Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Finn V. Dohrn

Adaptives maschinelles Lernen mit Mixture of Experts

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Finn V. Dohrn

Adaptives maschinelles Lernen mit Mixture of Experts

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Marina Tropmann-Frick
Zweitgutachter: Prof. Dr.-Ing. Olaf Zukunft

Eingereicht am: 28. März 2025

Finn V. Dohrn

Thema der Arbeit

Adaptives maschinelles Lernen mit Mixture of Experts

Stichworte

Kontinuierliches adaptives Lernen, Online Maschinelles Lernen, Streamingdaten, Dynamische Umgebung, Mixture of Experts

Kurzzusammenfassung

Bis 2025 werden 41,6 Milliarden IoT-Geräte erwartet. Dadurch wächst die Menge an kontinuierlich generierten Daten rasant. Klassische maschinelle Lernverfahren, die auf statischen Datensätzen trainiert werden, sind nicht in der Lage, sich kontinuierlich an dynamische Umgebungen anzupassen. Problematisch ist dabei der Konzeptdrift, der dazu führt, dass Modelle im Laufe der Zeit an Genauigkeit verlieren.

Zur Steigerung der Robustheit und Genauigkeit des kontinuierlichen Lernens stellt diese Arbeit einen neuartigen Ansatz vor, der auf den Erkenntnissen einer systematischen Literaturrecherche basiert. Dieser Ansatz kombiniert Online Maschinelles Lernen (OML) mit Mixture of Experts (MoE) in einer neuartigen inkrementellen MoE-Architektur. Zur Evaluierung dieser hybriden Architektur werden verschiedene Experimente zur Regression und Klassifikation vorgestellt, die durch geeignete Methoden bewertet werden.

Diese Arbeit präsentiert die Implementierung des Frameworks `riverMoE`, welches die konzipierte hybride Architektur als Erweiterung eines bestehenden OML-Frameworks realisiert. Die Ergebnisse der durchgeführten Experimente zeigen, dass die gezielte gewichtete Kombination von Einzelprognosen auch mit inkrementellen Lernalgorithmen zu präziseren Gesamtprognosen führen kann. Erwartungsgemäß erfordert die Nutzung neuronaler Komponenten einen erhöhten Rechen- und Speicheraufwand. Zudem zeigen die Experimente in adaptiven Umgebungen eine höhere Stabilität gegenüber Konzeptdrift, die jedoch von der Initialisierung abhängt.

Auf Grundlage der Ergebnisse wird empfohlen, die bisher umgesetzten inkrementellen MoE-Varianten um zusätzliche Ansätze zu erweitern und weitere Experimente durchzuführen. Darüber hinaus sollte die Erklärbarkeit des Gates in der Kombination der einzelnen Experten eingehender untersucht werden.

Finn V. Dohrn

Title of the paper

Adaptive Machine Learning with Mixture of Experts

Keywords

Continual adaptive Learning, Online Machine Learning, Streaming data, Dynamic Environments, Mixture of Experts

Abstract

41.6 billion IoT devices are expected by 2025. As a result, the amount of continuously generated data is growing rapidly. Traditional machine learning methods that are trained on static data sets and are not able to continuously adapt to dynamic environments. The problem here is concept drift, which leads to models losing accuracy over time.

In order to make continuous learning more robust and accurate, this paper presents a novel approach based on the findings of a systematic literature review. This approach combines Online Machine Learning (OML) with Mixture of Experts (MoE) in a novel incremental MoE architecture. To evaluate this hybrid architecture, various regression and classification experiments are presented and evaluated using appropriate methods.

This paper presents the implementation of the framework `riverMoE`, which realizes the designed hybrid architecture as an extension of an existing OML framework. The results of the experiments show that the targeted weighted combination of individual forecasts can also lead to more precise overall forecasts when using incremental learning algorithms. As expected the use of neural components requires increased computing and memory resources. Furthermore, the experiments in adaptive environments show a higher stability against concept drift which depends on the initialization.

Based on the results, it is recommended to extend the existing implemented MoE variants and to conduct further experiments. In addition, the explainability of the gate in the combination of the individual experts should be examined in more detail.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

ADWIN Adaptive Windowing. 17, 48

API Application Programming Interface. 45

ARIMA Autoregressive Integrated Moving Average. 37

BCE Binary Cross Entropy. 20, 45

BML Batch maschinelles Lernen. 1, 11, 73

CART Classificiation And Regression Tree. 12

CE Cross Entropy. xi, 20, 45, 62

CMA Cumulative Moving Average. 16, 55, 65

CNN Convolutional Neural Network. 21

DMoE Deep Mixture-of-Experts. 25

DOP Dynamisches Optimierungs-Problem. 6, 40, 46, 63

EM Erwartungs-Maximierung. 24

EWMA Exponentially Weighted Moving Average. 16

FIFO First-In-First-Out. 7

FN False Negative. 14

FNN Feedforward Neural Network. 20, 32

FP False Positive. 14

GBM Gradient Boosting Machines. 37

GCN Graph Convolutional Network. 37

HAT Hoeffding Adaptive Tree. 47

HME Hierarchisches Mixture of Experts. 24

HPO Hyperparameteroptimierung. 13, 73

HT Hoeffding Tree. 47

IDC International Data Corporation. 2

IoT Internet of Things. 2, 78

KI Künstliche Intelligenz. 1, 5

KNN Künstliches Neuronales Netz. 19, 32

KS Kolmogorov-Smirnov. 18

KSWIN Kolmogorov-Smirnov Windowing. ix, 18

LLM Large Language Model. 26

LSTM Long Short Term Memory. 32

MAE Mean Absolute Error. 13

Mini-BML Mini-Batch maschinelles Lernen. 11

ML Maschinelles Lernen. 1, 9

MLP Mehrschichtiges Perzeptron. 19

MMoE Multi-Mixture-of-Experts. 30, 34

MoE Mixture of Experts. 1, 19, 78

MSLE Mean Squared Logarithmic Error. 13

MSE Mean Squared Error. 20, 45

MTL Multi-Task-Lernen. 10, 21, 30, 34, 80

NAS Neural Architecture Search. 35

NTG Negative Transfer Gap. 35

OML Online maschinelles Lernen. 1, 5, 9, 31, 78

PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses. 28

RMSLE Root Mean Squared Logarithmic Error. 13

RMSE Root Mean Squared Error. 13

RNN Recurrent Neural Network. 20, 32

ROC-AUC Receiver Operating Characteristic - Area under Curve. 15

SAMoE Streaming Adaptive Mixtures of Experts. 44

SGD Stochastic Gradient Descent. 20, 41, 45, 47

SVM Support Vector Machines. 37

TN True Negative. 14

TP True Positive. 14

WCMA Weighted Cumulative Moving Average. 16

1. Einleitung

Künstliche Intelligenz (KI) und besonders *Maschinelles Lernen* (ML) hat sich in den vergangenen Jahren rasant weiterentwickelt und findet Anwendung in einer Vielzahl von Bereichen, darunter automatisierte Entscheidungsfindung, natürliche Sprachverarbeitung und Bildverarbeitung (Fritz, 2022, S. 3f.). Viele klassische Lernverfahren basieren jedoch auf dem sogenannten *Batch maschinelles Lernen* (BML), bei dem das Modell mit einem festen Datensatz trainiert wird und anschließend statisch bleibt. In dynamischen Umgebungen, in denen kontinuierlich neue Daten verfügbar sind, führt dies zu Herausforderungen: Neue Informationen können nicht effizient integriert werden, und das erneute Training mit einer wachsenden Datenmenge ist rechenintensiv und teuer (Bartz-Beielstein und Bartz, 2023, S. 10f.).

Ein vielversprechender Lösungsansatz ist *Online maschinelles Lernen* (OML), dass es Modellen ermöglicht, sich fortlaufend an neue Daten anzupassen, ohne die gesamte Modellarchitektur neu trainieren zu müssen. Dieses inkrementelle Lernen ist insbesondere in Szenarien mit veränderlichen Datenverteilungen von Vorteil, da Modelle kontinuierlich an neue Gegebenheiten angepasst werden können (Bartz-Beielstein und Bartz, 2023, S. 11ff.). Ein weiteres aktuelles und relevantes Forschungsfeld ist der Ansatz *Mixture of Experts* (MoE), der sich bereits in großen Sprachmodellen von beispielsweise *Google* etabliert hat (N. Du et al., 2022). Diese Architektur kombiniert mehrere spezialisierte Expertenmodelle, die durch einen Gating-Mechanismus dynamisch für verschiedene Eingaben aktiviert werden (Jacobs et al., 1991). Dadurch können komplexe Probleme modularisiert und effizienter gelöst werden. Obwohl beide Methoden – OML als inkrementelles Lernen und MoE – unabhängig voneinander erforscht werden, gibt es bisher nur wenige Arbeiten, die eine Kombination dieser Ansätze untersuchen.

1.1. Problemstellung und Motivation

Eine solche Verbindung könnte erhebliche Vorteile bringen: Während das inkrementelle Lernen die kontinuierliche Anpassung an neue Daten ermöglicht, kann MoE dazu beitragen, das Lernen gezielter zu strukturieren und unterschiedliche Experten für verschiedene Datensegmente oder Aufgaben zu spezialisieren. Dadurch könnte ein Modell entstehen, das sowohl flexibel als auch

skalierbar ist und zugleich Probleme wie das Vergessen von bereits gelerntem („Katastrophales Vergessen“) oder ineffizientes Ressourcenmanagement adressiert.

Forschung in dieser Richtung ist notwendig, da die Menge an Daten in vielen Bereichen exponentiell anwächst. Die globale Datenmenge wird nach Reinsel, Gantz und Rydning (2018) bis zu dem Jahr 2025 auf bis zu 175 Zettabyte (1 Zettabyte $\hat{=} 1 \cdot 10^{21}$ Bytes) geschätzt. Der Bericht erfasst zusätzlich den Anteil von Echtzeitdaten. Ein aktuellerer Bericht, der International Data Corporation (IDC) von Wright (2024) kommt auf 182 Zettabytes Gesamtmenge bis 2025. Die Abbildung 1.1 zeigt die kombinierten Daten von 2010 bis 2025 in Zettabyte.

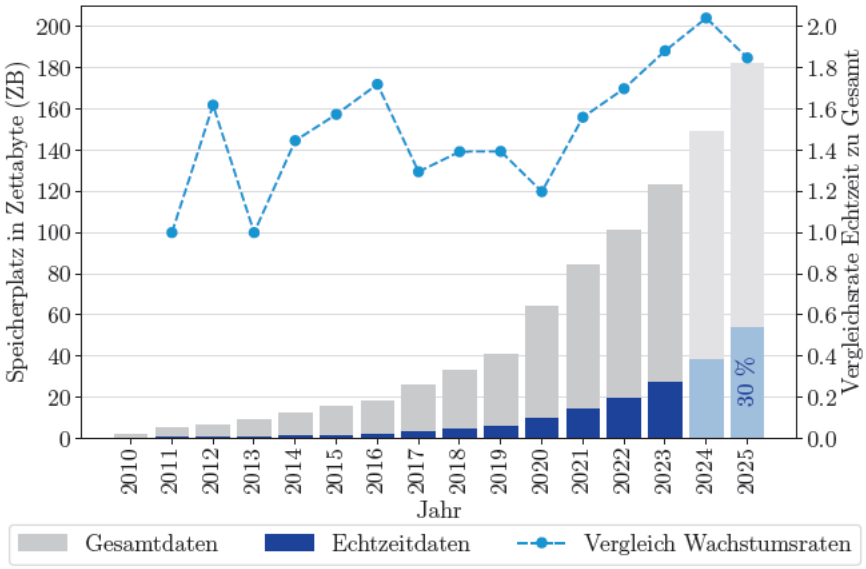


Abbildung 1.1.: Weltweiter Speicherplatzbedarf insgesamt und für Echtzeitdaten von 2010 bis 2025 in Zettabyte. Die Daten von 2024 bis 2025 sind geschätzt.

Die Gesamtmengen von 2024 bis 2025 wurden in beiden Berichten geschätzt. Der Anteil der Echtzeitdaten ist im Jahr 2025 bei 30 %. Wird die jährliche Wachstumsrate des Verhältnisses der Gesamtdaten zu den Echtzeitdaten betrachtet, kann erkannt werden, dass der Anteil der Echtzeitdaten im Verhältnis stärker wächst. Der Bedarf an der Verarbeitung von Echtzeitdaten könnte damit ebenfalls steigen. Das liegt zum großen Teil daran, dass immer mehr Geräte mit dem Internet verbunden sind. 41,6 Milliarden *Internet of Things* (IoT) Geräte werden bis 2025 erwartet, welche zum täglichen Wachstum von Datenmengen beitragen. In Autos, Maschinen oder im Smarthome sind viele Sensoren verbaut, die laufend große Datenmengen erzeugen. Kameras, die beim Auto beispielsweise für Fahrerassistenzsysteme verwendet werden, können pro Stunde bis zu drei Terabyte an Daten erzeugen und speichern (Fritz, 2022, S. 59).

Das Training mit diesen Daten muss teilweise in Echtzeit erfolgen, um die wachsenden Datenmengen zu bewältigen. Im klassischen BML ist dies nicht möglich, da das Modell nach dem Training nicht mehr verändert werden kann. Beim Eintreffen von neuen Daten muss das Modell neu trainiert werden (Bartz-Beielstein und Bartz, 2023, S. 8.). Dies erzeugt einen hohen Rechenaufwand, dauert sehr lange und kostet viel Energie. Inkrementelles Lernen ist daher eine Lösung, um die Modelle kontinuierlich an neue Daten anzupassen. Das Hauptproblem klassischer inkrementeller Lernansätze ist, dass neue Daten das Modell oft zu stark beeinflussen können. Gleichzeitig können inkrementelle Modelle Schwierigkeiten haben, neue Konzepte effizient zu integrieren, ohne an Stabilität zu verlieren (Bartz-Beielstein und Bartz, 2023, S. 41ff.).

MoE bietet eine mögliche Lösung für dieses Problem, indem es verschiedene Experten für unterschiedliche Datenbereiche oder Lernaufgaben trainiert. Ein Gating-Mechanismus entscheidet dynamisch, welcher Experte für eine bestimmte Eingabe zuständig ist, wodurch das Modell insgesamt robuster und genauer sein kann. Allerdings bringen klassische MoE-Modelle ebenfalls Herausforderungen mit sich, darunter ein gut funktionierender Gating-Mechanismus (Jacobs et al., 1991). Zudem wurde MoE bislang überwiegend in statischen Umgebungen erforscht und selten mit OML kombiniert.

1.2. Zielsetzung

Die Kombination dieser beiden Methoden könnte daher neue Möglichkeiten für adaptive, spezialisierte und kontinuierlich lernende Modelle eröffnen. Durch eine gezielte Einbindung von MoE in ein inkrementelles Lernsystem könnte die Spezialisierung von Experten dazu beitragen, den Einfluss neuer Daten besser zu steuern und das Risiko des Wissensverlusts zu reduzieren. Gleichzeitig könnte das Gating-Netzwerk so gestaltet werden, dass es neue Konzepte erkennt und entscheidet, ob bestehende Experten angepasst oder neue Experten hinzugefügt werden sollten.

Das Ziel dieser Arbeit ist daher die Entwicklung und Evaluierung eines kombinierten Ansatzes aus inkrementellem Lernen und Mixture of Experts, um die jeweiligen Stärken beider Methoden zu vereinen und ihre individuellen Schwächen zu adressieren. Konkret verfolgt diese Arbeit folgende Ziele:

1. Entwicklung eines hybriden Frameworks, das kontinuierlich neue Daten integriert und dabei Experten spezialisiert.
2. Optimierung des Gating-Mechanismus, sodass dieser sich dynamisch an veränderte Datenverteilungen anpasst.

3. Experimentelle Evaluierung, um zu prüfen, inwiefern der entwickelte Ansatz bestehende inkrementelle Lernverfahren hinsichtlich Genauigkeit, Effizienz und Adaptivität übertrifft.

Die Forschungsfragen, passend zu den gesetzten Zielen dieser Arbeit, werden im Abschnitt 3.3 konkret formuliert. Diese Arbeit trägt damit bei, eine Forschungslücke an der Schnittstelle von kontinuierlichem Lernen und modularen, spezialisierten Expertenmodellen zu schließen. Die gewonnenen Erkenntnisse sollen nicht nur zur theoretischen Weiterentwicklung dieser Ansätze beitragen, sondern auch praktische Anwendungen für reale Datenszenarien aufzeigen. Demzufolge ist die Implementierung eines produktiv nutzbaren Frameworks ein Teil dieser Arbeit.

1.3. Aufbau der Arbeit

Diese Arbeit gliedert sich in mehrere zentrale Kapitel. Zunächst werden in **Kapitel 2** die theoretischen Grundlagen erörtert, darunter Adaptivität, Big Data und Online-Maschinelles Lernen. Ein besonderer Fokus liegt auf der Drifterkennung und -behandlung sowie der Expertenmischung als Kernkonzept des adaptiven maschinellen Lernens mit Mixture of Experts.

Der Stand der Forschung wird in **Kapitel 3** durch eine systematische Literaturrecherche erfasst, gefolgt von einer qualitativen Analyse relevanter Arbeiten zu Gating-Mechanismen, Multi-Task-Experten und adaptivem Verhalten innerhalb MoE-Architekturen. Daraus resultierend werden bestehende Forschungslücken identifiziert.

Die Methodik in **Kapitel 4** beschreibt die Entwicklung einer inkrementellen MoE-Architektur, die in einem neuartigen Framework umgesetzt wurde. Das Framework kann als Erweiterung eines Basis-Frameworks genutzt werden. Eine Beschreibung von Experimenten zur Evaluierung von Regressions-, Klassifikations- und Driftproblemen wird ebenfalls zusammengestellt. Für die Experimente werden ausgewählte Lernalgorithmen, Parameter, Datensätze und Evaluationsmethoden vorgestellt.

In dem **Kapitel 5** wird zunächst die prototypische Implementierung des Frameworks sowie dessen Architektur genauer beschrieben. Anschließend werden die Ergebnisse der durchgeführten Experimente präsentiert und diskutiert. Für die Evaluierung wurde das neue Verfahren mit anderen OML-Verfahren verglichen und bewertet. Am Ende des Kapitels werden die Limitationen dieser Arbeit aufgezeigt.

Abschließend fasst **Kapitel 6** die gewonnenen Erkenntnisse zusammen, zieht ein Fazit der vorliegenden Arbeit und gibt einen Ausblick auf zukünftige Forschungsrichtungen.

2. Grundlagen

Ausgehend von dem Thema „Adaptives maschinelles Lernen mit Mixture of Experts“ beginnt diese Arbeit mit den theoretischen Grundlagen, die zur Beantwortung der Forschungsfragen notwendig sind. Zunächst wird der Begriff der Adaptivität eingeführt. Im Anschluss werden Datenströme und deren Eigenschaften aus dem „Big Data“-Umfeld erklärt. Zusammen mit der Adaptivität ergeben sie die Grundlage für den Einsatz von Online maschinelles Lernen. Dabei findet auch die Einordnung und Abgrenzung zum offline maschinellem Lernen statt. Am Ende dieses Kapitels wird das Architekturprinzip der Expertenmischung beschrieben.

2.1. Adaptivität

Das Konzept der Adaptivität gewinnt in verschiedenen Forschungsbereichen zunehmend an Bedeutung. Immer mehr Informationssysteme sind adaptiv und werden in den verschiedensten Disziplinen eingesetzt. Dazu zählen Anwendungen wie KI, Robotik für dynamische Umgebungen, Netzwerke oder Betriebssysteme, um eine effektive Ressourcennutzung in der Informatik zu ermöglichen (Raibulet, 2008, S. 343). Der Einsatz von personalisierter Lehrsoftware für Schüler:innen und Lehrer:innen zur Steigerung der Effizienz in der Pädagogik (Miller, 2005) ist ebenfalls eine mögliche Anwendung. Adaptivität beschreibt in der Informatik die Fähigkeit von Systemen, sich dynamisch an veränderte Bedingungen, Benutzerpräferenzen oder neue Informationen anzupassen. Eine generelle Definition von Adaptivität ist schwer abzuleiten, da sie je nach Anwendung unterschiedlich verstanden wird. Es können aber drei generelle Kriterien benannt werden (Raibulet, 2008, S. 344ff):

- Reaktion auf Änderungen innerhalb oder außerhalb des Systems,
- Autonome und reflektierende Anpassungen durch das System selbst,
- Anpassungen zur Laufzeit

Bry und Henze (2005) unterscheiden zwischen *adaptiver* und *adaptierbarer* Software. Der Unterschied wird durch die Abbildung 2.1 deutlich. Der Anwender interagiert beispielsweise mit

seiner Umgebung, etwa einem Musik-Streaming-Dienst, in dem Musik gehört wird. Dadurch entstehen neue Informationen, welche die Empfehlungen für eine Playlist automatisch erzeugen und als Ergebnis zurückliefern. Die Playlist wird wieder vom Nutzer konsumiert, was neue Daten erzeugt. Bei Adaptivität sollen die veränderten Bedingungen selbstständig mit den neuen Daten ohne Nutzerintervention erlernt und geeignete Anpassungen, unter anderem die Empfehlungen innerhalb der Playlist, durchgeführt werden. Wenn die Sortierung der Titel als Einstellung durch den Nutzer initiiert wird, werden die Parameter des adaptiven Systems angepasst. Damit wird die Darstellung der Titel-Auflistung verändert. Das entspricht der Adaptierbarkeit eines Systems.

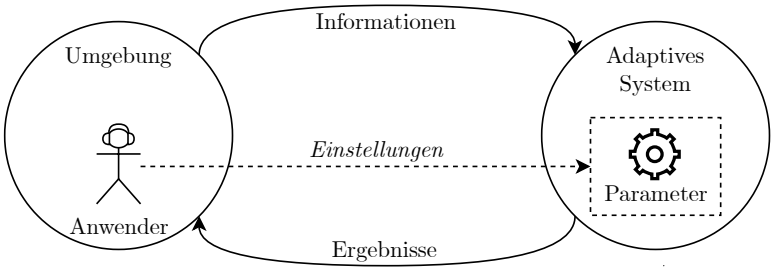


Abbildung 2.1.: Konzept der Adaptivität als Interaktion zwischen Umgebung und System.

Ein möglicher Einsatz von Adaptivität im Kontext des maschinellen Lernens wäre das adaptive Lernen. Durch den Einsatz von adaptiven Methoden soll sich ein maschinell trainiertes Modell flexibel und individuell an neue Trends in den zugrunde liegenden Daten einer neuen Situation anpassen (Bartz-Beielstein und Bartz, 2023, S. 11). Ziel ist, eine mögliche kontextabhängige Lösung für die gegebenen dynamischen Daten zu finden. Das zu lösende Problem wird daher als *Dynamisches Optimierungs-Problem* (DOP) definiert. Das adaptive Lernen wird durch kontinuierliches Lernen ermöglicht, dessen Trainingsdaten aus Datenströmen kommen.

2.2. Big Data

Der abstrakte Oberbegriff „Big Data“ wird mit dem Bestand und dem Wachstum großer gespeicherter Datenmengen assoziiert. Das heißt „Big Data“ bezeichnet Datenmengen, die so groß oder komplex sind, dass sie mit traditionellen Analyseverfahren wie SQL in relationalen Datenbanken nicht mehr bearbeitet werden können. Diese Daten erfordern den Einsatz neuer Techniken und Technologien zur Verarbeitung und Analyse (Warren, 2015, S. 2ff). Eine einheitliche Definition für diesen Begriff gibt es in der vorherrschenden Literatur nicht.

2.2.1. Eigenschaften nach dem V-Modell

Konkreter definiert sind jedoch die Eigenschaften nach dem „3V"-Modell von Big Data, die von Laney (2001) in einem Forschungsbericht definiert wurden und die Herausforderungen des Datenwachstums beschreiben:

- *Volume*: definiert die Anforderung, mit immensen Mengen von Daten, die in solchen Big Data Anwendungen üblicherweise generiert werden, umzugehen. Dies können Terabyte an Videomaterial sein, das von einer Kamera aufgenommen und dann gespeichert wurde.
- *Velocity*: bezeichnet die hohe Geschwindigkeit, in der diese Daten generiert, analysiert und verarbeitet werden können. Daten können in einem Datenstrom kontinuierlich generiert werden. Beispielsweise entsteht pro Flugstunde ein Terabyte an Sensordaten.
- *Variety*: Die letzte Eigenschaft steht für die Vielfalt der Datentypen und Quellen. Strukturierte Daten wie Tabellen oder unstrukturierte Daten wie Fotos aus unterschiedlichsten Quellen können gemeinsam abgespeichert werden. Die meisten Daten, die heute vorkommen, sind unstrukturiert, unterliegen also keinem bestimmten Schema. Die Herausforderung ist hier, mit geeigneten Algorithmen die Daten strukturiert einordnen zu können.

Daneben gibt es noch viele weitere „V"-Eigenschaften in der Literatur, die in dieser Arbeit nicht näher betrachtet werden, unter anderem: *Volatility* (Volatilität), *Variability* (Veränderlichkeit), *Veracity* (Korrektheit), *Viability* (Viabilität), *Visualization* (Visualisierung), *Virality* (Viralität), *Viscosity* (Viskosität) oder *Validity* (Gültigkeit). Diese Eigenschaften bringen ebenfalls Probleme und Herausforderungen für Big Data Systeme mit. Beispielsweise ist bei der *Volatility* die Lebensdauer der Daten gemeint. Bei medizinischen Daten sollten die Daten immer vorhanden sein, bei Echtzeitdaten von sozialen Medien ist dies nicht notwendig. Für die *Variability* sind mögliche Inkonsistenzen in den variierenden Strukturen der Daten ein Problem (Khan et al., 2018; Fritz, 2022; Rahul, Banyal und Arora, 2023).

2.2.2. Datenströme

Als *Datenstrom* wird in der Informatik ein kontinuierlicher Fluss von Daten bezeichnet. In der Abbildung 2.2 wird schematisch der Aufbau eines Datenstroms gezeigt. Ein Datenstrom ist üblicherweise eine *First-In-First-Out* (FIFO)-Queue. Ein Datenstrom-Datensatz $\mathcal{D}$ besteht aus mehreren geordneten Dateninstanzen d. Die Dateninstanzen innerhalb des Stroms sind *Streamingdaten*. Zum Zeitpunkt t wird der Datensatz d_t verarbeitet. Jeder Datenpunkt d_t kann

dabei ein einzelner Wert oder ein Vektor sein. Die Ordnung ist implizit durch die zeitliche Reihenfolge der eintreffenden Daten des Datenstroms gegeben. Es gilt $t \geq t-1$, daraus folgt, dass eine Dateninstanz d_{t-1} älter ist als d_t (Bartz-Beielstein und Bartz, 2023, S. 2).

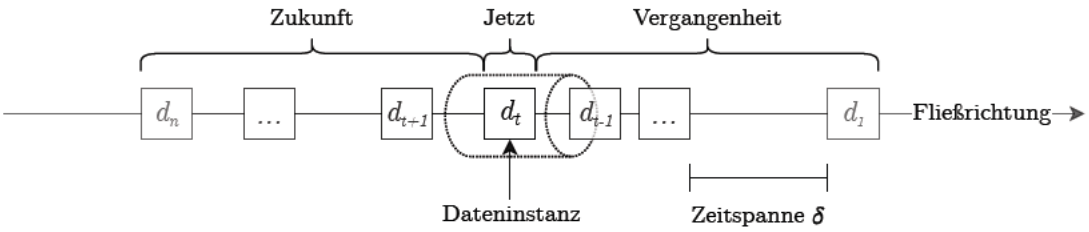


Abbildung 2.2.: FIFO-Queue mit Streamingdaten über einen zeitlichen Verlauf.

Die Dateninstanzen in dem Datenstrom sind lose strukturiert, nur einmal verfügbar und beinhalten teilweise unvorhersehbare *Änderungen*, welche auch als „Drift“ bezeichnet werden (Bartz-Beielstein und Bartz, 2023, S. 2). Die Kardinalität des Datensatzes $|\mathcal{D}|$ beschreibt seine Größe. Die wichtigste Unterscheidung in der Kardinalität liegt darin, ob ein Datensatz endlich oder unendlich ist. Bei einem endlichen Datensatz gibt es eine N-endliche Anzahl von Dateninstanzen und wird als Menge wie folgt definiert: $\{d_t\}_{t=1}^{N}$. Innerhalb eines Datenstroms wird von einem unbegrenzten Datensatz gesprochen, wenn er unendlich viele Dateninstanzen beinhaltet. Als Menge beschrieben, heißt das: $\{d_t\}_{t=1}^{\infty}$ (Akidau, Chernyak und Lax, 2018).

Streamingdaten sind eine Teilmenge von Big Data, denen die „V“-Eigenschaften zugeordnet werden können. Sie liegen üblicherweise in großen Mengen (*Volumen*) vor, die in hoher Geschwindigkeit *Velocity* erzeugt werden. Dabei können die Streamingdaten in sehr unterschiedlichen Formaten vorliegen (*Variety*). Diese Eigenschaften werden als „vertikale Vielfalt“ bezeichnet. Zusätzlich sind Daten innerhalb eines Datenstroms oft strukturlos und können im Laufe der Zeit variieren (*Variability*), woraus graduell oder abrupt ein Drift auftreten kann. Diese Eigenschaft wird als „horizontale Vielfalt“ bezeichnet. Weiterhin sind Streamingdaten nur einmal verfügbar (*Volatilität*) (Bartz-Beielstein und Bartz, 2023).

Die Zeit zwischen zwei beliebigen Dateninstanzen wird als Zeitspanne δ bezeichnet und berechnet sich über die zeitliche Differenz zweier Zeitpunkte T: $T_t - T_{t-1}$. Bei der Division der Kardinalität des Datensatzes, die innerhalb dieses Zeitraumes geflossen sind, mit der Zeitspanne resultiert die durchschnittliche Datenrate v:

$$v = \frac{|\mathcal{D}_{\text{sub}}|}{\delta} = \frac{|\mathcal{D}_t| - |\mathcal{D}_{t-1}|}{T_t - T_{t-1}}, \mathcal{D}_t, \mathcal{D}_{t-1} \subseteq \mathcal{D} \wedge t \geq t-1 \wedge \delta \in \mathbb{R}^+ \tag{2.1}$$

Die durchschnittliche Datenrate entspricht der Menge von Datensätzen pro Zeitspanne, also der Fließgeschwindigkeit. Sind zum Zeitpunkt 12:00 Uhr bereits 100 Dateninstanzen durch den

Strom gelaufen, während um 10:00 Uhr nur 50 Instanzen vorhanden waren, ergibt sich mit der Formel 2.1 eine durchschnittliche Geschwindigkeit von 25 Dateninstanzen pro Stunde zwischen diesen Zeitpunkten. Eine fortlaufende Datenrate $\{v_i\}_{i=1}^{N}$ lässt sich mit der Betrachtung der einzelnen Datenrate zwischen den einzelnen Dateninstanzen ableiten. Diese individuelle Datenrate kann gleichbleibend sein, dann gilt $v_1 = \cdots = v_N$. Wenn die Dateninstanzen, wie in Abbildung 2.2, unterschiedliche Abstände haben, variiert die Datenrate.

