Semantic Integration Patterns for AI

Deliverable 2.3b
des Forschungsprojektes i-Twin

Christian Borgelt[1], Sebastian Baron[1] und Marleen Bahe[2]

[1] **Paris-Lodron-Universität Salzburg**
[2] **Salzburg Research Forschungsgesellschaft mbH**

1. März 2024

Das Projekt i-Twin wird gefördert vom BMK und von der FFG aus Mitteln des Programms ITK der Zukunft.

Inhaltsverzeichnis

1 Zusammenfassung

Publizierbare Version

Sollen im Rahmen eines Fertigungsbetriebes analytische Services zur Überwachung oder Optimierung des Produktionsprozesses in die bestehende Produktionsumgebung integriert werden, ergeben sich verschiedene Problemstellungen. In der Regel handelt es sich bei den auf den Steuereinheiten anfallenden Sensordaten um sehr hochauflösende Zeitreihen, welche häufig aufgrund beschränkter Ressourcen in voller Auflösung nicht übertragen werden können, bzw. ist eine derartig hohe Abtastrate für die nachgelagerten analytischen Services auch gar nicht erforderlich. Häufig werden von solchen Services nicht die vollständigen rohen Sensordaten benötigt, sondern bestimmte Merkmale, welche aus den Rohdaten berechnet werden können. Die Definition, Berechnung und Bereitstellung solcher Merkmale sollte idealerweise von einem zentralen Modul übernommen werden, welches Zugriff auf die Rohdaten der Produktionsmaschinen hat. Handelt es sich bei den analytischen Services um Machine-Learning-Anwendungen, ist eine einfache Übertragbarkeit zuvor trainierter Modelle ebenfalls wünschenswert. Dieser Beitrag stellt Lösungsansätze der genannten Problemstellungen in Form von Methoden zur effizienten Sensordatenkomprimierung, einem zentralisierten Modul zur Merkmalserzeugung und -verwaltung sowie Rahmenwerken zur Übertragung erzeugter Modelle auf andere Laufzeitumgebungen vor.

i-Twin

i-Twin erforscht Interoperabilitätskonzepte für daten-getriebene digitale Zwillinge in der Fertigungsindustrie. Das Projekt propagiert eine Open-Source-Middleware für die Integration von Fertigungs-IT-Systemen und vernetzten Anlagen auf der Grundlage von Semantic Integration Patterns. Das vorrangige Ziel von i-Twin ist es, den Integrationsaufwand zu reduzieren und den Austausch von Stamm- und Betriebsdaten in Fertigungsnetzwerken zu ermöglichen. Die Ergebnisse werden in einem Forschungslabor und in einem industriellen Asset-Management-Szenario validiert. Das Projektkonsortium unter der Leitung der Salzburg Research verbindet die Forschungsinteressen von drei Systemanbietern (H&H Systems: CMMS, COPA-DATA: OT Software Plattform, IcoSense: Edge-Nodes) und eines Industrieunternehmens (INNIO Jenbacher: diskrete Fertigung) mit der Expertise der beteiligten Forschungspartner (Universität Salzburg: Data Science, Salzburg Research: Motion Data Intelligence). Das Projekt i-Twin wird gefördert vom BMK (Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie) und von der FFG (Österreichische Forschungsförderungsgesellschaft mbH) aus Mitteln des Programms IKT der Zukunft.

2 Überblick

Im Rahmen dieses Deliverable D2.3b werden drei Hauptaspekte behandelt, die im folgenden kurz charakterisiert werden, bevor in den folgenden Abschnitten Details vorgestellt werden.

2.1 Sensordatenkomprimierung

In der Praxis tritt sehr oft die Situation auf, dass die Sensordaten, die ggf. zu analysieren sind, auf einer Steuereinheit anfallen. Diese Steuereinheit hat aber vorrangig die Aufgabe, einen Prozess zu steuern und muss daher die Datenaufzeichnung und Übertragung als untergeordneten Prozess behandeln. Durch Bandbreitenbeschränkung des Übertragungskanals oder mangelnde Rechenleistung der Steuereinheit ist es dann oft nicht möglich, die Sensordaten mit der vollen zeitlichen Auflösung an einen anderen Rechner zu übertragen, der diese Daten sammelt und einer Analyse zuführt, was zu einem Verlust wertvoller, oft sogar entscheidender Information führt. Um mit diesem Problem umzugehen, wird ein sowohl bzgl. Rechenzeit als auch Speicherbedarf effizientes Verfahren benötigt, das mitlaufend (“*online*”) die Sensordaten so komprimiert, dass trotz einer geringeren Abstastung des eigentlichen Signals möglichst wenig Information verlorengeht.

2.2 Merkmalsberechnung und Verfügbarmachen berechneter Merkmale

Gerade dann, wenn Zeitreihendaten analysiert werden, aber auch für Daten anderen Charakters, besteht eine zentrale Problemstellung in der Aufbereitung, Vorverarbeitung und Aggregation der Rohdaten, welche von Sensoren und Logbuchprozessen auf Industriemaschinen erzeugt werden. Zwar könnte man diese Vorverarbeitung auch für jeden Einzelfall im zu erzeugenden Modell durchführen, doch führt dies zu unnötigem, weil redundantem Aufwand, da Datenanalysemodelle oft ähnliche Aufbereitungen, Transformationen und Aggregationen erfordern. Es ist daher wünschenswert, diese Aufbereitungen und Aggregationen in ein getrenntes, konfigurierbares Modul, nämlich einen Merkmalsspeicher (“*feature store*”) auszulagern. Merkmalsspeicher übernehmen in automatisierter und zentralisierter Weise die Merkmalserzeugung für eine oder mehrere Machine-Learning- oder Data-Analytics-Anwendungen. Sie sorgen dafür, dass Merkmale einfach definiert und konsistent berechnet werden, auch wenn Änderungen im datengenerierenden Prozess auftreten (beispielsweise eine veränderte Abtastrate).

2.3 Ausführungsmodell und -format für die erzeugten Modelle

Wegen der oft hohen Anforderungen an Rechenleistung und Speicherausstattung wird der Modellierungsschritt (siehe CRISP-DM-Modell [Chapman *et al.* 1999]) der Datenanalyse (“*Data Mining*”) meist auf Servern oder leistungsfähigen Workstations durchgeführt. Hier stehen gewöhnlich spezielle Entwicklungsumgebungen (z.B. Knime oder RapidMiner) oder Programmiersprachen (z.B. Python oder R) zur Verfügung, die die dem Datenanalysten seine Aufgabe sehr erleichtern, indem sie die Vorverarbeitung, die explorative Datenanalyse speziell durch Datenvisualisierung und die Modellerzeugung mit mächtigen Werkzeugen unterstützen. Diese Umgebungen sind jedoch meist nicht geeignet, “*on the edge*”, also auf dem späteren Einsatzrechner für das erzeugte Modell (“*deployment*” im CRISP-DM-Modell, [Chapman *et al.* 1999]) installiert zu werden. Hier stehen spezielle Modellformate zur Verfügung, mit denen erzeugte Datenanalysemodelle in standardisierter Form, ggf. mit zusätzlichen Sicherungen gegen Laufzeitfehler, in ressourcensparender Weise in speziellen Laufzeitumgebungen ausgeführt werden können. Beispiele für solche Formate sind PMML (Predictive Model Markup Language), PFA (Portable Format for Analytics) und ONNX (Open Neural Network eXchange).

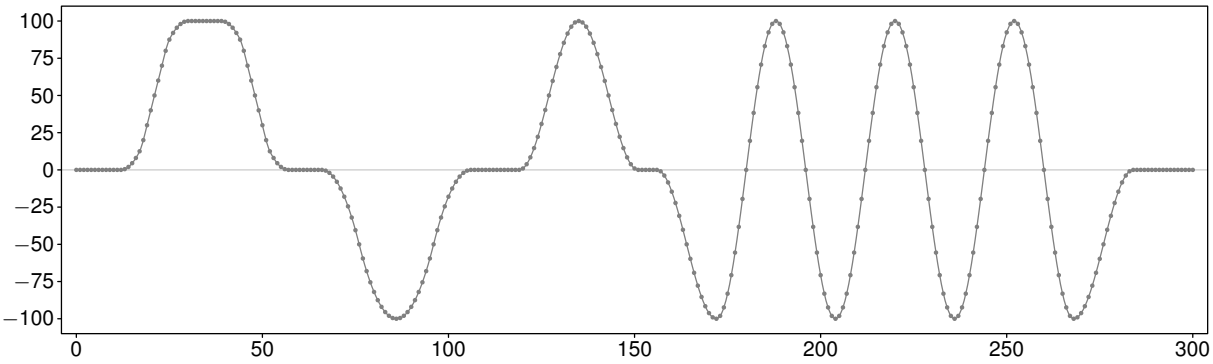

Abbildung 1: Fiktive Sensordaten-Zeitreihe.

3 Sensordatenkomprimierung

In der Praxis tritt sehr oft die Situation auf, dass die z.B. auf einer Maschine aufgenommenen Sensordaten, die ggf. zu analysieren sind, auf einer Steuereinheit anfallen. Diese Steuereinheit hat aber, wie ihr Name schon sagt, vorrangig die Aufgabe, einen Prozess (z.B. die Bearbeitung eines Werkstücks) zu steuern und muss daher die Datenaufzeichnung und Übertragung als untergeordneten Prozess behandeln. Durch Bandbreitenbeschränkung des Übertragungskanals oder wegen mangelnder Rechenleistung der Steuereinheit ist es dann oft nicht möglich, die Sensordaten mit der vollen zeitlichen Auflösung, mit der sie auf der Steuereinheit vorliegen, an einen anderen Rechner zu übertragen, der diese Daten sammelt und einer Analyse zuführt. Dies führt zu einem Verlust wertvoller, oft sogar entscheidender Information. Um mit diesem Problem umzugehen, werden sowohl bzgl. Rechenzeit als auch Speicherbedarf effiziente Verfahren benötigt, die mitlaufend (“*online*”) die Sensordaten so komprimieren, dass trotz einer geringeren Abstastung des ursprünglichen Signals möglichst wenig Information verlorengeht. In den folgenden Abschnitten werden daher prototypisch einfache Verfahren vorgestellt, mit denen eine sowohl bzgl. Laufzeit als auch Speicherbedarf effiziente Komprimierung von Sensordaten-Zeitreihen bei erträglichem (und über Parameter im Rahmen der möglichen Übertragungsdichte steuerbarem) Genauigkeitsverlust vorgenommen werden kann.

3.1 Problemstellung

Das durch Sensordatenkomprimierung zu lösende Problem soll hier anhand der in Abbildung 1 gezeigten fiktiven eindimensionalen (also nur eine Messgröße erfassenden) Sensordaten-Zeitreihe illustriert werden. Diese Zeitreihe besteht aus 301 äquidistanten Messpunkten (Indizes 0 bis 300 auf der horizontalen Achse) mit Messwerten aus dem Intervall $[-100, 100]$ (auf der vertikalen Achse). Wir nehmen an, dass diese Zeitreihe auf einer Steuereinheit gemessen wird, es aber nicht möglich ist, sie mit voller zeitlicher Auflösung an einen anderen Rechner (z.B. den Rechner, auf dem eine Analyse dieser Daten vorgenommen werden soll) zu übertragen (z.B. weil die Bandbreite der Übertragungsstrecke nicht ausreicht). Stattdessen wollen wir annehmen, dass eine Übertragung von nur etwa einem Zehntel der Messpunkte möglich ist. Diese übertragenen Messpunkte müssen nicht unbedingt äquidistant sein, sollten aber eine gewisse lokale Dichte nicht übersteigen (die in der Anwendung von der konkreten Steuereinheit, ihrer Rechenleistung und der Bandbreite des Übertragungskanals abhängt).

Ein sehr einfacher Ansatz, die Messdaten näherungsweise zu übertragen, ist offenbar, nur jeden k-ten Datenpunkt, $k > 1$, zu übertragen. Dadurch wird das Signal, das durch den Sensor gemessen wird, natürlich gröber abgetastet. Aber da das Signal keine allzu schnellen Schwankungen zeigt, könnte man sinnvoll erwarten, dass der prinzipielle Verlauf des Signals noch

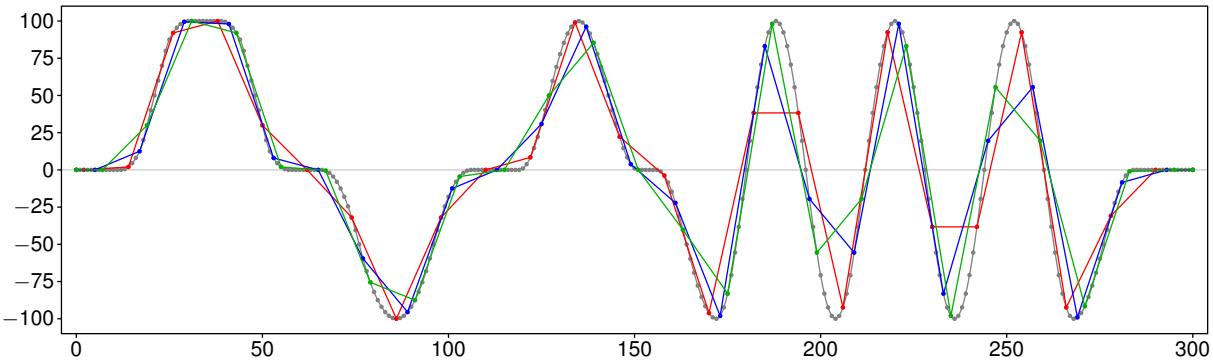

Abbildung 2: Drei Abtastungen des Signals aus Abbildung 1, bei denen nur jeder 12. Messpunkt übertragen wird, die aber verschiedene Phase haben.

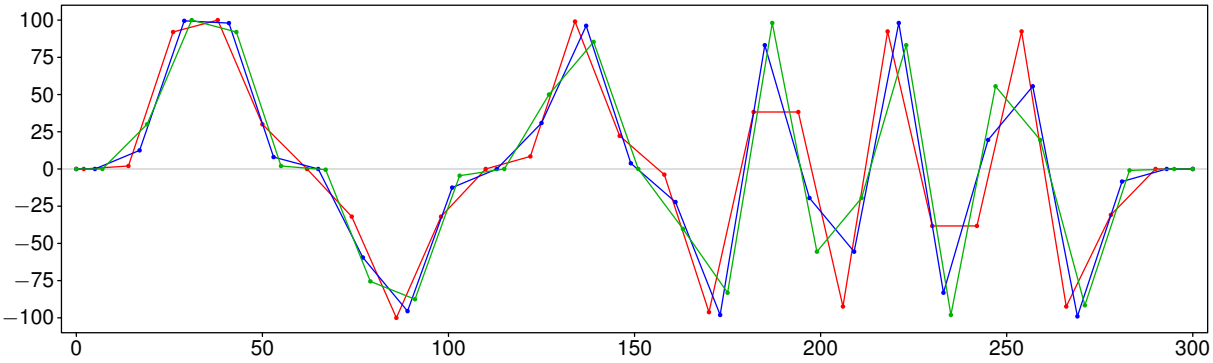

Abbildung 3: Drei Abtastungen des Signals aus Abbildung 1, bei denen nur jeder 12. Messpunkt übertragen wird, aber ohne das Originalsignal, was die Unterschiede deutlich hervortreten läßt.

erkennbar bleibt, wenn die Abtastschrittweite k nicht zu groß gewählt wird.

Wie die genaue Abtastung aussieht, hängt jedoch nicht nur von der Abtastschrittweite, sondern auch von ihrer Phase relativ zum Signal ab. Wenn es keinen “natürlichen” Startpunkt gibt (was in der Praxis schon wegen Messungenauigkeiten, aber auch wegen Variationen im Prozessverlauf meist der Fall ist), können recht unterschiedliche komprimierte Signale entstehen. Dies ist in Abbildung 2 gezeigt, in der vor dem Hintergrund der Originalabtastung in rot, blau und grün drei gröbere Abtastungen gezeigt sind, die alle drei jeden 12. Messpunkt übertragen, aber zu unterschiedlichen Zeitpunkten einsetzen. Obwohl alle drei Abtastungen durch gleichartige Aussonderungen von Messpunkten entstanden sind, ist der Verlauf dieser drei Abtastungen recht deutlich verschieden, besonders im Bereich ab ca. Index 175, ab dem sich das Originalsignal etwas schneller ändert. Dies ist noch deutlicher in Abbildung 3, in der das Originalsignal fehlt, so dass die Unterschiede besonders deutlich hervortreten.

Durch eine einfache äquidistante Abstastung (d.h. mit konstanter Schrittweite k) kann folglich das gleiche Signal nach der Übertragung sehr verschieden aussehen, sogar dann, wenn der grobe Signalverlauf noch einigermaßen ähnlich ist. Hat man die Situation, dass eine Maschine immer wieder den gleichen Prozess durchläuft (z.B. weil eine Serie gleichartiger Werkstücke bearbeitet wird), so ist dies sehr nachteilig, da dann eigentlich ähnliche, ggf. sogar exakt gleiche Prozessverläufe durch leichte Verschiedenheit der Phase der Abtastung nach der Übertragung als sehr unterschiedlich erscheinen können. Dies erschwert eine Überwachung des Produktionsprozesses, bei der z.B. anomales Verhalten erkannt werden soll, erheblich: Allein durch die verschiedene Abtastung erscheinen die übertragenen Signale trotz gleichen Quellsignals als recht stark schwankend, so dass nur sehr starke Abweichungen von einem

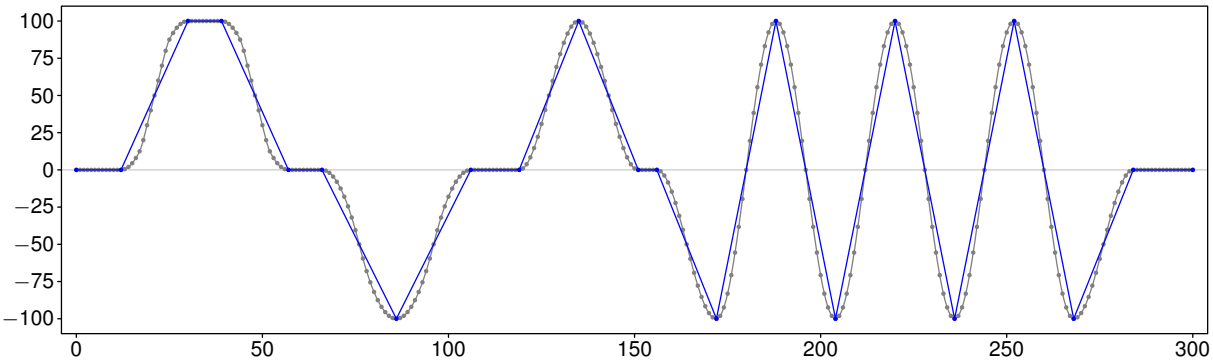

Abbildung 4: Eine Abtastung des Signals aus Abbildung 1 mit variabler Schrittweite, bei der speziell Extremwerte und Punkte übertragen werden, an denen sich der Signalverlauf ändert.

normalen Verhalten als Anomalie erkannt werden können. Kleinere Abweichungen könnten ihren Grund schon allein in der durch die verschiedene Phase der Abtastung erzeugten Variation haben und können daher nicht — zumindest nicht statistisch verlässlich — identifiziert werden.

Eine mögliche Lösung des aufgezeigten Problems besteht darin, die Abtastschrittweite variabel zu gestalten und speziell Messpunkte zu übertragen, die Extremwerte des Signals (lokale Minima und Maxima) oder Punkte darstellen, an denen sich der Signalverlauf (deutlich) ändert. Für das obige Beispiel (siehe Abbildung 1) könnte eine solche Abtastung zu einer Übertragung des in Abbildung 4 gezeigten komprimierten Signals führen. Obwohl in diesem Beispiel nur insgesamt 21 Punkte übertragen werden, während bei den in den Abbildungen 2 und 3 gezeigten Abtastungen jeweils 25 Punkte übertragen wurden, wird das Originalsignal wesentlich besser wiedergegeben. Da sich die Abtastung an hervorstechenden Messpunkten orientiert (Extremwerte und Punkte an denen sich der Signalverlauf signifikant ändert), ist außerdem mit einer wesentlich geringeren Variation des übertragenen Signals für nur geringfügig verschiedene Originalsignale zu rechnen. Dadurch sollte es möglich werden, anomales Verhaltens auch bei deutlich kleineren Abweichungen vom normalen Prozessverlauf zu erkennen.

Das zu lösende Problem kann folglich zusammenfassend so beschrieben werden: Um ein Signal durch eine Aussonderung der Messpunkte komprimiert unter möglichst geringem Informationsverlust zu übertragen, sollte der Sender der Daten speziell auffällige Messpunkte wie Extremwerte und Punkte, an denen sich der Signalverlauf signifikant ändert, für eine Übertragung auswählen. Da diese Komprimierung auf einer Steuereinheit geschehen muss, die oft über nur begrenzte Rechenleistung und Speicherkapazität verfügt, müssen die zu übertragenden Messpunkte möglichst berechnungs- und speichereffizient ausgewählt werden können.

3.2 Mitlaufende Komprimierung

Um Messpunkte in der gewünschten Weise auszuwählen, erscheint es auf den ersten Blick am natürlichsten, die Messdaten in Abschnitten oder Zeitfenstern (von z.B. 50 oder 100 Messpunkten) zwischenzuspeichern und diese Abschnitte auf Extremwerte und Änderungspunkte zu untersuchen. Dies hätte den Vorteil, dass zumindest für die inneren Punkte des Zeitfensters sowohl Vorgänger als auch Nachfolger bekannt sind (für die meisten Punkte sogar mehrere Vorgänger und Nachfolger), was die Analyse und Messpunktauswahl erheblich vereinfacht. Bei einer solchen Zwischenspeicherung könnten auch mehrere alternative Auswahlen von Messpunkten erstellt und bezüglich eines Fehlerkriteriums verglichen werden, so dass eine bzgl. eines solchen Fehlerkriteriums beste Auswahl von Messpunkten gefunden werden kann.

Wegen der meist begrenzten Speicherausstattung der Steuereinheit, die die Auswahl der zu übertragenden Messpunkte vornehmen muss, verbietet sich jedoch i.a. die Speicherung eines (längeren) Abschnitts des Originalsignals. Meist können nur einige wenige Messpunkte oder aus den Messpunkten berechnete Größen gespeichert werden. Außerdem ist die Analyse eines zwischengespeicherten Zeitfensters ggf. recht aufwendig und kann schnell die Rechenzeit überschreiten, die auf die Messpunktauswahl aufgewendet werden kann.

Idealerweise sollte die gesuchte Messpunktauswahl daher mitlaufend (“*online*”) berechnet werden können. Das bedeutet, dass mit jedem neuen Messpunkt, der verfügbar wird, unmittelbar entschieden wird, ob dieser Punkt (oder ein einzelner, für eine mögliche spätere Übertragung zurückgehaltener Vorgängerpunkt) übertragen werden sollte oder nicht. In diesem Fall wären nur sehr wenige Messpunkte zwischenzuspeichern (in den im folgenden betrachteten Ansätzen sind zu jedem Zeitpunkt der Verarbeitung sogar nur höchstens drei Messpunkte verfügbar, nur zwei werden zwischengespeichert) und ggf. einige (wenige) aus den vorangehenden Messpunkten berechnete Aggregate, die zur Entscheidungsfindung dienen.

Der folgende Abschnitt beschreibt einfache Algorithmen für eine solche mitlaufende (“*online*”) Komprimierung, die auf der Annäherung des Originalsignals durch einen Polygonzug beruhen. Dies ist eine der einfachsten und am effizientesten zu berechnenden Formen der Signalkomprimierung. Komplexere Näherungen, z.B. durch stückweise quadratische Funktionen oder (kubische) Splines, könnten zwar ebenfalls in Betracht gezogen werden, doch rechtfertigt die erzielbare Genauigkeitsverbesserung nicht den erhöhten Speicher- und Rechenaufwand.

3.3 Annäherung durch Polygonzug

Die Grundidee der mitlaufend zu berechnenden Komprimierung eines (abgetasteten) Signals durch einen Polygonzug besteht darin, unter Zuhilfenahme einiger weniger Hilfsgrößen für jeden neuen Messpunkt zu prüfen, wie gut er durch eine Erweiterung einer linearen Näherung des aktuellen Abschnitts erfasst werden kann. Die Güte dieser Erfassung des neuen Punktes wird durch einen Schwellenwert für ein Fehlermaß bestimmt, wobei im folgenden sechs verschiedene Fehlermaße (genauer: zwei Gruppen zu vier und zwei Maßen) betrachtet werden. Bleibt ein solches Fehlermaß der Punkte des aktuellen Abschnitts bei Einbeziehung des nächsten Punktes nicht unter dem benutzerspezifizierten Schwellenwert, so wird der (zu diesem Zweck gemerkte) Vorgängerpunkt übertragen und ein neuer linearer Abschnitt begonnen.

Zusätzlich sollten Prüfungen auf Extremwerte (lokale Minima und Maxima) vorgenommen werden, um die lineare Näherung an solchen Extrempunkten auch dann zu beenden (und einen neuen linearen Abschnitt zu beginnen), wenn die Erweiterung um den nächsten, schon jenseits eines lokalen Minimums oder Maximums liegenden Punkt noch möglich wäre, ohne die benutzerspezifizierte Fehlerschranke zu überschreiten. Als letzter Bestandteil des Verfahrens erfordert ein (fast) horizontaler Signalverlauf eine Sonderbehandlung.

Zur Ableitung der nötigen Berechnungsformeln rekapitulieren wir zunächst die Grundlagen der univariaten linearen Regression, auf vier der folgenden Verfahren aufbauen (die zwei übrigen verwenden einen einfacheren Näherungsansatz). Bei der univariaten linearen Regression ist ein Datensatz $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ von n Datenpunkten gegeben, die jeweils aus einem Wert x_i der unabhängigen Variable und einem Wert y_i der abhängigen Variable bestehen, $i = 1, \ldots, n$. Die Hypothese über die funktionale Abhängigkeit der abhängigen von der unabhängigen Variable lautet $Y = f(X) = a + bX + \epsilon$. Hier ist X eine Zufallsvariable, die die Verteilung der unabhängigen Variable beschreibt und ϵ ein zufälliger Störterm, der den Erwartungswert 0 besitzt. Die Parameter a und b beschreiben die lineare Abhängigkeit.

Ziel der univariaten linearen Regression ist es, die unbekannten Parameter a und b so zu bestimmen, dass die gegebenen Datenpunkte durch die Funktion $Y = f(X) = a + bX$ möglichst gut angenähert werden. Als Fehlerkriterium wird dabei üblicherweise die Summe der quadrierten Abweichungen zwischen den tatsächlichen Zielwerten y_i und den aus den zugehörigen Werten x_i mit Hilfe der Parameter a und b berechneten hypothetischen Zielwerten $\tilde{y}_i = a + bx_i$ verwendet. Diese Fehlerquadratsumme, d.h., das Fehlerfunktional

$$E(a, b) = \sum_{i=1}^{n} (a + bx_i - y_i)^2,$$

ist durch eine geeignete Wahl der Parameter a und b zu minimieren.

Eine Lösung dieses Optimierungsproblems läßt sich mit dem klassischen Ansatz bestimmen, der ausnutzt, dass notwendige Bedingungen für ein Minimum einer Funktion sind, dass die partiellen Ableitungen nach den Parametern der Funktion verschwinden.[1] Dies liefert:

$$\begin{aligned} \frac{\partial F}{\partial a} &= 2 \sum_{i=1}^{n} (a + bx_i - y_i) &&\stackrel{!}{=} 0, \\ \frac{\partial F}{\partial b} &= 2 \sum_{i=1}^{n} (a + bx_i - y_i) \cdot x_i &&\stackrel{!}{=} 0. \end{aligned}$$

[1] Es handelt sich um notwendige, aber nicht hinreichende Bedingungen, weil auch an einem Maximum oder einem Sattelpunkt der Funktion die partiellen Ableitungen verschwinden.

