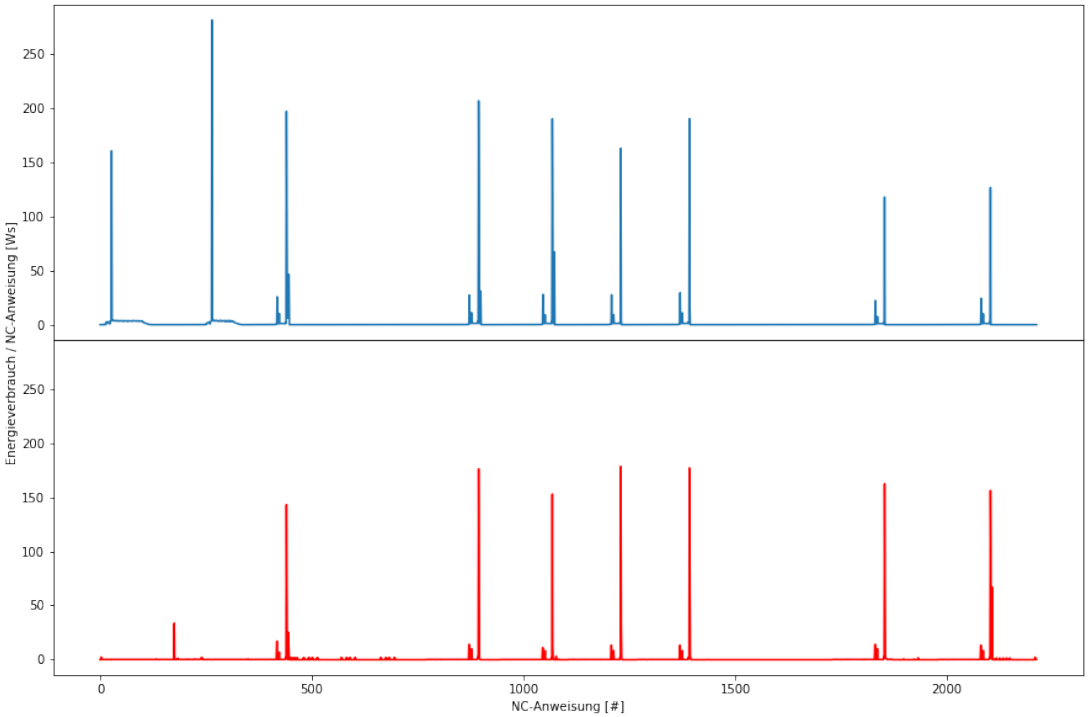
1.6.5 Spindel

[27]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,10))

y2['ENERGY|S'].plot(ax=axs[0])
df_pred[3].plot(ax=axs[1], color='r')
#axs[1].invert_yaxis()

plt.xlabel("NC-Anweisung [#]")
fig.text(0.08, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')

plt.subplots_adjust(hspace=0)
#y['POWER|S'].plot(figsize=(15,6))
```



1.6.6 Toolchange

[28]:
```
fig, axs = plt.subplots(2, sharex=True, sharey=True, figsize=(15,10))

y2['ENERGY|T'].plot(ax=axs[0])
#df_pred[4].plot(ax=axs[1], color='r')

#axs[1].invert_yaxis()
df_pred[4].plot(ax=axs[1], color='r')
#axs[1].invert_yaxis()

plt.xlabel("NC-Anweisung [#]")
fig.text(0.09, 0.5, 'Energieverbrauch / NC-Anweisung [Ws]', va='center',␣
 ↪rotation='vertical')


plt.subplots_adjust(hspace=0)
#y['POWER|T'].plot(figsize=(15,6))
```



1.7 Vorhersagegüte Quantifizieren

[29]:
```
# Explained variance score

from sklearn.metrics import explained_variance_score

y_true = y2
y_pred = df_pred

explained_variance_score(y_true, y_pred,  multioutput= 'raw_values')
```

[29]:
```
array([0.71104959, 0.53213094, 0.79234468, 0.42165383, 0.54198162])
```

[30]:
```
# mean absolut error

from sklearn.metrics import mean_absolute_error

y_true = y2
y_pred = df_pred

mean_absolute_error(y_true, y_pred, multioutput='raw_values')
```

[30]:
```
array([ 2.55494353,  0.83451496,  1.84957815, 98.39016246,  1.1629678 ])
```

[31]:
```
# root mean square error (RMS)

from sklearn.metrics import mean_squared_error
import math

y_true = y2
y_pred = df_pred

scores = mean_squared_error(y_true, y_pred, multioutput='raw_values')
#print(scores)