Streamingdaten setzen voraus, spezielle Analysemethoden anzuwenden, die in Echtzeit oder nahezu Echtzeit funktionieren. Das Abspeichern der Daten ist aufgrund der Menge meistens nicht möglich oder nicht sinnvoll. Mithilfe von intelligenten Algorithmen können etwa Erkenntnisse aus Video-Streamingdaten abgeleitet werden, um den Zustand einer Straße für eine mögliche Wartung zu ermitteln.

2.3. Online Maschinelles Lernen

Online maschinelles Lernen ist eine Methode, die es ermöglicht, Modelle fortlaufend mit neuen Daten aus einem Datenstrom zu aktualisieren, anstatt auf statischen Datensätzen zu basieren. *Statische Daten* sind Daten, die zu einem bestimmten Zeitpunkt erfasst wurden und nicht mehr verändert werden. Sie können beliebig oft abgerufen werden und liegen häufig in Tabellenform vor. (Bartz-Beielstein und Bartz, 2023, S. 3). Bei der folgenden Betrachtung wird der Fokus auf Streamingdaten gelegt. Durch eine *Serialisierung*, also die datenstromartige Aneinanderreihung der Daten, können statische Daten in einen endlichen Datenstrom umgeformt werden. In den folgenden Erläuterungen wird der Fokus auf Streamingdaten mit zeitlicher Ordnung gelegt, daher wird der Zeit-Index t verwendet.

2.3.1. Maschinelle Lernverfahren

Maschinelles Lernen bietet eine Möglichkeit, Muster in großen und komplexen Datenmengen zu erkennen. Dabei verbessern Algorithmen ihre Leistung durch Training, indem sie große Datenmengen analysieren und eigenständige Optimierungen vornehmen. Zwei wesentliche Lerntypen sind das überwachte und das unüberwachte Lernen, die aus der Abbildung 2.3 entnommen werden können (Fritz, 2022, S. 142f).

Beim *überwachten* Lernen bestehen die Datenpunkte d_t aus Datenpaaren $d_t = (x_t, y_t(x_t))$. Das x_t steht für die Eingangsdaten. Das kann ein einzelner Wert oder ein Vektor aus n-*Merkmalen* sein: $x_t = (x_{t,1}, x_{t,2}, \ldots, x_{t,n})$. Merkmale werden auch als *Attribute*, *Features* oder *unabhängige Variablen* bezeichnet. Es gilt damit $x_t \in \mathbb{R}^m$. Bei dem $y_t(x_t) \in \mathbb{R}$ handelt es sich um die zugehörige, korrekte Ausgabe und wird als *Label*, *Zielvariable* oder auch *abhängige*

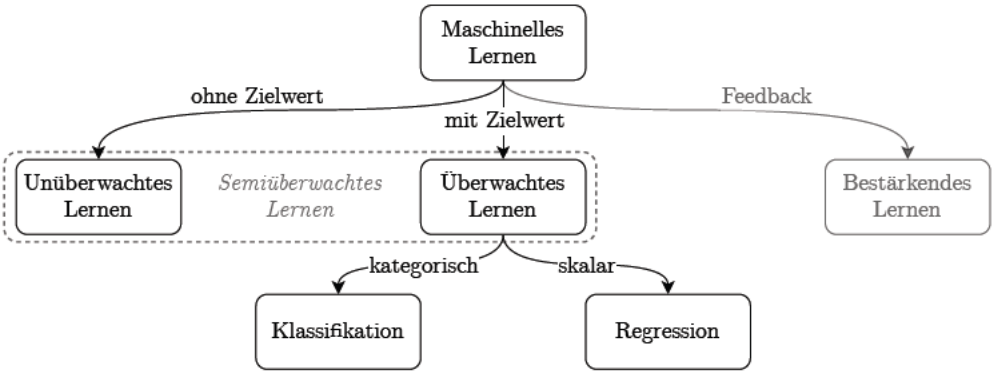


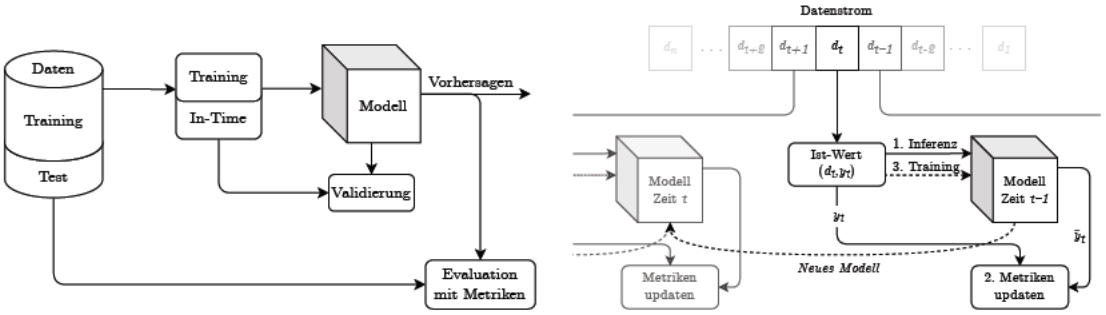
Abbildung 2.3.: Übersicht und Einordnung von ML für verschiedene Problemstellungen.

Variable betitelt. Der endliche Datensatz für überwachtes Lernen kann als $\mathcal{D}_\mathrm{S} = \{d_t\}_{t=1}^N = \{(x_t, y_t(x_t)\}_{t=1}^N$ beschrieben werden. Das überwachte Lernen wird für *Klassifikations-* und *Regressionsprobleme* eingesetzt, bei denen das Ziel darin besteht, präzise Vorhersagen für die bekannten Zielvariablen zu treffen. Bei der Regressionsanalyse wird versucht, skalare Werte vorherzusagen, wohingegen bei der Klassifikation eine kategorische Klasse den Eingangsdaten zugeordnet wird. Der Hauptvorteil des überwachten Lernens liegt darin, dass die Modelle konkret auf bekannte Label optimiert werden. Ein Beispielverfahren für Regression wäre die lineare Regression und für die Klassifikation die logistische Regression. Die Anwendung findet sich in der Bildklassifikation oder in der Vorhersage von Verkaufszahlen (Fritz, 2022, S. 146f). Existieren mehrere Label $y \in \mathbb{R}^n$ mit $n > 1$ pro Datensatz, handelt es sich um eine *Multi-Ziel-Regression* beziehungsweise *Multi-Ziel-Klassifikation.* Dabei werden mehrere Aufgaben gleichzeitig trainiert. Das findet besonders im *Multi-Task-Lernen* (MTL) Anwendung.

Im Gegensatz dazu arbeitet das ***unüberwachte*** Lernen ohne gelabelte Daten und konzentriert sich auf das Erkennen verborgener Strukturen innerhalb der Daten. Der Datensatz besteht damit aus keinem Datenpaar, sondern nur den Features: $d_t = x_t \implies \mathcal{D}_\mathrm{U} = \{x_t\}_{t=1}^N$. Das Ziel ist es, Muster und Gruppen zu identifizieren, ohne dass eine Zielvariable vorgegeben ist. Unüberwachtes Lernen wird oft für Clusteranalysen eingesetzt und findet Anwendung in Bereichen wie der Kundensegmentierung oder der Erkennung von Anomalien. Die Herausforderung besteht hierbei darin, sinnvolle Strukturen aus den Daten zu extrahieren und die Ergebnisse entsprechend zu interpretieren. Ein beispielhaftes Verfahren ist der k-Means-Algorithmus (Fritz, 2022, S. 148f). Die kombinierte Form von überwachten und unüberwachten Lernen, dem *Semiüberwachten* Lernen und auch das Lernen mit Feedback, dem *bestärkendem* Lernen werden in dieser Arbeit nicht näher betrachtet.

2.3.2. Inkrementelles- und Batch-Lernen

Bei dem Training von ML gibt es jeweils eine Strategie für statische Daten und eine für Streamingdaten. Bei kontinuierlichen Datenströmen wird das *inkrementelle Lernen* mit OML angewendet, das auch *Online-Lernen* oder *Stream-Lernen* genannt wird. Der Begriff „Online-Lernen“ wird häufig mit einer anderen Bedeutung im Kontext der Bildungsforschung verwendet, weswegen in dieser Arbeit der Begriff Online *maschinelles* Lernen verwendet wird. Wenn ein Datensatz vollständig und statisch vorliegt, wird das *stapelweise Lernen* mit *Batch maschinelles Lernen* (BML) verwendet. Analog zum Online-Lernen kann diese Strategie *Offline-Lernen* genannt werden (Bartz-Beielstein und Bartz, 2023, S. 3ff). Durch die Abbildung 2.4 wird der Unterschied beider Lernstrategien deutlicher.



(a) Batch maschinelles Lernen (BML)

(b) Online maschinelles Lernen (OML)

Abbildung 2.4.: Schematischer Aufbau beider ML-Lernstrategien.

Bei dem Batch-Training aus Abbildung 2.4a wird der gesamte statische Datensatz in einen Trainings- und Testdatensatz geteilt. Mit den Trainingsdaten werden die Algorithmen trainiert, damit das Modell bestmögliche Vorhersagen ausgibt. Während des Trainings wird ein Teil der Daten verwendet, um eine Validierung während des Trainings zu berechnen, die die Modellgüte beurteilt. Nach Fertigstellung des Trainings und der sukzessiven Validierung können andere Daten verwendet werden, um mit einer Inferenz eine Vorhersage zu berechnen. Für die Evaluation des Modells werden, im Falle des überwachten Lernens, Metriken mithilfe der ungesehenen Testdaten berechnet. Dies ist eine quantitative Einschätzung über die Güte des trainierten Modells (Fritz, 2022, S. 146ff). Bei Eintreffen von neuen Datenpunkten ist ein erneutes Training mit dem gesamten Datensatz notwendig. Werden Teilmengen des gesamten Datensatzes, sogenannte *Mini-Batches*, zum wiederholten Erstellen eines Modells verwendet, wird das *Mini-Batch maschinelles Lernen* (Mini-BML) genannt. Die einzelnen Mini-Batches werden in der Regel nur einmal verwendet. Dadurch werden weniger Daten in den Hauptspeicher geladen (Bartz-Beielstein und Bartz, 2023, S. 11).

Durch das Training mit BML gibt es Probleme. Wenn die Größe des gesamten Datensatzes die Größe der verfügbaren Menge an Arbeitsspeicher überschreitet, können die Algorithmen nicht mehr lernen. Eine mögliche Lösung gibt es durch Optimierung von Datentypen oder die Dimensionsreduktion auf Kosten von Modellgüte. Ein weiteres Problem, das beim BML auftreten kann, sind dynamische Strukturveränderungen. Merkmale und Zielgrößen können sich über die Zeit verändern, was zu verschiedenen „Drifts“ führt, die Leistungsminimierungen von Modellen verursachen. BML-Modelle können nicht aus neuen Daten mit unbekannten Attributen lernen. Das Modell muss mit einem neuen statischen Datensatz lernen (Bartz-Beielstein und Bartz, 2023, S. 3ff).

Eine mögliche Lösung für diese Probleme ist das inkrementelle Lernen mit einem Datenstrom, siehe dazu Abbildung 2.4b. Es wird elementweise ein Datenpunkt aus den Streamingdaten im Training verwendet. Mit diesem Datenpunkt wird eine Inferenz mit dem bisherigen Modell berechnet. Für das überwachte Lernen wird der tatsächliche Wert zusammen mit der Vorhersage genutzt, um die Metriken des zu diesem Zeitpunkt existierenden Modells direkt zu aktualisieren. Nach der Aktualisierung wird durch eine marginale Anpassung eine neue Modellversion erzeugt. Dieser Trainingsablauf wiederholt sich inkrementell für jede neu eintreffende Instanz und kann damit adaptiv auf dynamische Veränderungen reagieren. Die einzelnen Instanzen können damit nur einmal verwendet werden (Bartz-Beielstein und Bartz, 2023, S. 10f). Für das OML werden Lernalgorithmen benötigt, die einzelne Instanzen verarbeiten können, ohne diese erneut abrufen zu können. So funktioniert der klassische Entscheidungsbaum *Classificiation And Regression Tree* (CART) nicht. Seine OML-Alternative ist der *Hoeffding-Baum*-Algorithmus (Bartz-Beielstein und Bartz, 2023, S. 16ff).

2.3.3. Evaluationsmethoden

Nachdem die Vorteile von OML für adaptives Lernen im Vergleich zu BML herausgestellt wurden, liegt der Fokus im Folgenden auf OML sowie seiner speziellen Evaluationsstrategie.

Modell Das Ziel beider Lernstrategien ist das Training eines maschinellen Modells unter Verwendung eines Algorithmus, der aus den Eingangsdaten lernt. Das Modell $f_t : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^n$ mit $x_t \in \mathbb{R}^m \wedge \hat{y}_t \in \mathbb{R}$ wird generell als Funktion mit einer Konfiguration ρ als Vektor parametrisiert und gemäß Formel 2.2 definiert (Raschka, Mirjalili und Lorenzen, 2018, S. 26ff.):

$$f_t(x_t; \rho) = \hat{y}_t \tag{2.2}$$

Der Wert $\hat{y}_t$ steht für die vom Modell f_t berechnete Vorhersage für den Datenpunkt x_t zum Zeitpunkt t mit der Modellkonfiguration ρ. Das generelle Ziel beim überwachten Lernen ist,

eine optimale Hyper-Parametrisierung ρ^* aus der Menge der möglichen Kombinationen von Hyperparameter zu erlangen, sodass das Modell die bestmöglichen Evaluationsmetriken erzielt. Das Durchlaufen verschiedener Kombinationen als sogenanntes *Grid-Search* von Hyperparametern und Evaluationsmetriken ermöglicht im überwachten Lernen eine Suche nach den bestmöglichen Parametern. Dieser Vorgang wird *Hyperparameteroptimierung* (HPO) genannt (Raschka, Mirjalili und Lorenzen, 2018, S. 216ff.). Beim unüberwachten Lernen gibt es keine Vorgabe für richtig oder falsch. Aber es gibt Parameter, die optimiert werden können, wie die Anzahl der Cluster. Mithilfe der *Elbow-Methode* nach Formel 2.3 lässt sich die optimale Anzahl von Clustern darstellen. Sie untersucht die Summe der quadratischen Abweichung innerhalb der Cluster für verschiedene Clusteranzahlen K (Raschka, Mirjalili und Lorenzen, 2018, S. 362ff).

$$\textit{Inertia} = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2 \text{ mit } K \in \mathbb{N} \tag{2.3}$$

Wobei x für den einzelnen Datenpunkt und μ_i für den Schwerpunkt des Clusters C_i steht. $\|x - \mu_i\|^2$ ist die euklidische Distanz des Punktes zum Clusterschwerpunkt. Der Punkt, an dem die *Inertia* signifikant weniger stark abnimmt, ist der „Knick“ und wird als optimale Clusteranzahl betrachtet. Dies ist eine effektive Optimierungsmethode zur Clusterbildung.

Evaluationsmetriken Je nach Lernverfahren und Problemstellung gibt es unterschiedliche Evaluationsmetriken, die angewendet werden können, um die Güte des Modells zu bewerten. In der Tabelle 2.1 wird eine Auswahl von populären Metriken aufgelistet. Bei der Regression wird die Differenz zwischen dem tatsächlichen Wert y und der Vorhersage $\hat{y}$ der Zielvariable (Label) berechnet, um einen Fehler, auch *Residuum* genannt, zu ermitteln. Diese Residuen werden verwendet, um verschiedene Metriken zu ermitteln, die eine Aussage über die Genauigkeit des Modells geben. Die Metriken R2 und *Mean Absolute Error* (MAE) sind leicht interpretierbar, aber spiegeln nicht immer ideal die Modellleistung wider. MSE und *Root Mean Squared Error* (RMSE) bestrafen größere Fehler mehr, was vorteilhaft ist, wenn große Fehler problematisch sind. Ein Nachteil ist jedoch ihre geringere Interpretierbarkeit sowie ihre höhere Empfindlichkeit gegenüber Ausreißern. Durch die Substitution von $y = \log(1 + y)$ bei MSE und RMSE wird der Logarithmus eingesetzt, der den Effekt von Ausreißern mildert. Daraus resultieren neue Metriken wie der RMSLE oder MSLE. Dieser Wert lässt sich ebenfalls schwer interpretieren und ist nicht für negative oder Nullwerte geeignet, da der Logarithmus für diese Werte nicht definiert ist. Je nach Zielgruppe ist die richtige Kombination von Metriken wichtig (Raschka, Mirjalili und Lorenzen, 2018, S. 335ff.).

Abkürzung	Name	Formel	Ziel
MSE	Mittlerer quadratischer Fehler	$\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2$	minimieren
RMSE	Standardfehler der Regression	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$	minimieren
MAE	Mittlerer absoluter Fehler	$\frac{1}{n}\sum_{t=1}^{n}\lvert y_t - \hat{y}_t\rvert$	minimieren
R^2 (auch „R2“)	Bestimmtheitsmaß	$1 - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$	möglichst 1

Tabelle 2.1.: Auswahl gängiger Evaluationsmetriken für Regressionen unter Verwendung des tatsächlichen Wertes y und des errechneten Wertes $\hat{y}$.

Bei den Klassifikationen wird eine *Konfusionsmatrix* als Kontingenztabelle verwendet, um die Häufigkeit der vorhersagten und tatsächlichen Klassen gegenüberzustellen. Für die binäre Klassifizierung mit zwei Klassen sieht die Matrix aus wie in Abbildung 2.5a. Dabei können Vorhersagen „True“ oder „False“ sein. Bei zwei Klassen „Positive“ und „Negative“ ergeben sich damit vier mögliche Zustände: *True Positive* (TP), *False Positive* (FP), *True Negative* (TN), *False Negative* (FN). Der Zustand TP steht für eine korrekte Vorhersage der, hier, positiven Klasse, wohingegen FP für eine fälschlicherweise positiv zugeordnete Klasse steht. Aus der Tabelle 2.5b werden die Metriken zur Bewertung eines Klassifizierungsmodells aufgelistet.

$\Sigma = \lvert P\rvert + \lvert N\rvert$		Vorhersagte Klasse	
		Positive	Negative
Tatsächliche Klasse	Positive (P)	TP	FN
	Negative (N)	FP	TN

(a) Konfusionsmatrix

Name	Formel
Accuracy	$\frac{\text{TP + TN}}{\text{TP + TN + FP +FN}}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
Specificity	$\frac{TN}{TN+FP}$
F1-Score	$2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$
ROC-AUC	$\int_0^1 \frac{TP}{TP+FN}\, d(1 - \frac{TN}{TN+FP})$

(b) Auswahl an Klassifikationsmetriken

Abbildung 2.5.: Auswahl gängiger Metriken für Klassifizierung mit zusammenhängender Konfusionsmatrix.

Die *Accuracy* lässt sich einfach interpretieren und ist geeignet, wenn Klassen gleichmäßig verteilt sind. *Precision* (Präzision) ist geeignet, um die Leistung für FP-Fälle und *Recall* (Sensitivität) um TP-Fälle aufzuzeigen. Eine Metrik alleine wird zur Beurteilung des Modells nicht empfohlen. Besser ist der F1-Score, der das harmonische Mittel aus Precision und Recall bildet.

Der F1-Score ist aufgrund der Gewichtung schwer interpretierbar. Die *Receiver Operating Characteristic - Area under Curve* (ROC-AUC) ist der Flächeninhalt unter der ROC-Kurve, die die True-Positive-Rate (Recall) und False-Positive-Rate (1 - Specificity) darstellt. Der Wert ist gut geeignet für Modelle mit unausgewogenen Klassenverhältnissen, ist aber schwer zu interpretieren und liefert nur eine relative Qualität. Die bisherigen Bewertungskriterien bezogen sich auf binäre Klassifizierungen mit nur zwei Klassen. Mithilfe einer One-vs.-All Mittelwertbildung lassen sich die Metriken auch auf Problemstellungen mit mehreren Klassen erweitern (Raschka, Mirjalili und Lorenzen, 2018, S. 220ff.).

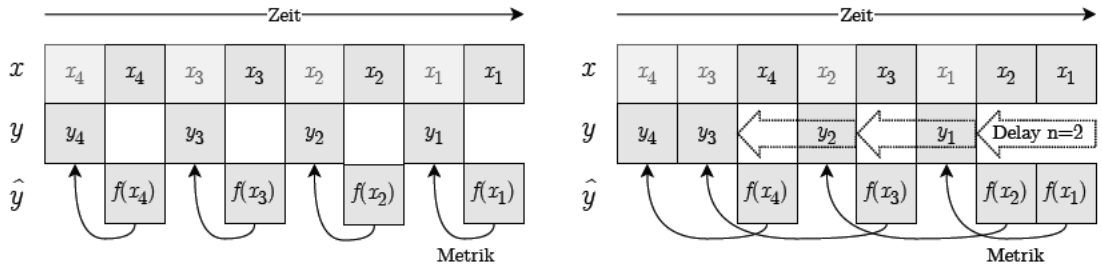
Das unüberwachte Lernen hat kein Label in ihren Datensätzen. Daher werden andere Metriken zur Beurteilung der Clustergüte eingesetzt. Eine gängige Metrik ist der *Silhouette-Koeffizient* s_C. Dieser bewertet die Qualität der Clusterbildung für verschiedene Anzahlen von Clustern. Das Bewertungskriterium lässt sich leicht berechnen und wird in Formel 2.4 definiert (Raschka, Mirjalili und Lorenzen, 2018, S. 363).

$$s_C = \frac{1}{n_C} \sum_{o \in C} s(o) = \frac{1}{n_C} \sum_{o \in C} \frac{\text{dist}(B, o) - \text{dist}(A, o)}{\max(\text{dist}(B, o), \text{dist}(A, o))} \qquad s_C, s(o) \in [-1; 1] \tag{2.4}$$

Dabei wird $\text{dist}(A, o)$ als mittlerer Abstand zu allen anderen Objekten des Clusters A und $\text{dist}(B, o)$ als mittlerer Abstand zu den Objekten des nächstgelegenen Clusters verwendet. Die Differenz wird durch den größeren Abstand der beiden Abstände normiert, sodass der Wert der Silhouette s zwischen -1 und 1 liegt. Der Koeffizient entspricht dem arithmetischen Mittel aller n_C Silhouetten s des Clusters C. Der Wertebereich ist daher identisch zur Silhouette. Das Kriterium ist einfach zu interpretieren, da der Wert 1 einem gut getrennten Cluster entspricht. Die Berechnung kann bei größeren Daten aufwendig sein, da paarweise Distanzen ermittelt werden müssen (Raschka, Mirjalili und Lorenzen, 2018, S. 362ff).

Evaluationsmethodik Alle vorgestellten Metriken aus dem überwachten und unüberwachten Lernen können im BML, aber auch im OML angewendet werden. Nur die Berechnung der Metriken unterscheidet sich. Beim BML werden die Metriken auf dem gesamten Datensatz berechnet. Beim OML wird die *progressive Validierung* verwendet, die beispielhaft in Abbildung 2.6a dargestellt ist und schematisch in Abbildung 2.4b erklärt wird. Zunächst wird für einen ankommenden Datenpunkt die Vorhersage mit dem Modell $\hat{y} = f(x)$ berechnet. Im nächsten Zeitschritt wird die Grundwahrheit y herangezogen. Mit dem tatsächlichen Wert y und der Vorhersage $\hat{y}$ wird dann die Metrik und das Modell aktualisiert. In Abbildung 2.6b wird dieser Vorgang der *verzögerten progressiven Validierung* dargestellt. Das Eintreffen der tatsächlichen

Werte y wird um einen Delay d verzögert. Dadurch wird zwar die Prognose berechnet, aber erst zu einem späteren Zeitpunkt geprüft, wie gut die Vorhersagen waren (Evaluation). Damit wird das Modell nicht sofort aktualisiert, was für einen produktiven Betrieb wichtig ist (Bartz-Beielstein und Bartz, 2023, S. 46f).



(a) Evaluation mit y und $\hat{y}$ ohne Delay

(b) Evaluation mit y und $\hat{y}$ mit Delay = 2

Abbildung 2.6.: Konzept der Progressiven Validierung mit einem Datensatz der Größe 4.

Die Aktualisierung der Metriken innerhalb der Progressiven Validierung berechnet sich durch einen kumulierten Durchschnitt, dem *Weighted Cumulative Moving Average* (WCMA), der in Formel 2.5 beschrieben wird:

$$\begin{aligned} \mathrm{W}_t(w_t) &= \mathrm{W}_{t-1} + w_t \text{ mit } w_t \in [0;1] \\ s_{t+1}(x_t; w_t) &= \begin{cases} 0 & \text{falls } t = 1 \\ s_t + \frac{w_t}{\mathrm{W}_t(w_t)}(x_t - s_t) & \text{sonst} \end{cases} \end{aligned} \tag{2.5}$$

Dabei steht x_t für die Bewertung des neuen Datenpunktes zum Zeitpunkt t. Mit W_t werden die Gewichte w_t über die Zeit kumuliert. Die Differenz zwischen dem aktuellen und dem vorherigen Wert wird mit dem Gewichtungsfaktor $\frac{w_t}{W_t(w_t)}$ multipliziert und zum alten Wert kumuliert. Wenn alle $w_t = 1$ sind, entspricht die Formel 2.5 dem klassischen *Cumulative Moving Average* (CMA). Alternative Verfahren für OML sind unter anderem der *Exponentially Weighted Moving Average* (EWMA) mit einer exponentiellen Glättung, ohne Kumulierung: $s^e_{t+1}(x_t) = \alpha \cdot x_t + (1 - \alpha) \cdot s^e_t$. Je höher der Wert von $\alpha \in [0;1]$, desto stärker wird der neueste Datenpunkt gewichtet. Ein Vorteil beider Ansätze ist, dass lediglich der aktuelle und vorherige Datenpunkt benötigt wird, was Speicherplatz spart (Finch, 2009).

2.3.4. Drifterkennung und -behandlung

Mit der Zeit können ML-Modelle $f : X \to Y$ mit $x_t \in X$ und $y_t \in Y$ unzuverlässig werden, da die von ihnen gelernten Beziehungen sich verändern können („Drift“). Eine Übersicht der

Driftarten wird in Tabelle 2.2 aufgezeigt. Die Beziehung zwischen X und Y entspricht einem *Konzept*: $P_t(X \cap Y) = P_t(Y|X) \cdot P_t(X) = P_t(X|Y) \cdot P_t(Y)$. Die A-posteriori Wahrscheinlichkeit $P_t(Y|X)$ entspricht der Verteilung der Zielwerte Y für die gegebenen Merkmale X zum Zeitpunkt t. Das $P(X|Y)$ steht für das Auftreten der Merkmale X unter der Bedingung des Zielwertes Y. $P(X)$ steht für die Evidenz der Merkmale und $P(Y)$ für die A-priori Wahrscheinlichkeit der Zielwerte. Ein *Feature-Drift* beschreibt die Veränderung der unabhängigen Variable X bei gleichbleibendem Konzept. Für ein *Label-Drift* verhält es sich gleich, nur dass es die abhängige Variable Y betrifft. Verändert sich die A-posteriori Wahrscheinlichkeit, kann das Modell keine zuverlässigen Vorhersagen zu den Beziehungen machen, was *Konzeptdrift* genannt wird. Ein Drift kann zwischen zwei Zeitpunkten t und $t + 1$ oder einer Zeitperiode $t_{[t_1,t_2]}$ und $t_{[t_{2+1},t_3]}$ auftreten (Moreno-Torres et al., 2012).

Dirftart	Bedingungen
Feature-Drift	$P_t(X) \neq P_{t+1}(X) \wedge P_t(Y\|X) = P_{t+1}(Y\|X)$
Label-Drift	$P_t(Y) \neq P_{t+1}(Y) \wedge P_t(Y\|X) = P_{t+1}(Y\|X)$
Konzeptdrift	$P_t(Y\|X) \neq P_{t+1}(Y\|X) \implies P_t(Y \cap X) \neq P_{t+1}(Y \cap X)$

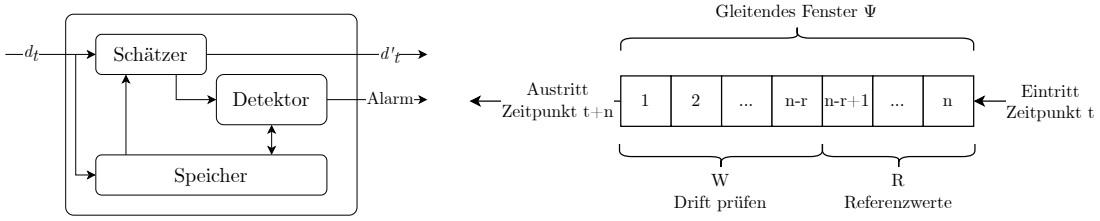
Tabelle 2.2.: Übersicht der Driftarten und deren Bedingungen.

Zur Veranschaulichung der verschiedenen Driftarten kann die Wohnflächenberechnung genutzt werden. Feature-Drift tritt auf, wenn die Berechnung der Wohnfläche verändert wird, die Einfluss auf die Verteilung des gleichnamigen Merkmals hat. Steigt der durchschnittliche Verkaufspreis, der vorhergesagt werden soll, und zwar unabhängig von den Merkmalen, handelt es sich um Label-Drift. Exemplarisch dafür wäre eine Gesetzesänderung, die den Einbau von Wärmepumpen in Neubauten vorschreibt, was Einfluss auf das bestehende trainierte ML-Modell zur Vorhersage von Wärmepumpenanfragen hätte und damit zu Konzeptdrift führt.

Die Erkennung von Drift kann in zwei Kategorien eingeteilt werden: *explizite* und *implizite* Verfahren. Die explizite Drifterkennung ermittelt mithilfe des Labels Modellmetriken, die durch das Verfahren im Laufe der Zeit überwacht werden. Dieses Verfahren kann gut im überwachten Lernen eingesetzt werden. Bei der impliziten Drifterkennung verlassen sich die Algorithmen auf die Eigenschaften der Merkmalswerte, da die Daten nicht gelabelt sind. Dadurch erzeugen sie häufiger Fehlalarme, sind aber nützlich für Anwendungen im unüberwachten Lernen (Bartz-Beielstein und Bartz, 2023, S. 27ff).