Aus diesen Bedingungen erhält man das System der sogenannten *Normalgleichungen*:

$$\begin{aligned} an &+ b\sum_{i=1}^{n} x_i &&= \sum_{i=1}^{n} y_i \\ a\sum_{i=1}^{n} x_i &+ b\sum_{i=1}^{n} x_i^2 &&= \sum_{i=1}^{n} x_i y_i \end{aligned}$$

Mit den folgenden Abkürzungen für die auftretenden Summen:

$$s_x = \sum_{i=1}^{n} x_i, \qquad s_y = \sum_{i=1}^{n} y_i, \qquad s_{xx} = \sum_{i=1}^{n} x_i^2, \qquad s_{yy} = \sum_{i=1}^{n} y_i^2, \qquad s_{xy} = \sum_{i=1}^{n} x_i y_i,$$

die sich aus den gegebenen Daten unmittelbar berechnen lassen, können die Normalgleichungen vereinfacht geschrieben werden als:

$$\begin{aligned} a\,n &+ b\,s_x &&= s_y, \\ a\,s_x &+ b\,s_{xx} &&= s_{xy}. \end{aligned}$$

Im univariaten linearen Fall sind die Normalgleichungen also ein lineares Gleichungssystem mit zwei Gleichungen und zwei Unbekannten (nämlich a und b). Wir schreiben dieses Gleichungssystem in Matrix-Vektor-Schreibweise als

$$\mathbf{C} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \vec{r} \qquad \text{mit} \qquad \mathbf{C} = \begin{pmatrix} n & s_x \\ s_x & s_{xx} \end{pmatrix} \qquad \text{und} \qquad \vec{r} = \begin{pmatrix} s_y \\ s_{xy} \end{pmatrix}.$$

Die *Cramersche Regel* erlaubt es, die Lösung dieses Gleichungssystems anzugeben als (die senkrechten Striche bezeichnen die Bildung der Determinante der eingeschlossenen Matrix):

$$a = \frac{|\mathbf{C}_a|}{|\mathbf{C}|} \qquad \text{und} \qquad b = \frac{|\mathbf{C}_b|}{|\mathbf{C}|},$$

(vorausgesetzt, es ist $|\mathbf{C}| \neq 0$, also $\mathbf{C}$ eine reguläre, d.h. invertierbare Matrix) wobei

$$\mathbf{C}_a = \begin{pmatrix} s_y & s_x \\ s_{xy} & s_{xx} \end{pmatrix} \qquad \text{und} \qquad \mathbf{C}_b = \begin{pmatrix} n & s_y \\ s_x & s_{xy} \end{pmatrix}.$$

Die sich ergebende Lösung kann einfacher geschrieben werden, indem man auf die (empirischen) Mittelwerte der beiden Koordinaten der Datenpunkte, also

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{s_x}{n} \qquad \text{und} \qquad \mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{s_y}{n},$$

und ihre (empirischen) Varianzen σ_{xx} und σ_{xx} sowie ihre (empirische) Kovarianz σ_{xy}, also

$$\sigma_{xx} = \frac{s_{xx} - n\mu_x^2}{n-1}, \qquad \sigma_{yy} = \frac{s_{yy} - n\mu_y^2}{n-1}, \qquad \sigma_{xy} = \frac{s_{xy} - n\mu_x\mu_y}{n-1}$$

zurückgreift.[2] Mit diesen Größen können die in der Cramerschen Regel auftretenden Determi-

[2] Varianzen und Kovarianz wurden hier mit der sogenannten Bessel-Korrektur, also einem Teilen durch $n-1$ berechnet, was jedoch für die folgende Rechnung keine Rolle spielt, da sich dieser Faktor (wie auch der Faktor $\frac{1}{n}$ aus den Mittelwertsformeln) in den Lösungsformeln für die Parameter a und b herauskürzt. Es könnte also in der Varianzberechnung genauso gut mit einem Faktor $\frac{1}{n}$ gerechnet werden — das Ergebnis ändert sich dadurch nicht.

nanten geschrieben werden als[3]

$$\begin{aligned}
|\mathbf{C}| &= n s_{xx} - s_x^2 &&= n s_{xx} - n^2 \mu_x^2 \\
&= n(s_{xx} - n\mu_x^2) &&= n(n-1)\sigma_{xx},
\end{aligned}$$

$$\begin{aligned}
|\mathbf{C}_b| &= n s_{xy} - s_x s_y &&= n s_{xy} - n^2 \mu_x \mu_y \\
&= n(s_{xy} - n\mu_x\mu_y) &&= n(n-1)\sigma_{xy},
\end{aligned}$$

$$\begin{aligned}
|\mathbf{C}_a| &= s_y s_{xx} - s_{xy} s_x &&= n\mu_x s_{xx} - n\mu_x s_{xy} \\
&= n\mu_y \left((n-1)\sigma_{xx} + n\mu_x^2\right) - n\mu_x \left((n-1)\sigma_{xy} + n\mu_x\mu_y\right) \\
&= n(n-1)\mu_y\sigma_{xx} + n^2\mu_x^2\mu_y - n(n-1)\mu_x\sigma_{xy} - n^2\mu_x^2\mu_y \\
&= n(n-1)(\mu_y\sigma_{xx} - \mu_x\sigma_{xy}).
\end{aligned}$$

Mit dieser Darstellung können die Lösungen (statistisch eigentlich: Schätzungen) für die beiden unbekannten Parameter a und b schließlich sehr bequem geschrieben werden als

$$\begin{aligned}
b &= \frac{n(n-1)\sigma_{xy}}{n(n-1)\sigma_{xx}} &&= \frac{\sigma_{xy}}{\sigma_{xx}} \qquad \text{und} \\
a &= \frac{n(n-1)(\mu_y\sigma_{xx} - \mu_x\sigma_{xy})}{n(n-1)\sigma_{xx}} &&= \mu_y - \frac{\sigma_{xy}}{\sigma_{xx}}\mu_x.
\end{aligned}$$

Für die folgenden Anwendungen ist es allerdings günstiger, statt der Varianzen und Kovarianzen die entsprechenden Summen der quadratischen Abweichungen bzw. der Produkte der Abweichungen von den Mittelwerten zu verwenden:

$$\delta_{xx} = (n-1)\sigma_{xx}, \qquad \delta_{yy} = (n-1)\sigma_{yy}, \qquad \delta_{xy} = (n-1)\sigma_{xy}.$$

Da sich die Faktoren $(n-1)$ wegkürzen, erhält man

$$a = \mu_y - \frac{\delta_{xy}}{\delta_{xx}}\mu_x \qquad \text{und} \qquad b = \frac{\delta_{xy}}{\delta_{xx}}.$$

Damit kann die Summe der Abweichungsquadrate geschrieben werden als[4]

$$E = E(a,b) = \sum_{i=1}^{n}(a + bx_i - y_i)^2 = \left(\delta_{yy} - \frac{\delta_{xy}^2}{\delta_{xx}}\right) = \frac{1}{\delta_{xx}}|\Delta|,$$

$$\text{wobei} \qquad |\Delta| = \begin{vmatrix} \delta_{xx} & \delta_{xy} \\ \delta_{xy} & \delta_{yy} \end{vmatrix} = (n-1)\begin{vmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{vmatrix} = (n-1)|\Sigma|$$

die Determinante der Matrix der Summe der quadratischen Abweichungen bzw. Produkte der Abweichungen von den Mittelwerten ist, die sich von der (empirischen) Kovarianzmatrix Σ nur um den Faktor $(n-1)$ unterscheidet.

Wie oben ausgeführt, soll ein weiterer Datenpunkt durch Ausweitung des aktuellen linearen Abschnitts erfasst werden, wenn die Güte der linearen Passung über einem von einem Benutzer spezifizierten Mindestwert liegt. Zum Messen der Güte ist die Abweichungs- oder Fehlerquadratsumme jedoch nicht gut geeignet, da sie (1) eine quadratische Größe ist, was die Wahl eines geeigneten Schwellenwertes erschwert und (2) als Summe über die Datenpunkte von der Anzahl der eingehenden Datenpunkte abhängt. Besser ist daher die Verwendung der Quadratwurzel aus dem mittleren quadratischen Fehler (*"root mean squared error"*), also

$$e_{\text{RMSE}} = \sqrt{\tfrac{1}{n}E} = \sqrt{\tfrac{1}{n}\left(\delta_{yy} - \frac{\delta_{xy}^2}{\delta_{xx}}\right)}.$$

[3] Die in diesen Umformungen verwendeten Beziehungen sind in Anhang A hergeleitet.

[4] Eine ausführliche Herleitung dieser Beziehung findet sich in Anhang A.

Für diesen Fehler gibt ein Benutzer einen Schwellenwert θ_{RMSE} vor. Wird dieser Schwellenwert durch das Einbeziehen des nächsten Datenpunktes überschritten, so wird der aktuelle lineare Abschnitt beendet und einer neuer begonnen. Zur mitlaufenden Berechnung des jeweiligen Fehlers (d.h., für jeden neuen Datenpunkt wird der Fehler berechnet, der sich durch Einbeziehen des neuen Datenpunktes relativ zu einer Regressionsgeraden ergibt) müssen gemäß der obigen Formel lediglich die Summen δ_{xx} und δ_{yy} der Abweichungsquadrate und die Summe δ_{xy} der Produkte der Abweichungen fortgeschrieben werden.

Auf den ersten Blick mag es so erscheinen, als ob zur Berechnung dieser Summen (δ_{xx}, δ_{yy} und δ_{xy}) die Datenpunkte des aktuellen Abschnitts abgespeichert werden müßten. Dies ist jedoch nicht der Fall, da mit Hilfe einer einfachen Erweiterung des Welford-Algorithmus[5] der Fehler nur unter Speicherung dieser Summen, der Anzahl der eingehenden Datenpunkte und der Mittelwerte μ_x und μ_y berechnet werden kann. Man geht dazu so vor:

1. Benutzervorgabe: Schwellenwert $\theta_{RMSE} \geq 0$.
2. Berechne $\theta_{\text{MSE}} = \theta^2_{RMSE}$ (um die Berechnung der Quadratwurzel zu vermeiden).
3. Initialisiere $i = 0$, $\mu_x = 0$, $\mu_y = 0$, $\delta_{xx} = 0$, $\delta_{yy} = 0$ und $\delta_{xy} = 0$.
4. Für jeden (neuen) Datenpunkt (x, y) berechne:

$$
\begin{aligned}
k &\leftarrow i \\
i &\leftarrow i + 1 \\
k &\leftarrow \tfrac{k}{i} \\
d_x &\leftarrow x - \mu_x \\
d_y &\leftarrow y - \mu_y \\
\mu_x &\leftarrow \mu_x + \tfrac{d_x}{i} \\
\mu_y &\leftarrow \mu_y + \tfrac{d_y}{i} \\
\delta_{xx} &\leftarrow \delta_{xx} + k d_x^2 \\
\delta_{yy} &\leftarrow \delta_{yy} + k d_y^2 \\
\delta_{xy} &\leftarrow \delta_{xy} + k d_x d_y \\
e_{\text{MSE}} &\leftarrow \tfrac{1}{i}\left(\delta_{yy} - \delta_{xy}^2/\delta_{xx}\right) \\
e_{\text{MSE}} &\leq \theta_{\text{MSE}}?
\end{aligned}
$$

Zum Fortschreiben des Fehlers müssen also lediglich sechs Variablen gespeichert werden:

- Die Anzahl der Datenpunkte in Form eines Index i
  (Index des letzten verarbeiteten Datenpunktes).
- Die Mittelwerte μ_x und μ_y für die bisher verarbeiteten Datenpunkte.
- Die Summen δ_{xx}, δ_{yy} und δ_{xy} für die bisher verarbeiteten Datenpunkte.

[5] Eine vollständige Herleitung des Welford-Algorithmus ist in Anhang B angegeben.

Hinzu tritt natürlich außerdem der vom Benutzer vorgegebene Schwellenwert, der jedoch als $\theta_{\text{MSE}} = \theta^2_{\text{RMSE}}$ abgelegt wird, um die recht teure Berechnung der Quadratwurzel vor dem Vergleich des aktuellen Fehlers mit dem Schwellenwert zu vermeiden. Da das Wurzelziehen eine montone Operation ist, bleiben Ordnungsrelationen erhalten, und es kann statt $e_{\text{RMSE}} > \theta_{\text{RMSE}}$ offenbar genauso gut $e_{\text{MSE}} > \theta_{\text{MSE}}$ getestet werden.

Zusätzlich werden für die Berechnung vier temporäre Variablen benötigt, die Zwischenergebnisse beinhalten, aber zwischen den Berechnungen für zwei aufeinanderfolgende Datenpunkte nicht gespeichert werden müssen:

- Ein Merker k für den alten Indexwert i, in dem anschließend der Bruch $\frac{i}{i+1}$ abgelegt wird.
- Zwei Variablen d_x und d_y für die Abweichungen der Werte des aktuellen Datenpunktes von den jeweiligen Mittelwerten μ_x und μ_y.
- Eine Variable, in der der aktuelle Fehlerwert e_{MSE} berechnet wird, damit er mit dem vom Benutzer vorgegebenen Schwellenwert verglichen werden kann.

Mit diesen Variablen kann der Fehler ohne (Zwischen-)Speicherung der Datenpunkte mit geringem und festem Speicheraufwand leicht fortgeschrieben und eine Entscheidung über die Ausweitung des aktuellen linearen Abschnitts getroffen werden.

Wie oben bereits erwähnt, tritt zu diesem Fehlerkriterium eine Erkennung (lokaler) Minima oder Maxima, um einen linearen Abschnitt nicht über solche Extrempunkte hinaus zu erweitern, sondern an solchen Extrempunkten auf jeden Fall einen neuen linearen Abschnitt zu beginnen, da sich der Signalverlauf offenbar signifikant ändert. Ein (lokales) Minimum oder Maximum kann leicht wie folgt erkannt werden. Seien (x_i, y_i), (x_j, y_j) und (x_k, y_k) drei Datenpunkte mit $i < j < k$, allerdings nicht notwendigerweise $j = i + 1$ und $k = j + 1$. (Der Grund für dieses allgemeinere Kriterium wird weiter unten klar werden — in der Anwendung ist stets $k = j + 1$, aber nicht immer $j = i + 1$.) Dann bildet der mittlere Datenpunkt (x_j, y_j) ein lokales Minimum oder Maximum (bzgl. dieser drei Datenpunkte), wenn gilt

$$\text{sgn}(y_j - y_i) \neq \text{sgn}(y_k - y_j), \qquad \text{wobei} \qquad \text{sgn}(z) = \begin{cases} -1, & \text{wenn } z < 0, \\ 0, & \text{wenn } z = 0, \\ +1, & \text{wenn } z > 0, \end{cases}$$

Man beachte, dass mit diesem Kriterium auch “Schultern” (z.B. ansteigender Verlauf, der in einen konstanten Verlauf übergeht oder ein fallender Verlauf, der in einen konstanten Verlauf übergeht, als (lokale) Minima oder Maxima erkannt werden. Für die Anwendung ist dies eine sinnvolle Erweiterung gegenüber einer strengen Erkennung eines (lokalen) Minimums oder Maximums, für die auf $\text{sgn}(y_j - y_i) \cdot \text{sgn}(y_k - y_j) < 0$ geprüft werden müßte.

Dieses Extremwertkriterium wird vor dem Fehlerkriterium für eine lineare Näherung (siehe oben) geprüft. Ist es erfüllt, wird unabhängig davon, ob das Fehlerkriterium erfüllt ist oder nicht, der aktuelle lineare Abschnitt beendet und ein neuer begonnen.

Die Prüfung auf ein (lokales) Minimum oder Maximum (oder auch einer “Schulter”) erscheint zwar sehr natürlich, führt aber auch zu einem Problem. Ist der Signalverlauf horizontal, d.h., ändert sich der Wert der abhängigen Variable Y über eine gewisse Zeit nicht, ist das Signal aber mit einem gewissen Rauschen behaftet (vgl. den oben erwähnten Störterm ϵ), so kann es durch das Extremwertkriterium zu einer Erkennung vieler (linearer) Abschnitte zwischen (lokalen) Minima und Maxima kommen, ohne dass eine echte Änderung des Signalverlaufes vorliegt. Ein horizontaler (oder fast horizontaler Verlauf, mit einer Steigung kleiner als das Signalrauschen) muss daher gesondert behandelt werden.

Um (fast) horizontale Signalverläufe zu behandeln, wird geprüft, ob alle Datenpunkte seit dem letzten (für eine Übertragung) ausgewählten Punkt innerhalb eines “Schlauches” eines vom Benutzer vorzugebenden maximalen Durchmessers liegen, d.h., ob die Differenz zwischen dem maximalen und dem minimalen y-Wert dieser Datenpunkte höchstens so groß ist wie ein vom Benutzer vorzugebender Schwellenwert θ_{yr} (“yr” für y-range).

Um dieses Kriterium zu testen, werden für den jeweils aktuellen Abschnitt der minimale und der maximale y-Wert bestimmt und ihre Differenz mit dem vom Benutzer vorgegebenen Schwellenwert θ_{yr} verglichen. Dies kann leicht mitlaufend geschehen:

1. Setze $y_{\text{min}} = \infty$ und $y_{\text{max}} = -\infty$.
2. Für jeden (neuen) Datenpunkt (x, y), berechne

$$\begin{aligned} y_{\text{min}} &\leftarrow \min\{y_{\text{min}}, y\} \\ y_{\text{max}} &\leftarrow \max\{y_{\text{max}}, y \end{aligned}$$

3. Wenn $y_{\text{max}} - y_{\text{min}} \leq \theta_{yr}$, dann erweitere den aktuellen (horizontalen) Abschnitt, sonst beginne einen neuen Abschnitt.

Damit haben wir nun alle Kriterien beisammen, die für eine Kompression von Sensordaten mit Hilfe von Polygonzügen benötigt werden. (Später werden lediglich noch Variationen des Fehlerkriteriums als Alternativen zur Wurzel aus dem mittleren quadratischen Fehler besprochen.) Wir können damit jetzt dieses Verfahren vollständig beschreiben.

Zusätzlich zu den bereits aufgeführten Variablen speichert das Verfahren zwei Datenpunkte: den *Anker*, der den Beginn des aktuellen linearen oder horizontalen Abschnitts markiert, und den *Kandidaten*, der der letzte verarbeitete Datenpunkt ist und der ein mögliches Ende des aktuellen (linearen oder horizontalen) Abschnitts darstellt. Bei Eintreffen eines neuen Datenpunktes werden die oben besprochenen Kriterien in der Reihenfolge Horizontalkriterium, Extremwertkriterium, und schließlich Fehlerkriterium geprüft. Zeigt eines dieser Kriterien das Ende eines (linearen oder horizontalen) Abschnitts an, so wird der aktuelle Kandidat als nächster Eckpunkt des Polygonzuges übertragen und zum neuen Anker, während die Kandidatenvariable geleert wird. Im Detail wird dabei so vorgegangen:

1. Benutzervorgabe: $\theta_{\text{RMSE}} \geq 0$ (maximaler mittlerer quadratischer Fehler) und $\theta_{\text{yr}} \geq 0$ (maximaler y-Wertebereich für einen horizontalen Verlauf).
2. Berechne $\theta_{\text{MSE}} = \theta^2_{RMSE}$ (um die Berechnung der Quadratwurzel zu vermeiden).
3. Initialisiere $p_{\text{anchor}} = \emptyset$ (Anker) und $p_{\text{cand}} = \emptyset$ (Kandidat) und $i = 0$ (Index, Punktanzahl). (Das Symbol $\emptyset$ wird hier benutzt, um anzuzeigen, dass noch kein Punkt zugewiesen ist.)
4. Für jeden (neuen) Datenpunkt (x, y):
   (a) Wenn $p_{\text{anchor}} = \emptyset$, setze $p_{\text{anchor}} = (x, y)$, $i = 1$, $y_{\text{min}} = y_{\text{max}} = y$, $\mu_x = x$, $\mu_y = y$, $\delta_{xx} = \delta_{yy} = \delta_{xy} = 0$ und gehe ohne Ausgabe zum nächsten Datenpunkt. (Man beachte: $p_{\text{anchor}} = \emptyset$ sollte nur für $i = 0$ auftreten.)
   (b) Aktualisiere die Variablen i, μ_x, μ_y, δ_{xx}, δ_{yy} und δ_{xy} mit dem neuen Punkt (x, y) gemäß dem erweiterten Welford-Algorithmus (siehe oben) sowie y_{min} und y_{max}.
   (c) Wenn $p_{\text{cand}} = \emptyset$, setze $p_{\text{cand}} = (x, y)$. (Man beachte: Für $p_{\text{cand}} = \emptyset$ gilt für das Fehlerkriterium immer $e_{\text{MSE}} = 0 \leq \theta_{\text{MSE}}$, da nur zwei Punkte, der Anker und der Kandidat, vorliegen, die durch eine Gerade immer perfekt beschrieben werden, nämlich die Gerade durch diese beiden Punkte.) Anschließend gehe ohne Ausgabe zum nächsten Datenpunkt.
   (d) Berechne $r^{(\text{new})} = \max\{y, y_{\text{max}}\} - y_{\text{min}}$, den y-Wertebereich unter Einbeziehung des neuen Punktes (x, y). Wenn $r^{\text{new}} \leq \theta_{\text{yr}}$, dann liegen alle Punkte vom Anker bis zum neuen Punkt (x, y) innerhalb eines "Schlauches" mit Durchmesser θ_{yr}. Es kann folglich der aktuelle (horizontale) Abschnitt erweitert werden. Daher wird der Kandidat verworfen und durch den neuen Punkt (x, y) ersetzt: $p_{\text{cand}} = (x, y)$. Anschließend gehe ohne Ausgabe zum nächsten Datenpunkt.

(e) Berechne $r^{(\text{old})} = y_{\max} - y_{\min}$, d.h., den y-Wertebereich ohne den neuen Punkt (x, y). Wenn $r^{\text{old}} \leq \theta_{\text{yr}}$, dann liegen alle Datenpunkte vom Anker bis zum Kandidaten innerhalb eines "Schlauches" mit Durchmesser θ_{yr}, aber durch den neuen Datenpunkt wird dieser "Schlauch" verlassen. Der aktuelle horizontale Abschnitt kann daher nicht erweitert werden, sondern muss mit dem Kandidaten enden. Der Kandidat wird zum neuen Anker: $p_{\text{anchor}} = p_{\text{cand}}$, und der neue Punkt ersetzt den Kandidaten $p_{\text{cand}} = (x, y)$. Setze $y_{\min} = \min\{y_{\text{anchor}}, y\}$ und $y_{\max} = \max\{y_{\text{anchor}}, y\}$. Gehe danach unter Ausgabe des neuen Ankers zum nächsten Datenpunkt.

(f) Prüfe, ob der Kandidat ein Extremwert unter den drei Punkten p_{anchor}, p_{cand} und (x, y) ist, d.h., ob $\operatorname{sgn}(y_{\text{anchor}} - y_{\text{cand}}) \neq \operatorname{sgn}(y_{\text{cand}} - y)$. Ist dies der Fall, wird durch den Kandidaten ein (linear) steigender oder fallender Abschnitt beendet. Dieser Kandidat wird daher zum neuen Anker: $p_{\text{anchor}} = p_{\text{cand}}$, und der Kandidat wird durch den neuen Punkt ersetzt: $p_{\text{cand}} = (x, y)$. Setze $y_{\min} = \min\{y_{\text{anchor}}, y\}$ und $y_{\max} = \max\{y_{\text{anchor}}, y\}$. Gehe danach unter Ausgabe des neuen Ankers (der den vorangehenden linearen Abschnitt beendet) zum nächsten Datenpunkt.

(g) Berechne $e_{\text{MSE}} = \frac{1}{i}\left(\delta_{yy} - \delta_{xy}^2/\delta_{xx}\right)$. Wenn $e_{\text{MSE}} \leq \theta_{\text{MSE}}$, dann kann der aktuelle lineare (steigende oder fallende) Abschnitt erweitert werden. Daher wird der Kandidat verworfen und durch den neuen Punkt (x, y) ersetzt: $p_{\text{cand}} = (x, y)$. Anschließend gehe ohne Ausgabe zum nächsten Datenpunkt.

(h) Ist dagegen $e_{\text{MSE}} > \theta_{\text{MSE}}$, so muss der aktuelle lineare Abschnitt beendet werden, da eine Erweiterung zu einem zu großen Fehler führt. Dieser Abschnitt wird durch den Kandidaten beendet. Dieser Kandidat wird daher zum neuen Anker: $p_{\text{anchor}} = p_{\text{cand}}$, und der Kandidat wird durch den neuen Punkt ersetzt: $p_{\text{cand}} = (x, y)$. Setze $y_{\min} = \min\{y_{\text{anchor}}, y\}$ und $y_{\max} = \max\{y_{\text{anchor}}, y\}$. Gehe danach unter Ausgabe des neuen Ankers (der den linearen Abschnitt beendet) zum nächsten Datenpunkt.

Eine prototypische Implementierung dieses Verfahrens als Python-Klasse findet sich in Anhang C. Man beachte, dass diese Implementierung nicht für den direkten Einsatz auf einer Steuereinheit geeignet ist, da es kaum möglich ist, auf einer Steuereinheit eine Python-Umgebung zu installieren. Außerdem ist Python nicht für seine Laufzeiteffizienz bekannt, so dass selbst dann, wenn eine Installation möglich wäre, kaum die nötige Geschwindigkeit erreicht würde. Diese Implementierung zeigt daher nur das Prinzip, müßte aber für den praktischen Einsatz in einer Programmiersprache wie C/C++ reimplementiert werden.

Bisher wurde als Fehlerkriterium der mittlere quadratische Fehler verwendet. Zwar hat dieser Vorteile gegenüber der Fehlerquadratsumme, aber immer noch den Nachteil, dass er vom y-Wertebereich des Signals abhängt. Es muss daher für jeden Sensor, dessen Daten auf diese Weise komprimiert werden sollen, abhängig vom y-Wertebereich eine eigene Fehlerschranke definiert werden. Auch wenn dies unter Ausnutzung von Hintergrundwissen möglich sein mag, wäre es bequemer, ein Maß für die Güte einer linearen Abhängigkeit zu verwenden, das vom y-Wertebereich unabhängig ist. Ein solches Maß ist der Pearsonsche Korrelationskoeffizient:

$$r = \frac{\sigma_{xy}}{\sqrt{\sigma_{xx}\sigma_{yy}}}.$$

Unter Ausnutzung der bereits weiter oben verwendeten Beziehungen

$$\delta_{xx} = (n-1)\sigma_{xx}, \qquad \delta_{yy} = (n-1)\sigma_{yy}, \qquad \delta_{xy} = (n-1)\sigma_{xy}.$$

kann der Korrelationskoeffizient direkt aus den Summen der Abweichungsquadrate und der Summe der Produkte der Abweichungen von den jeweiligen Mittelwerten berechnet werden:

$$r = \frac{\delta_{xy}}{\sqrt{\delta_{xx}\delta_{yy}}}.$$

Der Korrelationskoeffizient liegt stets im Intervall $[-1, 1]$, wobei ein negativer Wert eine fallende und ein positiver Wert eine steigende lineare Abhängigkeit anzeigt. Um nicht zwischen steigenden und fallenden Abhängigkeiten unterscheiden zu müssen, sollte entweder der Betrag $|r|$ des Korrelationskoeffizienten oder sein Quadrat r^2, das auch als Bestimmtheitsmaß bekannt ist, verwendet werden. Wir entscheiden uns hier für die zweite Möglichkeit:

$$r^2 = \frac{\delta_{xy}^2}{\delta_{xx}\delta_{yy}}.$$

Für den (quadrierten) Korrelationskoeffizienten kann analog zur Schranke θ_{RMSE} für die Wurzel aus dem mittleren quadratischen Fehler eine Schranke θ_{corr} vorgegeben werden. Einziger Unterschied ist, dass θ_{RMSE} eine Obergrenze, aber θ_{corr} eine Untergrenze darstellt. Solange der quadrierte Korrelationskoeffizient (das Bestimmtheitsmaß) über dieser Schranke liegt, kann der aktuelle (lineare) Abschnitt erweitert werden; fällt er unter diese Schranke, so wird der aktuelle Abschnitt beendet und ein neuer begonnen. Zur Verwendung des Korrelationskoeffizienten muss folglich in dem oben beschriebenen Verfahren lediglich die Benutzervorgabe von θ_{RMSE} durch die Vorgabe von $\theta_{\text{corr}} \in [0, 1]$ ersetzt werden und die Schritte 4(g) und 4(h) durch:

4. Für jeden Datenpunkt (x, y):
   ...
   (g) Berechne $r^2 = \frac{\delta_{xy}^2}{\delta_{xx}\delta_{yy}}$. Wenn $r^2 \geq \theta_{\text{corr}}^2$, dann kann der aktuelle lineare (steigende oder fallende) Abschnitt erweitert werden, da der Fehler klein genug bleibt. Daher wird der Kandidat verworfen und durch den neuen Punkt (x, y) ersetzt: $p_{\text{cand}} = (x, y)$. Anschließend gehe ohne Ausgabe zum nächsten Datenpunkt.
   (h) Ist dagegen $r^2 < \theta_{\text{corr}}^2$, so muss der aktuelle lineare Abschnitt beendet werden, da eine Erweiterung zu einem zu großen Fehler führt. Dieser Abschnitt wird durch den Kandidaten beendet. Dieser Kandidat wird daher zum neuen Anker: $p_{\text{anchor}} = p_{\text{cand}}$, und der Kandidat wird durch den neuen Punkt ersetzt: $p_{\text{cand}} = (x, y)$. Setze $y_{\text{min}} = \min\{y_{\text{anchor}}, y\}$ und $y_{\text{max}} = \max\{y_{\text{anchor}}, y\}$. Gehe danach unter Ausgabe des neuen Ankers (der den linearen Abschnitt beendet) zum nächsten Datenpunkt.

Eine prototypische Implementierung dieses Verfahrens als Python-Klasse findet sich in Anhang C. Für diese Implementierung gelten die gleichen Bemerkungen, die bereits oben zur Implementierung des Verfahrens mit der Wurzel aus dem mittleren quadratischen Fehler gemacht wurden: Sie ist beispielhaft, für einen direkten Einsatz auf einer Steuereinheit nicht brauchbar und müßte für einen solchen Einsatz in eine Sprache wie C/C++ portiert werden.

Ein Nachteil der beiden soeben beschriebenen Verfahren ist, dass zur Berechnung des Fehlers oder des (quadrierten) Korrelationskoeffizienten auf die Regressionsgrade für die betrachteten Datenpunkte zurückgegriffen wird, diese Gerade im allgemeinen jedoch nicht durch die übertragenen Punkte geht. Dies ist in Abbildung 5 illustriert. Nehmen wir an, dass die im linken Teilbild gezeigten Punkte einen linearen Abschnitt bilden, also lediglich der erste und der letzte dieser Punkte für die Polygonzugnäherung übertragen werden. Dann wird der Fehler, unabhängig davon, ob mit der Wurzel der Fehlerquadratsumme oder dem (quadrierten) Korrelationskoeffizienten gearbeitet wird, anhand der Regressionsgerade dieser Datenpunkte eingeschätzt. Diese Regressionsgerade ist im rechten Teilbild in blau gezeigt. Offenbar geht sie nicht durch die übertragenen Punkte, also den ersten Punkt $(1, 1)$ und den letzten Punkt $(8, 6)$. Die Übertragung legt jedoch die Gerade nahe, die durch diese beiden Punkte geht, also die im rechten Teilbild rot gezeichnete Gerade. Diese minimiert zwar nicht die Fehlerquadratsumme, aber es ist die Gerade, die nach der Übertragung "gesehen" wird. Deshalb ist es angemessener, diese Gerade zur Fehlereinschätzung zu verwenden, damit sich der Fehler darauf bezieht, wie der sich ergebende Polygonzug von den Datenpunkten abweicht.