for i in range(len(scores)):
    scores[i] =  math.sqrt(scores[i])

print(scores)
```

```
[ 24.74521345   5.92735161  23.95258919 651.9594186    8.82653315]
```

[32]:
```
# Vergleich von Prototypen

from sklearn.metrics import mean_absolute_error
from math import sqrt, pow
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import r2_score
from math import sqrt

gesAbwArr = []
maeArr = []
rmsArr = []
exVarArr = []

scoresFrame = pd.DataFrame()

for j in range(len(y2.columns)):
    deviation = []
    for i in range(len(y2.index)):
        if y2.iloc[i, j] == 0 and df_pred.iloc[i, j] == 0:
            deviation.append(0)
        elif y2.iloc[i, j] == 0:
            deviation.append(0)
        else:
            deviation.append( ( y2.iloc[i, j] - df_pred.iloc[i, j]) / y2.
 ↪iloc[i, j])
    scoresFrame.insert(j, str(j), deviation)
```

```
    #scoresFrame[str(j)] = deviation

    mae = round(mean_absolute_error(y2.iloc[:,j], df_pred.iloc[:,j]), 2)

    exVar = round(explained_variance_score(y2.iloc[:,j], df_pred.iloc[:,j]), 2)
    r2 = round(r2_score(y2.iloc[:,j], df_pred.iloc[:,j]), 2)


    rms = round(sqrt(mean_squared_error(y_true.iloc[:,j], y_pred.iloc[:,j])), 2)

    sum_deviation = 0
    for i in range(len(scoresFrame)):
        sum_deviation += abs(scoresFrame.iloc[i, j])  #Fehler?
    mean_deviation = sum_deviation / len(scoresFrame)
    mre = round(mean_deviation * 100, 2)
    mre2 = round(mae/(y2.iloc[:,j].mean())*100, 2)
    gesAbw = round((((y2.iloc[:, j].sum() / df_pred.iloc[:, j].sum())-1)*100),␣
↪2)

    gesAbwArr.append(gesAbw)
    maeArr.append(mae)
    rmsArr.append(rms)
    exVarArr.append(exVar)

    print(f'\n------------------------------------ \n'
          f'{y2.columns[j]}\n'
          f'Gesammtabweichung {gesAbw}%\n'
          f'mean absolut error: {mae} Ws. \n'
#          f'mean of deviations: \u00B1 {mre}% \n'
#          f'mae/mean_meassured: \u00B1 {mre2}%\n'
          f'RMSE: {rms}\n'
          f'Explained Variance: {exVar}\n'
#          f'R2: {r2}\n'
          f'-------------------------------------\n')

print(gesAbwArr)
print(maeArr)
print(rmsArr)
print(exVarArr)
print()

row = 10
print(y2.iloc[[row]])
print(df_pred.iloc[[row]])
print(scoresFrame.iloc[[row]])
```

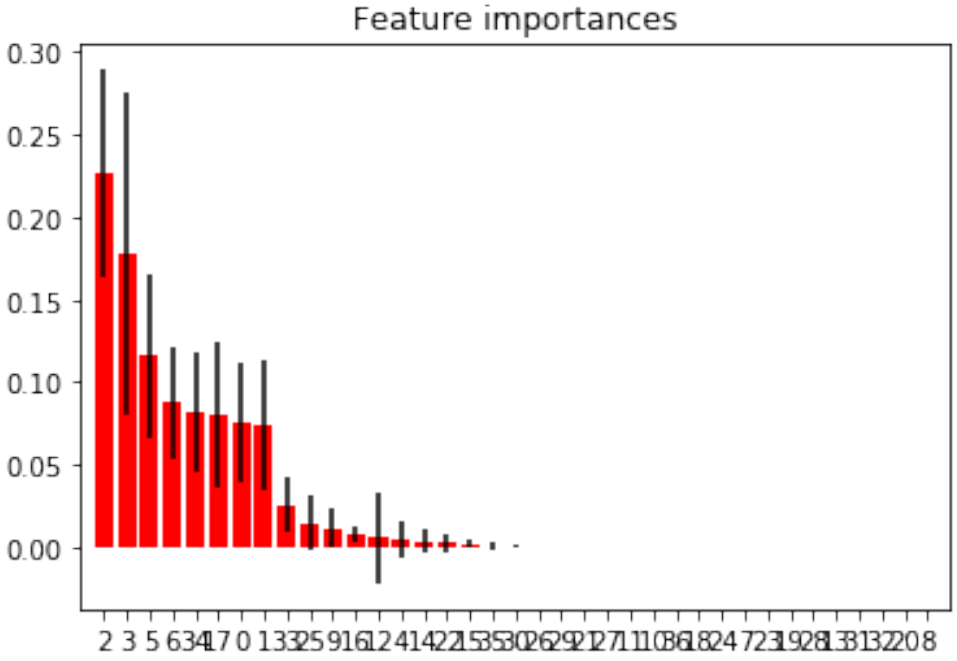
```
-----------------------------------
ENERGY|x
Gesammtabweichung 2.34%
mean absolut error: 2.55 Ws.
RMSE: 24.75
Explained Variance: 0.71
-----------------------------------