Drifterkennung kann in BML und OML durch statistische Tests umgesetzt werden. Das *Adaptive Windowing* (ADWIN) wird für explizite Drifterkennung angewendet und verwaltet ein gleitendes Fenster Ψ konfigurierbarer Länge. Innerhalb des Fensters werden kürzlich beobachtete Datenpunkte gespeichert. Das Verfahren teilt das Fenster in zwei Teilfenster

Ψ_0 und Ψ_1 und vergleicht ihre Mittelwerte. Ein statistischer Hypothesentest prüft, ob die Differenz zwischen den Mittelwerten signifikant ist. Falls ja, wird davon ausgegangen, dass ein Konzeptdrift aufgetreten ist, und das ältere Fensterteil wird entfernt. Der Parameter von ADWIN dafür ist die Konfidenzgrenze δ, die angibt, wie viel Vertrauen in die Ausgabe des Algorithmus gegeben werden soll. Das Verfahren kann gut bei abrupten Änderungen verwendet werden (Bifet, 2017).



(a) Generelle *Drift-Detektor* Architektur (b) Speicherstrategie des KSWIN-Verfahren

Abbildung 2.7.: Architektur von Detektoren zur Erkennung von Veränderungen mit einer beispielhaften Speicherverwaltung von Kolmogorov-Smirnov Windowing.

Die *Kolmogorov-Smirnov Windowing* (KSWIN) Methode basiert auf dem nicht-parametrischen statistischen *Kolmogorov-Smirnov* (KS) Test, der die Ähnlichkeit zweier Verteilungen vergleicht, indem er den maximalen Unterschied zwischen ihren kumulativen Verteilungsfunktionen misst. Das KSWIN nutzt ein gleitendes Fenster Ψ der Größe n wie in Abbildung 2.7b dargestellt. Die letzten $r \in \Psi$ Stichproben des gleitenden Fensters repräsentieren die kumulative Verteilungsfunktion der Referenz-Stichproben aus dem älteren Fensterfragment F_R. Das neuere Fragment F_W beinhaltet die kumulative Verteilungsfunktion der Stichproben bis $n - r$, die auf Drift getestet werden sollen. Ein Konzeptdrift ist in den Daten gegeben, wenn folgende Bedingungen aus Formel 2.6 gegeben sind (Raab, Heusinger und Schleif, 2020):

$$dist(R, W) > \sqrt{\frac{-\ln(\alpha)}{r}} \text{ mit } dist(R, W) = \sup_x |F_W(x) - F_R(x)| \tag{2.6}$$

$dist(R, W)$ steht für den maximalen Abstand beider Verteilungsfunktionen $F_W(x)$ und $F_R(x)$. Der Parameter α steht für den Schwellenwert des statistischen KS-Tests. Je größer das Fenster ist, desto besser eignet es sich für inkrementelle Drifts mit schwankenden Änderungen und vice versa (Raab, Heusinger und Schleif, 2020).

Die Architektur von Drifterkennungsmethoden wie ADWIN und KSWIN ist in 2.7a dargestellt. Er besteht aus einem *Speicher*, wo Stichproben und relevante statistische Daten zwischengespeichert werden. Der *Schätzer* berechnet die benötigten Statistiken wie Mittelwert oder Verteilungen. Der *Drift-Detektor* prüft die Bedingung, ob eine Veränderung vorliegt oder

nicht. Dafür verwendet er die Ausgaben des Schätzers und kann zusätzlich auf den Speicher zurückgreifen (Bifet, 2017). Erkennt der Algorithmus eine signifikante Änderung in der Verteilung der Daten aus dem Datenstrom oder in einer Modellmetrik, dann lösen er einen Alarm aus, um eine Modifikation der Algorithmen zu initiieren. Bei einer abrupten Änderung muss ein neues Modell trainiert werden. Bei schrittweisen Änderungen können die Parameter θ auch angepasst werden (Bartz-Beielstein und Bartz, 2023, S. 25f).

Die größte Herausforderung bei OML-basierten Verfahren ist das mögliche *katastrophale Vergessen*. Es entspricht dem Phänomen, dass ein Modell beim inkrementellen Lernen neuer Daten seine Fähigkeit verliert, vorher gelernte Informationen korrekt zu nutzen. Das OML bietet Möglichkeiten, um dem Problem entgegenzuwirken: Frühere Daten werden gespeichert und später in das Training einbezogen (*Replay Memory*) oder es werden Regulierungen eingeführt, um drastische Änderungen (Drift) durch Gewichte zu minimieren (Bartz-Beielstein und Bartz, 2023, S. 41f). Möglich wäre auch verschiedene OML-Experten zu trainieren.

2.4. Expertenmischung

Die Mixture of Experts-Architektur wurde erstmals von Jacobs et al. (1991) in der Arbeit „Adaptive Mixture of Local Experts“ eingeführt. MoE verfolgt das Prinzip des *Teile- und Herrsche* (engl. Divide and Conquer). Bei dieser Strategie wird ein Problem in Teilprobleme zerlegt, diese einzeln gelöst und anschließend zu einer Gesamtlösung zusammengesetzt (Ernst, 2000, S. 435f.). Jacobs et al. haben ein neues überwachtes Lernverfahren für Systeme vorgestellt, das aus vielen unterschiedlichen neuronalen Netzen besteht. Jedes dieser *Expertennetzwerke* lernt dabei eine Teilmenge der gesamten Menge an Eingangsdaten in Trainingsfällen zu verarbeiten (Jacobs et al., 1991).

2.4.1. Neuronale Netze

Ein *Mehrschichtiges Perzeptron* (MLP) ist ein (tiefes) *Künstliches Neuronales Netz* (KNN), das aus mehreren Schichten von künstlichen Neuronen besteht und zur Lösung von Klassifikations- und Regressionsaufgaben verwendet wird. Die Architektur besteht grundlegend, wie in Abbildung 2.8 gezeigt, aus einer Eingabeschicht, optionalen verborgenen Schichten und einer Ausgabeschicht. In jeder Schicht befinden sich ein oder mehrere künstliche Neuronen. Die Eingabeschicht enthält die Merkmale und Zielwerte aus dem Datensatz $\mathcal{D}_S$. Die Ausgabeschicht die berechneten Prognosen $\hat{y}_t$. Dazwischen befindet sich eine oder mehrere Schichten, in denen jedes Neuron alle Neuronen der vorherigen Schicht über gewichtete Verbindungen $w_{ij} \in \mathbb{R}$ und mit einem Bias-Wert $b_j \in \mathbb{R}$ verbindet (Raschka, Mirjalili und Lorenzen, 2018, S. 385ff).

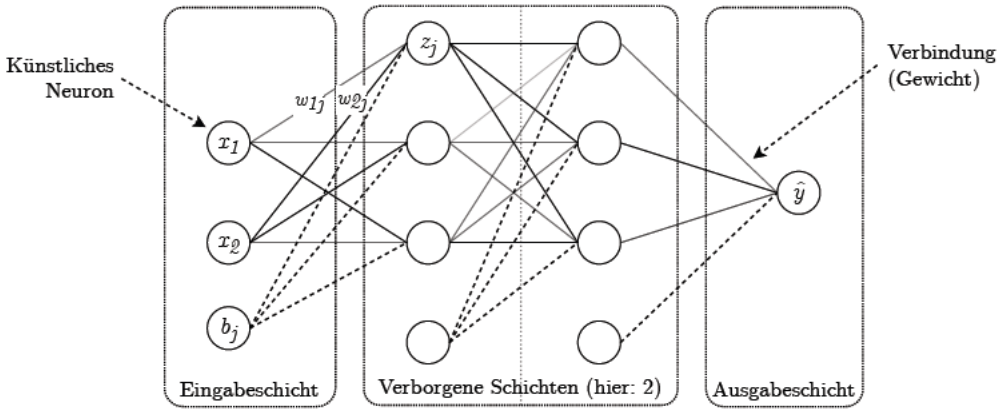


Abbildung 2.8.: Aufbau eines mehrschichtigen künstlichen neuronalen Netzwerks.

Das MLP lernt in mehreren Epochen und berechnet zu Beginn einer Epoche, dem *Feed Forward*, alle gewichteten Summen: $z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j$. Für ein Neuron muss eine *Aktivierungsfunktion* ausgewählt werden, die das Ergebnis z_j der gewichteten Summe verarbeitet. Die Ausgabe der Aktivierungsfunktion $\sigma(z)$ entscheidet über die Intensität des Neurons im gesamten Netzwerk. Eine gängige Aktivierungsfunktion ist *ReLU*: $\sigma(z) = \max(0, z)$. Nach Durchlauf aller Schichten wird rückwärts die *Backpropagation* durchgeführt. Dabei werden die Gewichte w_{ij} und der Bias-Wert b_j so angepasst, dass ein Fehler einer Verlustfunktion $\mathcal{L}(y(x), \hat{y}(x))$ möglichst minimiert wird. Für die Regression wäre der *Mean Squared Error* (MSE) mit $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$ oder für die Klassifikation die Kreuzentropie $-\sum_{i=1}^{n}\sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij})$ für K-Klassen möglich. Allgemein wäre das *Cross Entropy* (CE) für binäre Klassen *Binary Cross Entropy* (BCE). Dafür berechnet *Stochastic Gradient Descent* (SGD) die Ableitungen der Verlustfunktion $\mathcal{L}$ für einen aktuellen Wert, womit das neue Gewicht $w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}(y(x), \hat{y}(x))}{\partial w_{ij}}$ ist und für den neuen Bias-Term $b_j \leftarrow b_j - \eta \frac{\partial \mathcal{L}(y, \hat{y})}{\partial b_j}$ gilt. Das η steht für die Lernrate, ein Hyperparameter, der die Größe der Lernschritte festlegt. Dieses klassische Netzwerk, ohne Rückkopplungen und Schleifen, wird auch *Feedforward Neural Network* (FNN) genannt (Raschka, Mirjalili und Lorenzen, 2018, S. 385ff). Für das OML kann das KNN iterativ und ohne begrenzte Speicheranforderung mit Datenstrom-Datenpaaren trainiert werden. Das Überschreiben von Gewichten in KNN führt zum *katastrophalen Vergessen*. OML bietet Mechanismen, die das Modell stabiler und robuster gegenüber adaptiven Veränderungen im Datenstrom machen (Bartz-Beielstein und Bartz, 2023, S. 41).

Neben dem FNN gibt es weitere Arten, wie RNN und CNN. Bei einem *Recurrent Neural Network* (RNN) sind Schleifen in den Verbindungen enthalten, sodass es Informationen aus früheren Zuständen speichern und zeitliche oder sequenzielle Daten wie Texte oder Zeitreihen

verarbeiten kann. Ein *Convolutional Neural Network* (CNN) ist ein neuronales Netzwerk, das Faltungsoperationen nutzt, um räumliche Hierarchien in Daten, insbesondere bei Bildern, zu erfassen und Muster wie Kanten oder Formen effizient zu erkennen.

2.4.2. Experten mit Gating-Netzwerk

Das Grundprinzip von MoE wird durch Abbildung 2.9 erklärt. Es existiert ein Eingabevektor $x \in \mathbb{R}^n$, der die Eingangsdaten für das MoE-Modell beinhaltet. Weiterhin gibt es eine Menge von N *Experten*, die als Funktionen die Eingabedaten: $\{E_1(x), E_2(x), \ldots, E_N(x)\}$ bekommen. Ein Experte erzeugt für die Eingangsdaten x eine Ausgabe $y_i = E_i(x)$. Ein Experte kann unter anderem ein lineares Modell als neuronales Netz sein: $y_i = E_i(x) = \theta_i x + b_i$. Dabei ist θ_i die gewichtete Matrix des i-ten Expertenmodells und b_i ist der *Bias*.

Daneben gibt es noch *Gating-Netzwerke* $G(x)$, eine Funktion, häufig auch ein neuronales Netz, das als Ergebnis eine Menge von Gewichten für jeden einzelnen Experten zurückgibt: $\{w_1(x), w_2(x), \ldots, w_N(x)\}$. Der Einsatz von mehreren Gating-Netzwerken pro MoE wird *Multi-Gating* genannt und wird häufig im Kontext des Multi-Task-Lernen eingesetzt.

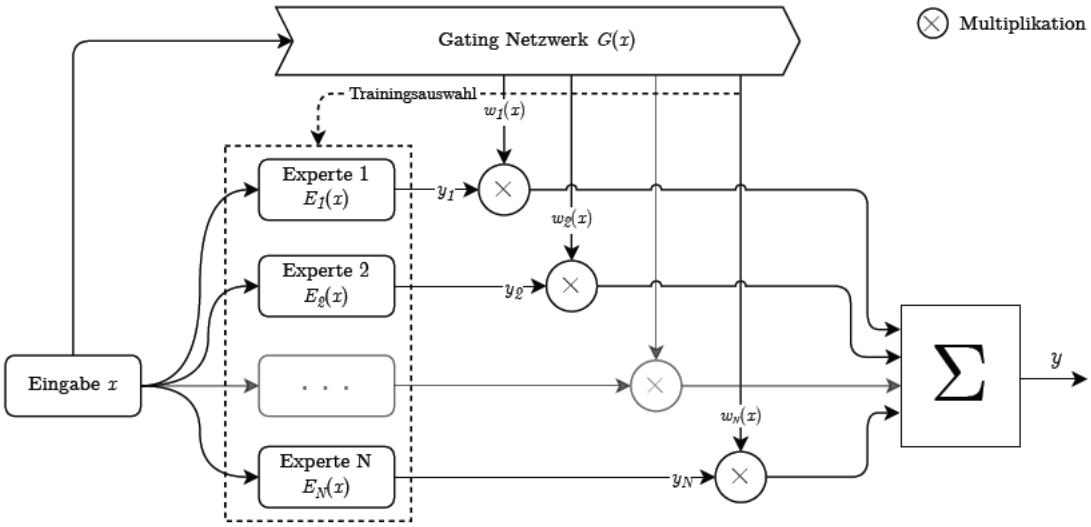


Abbildung 2.9.: Generelle Architektur eines Mixture of Experts Ansatzes.

Angenommen, das Gating-Netzwerk sei parametrisiert mit einer Gewichtsmatrix ϕ und einem möglichen *Bias*-Term b_g, dann sei $g_i(x) = \phi_i x + b_{g_i}$ die Ausgabe für jeden i-ten Experten. Das Gating-Netzwerk entscheidet über die Höhe des Beitrags jedes Expertenergebnisses. Dies wird in der Regel mit einer *Softmax*-Funktion wie aus Formel 2.7 erreicht, um sicherzustellen, dass die Gewichte eine Wahrscheinlichkeitsverteilung bilden:

$$w(x)_i = \frac{e^{g_i(x)}}{\sum_{j=1}^{N} e^{g_j(x)}} = \frac{e^{\phi_i x + b_{g_i}}}{\sum_{j=1}^{N} e^{\phi_j x + b_{g_j}}} = \quad \text{für } i = 1, 2, \ldots, N \tag{2.7}$$

Dabei repräsentiert jedes $w_i \in [0, 1]$ den anteiligen Beitrag des i-ten Experten als Gewicht. Die Summe aller Gewichte muss eins ergeben: $\Sigma_{i=1}^{N} w_i(x) = 1$. Am Ende werden die Ausgaben der einzelnen Experten mit den Gewichten der dazugehörigen Experten aus dem Gating-Netzwerk multipliziert. Zusammengesetzt ergibt sich die Gesamtprognose y für die Eingabe x der Mixture of Experts Netzwerk aus Formel 2.8:

$$y(x) = \sum_{i=1}^{N} E_i(x) \cdot w_i(x) = \sum_{i=1}^{N} (\theta_i x + b_i) \cdot \frac{e^{\phi_i x + b_{g_i}}}{\sum_{j=1}^{N} e^{\phi_j x + b_{g_j}}} \tag{2.8}$$

Anhand des überwachten Lernens wird im Training mit der Ausgabe des Netzwerks und dem tatsächlichen Wert trainiert. Dabei kann sowohl das Gating-Netzwerk als auch die Experten mit gradientenbasierten Optimierungsmethoden trainiert werden, wie SGD. Die Verlustfunktion $\mathcal{L}$ kann etwa der mittlere quadratische Fehler MSE bei Regressionen oder einer der Kreuzentropien bei Klassifizierung sein. Es sind auch andere Verlustfunktionen möglich. Das Experten-Netzwerk kann ein beliebiges KNN sein. Das Gating-Netzwerk entscheidet durch den gewichteten Beitrag und mithilfe der Verlustfunktion in der Backpropagation, wie stark ein Experte genutzt wird. Damit spezialisieren sich die Experten mit der Zeit. Ein MoE wird durch die Hyperparameter, Größe sowie Anzahl der Experten- und Gating-Netzwerke charakterisiert.

2.4.3. Variationen von Gating-Netzwerken

Es gibt verschiedenste Ansätze zur Auswahl von Experten, die im Gate umgesetzt werden können. In der Tabelle 2.3 werden die bekanntesten Gating-Ansätze mit ihren Vorteilen und Nachteilen aufgeführt.

Name	Anz. Experten	Vorteil	Nachteil
Hard-MoE	1	Geringer Aufwand	Schlechtere Leistung
Soft-MoE	$1..N$	Hohe Flexibilität	Hoher Aufwand
Dense-MoE	N	Genaue Vorhersage	Mögliche Redundanz
Sparse-MoE	Top(k)	Effizient, Skalierbar	Komplex, Überanpassung
Adaptive-MoE	$x_t \rightarrow n_t$	Effektiv, Flexibel	Komplex, Latenz

Tabelle 2.3.: Gating-Architekturen mit Vor- und Nachteilen.

Ein möglicher Ansatz ist *Hard-MoE*. Im Hard MoE-Ansatz wird nur eine begrenzte Anzahl (häufig lediglich einen) der verfügbaren Experten für die Verarbeitung eines Inputs aktiviert. Das Gate wählt basierend auf den Eingaben einen spezifischen Experten aus und deaktiviert die anderen. Dafür wird das Gating-Netzwerk als *One-Hot-Encoding*-Vektor trainiert, der den passenden Experten auswählt. Der Rechenaufwand ist gering, da hier nur ein Experte für eine Aufgabe aktiviert wird. Dadurch ist es auch möglich, einfach auf eine höhere Anzahl an Experten zu skalieren. Ein Nachteil könnte darin bestehen, dass die Leistung und Konvergenz darunter leidet, wenn nur ein einzelner Experte ausgewählt wird (Jacobs et al., 1991).

Eine alternative Gating-Architektur dazu ist *Soft*-MoE. Dort werden mehrere Experten gleichzeitig aktiviert und genutzt. Das Gate verteilt dabei Gewichtungen auf alle Experten mithilfe der *Softmax*-Funktion, sodass jede Experteneinheit einen Anteil an der Verarbeitung eines Inputs hat. Da mehrere Experten aktiviert werden, kann das Modell durch eine breitere Wissensbasis oft präzisere Vorhersagen treffen. Die gleichzeitige Aktivierung und Gewichtung aller Experten führt zu einem höheren Rechenaufwand und erhöhter Speichernutzung (M. I. Jordan und R. A. Jacobs, 1994). Bei den *Dense*-MoE werden alle Experten aktiviert. Das ermöglicht das Training von mehreren spezialisierten Modellen, mündet aber gleichzeitig, ähnlich wie bei Soft-MoE, in hohen Rechenaufwand und Speicherbedarf. Zusätzlich besteht die Gefahr der Redundanz, wenn immer alle Experten trainiert werden.

Eine Mischung aus beiden Architekturen sind die *Sparsley Activate* MoE (kurz: *Sparse*-MoE). Sparse-MoE aktiviert nur eine geringe Anzahl der verfügbaren Experten pro Eingabe, um Rechenaufwand zu sparen. Diese Architektur trainiert mit Rauschen ein sparsames Routing, wobei eine geringe Anzahl von den besten k-Experten abhängig von der Eingabe aktiv bleibt. Die verbliebenen Experten werden dann mit Softmax, wie in Soft-MoE, gewichtet. Diese Architektur hat eine hohe Effizienz, da nur ein kleiner Teil der Experten aktiv ist, was die Rechenkapazität und den Speicherbedarf reduziert. Außerdem können große Modelle mit einer Vielzahl von Experten ohne hohe Kosten betrieben werden. Eine Herausforderung ist die Komplexität eines effizienten Routings des Gating-Netzwerks. Außerdem gibt es bei zu klein gewählten k das Risiko, dass einige Experten zu stark trainiert werden (Shazeer et al., 2017).

Bei dem *Adaptive-MoE* passt sich die Anzahl aktivierten Experten dynamisch basierend auf den Anforderungen einer Eingabe an, ohne eine konstante Vorgabe k. Dies bedeutet, dass einfache Eingaben nur wenige Experten nutzen, während komplexe Eingaben mehrere Experten aktivieren. Dies schafft eine gute Mischung aus Effizienz und Genauigkeit. Ressourcen werden nur dann intensiv genutzt, wenn die Eingabe es erfordert. Adaptive MoE-Modelle sind flexibler und können Ressourcen entsprechend der Eingabe anpassen. Ein Nachteil ist das komplexe Design sowie das Training des adaptiven Gating-Netwerks. Ein weiterer Nachteil

ist, dass die Berechnungsverzögerungen von der Anzahl aktiver Experten abhängen, was die Latenz unvorhersehbar macht (J. Li et al., 2023).

2.4.4. Verkettung von MoE

Werden MoE-Modelle hierarchisch verknüpft oder verschachtelt, wird dies als *Hierarchisches Mixture of Experts* (HME) betitelt. Eingeführt wurde diese baumartige Architektur von M. I. Jordan und R. A. Jacobs (1994). In Abbildung 2.10 ist ein hierarchisches Mixture of Expert Modell der Höhe $h = 4$ mit zwei Experten-Netzwerken pro MoE dargestellt.

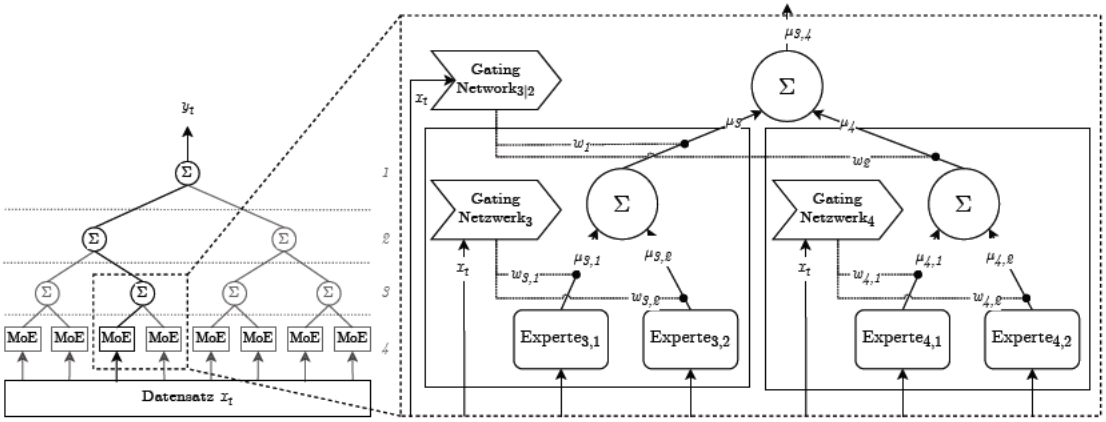


Abbildung 2.10.: Architektur eines Hierarchischen MoE als binärer Baum mit vier Ebenen.

Aufgebaut sind die HME als binäre Bäume. Sie besitzen 2^{h-1} MoE (Rechtecke) in den Blättern und $2^{h-1} - 1$ alleinstehende Gating-Netzwerke (Kreise) in den Knoten (Gumm, Sommer und Hesse, 2011, S. 387f). Das j-Experten-Netzwerk des i-ten MoE kann über das Tupel $E_{i,j}$ lokalisiert werden. Die Ergebnisse der MoE werden mit den alleinstehenden Gating-Netzwerken bis zum Wurzelknoten gewichtet aufsummiert. In der obersten Ebene, dem Wurzelknoten, wird das Endergebnis y ermittelt.

Das Training der probabilistischen Modellierung wird durch zwei Schritte der *Erwartungs-Maximierung* (EM) umgesetzt. Wie in Abbildung 2.10 dargestellt, verläuft das Training *bottom-up*. Jedes MoE generiert je ein Zwischenergebnis μ_i mit der bedingten Wahrscheinlichkeit $p(x_t; \theta)$, mit den Eingangsdaten x_t, den tatsächlichen Werten y_t und dem Parameter θ. Die Gewichte der Gating-Netzwerke w führen über den *Satz von Bayes*[1] zum Erwartungswert $\gamma_{i,j}$ (Schritt „E“). Die neuen Parameter für θ^* und ϕ^* werden durch die Maximierung der *Log-Likelihood*-Funktion (Schritt „M“) nach Formel 2.9 abgeleitet.

[1] Satz von Bayes: $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$

$$\gamma_{t,i,j} = \underbrace{\frac{w_i(x_t|\phi)p_i(x_t;\theta_i)}{\sum_j w_j(x_t;\phi)p(x_t;\theta_j)}}_{Erwartung} \rightarrow \begin{aligned} \theta^*_{i,j} &= \arg\max_{\theta_{i,j}} \sum_{t}^{n\in|\mathcal{D}_S|} \gamma_{t,i,j} \log p(x_t;\theta_{i,j}) \\ \phi^*_{i,j} &= \underbrace{\arg\max_{\phi_{i,j}} \sum_{t}^{n\in|\mathcal{D}_S|} \sum_{k} \gamma_{t,k} \sum_{l} \gamma_{t,l|k} \log w_{t,l|k}}_{Maximierung} \end{aligned} \tag{2.9}$$

k steht für das Gating-Netzwerk in dem Knoten und l für die Ebene des binären Baumes. Dieser Ansatz ist effizient bei großen Modellen, da hierarchische Strukturen erlauben, nur relevante Experten zu aktivieren, was die Rechenleistung optimiert. Ein Vorteil ist auch, dass Experten in verschiedenen Ebenen auf spezifische Teilaufgaben spezialisiert werden können. Die Umsetzung von Hierarchischen-Gates ist komplex und zeitgleich auch schwierig zu trainieren (M. I. Jordan und R. A. Jacobs, 1994; M. Jordan und R. Jacobs, 1991).

Ein ähnlicher Ansatz, der MoE in Ebenen einsetzt, sind die *Deep Mixture-of-Experts* (DMoE). Statt einer baumartigen Struktur wird eine lineare Verkettung, wie in Abbildung 2.11, vorgenommen. Die Forscher Eigen, Ranzato und Sutskever (2014) verwenden die Ausgabe des ersten MoE als Eingabedaten des nächsten MoE und so weiter. Daraus ergibt sich folgender Ansatz: $x_i = y_{i-1} \implies y_i = f_i(y_{i-1})$.

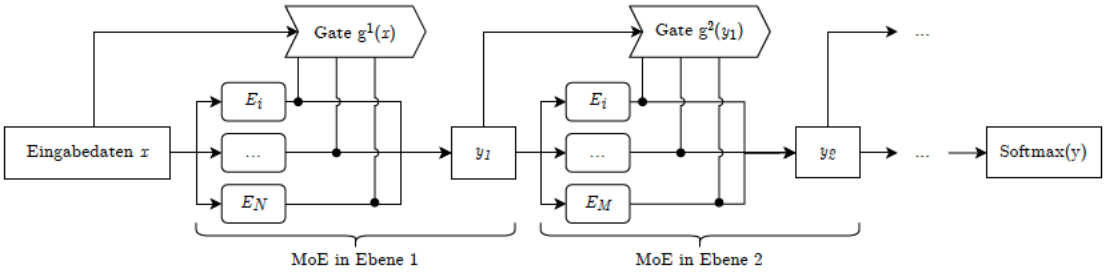


Abbildung 2.11.: Verkettung von MoE in Deep-MoE.

Jedoch führt der Einsatz von SGD zu einem degenerierten lokalen Minimum, da Experten, die in den ersten Ebenen bevorzugt werden, durch steigende Gating-Gewichte immer mehr dominieren. Zur Minderung dieses Effekts, wird das Gating-Gewicht eines Experten i der Ebene l auf 0 gesetzt $g^l_i(x_t) = 0$, wenn die relative Gating-Zuweisung einen Schwellenwert m übertritt: $G^l_i(t) - \overline{G^l}(t) > m$. Dabei entspricht $G^l_i(t) = \sum_{t'=1}^{t} g^l_i(x_{t'})$ dem aufsummierten Gating-Gewicht bis zum Zeitpunkt t des Experten i in der Ebene l und $\overline{G^l}(t) = \frac{1}{N}\sum_{i=1}^{N} G^l_i(t)$ dem Ebenen-Durchschnitt seiner Experten. Die DMoE bilden eine Grundlage für den Einsatz von MoE in Transformern.

2.4.5. Einsatz in Transformern

MoE werden immer häufiger in der Transformer-Architektur eingesetzt, aus denen beispielsweise *Large Language Model* (LLM) trainiert werden. Zu den bekanntesten LLM-Modellen zählen GLaM von *Google*, NLLB-200 von *Meta AI* und 8x7B von *Mixtral AI*. Alle genannten LLM nutzen Sparse-MoE mit $k = 2$, um die besten zwei Experten für eine Eingabe auszuwählen. Das Modell von Meta nutzt zusätzlich eine hierarchische Struktur, wie in HME (N. Du et al., 2022; Team et al., 2022; A. Q. Jiang et al., 2024). Ein Transformer besteht aus einem *Encoder* und einem *Decoder* und wird mit der Abbildung 2.12 illustriert.