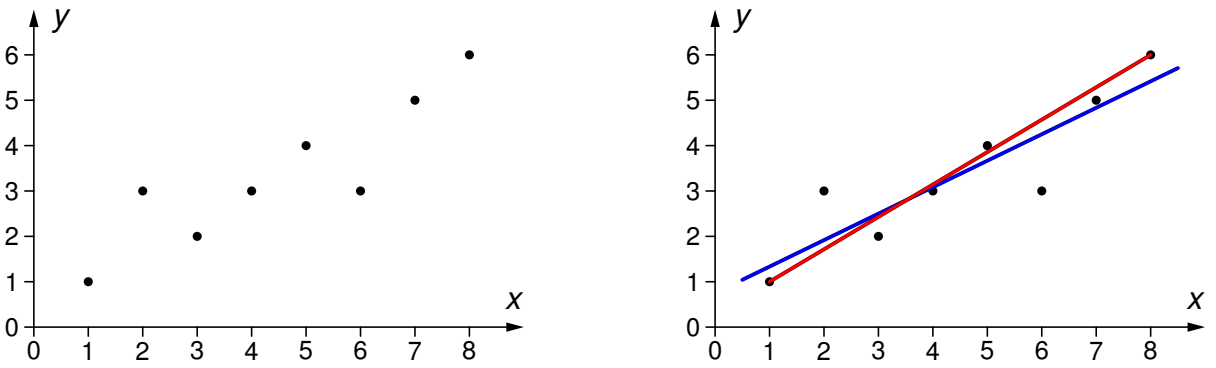

Abbildung 5: Eine Regressionsgerade geht im allgemeinen nicht durch den ersten und letzten Datenpunkt eines linearen Abschnitts, aber dies sind die Datenpunkte, die übertragen werden.

Um diesen Fehler zu berechnen, betrachten wir den Fall, dass eine Regressionsgerade bestimmt werden soll, die durch einen vorgegebenen Punkt (x_0, y_0) geht. In der Anwendung wird dies stets der Ankerpunkt sein, aber die folgende Beschreibung ist allgemeiner gehalten. Wir gehen also von der Hypothese aus, dass die funktionale Abhängigkeit der abhängigen von der unabhängigen Variable $Y = f(X - x_0) = y_0 + b(X - x_0) + \epsilon$ lautet. Hier ist X wieder eine Zufallsvariable, die die Verteilung der unabhängigen Variable beschreibt und ϵ ein zufälliger Störterm, der den Erwartungswert 0 besitzt. Der Parameter b beschreibt die lineare Abhängigkeit. Es gibt in diesem Modell keinen konstanten Term a ("*intercept* "), damit die Regressionsgerade auf jeden Fall durch den vorgegebenen Punkt (x_0, y_0) verläuft.

Ziel der Regression ist es, den Steigungsparameter b so zu bestimmen, dass die gegebenen Datenpunkte durch die Funktion $Y = f(X - x_0) = y_0 + b(X - x_0)$ möglichst gut angenähert werden. Für die Berechnungen ist es bequem, die Datenpunkte so zu transformieren, dass sie relativ zum Punkt (x_0, y_0) angegeben werden. Man könnte auch sagen, sie werden so verschoben, dass der Punkt (x_0, y_0) im Ursprung des Koordinatensystems liegt. Folglich ist es günstig, das um x_0 und y_0 „verschobene" Modell $Y - y_0 = b(X - x_0)$ zu betrachten.

Als Fehlerkriterium verwenden wir zunächst wieder die Summe der quadrierten Abweichungen zwischen den tatsächlichen Zielwerten y_i und den aus den zugehörigen Werten x_i mit Hilfe des Parameters b berechneten hypothetischen Zielwerten $y_0 = y_0 + b(x_i - x_0)$ verwendet. Diese Fehlerquadratsumme, d.h., das Fehlerfunktional

$$E(b) = \sum_{i=1}^{n} (b(x_i - x_0) - (y_i - y_0))^2.$$

ist durch eine geeignete Wahl des Parameters b zu minimieren.

Eine Lösung dieses Optimierungsproblems läßt sich wieder mit dem klassischen Ansatz bestimmen, der ausnutzt, dass notwendige Bedingungen für ein Minimum einer Funktion sind, dass die partiellen Ableitungen nach den Parametern der Funktion verschwinden. Hier gibt es nur den einen Parameter b; also muss gelten

$$\frac{\partial E}{\partial a} = 2 \sum_{i=1}^{n} (b(x_i - x_0) - (y_i - y_0)) \stackrel{!}{=} 0$$

Dies liefert die Normalgleichung

$$b \sum_{i=1}^{n} (x_i - x_0) = \sum_{i=1}^{n} (y_i - y_0)$$

(nur eine Gleichung, da der Parameter a fehlt), also

$$b = \frac{\sum_{i=1}^{n}(y_i - y_0)}{\sum_{i=1}^{n}(x_i - x_0)}.$$

Die Summe der Fehlerquadrate kann geschrieben werden als

$$\begin{aligned} E^{(0)} &= \sum_{i=1}^{n}(b(x_i - x_0) - (y_i - y_0))^2 \\ &= b^2 \sum_{i=1}^{n}(x_i - x_0)^2 - 2b \sum_{i=1}^{n}(x_i - x_0)(y_i - y_0) + \sum_{i=1}^{n}(y_i - y_0)^2 \\ &= b^2 s_{xx} - 2b\, s_{xy} + s_{yy}, \end{aligned}$$

wobei die s_{zz} die Summen für die um x_0 und y_0 „verschobenen“ Datenpunkte sind, also

$$s_{xx} = \sum_{i=1}^{n}(x_i - x_0)^2, \qquad s_{yy} = \sum_{i=1}^{n}(y_i - y_0)^2, \qquad s_{xy} = \sum_{i=1}^{n}(x_i - x_0)(y_i - y_0).$$

Man beachte, dass in diesem Ansatz nur diese Summen fortgeschrieben werden müssen und nicht die Summen der Abweichungsquadrate von den Mittelwerten, also nicht δ_{xx}, δ_{yy} und δ_{xy}. Dadurch braucht man den Welford-Algorithmus nicht und kann damit auch auf die Berechnung/Fortschreibung der Mittelwerte μ_x und μ_y verzichten, was Speicher spart.

Statt der Summe E der quadratischen Abweichungen ist es aber auch hier wieder besser, die Quadratwurzel aus dem mittleren quadratischen Fehler (*“root mean squared error”*) für die Filterung einzusetzen, da diese nicht von der Zahl der Datenpunkte abhängt und die gleiche Einheit hat wie die Zielgröße y (statt eine quadratische Größe zu sein). Wir verwenden also

$$e^{(0)}_{\text{RMSE}} = \sqrt{\tfrac{1}{n}E^{(0)}} = \sqrt{\tfrac{1}{n}(b^2 s_{xx} - 2b\, s_{xy} + s_{yy})}.$$

Mit diesem Maß kann der Filteralgorithmus genauso durchgeführt werden wie oben für e_{RMSE} angegeben (siehe Seite 14f). Allerdings darf nicht direkt mir den Datenpunkten selbst gearbeitet werden, sondern es müssen stets die Differenzen $x_i - x_0$ und $y_i - y_0$ betrachtet werden.

In der Anwendung wird man natürlich nicht einen beliebigen Punkt (x_0, y_0) vorgeben, der mit den Daten nichts zu tun hat, durch den die Regressionsgerade verlaufen muss, sondern den Anker wählen, also $(x_0, y_0) = (x_{\text{anchor}}, y_{\text{anchor}})$. Es sollte außerdem klar sein, dass sich an der Berechnung der Fehlerquadratsumme nichts ändert, wenn statt des oben bestimmten optimalen Wertes für den Paramater b die Steigung der Gerade durch den Anker und einen neuen Datenpunkt (x, y), also der Gerade durch $(x_{\text{anchor}}, y_{\text{anchor}})$ und (x, y), verwendet wird. In diesem Fall ist die Steigung b zu berechnen als

$$b = \frac{y - y_{\text{anchor}}}{x - x_{\text{anchor}}}.$$

Wird mit diesem Wert für die Steigung b gearbeitet, so wird der Fehler stets relativ zu der Gerade eingeschätzt, die nach der Übertragung der ausgesonderten Punkte „gesehen“ wird.

Wie oben hat allerdings die Quadratwurzel des mittleren quadratischen Fehlers aber immer noch den Nachteil, dass er vom y-Wertebereich des Signals abhängt. Es muss daher für jeden Sensor, dessen Daten auf diese Weise komprimiert werden sollen, abhängig vom y-Wertebereich eine eigene Fehlerschranke definiert werden. Bequemer wäre es, ein Maß für die Güte der übertragenen Geradenstücke zu haben, das vom y-Wertebereich unabhängig ist. Oben haben wir dafür den quadrierten Pearsonschen Korrelationskoeffizienten verwendet, der

für eine lineare Regression mit dem Bestimmtheitsmaß identisch ist. Das Bestimmtheitsmaß (“*coefficient of determination*”) ist jedoch eigentlich allgemeiner und definiert als

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \mu_y)^2},$$

also als 1 minus die Summe der Residuenquadrate (Fehlerquadrate) der Regressionsfunktion f geteilt durch die Varianz der Zielgröße y. Dises Maß kann (innerhalb gewisser Grenzen) auch für nicht-lineare Regressionsfunktionen oder lineare Funktionen, die von der optimalen Regressionsgerade abweichen, eingesetzt werden.

Für den Spezialfall eines Regressionsmodels mit einer Gerade durch den Anker, wie er gerade betrachtet wurde, erhält man:

$$\begin{aligned} R_0^2 &= 1 - \frac{\sum_{i=1}^{n}((y_i - y_0) - b(x_i - x_0))^2}{\sum_{i=1}^{n}((y_i - y_0) - \mu_{y-y_0})^2} \\ &= 1 - \frac{E^{(0)}}{s_{yy}} = 1 - \frac{b^2 s_{xx} - 2bs_{xy} + s_{yy}}{s_{yy}} = \frac{2bs_{xy} - b^2 s_{xx}}{s_{yy}}. \end{aligned}$$

Dieses Maß kann analog zum quadrierten Korrelationskoeffizienten eingesetzt werden (Anpassungen wie auf Seite 16 angegeben), d.h., es wird ein Schwellenwert θ_{CoD} vorgegeben, dessen Unterschreiten zu einem Beenden des aktuellen und Beginnen eines neuen linearen Abschnitts führt. Außerdem darf natürlich wieder nicht direkt mir den Datenpunkten selbst gearbeitet werden, sondern es müssen stets die Differenzen $x_i - x_0$ und $y_i - y_0$ betrachtet werden.

Auch hier kann natürlich die Steigung b berechnet werden als

$$b = \frac{y - y_{\mathrm{anchor}}}{x - x_{\mathrm{anchor}}}.$$

Wird mit diesem Wert für die Steigung b gearbeitet, so wird der Fehler stets relativ zu der Gerade eingeschätzt, die nach der Übertragung der ausgesonderten Punkte „gesehen“ wird.

In den bisher verwendeten Methoden beruhte das Fehlermaß auf den quadrierten Abweichungen zwischen den tatsächlichen Zielwerten y und den aus den zugehörigen Werten x mit Hilfe der Parameter a und b der linearen Abhängigkeit berechneten hypothetischen Zielwerten $\hat{y}_i = a + bx_i$. Als Alternative bietet sich auch noch eine Betrachtung der Summe der Absolutwerte dieser Abweichungen an. Allerdings läßt sich diese Summe kaum exakt berechnen, ohne alle Datenpunkte des aktuellen Abschnitts zwischenzuspeichern. Es ist aber möglich, eine (pessimistische) Näherung dieser Summe zu berechnen, mit der die Datenpunkte ebenfalls wirksam gefiltert und durch einen Polygonzug angenähert werden können.

Dieser Ansatz hat gegenüber den beiden vorangehenden den Vorteil, dass auf den erweiterten Welford-Algorithmus und damit auf die für ihn benötigten Variablen verzichtet werden kann. Statt einer Regressionsgerade wird stets die Gerade von Anker zum Kandidaten bzw. zum neuen Datenpunkt betrachtet, wie dies auch bei den beiden vorangehenden Ansätzen der Fall war. Dies kann, zusätzlich zum geringeren Speicherbedarf, als weiterer Vorteil gesehen werden, da ja der am Ende enstehende Polygonzug genau aus solchen Geraden besteht und nicht aus den Ausgleichsgeraden, durch die die Datenpunkte eines (linearen) Abschnitts angenähert werden. Es werden folglich außer dem Anker, dem Kandidaten und der Punktanzahl/dem Punktindex i nur eine Variable zum Aufsummieren der absoluten Abweichungen (für die Filterung linearer Abschnitte) nur noch die Variablen $y_{\min}$ und $y_{\max}$ für die Horizontalfilterung benötigt. (Die Extremwertfilterung verwendet nur Anker, Kandidaten und neuen Datenpunkt, die ja sowieso gespeichert werden müssen.) Im Vergleich dazu müssen die beiden vorangehenden Verfahren die drei Summen s_{xx}, s_{yy} und s_{xy} fortschreiben (aber immerhin nicht mehr

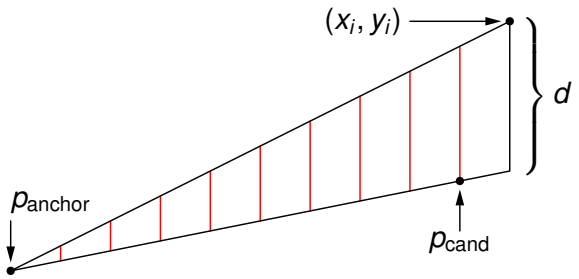

Abbildung 6: Illustration der Berechnung der Summe der zusätzlichen absoluten Fehler durch den Übergang von der alten Geraden p_{anchor}—p_{cand} auf die neue Gerade p_{anchor}—(x, y). Die Summe der absoluten Fehler wird höchstens um die Summe der Längen der roten Linien erhöht. Die Längen dieser Linien sind $d_j = \frac{x_j - x_{\text{anchor}}}{x - x_{\text{anchor}}}$, wobei j die Indizes der Punkte zwischen p_{anchor} und p_{cand} durchläuft (jede rote Linie gehört zu einem Punkt).

die Mittelwerte μ_x und μ_y wie die ersten beiden Verfahren). Ein Nachteil gegenüber den beiden vorangehenden Ansätzen ist jedoch, dass die Summe der absoluten Abweichungen nur pessimistisch geschätzt und nicht wie die Fehlerquadratsumme exakt berechnet werden kann.

Die Berechnung einer Näherung der Summe der absoluten Abweichungen beruht auf dem Strahlensatz. Die Grundidee ist, die Abstände zwischen der alten Gerade p_{anchor}—p_{cand} und der (potentiellen) neuen Gerade p_{anchor}—(x, y) an allen Punkten zwischen x_{anchor} und x zu betrachten, wobei implizit angenommen wird, dass die x-Werte äquidistant sind (was für das zu filternde Originalsignal sinnvoll erwartet werden kann). Am Anker ist dieser Abstand offenbar 0 (da ja beide Geraden durch den Anker gehen). Der Abstand am Punkt x ist die (absolute) Differenz zwischen y und einem aus der Geraden p_{anchor}—p_{cand} berechenbaren Schätzwert $\hat{y}$ und sei mit d bezeichnet. An den Punkten x_j zwischen dem Anker und dem Punkt x sind die Abstände unter der (pessimistischen) Annahme, dass sie alle in die gleiche Richtung gehen und nicht von späteren Veränderungen der Gerade wieder verringert werden (man beachte: hierin besteht die pessimistische Näherung), nach Strahlensatz

$$d_j = \frac{x_j - x_{\text{anchor}}}{x - x_{\text{anchor}}}$$

Die Gesamterhöhung der Summe der absoluten Abweichungen ist folglich

$$D' = \frac{i \cdot d}{2},$$

wenn wir den Abstand d (Dreiecksseite) einbeziehen, denn dann schaut man effektiv auf äquidistante “Sehnen” in einem Dreieck mit dem Ecken p_{anchor}, (x, y) und $(x, y \pm d)$ (siehe Abbildung 6). Allerdings trägt der Abstand d tatsächlich nicht zum Fehler bei, weil die neue Gerade (nach einer potentiellen Erweiterung des aktuellen linearen Abschnitts) durch den Punkt (x, y) geht. Folglich müssen wir d abziehen und erhalten

$$D = D' - d = \frac{i \cdot d}{2} - d = \frac{i - 2}{2} d.$$

Dieser Fehler, der allein durch den Übergang vom Kandidaten auf den neuen Datenpunkt entsteht, ist in einer internen Variable Δ zu aggregieren, die (eine Näherung) der Summe der absoluten Fehler für die Datenpunkte im aktuellen linearen Abschnitt fortschreibt. D.h., zu Beginn eines neuen linearen Abschnitts wird $\Delta = 0$ gesetzt und anschließend für für jeden Datenpunkt, um den der aktuelle lineare Abschnitt erweitert wird, berechnet

$$\Delta \leftarrow \Delta + D.$$

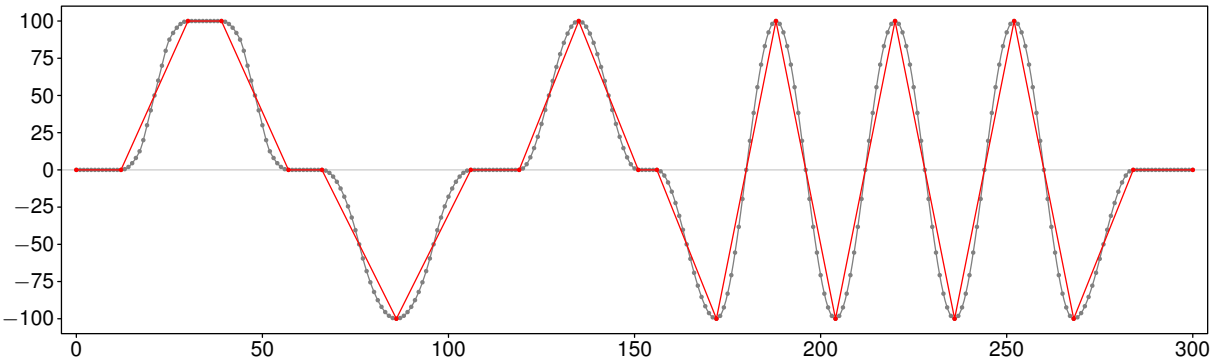

Abbildung 7: Abtastung des Signals aus Abbildung 1 (grau) mit variabler Schrittweite, die von dem vorgestellten Verfahren mit $\theta_{yr} = 0.1$ und entweder $\theta_{\text{RMSE}} = 10$, $\theta_{\text{corr}} = 0.9$, $\theta_{\text{MAE}} = 24$ oder $\theta_{\text{RAE}} = 0.9$ erzeugt wird. Sie stimmt mit der “idealen” Abtastung aus Abbildung 4 überein.

Da diese Summe wieder von der Zahl der Datenpunkte abhängt (wie oben die Summe der quadratischen Abweichungen), ist sie nicht gut direkt einsetzbar. Besser ist es, aus *Delta* den mittleren absoluten Fehler zu berechnen:

$$e_{\text{MAE}} = \frac{\Delta}{i}.$$

Für diesen mittleren absoluten Fehler kann nun ein Benutzer analog zu θ_{RMSE} eine Obergrenze θ_{MAE} vorgeben. Außerdem sind natürlich die Schritte 4(g) und 4(h) in dem oben für die Wurzel aus dem mittleren quadratischen Fehler beschriebenen Verfahren anzupassen, so dass sie auf e_{MAE} und θ_{MAE} Bezug nehmen und schließlich ist $\Delta = 0$ überall dort einzufügen, wo der Anker ersetzt wird (wo auch y_{min} und y_{max} neu initialisiert werden).

Für den mittleren absoluten Fehler gilt jedoch auch, was oben über die Wurzel aus dem mittleren quadratischen Fehler gesagt wurde. Welche Werte für den Schwellenwert θ_{MAE} geeignet sind, hängt vom y-Wertebereich des zu komprimierenden Signals ab und muss daher ggf. für jedes zu komprimierende Signal getrennt bestimmt werden.

Dieses Problem kann man umgehen, indem man den mittleren absoluten Fehler zum y-Wertebereich des aktuellen Abschnitts in Beziehung setzt, d.h. den relativen mittleren absoluten Fehler berechnet:

$$e_{\text{RAE}} = \frac{e_{\text{MAE}}}{y_{\text{max}} - y_{\text{min}}}.$$

Für diesen relativen mittleren absoluten Fehler kann nun ein Benutzer wieder eine Obergrenze θ_{RAE} vorgeben. Außerdem sind natürlich wieder die Schritte 4(g) und 4(h) anzupassen.

Eine prototypische Implementierung dieser beiden Verfahren, mit dem mittleren absoluten Fehler und dem relativen mittleren absoluten Fehler, als Python-Klasse findet sich in Anhang C. Für diese Implementierungen gelten die gleichen Bemerkungen, die bereits oben gemacht wurden: Sie sind beispielhaft, für einen Einsatz auf einer Steuereinheit nicht direkt brauchbar, und müßten für einen solchen Einsatz in eine Sprache wie C/C++ portiert werden.

Dass alle beschriebenen Verfahren, ob basierend auf der Wurzel aus dem mittleren quadratischen Fehler zur Regressionsgeraden, dem quadrierten Korrelationskoeffizienten, der Wurzel aus dem mittleren quadratischen Fehler zur Geraden durch den Anker und einen neuen Punkt, dem Bestimmtheitsmaß dieser Geraden, dem mittleren absoluten Fehler oder dem relativen absoluten Fehler, zumindest für das in Abbildung 1 gezeigte Signal eine wirksame Komprimierung liefern, ist in Abbildung 7 gezeigt. Alle vier Verfahren liefern das gleiche, mit der “idealen” Abtastung aus Abbildung 4 übereinstimmende Ergebnis, wenn man als Parameter $\theta_{yr} = 0.1$ (maximaler y-Wertebereich für einen horizontalen Abschnitt) und entweder $\theta_{\text{RMSE}} = 10$, $\theta_{\text{corr}} = 0.9$, $\theta^{(0)}_{\text{RMSE}} = 10$, $\theta_{CoD} = 0.75$, $\theta_{\text{MAE}} = 24$ oder $\theta_{\text{RAE}} = 0.9$ wählt.

Welches der Verfahren für eine konkrete Aufgabe am geeignetsten ist, hängt von der Anwendung und den Daten ab, die sie produziert. Ein Vorteil der Wurzel aus dem mittleren quadratischen Fehler und des quadrierten Korrelationskoeffizienten ist, dass die Fehlermaße nicht genähert sind (es wird eine Regressiongerade als Näherung der Datenpunkte berechnet, aber der Fehler dieser Gerade bzw. das Bestimmtheitsmaß der Datenpunkte wird exakt berechnet). Verwendet man den Regressionsansatz, der statt der optimalen Regressionsgerade (die im allgemeinen von den übertragenen Geradenstücken abweicht) die Gerade durch den Anker und einen neuen Datenpunkt betrachtet, so wird der Fehler bezüglich der tatsächlich übertragenen Geradenstücke eingeschätzt und es müssen sogar weniger Größen fortgeschrieben werden.

Im Gegensatz dazu arbeiten die Verfahren, die den mittleren absoluten Fehler oder den relativen mittleren absoluten Fehler verwenden, nur mit einer (pessimistischen) Näherung dieses Fehlers. Der Fehler wird tendenziell überschätzt und die Überschätzungen können sich über mehrere Punkte aggregieren, so dass eine lineare Näherung u.U. verworfen wird, obwohl sie bei exakter Berechnung dieser Fehlermaße noch akzeptabel wäre. Diesem Nachteil steht jedoch der Vorteil gegenüber, dass statt sechs Variablen (zusätzlich zur Punktanzahl/zum Punktindex i) für den Welford-Algorithmus bei Verwendung einer allgemeinen Regressionsgeraden bzw. drei Variablen für den Ansatz, der eine Gerade durch den Anker und einen neuen Datenpunkt betrachtet, nur eine einzige Variable Δ zum Aufsummieren der absoluten Fehler benötigt wird. Außerdem sind die durchzuführenden Rechnungen etwas einfacher und erfordern etwas weniger Schritte. Wenn der Speicherplatz auf der Steuereinheit, die die Sensordatenkomprimierung vornehmen soll, sehr knapp ist und die Rechenleistung sehr begrenzt, sind diese Verfahren daher klar im Vorteil. Die Näherung dürfte außerdem meist relativ harmlos sein.

4 Merkmalsberechnung und Verfügbarmachen berechneter Merkmale

Gerade dann, wenn Zeitreihendaten analysiert werden, aber auch für Daten anderen Charakters, besteht eine zentrale Problemstellung in der Aufbereitung, Vorverarbeitung und Aggregation der Rohdaten, welche von Sensoren und Logbuchprozessen auf Industriemaschinen erzeugt werden. Zwar könnte man diese Vorverarbeitung auch für jeden Einzelfall im zu erzeugenden Modell durchführen, doch führt dies zu unnötigem, weil redundantem Aufwand, da Datenanalysemodelle oft ähnliche Aufbereitungen, Transformationen und Aggregationen erfordern. Es ist daher wünschenswert, diese Aufbereitungen und Aggregationen in ein getrenntes, konfigurierbares Modul, nämlich einen Merkmalsspeicher (“*feature store*”) auszulagern. Nachfolgend werden zunächst die zentralen Charakteristiken eines solchen Merkmalsspeichers diskutiert, sowie dessen mögliche Einbindung im Kontext der i-Twin-Plattform und letztlich ein Ansatz zur Implementierung eines solchen Merkmalsspeichers unter Verwendung von Apache Spark, welcher in der Lage ist, die erwähnten Charakteristiken bereitzustellen.

4.1 Eigenschaften eines Merkmalsspeichers

Merkmalsspeicher übernehmen in automatisierter und zentralisierter Weise die Merkmalserzeugung für eine oder mehrere Machine-Learning- oder Data-Analytics-Anwendungen. Sie sorgen dafür, dass Merkmale einfach definiert und konsistent berechnet werden, auch wenn Änderungen im datengenerierenden Prozess auftreten (beispielsweise eine veränderte Abtastrate eines Sensors). Damit wird auch Redundanz bei der Merkmalsberechnung verhindert, vor allem dann, wenn mehrere Applikationen dieselben Merkmale benötigen.

Was die Bereitstellung der erzeugen Merkmale angeht, so sind Merkmalsspeicher üblicherweise sowohl in der Lage, diese direkt an die jeweiligen Anwendungen weiterzuleiten, welche sich für die entsprechenden Merkmale registriert haben, können diese aber — sofern nötig — ebenfalls persistent in Form von Datensätzen ablegen, was insbesondere für das Trainieren von Machine-Learning-Modellen benötigt wird.

Falls ein solcher Ausgabedatensatz Merkmale mehrerer unterschiedlicher Signale vereint, ist es außerdem nötig, diese in eine einheitliche und äquidistante zeitliche Auflösung zu überführen (z.B. durch Abtastratentransformation). Dies ist essentiell, um ein Ausgabeformat sicherzustellen, welches wiederum als Eingabe für nachfolgende Machine-Learning-Modelle dienen kann. Hierfür müssen die berechneten Merkmale unter Umständen auf eine äquidistante Zeitreihe, welche die gewünschte Abtastrate aufweist, interpoliert werden.

Des weiteren gewährleistet eine konstante Überwachung der Eingabedaten, aus welchen die Merkmale berechnet werden, das frühzeitige Erkennen von substantiellen Veränderungen im datengenerierenden Prozess. Eine solche Veränderung zu erkennen ist insbesondere im Kontext von Machine-Learning-Modellen essentiell, da sie in der Regel zur Folge hat, dass die berechneten Merkmale nicht mehr mit jenen vergleichbar sind, welche zum Trainieren des Modells verwendet wurden und deshalb das Modell auch nicht mehr zuverlässig auf diese angewendet werden kann. Deshalb werden die Eingangssignale, welche zur Merkmalsberechnung verwendet werden, durch einen Monitoring-Prozess überwacht, sodass eine solche Veränderung frühzeitig erkannt und gemeldet werden kann. [Chapman *et al.* 1999]

Schlussendlich sollte — wie Eingangs erwähnt — die Interaktion des Nutzers mit dem Merkmalsspeicher, also die Definition neuer sowie die Verwaltung und der Bezug bestehender Merkmale, über eine möglichst einfache Schnittstelle erfolgen, welche gleichzeitig eine möglichst hohe Flexibilität bzw. ein möglichst großes Spektrum an Möglichkeiten für die Definition von Merkmalen gewährleistet.

4.2 Einbindung eines Merkmalsspeichers in die i-Twin-Plattform

Im Rahmen der i-Twin-Datenintegrationsschicht (“*data integration layer*”) erfolgt die Weitergabe von Sensor- und ähnlichen, von Produktionsmaschinen erhobenen Daten an Anwendungen durch das Verteilungsnetzwerk (“*distribution network*”). Hierbei subskribieren Anwendungen auf der Ebene der Anwendungsschicht (“*application layer*”) sich für die Themen (“*topics*”) jener Ereigniselemente, welche die Daten der zu überwachenden I4.0-Komponente zur Verfügung stellt und erhalten für die Dauer dieser Subskribierung entsprechende Ereignisnachrichten (“*event messages*”), welche zusätzlich zu den Ereignisdaten (also beispielsweise Sensormesswerte) auch notwendige Meta-Informationen wie beispielsweise Zeitstempel oder auch eine Referenz auf das zu überwachende Element (“*observable reference*”) enthalten.