-----------------------------------
ENERGY|y
Gesammtabweichung -3.01%
mean absolut error: 0.83 Ws.
RMSE: 5.93
Explained Variance: 0.53
-----------------------------------


-----------------------------------
ENERGY|z
Gesammtabweichung -5.8%
mean absolut error: 1.85 Ws.
RMSE: 23.95
Explained Variance: 0.79
-----------------------------------


-----------------------------------
ENERGY|S
Gesammtabweichung 7.27%
mean absolut error: 98.39 Ws.
RMSE: 651.96
Explained Variance: 0.42
-----------------------------------


-----------------------------------
ENERGY|T
Gesammtabweichung 39.39%
mean absolut error: 1.16 Ws.
RMSE: 8.83
Explained Variance: 0.54
-----------------------------------

[2.34, -3.01, -5.8, 7.27, 39.39]
[2.55, 0.83, 1.85, 98.39, 1.16]
[24.75, 5.93, 23.95, 651.96, 8.83]
[0.71, 0.53, 0.79, 0.42, 0.54]
```

```
    ENERGY|x  ENERGY|y  ENERGY|z  ENERGY|S  ENERGY|T
10       0.0       0.0       0.0       0.0  0.085961
      0    1    2    3         4
10  0.0  0.0  0.0  0.0  0.204536
      0    1    2    3         4
10  0.0  0.0  0.0  0.0 -1.379398
```

[33]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier

importances = test_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in test_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. %s (%f)" % (f + 1, X.columns[f] , importances[indices[f]]))
#indices[f]

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
        color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. delta_X (0.226835)
2. delta_Y (0.177964)
3. delta_Z (0.115868)
4. delta_S (0.087829)
5. F_val (0.081884)
6. S (0.080315)
7. D_W (0.075461)
8. Toolchange (0.073734)
9. TurnOp (0.025971)
10. Commands_G0 (0.014804)
11. Commands_G0 G40 G60 (0.011718)
12. Commands_G0 M106 (0.007412)
```

13. Commands_G0 M3 (0.005894)
14. Commands_G09 (0.004572)
15. Commands_G1 (0.003862)
16. Commands_G1 G60 (0.002398)
17. Commands_G2 (0.002037)
18. Commands_G4 (0.000538)
19. Commands_G40 (0.000387)
20. Commands_G41 G1 (0.000241)
21. Commands_G4F1 (0.000127)
22. Commands_G54 G0 (0.000076)
23. Commands_G90 (0.000024)
24. Commands_G91 (0.000023)
25. Commands_G94 (0.000006)
26. Commands_G94 G1 G90 (0.000006)
27. Commands_M168 (0.000006)
28. Commands_M169 M167 (0.000003)
29. Commands_M17 (0.000002)
30. Commands_M27 M28 (0.000001)
31. Commands_M5 (0.000001)
32. Commands_M58, (0.000000)
33. Commands_M59 (0.000000)
34. Commands_MSG (0.000000)
35. D_D0 (0.000000)
36. D_D1 (0.000000)
37. D_D=$P_TOOL (0.000000)



[34]:
```
for f in range(X.shape[1]):
    print("%s" % (X.columns[f]))
```

```
delta_X
delta_Y
delta_Z
delta_S
F_val
S
D_W
Toolchange
TurnOp
Commands_G0
Commands_G0 G40 G60
Commands_G0 M106
Commands_G0 M3
Commands_G09
Commands_G1
Commands_G1 G60
Commands_G2
Commands_G4
Commands_G40
Commands_G41 G1
Commands_G4F1
Commands_G54 G0
Commands_G90
Commands_G91
Commands_G94
Commands_G94 G1 G90
Commands_M168
Commands_M169 M167
Commands_M17
Commands_M27 M28
Commands_M5
Commands_M58,
Commands_M59
Commands_MSG
D_D0
D_D1
D_D=$P_TOOL
```

[35]:
```
X.columns[1]
```

[35]: 'delta_Y'