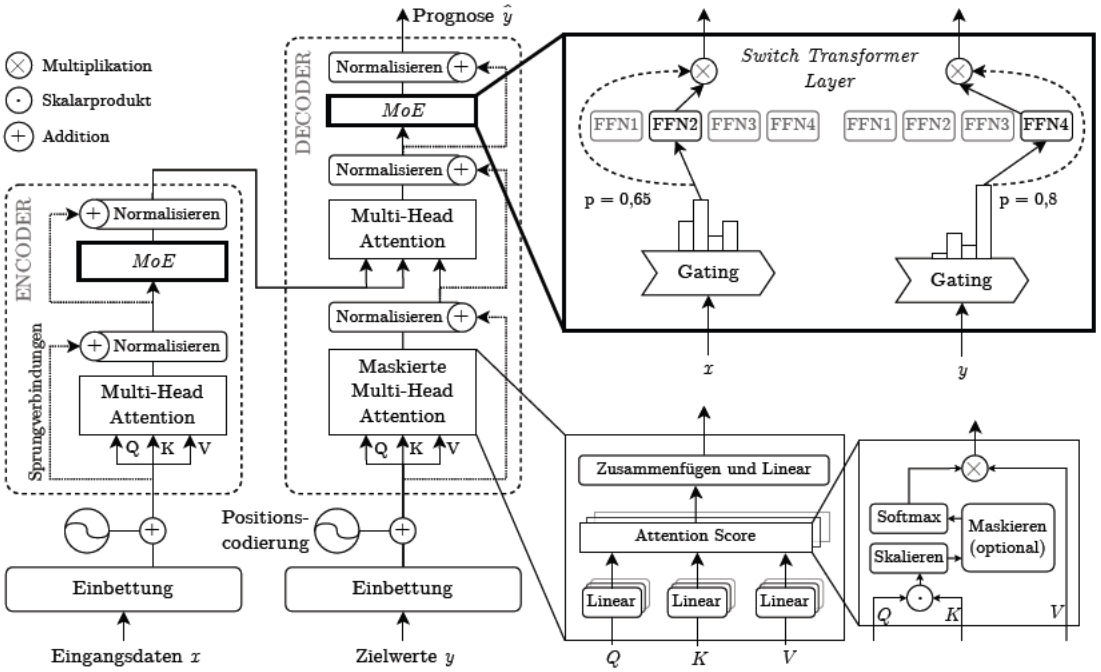


Abbildung 2.12.: Transformer mit FNN (kursiv) ersetzt durch MoE. Hier: Sparse-MoE mit $k = 1$.

Der Encoder wandelt Eingabesequenzen, wie einzelne Wörter eines Satzes, als Token in einen kontinuierlichen Repräsentationsvektor um. Der Decoder wandelt diese Repräsentation dann in eine Zielsequenz um. In der Einbettung werden die Token in numerische Vektoren umgewandelt. Die Positionscodierung sorgt für die richtige Reihenfolge. In den *Attention*-Ebenen wird die Beziehung zwischen dem Token zu allen anderen Token und sich selbst als Ähnlichkeit berechnet. Dafür wird die Abfrage ($Q = XW^Q$), der Schlüssel ($K = XW^K$) und der Wert ($V = XW^V$) mit dem Eingangsvektor X und einer lernbaren Gewichtungsmatrix

W verrechnet. Zwischen den Zeilen von K und den transponierten Spalten von Q wird das Skalarprodukt angewendet. Der *Softmax*-Wert wird dann mit V multipliziert (Formel 2.10).

$$\text{AttentionScore}(Q, K, V) = \textit{Softmax}\left(\frac{Q \cdot K^\top}{\sqrt{d_K}}\right) V \tag{2.10}$$

d_K steht für die Dimension der Schlüssel (K). Ein anschauliches Beispiel ist der Satz „Katzen lieben Fisch“, bei dem errechnet wird, wie stark „Katze“ (Q) zu allen anderen Wörtern (K) passt (z.B. „Fisch“ ist relevanter als „lieben“). Diese Relevanz beeinflusst dann, welche Informationen (V) an „Katze“ weitergegeben werden. In der *Multi-Head Attention* Ebene, werden mehrere parallele *Attention* verwendet, um verschiedene Arten von Beziehungen zwischen Wörtern zu lernen. In einem traditionellen Transformer würde auf diesen Ebenen ein FNN folgen. In den MoE-Transformern werden die FNN durch MoE ersetzt. Die Ergebnisse aller Ebenen werden mit ihren jeweiligen ursprünglichen Eingabedaten über die *Sprungverbindung* addiert und normalisiert. Das stabilisiert den Gradientenfluss durch das tiefe Netzwerk und verbessert die Konvergenz des Trainings. Am Ende des Decoders ist es noch möglich, durch eine Softmax-Ebene, die Prognosen $\hat{y}$ als Wahrscheinlichkeitsverteilung auszugeben. (Vaswani et al., 2017; Fedus, Zoph und Shazeer, 2022).

In der Abbildung 2.12 wird ein *Switch Transformer* nach Fedus, Zoph und Shazeer (2022) mit einem *Switch Routing* verwendet. Diese sind eine Vereinfachung der Sparse-MoE von Shazeer et al. (2017), da nur ein Experte ($k = 1$) pro Token gewählt wird. Damit kann der Rechenaufwand für das Gating und den Speicherplatz für Experten bei gleichbleibender Modellgüte reduziert werden. Die verschiedenen Experten der Transformer-MoE konzentrieren sich jeweils beispielsweise auf Satzzeichen, Verben, visuelle Beschreibungen und weiteres. Mit den reduziert aktivierten MoE kann die Anzahl der Parameter durch neue Experten-Netzwerke stark erhöht werden, ohne den linearen Anstieg der Berechnungskosten. Diese Experten-Netzwerke können damit spezielle Aufgaben effektiver lösen, da sie unterschiedliche Teile des Modells für unterschiedliche Aufgaben verwenden. Kleinere Transformer-MoE erzielen damit genauere Ergebnisse als vergleichbare oder größere Transformer Modelle ohne MoE (A. Q. Jiang et al., 2024). Transformer-MoE nutzen Parameter der trainierten Experten-Netzwerke $\mathcal{P}_E$ und gemeinsame Parameter $\mathcal{P}_{\text{geteilt}}$, die immer genutzt werden, wie in den *Attention*-Ebenen. Mit der Anzahl der aktiven Experten $|E|$ ergibt sich die Gesamtanzahl: $\mathcal{P}_{\text{Gesamt}} = \mathcal{P}_{\text{geteilt}} + |E| \cdot \mathcal{P}_E$.

3. Verwandte Arbeiten

Zunächst wurden die verwandten Arbeiten aus aktueller Forschung betrachtet. Dafür wurde eine systematische Literaturrecherche nach dem PRISMA-Schema von Page et al. (2021) durchgeführt. Das Ziel der Literaturrecherche ist es, einen Überblick über die aktuelle Forschung von MoE im Kontext adaptiver ML-Verfahren zu erhalten. Dabei soll untersucht werden, wie Gating-Verfahren in MoE eingesetzt werden können, um in adaptiven und aufgaben-agnostischen Umgebungen angewendet zu werden. Aus diesen Erkenntnissen sollen offene Probleme, Herausforderungen und Chancen abgeleitet werden, die näher in dieser Arbeit beleuchtet werden.

3.1. Systematische Literaturrecherche

Um die besten Ergebnisse während der Recherche mit relevanten Veröffentlichungen zu erhalten, wurden indizierte Literaturverzeichnisse verwendet. Diese Quellen ermöglichen eine breite Suche nach veröffentlichten Arbeiten, die andernfalls übersehen werden würden. Bei der Veröffentlichung wurden nur *peer-reviewed* Artikel berücksichtigt, weswegen nur die Verzeichnisse *IEEE Explore* (IEEE), *SpringerLink* (SL), *Elsevier Science Direct* (SD) und *ACM Digital Library* (ACM) durchsucht wurden. Für ACM wurde auch die Literatur der erweiterten Datenbank berücksichtigt. Für die Suche in den Datenbanken wurde einheitlich die gleiche Suchanfrage verwendet und ist in englischer Sprache erfolgt, um eine möglichst breite Literaturauswahl zu erhalten:

```
("Mixture of Experts" OR "MoE") AND ("Machine Learning"
OR "ML") AND ("Gating Network") AND ("Adaptive" OR "Task
agnostic")
```

Zunächst wurden die Begriffe „Mixture of Experts“ und „Machine Learning“ zur Suchanfrage hinzugefügt, um Ergebnisse im Kontext des maschinellem Lernens und der Expertenmischung zu erhalten. In vielen Arbeiten werden häufig Abkürzungen verwendet. Angesichts dessen wurde „MoE“ für Mixture of Experts und „ML“ für Machine Learning ergänzt. Zusätzlich wurden noch die Begriffe „Adaptive“ und „Task agnostic“ im Suchtext hinzugefügt, um Veröffentli-

chungen zu erhalten, die sich mit adaptiven oder aufgaben-unspezifischen Problemstellungen beschäftigt haben. Ein zusätzlicher Fokus auf Gating-Netzwerke wird durch das Hinzufügen des Begriffs „Gating Network“ gelegt.

Die einzelnen Begriffe wurden mit den boolschen Operatoren verknüpft. Der Operator für Disjunktion OR wird verwendet, um Begriffe und ihre Abkürzungen gleichermaßen im Einschluss zu berücksichtigen. Die verknüpfte Konjunktion AND verringert als Schnittmenge der Teilsuchergebnisse das gesamte Suchergebnis. Damit die Reihenfolge der Wörter innerhalb der Begriffe richtig interpretiert wird, wurde mithilfe der Anführungszeichen eine Phrasensuche durchgeführt. Der Zeitraum für die Veröffentlichung wurde auf vier Jahre festgelegt: 2021 bis 2024, um alle aktuellen Veröffentlichungen zu erhalten. Die Suche wurde im August 2024 durchgeführt.

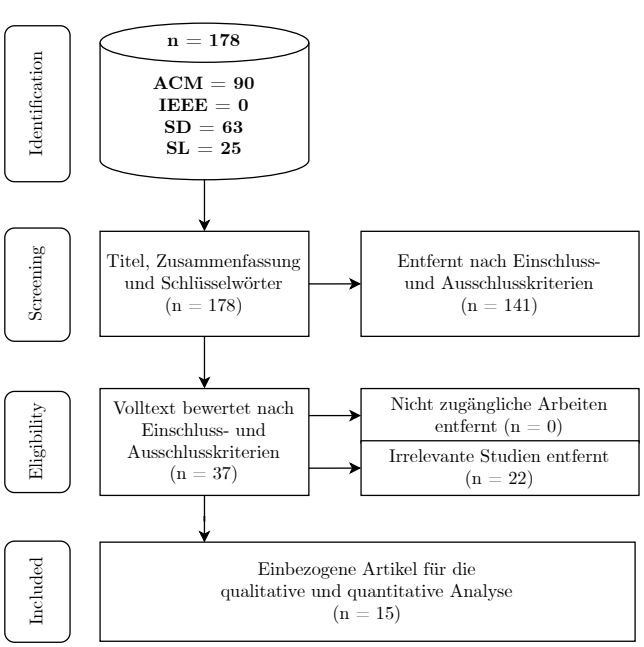
Während der *Screening*-Phase wurden Leitlinien auf Titel, Schlagwörter und Zusammenfassung angewendet, um irrelevante Veröffentlichungen auszuschließen. Ergebnisse wurden nicht weiter berücksichtigt wenn:

1. Adaptives Lernen nicht im Sinne von künstlicher Intelligenz gemeint ist.
2. Reinforcement Learning eingesetzt wird.
3. Der Schwerpunkt nicht auf MoE liegt.

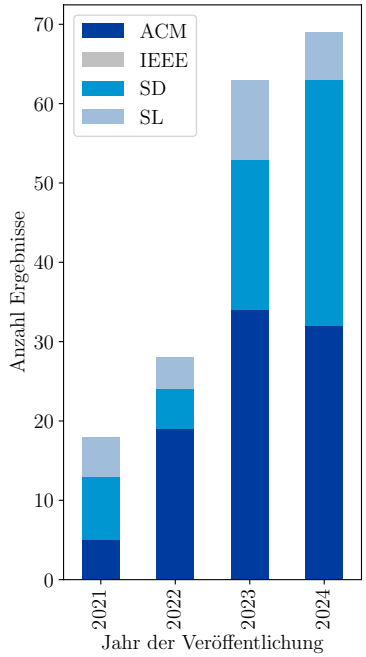
Suchergebnisse aus *Proceedings* wurden nicht explizit berücksichtigt, da die veröffentlichten Arbeiten bereits in den Suchergebnissen angezeigt werden. Diese Kriterien konnten verwendet werden, um die Anzahl der Volltextarbeiten für die spätere Evaluierung zu reduzieren. Damit ist ein Fokus auf potenziell relevante Arbeiten in der Phase *Eligibility* möglich.

In Abbildung 3.1a wird die systematische Literaturrecherche schematisch dargestellt. Die initiale Suche mit der Suchanfrage ergab 178 Veröffentlichungen, wovon 90 aus der *ACM Digital Library*, keine aus *IEEE Explore* sowie 63 aus *Elsevier Science Direct* und 25 von *SpringerLink* kamen. Nach der ersten Sichtung der Titel, Schlagwörter und Zusammenfassung konnten 141 Suchergebnisse entfernt werden. Davon waren 35 Duplikate, die als Redundanz in mehreren Datenbanken auftauchten. In der nächsten Eignungsprüfung wurden 22 irrelevante Arbeiten aus den Suchergebnissen entfernt. In der finalen qualitativen und quantitativen Betrachtung wurde die Auswahl auf 15 Artikel beschränkt, um den Rahmen der Arbeit nicht zu sprengen.

Wie in der Abbildung 3.1b zu erkennen, ist ein klarer Trend an MoE für adaptive und aufgaben-agnostische Problemstellungen in den vergangenen vier Jahren zu erkennen. Vor allem in den Jahren 2023 und 2024 ist die Anzahl der Veröffentlichungen stark gestiegen. Im nächsten Abschnitt werden die ausgewählten Artikel für die qualitative Analyse näher betrachtet.



(a) PRISMA-Flussdiagramm



(b) Häufigkeit nach Jahr (N = 178)

Abbildung 3.1.: Ergebnisse der systematischen Literaturrecherche.

3.2. Qualitative Analyse

Die ausgewählten 15 Veröffentlichungen wurden nach ihrem MoE-Schwerpunkt in die drei Kategorien „Gating-Ansätze“, „Multi-Task“ und „Adaptives Verhalten“ eindeutig zugeordnet. Die Kategorien wurden aus den Schlagwörtern `Gating Network`, `Adaptive` und `Task agnostic` (für Multi-Task-Lernen) der oberen Suchanfrage abgeleitet. Eine Übersicht der Zuordnung nach Quelle zeigt Tabelle 3.1. Zuerkennen ist, dass viele Ansätze sich hauptsächlich auf neue oder spezielle Gating-Ansätze konzentrieren. Mit einem speziellen Ansatz, dem *Multi-Mixture-of-Experts* (MMoE) Ansatz, lassen sich Aufgaben des Multi-Task-Lernen lösen. Am wenigsten wird sich mit Ansätzen beschäftigt, die sich in adaptiven Szenarien bewegen.

Alle Arbeiten haben die Gemeinsamkeit der grundlegend verwendeten MoE-Architektur. In der ersten Kategorie „Gating-Ansätze“ geht es um Veröffentlichungen, die sich auf MoE- oder Gating-Mechanismen konzentrieren. Diese Mechanismen ermöglichen eine dynamische Auswahl von Modellkomponenten und verbessern die Leistung bei komplexen Aufgaben, indem sie sich auf relevante Merkmale oder Experten konzentrieren. Mit der „Multi-Task“-Kategorie sind Beiträge ausgewählt, die MoE nutzen, um mehrere Aufgaben gleichzeitig bewältigen zu können oder so konzipiert sind, dass sie nicht an bestimmte Aufgaben gebunden sind. Sie

Kategorie	Anzahl	Referenz
Gating-Ansätze	7	Huang et al. (2024); Z. Chen et al. (2024); Hihn und Braun (2024); Hihn und Braun (2023); Chen, Yue und Shi (2023); Wang et al. (2021); Kobayashi und Shirayama (2021)
Multi-Task	5	Hu et al. (2024); Park et al. (2024); S. Jiang et al. (2024); Rahman et al. (2024); J. Du et al. (2022)
Adaptives Verhalten	3	C. Chen et al. (2024); Sharma, Henderson und Ghosh (2023); W. Chen et al. (2023)

Tabelle 3.1.: Übersicht der Kategorien aus der qualitativen Literraturrecherche.

zielen darauf ab, die Effizienz und Leistung durch die Nutzung von gemeinsamem Wissen über Aufgaben hinweg zu verbessern. Die Kategorie „Adaptives Verhalten“ umfasst Arbeiten, die sich auf MoE-Modelle konzentrieren, die sich an veränderte Datenverteilungen oder Umgebungen anpassen. Sie betonen das Lernen aus neuen Daten unter Beibehaltung des zuvor erworbenen Wissens, was in dynamischen Umgebungen, wie OML, von entscheidender Bedeutung ist.

3.2.1. Gating-Ansätze

MoE vs. CNN In dem Paper von Z. Chen et al. (2024) wurde untersucht, wie Router lernen Daten effektiv den richtigen Experten zuzuweisen und welche Vorteile das *Divide-and-Conquer*-Prinzip der MoE gegenüber CNN bringt. Nach der Einführung in die theoretischen Grundlagen wird eine Datenverteilung mit Clusterstrukturen eingeführt, welche die Vorteile der MoE-Architektur verdeutlichen kann. Empirisch wurde dies durch Experimente auf synthetischen und realen Datensätzen (CIFAR-10 und CIFAR-10-Rotate) evaluiert. Verglichen wurde ein klassisches CNN mit zwei Schichten, mit einem Sparse-MoE Top$(k = 1)$. Auf synthetischen Datensätzen erreicht MoE mit nicht linearen Experten eine Genauigkeit von $> 99\%$, während lineare MoE-Modelle und das CNN deutlich schlechter abschneiden. In den realen Daten hängt es von der Clusterstruktur der Daten ab. In dem modifizierten realen CIFAR-10-Rotate Datensatz zeigt MoE klare Verbesserungen gegenüber dem CNN. Die Entropie, die die Datenverteilung über die nicht linearen Experten beschreibt, ist fast null. Das deutet auf eine klare Spezialisierung hin.

Bayesian-Gating Die Forscher Kobayashi und Shirayama (2021) haben eine Methode mit Bayesschen Netzwerken entwickelt, die dem MoE-Ansatz ähnelt. Ziel war eine möglichst robuste und genaue Vorhersage, ob die Rendite des Nikkei 225 Börsenindex in der nächsten Periode über oder unter dem Durchschnitt liegt (binäre Klassifikation). Dafür wurden die

Trainingsdaten aus sechs Börsenindizes mit `K-means`, ähnlich wie in vorherigen Arbeiten, für sieben Experimente in ein dynamisches und zwei bis sieben normale Cluster aufgeteilt. Das Bayessches-Netzwerk wurde für jedes KNN (FNN, RNN und LSTM) trainiert, damit es die Beziehung zwischen Eingabedaten und dem KNN bestmöglich darstellt. Dieser Gating-Mechanismus wählt das beste KNN für die Vorhersage aus, indem er die Wahrscheinlichkeit berechnet, mit der ein Datenpunkt zu einem bestimmten Cluster und somit zu einem KNN gehört. Verglichen wurden die Ergebnisse mit einfachen und mehrfachen KNN, einem Deep-MoE und Hard-MoE sowie einem früheren Ansatz der Autoren mit Naive-Bayes-Klassifikator. Das Bayesian-Gating erreicht bei $K = 6$ Clustern die besten Ergebnisse (Accuracy: 68,36 %; F1-Score: 66,89 %) und ist damit besser als alle anderen Ansätze. Das probabilistische Gating ermöglicht zudem stabile Vorhersagen.

WEKT Das *Option-Weighting-Enhanced Mixture-of-Experts Knowledge Tracing* (WEKT)-Modell wird von Huang et al. (2024) vorgestellt. Ziel ist es, die Wissenszustände von Lernenden genauer zu modellieren, indem sowohl die Korrektheit der Antworten als auch die Wahl von Optionen bei Multiple-Choice-Fragen aus Aufgaben berücksichtigt wird. In der Methodik werden mehrere Ansätze kombiniert. Das WEKT-Framework nutzt gewichtete Bewertungen der Antwortoptionen, die Teilleistungen und Fehlerarten berücksichtigen. Ein Gating-Mechanismus des MoE filtert und kombiniert die Outputs der aufgabenspezifischen Experten dynamisch. Dabei werden zwei Aufgaben simultan gelöst: Vorhersage der Antwortkorrektheit und der gewählten Option. Als Expertennetzwerke werden LSTMs und Multi-Head-Attention-Schichten genutzt, um die insgesamt 34,45 Millionen Daten der Lernhistorie von Studierenden zu analyisieren. WEKT übertraff zehn Referenzmodelle in allen Metriken (Accuracy, AUC und RMSE). Zusätzlich ist das Modell effizienter als die vergleichenden Modelle, was auf den Einsatz von MoE zurückzuführen ist. WEKT erzielt präzisere Vorhersagen der Lernendenleistung, indem es als neuen Ansatz die Teilleistungen durch Options-Gewichtung berücksichtigt und Fehlerursachen analysiert.

MoVE Die Forscher Hihn und Braun beschäftigen sich in zwei Veröffentlichungen von 2023 und 2024 mit dem Problem des katastrophalen Vergessens in MoE. Dieses Problem ist insbesondere in kontinuierlichen Lernumgebungen, wie der untersuchten OML-Umgebung problematisch. Beide Studien zielen darauf ab, task-agnostische Ansätze zu entwickeln, die ohne explizite Aufgabeninformationen funktionieren und Wissen nachhaltig bewahren. Im Jahr 2023 wurde ein hierarchischer Ansatz mit *Mixture-of-Variational-Experts* (MoVE)-Schichten eingeführt, die spezialisierte Sub-Netze durch eine Gating-Policy aktivieren und Diversität sowie Spezialisierung fördern. Der Mechanismus nutzt die Kullback-Leibler-Divergenz (KL-

Divergenz) nach Kullback und Leibler (1951), um die Abweichung zwischen posterioren und prioren Verteilungen der Expertenparameter zu minimieren und so ein Gleichgewicht zwischen Lernen und Vergessen zu schaffen. Die Arbeit von 2024 erweitert diesen Ansatz, indem sie Mutual Information einführt, um Experten effizient zu spezialisieren, und Dirichlet-Prozesse zur dynamischen Erweiterung der Experten nutzt. Diese Methode verzichtet auf generative Modelle und Replay-Mechanismen, was ihre Flexibilität erhöht. Beide Arbeiten führten Experimente auf Standard-Datensätzen wie Split-MNIST und Split-CIFAR-10/100 sowie auf Reinforcement Learning-Aufgaben durch. Die Ergebnisse zeigen, dass beide Ansätze effektiv das Vergessen reduzieren und spezialisierte Experten für verschiedene Aufgaben schaffen. Während die Arbeit von 2023 durch eine hierarchische Struktur und gezielte Diversitätsziele überzeugt, liefert die Arbeit von 2024 eine flexiblere, online-fähige Lösung für dynamische und komplexe CL-Szenarien (Hihn und Braun, 2023; Hihn und Braun, 2024).

LTMoE *Space-Time Video Super-Resolution* (STVSR) dient dazu, Videos mit niedriger Auflösung und niedriger Bildrate in hochauflösende Videos mit hoher Bildrate zu transformieren. Chen, Yue und Shi (2023) führen das *Long-Term Temporal Feature Aggregation Network* (LTFA-Net) ein, das für die Feature-Interpolation ein neuartiges LTMoE nutzen. *Long-Term Mixture-of-Experts* kombiniert mehrere Experten mit Convolutional-Schichten (u.a. ConvNext von Liu et al., 2022), um räumlich-zeitliche Features aus mehreren benachbarten Frames zu extrahieren und mit einem Soft-Gating-Netz zu gewichten. In Phase zwei und drei werden Convolutional-Schichten genutzt, um lokale und globale Bewegungen zu verbessern. In der letzten Phase wird die räumliche Auflösung durch ConvNext-Blöcke erhöht. Mit dem Datensatz Vimeo-90K wurde das Netzwerk trainiert. Evaluiert wurde mit einem Adobe- und GoPro-Datensatz. Alle Videos wurden vorab durch bikubische Interpolation in der Auflösung verringert. Mithilfe des Signal-Rausch-Verhältnisses und der wahrgenommenen Ähnlichkeit zwischen zwei Bildern hinsichtlich Helligkeit, Kontrast und Struktur wurden die Ergebnisse bewertet. Quantitativ erzielte das LTFA-Net bessere Ergebnisse als vergleichbare Ansätze und mit einer Inferenzgeschwindigkeit von 14,53 Bilder pro Sekunde bei 12,41 Millionen Parametern. Eine Leistungssteigerung wurde vor allem bei Gating-Netzen bemerkt, die Experten disjunkt gewichten.

GNN Das Paper von Wang et al. (2021) stellt ein Framework für die interaktive Steuerung virtueller Charaktere in Echtzeit vor. Als Grundlage wird ein *Gated Neural Network* (GNN) als Gating-Ansatz in der MoE-Architektur eingeführt. Ziel ist es, eine hohe Bewegungsrealität und Anpassungsfähigkeit an verschiedene Eingaben und Umgebungen zu erreichen, während gleichzeitig die Effizienz der Berechnungen optimiert wird. In der Architektur kommt ein

MoE zum Einsatz, das zwei Gruppen von Experten-Netzwerke verwalten soll. In dem einen Netzwerk geht es um die feine Anpassung von Bewegungen (Geschwindigkeit), in dem anderen Netzwerk um große Übergänge zwischen Bewegungsmodi (gehen zu laufen). Das GNN kann zeitliche Sequenzen in Echtzeit verarbeiten und Gewichte errechnen und anpassen. Damit ist es verantwortlich für die Auswahl von Experten in beiden Experten-Gruppen. Die Experimente basieren auf offenen Motion-Capture-Datensätzen von CMU und der Universität Edinburgh. Diese enthalten realistische Bewegungsdaten in verschiedenen Szenarien (Gehen, Laufen, Springen, Bücken). Das Modell wird mit bestehenden Methoden wie PFNN (Phase-Functioned Neural Network ohne MoE) und MANN (Mode-Adaptive Neural Network ohne MoE) verglichen. Der neue Ansatz übertraf die anderen Methoden durch glattere Übergänge, menschlichere Bewegungen und eine höhere Berechnungsgeschwindigkeit, was auf das dynamische GNN-Gating zurückgeführt wurde.

3.2.2. Umsetzung von Multi-Task Experten

Multi-Task-Lernen (MTL) wird in vielen realen Anwendungen verwendet, um mehrere Ziele gleichzeitig zu optimieren. Ein Beispiel ist das Empfehlungssystem für das nächste Video auf YouTube (Zhao et al., 2019). Alle ausgewählten Beiträge zur Lösung von Multi-Task Problemen verwenden *Multi-Mixture-of-Experts* (MMoE), auch Multi-Gate-MoE genannt. Diese Architektur wurde von Ma et al. (2018) eingeführt. Wie in Abbildung 3.2 gezeigt, verwendet dieser Ansatz, im Gegensatz zum klassischen One-Gate-MoE, mehrere Gates – pro Aufgabe.

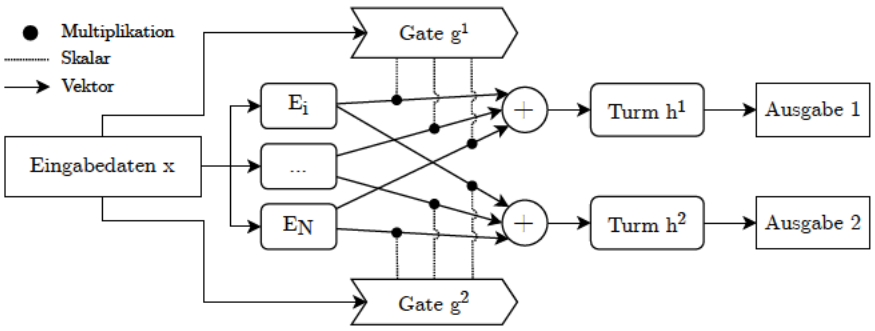


Abbildung 3.2.: Multi-Gate Mixture of Experts mit zwei Gates für Multi-Task Aufgaben.

Bei K-Aufgabenstellungen entscheidet jedes aufgabenspezifische Gate g^k mithilfe der errechneten Gewichte $g^k(x_t) = \text{Softmax}(\theta_{gk} \cdot x_t)$ über den Beitrag einzelner Experten zur Lösung ihrer unterschiedlichen Aufgaben. Die Experten werden unter den Aufgaben geteilt. Die gewichteten Ergebnisse der einzelnen Experten fließen dann als Eingabe in den aufgabenspezifischen „Turm“. Dieses trainierbare neuronale Netz h^k wird genutzt, um die Optimierung

der Aufgabe zu entkoppeln. Die Gesamtausgabe für eine Aufgabe k ergibt sich damit aus Formel 3.1.

$$y_k = h^k \left(\sum_{i=1}^{n} g_i^k(x_t) f_i(x_t) \right) \tag{3.1}$$

Anhand von Experimenten zeigten die Autoren, dass die MMoE-Architektur die Basismethoden übertrifft, insbesondere wenn die Korrelation zwischen den Aufgaben gering ist. Diese Architektur ermöglicht es, Experten je nach Anforderung unterschiedlich stark zu nutzen. Trotz zusätzlicher Gating-Netzwerken, bleibt die Parameterzahl überschaubar.

ACMoE Die MMoE wurden von Park et al. (2024) für ein Empfehlungssystem mit zwei Hauptansätzen verwendet: *Single-Domain Sequential Recommendation* (SDSR) und *Cross-Domain Sequential Recommendation* (CDSR). Beim Ersteren handelt es sich um Empfehlungen, basierend auf Interaktionen mit einer einzigen Domäne. Beim Zweiteren basieren die Empfehlungen auf mehreren Domänen. Die Herausforderung ist der negative Transfer zwischen schwach korrelierten Domänen, der die Leistung von CDSR beeinträchtigt. Der *Negative Transfer Gap* (NTG) ist der Leistungsunterschied, hier zwischen SDSR und CDSR. Ziel der Forscher war die Entwicklung eines Frameworks, das den NTG in CDSR minimiert und die Leistung aller Domänen stetig verbessert. So wurde das *Asymmetric Cooperative Network mit Mixture-of-Sequential Experts* (ACMoE) entwickelt. Genutzt wird ein MTL-Szenario, das SDSR und CDSR gleichzeitig behandelt. Der Gating-Mechanismus verteilt die Eingaben auf die Experten für SDSR und CDSR. Für die Experimente wurden fünf Domänen aus einem Amazon- und einem Telco-Datensatz verwendet. Das ACMoE-Modell wurde mit 25 state-of-the-art Modellen und domänenspezifischen Metriken wie *Klickrate* der besten fünf Produkte verglichen. Das entwickelte Modell übertraf die Referenzmodelle um 38,81 % in der Klickrate der Telco-Domäne. In der produktiven Umsetzung wurde eine 21,4 % Steigerung der Klickrate gemessen. Insgesamt hat sich der NTG reduziert und eine Verbesserung in allen Domänen entwickelt.