Der Merkmalsspeicher steht so in gewisser Weise zwischen der Datenintegrationsschicht (“*data integration layer*”) und der Anwendungsschicht (“*application layer*”) (siehe Abbildung 8). Er bezieht seine Eingabedaten, welche für die Merkmalsberechnung herangezogen werden, durch das oben angesprochende Verteilungsnetzwerk (“*distribution network*”) in dem er sich für die Themen (“*topics*”) subskribiert, welche jene Daten zur Verfügung stellen, aus denen Merkmale berechnet werden sollen. Damit diese der vom Merkmalsspeicher geforderten Datenstruktur entsprechen, existiert ein entsprechende Interface (“*Featureable*”) welches von der Klasse, die das Event-Element repräsentiert implementiert werden muss, damit diese vom Merkmalsspeicher entgegengenommen werden kann und Merkmale aus dem entsprechenden Payload berechnet werden können. Die Anforderungen, welche das Interface stellt sind dabei lediglich eine Funktion, welche die entsprechenden Merkmaldaten zurückgiebt sowie eine weitere Funktion welche den Zeitstempel der Erzeugung des Merkmals zurückgibt, welcher wiederum für Merkmale mit Zeitkontext (z.B. Reampling) verwendet werden. Für den Fall, dass ein Merkmal registriert wird, bevor das entsprechende Submodell der Merkmalsquelle registriert wird, enthält der Merkmalspeicher ähnlich wie der Connector einen Model-Change-Listener um auf eine spätere Erstellung des entsprechenden Submodells mit der Merkmalsberechnung, speicherung bzw. -bereitstellung unmittelbar reagieren kann, ohne dass hierfür eine entsprechende Benachrichtigung durch den Nutzer vonnöten ist. Des weiteren wird das sematische Nachschlagen (“*semantic lookup*”) verwendet, um über die Referenz (“*observable reference*”) der Ereignisnachrichten jene Metadaten der entsprechenden Anlage (“*asset*”) abzurufen, die für die Merkmalsberechnung und -bereitstellung benötigt werden, wie beispielsweise die Abtastrate des Signals oder dessen physikalische Einheit.

Anwendungen welche nun Merkmale aus dem Merkmalsspeicher beziehen wollen, tun dies wie im Fall des Verteilungsnetzwerks in Form von Subskriptionen, allerdings erfolgen diese Subskriptionen nunmehr beim Merkmalsspeicher bzw. werden durch diesen verwaltet und beziehen sich auf bereits bestehende Merkmale bzw. müssen neue Merkmale in einem vorhergehenden Schritt zunächst angelegt werden. Werden die Rohdaten ohne jedwede Transformation oder Aggregation benötigt, können die Subskriptionen natürlich nach wie vor beim Verteilungsnetzwerk selbst erfolgen. Die Rückgabe der berechneten Merkmale hingegen kann allerdings wieder durch das Verteilungsnetzwerk in Form eines neuen Event-Elements erfolgen, sodass sich die Kommunikation einer Anwendung mit dem Merkmalsspeicher nicht wesentlich von jener mit dem Vereilungsnetzwerk unterscheidet. Andere Rückgabeformate, sowie eine (vorübergehende) Speicherung der Merkmale sind allerdings ebenfalls nötig, wie im weiteren Verlauf des Kapitels erläutert wird. Benötigt eine Machine-Learning- oder Data-Analytics-Anwendung Daten aus mehreren Quellen, müssen die entsprechenden Merkmale im Sinne eines Zeitreihenverbundes (“*time series join*”) bzw. einer Interpolation auf einen vorgegebenen Zeitvektor vereinigt werden. Dies ähnelt dem Konzept der Multi-Source-Streaming-App welche in D21 (System Design) vorgestellt wurde. Enthält eine einzelne Quelle Zeitreihen mehrere Signale kann dies ebenfalls von Seiten des Merkmalsspeichers entgegengenommen werden, sofern sich die Zeitstempel (und damit natürlich auch die Abtastraten) der einzelnen Signale nicht unterscheiden, also nur ein Vektor mit Zeitstempeln zurückgegeben wird.

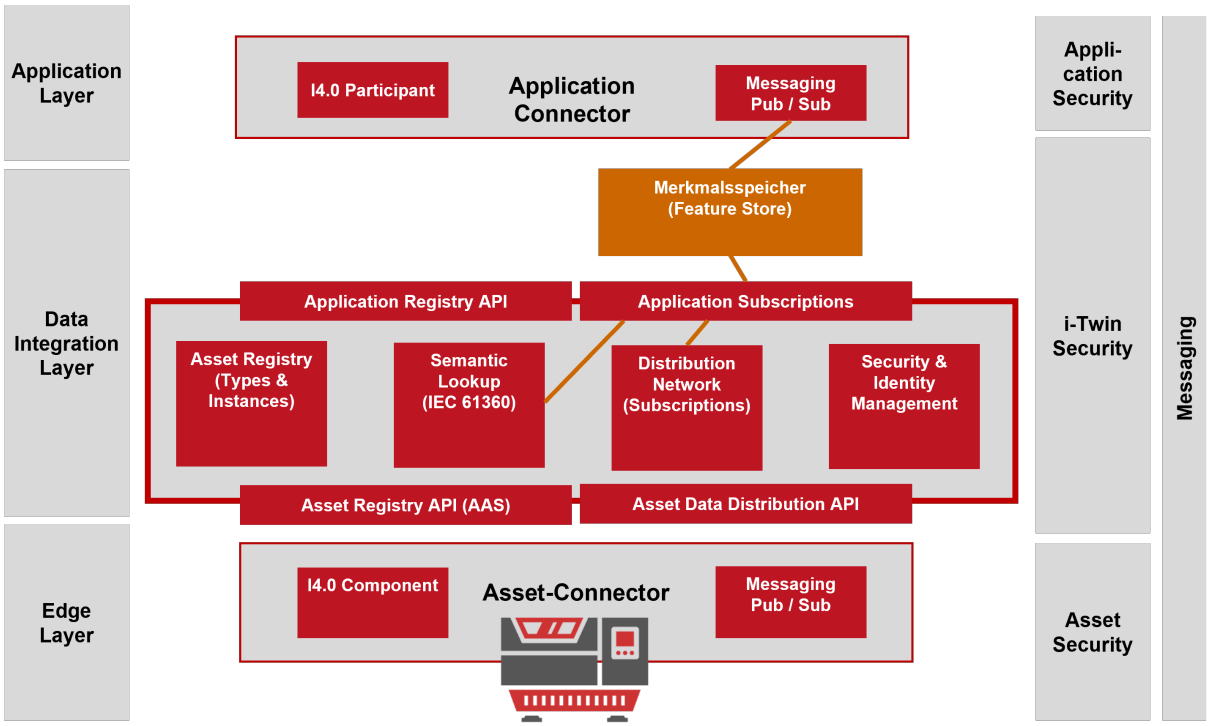

Abbildung 8: Einbindung des Merkmalsspeichers in die i-Twin-Plattform

4.3 Implementierungsansatz mit Apache Spark

Im Rahmen dieses Projektes soll der Merkmalsspeicher mithilfe von Apache Spark implementiert werden. Apache Spark ist eine einheitliche Plattform für Big-Data-Applikationen bzw. stellt zum gegenwärtigen Zeitpunkt die größte dieser Art dar und wird in naher Zukunft aller Voraussicht nach ein Grundpfeiler für Applikationen, welche die Verarbeitung großer Datenmengen ermöglichen bleiben. Spark ist in der Programmiersprache Scala programmiert, kompiliert allerdings ebenso wie Java auf der Java-Virtual-Machine (JVM), was eine einfache Integration in bestehende Java-Anwendungen ermöglicht. Spark stellt mehrere APIs zur Big-Data verarbeitung zur Verfügung und gewährleistet eine hochperformante Ausführung unabhängig davon, wie diese Schnitstellen miteinander kombiniert werden. Ebenfalls vereinfacht Spark die Skalierung einer Big-Data-Anwendung erheblich. Nachfolgend werden jene Komponenten von Spark vorgestellt, welche eine zentrale Rolle für die Implementierung des Merkmalsspeichers darstellen werden. Eine ausführliche Auflistung aller von von Spark zur Verfügung gestellten APIs findet sich in [Chambers and Zaharia 2018]:

- **Spark core:**
  Spark core stellt die Grundlage des Spark-Sytems dar und stellt dementsprechend als Treiberprozess Grundfunktionalitäten wie die Aufgabenanalyse, -planung und -verteilung an die Ausführungprozesse, Input-/Output usw. zur Verfügung. Die Interaktion zwischen dem Spark-Core-Treiber mit den Auswührungsprozessen wird im Unterkapitel zur Merkmalsberechnung näher erläutert. Spark Core übersetzt des weiteren den Ausführungscode in Scala bevor er an die Ausführungsprozesse weitergeleitet wird, wodurch es möglich wird, die eigentliche Aufgabendefinition in unterschiedlichen Programmiersprachen (neben Java bestehen auch Interfaces zu Python und R) zu verfassen. Die Grundlegende Datenstruktur, auf der Spark Core operiert sind sogenannte Resilient-Distributed-Datasets (RDDs), auf welche nahezu sämtliche Spark-Funktionalitäten aufbauen. Diese RDDs stellen einen Teilbestand von Daten dar, welcher über mehrere Rechner verteilt

sein können und auf welchen grundlegende Transformationen (filter, map, reduce, usw.) durchgeführt werden können. Diese Transformationen können als gerichteter azyklischer Graph verstandern werden. In der Regel werden die Tranformationsanweisungn von Seiten des Nutzers allerdings nicht auf RDDs direkt definiert sondern auf einer höheren Abstraktionsebene wie DataFrames welche im Folgenden noch vorgestellt werden.

- **Spark structured streaming:**
  Structured Streaming ermöglicht eine performante Verarbeitung und Transformation von Datenströmen aus verschiedenen Quellen wie Kafka, Flume, Web-Sockets etc.. Diese API teilt jene Ströme in einzelne Pakete auf und übersetzt bzw. optimiert Transformationen welche auf Ebene der jeweils genutzten Programmiersprache definiert wurden in RDD-Syntax. Des weiteren kümmert sich Structured Streaming um eine einfach zu bedienende bzw. teils sogar vollständig automatisierte Implementierung von Funktionen, welche bei der Verarbeitung von Datenströmen gemeinhin benötigt werden. Dazu gehören etwa Checkpointing, um bei einer Unterbrechung der Anwedung später an der korrekten Stelle des Datenstroms fortfahren zu können, sowie Late-Arriver-Handling um eine konsistente Merkmalsberechnung zu ermöglichen selbst wenn einzelne Datenpunkte verspätet eingelangen. Dies ist insbesondere nützlich, wenn Merkmale aus mehreren Datenquellen berechnet werden sollen, da es zu keinem unvollständigen Ausgabedatensatz kommt, falls eine der Quellen einzelne Datenpunkte verspätet liefert. Eine graphische veranschalichung des late-arriver-handlings ist in Abbildung 9 dargestellt.

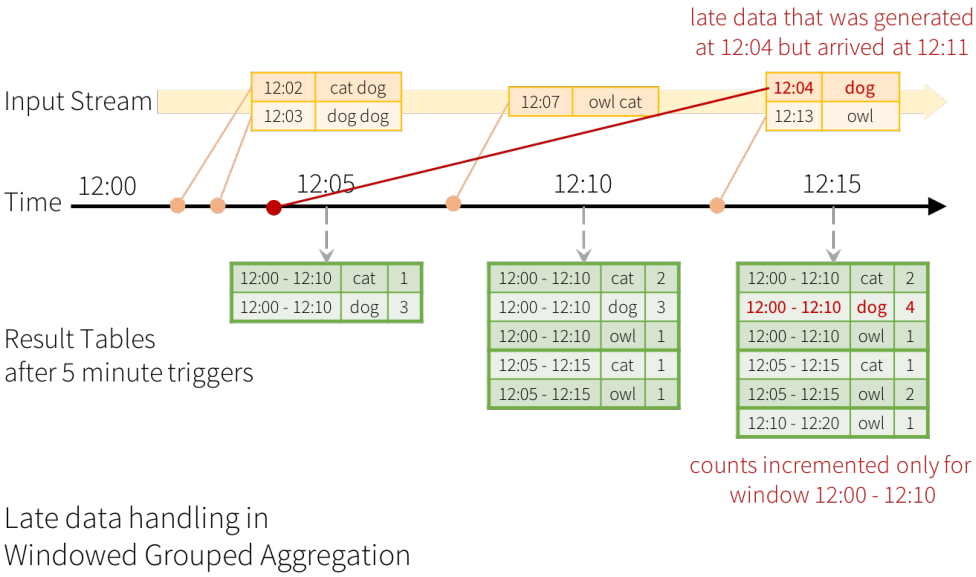

Abbildung 9: Schema des Late-Arriver-Handlings in Spark

- **Spark Dataframes und Datasets:**
  In der Regel werden Operaionen in Spark auf der DataFrame API definiert. Diese stellt Daten schlichthin in tabellarischer Form mit Zeilen und Spalten dar. Ähnlich wie in SQL-Datenbanken muss für die Spalten zuvor ein Schema definiert werden, welches deren Namen und Datentyp festlegt. Da DataFrames auf den zuvor erwähnten RDDs basieren kann eine solche Tabelle ebnfalls problemlos verteilt über mehrere Rechner existieren. Im Gegensatz zu RDDs können Transformationen auf DataFrames allerdings viel

einfacher Definier werden, Spark ermöglich sogar die Anwendung von SQL-Statements auf DataFrames, welche von Spark-core in die korrespondierenden RDD-Operationen übersetzt werden. Datases können wiederum als eine Erweiterung von DataFrames versanden werden, welche es einem ermöglichen, Java- bzw Scala-Klassen direkt in das Spark-Sysem zu integrieren und Operationen direkt auf Kollektionen dieser Klassen zu definieren, während sie von Spark als DataFrames behandelt werden und jderzeit auch als solche dargestellt werden können. Die entsprechende Klasse muss dafür in Java nach dem Java-Bean-Muster definiert sein bzw. als Case-Class in Scala.

Abbildung 10 stellt die zentralen Komponenten des Merkmalsspeichers sowie dessen Schnittstellen zur Definition, Verwaltung und Subskription von Merkmalen dar.

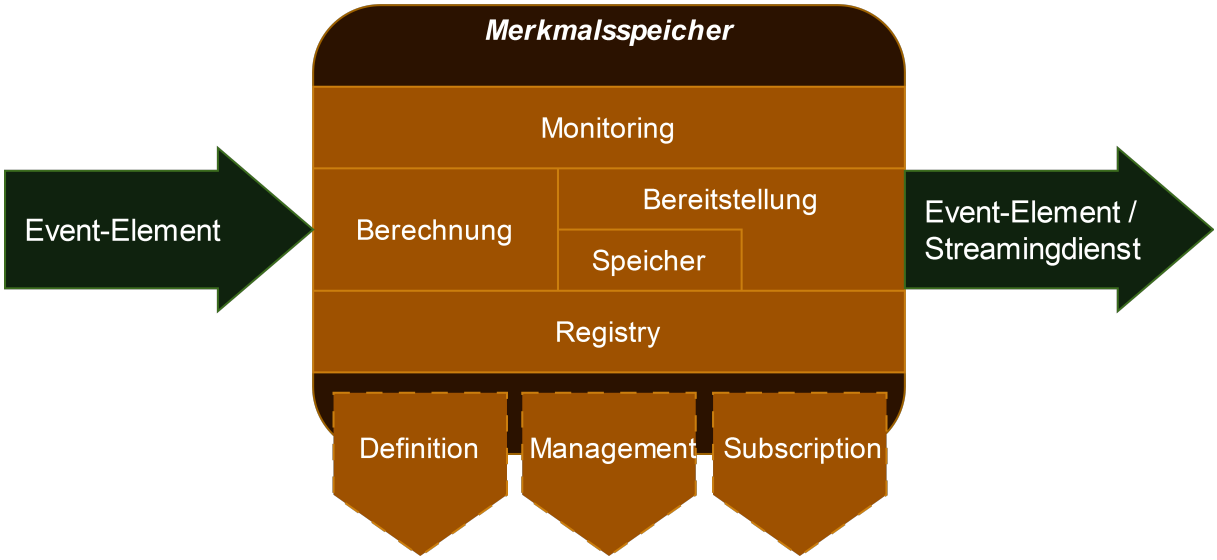

Abbildung 10: Grundstruktur des Merkmalsspeichers (links) bzw. einzelner Merkmale (rechts).

Nachfolgend werden diese zentralen Elemente des Merkmalsspeichers sowie Ansätze zu deren Umsetzung unter Verwendung von Apache Spark erläutert.

4.3.1 Merkmalsdefinition und -verwaltung

Mit Spark SQL bietet Spark eine weitere Funktionalität, die SQL-ähnliche Abfragen auf DataFrames, einer zentralen Datenstruktur in Apache Spark, ermöglicht. Zusätzlich ist es möglich, neben den von Spark zur Verfügung gestellten Funktionen eigene benutzerdefinierte Funktionen (“*user-defined functions*”, UDFs) zu registrieren und in späteren SQL-Abfragen zu verwenden [Luu 2021]. Hierdurch wird eine einfache Definition von Merkmalen ermöglicht, da SQL sehr weit verbreitet ist und vielfach verwendet wird. Derart definierte merkmale sind allerdings beschränkt, da sie keinen Zeinkontext beinhalten dürfen, wehalb SQL-Funktionen wie beispielsweise “*LAG*” nicht verwendet werden können. Temporale Merkmale werden von Spark gesondert behandelt da das zuvor erwähne Late-Arriver-handling angewandt wird. Deshalb muss eine zeitliche Obergrenze (in Spark als “*watermark*” bezeichnet) definiert werden, für welche Spark die Merkmalsberechnung zurückhält um auf Late-Arrivers zu warten. Ohne eine solche Watermark würde Spark die Zwischergebnisse der berechneten Merkmale ewig im Speicher behalten, da diese sich theoretisch jederzeit durch zu spät eingelangte Datenpunkte verändern könnten. Um dennoch eine für den Nutzer einfache Bedienbarkeit zu gewährleisten wird werden werden mehrere häufig verwendete zeiabhängige Merkmale bereits über einfach zu handhabende Funktionen zur Verfügung gestellt. Sollten diese nicht ausreichen, ist es

darüber hinaus möglich, solche Merkmale selbst zu definieren. Spark stellt dafür die Funktionen "*MapGroupWithState*" sowie "*FlatMapGroupWithState*"zur verfügung, deren Anwendung allerding etwas umständlicher ist als bereits vordefinierte Merkmale zu verwenden. Der Unterschied besteht darin, ob ein einzelner Eingabedatenpunkt auf einen oder potentiel mehrere (bzw. keinen) Ausgabedaenpunkte gemappt werden soll. Neben den vordefinierten temporalen Merkmalen werden auch Implementierungen für eine Reihe simplerer Merkmale zur Verfügung gestellt. Diese stellen zum Großteil Wrapper für Funktionen dar, welche Spark bereits standardmäßig anbietet, sodass sie im Rahmen des Merkmalsspeichermoduls verwendet werden können. Eine Auflistung aller seitens des Merkmalsspeichers zur Verfügung gestellten Merkmale findet sich in Tabelle 1. Diese ist nach in folgende Gruppen unterteilt:

- **Einfache Funktionen:** Hierbei handelt es sich um Funktionen, die lediglich auf Basis einer einzelnen Zeile bzw. eines einzelnen Wertes berechnet werden.
- **Aggregatsfunktionen:** Diese Funktionen berechnen globale Aggregeate einer Tabelle. Die Ausgaben der entsprechenden Pipeline beinhalten deshalb auch lediglich den aktualisierten Aggregatswert. Diese Funktionen können ebenfalls alle als Fenstefunktionen verwendet werden, sodass sie Fensterweise lokale Aggregate berechnen.
- **Multicol Funktionen:** Diese Funktionen beziehen sich ebenso wie einfache Funktionen nur auf einzelne Zeilen, beziehen allerdings den Input mehrer Spalten mit ein.
- **Signalverarbeitungsfunktionen:** Bei diesen Funktionen handelt es sich um in der Signalverarbeitung häufig verwendete Methoden, welche zu ihrere Berchnung zeitlichen Kontext benötigen.
- **Fensterfunktionen:** Diese Funktionen beziehen sich auf einen Teilausschnitt der Tabelle, bilden allerdings keine Aggregate, weshalb die Anzahl der Ausgabezeilen jener der Eingabezeilen entspricht. Zur Bildung von Fenstern gibt es zwei mögliche Funktionen:
  - `group_time(Integer winlen, Integer olap)`
    Bildet Fenster nach dem verfügbaren Zeitvektor welche eine länge von `winlen` Sekunden besitzen uns sich um `olap` Sekunden überlappen
  - `group_by(String colname)`
    Bildet Fenster auf basis der Werte welche der `colname` Spalte.
- **Hilfsfunktionen:** Diese Funktionen berechnen keine neuen Variablen sondern helfen dabei, berechnete Variablen zu verwalten.

Eine weitere Schnittstelle ermöglicht die Verwaltung bestehender Merkmale. Mittels dieser Schnittstelle soll es sowohl möglich sein, bestehende Merkmale zu durchsuchen als auch Merkmalsdefinitionen zu verändern, wodurch jeweils eine neue Version des betreffenden Merkmals angelegt wird (siehe Merkmalsregistratur).

4.3.2 Merkmalsregistratur

Merkmale werden als Pipelines registriert. Eine Pipeline wiederum besitzt drei Komponenten:

- **Quelle:** Die Datenquelle, welche die Rohdaten für das zu berechnende Merkmal liefert. Wie bereits erwähnt, wird die entsprechende Quelle über die "*observable Reference*" referenziert und sobald eine Pipline gestartet wird, wird ein ensprechendes Event-Element registriert. Ebenso muss ein Schema für den Datenstrom spezifiziert werden, da von Seiten des Event-Elementes lediglich Java-Object-Arrays übergeben werden. Eine Übersicht, wie die Datentypen von Spark mit jenen von Java zusammenhängen, findet sich in Tabelle 2.

Einfache Funktionen		
abs(String *spalte*)	*spalte*: Spaltenname	Berechnet den Absolutbetrag der Werte einer numerischen Spalte
acos(String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkuskosinus der Werte einer numerischen Spalte
acosh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkuskosinus-hyperbolicus der Werte einer numerischen Spalte
asin (String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkussinus der Werte einer numerischen Spalte
asinh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkussinus-hyperbolicus der Werte einer numerischen Spalte
atan (String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkustangens der Werte einer numerischen Spalte
atanh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Arkustangens -hyperbolicus der Werte einer numerischen Spalte
ceil (String *spalte*, Integer *decimals*)	*spalte*: Spaltenname, *decimals*: Dezimalstelle	Rundet den Wert einer numerischen Spalte auf *decimals* Dezimalstellen auf
cos (String *spalte*)	*spalte*: Spaltenname	Berechnet den Cosinus der Werte einer numerischen Spalte
cot (String *spalte*)	*spalte*: Spaltenname	Berechnet den Cotangens der Werte einer numerischen Spalte
cosh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Cosinus-hyperbolicus der Werte einer numerischen Spalte
date_add (String *spalte*, Integer*: days*)	*spalte*: Spaltenname, *days*: Zu addierende Tage	Addiert *days* Tage zu einer DateTime Spalte
date_format (String *spalte*, String *format*)	*spalte*: Spaltenname, format: DateTime Muster	Konvertiert einen DateTime Zeitstempel in das mit *format* spezifizierte Format, z.B. *dd.MM.yyyy* würde einen String der Form 15.01.2024 zurückgeben
date_sub (String *spalte*, Integer*: days*)	*spalte*: Spaltenname, *days*: Zu subtrahierende Tage	Subtrahi*days* Tage zu einer DateTime Spalte
day (String *spalte*)	*spalte*: Spaltenname	Extrahiert den Tag eines DateTime Zeitstempelt aus der spezifizierten Spalte
degrees (String *spalte*)	*spalte*: Spaltenname	Konvertiert einen in Radianten spezifizierten Winkel in den entsprechenden Wert in Grad.
exp (String *spalte*)	*spalte*: Spaltenname	Berechnet das Exponential der Werte der spezifizierten Spalte
factorial (String *spalte*)	*spalte*: Spaltenname	Berechnet das Factorial der Werte der spezifizierten Spalte
floor (String *spalte*, Integer *decimals*)	*spalte*: Spaltenname, *decimals*: Dezimalstelle	Rundet den Wert einer numerischen Spalte auf *decimals* Dezimalstellen auf
hours (String *spalte*)	*spalte*: Spaltenname	Extrahiert die Stunde eines DateTime Zeitstempelt aus der spezifizierten Spalte
intsr (String *spalte*, String *substring*)	*spalte*: Spaltenname, substring: Suchstring	Gibt die Position des ersten Vorkommens von *substring* zurück
isnan (String *spalte*)	*spalte*: Spaltenname	Gibt einen boolschen Wert zurück, welcher angibt ob der Wert eine Zeile NA ist
isnull (String *spalte*)	*spalte*: Spaltenname	Gibt einen boolschen Wert zurück, welcher angibt ob der Wert eine Zeile null ist
lcase (String *spalte*)	*spalte*: Spaltenname	Konvertiert alle Zeichen einer String-Spalte in Kleinbuchstaben
len (String *spalte*)	*spalte*: Spaltenname	Gibt die Anzahl an Zeichen einer String-Spalte zurück
ln (String *spalte*)	*spalte*: Spaltenname	Berechnet den natürlichen Logarithmus einer numerischen Spalte
log (String *spalte*, Integer *base*)	*spalte*: Spaltenname, *base*: Basis	Berechnet den Logarithmus einer numerischen Spalte zur angegebenen Basis
ltrim (String *spalte*)	*spalte*: Spaltenname	Entfernt Leerzeichen vom linken Ende einer String-Spalte
minute (String *spalte*)	*spalte*: Spaltenname	Extrahiert die Minute eines DateTime Zeitstempelt aus der spezifizierten Spalte
month (String *spalte*)	*spalte*: Spaltenname	Extrahiert den Monat eines DateTime Zeitstempelt aus der spezifizierten Spalte
pow (String *spalte*, Integer *exp*)	*spalte*: Spaltenname, *exp*: Exponent	Berechnet die Potenz zur Basis einer numerischen Spalte mit dem Exponenten *exp*
radians (String *spalte*)	*spalte*: Spaltenname	Konvertiert einen in Grad spezifizierten Winkel in den entsprechenden Wert in Radianten.
regexp_extract (String *spalte*, String *exp*)	*spalte*: Spaltenname, *exp*: Regex-Ausdruck	Extrahiert den Regex-Ausdruck *exp* aus einer String-Spalte
regexp_like (String *spalte*, String *exp*)	*spalte*: Spaltenname, *exp*: Regex-Ausdruck	Gibt einen boolschen Wert zurück ob der Regex-Ausdruck *exp* in einer String-Spalte vorkommt
regexp_extract (String *spalte*, String *exp*,String *replacement*)	*spalte*: Spaltenname, *exp*: Regex-Ausdruck, *replacement*: Ersatz-String	Ersetzt den Regex-Ausdruck *exp* aus einer String-Spalte durch *replacement*
reverse (String *spalte*)	*spalte*: Spaltenname	Kehrt die Zeichenfolgen einer String-Spalte um
round (String *spalte*, Integer *decimals*)	*spalte*: Spaltenname	Rundet den Wert einer numerischen Spalte auf *decimals* Dezimalstellen.
sec (String *spalte*)	*spalte*: Spaltenname	Extrahiert die Minute eines DateTime Zeitstempelt aus der spezifizierten Spalte
shiftleft (String *spalte*, Integer *numBits*)	*spalte*: Spaltenname, *numBits*: Anzahl Bits	Versetzt die Werte einer numerischen Spalte um *numBits* nach links
shiftright (String *spalte*,Integer *numBits*)	*spalte*: Spaltenname, *numBits*: Anzahl Bits	Versetzt die Werte einer numerischen Spalte um *numBits* nach rechts
sign (String *spalte*)	*spalte*: Spaltenname	Wandelt eine numerische Spalte in einen boolschen Ausdruck um der das Vorzeichen des Wertes angibt (true: positives Vorzeichen, false: negatives Vorzeichen)
sin (String *spalte*)	*spalte*: Spaltenname	Berechnet den Sinus der Werte einer numerischen Spalte
sinh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Sinus-hyperbolicus der Werte einer numerischen Spalte
sqrt (String *spalte*, Integer *n*)	*spalte*: Spaltenname, *n*: Wurzelexponent	Berechnet die *n*-te Wurzel einer numerischen Spalte
substr (String *spalte*, Integer *pos*,Integer l*en*)	*spalte*: Spaltenname, *pos*: Startposition *len*: Stringlänge	Gibt den Teilstring einer String-Spalte zurück, welcher an *pos* Position beginnt und Länge *len* hat
tan (String *spalte*)	*spalte*: Spaltenname	Berechnet den Tangens der Werte einer numerischen Spalte
tanh (String *spalte*)	*spalte*: Spaltenname	Berechnet den Tangens-hyperbolicus der Werte einer numerischen Spalte
ucase (String *spalte*)	*spalte*: Spaltenname	Konvertiert alle Zeichen einer String-Spalte in Großbuchstaben
weekday (String *spalte*)	*spalte*: Spaltenname	Extrahiert den Wochentag eines DateTime Zeitstempelt aus der spezifizierten Spalte
year (String *spalte*)	*spalte*: Spaltenname	Extrahiert das Jahr eines DateTime Zeitstempelt aus der spezifizierten Spalte
Aggregatsfunktionen		
avg (String *spalte*)	*spalte*: Spaltenname	Gibt den Durchschnittswert einer numerischen Spalte zurück
count (String *spalte*)	*spalte*: Spaltenname	Gibt die Anzahl an Zeilen in einer Spalte zurück
count_distinct (String *spalte*)	*spalte*: Spaltenname	Gibt die Anzahl an Zeilen in einer Spalte mit einzigartigen Werten zurück
count_if (String *spalte*)	*spalte*: Spaltenname	Gibt die Anzahl an Zeilen in einer boolschen Spalte zurück welche WAHR-Werte enthalten
every (String *spalte*)	*spalte*: Spaltenname	Gibt einen boolschen Wert zurück der angibt, ob alle Zeilen einer boolschen Spalte WAHR sind
first (String *spalte*)	*spalte*: Spaltenname	Gibt den ersten Wert einer Spalte zurück
kurtosis (String *spalte*)	*spalte*: Spaltenname	Berechnet die Kurtosis (4. standartisierter Moment) der Werte einer numerischen Spalte
last (String *spalte*)	*spalte*: Spaltenname	Gibt den letzten Wert einer Spalte zurück
max (String *spalte*)	*spalte*: Spaltenname	Gibt den Maximalwert einer numerischen Spalte zurück
median (String *spalte*)	*spalte*: Spaltenname	Gibt den Medianwert einer numerischen Spalte zurück
min (String *spalte*)	*spalte*: Spaltenname	Gibt den Minimumswert einer numerischen Spalte zurück
mode (String *spalte*)	*spalte*: Spaltenname	Gibt den Modalwert einer numerischen Spalte zurück
not (String *spalte*)	*spalte*: Spaltenname	Invertiert die Werte einer boolschen Zeile
percentile (String *spalte*, Floatarray *percentage*)	*spalte*: Spaltenname, *percentage:* Prozentwert	Berechnet die Perzentile einer numerischen Spalte für die angegebenen Prozentwerte (angegeben als Anteilswert im Bereich [0.0, 1.0]
product (String *spalte*)	*spalte*: Spaltenname	Gibt das Produkt der Werte einer numerischen Spalte zurück
skewness (String *spalte*)	*spalte*: Spaltenname	Berechnet die Schiefe (zentraler Moment 3. Ordnung) der Werte einer numerischen Spalte
some (String *spalte*)	*spalte*: Spaltenname	Gibt einen boolschen Wert zurück der angibt, ob mindestens eine Zeile einer boolschen Spalte WAHR ist
std (String *spalte*)	*spalte*: Spaltenname	Berechnet die Standardabweichung der Werte einer numerischen Spalte
sum (String *spalte*)	*spalte*: Spaltenname	Gibt die Summe der Werte einer numerischen Spalte zurück