AutoMTL Eine weitere Forschung beschäftigt sich mit der Reduzierung des negativen Transfers, um die Leistung aufgabenspezifischer Anpassungen zu verbessern. Dafür haben die Forscher S. Jiang et al. (2024) ihr Framework *AutoMTL* vorgestellt, das mithilfe von *Neural Architecture Search* (NAS) automatisch optimale Architekturen und „Expert-Sharing-Modi“ mit MMoE für MTL-Modelle entwirft. Das AutoMTL sucht in einem zwei-ebenen Suchraum, um zum einen die geteilte und zum anderen die spezifischen Expertennetzwerke auszuwählen. Damit können verschiedenen MoE-basierte Modelle kombiniert werden. Für die Suche wird der

Algorithmus *Progressively Discretizing Differentiable Architecture Search* (PD-DARTS) verwendet. Während der Suche wird das Supernetzwerk schrittweise diskretisiert, indem unwichtige Architekturkomponenten basierend auf Entropie-Metriken eliminiert werden, was die Effizienz steigert und die Notwendigkeit eines vollständigen Neutrainings vermeidet. Der Suchraum des AutoMTL umfasst auch Mechanismen zur Auswahl relevanter Attribute, um sie für die verschiedenen Experten zu optimieren. Die Evaluierung erfolgte auf fünf öffentlich zugänglichen Datensätzen: *UserBehavior-2017*, *IJCAI-2015*, *KuaiRand-Pure*, *QB-Video* und *AliCCP* mit unterschiedlichen MTL-Aufgaben und der AUC-Metrik. Diese decken verschiedene Szenarien ab, darunter Benutzerverhalten und Videobewertungen. AutoMTL übertrifft vergleichbare MTL-Modelle wie klassische MMoE oder AdaTT von D. Li et al. (2023) und PLE von Tang et al. (2020) in allen Datensätzen. Der flexible Suchprozess ermöglicht das Finden idealer MMoE-Netzwerke und benötigt ähnlich viel Rechenzeit wie das Training. Das hebt, laut Autoren, die Praxistauglichkeit hervor.

HTMN Die Generalisierungsfähigkeit in MTL-Modellen bei schwach korrelierenden Daten haben auch die Forscher J. Du et al. (2022) untersucht. Dafür stellen sie den neuen Ansatz *Hierarchical Task-aware Multi-Head Attention Network* (HTMN) vor, um globale (Aufgabenübergreifende) und lokale (aufgabenspezifische) Merkmale effizient zu extrahieren und adaptiv zu integrieren zu können. Es besteht aus zwei Komponenten. Das *Multi-Level Task-aware* Netzwerk extrahiert globale und lokale Merkmale durch spezialisierte MoE, die mit mehreren Gates pro Aufgabe gewichtet werden. Im Anschluss wird der Self-Attention Mechanismus im *Hierachischem Multi-Head Attention* Netzwerk angewendet, um lokale Merkmale für jede Aufgabe zu erfassen. Das Endergebnis wird, wie im MMoE, durch separate Aufgaben-Türme generiert. Für die Experimente wurden Einkommensdaten und Filmbewertungen verwendet. Es wurden insgesamt drei Gruppen mit schwach korrelierenden Vorhersage-Aufgaben verglichen – wie Einkommen und Familienstand (Pearson-Korrelationskoeffizient: 0,176). Das HTMN wurde mit zehn anderen Modellen und Metriken wie ROC-AUC, F1-Score, MSE und MAE verglichen. Das HTMN übertrifft die anderen Modelle in nahezu allen Aufgaben und Metriken, insbesondere bei den schwach korrelierten kombinierten Attributen. Zudem konvergiert das Modell schneller. Der hierarchische Attention-Mechnismus bietet Potenzial für zukünftige Forschungen in ressourcenschonenden MTL-Szenarien.

STGT Die Forscher Hu et al. (2024) haben eine kurzfristige Vorhersage von Ein- und Ausstiegspassagierströmen in städtischen Schienenverkehrssystemen mit MMoE vorhergesagt. Die Herausforderung liegt in der Modellierung der komplexen raumzeitlichen Abhängigkeiten und der intrinsischen Beziehungen zwischen Ein- und Ausströmen, um die Genauigkeit und Ro-

bustheit der Vorhersagen zu erhöhen. Für die Lösung des Problems wurde ein *Spatio-Temporal Graph Transformer* (STGT) Modell in ein MTL-Framework integriert. Der Graph-Transformer nutzt Attention-Mechanismen, um die raumzeitlichen Merkmale zu extrahieren. Mithilfe des MMoE wird die Beziehung zwischen Ein- und Ausströmen durch dynamische Gewichtung der Expertennetzwerke modelliert. Bei der Lösung werden auch externe Merkmale wie das Wetter, Zugfahrpläne oder die Zugänglichkeit von Busstationen berücksichtigt. Als Datensatz wurde ein realer Datensatz des Pekinger U-Bahn-Systems aus einem Jahr verwendet. Die Daten umfassen Ein- und Ausstiegszeiten, Stationsnamen sowie externe Faktoren wie Wetter und Busanbindung. Verglichen wurde der neue Ansatz mit traditionellen Methodiken für Zeitreihen wie *Autoregressive Integrated Moving Average* (ARIMA), mithilfe von maschinellem Lernen wie *Support Vector Machines* (SVM) und graph-basierte neuronale Netze wie *Graph Convolutional Network* (GCN). Das STGT-MMoE-Modell übertraf alle Vergleichsmodelle mit geringeren Fehlern. Der MAE wurde um 11,2 % reduziert, der RMSE um 10,7 %. Der R2-Wert liegt bei 0,88. Die Kombination von Multi-Task-Learning und den dynamischen Fähigkeiten von Graph Transformer und MMoE erwies sich als besonders effektiv. Zukünftige Forschung sollte sich auf Vorhersagen unter außergewöhnlichen Bedingungen (unter anderem extreme Wetterereignisse) konzentrieren.

GESME Eine weitere Arbeit, die sich mit MTL befasst, ist von Rahman et al. (2024). Das Paper befasst sich mit der Entwicklung eines Vorhersagemodells für eine Plattform, auf der Transport-Dienstleistungen in Echtzeit gebucht werden können. Das Ziel war die simultane Prognose für verschiedene Aufgaben wie Nachfrage- und Angebotslücken mit einer MTL-Architektur zu lösen, statt für jede einzelne Aufgabe und Stadt ein eigenes Modell zu trainieren. Dafür entwickelten die Forscher das *Gated Ensemble of Spatio-Temporal Mixture of Experts* (kurz GESME-Net). Es kombiniert durch Gating-Netzwerke dynamisch verschiedene MoE untereinander. Jedes MoE besitzt spezialisierte Experten-Netzwerke wie CNN, RNN oder Conv-RNN, um räumliche und zeitliche Abhängigkeiten besser zu berücksichtigen. Eine von den Aufgaben unabhängige Ebene passt die Gewichtung an, um wichtige Attribute für jede Aufgabe hervorzuheben. Es wurden insgesamt 25 Millionen Datensätze für die Experimente verwendet. Verglichen wurde die Leistung des GESME-Net mit Standard-Modellen wie XGBoost, *Gradient Boosting Machines* (GBM) oder Deep-Learning-Ansätzen wie GCN, die auch von Hu et al. (2024) verwendet wurden. Zusätzlich wurde die Bedeutung der einzelnen Komponenten und das Hyperparameter-Tuning untersucht. Das GESME-Net übertrifft alle Benchmarks in Bezug auf Fehlermetriken. Städte-übergreifend zeigten die Prognosen mit MTL-Architektur bessere Generalisierung. Der MoE mit CNN (Conv-MoE) trug am stärksten zur Modellleistung bei, in

dem räumliche Abhängigkeiten effektiv erfasst wurden. Daneben konnte eine Effizienzsteigerung erzielt werden, da ein Modell für mehrere Aufgaben Wartungs- und Berechnungskosten reduziert.

3.2.3. Adaptives Verhalten mit MoE

LLL Eine weitere Forschung, die sich mit dem katastrophalen Vergessen im Kontext von NLP beschäftigt, kommt von W. Chen et al. (2023). Das Pre-Training von LLM ist ein Ansatz, um starke allgemeine Sprachrepräsentation zu entwickeln. Jedoch entstehen Herausforderungen, wenn neue Datenverteilungen sequenziell, wie im Streaming, verarbeitet werden müssen. Das klassische *Fine-Tuning* von LLM führt oft zu katastrophalem Vergessen. Als Ansatz wählen die Autoren den *Lifelong Learning* (LLL)-Ansatz. Dabei wird die Anzahl der Experten im MoE progressiv erweitert, um neue Datenverteilungen zu modellieren. Die Destillation alter Modellausgaben und das Sperren alter Experten werden als Regularisierung verwendet, um altes Wissen vor dem Überschreiben zu schützen. Der Gating-Mechanismus wählt dann aus gesperrten und aktiven Experten eine geringe, aber relevante Anzahl aus. Es wurden drei Textdatensätze (A = Wikipedia und Webseiten; B = Internationale Inhalte; C = Informelle Dialoge aus sozialen Medien) verwendet, die sprachlich unterschiedlich sind und damit auch eine disjunkte Verteilung haben. Die Modelle werden sequenziell trainiert: $A \rightarrow B \rightarrow C$, um das Vergessen zu messen. In den Experimenten wurden die Modelle auf 21 NLP-Aufgaben getestet. Verglichen wurde der Lifelong-MoE-Ansatz mit traditionellen Methoden wie *Memory Replay*, bei dem bereits verwendete Informationen erneut zum Training verwendet werden, um dem katastrophalen Vergessen entgegenzuwirken. Die Forscher kamen zu den Ergebnissen, dass der Verlust von Wissen aus früheren Verteilungen stark reduziert wurde. Dabei erreicht das Lifelong-MoE-Modell vergleichbare Ergebnisse zu den Referenzergebnissen. Dieser Ansatz bietet eine Grundlage für Sprachmodelle, die in dynamisch ändernden Szenarien eingesetzt werden können.

BP-MoE In der Arbeit von C. Chen et al. (2024) geht es um eine adaptive Zuordnung von MoE-Experten basierend auf Verhalten. Das Paper behandelt die Herausforderung der temporalen Graphen, deren Knoten, Kanten und Attribute sich im Laufe der Zeit ändern und dabei unterschiedliche evolutionäre Präferenzen aufweisen. Ziel war die verbesserte Vorhersage von Knoten- und Kantenattributen. Das wollten die Forscher durch ein *Behavior Pattern-aware Mixture-of-Experts* (BP-MoE)-Modell erreichen. Dabei betrachtet das Framework, ähnlich wie in der Arbeit über MoVE, drei Perspektiven, die durch spezifische KNN abgebildet werden: langfristige, räumliche und kurzfristige Verhaltensmuster. Das Gating-Netzwerk aktiviert

adaptiv Experten, basierend auf den Verhaltensdaten. Neben der Haupttrainingsfunktion (Binary Cross-Entropy) werden zusätzliche *Importance Loss* und *Load Loss* eingeführt, um ein ausgewogenes Training der Experten zu gewährleisten. Verschiedene reale Datensätze aus dem Internet wie Wikipedia, Reddit oder LastFM wurden in den Experimenten genutzt, um etwa die Verbindung zwischen Knoten vorherzusagen oder die Klassifizierung von Knotenattributen, die aus den Datensätzen kommen. Das BP-MoE übertraf in der durchschnittlichen Genauigkeit (Precision) alle vergleichbaren Modelle in allen Datensätzen und Szenarien. Der kurzfristige Verhaltensexperte hatte den größten Einfluss auf die Modellleistung. Das BP-MoE zeigte auch eine stabile Leistung bei wenigen Trainingsdaten, was auf die adaptive Eigenschaft zurückzuführen ist.

FEAMoE Der Ansatz *Fair, Explainable and Adaptive Mixture of Experts* (FEAMoE) der Forscher Sharma, Henderson und Ghosh (2023) vereint drei Schlüsselmerkmale im Einsatz von MoE in sensiblen Umgebungen: Fairness, Erklärbarkeit und Anpassungsfähigkeit. Es sollen Fairnessverluste und Genauigkeitsverluste durch Konzeptdrifts minimiert und gleichzeitig schnelle und interpretierbare Erklärungen ermöglicht werden. Trainiert wird das FEAMoE mit einem Soft-MoE in einer inkrementellen OML-Umgebung. Dabei werden neue Experten hinzugefügt, um Drift entgegenzuwirken. Eingesetzt werden logistische Regressionsmodelle als Experten. Damit ist eine schnelle Trainings- und Inferenzzeit gegeben, was in einem Online-Learning Szenario entscheidend ist. Ein FNN mit Softmax-Aktivierung wählt die relevanten Experten aus. Die Fairness wird umgesetzt, indem drei gängige Fairness-Metriken in die Verlustfunktion der logistischen Regression integriert werden: demografische Parität, Gleichheit der Chancen und belastungsbasierte Fairness. Die Erklärbarkeit wird durch *Shapley-Werte* nach Shapley (1988) in den einzelnen linearen Expertenmodellen effizient errechnet. Die Experimente wurden mit drei Datensätzen aus Einkommen, Strafjustiz und Kreditvergabe trainiert, die demografische Daten und Labels enthalten, die zum Zweck der Fairness-Evaluierung genutzt werden. Der *Home Mortgage Disclosure Act* (HMDA) Datensatz hat zusätzlich noch ein Drift in den Daten. Die ausgeführten Experimente zeigten, dass die Fairness-Metriken durch die eingesetzte Verlustfunktion gegenüber klassischen MoE verbessert wurden. FEAMoE konnte sich dynamisch an den Drift anpassen und erzielte bessere Ergebnisse als neural-basierte Modelle in einer kontinuierlichen OML-Umgebung.

3.3. Identifizierung von Forschungslücken

Die Literaturrecherche zeigt, dass viele neue Verfahren für MoE entwickelt werden, um Anwendungsfälle des Offline-Lernens für Single- oder Multi-Task-Aufgaben zu lösen. Das Ziel einer Untersuchung von adaptivem Verhalten im MoE-Kontext konnte am wenigsten in den Veröffentlichungen festgestellt werden. Interessant ist der Ansatz *FEAMoE* von Sharma, Henderson und Ghosh (2023), die MoE in einer inkrementellen OML-Umgebung untersucht haben. Der Fokus lag bei der Arbeit auf Erklärbarkeit und Fairness. Demzufolge wurden einfache Algorithmen wie die logistische Regression gewählt.

Im Rahmen dieser Arbeit wird der Ansatz aufgegriffen und die Zielsetzung aus Abschnitt 1.2 bearbeitet. Die Zielsetzung umfasst eine Kombination von OML und MoE in einem Framework. Das Lösen eines *Dynamisches Optimierungs-Problem* wird damit möglich gemacht. Daher stellt sich die folgende Forschungsfrage RQ1:

RQ1. Wie kann ein MoE-Ansatz mit Online-Lern-Mechanismen für dynamische Umgebungen unter Verwendung von Streamingdaten umgesetzt werden?

Der Einsatz einer dynamischen Expertenmenge kam in allen drei Forschungen zu adaptiven MoE vor. Interessant ist, wie in der Arbeit von C. Chen et al. (2024), der Einsatz von aktiven und inaktiven Experten, die abhängig vom Verhalten ausgewählt werden. Eine ressourcensparende Auswahl von Experten ist für eine Echtzeit-Verarbeitung mit Streamingdaten notwendig. Als weitere Frage ergibt sich die Forschungsfrage RQ2:

RQ2. In welchem Maße können dynamisch hinzugefügte Experten, basierend auf Drifterkennung, die Anpassungsfähigkeit in dynamischen Umgebungen verbessern?

In dem Einsatz von MoE in progressiv wachsenden Datensätzen aus Datenströmen sind verschiedene Metriken interessant, die gegebenenfalls von offline Modellen abweichen. Die Evaluierung der Methode und die dafür notwendigen Definitionen der verschiedenen Metriken sind Teil dieser Arbeit und werden durch die Forschungsfrage RQ3 betrachtet:

RQ3. Welche Metriken sind erforderlich, um die Leistung von Mixture of Experts bei progressiv steigenden Streamingdaten sinnvoll zu bewerten?

Anhand der Kombination von OML und der MoE-Architektur sollte ein Verfahren entstehen, das mit Datenströmen analytisch arbeiten kann, die auch eine hohe Datenrate besitzen. Das Grundprinzip *Divide and Conquer* von MoE kann dazu beitragen, gezielt und ressourcensparend auf neue und veränderte Daten zu reagieren. Das ermöglicht den Einsatz von komplexeren Lernverfahren für inkrementelle Daten, die etwa aus IoT-Geräten stammen.

4. Methodik

In diesem Kapitel wird das Vorgehen beschrieben, das zur Beantwortung der Forschungsfragen herangeführt wird. Ziel ist es, die *MoE*-Architektur mit Online-Learning Mechanismen des *OML* zu einem inkrementellen MoE in Form eines Frameworks zu vereinen (RQ1). Zu diesem Zweck wird das Konzept einer neuen MoE-Architektur vorgestellt, die adaptiv in Streamingszenarien agieren kann (RQ2). Die neue Architektur wird durch verschiedene Experimente und passende Metriken evaluiert (RQ3), die in diesem Kapitel aufgeführt werden.

4.1. Mischung adaptiver Experten

Die grundlegende Architektur eines klassischen MoE besteht aus einem neuronalen Gate G und mehreren (verschiedenen) neuronalen Experten $E = \{E_1, E_2, ..., E_N | N \in \mathbb{N} \setminus \{0\}\}$. Die wichtigste Komponente eines MoE ist das Gate, dass für eine Eingabe $d = (x, y) \wedge d \in \mathcal{D}$ die besten Gewichte der Experten wählt, um die Vorhersagen zu dem besten Gesamtergebnis zu kombinieren. Bei einer Expertenanzahl von $|E| = 1$ entfällt das Gating und die Prognose kann direkt aus dem einzigen Expertenmodell, ohne Gewichtung, erzeugt werden $\hat{y} = E_1(x)$ (Jacobs et al., 1991). Das Gating-Problem kann für den Datensatz $\mathcal{D}$, mit n-Zeilen und m-Attributen, formal wie in Formel 4.1 zusammengefasst werden:

$$G(x) = \begin{cases} \mathbb{R}^{n \times m} \rightarrow \mathbb{N}^{|E|} & \text{, wenn } |E| > 1 \\ \mathbb{R}^{n \times m} \rightarrow 1 & \text{, wenn } |E| = 1 \end{cases} \tag{4.1}$$

Zu den klassischen BML-Algorithmen gibt es häufig (adaptive) OML-Alternativen, die im Streaming-Kontext inkrementell lernen. Beispielsweise ist der *Hoeffding Tree* ein inkrementelles Lernverfahren, dass für Klassifikation und Regression statt des *Decision Tree* eingesetzt werden kann (Domingos und Hulten, 2000). Andere Verfahren wie *Lineare Regression* oder *Logarithmische Regression* können mithilfe von SGD inkrementell trainiert werden (Bartz-Beielstein und Bartz, 2023, S. 12). Da OML-Verfahren mit Streamingdaten arbeiten können und in der Regel schneller als BML-Verfahren sind, eignen sich die Verfahren für die Umsetzung eines inkrementellen MoE.

Zunächst wird betrachtet, welche Kombinationen von Experten und Gating-Methoden möglich sind. Die Kombinationen sind in der Regel durch die Art des Skalenniveaus der Daten mit der Ausgabe des Gating-Modells und der Expertenmodelle beschränkt. In Tabelle 4.1 sind die möglichen Kombinationen gelistet. Die Kombinationen sind in vier Kategorien unterteilt: Regressor, Klassifikator, Neuronales Netz und Clustering. Die Kombinationen sind entweder direkt möglich oder durch eine Transformation der Daten, um sie als Gating zu verwenden, möglich. Einige Kombinationen sind gar nicht möglich, da die Lernart dies nicht zulässt.

Experte(n) \ Gate	Regressor	Klassifikator	Neuronales Netz	Clustering
Regressor	(✓)	(✓)	✓	–
Klassifikator	(✓)	✓	✓	–
Neuronales Netz	(✓)	✓	✓	–
Clustering	(✓)	✓	✓	✓

Tabelle 4.1.: Mögliche Kombination verschiedener ML-Lernarten für das Gating und Experten in MoE. ✓ = Möglich, (✓) = Mit Transformation möglich, – = Nicht möglich.

Wird ein Regressor als Gate herangezogen, besteht die Herausforderung, dass die vorliegenden Labels skalar sind und nicht direkt als Klassen interpretiert werden können. Eine Möglichkeit ist die Transformation der Labels in Klassen, um ein Klassifikationsproblem zu lösen. Eine andere Möglichkeit ist die manuelle Festlegung von Klassen, die auf den Features oder Labels basieren. Ein Klassifikator als Gate kann direkt verwendet werden, wenn die Anzahl der Klassen der Anzahl der Experten entspricht. Damit ergibt sich zeitgleich eine Einschränkung, da die Expertenanzahl an die (adaptive) Anzahl Klassen gebunden ist: $|E| = |K|$. Vor dem Training der Regressor-Experten müssen die nominalen oder ordinalen Klassen in skalare Werte transformiert werden. Das ist beispielsweise mit `LabelEncoding` oder `OneHotEncoding` und *Multi-Output-Regression* möglich. Ein neuronales Netz als Gate kann flexibel im Eingang und Ausgang sein, sodass die Anzahl der Eingangsneuronen den Attributen und die Anzahl der Ausgangsneuronen den Experten entspricht. Ein Clustering-Verfahren kann für eine beliebige Datenmenge eine vorgegebene Clustermenge zurückgeben, sodass die Anzahl der Cluster den Experten entspricht. Damit wäre dieser Algorithmus als Gating geeignet, da die Cluster durch den Silhouetten-Koeffizienten gut evaluiert und erweitert werden könnten. Das Clustering entspricht aber dem unüberwachten Lernen und beinhaltet keine Labels. Ein Training von Experten mit überwachtem Lernen ist damit nicht möglich. Der Einsatz von nicht neuronalen Gating-Verfahren ist möglich, jedoch nicht empfohlen, da zusätzlicher (manueller) Aufwand berücksichtigt werden muss und Einschränkungen vorliegen. Die höchste Flexibilität für das Gating, bietet das neuronale Netz, da hier die Struktur des Netzwerks beliebig verändert

werden kann. Für die Experten kann jede Lernart für ein passendes Problem verwendet werden. Angesichts dessen wird sich in dieser Arbeit auf die Kombination von neuronalen Gates und mit neuronalen und nicht neuronalen Experten für Regression und Klassifikation fokussiert.

4.1.1. Inkrementelle MoE-Architektur

In dieser Arbeit wird eine neuartige Architektur für *inkrementelle MoE* eingeführt. Ein genereller Aufbau der Architektur lässt sich aus der Abbildung 4.2 entnehmen. Vereint wird das Training und die Inferenz mit einzelnen Datensätzen, wie bei OML aus Abbildung 2.4b, mit dem Konzept des MoE aus Abbildung 2.9.

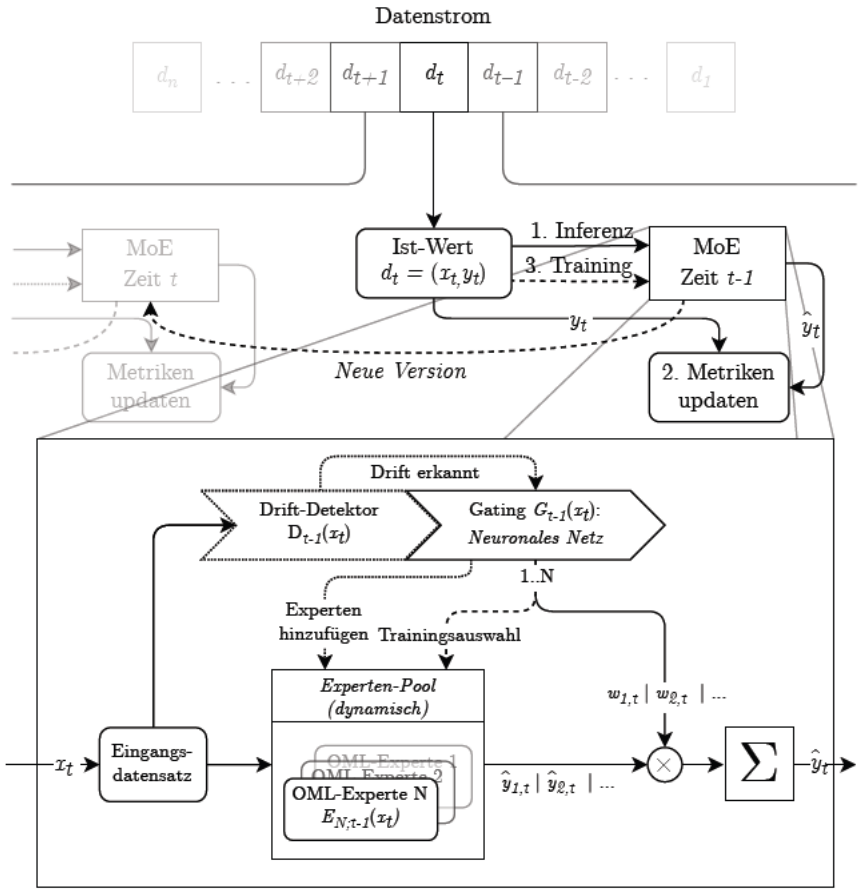


Abbildung 4.1.: Generelle Architektur des inkrementellen MoE. Mit Drift-Detektor bei *SAMoE*.

Die neue Architektur lernt ohne Epochen und mit nur einem Datensatz pro Zeitpunkt. Es handelt sich also um ein Batch-Lernen der Größe $n = 1$, das dem OML entspricht. In jedem Schritt wird, wie im OML, zunächst mit dem Datensatz eine (gewichtete) Inferenz berechnet

und zurückgegeben und damit die progressive Metrik aktualisiert. Danach wird das MoE mit einem Datensatz trainiert, was das Gating-Modell und die Experten sukzessiv aktualisiert. Ergänzend wird die neue *Streaming Adaptive Mixtures of Experts* (SAMoE) Variante eingeführt, die zusätzlich einen Drift-Detektor enthält, um neue Experten, wie in *FEAMoE* von Sharma, Henderson und Ghosh (2023), dynamisch hinzuzufügen. Wird durch Verfahren wie *ADWIN* in den Eingangsdaten ein Drift erkannt, wird ein Experte adaptiv zum *Experten-Pool* hinzugefügt.

Wie in einem MoE, gibt das Gating-Modell, für einen einzelnen eingegangenen Datensatz d_t aus dem Datenstrom, eine Wahrscheinlichkeitsverteilung als Gewichte zurück: $G(x_t) = w_{1,t}, w_{2,t}, ..., w_{n,t}$. Die Gewichte werden für die Experten $E = E_1, E_2, ..., E_n$ verwendet, um die Prognose zu einer skalaren Gesamtprognose zu kombinieren (Formel 4.2):

$$\hat{y}_t = \sum_{i=1}^{|E|} w_{i,t} \cdot E_i(x_t) \tag{4.2}$$

Es wird die neue *Streaming Adaptive Mixtures of Experts* (SAMoE) Variante eingeführt, die zusätzlich einen Drift-Detektor enthält, um bei Drift neue Experten dynamisch hinzuzufügen. Das Gating ist ein MLP, das flexibel seine Eingangsneuronen und Ausgangsneuronen an die inkrementellen Daten anpasst. Wie in Abbildung 4.2 dargestellt, entspricht die Anzahl der Eingangsneuronen der Anzahl der Attribute im Datensatz $x_t = \{x_{t,1}, x_{t,2}, ..., x_{t,m}\}$. Die Ausgangsneuronen entsprechen immer der Anzahl der Experten im MoE. Das adaptive Anpassen der Architektur ist damit möglich.

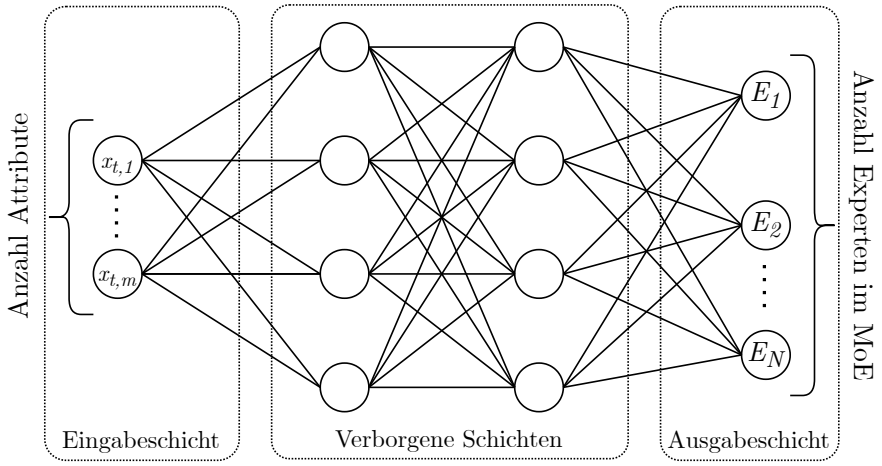


Abbildung 4.2.: Adaptives MLP mit variablen Eingangs- und Ausgangsneuronen als Gating.

Damit die initiale Auswahl der Experten nicht durch einen zufälligen Bias vorbestimmt wird, wird der Bias vor dem Training für das Gate auf 0 und die Gewichte auf eine Konstante $c \neq 0$

gesetzt. Wären alle Gewichte 0, könnte das Netzwerk nicht lernen. Diese Null-Initialisierung ist üblich in MoE (Z. Chen et al., 2024, S. 5). Das neuronale Netz des Gates wird, klassisch, mit Optimierungs- und Verlustfunktionen nach der Inferenz trainiert. Als Optimierungsfunktion kann beispielsweise SGD gewählt werden. Bei der Verlustfunktion $\mathcal{L}_{\text{Gate}}(\hat{y}_t, y_t)$ beispielsweise CE oder BCE für Klassifikation oder MSE für Regression. Das trainierte Gating-Modell wählt dann für eine Eingabe x_t die bestmöglichen Experten.

4.1.2. Auswahl des Basis-Frameworks

Derzeit gibt es viele Bibliotheken für den Einsatz von inkrementellem, maschinellem Lernen. Darunter sind MOA (*Massive Online Analysis*), implementiert in Java, oder RMOA, das in der Programmiersprache R umgesetzt ist. `river` ist ein Framework, das in Python implementiert ist. Von allen Frameworks ist es am breitesten aufgestellt und enthält aus jedem Bereich die wichtigsten Methoden für Anomalie, Regression, Klassifikation, Clustering und verstärkendes Lernen. Es enthält außerdem viele zusätzliche und aktuellere Methoden, welche in den R-Paketen nicht verfügbar sind (Bartz-Beielstein und Bartz, 2023, S. 93ff.). Aufgrund des breiten Funktionsumfangs und der gewählten Programmiersprache Python wird sich für `river` als Basis-Bibliothek entschieden.

`river` wurde von Montiel, Halford et al. (2021) umgesetzt und entstand aus gesammelten Erfahrungen der Pakete `creme` (Halford et al., 2020) und `scikit-multiflow` (Montiel, Read et al., 2018). In einem untersuchten Experiment konnte `river` seine zwei Vorgänger in Geschwindigkeit übertreffen. Die Modellgüte war erwartungsgemäß ähnlich gut. Das Framework ist auf dynamische Datenströme und kontinuierliches Lernen spezialisiert. Dabei wird ein Datensatz pro Zeitpunkt als *Schlüssel-Wert-Datenstruktur* (`dictonary` in Python) verarbeitet. Der auf Hash-Tabellen aufgebaute Datentyp hat eine effiziente Lese-, Anpassungs- und Löschkomplexität von $\mathcal{O}(1)$. Das Framework bietet mehrere hochmoderne Lernmethoden, Datengeneratoren/-transformatoren, Leistungsmetriken und Evaluationsmethoden an, die im Kern durch `Cython` (Behnel et al., 2011) umgesetzt sind. Dadurch ist eine effiziente Verarbeitung möglich.

Die in der Abbildung 2.4b der OML-Strategie vorgestellten Schritte *Vorhersagen*, *Evaluieren* und *Lernen* sind in der Bibliothek als einheitliche *Application Programming Interface* (API) umgesetzt, damit eine möglichst hohe Kompatibilität (beispielsweise bei einer Erweiterung) innerhalb der Bibliothek gewährleistet ist. Alle Vorhersagemodelle können mit `learn_one(x,y)` trainiert werden. Je nach Lernaufgabe liefern die Modelle Vorhersagen `y_pred` über die Methoden `predict_one(x)` (Klassifizierung, Regression und Clustering), `predict_proba_one(x)` (Wahrscheinlichkeitsverteilung der Klassifizierung) und

`score_one(x)` (Anomalieerkennung). Das Suffix „`*_one`“ zeigt an, dass es sich bei der Eingabe um eine einzelne Datenprobe handelt. Die Metriken für die Evaluation können mit `update(y, y_pred)` aktualisiert werden.

4.1.3. Umsetzung einer Framework-Erweiterung

Die in Abschnitt 4.1 vorgestellte Architektur ist eine adaptive und inkrementelle Ergänzung der MoE-Architektur. Wie in den Grundlagen bereits beschrieben, setzen die MoE üblicherweise neuronale Funktionen voraus, die durch die Basis-Bibliothek, bis auf ein einfaches MLP für Regressionen, nicht abgedeckt werden. Diese Funktionen werden in dieser Arbeit für das Gating und für neuronale Experten vorausgesetzt. Eine neuronale Erweiterung von `river` auf Basis von `PyTorch` bietet das Framework `deep-river` von Kulbach et al. (2025). Die neuronalen Modelle verwenden die gleiche API wie `river`, ergänzen den Umfang um den Einsatz von unterschiedlichsten NN-Architekturen wie FNN, RNN oder LSTM. Zusätzlich können Eingabedaten (Features) und Ausgabedaten (Labels) dynamisch angepasst werden.

Mithilfe von `deep-river` und der Basis-Bibliothek wurde das Framework `river`MoE entwickelt, das MoE für dynamische Datenströme und kontinuierliches Lernen umsetzt. Ziel war es dabei, die Strategie der zwei vorherigen Bibliotheken zu berücksichtigen. Das heißt, es wurde die gleiche Programmierschnittstelle verwendet, damit bereits umgesetzte Komponenten für Datensätze, Algorithmen und Evaluierungsmethoden hier wiederverwendet und gegebenenfalls erweitert werden können. Die *SAMoE*-Variante soll möglichst sparsam sein, weswegen sie auf Sparse-MoE, eine Erweiterung von Soft-MoE, mit *Top-K(*$k = 1$*)* aufbaut. Deswegen sind die Varianten Soft-MoE und Sparse-MoE notwendige Teile des `river`MoE-Prototyps und können verwendet werden, um ein *Dynamisches Optimierungs-Problem* zu lösen.

4.2. Experimente

In diesem Abschnitt werden die Experimente vorgestellt, die zur Beantwortung der Forschungsfragen durchgeführt werden. Zunächst werden die verwendeten Algorithmen und Datensätze vorgestellt. Zur Bewertung der Ergebnisse werden Evaluationsmethoden herangezogen, die hier näher erläutert werden. Anschließend wird die experimentelle Umgebung und die Experimente tabellarisch mit den gewählten Parametern aufgelistet.

4.2.1. Auswahl inkrementeller Algorithmen

Es gibt viele inkrementelle Lernverfahren, die bereits in `river` umgesetzt sind und in dieser Arbeit zum Einsatz kommen (Bartz-Beielstein und Bartz, 2023, S. 93ff.). Innerhalb dieser Arbeit

wird der Fokus nur auf die Problemstellungen Regression und Klassifikation gelegt. Eine Übersicht aller ausgewählten Algorithmen ist in Tabelle 4.2 dargestellt.

Aufgabe	Klasse in `river`/`deep-river`	Abkürzung
Regression	`HoeffingTreeRegressor`	HTR
	`HoeffingAdaptiveTreeRegressor`	HATR
	`LinearRegression`	LinR
	`DeepRegressor` (aus `deep-river`)	DeepR
	`StatisticalRegressor` (Durchschnitt)	BaseR
Klassifikation	`HoeffingTreeClassifier`	HTC
	`HoeffingAdaptiveTreeClassifier`	HATC
	`LogisticRegression`	LogR
	`DeepClassifier` (aus `deep-river`)	DeepC
	`NoChangeClassifier` (Letzte Klasse)	BaseC
Drift	`ADWIN`	ADWIN
	`KSWIN`	KSWIN

Tabelle 4.2.: Übersicht der eingesetzten OML-Algorithmen für diese Arbeit.

H(A)TR/H(A)TC Der *Hoeffding Tree* (HT) ist ein inkrementeller Entscheidungsbaum speziell für das Online-Lernen und die Verarbeitung von Datenströmen. Er basiert auf dem Hoeffding-Bound, der bestimmt, wie viele Datenpunkte erforderlich sind, um mit hoher Wahrscheinlichkeit die beste Attributaufteilung zu erkennen. Dies ermöglicht es dem Algorithmus, Entscheidungen auf Basis einer Teilmenge der Daten zu treffen, was bei großen oder kontinuierlichen Datenströmen entscheidend ist. Der *Hoeffding Adaptive Tree* (HAT) erweitert den Hoeffding Tree um Concept Drift-Erkennung. Bei Erkennung von Drift durch etwa ADWIN, wird eine neue Verzweigung hinzugefügt. In dynamischen Umgebungen, in denen sich die Datenverteilung im Laufe der Zeit ändert, ist ein statisches Modell wie der HT oft unzureichend.

LinR Die Lineare Regression ist ein inkrementelles Lernmodell zur Vorhersage kontinuierlicher Zielwerte. Sie basiert auf der Annahme einer linearen Beziehung zwischen den Eingangsmerkmalen und dem Zielwert. Das Modell berechnet Vorhersagen, indem es die Merkmale mit gelernten Gewichten multipliziert und summiert. Diese Gewichte werden, in der inkrementellen Variante, schrittweise mit jedem eingehenden Datenpunkt aktualisiert, häufig mithilfe von SGD. Bei nicht linearen Beziehungen oder starken Ausreißern kann die Leistung der linearen Regression jedoch beeinträchtigt werden.

LogR Die Logistische Regression ist ein Klassifikationsmodell, das Wahrscheinlichkeiten für Klassen vorhersagt. Sie eignet sich sowohl für binäre als auch für multi-klassige Probleme. Das Modell nutzt eine lineare Kombination der Eingangsmerkmale, deren Ergebnis durch die Sigmoid-Funktion (bei binärer Klassifikation) oder Softmax-Funktion (bei mehreren Klassen) in Wahrscheinlichkeiten umgewandelt wird. Wie die lineare Regression wird auch die logistische Regression, in der inkrementellen Variante, mit Methoden wie SGD schrittweise aktualisiert, was sie für Datenströme prädestiniert.

BaseR/BaseC Die Baseline-Modelle werden als Referenzpunkte in der Modellbewertung angewendet. Sie bieten eine einfache, schnelle und oft naive Lösung für ein Vorhersageproblem. Die anderen leistungsfähigeren Modelle sollten diese Baseline deutlich übertreffen. Gelingt dies nicht, ist der Einsatz eines aufwendigeren Verfahrens nicht gerechtfertigt. Für die Regression wird der `StatisticalRegressor` mit der Aggregtation „Durchschnitt“ verwendet, der den Durchschnitt der Zielwerte berechnet (BaseR). Für die Klassifikation wird der `NoChangeClassifier` eingesetzt, der immer die letzte Klasse vorhersagt (BaseC).

Drift Die Drifterkennungs-Algorithmen kommen zum einen in der adaptiven Version des HAT vor, werden aber auch für den Drift-Detektor in der *SAMoE*-Variante verwendet. Für diese Ausarbeitung wird das bereits vorgestellte *Adaptive Windowing* (ADWIN) genutzt. Das Verfahren ist in der Lage, Drifts in Datenströmen zu erkennen.

DeepR/DeepC Mit der Bibliothek `deep-river` können Regressionen, Klassifikationen und Anomalieerkennungen unter Einsatz von neuronalen Netzen mit Streamingdaten durchgeführt werden. In die Wrapper können nahezu beliebige neuronale Netzwerkarchitekturen als Pytorch-Module eingefügt werden. Die neuronalen Modelle sind in der Lage, komplexe, nicht lineare Beziehungen in den Daten zu modellieren und können durch die Anpassung der Ein- und Ausgangsneuronen an verschiedene Probleme zur Laufzeit angepasst werden. Der Nachteil ist, dass die Modelle aufgrund der hohen Anzahl an Parametern und der komplexen Struktur mehr Rechenleistung und Speicherplatz benötigen. Initiiert werden die neuronalen Netze mit einem Zufallsstartwert.

Alle ausgewählten Lernverfahren haben den Vorteil, dass sie in adaptiven Umgebungen funktionieren und dabei, je nach Einstellung, schnell sind und mit wenig Speicherplatz auskommen. Zudem haben nicht neuronale Algorithmen eine gute Interpretierbarkeit. Für HAT ergibt sich noch der Vorteil, dass er auf Drift reagieren kann und damit Konzeptdrift entgegenwirkt.

4.2.2. Eingesetzte Datensätze

Der Fokus dieser Arbeit liegt in der Evaluierung des umgesetzten inkrementellen MoE. Im Rahmen dieser Grundlagenforschung wird bei den Datensätzen kein spezifisches Domänenproblem festgelegt. Die unterschiedliche Auswahl der Datensätze soll den vielseitigen Einsatz von `riverMoE` verdeutlichen. Das Training erfolgte ausschließlich mit metrischen Attributen, die vorab durch *Standardisierung* vereinheitlicht wurden.

Bikes Der *Bikes*-Sharing-Datensatz ist im Rahmen eines privaten Projektes „OpenBikes“ von Halford (2016) entstanden. Dieser reale Datensatz beinhaltet neben dem Datum und der Uhrzeit noch sieben weitere Attribute sowie die Anzahl ausgeliehener Fahrräder für fünf verschiedene Fahrradstationen in Toulouse, Frankreich. Es gibt 182470 Einträge in dem Datensatz. Die Zielvariable ist die Anzahl der Fahrräder, die an einem bestimmten Tag und einer bestimmten Uhrzeit ausgeliehen werden. In dieser Arbeit werden nur die numerischen Wetterattribute *Temperatur* in Grad, *Luftfeuchtigkeit* in Prozent, *Wolkenbedeckung* in Prozent, *Luftdruck* in hPa und *Windgeschwindigkeit* in $\frac{m}{s}$ verwendet, um die Anzahl der ausgeliehenen Fahrräder vorherzusagen. Der Datensatz wird in dieser Arbeit verwendet, um die Leistung der Modelle bei einem Regressionsproblem zu testen.

Elec2 Der *Elec2*-Datensatz stammt von Harries, Wales et al. (1999) und wurde bereits in der Veröffentlichung von Montiel, Halford et al. (2021) für `river` zur Evaluierung verwendet. Diese Daten wurden alle 30 Minuten aus dem australischen Strommarkt von „New South Wales“ (NSW) zwischen 1996 und 1998 erhoben. In der Abbildung 4.3 ist der durchschnittliche Strompreis pro Tag dargestellt. Auf diesem Markt waren die Preise nicht festgeschrieben und werden von Angebot und Nachfrage auf dem Markt beeinflusst. Am 4. Mai 1997 wurde der Strommarkt im benachbarten Bundesstaat „Victoria“ mit dem NSW verbunden, dadurch kamen weitere Anbieter und Preise auf den Markt, die ebenfalls mit erhoben wurden und Einfluss auf den NWS-Preis hatten. Der Ausreißer aus der Abbildung 4.3, von November 1997, lässt sich auf eine unterbrochene Stromverbindung zwischen NWS/Victoria und damit verbundenes geringeres Angebot zurückführen.

Die Daten enthalten 8 numerische Attribute und 45312 Zeilen. Die Zielvariable ist die Klassifizierung von Elektrizitätsverbrauchsmustern in 2 Klassen („Preis aufwärts“ TRUE oder „Preis abwärts“ FALSE). Die Attribute enthalten eine Reihe von tatsächlichen Nachfragezahlen. Dies ist problematisch, da die tatsächliche Nachfrage nicht direkt vor der Zeit zur Verfügung stehen würde. Die Attribute werden von den Autoren als Projektion für die Attributnachfrage verwendet.

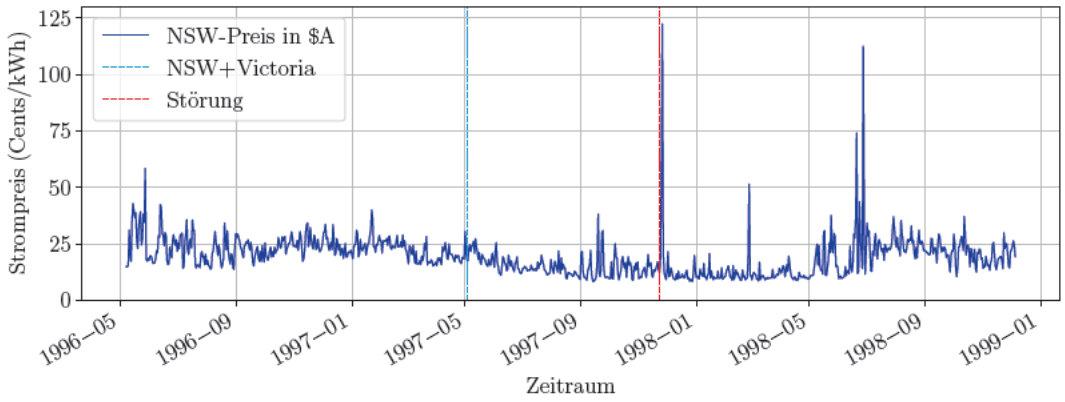
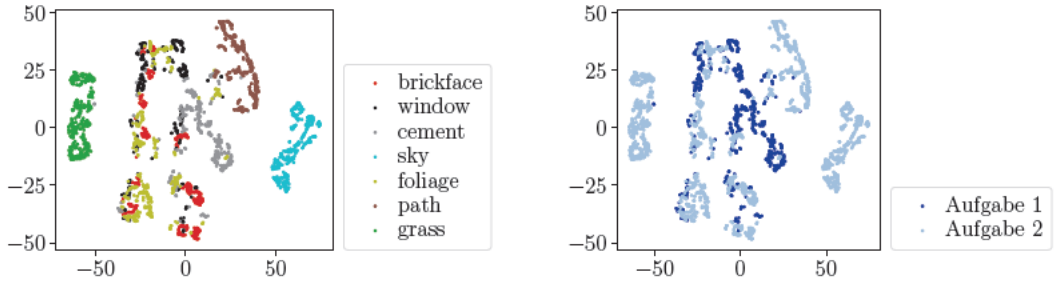


Abbildung 4.3.: Durchschnittlicher Strompreis (Cents/kWh) pro Tag im Elec2-Datensatz.

Image Segmentation Der *Image Segmentation*-Datensatz stammt von dem *UCI Machine Learning Repository* aus dem Jahr 1990. Dabei wurden sieben Bildsegementierungen aus verschiedenen Bildern extrahiert. Der Datensatz enthält 2310 Zeilen und 19 Spalten. Die Zielvariable ist die Klassifizierung von sieben Bildsegmenten nach Positionsattributen: „Mauerwerk“, „Zement“, „Fenster“, „Himmel“, „Laub“, „Weg“ und „Gras“. Mit Hilfe eines *t-SNE*-Plot wurden die 19 Attribute auf zwei Stück reduziert und in einem Streudiagramm in Abbildung 4.4a dargestellt.



(a) Für sieben Klassen.

(b) Umwandlung in zwei Aufgaben.

Abbildung 4.4.: *t-SNE*-Plot der Bildsegmentierungsdaten.

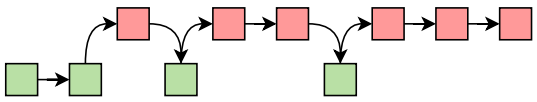
Die Verteilung der Klassen wird durch diese Darstellung deutlich. Der Datensatz wird in dieser Arbeit verwendet, um die Leistung der Modelle bei einem inkrementellen Drift, einem Zielwechsel zur Laufzeit, zu testen. Zu diesem Zweck werden die sieben Klassen in zwei unterschiedliche Aufgaben aufgeteilt. Die Erkennung der Klassen „Mauerwerk“, „Fenster“ und

„Zement“ gehört zur *Aufgabe 1*, die restlichen Klassen zur *Aufgabe 2*. In Abbildung 4.4b ist die Verteilung für die beiden Aufgaben dargestellt.

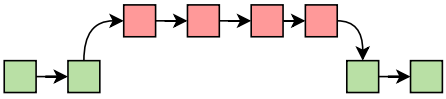
Friedman-Drift Der *Friedman-Drift*-Datensatz ist ein synthetischer Datensatz, der von Friedman (1991) erstellt wurde. Er besteht aus zehn numerischen Attributen. Bei jeder Beobachtung werden die zehn Ziffern zufällig aus einer Gleichverteilung gewählt: $x_j \sim \mathcal{U}(0, 1)$. Die Zielvariable ist eine lineare Kombination der relevanten Attribute und wird in `river` wie in Formel 4.3 ermittelt:

$$f(x) = 10\sin(\pi x_0 x_1) + 20(x_2 - \frac{1}{2})^2 + 10x_3 + 5x_4 + \epsilon \text{ mit } \epsilon \sim \mathcal{N}(\mu = 0, \sigma^2 = 1) \quad (4.3)$$

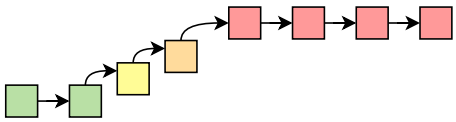
Nur die ersten fünf Merkmale $x_0, x_1, ..., x_4$ sind relevant für die Zielvariable. Das ϵ ist ein normalverteilter Fehlerterm mit einem Mittelwert von 0 und einer Standardabweichung von 1. Ein Wechsel der aktiven Variablen führt zu einer Veränderung, wodurch ein Konzeptdrift umgesetzt wird (Bartz-Beielstein und Bartz, 2023, S. 6ff.). Es gibt drei verschiedene Driftarten. Die verschiedenen Driftarten, die für diesen Datensatz in `river` implementiert sind, sind in Abbildung 4.5 dargestellt (Gulcan und Can, 2023). Ein Drift unterscheidet sich von einem Ausreißer, wie bei der Stromstörung im Elec2-Datensatz aus Abbildung 4.3.



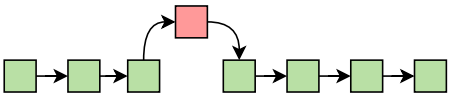
(a) Local Expanding Abrupt (LEA)



(b) Global Recurring Abrupt (GRA)



(a) Global and Slow Gradual (GSG)



(d) Ausreißer

Abbildung 4.5.: Verschiedene Driftarten (a), (b) und (c) im Vergleich zu Ausreißern (d).

LEA beschreibt einen zunächst lokalen Konzeptwechsel, der sich ausdehnt und dann zum Ursprung zurückkehrt. GRA bewirkt eine abrupte, globale Veränderung mit späterer Rückkehr. GSG hingegen entwickelt sich schrittweise und stabilisiert sich über die Zeit. Für diese Arbeit werden 12500 synthetische Datensätze mit Zufallsstartwert 42 aus dem Friedman-Datensatz mit dem Drift *GRA* (Abbildung 4.5b) an den Stellen 5000 und 7500 erzeugt. Der Datensatz wird in dieser Arbeit verwendet, um die Leistung bei einem Konzeptdrift zu testen.

4.2.3. Experimentelle Umgebung

Alle Experimente werden auf einem MacBook Pro in der *Jupyter Notebook*-Umgebung durchgeführt. Die genauen Spezifikationen des genutzten Gerätes werden in Tabelle 4.3 beschrieben.

Betriebssystem		**Prozessor**				**Arbeitsspeicher**
Name	Version	Modell	Architektur	Kerne	Threads	RAM
Darwin	24.3.0	Apple M1 Pro	arm64	10	10	32 GB

Tabelle 4.3.: Verwendete Hardware für die Experimente.

Es wird die Programmiersprache *Python* in der Version 3.11.7 genutzt. Die `river` Bibliothek wird in der Version 0.22.0 und die Erweiterung `deep-river` in Version 0.2.8 verwendet. Das implementierte `river`MoE-Framework wurde als Prototyp in der Version 0.1.0 für die Experimente evaluiert. Die Experimente werden auf der CPU ausgeführt. Alle Algorithmen in den Experimenten werden mit einem festen Zufallsstartwert von 42 verwendet. Damit sind die Experimente reproduzierbar und die Ergebnisse vergleichbar.

4.2.4. Übersicht der Experimente

Die Experimente sind in drei Gruppen unterteilt: *Regression* (R), *Klassifikation* (C) und *Drift* (D), die in diesem Abschnitt kurz vorgestellt werden. Es werden für die Experimente die gleichen Parameter und NN-Architekturen für Modelle genommen, damit die Ergebnisse vergleichbar sind. Zur Minimierung der Rechenzeit und des Speicherbedarfs kommen einfache neuronale Architekturen zum Einsatz, die in Tabelle 4.4 dargestellt werden.

Lernart	Typ	Architektur	$\mathcal{L}$	Optim.	Lernrate
Regression	Gating	m-ReLU-10-ReLU-$\lvert E\rvert$	MSE	SGD	0,001
	DeepR	m-ReLU-10-ReLU-1	MSE	SGD	0,001
Klassifikation	Gating	m-ReLU-10-ReLU-$\lvert E\rvert$	CE	SGD	0,01
	DeepC	m-ReLU-10-ReLU-$\lvert y\rvert$-Softmax	CE	SGD	0,01

Tabelle 4.4.: Verwendete adaptive NN-Architekturen für verschiedene neuronale Lernarten.

Die Eingangsneuronen sind von der Anzahl der Attribute m des jeweiligen Datensatzes abhängig. Für das Gating bestimmt die Anzahl der Experten $|E|$ die Anzahl der Ausgangsneuronen. Das Softmax im Gating wird durch das MoE implizit umgesetzt. Für die Regression ist die Anzahl der Ausgangsneuronen immer 1, für die Klassifikation entspricht die Anzahl der Ausgangsneuronen der Anzahl der Klassen $|y|$. Die *ReLU*-Funktion $\sigma(z) = \max(0, z)$ lässt

sich effizient berechnen und unterstützt bei der Generalisierbarkeit des NN, da Neuronen sparsamer aktiviert werden. Als Parameter für die neuronalen Modelle werden die bereits vorgestellten Verlustfunktionen und Optimierungsfunktionen verwendet. SGD wird vor allem für inkrementelles Lernen eingesetzt, da es mit einer kleinen Anzahl von Beispielen gut konvergiert. Für die Regression wird eine kleinere Lernrate verwendet als für Klassifikation, da zu hohe quadratische Fehler des MSE zur Instabilität führen können. Auch wenn das Gate der Regression ein Klassifikationsproblem ist, wird dennoch der MSE als Verlustfunktion genommen, da der tatsächliche Wert und die vorhersagten Werte der Experten skalar sind.

Typ	Attribut	Experiment R.1	Experiment R.2
Allg.	Datensatz	Bikes	Bikes
	Modelle	LinR, HTR, DeepR, MoE, BaseR	Top-K($k \in [1, 2, 3]$)
MoE	Anzahl Experten	3	3
	MoE OML-Experten	LinR, HTR, BaseR	LinR, HTR, BaseR
	MoE Strategie	Soft-MoE	Sparse-MoE
	Drifterkennung	–	–

Tabelle 4.5.: Übersicht der Experimente der Gruppe Regression (R).

In der Gruppe der Regression werden zwei Experimente betrachtet, die in Tabelle 4.5 aufgelistet werden. Im ersten Experiment R.1 werden die Modelle `LinR`, HTR, `DeepR`, `SoftMoE` und `BaseR` auf dem Bikes-Datensatz miteinander verglichen. Das zweite Experiment R.2 fokussiert sich auf das sparsame MoE „Sparse-MoE“ und untersucht die Auswirkung der Top-K-Strategie auf die Anzahl der drei Experten. Die MoE OML-Experten sind in beiden Experimenten `LinR`, HTR und `BaseR`. Die Drifterkennung wird in beiden Experimenten nicht verwendet.

Typ	Attribut	Experiment C.1	Experiment C.2
Allg.	Datensatz	Elec2	Elec2
	Modelle	LogR, HTC, DeepC, MoE, BaseC	Top-K($k \in [1, 2, 3]$)
MoE	Anzahl Experten	3	3
	MoE OML-Experten	LogR, HTC, BaseC	LogR, HTC, BaseC
	MoE Strategie	Soft-MoE	Sparse-MoE
	Drifterkennung	–	–

Tabelle 4.6.: Übersicht der Experimente der Gruppe Klassifikation (C).

In der Gruppe der Klassifikation, aus Tabelle 4.6, werden ebenfalls zwei Experimente betrachtet. Im ersten Experiment C.1 werden die Modelle `LogR`, HTC, `DeepC`, `SoftMoE` und

BaseC auf dem Elec2-Datensatz trainiert und verglichen. Das zweite Experiment C.2 fokussiert sich, analog zur Regression, auf das Sparse-MoE-Modell und untersucht die Auswirkung des gewählten k der drei Experten. Die MoE OML-Experten sind in beiden Experimenten LogR, HTC und BaseC. Die Drifterkennung wird in beiden Experimenten nicht verwendet.

Typ	Attribut	Experiment D.1	Experiment D.2
Allg.	Datensatz	Friedman	ImageSegmention
	Driftstrategie	GRA (Feature-Drift)	Label-Shift
	Modelle	DeepR, HATR(ADWIN), MoE	DeepC, HATC, MoE
	Problemklasse	Regression	Multi-Klassifikation
MoE	Anzahl Experten	3	1 (initial)
	MoE OML-Experten	LinR, HTR, BaseR	HTC
	MoE Strategie	Soft-MoE	SAMoE
	Drifterkennung	–	ADWIN

Tabelle 4.7.: Übersicht der Experimente der Gruppe Drift (D).

In der Drift-Gruppe, aus Tabelle 4.7, werden zwei Experimente betrachtet, bei denen die Drifterkennung der neuen inkrementellen MoE-Architektur evaluiert wird. In beiden Experimenten wird ADWIN mit einer Konfidenzgrenze von $\delta = 0{,}001$ verwendet. Im ersten Experiment D.1 werden die Modelle DeepR, HATR und SoftMoE auf der Friedman-Aufgabenstellung verglichen. Dabei soll untersucht werden, inwiefern der Drift bei dynamischem Gating von Nicht-Drift-Experten, gegenüber Algorithmen mit Drift-Verfahren oder neuronalen Netzen die Leistung über die Laufzeit beeinflusst. Im zweiten Drift-Experiment D.2 wird die SAMoE-Variante evaluiert. Dabei ändert sich die Multi-Klassifikationsaufgabe zur Laufzeit – im Testdatensatz wird auf beide Aufgaben geprüft. Initial wird in dieser Variante mit einem HTC-Experten gestartet. Bei der Drifterkennung sollte ein neuer Experte hinzugefügt werden.

Zur Begrenzung der Komplexität und Größe der HT- und HAT-Bäume wird die maximale Tiefe `max_depth` auf 3 gesetzt. Alle anderen Verfahren werden mit ihren Standardparametern verwendet. Eine Übersicht aller gesetzten Parameter für Klassifikation ist im Anhang in Tabelle A.1 und für Regression in Tabelle A.2.

4.2.5. Evaluationsmethoden

Zur Bewertung der Ergebnisse der durchgeführten Experimente werden verschiedene Metriken und Evaluationsmethoden herangezogen. Die verwendeten Metriken sind in der Bibliothek `river` implementiert und können direkt für die Evaluierung verwendet werden. Zunächst

einmal werden die bereits vorgestellten Modellmetriken herangezogen, um die Modellgüte zu bewerten.

Modellgüte Für die Regression wird der *MAE*, *RMSE* und *R2*-Wert, wie in Tabelle 2.1 beschrieben, verwendet. Der MAE lässt sich leicht interpretieren und ist unempfindlich gegenüber Ausreißern. Da zusätzlich auch die Bewertung größerer Fehler berücksichtigt werden soll, wird ebenfalls der RMSE herangezogen. Der R2-Wert wird berechnet, um die Varianz der Zielvariablen zu erklären. Dies ermöglicht eine intuitive Gesamtbewertung der Modellgüte.

Für die Klassifikation wird eine Auswahl an Metriken der Tabelle 2.5b verwendet. Dazu zählt die *Accuracy*, die *Precision*, der *Recall*. Die Accuracy ist eine häufig eingesetzte Gesamteinschätzung der Modellleistung, ist aber problematisch bei unausgeglichenen Klassen. Daher wird auch die Precision und der Recall herangezogen, um die Leistung des Modells besser zu bewerten. Der *F1-Score* ist das harmonische Mittel aus Precision und Recall und spiegelt eine zweite Gesamtbewertung der Modellleistung wider. Bei der Multi-Klassifizierung des Datensatzes *Image Segmentation* wird die *Micro*-Betrachtung der Metriken herangezogen, um die Gesamtleistung des Modells zu bewerten. Beim Mikro-Durchschnitt werden die Metriken über alle Klassen hinweg aggregiert, bevor die Berechnungen durchgeführt werden.