Multicol Funktionen		
concat (Stringarray *spalten)*	*spalten:* Spaltennamen	Concatiniert die angegebenen String-Spalten zu einer Spalte
contains (String *right*, String *left*)	*right.* rechter Spaltenname, *left*: linker Spaltenname	Prüft ob der ausdruck der linken String-Spalte in der rechten String-Spalte enthalten ist und gibt entsprechend einen boolschen Wert zurück
corr (String *right*, String *left*)	*right.* rechter Spaltenname, *left*: linker Spaltenname	Berechnet den Pearson Korrelationskoeffizien zwischen der linken und rechten Spalte
date_diff (String *right*, String *left*)	*right.* rechter Spaltenname, *left*: linker Spaltenname	Gibt die Anzahl an Tagen zurück die zwischen *right* und *left* liegen
make_date (String *year*, String *month*, String: *day*)	*year.* Jahres-Spaltenname, *month*: Monats-Spaltenname, *day:* Tages-Spaltenname	Vereint drei Spalten welche Jahr, Monat und Tag enthalten zu einer DateTime Spalte
try_divide(String *dividend*, String *divisor*)	*dividend*: Dividend-Spalte *divisor*: Divisor-Spale	Berechnet *divided* / *divisor.* Gibt null zurück falls die Werte der divisor-Spalte 0 sind
Signalverarbeitungsfunktionen		
filter (String *spalte*, String *method*, Integer *srate*, Integer *lower_bound*, Integer *upper_bound*, Integer *order)*	*spalte*: Spaltenname, *method*: Filtermethode (lowpass", "highpass", "bandpassöder "bandstopp" *srate*: Abtastrate des Signals *lower_bound*: Untergrenze des Filters in Hz (-1 oder null falls nicht benötigt) *upper_bound*: Obergrenze des Filters in Hz (-1 oder null falls nicht benötigt) *order*: Filterordnung	Wendet einen Butterworth Filter auf die Werte einer numerischen Spalte an.
resample (String *spalte*, String *method*, Integer *srate*, Integer *srate_new*)	*spalte*: Spaltenname, *method*: Resampling-Methode (entweder linearöder βpline") *srate*: Ausgangs-Abtastrate *srate_new*: Gewünschte Abtastrate	Führt ein Resampling der Werte einer numerischen Spalte auf die angegebene Abtastrate durch
Fensterfunktionen		
dense_rank (String *spalte*)	*spalte*: Spaltenname	Gibt den Rang jedes Element einer numerischen Zeile innerhalb eine Fensters zurück
nth_value (String *spalte*, Integer *offset*)	*spalte*: Spaltenname, *offset*:Versatz (gezählt ab 1)	Gibt den Wert innerhalb jedes Fensters zurück, der *offset* Zeilen vom Fensterbeginn entfernt liegt.
precent_rank (String *spalte*)	*spalte*: Spaltenname	Gibt den relativen Rang jedes Element einer numerischen Zeile innerhalb eine Fensters zurück.
rank (String *spalte*)	*spalte*: Spaltenname	Gibt den Rang jedes Element einer numerischen Zeile innerhalb eine Fensters zurück. Im Gegensatz zu dense_rank wird bei gleichen Werten derselbe Rang vergeben.
row_number (String *spalte*)	*spalte*: Spaltenname	Gibt die Anzahl an Zeilen innerhalb jedes Fenters zurück
Versatz		
lag (String *spalte*, Integer*: offset*)	*spalte:* Spaltenname,*offset:* Versatzdistanz	Gibt den Wert zurück, der *offset* Zeilen vor der aktuellen Zeile liegt
diff (String *spalte*)	*spalte*: Spaltenname	Berechnet die Differens einer numerischen Spalte zum vorhergehenden Wert
Hilfsfunktionen		
rename (String *spalte*, String *name*)	*spalte:* Spaltenname, *name:* neuer Spaltenname	Benennt eine existierende Spalte um
transmute (Funktion *fn*, String *name*)	*fn:* Simple Funktion, Fensterfunktion, Versatzfunktion oder Signalverarbeitungsfunktion, *name*; Spaltenname	Bewirkt, dass durch Anwendung von *fn* eine neue Spalte mit dem Namen *name* erzeugt wird, anstatt eine bestehende Spalte zu modifizieren

Tabelle 1: Auflistung der bereitgestellten Merkmalsfunktionen

- **Transformationen:** Nach Spezifikation der Quelle können eine oder mehrere Transformationen registriert werden, welche das Ausgabemerkmal berechnen. Transformationen bestehen entweder aus vorgefertigten Funktionen (siehe Tabelle 1), zeitunabhängigen SQL-Statements oder nutzerseitig registrierter Transormationen. Ebenfalls können auf diese Ebene mehrere Pipelines verbunden werden. Hierfür kann entweder eine neue Pipeline erstellt werden, welche eine andere Pipeline als Eingabe hat oder mehrer bestehende Piplines über einen Zeitreihenjoin zusammengeführt werden. Dabei muss allerdings ähnlich wie bei zeitabhängigen Merkmalen eine Watermark spezifiziert werden welche angibt, wie lange auf einen Late-Arriver einer der beiden Pipelines gewartet wird bevor dieser ignoriert wird. Das Zusammenführen mehrere Pipelines ist schematisch in Abbildung 11 dargestellt.
- **Sink:** Sinks stellen das Ausgabemedium einer Pipeline dar. Diese werden im Unterkapitel Merkmalsbereitstellung und Persistenz diskutiert.

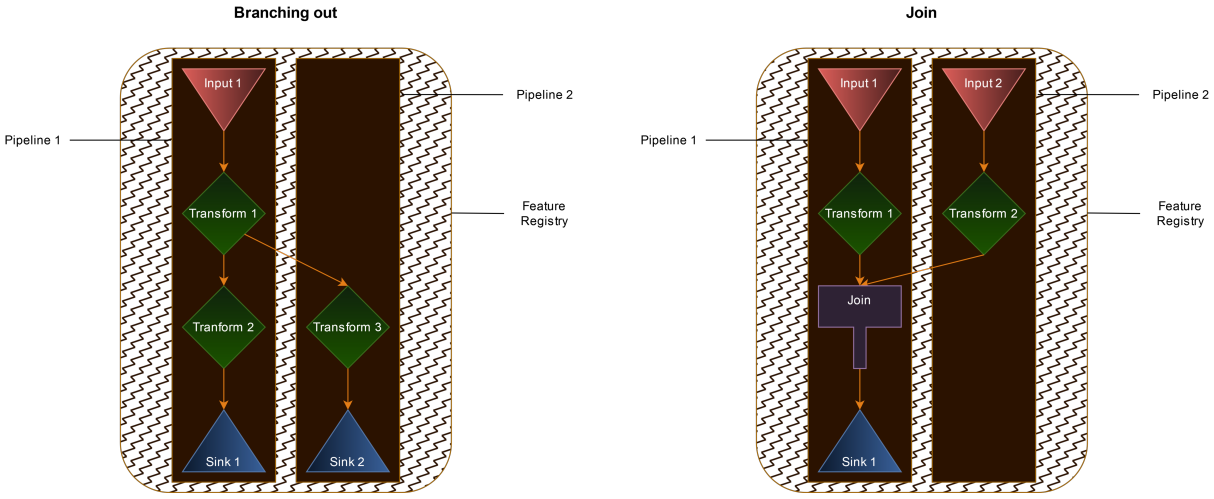

Abbildung 11: Schematische Struktur der Piplies bzw. der Möglichkeiten zum Zusammenführen mehrerer Pipelines

Zu jeder Pipeline werden außerdem Metadaten gespeichert, welche vor allem eine transparente Definition, vereinfachte Verwaltung, konsistente Berechnung sowie Unabhängigkeit vom datengenerierenden Prozess gewährleisten. Metadaten wie beispielsweise die Pipelinelemente, die Versionsnummer sowie Bezeichner für jene Anwendungen, welche sich für ein spezifisches Merkmal subskribiert haben erleichtern die Verwaltung und Berechnung einzelner Merkmale. Ein aussagekräftiger Name sowie eine optionale kurze Beschreibung des Merkmals erleichtert die Dokumentation des Merkmals und vereinfacht die Verwendung durch andere Nutzer. Sollen bestimmte Merkmale in einem persistenten Speicher abgelegt werden — beispielsweise um als Trainingsdatensatz für Machine-Learning-Modelle zu dienen — muss zudem (bei der Registrierung des Sinks) der Zeitabschnitt angegeben werden, für welchen die entsprechenden Merkmale gespeichert werden sollen.

Spark Datentyp	Java Datentype	Java-API zur Verwendung des Datentyps
ByteType	*byte* oder *Byte*	org.apache.spark.sql.types,ByteType
ShortType	*short* oder *Short*	org.apache.spark.sql.types.ShortType
IntegerType	*int* oder *Integer*	org.apache.spark.sql.types.IntegerType
LongType	*long* oder *Long*	org.apache.spark.sql.types.LongType
FloatType	*float* oder *Float*	org.apache.spark.sql.types.FloatType
DoubleType	*double* oder *Double*	org.apache.spark.sql.types.DoubleType
DecimalType	*java.math.BigDecimal*	org.apache.spark.sql.types.createDecimalType() org.apache.spark.sql.types.createDecimalType(precision, scale)
StringType	*String*	org.apache.spark.sql.types.StringType
BinaryType	*byte[]*	org.apache.spark.sql.types.BinaryType
BooleanType	*boolean* oder *Boolean*	org.apache.spark.sql.types.BooleanType
TimestampType	*java.sql.Timestamp*	org.apache.spark.sql.types.TimestampType
DateType	*java.sql.Date*	org.apache.spark.sql.types.DateType

Tabelle 2: Zusammenhang zwischen Spak und Java-Datentypen

4.3.3 Merkmalsberechnung

Spark ermöglicht generell eine parallelisierte Datenverarbeitung, somit können bei entsprechendem Bedarf und Verfügbarkeit von Ressourcen auch komplexe Merkmale in geringer Zeit berechnet werden. Diese Parallelsierung kann wie bereits erwähnt lokal durch die Verwendung mehrere Threads oder aber auch durch die Aufgabenverteilung auf mehrere Rechner geschehen. Zeitabhängige Transormationen verlangsamen diesen Prozess daher verständlicherweise da sie diese Paralellisierung einschränken. Eine weitere Konsequenz ist, dass alle in die Merkmalsberechnung involvierten Java-Klassen serialisierbar sein, also das "*Serializable*"-Interface implementieren müssen. Wie zu Eingang des Kapitels erwähnt, können Spark Datasets auf bestehenden Java-Klassen aufgebaut werden, die zentrale Klasse, auf welcher die Merkmale berechnet werden ist eine als "*LineWithTimestamp*" bezeichnete Java-Bean Klasse, welche einen Zeitstempel und ein oder mehrere Merkmale enthalten, deren Datentyp durch das im vorangegangenen Unterkapitel erwähnte Schema spezifiziert wird. Die parallelisierung selbst wird von Spark dadurch erreicht, dass Spark-Applikationen aus einem Treiberprozess ("*drivers*") bestehen, der Aufgaben analysiert und an mehrere Ausführungsprozesse ("*executors*") überträgt, sowie die Kommunikation mit dem Nutzer übernimmt. Der Treiber Prozess übersetzt den Nutzercode in Spark Code und überträgt diesen zur Ausführung an jene Anzahl an Ausführungsprozessen die vom Cluster Manager erfasst wurden. Der Cluster Manager überwacht die vorhanden Reourcen innerhalb des Clusters und kann daher auf zusätzlich hinzugekommene bzw. ausfallende Nodes reagieren, sodass der Nutzer die Clusterresourcen für die Spark-Applikation nicht selbst verwalten muss. Diese Architektur ist in Abbildung 12 näher veranschaulicht.

4.3.4 Merkmalsbereitstellung und Persistenz

Die Ausgabe des Merkmalsspeichers sollen Datensätze in einem einheitlichen Format sein, welche von nachfolgenden Anwendungen einfach weiterverarbeitet werden können. Insbesondere für Machine-Learning-Anwendungen bedeutet dies, dass die resultierende Zeitreihe äquidistant abgetastet sein soll und eventuell Merkmale mit unterschiedlichen zeitlichen Auflösungen zusammengeführt werden. Die Überführung der (möglicherweise nicht äquidistanten und mit unterschiedlicher Frequenz abgetasteten) Zeitreihen der betreffenden Merkmale in einen Datensatz mit äquidistanten Zeitabständen kann mithilfe der bereits erwähnten "*resample*"-Funktion erreicht werden. Für die Ausgabe der berechneten Merkmale selbst stehen sowohl kontinuierliche Ausgabemöglichkeiten also auch Optionen zur persistenten Speicherung zur Verfgügung. In der Regel wird der Merkmalsspeicher kontinuierliche Datenströme ("*data streams*") produzieren, welche an Anwendungen weitergegeben werden, welche sich

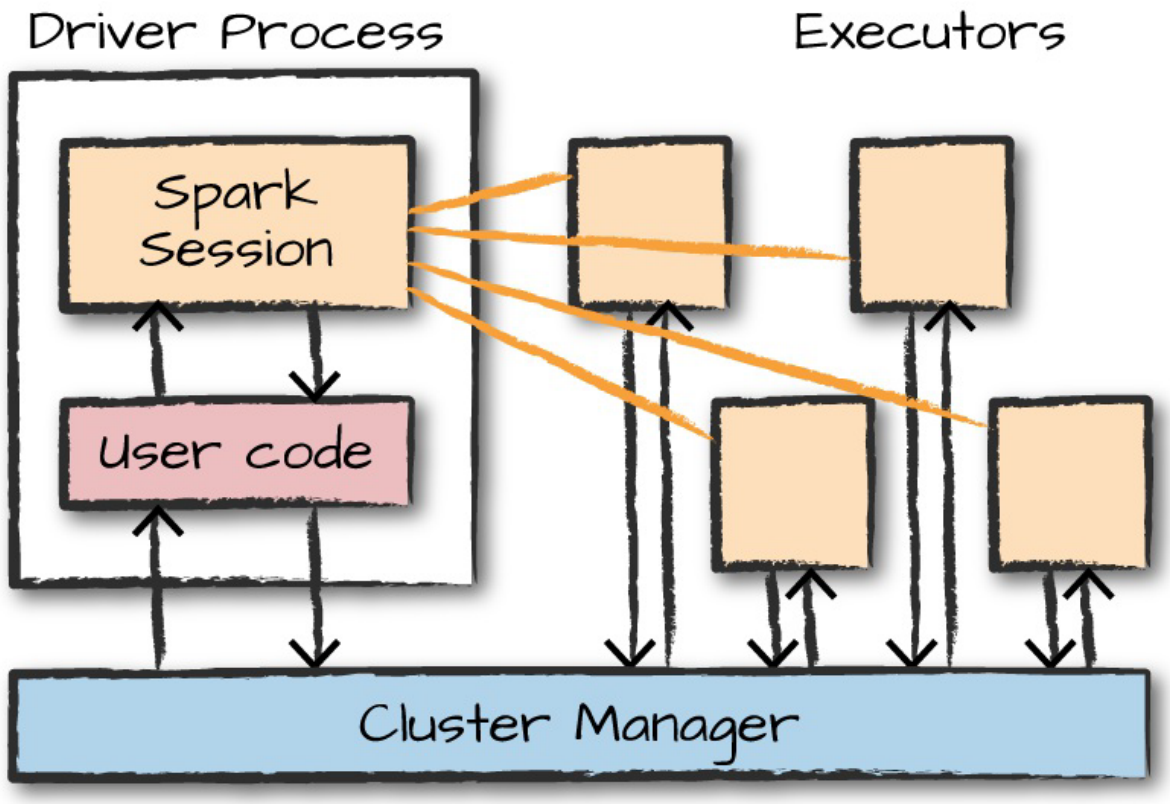

Abbildung 12: Spark Architektur

für entsprechenden Merkmale subskribiert haben. Hierfür kann als Sink (siehe Unterkapitel zur Merkmalsregistratur) sowohl ein Dienst für Datenstromverarbeitung (unterstützt werden Kafka und Redis Stream) als auch die Ausgabe als Event-Elemente benutzt werden. Insbesondere für das Training von Machine-Learning-Modellen wird es auch nötig sein, die erzeugten DataFrames zumindest für einen gewissen Zeitraum persistent abzuspeichern. Spark selbst stellt lediglich die Funktionseinheit zur Merkmalsberechnung selbst zur Verfügung, verfügt also über kein eigenes Filesystem, die Structured Streaming Applikation von Spark unterstützt allerdings Export von DataFrames in mehrere Dateiformate, was in der Regel ohne vorangegangene Transformation geschehen kann, da DataFrames ja wie erwähnt bereits eine tabellarische Form aufweisen. Die unterstützten Dateiformate für einen solchen Export sind CSV, JSON, ORC und Parquet. Zuletzt sollen natürlich auch die Nutzerseitig registrierten Merkmale bzw. die erstellen Pipelines gespeichert und zur Laufzeit verfügbar gemacht werden. Hierfür wird MongoDB verwendet, welche von der iTwin Plattfform auch für die Speicherung der erstellten Teilmodelle benutzt wird.

4.3.5 Merkmalsüberwachung

Wie bereits Eingangs erwähnt, ist es insbesondere für Machine-Learning-Anwendungen entscheidend, dass eine sogenannte Konzeptverschiebung (“*concept drift*”), also eine substantielle Veränderung des datengenerierenden Prozesses, frühzeitig erkannt wird. Machine-Learn-

ing-Modelle werden üblicherweise anhand historischer Daten trainiert und können deshalb die im Rahmen des Trainingsprozesses erzielte Genauigkeit auch nur so lange aufweisen, solange es zu keiner solchen Konzeptverschiebung kommt. Das Modell muss dann häufig an die Veränderung des datengenerierenden Prozesses durch Nach- oder Neutrainieren angepasst werden. Eine solche Konzeptverschiebung kann formal definiert werden als eine Veränderung der gemeinsamen Wahrscheinlichkeitsverteilung des Merkmals oder der Menge von Merkmalen X (beispielsweise Sensorwerte einer Produktionsmaschine) und der Zielvariable y (beispielsweise die Wartungsbedürftigkeit einer Produktionsmaschine) über einen bestimmten Zeitraum hinweg

$$P_t(X, y) \neq P_{t+1}(X, y)$$

Drückt man diese gemeinsame Wahrscheinlichkeitsverteilung als Produkt der A-priori-Wahrscheinlichkeit der Merkmalsdaten und der bedingten Wahrscheinlichkeit der Zielvariablen aus, so wird deutlich, dass sich eine solche Konzeptverschiebung auf zwei zugrundeliegende Ursachen zurückführen lässt.

$$P_t(X, y) = P_t(X) \times P_t(y|X)$$

Eine Veränderung der bedingten Wahrscheinlichkeit der Zielvariablen zu überwachen, ist üblicherweise beim Einsatz eines Machine-Learning-Modells in einer Produktionsumgebung nicht möglich, da es in der Regel nicht möglich ist, die Ausprägung der Zielvariable konstant zu erheben. Daher beschränkt man sich üblicherweise darauf, die A-priori-Verteilung der Eingangsvariablen des Machine-Learning-Modells zu überwachen: Eine Veränderung dieser Verteilung wird häufig als Eingabeverschiebung (“*input shift*”) oder Merkmalsverschiebung (“*feature drift*”) bezeichnet und ist graphisch in Abbildung 13 veranschaulicht.

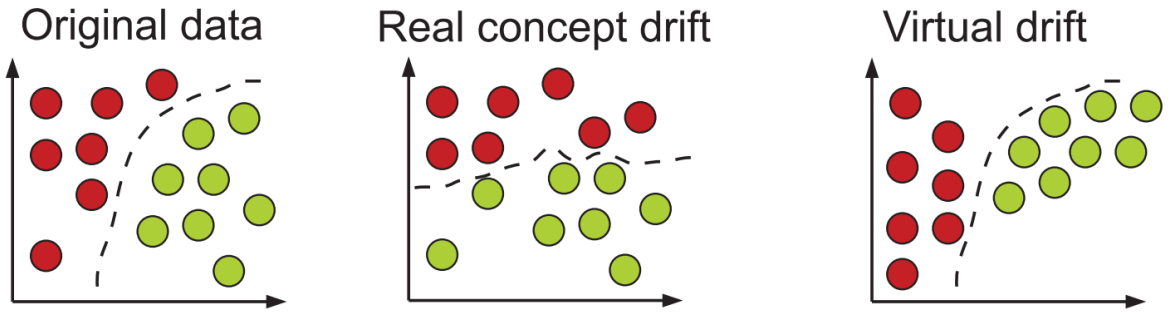

Abbildung 13: Veranschaulichung einer Eingabeverschiebung

Die rechte Abbildung stellt einen Sonderfall einer solchen Eingabeverschiebung dar, bei welcher sich die A-priori-Verteilung der Merkmale zwar verändert, die Modellgenauigkeit allerdings dennoch gleich oder zumindest ähnlich bleibt; ein solcher Fall wird üblicherweise als virtuelle (Eingabe-)Verschiebung (“*virtual input drif*”) bezeichnet [Gama *et al.* 2014].

Generell existiert eine große Bandbreite unterschiedlicher Methoden, um den Zeitpunkt einer solchen Verschiebung (auch als Änderungspunkt — “*change point*” — bezeichnet) möglichst frühzeitig zu erkennen, welche meist als Change-Point-Detection-Algorithmen bezeichnet werden. Diese Methoden können vor allem anhand der folgenden Kriterien eingeteilt werden.

- **Offline- und Online-Methoden:** Bei Offline-Methoden wird der Algorithmus nach Erhalt des vollständigen Eingabedatensatzes auf diesen angewandt, wohingegen bei Online-Methoden der Algorithmus gewissermaßen mit dem datengenerierenden Prozess “mit-

läuft" indem dieser inkrementell auf neue Datenpunkte des datengenerierenden Prozesses angewandt wird, sobald diese zur Verfügung stehen. (Vergleiche hierzu die mitlaufende Sensordatenkomprimierung in Abschnitt 3.)

- **Überwachte und Unüberwachte Methoden:** Bei überwachten ("*supervised*") Methoden sind sowohl die Merkmalsdaten als auch die gewünschten Ergebnisse des Modells bereits im voraus bekannt und es wird versucht, eine möglichst akkurate Abbildung von den Eingabedaten auf diese gewünschten Ergebnisse zu erzeugen. Dahingegen sind bei unüberwachten ("*unsupervised*") Modellen nur die Merkmalsdaten bekannt und es wird versucht, relevante Muster in diesen zu ermitteln.

- **Parametrische und nichtparametrische Modelle:** Bei parametischen Modellen ist die Modellstruktur im Vorhinein festgelegt, wohingegen sie bei nichtparametrischen Modellen aus den Daten selbst bestimmt wird.

Die oben aufgeführten Kategorisierungen verdeutlichen bereits, dass nur eine Teilmenge der existierenden Change-Point-Detection-Algorithmen für die vorliegende Problemstellung geeignet sein wird. Zum einen müssen die erhaltenen Daten mit möglichst geringer zeitlicher Verzögerung analysiert werden, was einen Online-Algorithmus voraussetzt. Desweiteren wird es in der Regel nicht möglich sein, Vorannahmen über alle möglichen Veränderungen, die sich im datengenerierenden Prozess ergeben können, zu treffen, was die Anwendung von Modellen des überwachten Lernens schwer möglich macht, da diese mithilfe historischer Daten lediglich auf das Erkennen bestimmter Arten von Veränderungen des datengenerierenden Prozesses trainiert werden können. Innerhalb der unüberwachten Algorithmen haben sich zudem vor allem die nichtparametrischen Modelle bewährt, da diese in der Regel sowohl eine höhere Genauigkeit als auch geringere Zeitkomplexität in der Ausführung aufweisen (siehe dazu z.B. [Aminikhanghahi and Cook 2017]).

Eine Übersicht über verschiedene Online-Algorithmen findet sich in [Namoano *et al.* 2019], eine zusammenfassende Auflistung ist in Tabelle 3 dargestellt. Viele der Algorithmen setzen allerdings vorraus dass beereits im Vorfeld bekannt is, welche Art der Veränderung ermittelt werden soll (bspw. Veränderungen des Mittelwertes, der Standardabweichung usw.). Hiervon kann für die Anwendung im Rahmen des Merkmalsspeichers allerdings nicht ausgegangen werden, da nur solche Veränderungen relevant sind, welche eine derartige Veränderung des eines oder mehrere Merkmale ("*feature drift*") bewirken, sodass diese nicht mehr mit den Trainingsdaten vergleichbar sind und somit von Seiten des Machine-Learning Modells mit diesen keine sinnvolle Vorhersage getroffen werden kann.

Category	Methods	P/NP	Inc	Time complexity	Limitation	Time Granularity	Robustness
Unsupervised	Mean and standard deviation threshold based methods	P	✔	$\mathcal{O}(n)$	NL	μsec	✔
	Cumulative sum	NP	✔	$\mathcal{O}(n^2)$	NL	μsec	
	Auto-regressive models	P	✔	$\mathcal{O}(n^3)$	ST/NL	sec	
	Martingale tests	P	✔	$\mathcal{O}(Kn^2)$	ST	sec	✔
	t-digest (Streaming percentile based detection)	NP		$\mathcal{O}(n^2)$	ST	μsec	
	Kullback-Leibler importance estimation procedure	P/NP		$\mathcal{O}(n^2)$	P = NST NP = NL		
	Online Bayesian probabilities methods	P	✔	$\mathcal{O}(n)$	i.i.d		
	Wavelet Transform	NP			NL		
	Fast Fourier Transform	NP			NL		
	Kernel Change Point	NP		$\mathcal{O}(n^3)$	i.i.d		
	Sliding Window and Bottom-up	P		$\mathcal{O}(Kn)$	NL	msec	
	Model fitting	P	✔		NL	sec	
Supervised	Hidden Markov Models	P	✔	TC	NL		
	K-Nearest Neighbor	NP		TC	NL		
	Support Vector Machine	P		TC	NL		✔
	Logistic Regression	P	✔	TC	NL		

Tabelle 3: Übersicht über Online-Point-Detection-Verfahren nach [Namoano *et al.* 2019].
P/NP: Parametric/Non-Parametric, Inc.: can be incremental, TC: Training Cost, NL: No Limitation, ST/NST: Stationary/Non-Stationary

Ausgehend von dieser Einschränkung erscheinen die Martingale Tests als geeignetste Variante zur Implementierung einer Change Point Detection. Dieser Test überprüft die Austauschbarkeit und damit die Gleichverteiltheit einer Sequenz von Zufallsvariablen anhand eines "Seltsamkeitsmaßes" welches der Anwender spezifiziert. Eine Sequenz $\{Z_i : 1 \leq i < \infty\}$ gilt dann als austauschbar, wenn die gemeinsame Verteilung unter jeder Permutation der Indexe der Zufallsvariablen gleich bleibt:

$$p(Z_1, Z_2, ..., Z_n) = p(Z_{\pi(1)}, Z_{\pi(2)}, ..., Z_{\pi(n)})$$

wobei π hier für alle möglichen Permutationen der Indexe $\{1, ..., n\}$ steht.