Die genannten Metriken werden häufig in der Literatur eingesetzt und werden für diese Arbeit gewählt, damit die Ergebnisse mit anderen Arbeiten vergleichbar sind. Alle Metriken werden als *Progressive Validierung* ohne Delay ($d = 0$) berechnet, wie bereits in den Grundlagen durch Abbildung 2.6a vorgestellt. Das heißt, es wird zuerst die Vorhersage berechnet, dann evaluiert und zum Schluss das Modell aktualisiert. Die Metriken werden in jeder Iteration als kumulierter gleitender Durchschnitt, wie in Formel 2.5 berechnet. Die Gewichtung wird mit $w_t = 1$ ausgeklammert und entspricht dem *Cumulative Moving Average* (CMA). Damit ist ein Vergleich der Leistung von verschiedenen OML-Methoden mit dem Framework `riverMoE` möglich, die in Liniendiagrammen über den zeitlichen Verlauf dargestellt werden.

Hardware Der Einsatz von OML-Algorithmen ist besonders in Streamingszenarien interessant. Eine verhältnismäßig schnelle Rechenzeit und möglichst geringer Speicherplatzverbrauch sind dabei entscheidend. Deswegen wird durch eine externe Evaluierungsmethode in jeder Iteration die *kumulierte Rechenzeit* und der verwendete *Speicherplatz* gemessen und als Liniendiagramm ausgegeben. Die Division der gesamten Laufzeit durch die Anzahl der Instanzen ergibt die durchschnittliche Laufzeit pro Instanz („Zeit/Instanz“) und wird als *Trainings- und Inferenzzeit* angegeben. Die Messung für die Rechenzeit in der Einheit Sekunden (s) und Mebibyte (MiB) für den Speicherbedarf. Die Trainings- und Inferenzzeit pro Datensatz wird in

Millisekunden (ms) angegeben. Bei MoE wird noch die jeweilige Speichergröße der Experten tabellarisch transparent dargestellt.

MoE-Evaluation Für die Evaluierung der MoE-Komponenten bedarf es spezieller Metriken für Gating und Expertenauswahl. Bei jeder Trainingsinstanz kann das vom Gate berechnete Gewicht gemessen werden. Zunächst wird erhoben, wie häufig ein Experte $j \in E$ nach den N-Durchläufen absolut ausgewählt wurde. Dafür wird die Indikatorfunktion $\mathbf{1}$ verwendet, um alle Experten, deren Gewichte w in einem Durchlauf größer als 0 sind, zu zählen. Mithilfe der absoluten Häufigkeit wird die relative Häufigkeit pro Experte j in Formel 4.4 ermittelt. Die individuellen Beiträge, die die Experten zu den Vorhersagen beitragen, werden in Formel 4.5 als Durchschnitt zusammengefasst.

$$\text{Rel. Häufigkeit}(j) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{w_{j,i}>0\}} \quad (4.4) \qquad \text{Beitrag}(j) = \frac{1}{N} \sum_{i=1}^{N} |w_{j,i}| \quad (4.5)$$

Beide Metriken sind Prozentwerte. Die Entropie der Gate-Gewichte für eine Instanz x_t wird verwendet, um die Spezialisierung der Experten eines Durchlaufs zu bewerten. Damit die Entropie-Werte vergleichbar sind, wird der Kehrwert der maximalen Entropie $\log(|E|)$ als Skalierungsfaktor verwendet, um die Werte zwischen 0 und 1 in der Formel 4.6 zu normieren.

$$\text{Normierte Entropie}(x_t) = -\frac{1}{\log(|E|)} \cdot \sum_{i=1}^{|E|} w_i \log(w_i) \in [0, 1] \quad (4.6)$$

Die Spezialisierung ist ein Maß für die Diversität der Experten und ähnelt der Evaluationsmethodik *Router Dispatch Entropy* von Z. Chen et al. (2024, S. 9). Je höher der Wert ist, desto gleichverteilter sind die Gewichte und vice versa. Bei einem Wert von 1 sind die Gewichte perfekt gleichverteilt, bei 0 ist nur ein Experte aktiv.

Alle Metriken werden für ein Modell berechnet und pro Durchlauf gemessen und mit den anderen Modellen verglichen. Dabei wird die Genauigkeit der Metriken auf drei Nachkommastellen gesetzt. Generell gilt, dass die Reihenfolge der Daten, also die Eintreffzeit der Dateninstanzen aus dem Datenstrom, einen Einfluss auf die Ergebnisse hat. Es handelt sich daher um eine *Prequential Validation*. Daher ist es üblich für die Gesamtbeurteilung, die Werte der letzten Iteration zu betrachten. Dies entspricht im produktiven Szenario dann der Leistung des Modells für neue eintreffende Dateninstanzen. Eine ausführliche Übersicht der Zwischenwerte mit gleichen Abständen wird im Anhang A.4 in Tabellen festgehalten.

5. Ergebnisse und Diskussion

Bevor die Ergebnisse der Experimente präsentiert werden, wird zunächst die prototypische Implementierung des `river`MoE-Frameworks vorgestellt. Dabei wird die entwickelte Architektur erklärt. Dies beantwortet die erste Forschungsfrage, ob ein Framework zur Generierung von MoE-Systemen für dynamische Umgebungen möglich ist (RQ1). Im zweiten Abschnitt dieses Kapitels werden die Evaluationsergebnisse der einzelnen Experimente vorgestellt und diskutiert. Dabei wird die zweite und dritte Forschungsfrage betrachtet, wie die Leistung gemessen werden kann (RQ3) und ob sich die Leistung in dynamischen Umgebungen durch die Architektur steigert (RQ2).

5.1. Prototypische Implementierung

Die Idee von $river_{\text{MoE}}$ wurde als Framework `river`MoE[1] in Python umgesetzt und auf Github veröffentlicht. Es dient als Erweiterung zu dem Basis-Framework `river`. Das Framework ermöglicht die Generierung von MoE-Systemen für dynamische Umgebungen. Dabei können verschiedene MoE-Varianten umgesetzt werden, die auf unterschiedlichen Gating- und Experten-Modellen basieren. Die Funktionsweise und der Aufbau der Software werden in den nächsten Abschnitten näher erläutert.

5.1.1. Framework-Architektur

Für die prototypische Umsetzung wurden 15 Python-Klassen implementiert, die in der Abbildung A.1 als Klassendiagramm im Anhang aufgelistet werden. Damit die grundlegende API aus dem Basis-Framework `river` und `deep-river` wiederverwendet werden kann, wurde primär mit Vererbungen von deren Klassen gearbeitet.

Basis MoE Zunächst wurde eine Basis-MoE Klasse `BaseMixtureOfExperts` erstellt, die die grundlegenden Funktionen von dem Standard-`Estimator` aus `river` erbt. Damit

[1] Github: https://github.com/bitnulleins/rivermoe

besitzt das MoE die gleichen Eigenschaften und Methoden wie alle Schätzer der `river`-Bibliothek. Jede MoE-Variante hat zusätzlich die Attribute und Methoden, wie sie in Abbildung 5.1 beschrieben sind.

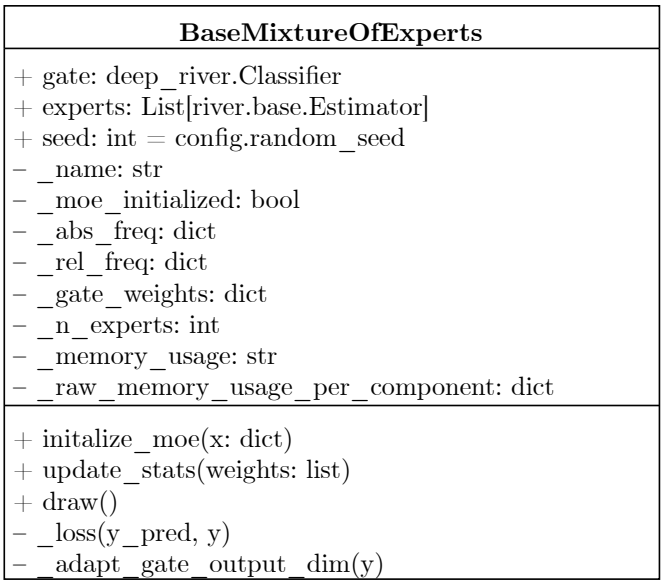


Abbildung 5.1.: Basis-Klasse als Ausschnitt aus dem Klassendiagramm zu `riverMoE`

Jede MoE-Art besteht aus einem Gate-Objekt und einem oder mehreren Experten-Objekten, die bei der Initialisierung des Objektes mitgegeben werden. Die `Classifier`-Klasse aus `deep-river` wird als Schätzer für das Gate erwartet. Bei den Experten kann es ein beliebiger Schätzer sein. Beides wird bei der Initialisierung geprüft, um die Funktionalität sicherzustellen. Der Zufallsstartwert des MoE wird auf 42 als Standardwert gesetzt. Alle anderen Attribute sind privat und werden intern verwendet. Die Anzahl der Experten kann aus `_n_experts` entnommen werden, das dynamisch durch einen Eigenschaftsdekorator erzeugt wird. Für das Gate wird ein neuronales Netz erwartet, wie bereits durch den Methodikteil der Tabelle 4.1 dieser Arbeit begründet. Die Experten hingegen können beliebige OML-Verfahren sein. Das Namen-Attribut soll bei der Identifizierung der MoE-Art unterstützen und wird im Standard durch den Klassennamen gefüllt. Diese Methode kann auch durch parameterabhängige Namen, wie k bei einem Sparse-MoE, überschrieben werden. Der Name wird dann unter dem Gate als MoE-Strategie in der Architekturskizze angezeigt, wenn die `draw()`-Methode aufgerufen wird. Eine erzeugte Skizze ist in Abbildung 5.2 zu sehen.

Bei dem Aufruf von `update_stats` werden jegliche Statistiken des MoE aktualisiert. Das ist im Standard der Nutzungszähler der aktiven Experten ($w_i > 0$) aus `_abs_freq`. In

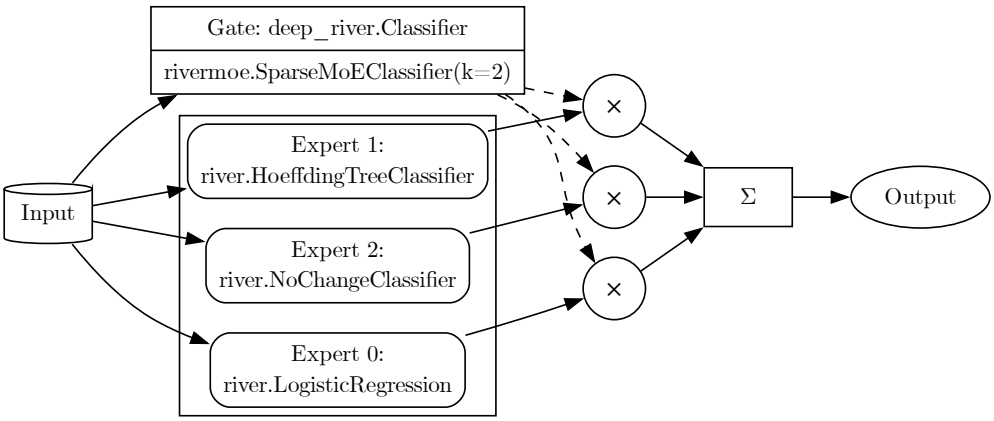


Abbildung 5.2.: Generierte Architekturskizze eines MoE mithilfe der `draw()`-Methode.

dem Attribut `_rel_freq` werden die relativen Häufigkeiten der Experten, für die Metrik aus Formel 4.4, als Eigenschaftsdekorator aus den absoluten Häufigkeiten berechnet. Die letzten Gating-Gewichte werden in `_gate_weights` gespeichert, um später die Beitragsmetrik (Formel 4.5) und normierte Entropie (Formel 4.6) berechnen zu können. Über die Speicherplatzwerte können Informationen zur Speicherung der Experten, des Gates – und in summierter Form – für das gesamte MoE abgerufen werden. Die adaptiven Eigenschaften des neuronalen Gates, die `deep-river` bereits umgesetzt hat, werden aktiviert. Damit können zur Lautzeit Eingangsdaten, das Label und die Anzahl an Experten im Gate verändert werden. Die Adaptivität entspricht einer der wichtigsten Eigenschaften von OML-Verfahren. Initialisiert werden die neuronalen Komponenten mithilfe von `_moe_initalize`. Dabei wird die Anzahl der Ausgangsneuronen, wenn nicht bereits gegeben, auf die Anzahl der Experten angepasst. Für das erste Training wird im Gate-Netzwerk der Bias b_0 auf 0 und die Startgewichte θ_0 auf einen konstanten Wert $\frac{1}{|E|}$ gesetzt, damit nicht, je nach Expertenreihenfolge, ein Experte bevorzugt wird. Die Experten aus den einzelnen MoE der Experimente wurden dafür experimentell variiert. Jede Reihenfolge führte korrekterweise zur gleichen Modellleistung.

Die grundlegenden Methoden der Basis-Regressoren und Basis-Klassifikatoren eines OML-Verfahrens sind die Methoden `learn_one` und `predict_one`. Dafür wurde zunächst von der Basis-MoE-Klasse geerbt und ein Basis-Regressor sowie ein Basis-Klassifikator implementiert. Diese wurden in den beiden Subklassen `MoEClassifier` und `MoERegressor` implementiert. Letztere Methode gibt die Vorhersage für einen Datensatz x_t zurück. Für Klassifikationen wird noch die `predict_proba_one` benötigt, die pro Klasse die Wahrscheinlichkeiten zurückgibt. Bei der Vorhersage der endgültigen Klassifikation wird die Klasse mit der höchsten Wahrscheinlichkeit zurückgegeben. Die Lernmethode `learn_one` trainiert

mit dem Wertepaar $d_t = (x_t, y_t)$ ein oder mehrere Experten und aktualisiert die Gewichte des Gates. Diese Methoden geben, wie in `river`, die grundlegende Funktionalität des Modells vor. In jedem Inferenz- und Trainingsschritt kann die Attributmenge durch neue, vorher ungesehene Attribute ergänzt werden. In der Klassifikation werden noch zusätzlich die eindeutigen Labels in jedem Schritt adaptiv aktualisiert. Bei der Umsetzung wurde auch die Eigenschaft berücksichtigt, dass ein Experte direkt ausgewählt wird, wenn im MoE nur ein Experte vorhanden ist. Dies lässt sich durch Abbildung 5.3 veranschaulichen. Hier sind die Leistungen des Experten und des *Single-Expert-MoE* identisch. Der generelle MoE-Klassen-Wrapper sorgt für etwas höheren Speicherbedarf und minimal höhere Laufzeiten.

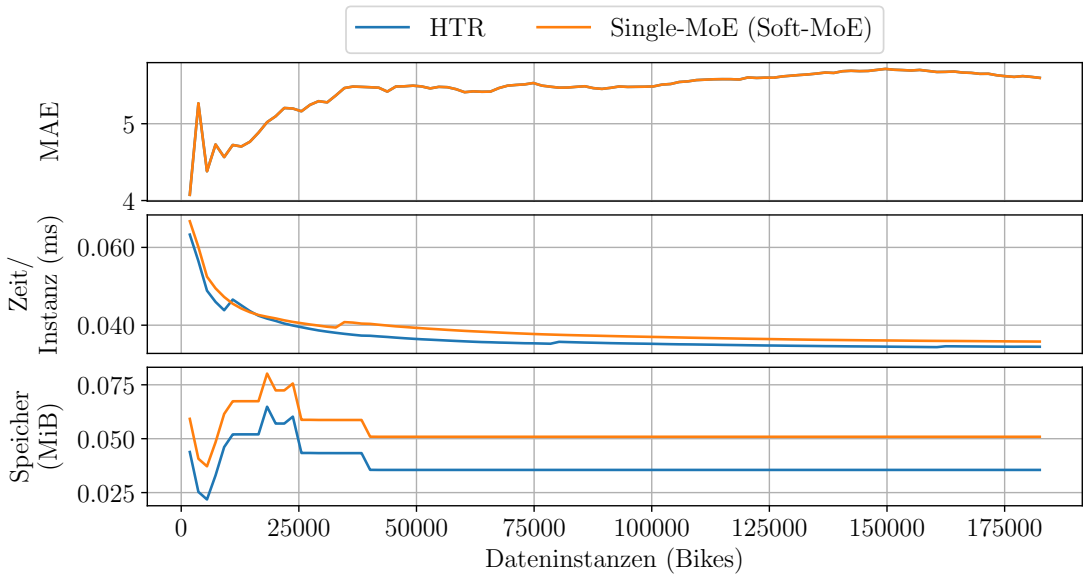


Abbildung 5.3.: Modellleistung ist bei Single-MoE identisch zur Expertenleistung ohne MoE.

MoE-Varianten Die grundlegenden MoE-Varianten, die in dieser Arbeit implementiert wurden, sind *Soft-Moe* nach Jacobs et al. (1991) und *Sparse-MoE* mit Top-K-Routing nach Shazeer et al. (2017). Für die Implementierung wird pro Problemart von dem MoE- und der Varianten-Klasse geerbt. Das generelle Vorhersage- und Lernverhalten einer MoE-Variante wird durch die Implementierung der abstrakten Methoden `_learn` und `_predict` bestimmt. Die wesentliche Gating-Logik der Variante wird durch die Implementierung der abstrakten `gating(x)`-Methode aus der `BaseVariant`-Klasse festgelegt.

Die Gating-Gewichte des Soft-Max entspricht dem Softmax[2] der Rohwerte $Z(x_t) = \theta_i \cdot x_t$ der Ausgabeneuronen: $G(x_t) = Softmax(Z(x_t))$. Für die Gesamtvorhersage werden die Vorhersagen aller Experten berechnet und durch die Gating-Gewichte gewichtet. Im Training wird die Gesamtvorhersage ebenfalls berechnet und dient neben dem tatsächlichen Wert als Eingabe für die Verlustfunktion, um das Gate sukzessiv zu verbessern.

Für das Sparse-MoE, bei dem nur die besten k-Experten aktiv sind, wurde zusätzlich der Parameter k eingeführt, der bei der Erstellung der MoE-Variante verlangt wird. Mit diesem Parameter werden immer die k-höchsten Gating-Gewichte w_i, mit einem Rauschen nach Shazeer et al. (2017) aus Formel 5.1 ausgewählt. Die besten Experten e werden über das Argument des Maximums abgeleitet: $e_i = \underset{i \in TopK(k)}{\arg\max}\, w_i$.

$$
\begin{aligned}
G(x_t) &= Softmax(KeepTopK(Z(x_t), k)) \\
Z(x_t)_i &= (\theta \cdot x_t)_i + \mathcal{N}(0,1) \cdot Softplus((\theta_{\text{Rauschen}} \cdot x_t)_i) \\
KeepTopK(W,k)_i &= \begin{cases} w_i, & \text{wenn } w_i \text{ in den besten } k\text{-Experte von } W \\ 0, & \text{sonst.} \end{cases}
\end{aligned}
\tag{5.1}
$$

Durch das Rauschen und der Softplus[3]-Funktion wird die Diversität der Experten erhöht. Die sparsame Auswahl an Experten reduziert Rechenaufwände im Training und erhöht die Inferenzgeschwindigkeit, da nicht alle Experten pro Durchlauf aktiv sind. Die Implementierung beider Varianten wurde mit `PyTorch`-Methoden umgesetzt, um die Kompatibilität zum bestehenden `deep-river` zu halten. Der generelle Ablauf für die Inferenz und das Lernen ist grundsätzlich für jeweils Regression und Klassifikation identisch. Ein passendes Aktivitätsdiagramm lässt sich aus dem Anhang A.2 entnehmen.

SAMoE Das *Sparse-MoE* stellt die Grundlage für *SAMoE*, eine adaptive MoE-Variante. Es wurde für die Klassifikation implementiert und erbt von der `SparseMoEClassifier`-Klasse. Der k-Wert wurde auf 1 gesetzt, damit immer nur ein Experte pro Durchlauf ausgewählt wird. Wie in der Architektur aus Abbildung 4.1 vorgestellt, ist das Novum der Variante die progressive Erweiterung des Expertenpools bei Drifterkennung. Deswegen wird ein Drift-Verfahren als *Drift-Detektor* bei der Initialisierung des Objektes erwartet. Die übergebenen (untrainierten) Experten-Algorithmen aus `experts` werden in einen Katalog `experts_catalog` geklont.

[2] $Softmax(z) = \frac{e^{z_i}}{\sum_{j=1} e^{z_j}}$
[3] $Softplus(z) = \ln(1 + e^z)$

Der Ablauf des dynamischen Hinzufügens von Experten wird durch das Listing aus 5.1 deutlich. Der Drift-Detektor wird in einem Trainingsschritt mit einem binären Fehlerwert $y_t \neq \hat{y}_t$ aktualisiert. Tritt ein Drift auf, ändert sich die Fehlerquote und der Drift-Detektor erkennt den Drift. Wird ein Drift erkannt, wird mithilfe von Modulo der letzte Experte aus dem Katalog zyklisch ausgewählt und zum Experten-Pool hinzugefügt. Dafür wird die `add_expert`-Methode aufgerufen. Dort wird die letzte Schicht im Gating um ein Ausgangsneuron adaptiv ergänzt und die Gewichte nach $\mathcal{N}(0, 1)$ normalverteilt initialisiert.

```
y_pred = self.predict_one(x)
error = 1 if y != y_pred else 0
self.drift_detector.update(error)
if self.drift_detector.drift_detected:
    expert_index = (self._n_experts-1) % len(self.expert_catalog)
    selected_expert = experts_catalog[expert_index]
    self.add_expert(selected_expert)
```

Listing 5.1: Dynamisches hinzufügen eines neuen Experten bei Drift-Erkennung.

NN-Generator Die neuronalen Gates und Experten können mithilfe eines entwickelten neuronalen Generators aus einer beliebigen Konfiguration erzeugt werden. Durch Angabe der Eingangs- und Ausgangsdimension, Aktivierungsfunktion und Ebenenkonfiguration erzeugt die `GenericNNArchitecture`-Klasse ein Pytorch `nn.Module`. Es werden linear verbundene Ebenen, aber auch LSTM-Schichten unterstützt. Bei den Aktivierungsfunktionen können bekannte Verfahren wie *ReLU* gewählt werden. Die generische NN-Architektur wird als Modul in der `GenericNN`-Klasse verwendet und stellt damit eine Aggregations-Beziehung im Klassendiagramm dar. Die entsprechenden Regressoren und Klassifikatoren erben die Eigenschaften von `GenericNN`. Das Anlegen eines neuronalen Schätzers für das Gate, wie in Tabelle 4.4 („m-ReLU-10-ReLU-$|E|$“), ist das durch Listing 5.2 möglich.

```
gate = GenericNNClassifier(
    layer_configs=[10,],
    activation_fn="relu",
    loss_fn="cross_entropy",
    output_activation=None,
    optimizer_fn="sgd"
)
```

Listing 5.2: Generierung eines inkrementellen Klassifikators mit einer versteckten Schicht aus 10 ReLU-aktivierten Neuronen und variablen Eingangs- und Ausgangsneuronen (ohne Aktivierung). Optimiert wird nach Cross Entropy (CE) mit *SGD*.

Die Verwendung des `riverMoE` ist voll kompatibel mit `river` und `deep-river` und wird mit der `NNGenerator`-Hilfsklasse vereinfacht. Damit lässt sich in wenigen Zeilen (Listing 5.3) ein inkrementelles MoE-Modell erstellen:

```
soft_moe = Soft-MoERegressor(
    gate = gate,
    experts = [LogisticRegression(), HoeffdingTreeClassifier()]
)
```

Listing 5.3: Einfaches Beispiel zur Erstellung eines inkrementellen MoE mit neuronalen SoftMax-Gate und zwei nicht neuronalen `river`-Experten.

Die Umsetzung eines Systems, das den MoE-Ansatz mit Online-Learning-Verfahren und in dynamischen Umgebungen ermöglicht, ist durch das `riverMoE`-Framework erreicht. Damit ist die Umsetzung und Kombination beider Ansätze erfolgt, was zur teilweisen Beantwortung der ersten Forschungsfrage (RQ1) führt. Die Evaluation des Frameworks erfolgt durch Simulation von Experimenten.

5.1.2. Simulation der Experimente

Alle Experimente aus Abschnitt 4.2.4 werden durch einen simulierten endlichen Datenstrom als produktives Szenario durchgeführt. Dabei fließen, anders als häufig in der Realität, die Daten mit einer gleichbleibenden Datenrate $v_{\text{konst.}}$, ohne Verzögerung, in das MoE-System. Durch die Verarbeitung einzelner Instanzen wird es möglich, auch theoretisch große Datenmengen ohne zusätzlichen Speicherplatz zu bearbeiten. Die Simulation spiegelt das Konzept der Adaptivität aus Abbildung 2.1 wider. Datensätze im Big Data können sich nach jedem Inferenz- und Trainingsschritt verändern und das MoE-System muss auf das *Dynamisches Optimierungs-Problem* reagieren.

Für die Evaluation wurde die `experiment`-Methode in der experimentellen Umgebung, dem Jupyter-Notebook, implementiert, die für die Simulation zuständig ist. Zur visuellen Darstellung der Ergebnisse erstellt diese Methode Grafiken und Tabellen zu den berechneten Metriken pro Modell und Experiment. Sinnvolle Metriken zur Evaluation wurden bereits im Abschnitt 4.2.5 vorgestellt. Durch das Hinzufügen neuer Merkmale zum Speicher- und Leistungsverhalten der Experten und des Gates konnten die MoE spezifischen Metriken realisiert werden. Beides trägt zur Beantwortung der dritten Forschungsfrage bei (RQ3). Wie effektiv die MoE-Architektur in dynamischen Umgebungen ist (RQ2), wird durch die Experimente evaluiert. Die Ergebnisse werden in den folgenden Abschnitten präsentiert und diskutiert.

5.2. Evaluation der Experimente

Es wurden insgesamt sechs Experimente durchgeführt, die in drei Kategorien unterteilt sind. Die erste Kategorie umfasst die Regressionsexperimente, die zweite Kategorie die Klassifikationsexperimente und die dritte Kategorie die Experimente zum katastrophalen Vergessen. Die Ergebnisse der Experimente werden in den folgenden Abschnitten präsentiert. Die Diskussion der Ergebnisse erfolgt in Bezug auf die Forschungsfragen.

5.2.1. Experimente für Regression

Experiment R.1 Im ersten Regressionsexperiment ging es um die Evaluation des MoE-Systems gegenüber LinR, HTR und DeepR. Werden die Liniendiagramme aus Abbildung 5.4 betrachtet, kann erkannt werden, dass die lineare Regression (LinR) besonders am Anfang eine sehr starke Abweichung im MAE hatte. Die anderen Modelle haben bereits anfangs einen recht niedrigen Fehlerwert. Im Verlauf ist es sichtbar, dass nicht neuronalen Modelle wie BaseR, HTR oder LinR am weningsten Zeit benötigt haben. Dieses Verhalten war zu erwarten, da aufwendigere *Forward* und *Backward*-Propragierungen bei neuronalen Netzen genutzt werden.

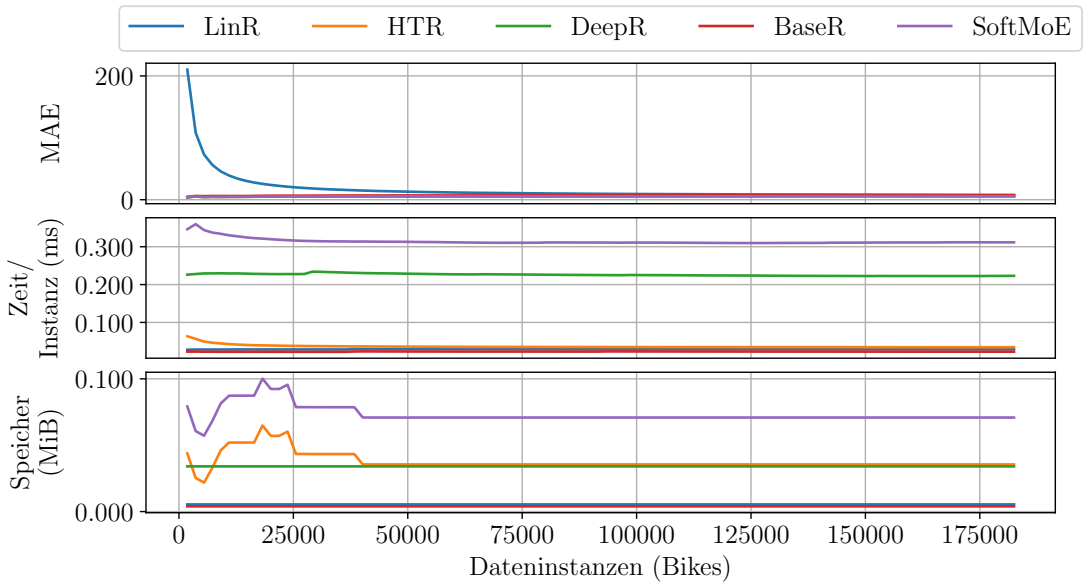


Abbildung 5.4.: Modellmetriken, Zeit- und Speicherbedarf pro Modell (Experiment R.1).

Eine bessere Übersicht der endgültigen Modellleistung zeigt Tabelle 5.1. Das Soft-MoE erzielt die besten Ergebnisse in MAE, RMSE und R2. Die Laufzeit ist jedoch am längsten und der Speicherbedarf am höchsten. Das Soft-MoE konnte nur sehr knapp bessere Ergebnisse als

das HTR erzielen. Dabei war das HTR-Modell, ohne neuronale Gating-Komponente, achtmal früher fertig und verbraucht nur die Hälfte des Speichers. Soft-MoE und HTR konnten beide das neuronale Netz DeepR übertreffen. Die lineare Regression schnitt in der progressiven Validierung schlechter als das Baseline-Modell ab und ist daher für das vorliegende Setting nicht geeignet.