Austauschbarkeit und Change-Point Detection hängen dabei wie folgt zusammen:
Sei $\{Z_i : 1 \leq i \leq n\}$ eine Sequenz von Zufallsvariablen mit bedingter Wahrscheinlichkeitsdichte $p_\theta(Z_i|Z_1, ..., Z_{i-1})$. Eine Veränderung des Datengenerierenden Prozesses kann angesehen werden als Veränderung des Parameters λ von $\lambda = \lambda_0$ zu $\lambda = \lambda_1$ zum Zeitpunkt t_0. Wenn, wie zuvor erwähnt, die Zufallsvariablen $Z_1, ..., Z_i$ unabhängig verteilt sind, gilt $p_\theta(Z_i|Z_1, ..., Z_{i-1}) = p_\theta(Z_i)$. Verändert sich θ beim Datenpunkt j zwischen 1 und n, führt dies zu einer Verletzung der Austauschbarkeit der Datenpunkte der gemeinsamen Verteilung

$$p_{\theta_0^{i-1} \times \theta_1^{n-j+1}}(Z_1, ..., Z_{j-1}, Z_j, Z_{j+1}, ..., T_{n-1}, Z_n) = p_{\theta_0}(Z_1)...p_{\theta_0}(Z_{j-1}p_{\theta_1}(Z_j)...p_{\theta_1}(Z_n)$$

Die Austauschbarkeit wird allerdings nicht für die rohen Signaldaten berechnet sondern für das bereits erwähnte Seltsamkeitsmaß, welches im Kontext des jeweils verwendeten Machine-Learning-Modells definiert werden muss und angiebt, wie sehr sich ein Datenpunkt von den anderen unterscheidet, bzw. im Kontext eines Machine-Learning-Modells, wie sehr sich ein Datenpunkt von den Trainingsdaten unterscheidet. Die Abhängigkeit der Chang-Point-Detection von diesem Seltsamkeitsmaß hebt das Rahmenwerk der Martingale-Tests von anderen Chang-Point-Detection Algorithmen ab, da eine Veränderung aus der Sicht des trainierten Modells selbst und nicht der rohen Signaldaten untersucht wird. Beispiele für solche Seltsamkeitsmaße wären die Lagrange-Multplikatoren bei Support-Vector-Maschinen oder der Abstand zum nähesten Clusterzentrum bei Clustering-Modellen.

Das Überprüfen der Austauschbarkeit und damit die Ermittlung eines Change-Points erfolgt als Online-Algorithmus anhand von Martingales, nach einem Konzept welches von Vovk et al. [Vovk 2003] erstmals vorgestellt wurde und bei Ho et al. [Ho 2010] näher beschrieben wurde. Nach Eingang eines neuen Datenpunktes wird ein Martingalewert berechnet, welcher die Stärke der Evidenz gegen die Nullhypothese der Austauschbarkeit der Seltsamkeitswerte angibt. Martingales sind wie folgt definiert: Eine Reihe von Zufallsvariablen $\{M_i : 0 \leq i < \infty\}$ wird als Martingale in Bezug auf eine Reihe von Zufallsvariablen $\{Z_i : 0 \leq i < \infty\}$ bezeichnet, wenn für alle $i \geq 0$ die folgenden Bedingungen gelten:

- M_i is eine messbare Funktion auf $Z_0, Z_1, ..., Z_i$
- $\mathbb{E}(|M_i|) < \infty$
- $\mathbb{E}(M_{n+1}|Z_0, ..., Z_n) = M_n$

Für ein gegebenes Seltsamkeitsmaß kann eine als "*Randomized Power Martingale*" bezeichnete und mit $\epsilon \in [0, 1]$ indizierte Familie von Martingales wie folgt berechnet werden:

$$M_n^{(\epsilon)} = \prod_{i=1}^{n} \left(\epsilon \hat{p}_i^{\epsilon - 1} \right)$$

mit den $\hat{p}$-Werten:

$$\hat{p}_i\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_i, y_i)\}, \theta_i) = \frac{\#\{j : s_j > s_i\} + \theta_i \#\{j : s_j = s_i\}}{i}$$

wobei s_j das Seltsamkeitsmaß für $(\mathbf{x}_i, y_i), j = 1, 2, ..., i$ ist und θ_i zufällig aus $[0, 1]$ für jede Instanz i gewählt wird. Der anfängliche Wert der Martingale $M_0^{(\epsilon)} = 1$

Solange die beobachteten Seltsamkeitswerte nicht länger austauschbar sind, sind die obigen $\hat{p}$-Werte unabhängig und gleichverteilt zwischen $[0, 1]$. Sobald die beobachteten Seltsamkeitsmaße allerdings vermehrt höhere Werte aufweisen, gilt dies nicht mehr und die $\hat{p}$-Werte werden kleiner. In der Folge werden die Martingale-Werte $M_n^{(\epsilon)}$ größer und weisen Evidenz gegen die Nullhypothese der Austauschbarkeit der Seltsamkeitswerte auf. Folglich gilt die Nullhypothese “Kein Change-Point inerhalb des beobachteten Datenstroms” nur solange:

$$0 < M_n^{(\epsilon)} < \lambda$$

wobei λ eine positive Zahl ist, welche in Abhängigkeit der tolerierten false-positive-Rate gewählt werden muss.

Der Algorithmus zur online Changepoint-Detection mit Martingale-Frameworks ist somit:

Algorithm 1: Martingale Test

input: $\mathbf{M(0)} \leftarrow 1$, initial Martingale value
input: $\mathbf{i} \leftarrow 1$, datapoint index
input: $\mathbf{T} \leftarrow \{\}$, current sequence of values without changepoint
input: λ, upper limit for Martingale Test
loop
  A new datapoint $\mathbf{x_i}$ is observed.
  if $T = \{\}$ **then**
    Set strangeness of $x_i \leftarrow 0$
  else
    Compute the strangeness of x_i and data points in T
  end
  Compute the $\hat{p}$-values $\hat{p}_i$ using (*)
  Compute $M(i)$ using (**)
  if $M(i) > \lambda$ **then**
    CHANGE DETECTED
    Set $M(i) \leftarrow 1$
    Reinitialize $T \leftarrow \emptyset$
  else
    Add x_i into T
  end
  $i \leftarrow i + 1$
until *end of data stream*;

5 Ausführungsmodell und -format für die erzeugten Modelle

Wegen der oft hohen Anforderungen an Rechenleistung und Speicherausstattung wird der Modellierungsschritt der Datenanalyse (“*modeling*” oder “*data mining*”, siehe CRISP-DM-Modell [Chapman *et al.* 1999, Chapman *et al.* 2000][6]) gewöhnlich auf Servern oder leistungsfähigen Arbeitsplatzrechnern (“*work stations*”) durchgeführt. Hier stehen gewöhnlich spezielle Entwicklungsumgebungen (z.B. Knime[7] oder RapidMiner[8]) oder höhere Programmiersprachen (z.B. Python [Python 2022] oder R [R 2022]) zur Verfügung, die einem Datenanalysten seine Aufgabe sehr erleichtern, indem sie die Vorverarbeitung, die explorative Datenanalyse speziell durch Datenvisualisierung, und die Modellerzeugung mit mächtigen Werkzeugen unterstützen.

Ergebnis eines Datenanalyseprozesses sind gewöhnlich Modelle des maschinellen Lernens (z.B. Entscheidungsbäume, Zufallswälder (“*random forests*”), Bayessche Netze und Klassifikatoren, Stützvektormaschinen (“*support vector machines*”), künstliche neuronale Netze etc.). Zu den Modellen selbst tritt in den meisten Fällen eine ggf. notwendige Vorverarbeitung (z.B. Skalierung, Normalisierung, Merkmalsgenerierung aus den verfügbaren Basismerkmalen etc.) sowie u.U. eine Nachverarbeitung (z.B. Skalierung) oder Entscheidungsableitung.

Diese Modelle sollen i.a. nicht nur längerfristig für Anwendungen bereitgehalten werden (Persistenz) und bequem auf neue Daten anwendbar sein, sondern ggf. auch “*on the edge*”, also auf einem Einsatzrechner nahe am Ort der Entstehung der modellierten Daten, eingesetzt werden (“*deployment*”, siehe CRISP-DM-Modell [Chapman *et al.* 1999, Chapman *et al.* 2000]), z.B. um einem Prozess zu überwachen und anomales Verhalten zu detektieren oder eine vorsorgliche Wartung (“*pre-emptive maintenance*”, häufig auch “*predictive maintenance*”, da Vorhersagen über zukünftige Wartungszeitpunkte gemacht werden) zu ermöglichen.

Für einen solchen Einsatz “*on the edge*” sind jedoch die Programme und Programmsysteme, mit denen Modelle entwickelt werden, oft nur eingeschränkt geeignet, da sie nicht oder nur mit großem Aufwand auf dem späteren Einsatzrechner installiert werden können. Denn sie erfordern meist zu viele Ressourcen, da ein Einsatzrechner “*on the edge*” schon aus Kostengründen vielfach nur mit reduzierter Leistung und begrenztem Speicher ausgestattet ist. Daher stehen für den Einsatz spezielle Modellformate zur Verfügung, mit denen erzeugte Datenanalysemodelle in portabler und standardisierter Form, ggf. mit zusätzlichen Sicherungen gegen Laufzeitfehler, in ressourcensparender Weise in speziellen Laufzeitumgebungen ausgeführt werden können. Diese Formate sollten von einer möglichst breiten Palette an Anwendungsprogrammen unterstützt werden. Auch sollten geeignete Bibliotheken vorliegen, die die Einbindung solcher Formate in Eigenentwicklungen vereinfachen. Im folgenden werden daher bekannte bestehende Formate wie z.B. PMML (Predictive Model Markup Language), PFA (Portable Format for Analytics) und ONNX (Open Neural Network eXchange) für die Speicherung von Modellen des maschinellen Lernens behandelt, ihre Vor- und Nachteile dargestellt und gegeneinander abgewogen und vorhandene Bibliotheken zum Schreiben und Wiedereinlesen dieser Formate und zum Ausführen der beschriebenen Modelle vorgestellt.

5.1 Predictive Model Markup Language (PMML)

Überblick Die *Predictive Model Markup Language (PMML)*[9] ist ein seit 1997 fortlaufend entwickeltes, auf der erweiterbaren Auszeichnungssprache (“*extensible markup language*”, XML) basierendes Standardformat zum Austausch von Vorhersagemodellen (“*predictive models*”) zwischen Datenanalysewerkzeugen und den Zielanwendungen, in denen erzeugte Modelle

[6] Nach Umfragen aus den Jahren 2007 und 2014 ist das CRISP-DM-Modell die am häufigsten eingesetzte Methodologie in Datenanalyse- und Data-Science-Projekten [Piatetsky-Shapiro 2014]:
`https://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html`

[7] `https://www.knime.com/`

[8] `https://rapidminer.com/`

[9] `https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language`

eingesetzt werden. Sie wurde ursprünglich von Robert Lee Grossman erdacht, der 1998 eine erste Version veröffentlichte. Neuere Versionen, einschließlich der im November 2019 veröffentlichten aktuellen Version 4.4, wurden von der Data Mining Group[10] entwickelt. Laut der Data Mining Group ist PMML ausgereift und einer der führenden Standards für statistische und Data-Mining-Modelle zur Klassifikation oder numerischen Vorhersage. Sie wird von über 30 kommerziellen und nicht-kommerziellen Anbietern und Organisationen unterstützt.

PMML bietet Datenanalyseplattformen und -programmen die Möglichkeit, Vorhersagemodelle (*predictive models*) in einem standardisierten Format zu beschreiben, und erleichtert es so, solche Vorhersagemodelle, die durch Algorithmen des Data Mining und des maschinellen Lernens erzeugt wurden, zwischen verschiedenen Systemen und Programmen auszutauschen: Ein Modell, das von einem System erstellt wurde, kann in ein anderes geladen und dort ausgeführt werden. Das Format unterstützt viele Standardmodelle wie z.B. logistische Regression, Entscheidungsbäume und vorwärtsbetriebene neuronale Netze.[1] Weiter können Eingaben vor- und Modellausgaben nachverarbeitet werden.

Format Wie bereits erwähnt, ist PMML ein standardisiertes Format, das auf der sogenannten *erweiterbaren Auszeichnungssprache*[11] (“*extensible markup language*”, XML) basiert. Allgemein ist XML eine Auszeichnungssprache (“*markup language*”) zur Darstellung hierarchisch strukturierter Daten im Format einer Textdatei, die sowohl von Menschen als auch von Maschinen lesbar ist. XML wird auch für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt. Eigentlich ist XML nicht selbst eine Sprache, sondern eine Metasprache, auf deren Basis durch strukturelle und inhaltliche Einschränkungen anwendungsspezifische Sprachen definiert werden. Eine solche Einschränkung ist die *Predictive Model Markup Language* (PMML), durch die speziell die Struktur und die Parameter von Vorhersagemodellen (“*predictive models*”) des Data Mining und des maschinellen Lernens beschrieben werden.

Da PMML auf XML basiert, hat die Spezifikation die Form eines XML-Schemas, dessen allgemeine Struktur (in der aktuellen Version 4.4) so aussieht[12]:

```
<?xml version="1.0"?>
<PMML version="4.4"
  xmlns="http://www.dmg.org/PMML-4_4"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header copyright="Example.com"/>
  <DataDictionary> ... </DataDictionary>
  ... a model ...
</PMML>
```

Ein PMML-Dokument enthält die folgenden Komponenten, von denen zwei bereits in der obigen allgemeinen Struktur angedeutet sind:[13]

- Kopf (*header*)
  Z.B. Urheberrechtsinformationen, eine informelle Beschreibung des Modells, Informationen über das Programm, Werkzeug oder die Plattform, mit der das Modell erzeugt wurde sowie ein Zeitstempel (“*timestamp*”), der das Erstellungsdatum angeben kann.
- Datenbeschreibung (*data dictionary*)
  Die Datenbeschreibung enthält Definitionen für alle möglichen Datenfelder, die von dem Modell benutzt werden, inklusive z.B. Skalenniveaus und Wertebereiche.

[10] `https://dmg.org/`, die DMG wurde 2008 gegründet.
[11] `https://de.wikipedia.org/wiki/Extensible_Markup_Language`
[12] `http://dmg.org/pmml/v4-4/GeneralStructure.html`
[13] `https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language`

- Datentransformationen (*data transformations*)
  Einfache Transformationen wie z.B. Quadrieren oder Logarithmieren.
- Normalisierung (*normalization*)
  Z.B. z-Normalisierung oder Abbildung auf einen Einheitsbereich wie $[0, 1]$ oder $[-1, 1]$, bei nominalen Werten auch eine 1-aus-n-Kodierung.
- Diskretisierung (*discretization*)
  Intervalleinteilung numerischer Werte, dadurch Abbildung eines metrischen auf ein nominales bzw. ordinales Merkmal.
- Werteabbildung (*value mapping*)
  Abbildung von nominalen auf nominale Werte, z.B. zur Vergröberung der Einteilung.
- Funktionen (*functions*)
  Merkmalsberechnung aus einem oder mehreren Basismerkmalen über vorgegebene (direkt aufrufbare) oder benutzerdefinierte Funktionen.
- Aggregation (*aggregation*)
  Bildung von Wertegruppen und ggf. Berechnung statistischer Kenngrößen je Gruppe.
- Model (*model*)
  Definition des eigentlichen Data-Mining-Modells. PMML sieht Attribute für verschiedene Arten von Modellen vor, wie z.B. Stützvektormaschinen (“*support vector machines*”), Assoziationsregeln, naive Bayes-Klassifikatoren, Clusteranalysemodelle, Textmodelle, Entscheidungsbäume , verschiedene Regressionsmodelle und künstliche neuronale Netze.
- Mining-Schema (*mining schema*)
  Liste aller Felder, die in dem Modell verwendet werden; sollte eine Teilmenge der Felder sein, die in der Datenbeschreibung angegeben wurden oder aus den Daten berechnet wurden. Die Felder können mit Zusatzinformationen z.B. zur Behandlung von Ausreißern oder fehlenden Werten versehen werden.
- Zielgrößen (*targets*)
  Nachverabeitung der durch ein Modell vorhergesagten Werte z.B. durch Skalierung.
- Ausgabe (*output*)
  Festlegung der gewünschten Ausgabefelder, die von dem Modell geliefert werden sollten.

Implementierungen Die Predictive Model Markup Language (PMML) wird von einer Vielzahl von Datenanalyseplattformen unterstützt, z.B. prudsys Expert Mining Suite, IBM Intelligent Miner, TIBCO Spotfire Miner, KXEN, SAS Enterprise Miner, SPSS Clementine, KNIME, um nur einige der bekanntesten zu nennen [Thilo 2002]. Für eine Einbindung in eigene Programme stehen u.a. zur Verfügung: die Python-Bibliothek Nyoka der Software AG als Open Source, die frei auf GitHub verfügbare Java-Bibliothek JPMML[14] [Fillbrunn 2014], oder die Xelopes-Bibliothek, von der es eine kommerzielle und eine Open-Source-Version gibt.

Die Tatsache, dass PMML ein XML-Dialekt ist, ermöglicht allerdings grundsätzlich das Einlesen mit allgemeinen Werkzeugen für die Verarbeitung von XML-Dateien. Dies liefert aber zunächst nur eine Möglichkeit, PMML-Dokumente einzulesen. Sie müssen aber auch interpretiert werden, d.h., das beschriebene Modell muss ausführbar gemacht werden. Dies liefern die oben angegebenen Bibliotheken, von denen JPMML als die empfehlenswerteste erscheint. Allerdings kann auch die verwendete Programmiersprache die Bibliothek bestimmen. Wenn in Python implementiert wird, ist die Nyoka-Bibliothek wegen ihrer nativen Python-Schnittstelle vorzuziehen (wenn auch Java prinzipiell in Python eingebunden werden kann, was jedoch u.U. Komplikationen mit sich bringt).

[14] `https://github.com/jpmml`

5.2 Portable Format for Analytics (PFA)

Überblick Das *Portable Format for Analytics (PFA)*[15] ist ein neuerer Standard für Modelle des maschinellen Lernens, der jedoch bisher noch nicht einmal Version 1.0 erreicht hat und dessen letzte Aktualisierung (auf die Version 0.8.1) aus dem Jahre 2015 stammt. PFA kombiniert Portabilität über Systeme und Plattformen hinweg (speziell von Datenanalyseplattformen zu operativen Anwendungen, die diese Modelle einsetzen) mit der Flexibilität, diese Modelle sowie ihre Vor- und Nachverarbeitung anpassen zu können. Denn sowohl Modelle als auch Vor- und Nachverarbeitungsschritte sind als Funktionen ausgelegt, die beliebig kombiniert und verkettet werden können. So ist es möglich, einfache aber auch sehr komplexe Verarbeitungsabläufe zu spezifizieren. Ein PFA-Dokument kann eine einfache Vorverarbeitung roher Daten beschreiben, die lediglich abgeleitete Merkmale erzeugt, oder einen hochentwickelten Satz simultan (nebenläufig) auszuführender Data-Mining-Modelle darstellen.

Alle Verarbeitungen, die ein PFA-Dokument spezifiziert, werden in JSON (JavaScript Object Notation) oder einer äquivalenten, vereinfachten und für Menschen noch besser lesbaren Form names YAML angegeben (was heute als “YAML Ain’t Markup Language” gelesen wird, ursprünglich aber “Yet Another Markup Language” bedeutete). In der JSON-Form ist ein PFA-Dokument besonders praktisch, weil es, wie jedes JSON-Dokument, ausführbares JavaScript ist (bzw. sein soll) und daher unmittelbar von einem normkonformen JavaScript-Interpretierer ausgeführt werden kann. Anders als für PMML ist daher keine eigene Implementierung einer Ausführungsumgebung nötig, wenn auch Implementierungen der Funktionen vorhanden sein müssen sind, die Standardmodelle darstellen.

Format Wie bereits erwähnt, ist PFA in JSON (JavaScript Object Notation) realisiert. Da JSON als JavaScript eine vollwertige Programmiersprache ist, sind der Verarbeitung und Funktionsdefinition quasi keine Grenzen gesetzt. Weil JavaScript die üblichen Kontrollstrukturen allgemeiner Programmiersprachen unterstützt, mit denen der Ablauf eines Programms gesteuert werden kann wie z.B. bedingte Anweisungen, begrenzte und unbegrenzte Schleifen und Funktionsdefinitionen, sind diese auch in PFA verfügbar, zuzüglich spezieller Funktionen, die Standardmodelle der Statistik, des Data Mining und maschinellen Lernens implementieren.

Durch diese Flexibilität kann der Autor eines PFA-Dokumentes im Prinzip beliebige neue Datenverarbeitungen und Modellklassen erstellen, die über vorhandene Funktionalität hinausgehen, wobei viele Bausteine solcher Modelle und Verarbeitungsschritte bereits zu Verfügung stehen. Neue Modelle können daher unmittelbar hinzugefügt und verwendet werden.

Die Funktionsbibliothek von PFA ist fein strukturiert und daher sehr detailliert anpaßbar, so daß mehrstufige Verarbeitungen definiert werden können, indem mehrere Funktionen miteinander verkettet werden. Weiter akzeptieren viele Bibliotheksfunktionen Rückruffunktionen (“*callbacks*”), mit denen das interne Verhalten dieser Funktionen (in bestimmten Grenzen) beeinflußt und auf spezifische Problemstellungen angepaßt werden kann. Vorhersage- und Auswertemodelle (“*scoring engines*”) können außerdem Daten gemeinsam bearbeiten oder externe Variablen aktualisieren, wie z.B. Einträge in einer Datenbank.

PFA verfügt weiter über ein echtes Typsystem, das sowohl für Parameter als auch für Daten gilt. In dieser Hinsicht weicht PFA von seiner Grundlage JavaScript ab, das eine dynamisch typisierte Sprache ist. Bei einer dynamisch typisierten Sprache finden Typprüfungen, also ob Objekte und Parameter den richtigen Typ haben, um mit ihnen die gewünschten Operationen ausführen zu können, im wesentlichen zur Laufzeit statt. Für die Ausführungssicherheit hat eine dynamische Typisierung jedoch viele Nachteile, weil ein auf fehlerhaften Typen beruhender Programmierfehler u.U. erst zur Ausführungszeit erkannt werden kann, was zu einem Zusammenbruch des Verarbeitungsprozesses führen kann.

[15] `https://en.wikipedia.org/wiki/Portable_Format_for_Analytics`

PFA hat dagegen ein Typsystem, das statisch, also bereits zur Übersetzungszeit des Programms geprüft werden kann. Zwar ist JavaScript und damit auch PFA im Prinzip eine interpretierte Sprache, jedoch wird eigentlich eine Art Zwischenkode für eine virtuelle Maschine interpretiert. Der JavaScript- oder PFA-Quelltext kann (und wird oft) vor der Ausführung in diesen Zwischenkode übersetzt. Bei dieser Übersetzung kann in PFA eine statische Typprüfung stattfinden, mit der sich viele Programmierfehler finden lassen, bevor es zur eigentlichen Ausführung und damit ggf. zu unerwünschten Wirkungen solcher Programmierfehler kommt. Außerdem vermeidet ein strenges Typensystem unsichere Typumwandlungen.

Ein PFA-Dokument ist eine Serialisierung (d.h., eine Kodierung in Form einer Zeichenkette) einer Auswertemethode ("*scoring engine*"), z.B. um Vorhersagen von Klassen oder numerischen Werten zu berechnen. Eine solche Auswertemethode ist ein ausführbares Programm, das eine wohldefinierte Eingabe und Ausgabe hat und eine rein mathematische Arbeit erledigt. Folglich hängen die Berechnungen nicht von der Umgebung ab, in der die Auswertemethode ausgeführt wird: bei gleichen Eingaben werden stets gleiche Ausgaben erzeugt.

Die Eingaben kommen aber natürlich aus irgendeiner Quelle und Ausgaben müssen an irgendeine Senke ausgegeben werden. Folglich muß ein Teil der Auswertemethode in Kontakt mit seiner Umgebung treten. Dieser Teil, der Verarbeitungsrahmen ("*pipeline framework*"), interpretiert Dateien oder bedient Netzwerkprotokolle, um Daten in den Auswertealgorithmus zu leiten. PFA wird daher stets im Zusammenspiel mit einem sogenannten Wirtssystem ("*PFA host*") benutzt, denn PFA-Quelltext wird auf diesem Wirtssystem in einer virtuellen Maschine ausgeführt (Kapselung für die Sicherheit der Ausführung).

Implementierungen Implementierungen, mit denen PFA-Modelle ausgeführt, erstellt und verändert werden können, gibt es in/für verschiedene Programmiersprachen, wie die folgende Liste zeigt, die auf sehr beliebte Datenanalysesprachen ausgerichtet ist:[16]

- Hadrian (Java/Scala/JVM)
  Hadrian ist eine vollständige Implementierung von PFA in der Programmiersprache Scala[17], auf die durch jede Programmiersprache zugegriffen werden kann, die die Java Virtual Machine (JVM)[18] verwendet, hauptsächlich natürlich Java selbst. Der Fokus liegt bei Hadrian auf dem Modelleinsatz, weswegen es flexibel gestaltet ist, so dass es auch in eingeschränkten Umgebungen laufen kann. Es ist auch recht schnell in der Ausführung.
- Titus (Python)
  Titus ist eine vollständige Implementierung von PFA in der Programmiersprache Python[19] [Python 2022]. Der Fokus von Titus liegt auf der Modellentwicklung. Folglich enthält Titus zusätzlich zu einem PFA-Ausführungsmodul Modellerstellungs- und -anpassungsfunktionen. Die Ursprungsversion von Titus beruhte of Python2.x, was inzwischen sein Lebensende erreicht hat. Aber mit Titus 2 liegt auch eine Version vor, die auf Python 3.x basiert.
- Aurelius (R)
  Aurelius ist ein Werkzeug und eine Bibliothek, um PFA-Dokumente in der Programmiersprache R[20] [R 2022] zu erzeugen. Der Fokus von Aurelius liegt auf der Übersetzung von R-Modellen in äquivalente PFA-Modelle. Um erzeugte PFA-Dokumente zu prüfen und die beschriebenen Auswertungs- und Vorhersagemodelle auszuführen, werden sie von Aurelius auf dem Weg über rPython an Hadrian (siehe oben) geschickt.

[16] https://en.wikipedia.org/wiki/Portable_Format_for_Analytics
[17] https://de.wikipedia.org/wiki/Scala_(Programmiersprache)
[18] https://de.wikipedia.org/wiki/Java_Virtual_Machine
[19] https://de.wikipedia.org/wiki/Python_(Programmiersprache)
[20] https://de.wikipedia.org/wiki/R_(Programmiersprache)

- Antinous (Modellentwicklung in Jython)[21]
  Antinous ist ein Zusatzprogramm für Hadrian, mit dem Modelle erstellt werden können und das erlaubt, Jython-Quelltext an allen Stellen auszuführen, an denen ein PFA-Modell zum Einsatz kommt. Antinous verfügt außerdem über eine Bibliothek von modellerzeugenden Algorithmen.

5.3 Open Neural Network eXchange (ONNX)

Überblick Das Open Neural Network eXchange (ONNX) Format ist das jüngste der drei Formate und erst seit 2017 verfügbar. Seine Entwicklung wurde vor allem durch die in den letzten Jahren erzielten rasanten Fortschritte im Bereich des Trainings großer und "tiefer" neuronaler Netze ("*deep learning*") angetrieben. Dieses "dritte goldene Zeitalter der künstlichen neuronalen Netze" wurde i.w. durch bessere Hardware (speziell Graphikkarten, die das Training künstlicher neuronaler Netze erheblich beschleunigen) ermöglicht, durch die neue und wesentlich komplexere Netzmodelle trainierbar wurden.

Während anfänglich der Fokus von ONNX vor allem auf einer kompakten Repräsentation und einfachen Konvertierung von Deep-Learning-Modellen zwischen verschiedenen Umgebungen lag, wurde der Standard in weiterer Folge auch dahingehend erweitert, dass klassische Machine-Learning-Modelle wie Zufallswälder ("*random forests*") oder Stützvektormaschinen ("*support vector machines*") unterstützt werden. Außerdem unterstützt ONNX die am häufigsten verwendeten Machine-Learning- bzw. Deep-Learning-Umgebungen wie etwa TensorFlow, PyTorch, SciKit-Learn, XGBoost und viele weitere.

ONNX definiert eine gemeinsame Menge von Operatoren, speziell Operatoren auf Tensoren, d.h. auf mehrdimensionalen numerischen Daten, die die Bausteine vom Modellen des maschinellen Lernens und speziell künstlicher neuronaler Netze bilden. Außerdem definiert es ein gemeinsames Dateiformat, mit dem Anwender in die Lage versetzt werden sollen, in ONNX beschriebene Modelle mit einer Vielzahl von Werkzeugen, Entwicklungs- und Laufzeitumgebungen und Übersetzern auszutauschen.

Format ONNX-Modelle werden als gerichtete Graphen mit zugehörigen Metadaten dargestellt. Diese Information wird mittels Protokollpuffern ("*protocol buffers*") serialisiert und als ONNX-Datei abgespeichert. Die Metadaten beinhalten Informationen wie beispielsweise die Versionsnummern des Models bzw. der ONNX-Version, welche zu dessen Erstellung verwendet wurde oder auch eine optionale Dokumentation des Modells. Ebenfalls sind Opset-Import-Informationen vorhanden welche sämtliche Operatoren auflistet, welche zur Ausführung des Modells zur Verfügung stehen müssen, sowie deren jeweilige Versionen. Diese Operatorenmengen werden mittels Domänennamen und Versionsnummer eindeutig identifiziert. ONNX selbst stellt zwei Operatorenmengen zur Verfügung: *ai.onnx* beinhaltet Operatoren, welche vor allem für Deep-Learning-Modelle benötigt werden, wie Matrixmultiplikation, rekurrente Schichten, Dropout, usw., während *ai.onnx.ml* Funktionalitäten für klassische Machine-Learning-Modelle wie TreeEnsembleRegressor oder SVM-Regressor enthält. Darüber hinaus können auch eigene Operatorenmengen definiert werden. Eine ausführliche Auflistung weiterer Metadaten findet sich unter `https://github.com/onnx/onnx/blob/main/docs/IR.md`

Der Ausführungsgraph selbst ist aus einzelnen Knoten aufgebaut. Diese enthalten wiederum jeweils eine Liste von Ein- und Ausgaben, den Namen des Operators, welcher durch den Knotens ausgeführt wird, sowie Attribute. Attribute stellen fixierte Parameter dar, welche einmalig instanziiert und anschließend nicht mehr verändert werden können. Dies können beispielsweise die gelernten Modellgewichte eine künstlichen neuronalen Netzes sein, aber auch Parameter, welche die Art wie der Operator ausgeführt wird näher spezifizieren, wie beispielsweise die Dropoutrate einer entsprechenden Dropout-Schicht.

[21] `https://de.wikipedia.org/wiki/Jython`

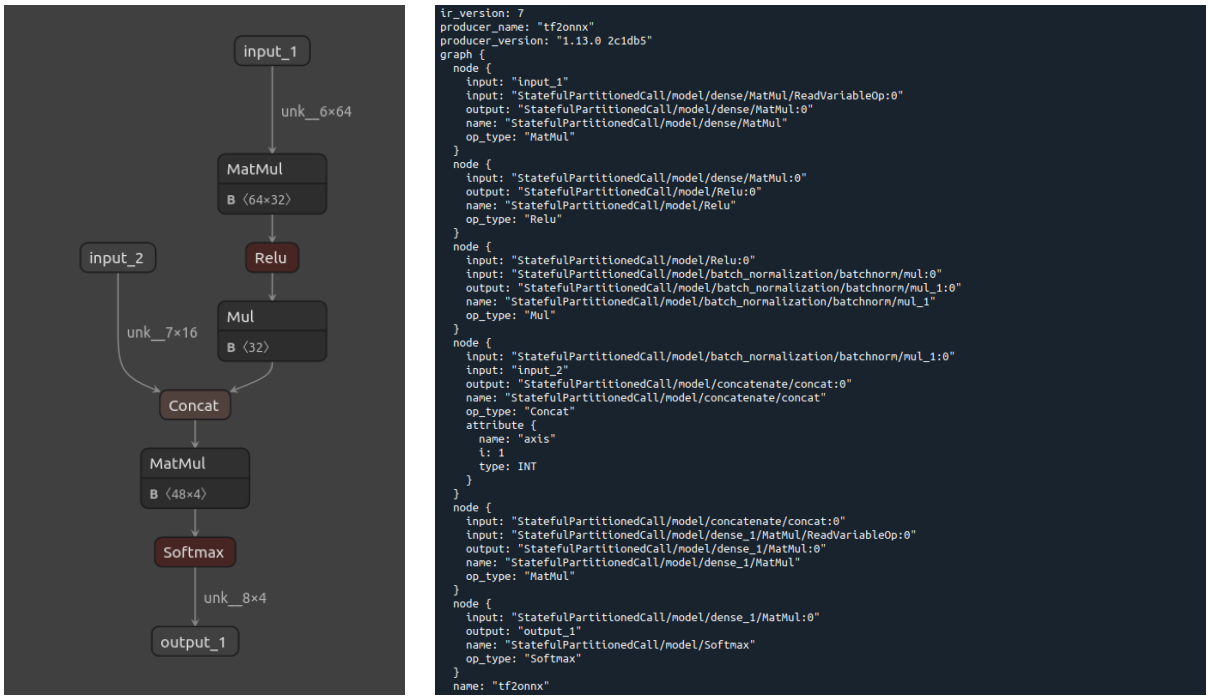

Abbildung 14: Darstellung eines einfachen ONNX-Models in graphischer Form (Netron) bzw. dessen Repräsentation duch ONNX

Abbildung 14 zeigt die ONNX-interne Repräsentation eines einfachen neuronalen Netzes, welches aus zwei Eingabeschichten, zwei linearen Schichten welche von einer Rectified Linear Unit (ReLU) bzw. Softmax Aktivierungsfunktion gefolgt werden, sowie einer Batch-Normalisierung. Die ebenfalls dargestellte graphische Veranschaulichung des Modells lässt sich direkt aus der ONNX-Datei mithilfe von Netron (`https://netron.app/`) erzeugen.

Wie bereits erwähnt, ist mithilfe von ONNX auch die Repräsentation von klassischen Machine-Learning-Modellen möglich, optimiert wurde ONNX allerdings für numerische Berechnungen mit Tensoren, welche die grundlegende Datenstruktur für neuronale Netze darstellen. Ein solcher Tensor besteht weißt folgende drei Eigenschaften auf:

- **Datentyp:** ONNX ist streng typisiert. Typwandlungen sind nur explizit innerhalb des Ausführungsgraphen möglich.
- **Shape:** Für die Ausführung des Modells selbst ist es nicht zwangsläufig nötig, die Gestalt der Eingabedaten zuvor definiert zu haben, allerdings beschleunigt dies die Ausführung.
- **Werte:** Ein Array, welches die eigentlichen numerischen Werte des Tensor enthält.

Implementierungen ONNX-Modelle wurden 2019 in Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch und OpenCV unterstützt, und es gibt Schnittstellen für viele andere gängige Frameworks und Bibliotheken. Dazu gehören unter anderem TensorFlow, Keras, SciKit-Learn, XGBoost und LightGBM um nur die bekanntesten zu nennen.

Mittels einer eigenen Laufzeitumgebung ist es außerdem möglich, ONNX-Modelle auf verschiedensten Produktionsumgebungen auszuführen. Die Laufzeitumgebung steht derzeit für die Programmiersprachen Python, Java, C, C++, C#, Objective-C und JavaScript zur Verfügung (außerdem für Ruby und Julia als Community Edition).

Darüberhinaus existiert seit 2021 ebenfalls eine eigene Laufzeitumgebung für das Training von PyTorch-Modellen. ONNX-Modelle können auch mittels externer Laufzeitumgebungen wie

Windows AI, TVM oder TensorRT (Nvidia), ausgeführt werden. Dies ermöglicht eine große Flexibilität in Bezug auf die zugrunde liegende Hardwareplattform, beispielsweise bietet microTVM die Ausführung von Deep-Learning Modellen auf verschiedenen Microcontroller-Architekturen an.

Eine Visualisierung der erstellten Modelle (wie in Abbildung 14 dargestellt kann des weiteren mit Netron, VisualDL und Zetane erstellt werden.

Zuletzt sei noch erwähnt, das ONNX bereits eine Umfangreiche Auswahl an bereits trainierten Deep-Learning Modellen mit den populärsten Architekturen aus verschiedensten Bereiche wie Computer Vision, Natural Language Processing uvm. bietet.

5.4 Vergleich und Bewertung

Die Predictive Model Markup Language (PMML) ist zwar ein sehr ausgereiftes und breit unterstützte Format zum programmunabhängigen Speichern von Modellen des maschinellen Lernens. Es unterstützt alle Standardmodellklassen und alle standardmäßig vorzunehmenden Vor- und Nachverarbeitungsschritte wie z.B. Normalisierungen oder andere Skalierungen. Es ist daher für alle Standardaufgaben geeignet und damit für die Praxis meist ausreichend. Es ist jedoch in seiner Flexibilität beschränkt. Damit neue Modellklassen oder neue Varianten bestehender Modellklassen unterstützt werden können, ist meist eine Erweiterung des Formats notwendig, so dass Modelle der neuen Klasse oder der Variante der alten Klasse beschrieben werden können. Auch wenn bestimmte, sehr begrenzte Erweiterungen in einer Version des Formats noch möglich sind, bedürfen stärkere Modifikationen, dass eine neue Formatversion definiert und veröffentlicht wird. Dieser Prozess kann, bedingt durch die nötige Abstimmung vieler Partner, recht lange dauern (wenn die Partner auch in der Data Mining Group[22] zusammengschlossen sind, was einen schnelleren Austausch ermöglicht).

Das Portable Format for Analytics (PFA) hat gegenüber PMML einige Vorteile. So ist es durch die Verwendung der JavaScript Object Notation (JSON) kompakter als das auf der erweiterbaren Auszeichnungssprache (*extensible markup language, XML*) basierende PMML. Zwar ist Speicherplatz heute sehr preiswert zu bekommen und das Lesen von und Schreiben auf nicht-flüchtige Speichermedien hat an Geschwindigkeit erheblich gewonnen. Dennoch sollte man, insbesondere auf Geräten “on the edge”, nicht unnötig Speicher vergeuden und Zugriffzeiten unnötig verlängern, wenn dies vermeidbar ist. Ein weiterer Vorteil von PFA ist die größere Flexibilität im Vergleich zu PMML, da es Möglichkeiten z.B. der Vor- und Nachverarbeitung der Daten bietet, die über die wesentlich rigideren Möglichkeiten weit hinausgehen, die PMML anbietet. Dies geschieht, ohne Sicherheit der Ausführung zu opfern, da PFA nur die Anwendung von Funktionen auf Daten erlaubt, etwa im Sinne eines Filters für einen Datenstrom, ohne dass diese Daten selbst geändert werden können. Allerdings scheint die Entwicklung von PFA eingeschlafen zu sein, da es seit 2015 bei der Version 0.8.1 verharrt, also noch nicht einmal eine offizielle Startversion 1.0 erreicht hat.

ONNX ist das neueste der betrachteten Formate und erst seit 2017 verfügbar. Es war ursprünglich auf künstliche neuronale Netze beschränkt und sollte eine kompakte Darstellung und einfache Übertragung zwischen verschiedenen Umgebungen unterstützen. Neuere Versionen unterstützen aber auch Standardmodellklassen, z.B. über SciKit-Learn. Es hat gegenüber den anderen Formaten den Vorteil, dass seine Entwicklung derzeit von mehreren einflußreichen Firmen aktiv vorangetrieben wird (z.B. Microsoft, Amazon, Facebook etc.). Dynamik und Triebkraft sprechen daher derzeit ziemlich eindeutig für ONNX. Da ONNX wie PFA ebenfalls über eine strenge Typisierung verfügt, ist sollte es möglich sein, Laufzeitfehler weitgehend zu vermeiden, was für eine verlässliche Ausführung auf einem Einsatzrechner in der Praxis sehr wesentlich ist. Zum gegenwärtigen Zeitpunkt erscheint es daher am empfehlenswertesten, auf das ONNX-Format zu setzen.

[22] `https://www.dmg.org`

Format	R		Python	
	Export	Import	Export	Import
PMML	ja (als pmml/xml)	nein	unklar (über sklearn?)	ja (nur Anwendung eines Modells, keine Modifikation)
PFA	nein	nein	ja	(ja) (nur PFA-eigene Befehle)
ONNX	nein	nein	ja (YAML / JSON)	(ja) (nur Anwendung eines Modells, keine Modifikation)
H5	ja	(ja) (nur, wenn aus R geschrieben)	ja (Model & Gewichte)	ja (Model & Gewichte) (Umwandlung in Keras-Modell nötig)
TensorFlow	ja (Model als `.pb`)	ja (als Keras-Modell)	ja (Model als `.pb`)	ja (als Keras-Modell)

Tabelle 4: Eine Übersicht ob und wenn ja, wie sich verschiedene Formate für Modelle des maschinellen Lernens aus R und Python exportieren und in diese Sprachen wieder importieren lassen. H5 und TensorFlow sind dabei Formate, die i.w. für die Speicherung der Parameter künstlicher neuronaler Netze geeignet sind.

5.5 Interoperabilität und praktische Erfahrungen

In einem Teilprojekt wurde durch praktische Experimente untersucht, mit welchen Formaten sich ein Modellaustausch zwischen den Programmiersprachen Python und R bewerkstelligen läßt, wobei vorrangig die oben betrachteten Formate PMML, PFA, und ONNX, aber auch spezifischere Formate wie das Hierarchical Data Format (HDF) in der speziellen Version H5 oder das Format der Deep-Learning-Umgebung TensorFlow, die jedoch i.w. zum Abspeichern der Parameter von künstlichen neuronalen Netzen dienen.

Gegeben die Tatsache, dass Python und R sehr beliebte Sprachen bzw. Umgebungen für die Datenanalyse sind und einige der besprochenen Formate schon älter sind, ist das Ergebnis relativ ernüchternd. Die gemachten Erfahrungen sind in Tabelle 4 zusammengefaßt, wobei Python leicht besser abschneidet als R. Besonders das Importieren von Modellen bereitet Schwierigkeiten, jedenfalls dann, wenn man diese Modelle in der neuen Umgebung anpassen / modifizieren möchte. Ein unverändertes Ausführen ist zumindest in Python meist möglich (wenn auch bei PFA auf PFA-eigene Befehle und Funktionen beschränkt). Eine Übertragung eines künstlichen neuronalen Netzes als Keras-Modell ist dagegen meist gut möglich. Dies beruht aber auf der zugrundeliegenden TensorFlow-Umgebung und den Schnittstellen, die Python und R zu dieser Umgebung bereitstellen.

6 Ausblick: InterOpera AI AAS Submodelle

Seit 2022 läuft das Projekt “InterOpera: Digitale Interoperabilität in kollaborativen Wertschöpfungsnetzwerken der Industrie 4.0” (`https://interopera.de/`), das auf eine standardisierte Umsetzung von Teilmodellen der Asset Administration Shell (AAS) in der Praxis hinarbeitet. InterOpera besitzt ein Projektkonsortium, in dem einflußreiche Organisationen im Bereich Industrie 4.0 vertreten sind (siehe Abbildung 15). Im Rahmen dieses Projektes sollen insgesamt 50 konkrete AAS-Teilmodelle entwickelt werden, sowie, abgeleitet aus Erfahrungen mit dem Entwurf konkreter Teilmodelle, ein Leitfaden zum strukturierten Entwurf solcher Teilmodelle.

Abbildung 15: InterOpera Projektkonsortium.
(Bildquelle: Infoveranstaltung KI-Teilmodelle am 24.02.2023[23] — © Mike Reichardt, DFKI)

Zur Verwendung der Asset Administration Shell in Rahmen der Verwaltung von Modellen der künstlichen Intelligenz und des maschinellen Lernens findet man bereits Veröffentlichungen, wie z.B. [Rauh *et al.* 2022]. Im Rahmen des InterOpera-Projektes sollen auch drei Teilmodelle für die Darstellung von Datensätzen (“*data set*”), Modellen (“*model nameplate*”, wobei allerdings der Fokus auf künstlichen neuronalen Netzen zu liegen scheint) und besonderer Informationen für die Modellanwendung (“*deployment*”) entwickelt werden. Die Ablage welcher Informationen in diesen drei Teilmodellen geplant ist, ist in Abbildung 16 gezeigt. Seit Juni 2023 ist das Projekt abgeschlossen und die drei Teilmodelle, welche die Erfassung und Verwaltung relevanter Information entlang des KI-Lebenszyklus abdecken sollen, fertiggestellt. Nachfolgend sollen die drei Teilmodelle kurz vorgestellt werden, außerdem wird auf Anknüpfungspunkte dieser Teilmodelle mit den bisher diskutierten Lösungsansätzen verwiesen.

[23] Infoveranstaltung zum Industrie 4.0-Projekt InterOpera und Asset Administration Shell-Teilmodellprojekten zur Künstlichen Intelligenz amd Freitag, 24. Februar 2023 von 13:00 bis 14:30 Uhr, `https://eveeno.com/143196811`

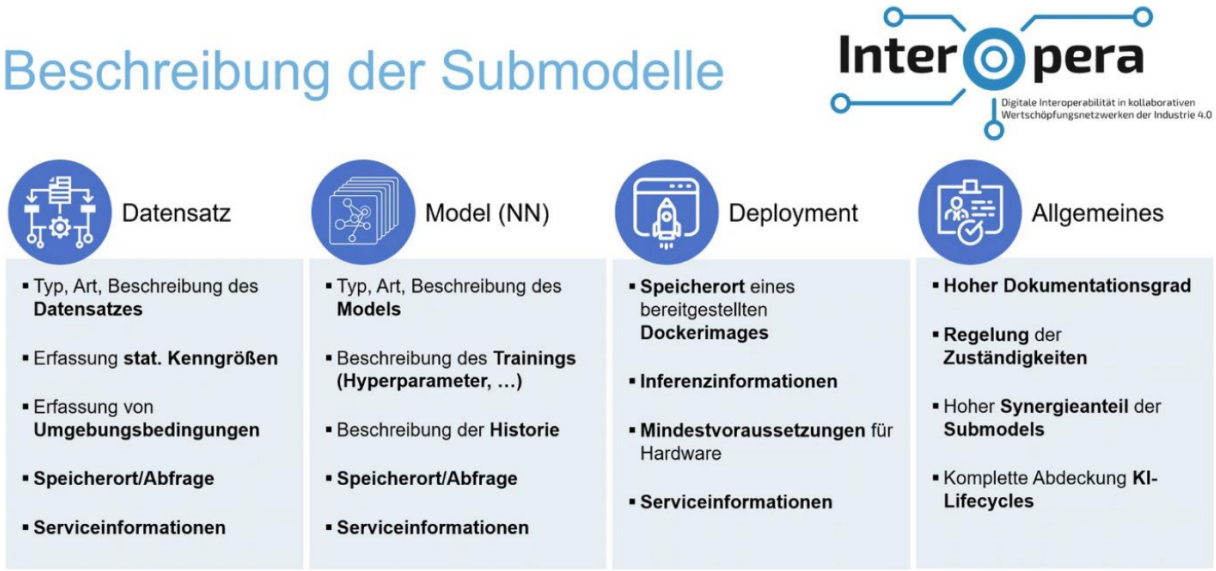

Abbildung 16: InterOpera Teilmodelle für Künstliche Intelligence.
(Bildquelle: Infoveranstaltung KI-Teilmodelle am 24.02.2023— © Mike Reichardt, DFKI)

6.1 AI Model Nameplate

Das Teilmodel AI Model Nameplate kann als eine Art Typenschild für KI-Modelle in der Produktionsindustrie verstanden werden, also eine standartisierte Dokumentation der wichtigsten Merkmale eines Maschine-Learning Modells. Da sich die relevanten Charakteristika von Maschine-Learning Modellen doch ziemlich nach der konkreten Art des Modells unterscheiden, wurde einerseits versucht, gemeinsame, für die Modellverwaltung relevante Merkmale herauszuarbeiten, wie beispielsweise den Speicherort oder die Trainingsergebnisse, andererseits ist auch die Angabe von Modellspezifischen Informationen möglich. Abbildung 17 zeigt die Elemente des resultierenden Submodells, nachfolgend wird kurz auf eizelne Submodellelemente eingegangen:

- **Input:** Enthält wesentliche Informationenen über die Struktur des Modellinputs, dessen Dimensionen bzw. Beschreibungen derer, sowie Erläuterungen zu etwaigen Vorverarbeitungsschritten.
- **Output:** Ähnlich wie beim Input wird hier die Struktur der Modellausgabe angegeben, also die Dimensionierung sowie Beschreibungen der einzelnen Dimensionen.
- **Details:** Hier werden vor allem Informationen angegeben, welche für das Ausführen des Modells in einer anderen Umgebung benötigt werden. Dazu gehört das Dateiformat, in welchem das Modell gespeichert wurde, das Rahmenwerk, welches benutzt wurde um das Modell zu erstellen (z.B. TensorFlow) bzw. die Programmiersprache in der dies erfolgte. Außerdem werden mögliche Voraussetzungen an die Entwicklungsumgebung wie etwa Module von Drittanbietern angegeben.
- **Dataset:** Hier findet sich eine Referenz auf ein Datensatz-Submodell, auf welches nachfolgend noch genauer eingegangen wird, sowie ein Zeitstempel, zu welchem jenes abgerufen wurde.
- **AIMethodSpecificInformation:** Wie bereits angesprochen finden sich hier modellspezifische Informationen, also solche, welche nur im Kontext des jeweils verwendeten Modells

Abbildung 17: Teilmodell AI Nameplate.

Relevanz haben. Dies könnte beispielsweise die verwendete Lernrate oder Fehlerfunktion sein.

6.2 AI Dataset

Wie der Name bereits nahelegt, dokumentiert das Teilmodell AI Dataset die relevanten Informationen eines KI-Datensatzes mit dem Ziel, die Auffindbarkeit, Nachvollziehbarkeit sowie Bewertung der Qualität für KI-Entwickler zu gewährleisten. Durch die im Teilmodell enthaltenen Informationen sollte es daher ohne zusätzlichen Kontext möglich sein, einen gegebenen Datensatz als Trainingsdatensatz für ein Maschine-Learning Modell zu verwenden. Ähnlich wie beim zuvor erläuterten Model Nameplate werden zunächst Datentyp-unspezifische Informationen angegeben, in einer eigenen Submodelcollection können allerdings auch Merkmale für spezielle Datentypen wie Bild-, Audio- oder Textdaten festgehaltern werden. Nachfolgend werden einige Elemente des Teilmodells näher erläutert:

- **Labeled:** Hier finden sich Verweise bzw. Erklärungen zu Annotationsdateien die als Zielvariablen für das Training von Maschine-Learning Modellen dienen können.
- **SizeInformation:** Diese Submodelcollection beinhaltet Details über die Aufteilung des Datensatzes in Trainings-, Validierungs- und Testdatensatz.
- **MetaData:** Dies beinhaltet neben dem Datentyp wie angesprochen auch Datentypspezifische Informationen in eigenen Submodelcollections wie “DetailsforAudio” oder “DetailsforImages”.

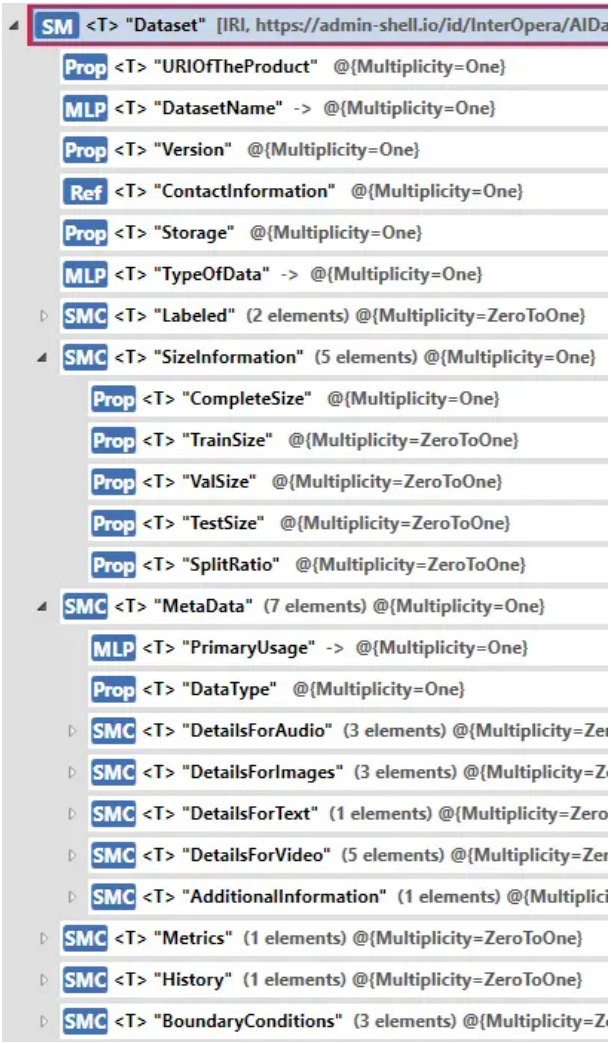

Abbildung 18: Teilmodell AI Dataset.

- **Metrics:** Hier finden sich statistische Kenngrößen, die den Datensatz näher beschreiben, wie Mittelwerte oder Varianzen.
- **History:** Ähnlich wie der Punkt “Dataset” im zuvor beschriebenen Model Nameplate enhält diese Submodelcollectiton Verweise auf Maschiene-Learning Submodelle welche diesen Datensatz zum Trainieren verwendet haben sowie einen Zeitstempel, wann dieser abgerufen wurde.
- **BoundaryConditions:** Diese Sammlung listet Umgebungsbedingungen zur Verwendung des Datensatzes sowie Verweise auf DataCollector-Teilmodelle, wie beispielsweise die Sensoren die zur Erzeugung des Datensatzes verwendet wurden.

6.3 AI Deployment

Dieses Teilmodell unterstützt bei der Integration und Inbetriebnahme von KI-Modellen in die Produktionsumgebung. Neben Verweisen auf das entsprechende Model Nameplate finden sich hier auch Hard- und Softwarevoraussetzungen zur Ausführung des Modells in einer neuen Umgebung auch mitlaufende Metriken, die Informationen über die Auslastung der Ausführungsumgebung oder Hinweise auf einen Drift der Inputsignale geben, welcher wie in Kapitel 4.3.5

erläutert möglicherweise eine Anpassung des Modells erfordert. Nachfolgend eine genauere Erläuterung zu einigen Elementen des Teilmodells:

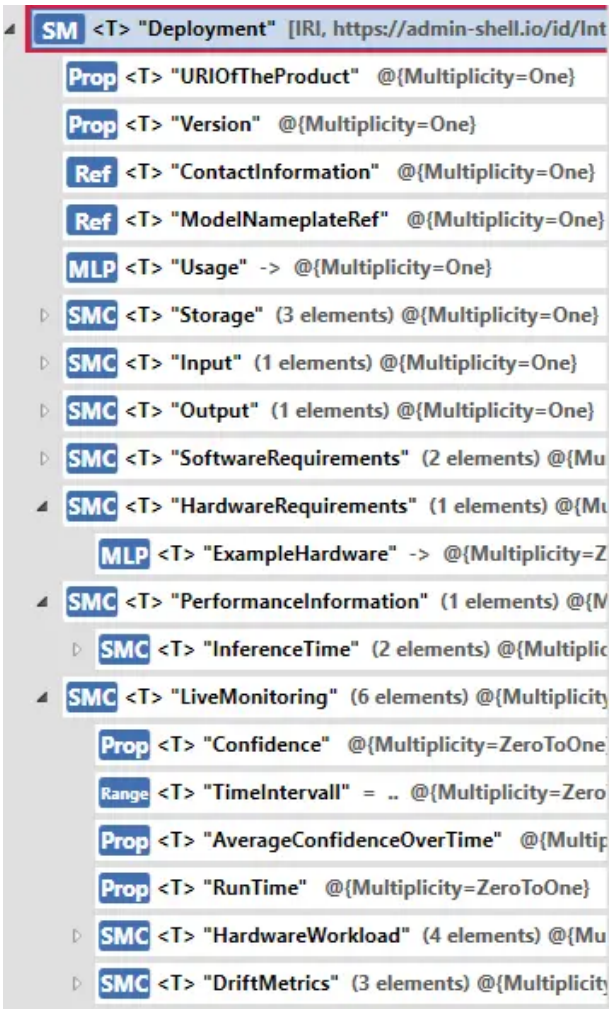

Abbildung 19: Teilmodell AI Deployment.

- **Storage:** Ein Verweis auf den Pfad zum gespeicherten Modell bzw. dessen Format.
- **Input** und **Output:** Erläuterungen zu den Dimensionen der Eingabe- und Ausgabedaten.
- **SoftwareRequirements:** Software, Module oder Bibliotheken, welche zur Ausführung des Modells benötigt werden.
- **HardwareRequirements:** Die Mindesthardware, welche zur Ausführung des Modells benötigt wird, wie beispielsweise die Größe des Arbeitsspeichers oder die Speicherkapazität einer GPU.
- **PerformanceInformation:** Hierbei handelt es sich um Informationen über die Inferenzzeit bzw. die zur Ausführung des Modells verwendete Hardware.
- **LiveMonitoring:** Diese Submodelcollection binhaltet mitlaufende Informaitonen über die Hardwareauslastung und die Laufzeit des Modells, allerdings auch Metriken um wie eingangs erwähnt einen möglichen Drift in den Eingabedaten erkennen zu können.

A Anhang: Herleitung der Abweichungsquadratsumme

In Abschnitt 3.3 wurde für die Parameter a und b der univariaten linearen Regression abgeleitet:

$$a = \mu_y - \frac{\delta_{xy}}{\delta_{xx}}\mu_x \qquad \text{und} \qquad b = \frac{\delta_{xy}}{\delta_{xx}},$$

wobei die δ_{xx}, δ_{yy} und δ_{xy} die Summen der quadratischen Abweichungen bzw. der Produkte der Abweichungen von den jeweiligen Mittelwerten sind (siehe auch weiter unten).

In Abschnitt 3.3 wurde nur das Ergebnis des Einsetzen dieser (Schätzungen der) Parameter a und b in die Formel für die Abweichungsquadratsumme angegeben. Die vollständige Ableitung — zum Zwecke der leichteren Prüfbarkeit recht kleinschrittig gehalten — lautet:

$$\begin{aligned}
E \;=\; E(a,b) \;&=\; \sum_{i=1}^{n}(a+bx_i-y_i)^2 \;=\; \sum_{i=1}^{n}(a^2+2abx_i-2ay_i-2bx_iy_i+b^2x_i^2+y_i^2)\\
&= na^2+2abs_x-2as_y-2bs_{xy}+b^2s_{xx}+s_{yy}\\
&= na^2+2nab\mu_x-2na\mu_y-2b\left(\delta_{xy}+n\mu_x\mu_y\right)+b^2\left(\delta_{xx}+n\mu_x^2\right)+\left(\delta_{yy}+n\mu_y^2\right)\\
&= na^2+2nab\mu_x-2na\mu_y-2b\delta_{xy}-2nb\mu_x\mu_y+b^2\delta_{xx}+nb^2\mu_x^2+\delta_{yy}+n\mu_y^2\\
&= n\left(\mu_y-\frac{\delta_{xy}}{\delta_{xx}}\mu_x\right)^2+2n\left(\mu_y-\frac{\delta_{xy}}{\delta_{xx}}\mu_x\right)\frac{\delta_{xy}}{\delta_{xx}}\mu_x-2n\left(\mu_y-\frac{\delta_{xy}}{\delta_{xx}}\mu_x\right)\mu_y\\
&\quad -2\frac{\delta_{xy}}{\delta_{xx}}\delta_{xy}-2n\frac{\delta_{xy}}{\delta_{xx}}\mu_x\mu_y+\frac{\delta_{xy}^2}{\delta_{xx}^2}\delta_{xx}+n\frac{\delta_{xy}^2}{\delta_{xx}^2}\mu_x^2+\delta_{yy}+n\mu_y^2\\
&= n\mu_y^2-2n\frac{\delta_{xy}}{\delta_{xx}}\mu_x\mu_y+n\frac{\delta_{xy}^2}{\delta_{xx}^2}\mu_x^2+2n\frac{\delta_{xy}}{\delta_{xx}}\mu_x\mu_y-2n\frac{\delta_{xy}^2}{\delta_{xx}^2}\mu_x^2-2n\mu_y^2+2n\frac{\delta_{xy}}{\delta_{xx}}\mu_x\mu_y\\
&\quad -2\frac{\delta_{xy}^2}{\delta_{xx}}-2n\frac{\delta_{xy}}{\delta_{xx}}\mu_x\mu_y+\frac{\delta_{xy}^2}{\delta_{xx}}+n\frac{\delta_{xy}^2}{\delta_{xx}^2}\mu_x^2+\delta_{yy}+n\mu_y^2\\
&= \delta_{yy}-\frac{\delta_{xy}^2}{\delta_{xx}} \;=\; \frac{1}{\delta_{xx}}(\delta_{xx}\delta_{yy}-\delta_{xy}^2) \;=\; \frac{1}{\delta_{xx}}\begin{vmatrix}\delta_{xx} & \delta_{xy}\\ \delta_{xy} & \delta_{yy}\end{vmatrix} \;=\; \frac{1}{\delta_{xx}}|\Delta|.
\end{aligned}$$

In dieser Ableitung wurden u.a. folgende Beziehungen benutzt:

$$\begin{aligned}
\mu_x = \tfrac{1}{n}s_x \quad &\Leftrightarrow \quad s_x = n\mu_x\\
\mu_y = \tfrac{1}{n}s_y \quad &\Leftrightarrow \quad s_y = n\mu_y\\
\delta_{xx} = s_{xx}-\tfrac{1}{n}s_x^2 \quad &\Leftrightarrow \quad s_{xx} = \delta_{xx}+\tfrac{1}{n}s_x^2 \;= \delta_{xx}+n\mu_x^2,\\
\delta_{yy} = s_{xx}-\tfrac{1}{n}s_x^2 \quad &\Leftrightarrow \quad s_{yy} = \delta_{yy}+\tfrac{1}{n}s_y^2 \;= \delta_{yy}+n\mu_y^2,\\
\delta_{xy} = s_{xy}-\tfrac{1}{n}s_xs_y \quad &\Leftrightarrow \quad s_{xy} = \delta_{xy}+\tfrac{1}{n}s_xs_y = \delta_{xy}+n\mu_x\mu_y.
\end{aligned}$$

Die Beziehung für δ_{xx} läßt sich folgendermaßen herleiten:

$$\begin{aligned}
\delta_{xx} \;&=\; \sum_{i=1}^{n}(x_i-\mu_x)^2 \;=\; \sum_{i=1}^{n}(x_i^2-2x_i\mu_x+\mu_x^2)\\
&= s_{xx}-2\mu_xs_x+n\mu_x^2 \;=\; s_{xx}-2\tfrac{1}{n}s_xs_x+n\left(\tfrac{1}{n}s_x\right)^2\\
&= s_{xx}-2\tfrac{1}{n}s_xs_x\tfrac{1}{n}s_x^2 \;=\; s_{xx}-\tfrac{1}{n}s_x^2.
\end{aligned}$$

Die beiden anderen Beziehungen, für δ_{yy} und δ_{xy}, ergeben sich völlig analog. Außerdem ergeben sich aus aus diesen Beziehungen über die Gleichungen

$$\sigma_{xx} = \tfrac{1}{n-1}\delta_{xx}, \qquad \sigma_{yy} = \tfrac{1}{n-1}\delta_{yy}, \qquad \sigma_{xy} = \tfrac{1}{n-1}\delta_{xy},$$

in denen die Varianzen und die Kovarianz mit Hilfe einer Bessel-Korrektur (Faktor $\frac{1}{n-1}$ statt $\frac{1}{n}$) berechnet werden, die Beziehungen

$$\begin{aligned}
\sigma_{xx} &= \tfrac{1}{n-1}\left(s_{xx} - \tfrac{1}{n}s_x^2\right) &\Leftrightarrow\quad s_{xx} &= (n-1)\sigma_{xx} + n\mu_x^2,\\
\sigma_{yy} &= \tfrac{1}{n-1}\left(s_{yy} - \tfrac{1}{n}s_y^2\right) &\Leftrightarrow\quad s_{yy} &= (n-1)\sigma_{yy} + n\mu_y^2,\\
\sigma_{xy} &= \tfrac{1}{n-1}\left(s_{xy} - \tfrac{1}{n}s_x s_y\right) &\Leftrightarrow\quad s_{xy} &= (n-1)\sigma_{xy} + n\mu_x\mu_y,
\end{aligned}$$

erhalten, die ebenfalls in Abschnitt 3.3 verwendet wurden.

B Anhang: Herleitung des Welford-Algorithmus

Der Welford-Algorithmus [Welford 1962] dient zum mitlaufenden Berechnen des Mittelwertes μ und der Summe δ^2 der quadratischen Abweichungen von diesem Mittelwert (aus der die Varianz σ^2 stets durch einfaches Teilen durch $n-1$ (Bessel-Korrektur) berechnet werden kann), wobei zusätzlich zu den fortzuschreibenden Größen (Mittelwert μ und Abweichungsquadratsumme δ^2), nur die Anzahl i der (bisher verarbeiteten) Werte und der jeweils neue, im nächsten Schritt einzubeziehende Wert benötigt werden.

Der Welford-Algorithmus arbeitet daher mit einem Index i, der die Werte 0 bis n durchläuft, und zwei Speichern, einem für den Mittelwert μ_i der ersten i Werte und einem für die Summe δ_i^2 der quadratischen Abweichungen der ersten i Werte von diesem Mittelwert μ_i.
Die mitlaufende Berechnung wird so durchgeführt:

1. Initialisiere $i = 0$, $\mu_0 = 0$ und $\delta_0^2 = 0$.
2. Für jeden Wert x_i, berechne (in dieser Reihenfolge):

$$\begin{aligned} \mu_{i+1} &\leftarrow \mu_i + \tfrac{1}{i+1}(x_i - \mu_i) \\ \delta_{i+1}^2 &\leftarrow \delta_i^2 + (x_i - \mu_{i+1})(x_i - \mu_i) \\ i &\leftarrow i+1 \end{aligned}$$

Die Formel für das Fortschreiben des Mittelwertes läßt sich sehr leicht herleiten, indem man einfach auf die Definition des Mittelwertes zurückgreift:

$$\begin{aligned} \mu_{i+1} &= \frac{1}{i+1}\sum_{k=0}^{i} x_k = \frac{1}{i+1}\left(x_i + \sum_{k=0}^{i-1} x_k\right) = \frac{x_i + i\cdot\mu_i}{i+1} \\ &= \frac{x_i + (i+1)\cdot\mu_i - \mu_i}{i+1} = \mu_i + \frac{x_i - \mu_i}{i+1}. \end{aligned}$$

Für die Herleitung der Formel zum Fortschreiben der Summe der quadratischen Abweichungen greift man zwar ebenfalls auf die Definition zurück, doch ist die Herleitung etwas komplizierter. Wir betrachten $\delta_j^2 = \sum_{k=0}^{j-1}(x_k - \mu_j)^2$. Dann können wir die Differenz zweier aufeinanderfolgender Summen δ_i^2 und δ_{i+1}^2 schreiben als:

$$\begin{aligned} \delta_{i+1}^2 - \delta_i^2 &= \sum_{k=0}^{i}(x_k - \mu_{i+1})^2 - \sum_{k=0}^{i-1}(x_k - \mu_i)^2 \\ &= (x_i - \mu_{i+1})^2 + \sum_{k=0}^{i-1}(x_k - \mu_{i+1})^2 - \sum_{k=0}^{i-1}(x_k - \mu_i)^2 \\ &= (x_i - \mu_{i+1})^2 + \sum_{k=0}^{i-1}\left(x_k^2 - 2x_k\mu_{i+1} + \mu_{i+1}^2 - x_k^2 + 2x_k\mu_i - \mu_i^2\right) \\ &= (x_i - \mu_{i+1})^2 + \sum_{k=0}^{i-1}\left((\mu_{i+1}^2 - \mu_i^2) - 2x_k(\mu_{i+1} - \mu_i)\right) \\ &= (x_i - \mu_{i+1})^2 + (\mu_{i+1} - \mu_i)\sum_{k=0}^{i-1}((\mu_{i+1} + \mu_i) - 2x_k) \\ &= (x_i - \mu_{i+1})^2 + (\mu_{i+1} - \mu_i)\left(\sum_{k=0}^{i-1}(\mu_{i+1} - x_k) + \underbrace{\sum_{k=0}^{i-1}(\mu_i - x_k)}_{=0}\right) \end{aligned}$$

$$
\begin{aligned}
&= (x_i - \mu_{i+1})^2 + (\mu_{i+1} - \mu_i)\left(\sum_{k=0}^{i-1}(\mu_{i+1} - x_k) + \underbrace{\sum_{k=0}^{i-1}(\mu_i - x_k)}_{=0}\right)\\
&= (x_i - \mu_{i+1})^2 + (\mu_{i+1} - \mu_i)\left(\underbrace{\sum_{k=0}^{i}(\mu_{i+1} - x_k)}_{=0} - (\mu_{i+1} - x_i)\right)\\
&= (x_i - \mu_{i+1})^2 + (\mu_{i+1} - \mu_i)(x_i - \mu_{i+1})\\
&= (x_i - \mu_{i+1}) \cdot ((x_i - \mu_{i+1}) + (\mu_{i+1} - \mu_i))\\
&= (x_i - \mu_{i+1}) \cdot (x_i - \mu_i).
\end{aligned}
$$

Wenn statt dieses Ergebnisses eine Formel gewünscht ist, in der nicht zwei aufeinanderfolgende Mittelwerte μ_i und μ_{i+1} auftreten, so kann man dieses Ergebnis weiter umformen zu

$$
\begin{aligned}
(x_i - \mu_{i+1}) \cdot (x_i - \mu_i) &= \left(x_i - \left(\mu_i + \frac{x_i - \mu_i}{i+1}\right)\right) \cdot (x_i - \mu_i)\\
&= \left((x_i - \mu_i) - \frac{x_i - \mu_i}{i+1}\right) \cdot (x_i - \mu_i)\\
&= \left(\tfrac{(i+1)(x_i-\mu_i)}{i+1} - \tfrac{x_i-\mu_i}{i+1}\right) \cdot (x_i - \mu_i)\\
&= \tfrac{(i+1)(x_i-\mu_i) - (x_i-\mu_i)}{i+1}(x_i - \mu_i)\\
&= \tfrac{i\cdot(x_i-\mu_i)}{i+1}(x_i - \mu_i)\\
&= \tfrac{i}{i+1}(x_i - \mu_i)^2.
\end{aligned}
$$

Beide Darstellungen eignen sich gut für eine Implementierung. Mit der ersten geht man so vor:

$$
\begin{aligned}
d &\leftarrow x_i - \mu\\
i &\leftarrow i + 1\\
\mu &\leftarrow \mu + \tfrac{d}{i}\\
\delta^2 &\leftarrow \delta^2 + d(x_i - \mu)
\end{aligned}
$$

Man beachte, dass durch ein temporäres Merken von $x_i - \mu$ gegenüber einem Merken des alten μ_i eine Operation gespart werden kann. Eine weitere Operation wird durch das frühzeitige Erhöhen von i gespart (da $i + 1$ auch zur Berechnung von μ benötigt wird).

Mit der zweiten Darstellung kann man so vorgehen:

$$
\begin{aligned}
k &\leftarrow i\\
i &\leftarrow i + 1\\
d &\leftarrow x_i - \mu\\
\mu &\leftarrow \mu + \tfrac{d}{i}\\
\delta^2 &\leftarrow \delta^2 + \tfrac{k}{i}d^2
\end{aligned}
$$

Man beachte wieder, wie durch die Reihung der Zuweisungen Operationen eingespart werden.

Obwohl die zweite Variante auf den ersten Blick etwas aufwendiger erscheint, ist sie für den hier vorliegenden Zweck günstiger, da ja zwei Mittelwerte μ_x und μ_y, zwei Summen δ_{xx} und δ_{yy} von quadratischen Abweichungen von Mittelwerten und eine Summe δ_{xy} von Produkten von Abweichungen von Mittelwerten fortgeschrieben werden müssen. Alle diese Fortschreibungen können sich die ersten beiden Zuweisungen (an k und i) teilen, die daher nur einmal ausgeführt werden müssen. Siehe dazu die Python-Implementierung im Anhang C.

C Anhang: Quelltexte zur Sensordatenkomprimierung

Die folgenden Quelltexte zeigen Beispielimplementierungen für die mitlaufende Filterung (*online filtering*) von Sensordaten zur Übertragung von einer Steuerung auf einen Auswerterechner, durch die die Daten unter deutlicher Komprimierung durch einen Polygonzug angenähert werden. Die Implementierungen sind Python-Klassen, wodurch sie für den direkten Einsatz nicht geeignet sind, da es kaum möglich ist, auf einer Steuereinheit eine Python-Umgebung zu installieren. Außerdem ist die Programmiersprache Python nicht für ihre Laufzeiteffizienz bekannt, so dass selbst dann, wenn eine Installation möglich wäre, kaum die nötige Geschwindigkeit erreicht würde. Diese Implementierungen zeigen daher nur das Prinzip, müßten aber für den praktischen Einsatz in einer Programmiersprache wie C/C++ reimplementiert werden.

C.1 Filterung mit Wurzel aus mittlerem quadratischen Fehler

```
class RMSEFilter:               # root mean squared error filter

    def __init__ (self, rmsemax, yrmax):
        self.msemax = rmsemax**2# limit for mean squared error
        self.yrmax  = yrmax     # limit for y-range for horizontal lines
        self.anchor = None      # anchor    coordinates
        self.cand   = None      # candidate coordinates

    def init (self, x,y):       # initialize aggregation variables
        self.min = self.max = y # initialize range of y-values,
        self.n   = 1            # point counter,
        self.mx,self.my = x,y   # means and sums of squared deviations
        self.dxx = self.dyy = self.dxy = 0

    def update (self, x,y):     # update aggregation variables
        if y < self.min: self.min = y   # update y-range
        if y > self.max: self.max = y   # with current point
        a         = self.n      # get old number of points
        self.n   += 1           # count the current point
        a        /= self.n      # divide by new number of points
        dx        = x-self.mx   # get difference to mean in x-direction
        self.mx  += dx/self.n   # update mean and
        self.dxx += a*dx*dx     # sum of squared deviations
        # self.dxx += dx*(x-self.mx) # standard Welford computation
        dy        = y-self.my   # get difference to mean in y direction
        self.my  += dy/self.n   # update mean and
        self.dyy += a*dy*dy     # sum of squared deviations
        # self.dyy += dy*(y-self.my) # standard Welford computation
        self.dxy += a*dx*dy     # update the product of deviations

    def shift (self, x,y):      # shift candidate to anchor
        self.anchor = self.cand # make candidate the anchor
        self.cand   = x,y       # make new point the candidate
        self.init(*self.anchor) # initialize aggregation variables with
        self.update(x,y)        # new anchor and update with new point

    def extreme (self, y):      # check for an extreme point (min or max)
```



```
        ax,ay = self.anchor     # compare signs of cy-ay and y-cy
        cx,cy = self.cand       # (needs to be written in a funny way)
        return (cy > ay) - (cy < ay) != (y > cy) - (y < cy)

    def filter (self, x,y):     # filter current point
        if self.anchor is None: # if there is no anchor,
            self.anchor = x,y   # set the anchor
            self.init(x,y)      # initialize aggregation variables
            return x,y          # return anchor / first point
        if self.cand is None:   # if there is no candidate,
            self.cand = x,y     # set the candidate
            self.update(x,y)    # update aggregation variables
            return None         # return 'no output point'
        c   = self.cand         # get the current candidate
        yro = self.max-self.min # and the old y-range
        self.update(x,y)        # update aggregation variables
        yrn = self.max-self.min # get the new y-range
        if yrn <= self.yrmax:   # if (almost) horizontal line,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        if yro <= self.yrmax:   # if end of (almost) horizontal line,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
        if self.extreme(y):     # if at an extreme point (max or min),
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
        mse = (self.dyy -self.dxy *(self.dxy /self.dxx)) /self.n
        if mse <= self.msemax:  # if mean squared error is small enough,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        else:                   # if mean squared error is too large,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate

    def term (self):            # terminate processing
        return self.cand        # return the current candidate
```

C.2 Filterung mit quadriertem Pearsonschen Korrelationkoeffizienten

```
class CorrFilter:               # correlation filter

    def __init__ (self, cormin, yrmax):
        self.cormin = cormin**2 # limit for squared Pearson correlation
        self.yrmax  = yrmax     # limit for y-range for horizontal lines
        self.anchor = None      # anchor    coordinates
        self.cand   = None      # candidate coordinates

    def init (self, x,y):       # initialize aggregation variables
        self.min = self.max = y # initialize range of y-values,
        self.n   = 1            # point counter,
        self.mx,self.my = x,y   # means and sums of squared deviations
```



```
        self.dxx = self.dyy = self.dxy = 0

    def update (self, x,y):     # update aggregation variables
        if y < self.min: self.min = y   # update y-range
        if y > self.max: self.max = y   # with current point
        a         = self.n      # get old number of points
        self.n   += 1           # count the current point
        a        /= self.n      # divide by new number of points
        dx        = x-self.mx   # get difference to mean in x-direction
        self.mx  += dx/self.n   # update mean and
        self.dxx += a*dx*dx     # sum of squared deviations
        # self.dxx += dx*(x-self.mx) # standard Welford computation
        dy        = y-self.my   # get difference to mean in y direction
        self.my  += dy/self.n   # update mean and
        self.dyy += a*dy*dy     # sum of squared deviations
        # self.dyy += dy*(y-self.my) # standard Welford computation
        self.dxy += a*dx*dy     # update the product of deviations

    def shift (self, x,y):      # shift candidate to anchor
        self.anchor = self.cand # make candidate the anchor
        self.cand   = x,y       # make new point the candidate
        self.init(*self.anchor) # initialize aggregation variables with
        self.update(x,y)        # new anchor and update with new point

    def extreme (self, y):      # check for an extreme point (min or max)
        ax,ay = self.anchor     # compare signs of cy-ay and y-cy
        cx,cy = self.cand       # (needs to be written in a funny way)
        return (cy > ay) - (cy < ay) != (y > cy) - (y < cy)

    def filter (self, x,y):     # filter current point
        if self.anchor is None: # if there is no anchor,
            self.anchor = x,y   # set the anchor
            self.init(x,y)      # initialize aggregation variables
            return x,y          # return anchor / first point
        if self.cand is None:   # if there is no candidate,
            self.cand = x,y     # set the candidate
            self.update(x,y)    # update aggregation variables
            return None         # return 'no output point'
        c   = self.cand         # get the current candidate
        yro = self.max-self.min # and the old y-range
        self.update(x,y)        # update aggregation variables
        yrn = self.max-self.min # get the new y-range
        if yrn <= self.yrmax:   # if (almost) horizontal line,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        if yro <= self.yrmax:   # if end of (almost) horizontal line,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
        if self.extreme(y):     # if at an extreme point (max or min),
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
```



```
        r = (self.dxy/self.dxx) *(self.dxy/self.dyy)
        if r >= self.cormin:    # if squared correlation is large enough,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        else:                   # if mean squared error is too large,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate

    def term (self):            # terminate processing
        return self.cand        # return the current candidate
```

C.3 Filterung mit mittlerem absoluten Fehler

```
class MAEFilter:                # mean absolute error filter

    def __init__ (self, maemax, yrmax):
        self.maemax = maemax    # limit for mean squared error
        self.yrmax  = yrmax     # limit for y-range for horizontal lines
        self.anchor = None      # anchor    coordinates
        self.cand   = None      # candidate coordinates

    def init (self, x,y):       # initialize aggregation variables
        self.min = self.max = y # initialize range of y-values,
        self.n   = 1            # point counter,
        self.sae = 0            # and sum of absolute errors

    def update (self, x,y):     # update the candidate
        if y < self.min: self.min = y   # update y-range
        if y > self.max: self.max = y   # with current point
        self.n += 1             # count the candidate point

    def shift (self, x,y):      # shift candidate to anchor
        self.anchor = self.cand # make candidate the anchor
        self.cand   = x,y       # make new point the candidate
        self.init(*self.anchor) # initialize aggregation variables with
        self.update(x,y)        # new anchor and update with new point

    def extreme (self, y):      # check for an extreme point (min or max)
        ax,ay = self.anchor     # compare signs of cy-ay and y-cy
        cx,cy = self.cand       # (needs to be written in a funny way)
        return (cy > ay) - (cy < ay) != (y > cy) - (y < cy)

    def filter (self, x,y):     # filter current point
        if self.anchor is None: # if there is no anchor,
            self.anchor = x,y   # set the anchor
            self.init(x,y)      # initialize aggregation variables
            return x,y          # return anchor / first point
        if self.cand is None:   # if there is no candidate,
            self.cand = x,y     # set the candidate
            self.update(x,y)    # update aggregation variables
            return None         # return 'no output point'
```



```
        c   = self.cand          # get the current candidate
        yro = self.max-self.min # and the old y-range
        self.update(x,y)         # update aggregation variables
        yrn = self.max-self.min # get the new y-range
        if yrn <= self.yrmax:    # if (almost) horizontal line,
            self.cand = x,y      # replace the candidate
            return None          # return 'no output point'
        if yro <= self.yrmax:    # if end of (almost) horizontal line,
            self.shift(x,y)      # shift candidate to anchor
            return c             # return previous candidate
        if self.extreme(y):      # if at an extreme point (max or min),
            self.shift(x,y)      # shift candidate to anchor
            return c             # return previous candidate
        ax,ay = self.anchor      # get current anchor
        cx,cy = self.cand        # and current candidate
        d = abs((x-ax) / (cx-ax) * (cy-ay) +ay -y)
        # compute y-distance of new point (x,y) from old line (ax,ay)--(cx,cy)
        self.sae += (self.n-2)/2 *d # update sum of squared errors
        mae = self.sae/self.n    # compute mean absolute error
        if mae <= self.maemax:   # if MAE is small enough,
            self.cand = x,y      # replace the candidate
            return None          # return 'no output point'
        else:                    # if mean squared error is too large,
            self.shift(x,y)      # shift candidate to anchor
            return c             # return previous candidate

    def term (self):             # terminate processing
        return self.cand         # return the current candidate
```

C.4 Filterung mit relativem absoluten Fehler

```
class RAEFilter:                 # relative absolute error filter

    def __init__ (self, raemax, yrmax):
        self.raemax = raemax     # limit for mean squared error
        self.yrmax  = yrmax      # limit for y-range for horizontal lines
        self.anchor = None       # anchor    coordinates
        self.cand   = None       # candidate coordinates

    def init (self, x,y):        # initialize aggregation variables
        self.min = self.max = y # initialize range of y-values,
        self.n   = 1             # point counter,
        self.sae = 0             # and sum of absolute errors

    def update (self, x,y):      # update the candidate
        if y < self.min: self.min = y   # update y-range
        if y > self.max: self.max = y   # with current point
        self.n += 1              # count the candidate point

    def shift (self, x,y):       # shift candidate to anchor
        self.anchor = self.cand # make candidate the anchor
```

```
        self.cand   = x,y       # make new point the candidate
        self.init(*self.anchor) # initialize aggregation variables with
        self.update(x,y)        # new anchor and update with new point

    def extreme (self, y):      # check for an extreme point (min or max)
        ax,ay = self.anchor     # compare signs of cy-ay and y-cy
        cx,cy = self.cand       # (needs to be written in a funny way)
        return (cy > ay) - (cy < ay) != (y > cy) - (y < cy)

    def filter (self, x,y):     # filter current point
        if self.anchor is None: # if there is no anchor,
            self.anchor = x,y   # set the anchor
            self.init(x,y)      # initialize aggregation variables
            return x,y          # return anchor / first point
        if self.cand is None:   # if there is no candidate,
            self.cand = x,y     # set the candidate
            self.update(x,y)    # update aggregation variables
            return None         # return 'no output point'
        c   = self.cand         # get the current candidate
        yro = self.max-self.min # and the old y-range
        self.update(x,y)        # update aggregation variables
        yrn = self.max-self.min # get the new y-range
        if yrn <= self.yrmax:   # if (almost) horizontal line,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        if yro <= self.yrmax:   # if end of (almost) horizontal line,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
        if self.extreme(y):     # if at an extreme point (max or min),
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate
        ax,ay = self.anchor     # get current anchor
        cx,cy = self.cand       # and current candidate
        d = abs((x-ax) /(cx-ax) *(cy-ay) +ay -y)
        # compute y-distance of new point (x,y) from old line (ax,ay)--(cx,cy)
        self.sae += (self.n-2)/2 *d # update sum of squared errors
        rae = self.sae/self.n/yrn # compute relative mean absolute error
        if rae <= self.raemax:  # if RAE is small enough,
            self.cand = x,y     # replace the candidate
            return None         # return 'no output point'
        else:                   # if mean squared error is too large,
            self.shift(x,y)     # shift candidate to anchor
            return c            # return previous candidate

    def term (self):            # terminate processing
        return self.cand        # return the current candidate
```

Literatur

[Aminikhanghahi and Cook 2017] A. Aminikhanghahi and D.J. Cook. A Survey of Methods for Time Series Change Point Detection. *Knowledge and Information Systems* 51(2):339–367. Springer Nature, New York, NY, USA 2017

[Chambers and Zaharia 2018] B. Chambers and M. Zaharia. *Spark: The Definitive Guide: Big Data Processing Made Simple*. O'Reilly Media Inc., Sebastopol, CA, USA 2018.

[Chapman *et al.* 1999] P. Chapman, J. Clinton, T. Khabaza, T. Reinartz, and R. Wirth. *The CRISP-DM Process Model*. NCR Systems Engineering Copenhagen / DaimlerChrysler AG / SPSS Inc. / OHRA Verzekeringen en Bank Groep B.V., USA / Denmark / Germany / Netherlands 1999.

[Chapman *et al.* 2000] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. *CRISP-DM 1.0 – Step-by-Step Data Mining Guide*. NCR Systems Engineering Copenhagen / DaimlerChrysler AG / SPSS Inc. / OHRA Verzekeringen en Bank Groep B.V., USA / Denmark / Germany / Netherlands 1999.
`http://www.statoo.com/CRISP-DM.pdf`

[Fillbrunn 2014] A. Fillbrunn. *Performante Auswertung von Vorhersagemodellen*. Diplomarbeit, Universität Konstanz, Deutschland 2014.

[Gama *et al.* 2014] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A Survey on Concept Drift Adaptation. *ACM Computing Surveys (CSUR)* 46(4):1–37. ACM, New York, NY, USA 2014

[Ho 2010] S. Ho and H. Weechsler A martingale framework for detecting changes in data streams by testing exchangeability *IEEE transactions on pattern analysis and machine intelligence* 32(12):2113,2127 USA 2010

[Luu 2021] H. Luu. *Beginning Apache Spark 3: With DataFrame, Spark SQL, Structured Streaming, and Spark Machine Learning Library*. Springer Nature, New Yor, NY, USA 2021

[Namoano *et al.* 2019] B. Namoano, A. Starr, C. Emmanouilidis, and R.C. Cristobal. Online Change Detection Techniques in Time Series: An Overview. *IEEE Int. Conf. on Prognostics and Health Management (ICPHM 2019)*, 1–10. IEE Press, Piscataway, NJ, USA, 2019

[Piatetsky-Shapiro 2014] G. Piatetsky-Shapiro. KDnuggets Methodology Poll: What main methodology are you using for your analytics, data mining, or data science projects? KDnuggets 2014

[Knuth 1998] D.E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, Boston, Ma, USA 1998

[Python 2022] G. van Rossum *et al. Python Language Reference*. Python Software Foundation, Wilmington, DE, USA 2001–2022
`https://www.python.org/`

[Rauh *et al.* 2022] L. Rauh, S. Gärtner, D. Brandt, M. Oberle, D. Stock, and T. Bauernhansl. Towards AI Lifecycle Management in Manufacturing Using the Asset Administration Shell (AAS). *Procedia CIRP* 107:576–581. `10.1016/j.procir.2022.05.028`

[R 2022] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria 1993–2022
`https://www.R-project.org/`

[Thilo 2002] M.O. Thilo. Evaluierung des Schnittstellenstandards Predictive Model Markup Language (PMML) für Data Mining. Diplomarbeiten Agentur diplom.de, Deutschland 2002

[Vovk 2003] V. Vovk, I. Nouretdinov, and A. Gammerman Testing exchangeability on-line *Procedia ICML* 768–775 2003

[Welford 1962] B.P. Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4(3):419-–420. Taylor & Francis Group, Milton Park, Abingdon-on-Thames, United Kingdom 1962

Impressum

Titel	Semantic Integration Patterns for AI
Bezeichnung	Deliverable 2.3b (i-Twin)
Autoren	Christian Borgelt, Sebastian Baron, Marleen Bahe
Dateiname	D23b_Semantic_Integration_Patterns_for_AI
Publikationsstatus	Public
Letzte Änderung	13.03.2024 von Georg Güntner
Kontakt	Paris Lodron Universität Salzburg Herr Univ.-Prof. Dr.-Ing. Christian Borgelt Hellbrunnerstraße 34/ Jakob-Haringer-Str. 2 | 5020 Salzburg | Austria T +43 (0) 662-8044-(5311 or 6351) | christian.borgelt@plus.ac.at
Copyright	Projektkonsortium i-Twin, Jänner 2024 p.a. Salzburg Research Forschungsgesellschaft m.b.H. Jakob Haringer Straße 5/3 | 5020 Salzburg | Austria T +43-662-2288-401 | i-twin-office@salzburgresearch.at

Das Projekt i-Twin wird gefördert vom BMK und von der FFG aus Mitteln des Programms IKT der Zukunft.