Modell	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
LinR	5,262s	0,005MiB	0,029ms	7,646	271,008	-919,481
HTR	6,333s	0,036MiB	0,035ms	5,597	7,369	0,32
DeepR	40,712s	0,034MiB	0,223ms	5,783	7,667	0,263
BaseR	**4,136s**	**0,004MiB**	**0,023ms**	7,247	8,933	-0
SoftMoE	56,855s	0,071MiB	0,312ms	**5,589**	**7,349**	**0,323**

Tabelle 5.1.: Übersicht Metriken aller Modelle (Experiment R.1). Beste Ergebnisse **markiert**.

Ein Ergebnis ist, dass der Speicherverbrauch vom Soft-MoE identisch verläuft wie der von HTR, bloß nach oben verschoben. Am Anfang steigt der Speicherbedarf, nach ungefähr 35000 Instanzen sinkt der Bedarf und bleibt konstant. Dieses Verhalten ist zu erwarten, da die anderen beiden MoE-Experten (LinR, BaseR) kaum Speicher benötigen und der Verbrauch daher primär von HTR getrieben wird. DeepR und HTR konvergieren zu einem gleichen Speicherverbrauch. Der Speicherverbrauch von Soft-MoE ist am höchsten. Das ist insofern nachvollziehbar, als ein MoE speichertechnisch aus einem neuronalen Gate und den einzelnen Experten besteht. Die detaillierten MoE-Metriken aus Tabelle 5.2 der letzten Instanz bestätigen diese Vermutung. Die Addition der Gewichte von DeepR, einem architektonisch ähnlichen Netzwerk zum Gate, zum Speicherbedarf der drei nicht neuronalen Experten, ergibt einen Gesamtwert von 0,081 MiB. Das Soft-MoE ist in der Realität mit 0,074 MiB etwas sparsamer. Dennoch lässt sich der größte Speicherbedarf des MoE dadurch erklären. Das schnellste und speichertechnisch leichteste Modell ist das Base-Modell, was darauf zurückzuführen ist, dass es sich lediglich um den Cumulative Moving Average handelt – also die Berechnung eines skalaren Werts.

		Häufigkeit		Speicherbedarf	
Modell	Beitrag Relativ	Absolut	Relativ	Absolut	Relativ
StatisticRegressor	0,00 %	182470	33,30 %	490 B	0,70 %
HoeffdingTreeRegressor	35,00 %	182470	33,30 %	33,1 KiB	47,40 %
LinearRegression	65,00 %	182470	33,30 %	2,23 KiB	3,20 %
Gate				33,96 KiB	48,70 %

Tabelle 5.2.: Detaillierte Metriken für das Soft-MoE (Experiment R.1).

Die Tabelle 5.2 zeigt einen detaillierten Einblick in das Soft-MoE-Modell. Im Soft-MoE wurden alle Experten mit 33,3 % gleich häufig verwendet. Bemerkenswert ist, dass die schlechtere Lineare Regression mit 65 % durchschnittlich den größten Beitrag an der Leistung hat, während sie beim HTR, mit besserer Modellgüte, nur 35 % beiträgt. Das Gating-Routing führt dennoch gewichtet zum Ergebnis (MAE: 5,589). Wie im oberen Absatz bereits beschrieben, hat das Gate mit 34 KiB die gleiche Größe wie das neuronale Netz in DeepR. Der HTR benötigt aber weniger Speicherplatz als das alleinstehende Modell. Das kann damit zusammenhängen, dass durch die geringere Nutzung ein kleinerer Entscheidungsbaum entstanden ist. Einen Einblick in die Gategewicht-Verteilung gibt die Abbildung 5.5.

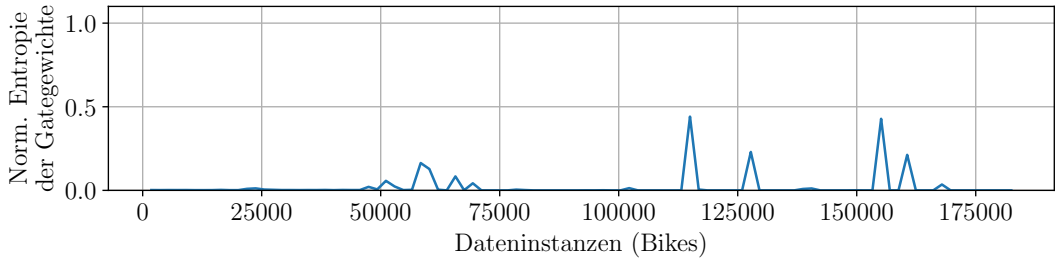


Abbildung 5.5.: Normierte Entropie der Soft-MoE Gategewichte (Experiment R.1).

Die normierte Entropie der Gategewichte über den zeitlichen Verlauf zeigt eine starke Tendenz von 0. Das spricht deutlich dafür, dass das Gate primär einzelne Experten gewichtet hat. Das Gate hat sich damit häufig spezialisiert.

Experiment R.2 Im zweiten Experiment wurden die Sparse-MoE Modelle untersucht. Zu diesem Zweck wurden jeweils die Top k 1, 2 oder 3 der drei Experten gewählt, um die Gesamtprognose zu berechnen. In der Abbildung 5.6 sind die Ergebnisse über den zeitlichen Verlauf dargestellt. Zu Beginn ist zu erkennen, dass anfangs primär der Entscheidungsbaum von HTR aufgebaut wurde, was die Geschwindigkeit und Modellleistung etwas negativ beeinflusst hat. Weiterhin ist erkennbar, dass die Modellleistung sinkt, je weniger Experten verwendet werden. Da Experten in der Regel unterschiedliche Stärken und Schwächen haben, erzielen sie oft kombiniert bessere Ergebnisse.

Die zeitlichen und speicherbedingten Vorteile der verschiedenen k-Einstellungen, werden durch die Ergebnistabelle 5.3 deutlicher. Das Sparse-MoE-Modell ($k = 1$), das Training und Inferenz mit nur einem Experten ausführt, arbeitete am schnellsten. Zeitgleich hat es die stabilsten Ergebnisse erzielt. Der RMSE, der größere Fehler bestraft, ist hier am niedrigsten. Zusätzlich konnte das Modell die Daten am genauesten beschreiben (R2 = 0,188). Der mittlere

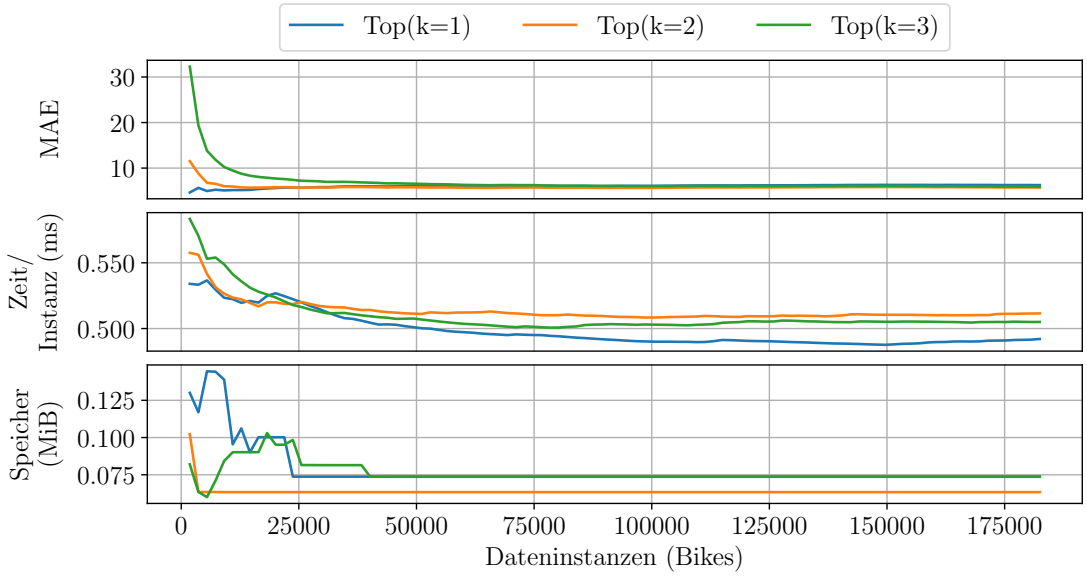


Abbildung 5.6.: Modellmetriken, Zeit- und Speicherbedarf pro Modell (Experiment R.2).

Fehler ist, wenn mehr Experten herangezogen werden ($k = 2$ oder $k = 3$), niedriger. Gut zu erkennen ist, dass die Anzahl und Auswahl der Experten einen Einfluss auf die Laufzeit und den Speicherbedarf haben. So verbraucht das Sparse-MoE mit zwei Experten ($k = 2$) am geringsten Speicher, hat aber länger gebraucht als das Sparse-MoE mit nur einem Experten ($k = 1$). Eine identische Leistung zwischen Soft-MoE und Sparse-MoE bei Nutzung aller Experten ($k = 3$) ist durch die unterschiedliche Umsetzung nicht gegeben.

Modell	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
Top(k=1)	**89,788s**	0,074MiB	**0,492ms**	6,267	**8,049**	**0,188**
Top(k=2)	93,331s	**0,063MiB**	0,511ms	**5,728**	21,515	-4,802
Top(k=3)	92,15s	0,074MiB	0,505ms	5,917	79,232	-77,677

Tabelle 5.3.: Metriken des Sparse-MoE (Experiment R.2). Beste Ergebnisse **markiert**.

Das Soft-MoE schneidet insgesamt besser ab als das Sparse-MoE. Eine mögliche Erklärung gibt das Sparse-MoE ($k = 3$) der Tabelle 5.4. Anders als bei dem Soft-MoE ist die relative Häufigkeit der Experten im Sparse-MoE sehr unterschiedlich, obwohl alle Experten ausgewählt wurden. In der Häufigkeit werden nur aktive Experten berücksichtigt, deren Gewicht größer 0 ist. Vermutlich wurde im Sparse-MoE durch das hinzugefügte Rauschen vor der Top-K-Auswahl eine klare Spezialisierung unter den Experten gefördert. Angesichts dessen wurde vermehrt

HTR gewählt, der auch mit 86 % den größten Beitrag hatte. Der HTR hatte bereits ohne MoE die besten Ergebnisse für MAE, RMSE und R2 (Abbildung 5.4).

		Häufigkeit		Speicherbedarf	
Modell	Beitrag Relativ	Absolut	Relativ	Absolut	Relativ
LinearRegression	11,20 %	33160	14,92 %	2,23 KiB	3,20 %
StatisticRegressor	2,80 %	33040	14,87 %	490 B	0,70 %
HoeffdingTreeRegressor	86,00 %	155977	70,20 %	33,1 KiB	47,50 %
Gate				33,95 KiB	48,70 %

Tabelle 5.4.: Detaillierte Metriken für das Sparse-MoE Top($k = 3$) (Experiment R.2).

Im Kontrast zum Sparse-MoE ($k = 3$) lässt sich beim Sparse-MoE ($k = 1$) aus Tabelle 5.5 erkennen, dass die relative Häufigkeit und der durchschnittliche Beitrag gleich verteilt sind und dem Soft-MoE sehr ähneln. Das könnte daran liegen, dass das Gate mit der zu kleinen Lernrate nicht gelernt hat, gut zwischen den Experten zu differenzieren. Es muss darauf hingewiesen werden, dass Experten in diesem Modell eine geringere Wahrscheinlichkeit haben, trainiert zu werden, als bei $k = 3$, wo alle Experten eine gleichmäßige Wahrscheinlichkeit besitzen. Diese durchschnittliche Trainingshäufigkeit liegt demnach bei $\frac{k}{|E|}$. Daraus könnte folgen, dass der HTR nicht zwingend die gleiche Trainingsmöglichkeit hatte wie im Soft-MoE und dadurch seltener als Experte dominierte.

		Häufigkeit		Speicherbedarf	
Modell	Beitrag Relativ	Absolut	Relativ	Absolut	Relativ
LinearRegression	30,00 %	60889	33,40 %	2,23 KiB	3,20 %
StatisticRegressor	31,00 %	61054	33,50 %	490 B	0,70 %
HoeffdingTreeRegressor	39,00 %	60527	33,20 %	33,14 KiB	47,50 %
Gate				33,95 KiB	48,60 %

Tabelle 5.5.: Detaillierte Metriken für das Sparse-MoE Top($k = 1$) (Experiment R.2).

Der Vorteil der Sparsamkeit des Sparse-MoE wurde in diesen Experimenten nur bedingt deutlich, da die eingesetzten nicht neuronalen Modelle als Experten bereits grundsätzlich schnell und sparsam sind. Der Effekt wird deutlicher, wenn neuronale Experten zum Einsatz kommen, die im Einzelnen eine hohe Trainings- und Inferenzzeit haben. Für das vorliegende Regressionsproblem konnte das MoE mit nicht neuronalen Experten teilweise bessere Ergebnisse erzielen als vergleichbare Modelle ohne MoE. Die Ergebnisse hängen aber auch von der Wahl der Parameter ab, die hier für den Vergleich identisch gesetzt wurden.

5.2.2. Experimente für Klassifikation

Experiment C.1 Im ersten Experiment zum Klassifikationsproblem wurde, analog zur Regression, das Soft-MoE mit anderen Modellen verglichen. Die Ergebnisse dazu finden sich in der Abbildung 5.7 wieder.

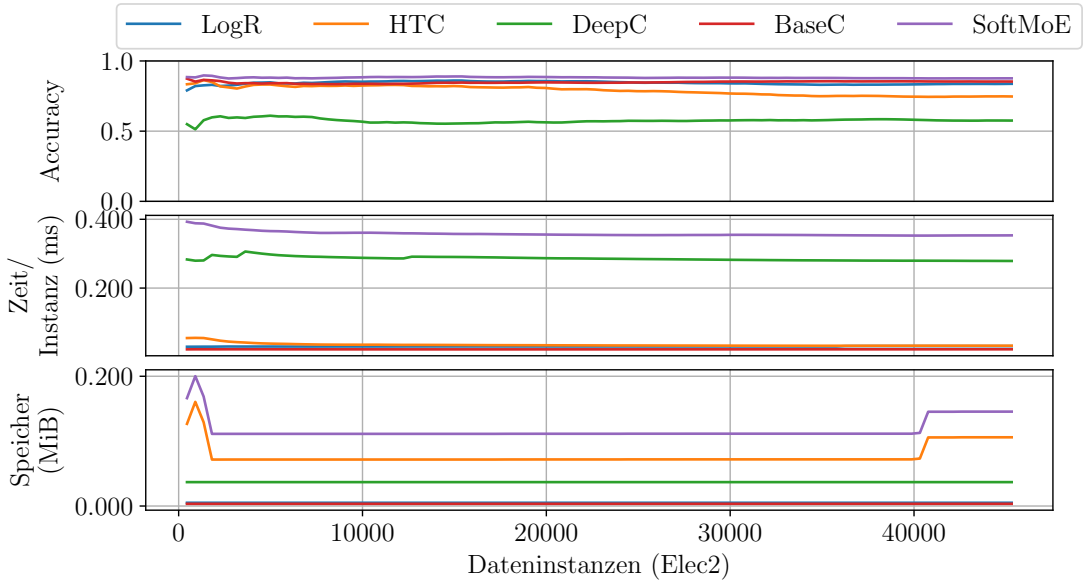


Abbildung 5.7.: Detaillierte Metriken für das Soft-MoE (Experiment C.1).

Über den gesamten Trainings- und Inferenzzeitraum bewegen sich die erfassten Metriken mit einigen Ausnahmen recht konstant. So ist der Speicherbedarf des MoE und des HTC am Anfang ähnlich hoch wie in den Regressionsexperimenten. Nach nur wenigen Hundert Durchläufen reduziert sich der Bedarf aber wieder. Der Anstieg und Abstieg des Speicherbedarfs verhält sich beim Soft-MoE und HTC wieder identisch, obwohl der Soft-MoE insgesamt mehr Speicher benötigt. Die Gründe dazu wurden bereits bei der Regression erörtert und treffen auch hier zu.

		Häufigkeit		Speicherbedarf	
Modell	Beitrag Relativ	Absolut	Relativ	Absolut	Relativ
LogisticRegression	41,80 %	45312	33,30 %	2,54 KiB	1,70 %
NoChangeClassifier	36,40 %	45312	33,30 %	531 B	0,40 %
HoeffdingTreeClassifier	21,80 %	45312	33,30 %	105,91 KiB	72,60 %
Gate				36,96 KiB	25,30 %

Tabelle 5.6.: Detaillierte Metriken für das Soft-MoE (Experiment C.1).

Interessant zu beobachten ist, dass der HTC über die Zeit an *Accuracy* verliert, obwohl der Soft-MoE seine Leistung hält. Das könnte daran liegen, dass der HTC-Algorithmus nicht in der Lage ist, die Datenstruktur zu adaptieren. Das Soft-MoE hingegen kann durch das dynamische Gate und die Experten besser auf die Daten reagieren. Die genauen Ergebnisse der letzten Instanz lassen sich in der Tabelle 5.7 ablesen.

Modell	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
LogR	1,411s	0,005MiB	0,031ms	0,837	0,84	0,762	0,799
HTC	1,471s	0,106MiB	0,032ms	0,747	0,742	0,619	0,675
DeepC	12,634s	0,037MiB	0,279ms	0,575	0,5	0	0
BaseC	**0,996s**	**0,003MiB**	**0,022ms**	0,853	0,827	0,827	0,827
SoftMoE	16,008s	0,146MiB	0,353ms	**0,876**	**0,867**	**0,836**	**0,852**

Tabelle 5.7.: Übersicht Metriken aller Modelle (Experiment C.1). Beste Ergebnisse **markiert**.

Für alle Modellmetriken hat das Soft-MoE die besten Werte erreicht. Erstaunlich ist, dass DeepC für die vorliegende Datenverteilung deutlich schlechter abschneidet als die anderen Modelle. Besonders im Recall hat DeepC keine Ergebnisse erhalten. Grund dafür könnte sein, dass das neuronale Netz für diese Problemstellung zu einfach ist. Das Baseline-Modell, das die letzte Klasse als Prognose zurückliefert, scheint auf diesem Datensatz besser zu funktionieren. In der Implementierung der Klassifikations-Varianten wurde auf die korrekte OML-Lernreihenfolge geachtet, dass zuerst vorhergesagt und dann gelernt wird. Andernfalls würde das Baseline-Modell (letzte Klasse) immer die richtige Klasse vorab lernen. Besonders im Bereich des Recalls und F1-Scores konnte das Soft-MoE mit HTC deutlich bessere Ergebnisse erzielen als HTC alleine.

Ein wichtiger Vorteil von MoE, mit Einsatz von nicht neuronalen Experten, ist ihre Erklärbarkeit, die auch schon von Sharma, Henderson und Ghosh (2023) in FEAMoE untersucht wurde. Die zugrundeliegenden Modelle, wie die logistische Regression oder der Entscheidungsbaum aus HTC, lassen sich einfach interpretieren. Die logistische Regression lässt sich als einfaches Wahrscheinlichkeitsmodell darstellen. Der trainierte Entscheidungsbaum aus dem HTC-Modell des Soft-MoE aus Abbildung 5.8 konnte einfach exportiert werden.

Eine weitere Möglichkeit, die zur Erklärbarkeit bei MoE beiträgt, sind die Gategewichte. Diese entscheiden über die Gesamtprognose aus den Teilprognosen. Die Metrik der normierten Entropie gibt Aufschluss darüber, wie das Gate die Experten gewichtet hat. In Abbildung 5.9 ist zu erkennen, dass die Entropie über den gesamten Zeitraum sehr hoch ist, es dennoch immer wieder auch zu Schwankungen kommt. Das spricht dafür, dass das Gate grundsätzlich eher unsicher ist und gleichverteilter gewichtet. Das kann auch daran liegen, dass alle der

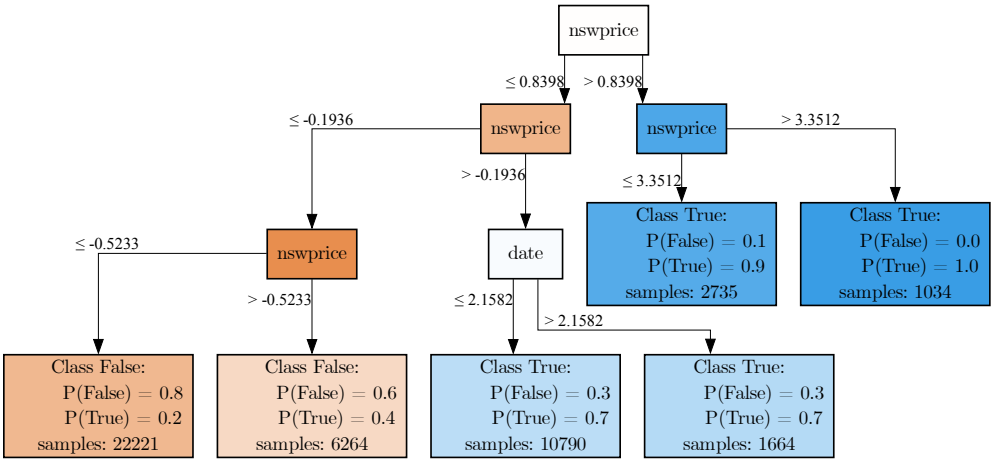


Abbildung 5.8.: HTC als exportierter Binärbaum als erklärbares Modell (Experiment C.1).

nicht neuronalen Experten ähnlich gut sind und tatsächlich die Kombination der gewichteten Wahrscheinlichkeiten zu den besten Ergebnissen führt. Das lässt sich ebenfalls durch den relativ gleichverteilten durchschnittlichen Beitrag aus Tabelle 5.6 ablesen.

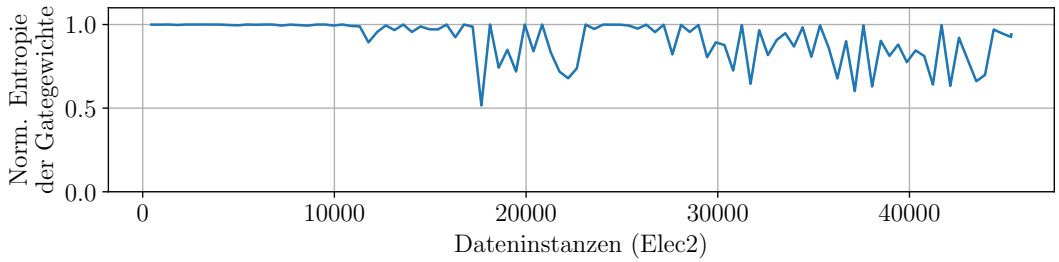


Abbildung 5.9.: Normierte Entropie der Soft-MoE Gategewichte (Experiment C.1).

Experiment C.2 Im zweiten Experiment zum Klassifikationsproblem, wurde das Sparse-MoE-Modell mit unterschiedlichen k-Werten untersucht. Der Ablauf des Experiments dazu findet sich in der Abbildung 5.10 wieder.

Zu erkennen ist, dass die Leistungen der einzelnen Sparse-Modelle fast identisch hoch und stabil sind. Ähnlich wie bei der Regression sinkt die Modellleistung, umso weniger Experten verwendet werden. Das lässt sich durch die tabellarischen Ergebnisse der letzten Instanz aus Tabelle 5.8 erkennen. Das schnellste und kleinste Modell war das Sparse-MoE Top($k = 1$), das nur einen Experten pro Instanz verwendet. Die Unterschiede zwischen den Sparse-MoE in der Laufzeit und Speicherverbrauch sind nur marginal. Die Komplexität der eingesetzten

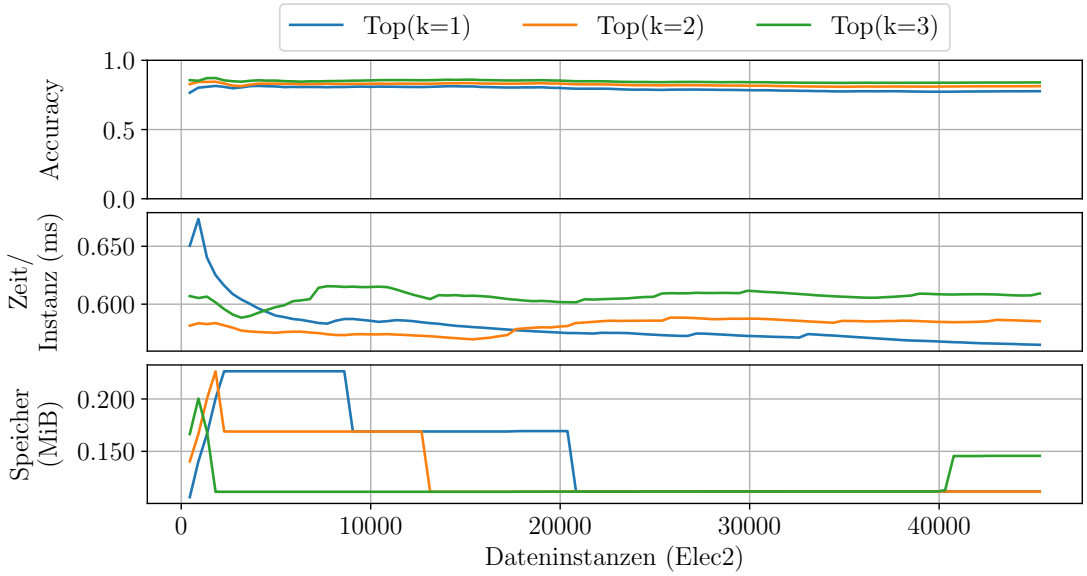


Abbildung 5.10.: Modellmetriken, Zeit- und Speicherbedarf pro Modell (Experiment C.2).

Experten-Modelle spielt eine Rolle, um den Vorteil von sparsamen Sparse-MoE deutlicher hervorzubringen. Der Speicherbedarf war am Anfang am stärksten, hat sich aber je schneller reduziert, umso mehr Experten im Einsatz waren. Das könnte wieder mit der durchschnittlichen Trainingshäufigkeit zusammenhängen, also dass das dominierende HTC-Modell bei $k = 3$ am ehesten vollständig trainiert war. Die Modellgüte mit den meisten Experten ist am höchsten, aber ebenfalls schlechter als beim Soft-MoE. Die Gründe dafür sind analog zum Regressionsproblem.

Modell	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
Top(k=1)	**25,596s**	0,112MiB	**0,565ms**	0,777	0,776	0,667	0,718
Top(k=2)	26,52s	**0,112MiB**	0,585ms	0,813	0,801	0,746	0,772
Top(k=3)	27,605s	0,146MiB	0,609ms	**0,84**	**0,838**	**0,773**	**0,804**

Tabelle 5.8.: Metriken des Sparse-MoE (Experiment C.2). Beste Ergebnisse **markiert**.

Eine mögliche Lösung wäre das Einführen von *Temperature Scaling* nach Nguyen, Akbarian und Ho (2024). Dabei handelt es sich um einen stetig sinkenden Temperaturparameter τ, in der Softmax-Funktion: $Softmax(\frac{z_i}{\tau})$. Damit ließe sich die Entropie der Gategewichtsverteilung steuern. Ein höherer Temperaturwert führt anfangs zu einer gleichmäßigeren Verteilung, sodass Experten häufiger trainiert werden können. Durch das stetige Sinken des Temperatur-

kalierungsfaktors spezialisiert sich das MoE allmählich. Die Herausforderung im OML-Kontext ist die Strategie zur Senkung des Temperaturwerts, da Streamingdaten kein festes Ende haben.

Insgesamt konnte `riverMoE` in den Klassifikationsexperimenten die Ergebnisse von Montiel, Halford et al. (2021) übertreffen. Die Entwickler von dem Basis-Framework `river` erzielten mit der logistischen Regression für Experimente mit dem *Elec2*-Datensatz eine Accuracy von 67,97 % und für HTC einen Wert von 75,55 %. Die genauen Parameter wurden dabei nicht genannt. Die durchgeführten Experimente dieser Arbeit führten mit gleichem Datensatz zu einer Accuracy von 83,73 % für die logistische Regression und 74,70 % für das HTC-Modell. Das Soft-MoE-Modell hat die höchste Accuracy von 87,62 % erreicht. In dieser Arbeit ist das eine Verbesserung von 4,66 % für LogR und 17,27 % für HTC.

Die Ergebnisse der bisherigen Experimente zu Regressions- und Klassifikationsproblemen realer Datensätze zeigen, dass eine gewichtete Kombination von (nicht-)neuronalen Experten zu besseren Ergebnissen führen kann, auch wenn der Speicher- und Rechenaufwand nachvollziehbar höher ist. Eine Rechengeschwindigkeit von durchschnittlich weniger als 0,5ms für die Inferenz- und Trainingszeit ist gegenüber Batch maschinelles Lernen, trotz der recht komplexen Struktur, ein Vorteil. Das MoE-System arbeitet dabei ähnlich wie die Ensemble-Methode `VotingClassifier` von Dietterich (2000), die Ergebnisse verschiedener ML-Algorithmen gewichtet zusammenfasst. Der Unterschied liegt lediglich darin, dass die Gewichte statisch festgelegt werden müssen oder zeitaufwendig durch Hyperparameteroptimierung optimiert werden müssen. Ein neuronales Gate in `riverMoE` kann dagegen gezielt auf Eingangsdaten trainiert werden, um effektiv die besten Gewichte zu ermitteln.

Die Resultate bestätigen noch einmal, dass eine Umsetzung der MoE-Architektur und seiner Varianten im Streaming-Kontext funktioniert und zu vergleichbaren oder besseren Ergebnissen führen kann. Damit wird die erste Forschungsfrage (RQ1), die auf die Machbarkeit zielte, beantwortet. Die ausgewählten Metriken und Evaluationsmethodik haben sich als sinnvoll erwiesen, um die Modelle zu vergleichen und zu bewerten. Der Einsatz von grafischen Verläufen hilft dabei, die Veränderungen über die Zeit zu visualisieren und zu interpretieren. Durch aggregierte Kennzahlen wie *normierte Entropie der Gategewichte* oder *durchschnittlicher Beitrag der Experten* konnten zusätzliche Einblicke in das Verhalten der MoE-Modelle geben. Die dritte Forschungsfrage zur Evaluationsmethodik (RQ3) ist damit beantwortet.

5.2.3. Experimente zu katastrophalem Vergessen

Die zweite Forschungsfrage (RQ2) hatte den Fokus auf die Anpassungsfähigkeit in dynamischen Umgebungen mit Drift. Dafür wurde das `riverMoE` so implementiert, dass es bei

Drift die Expertenanzahl dynamisch progressiv erhöht. Zur Untersuchung des Verhaltens bei „Katastrophalem Vergessen“ wurden folgende zwei Experimente durchgeführt.

Experiment D.1 Im ersten Experiment wurden künstliche Daten mit wiederkehrendem Konzept durch die *Friedman*-Formel generiert. Die resultierenden Ergebnisse des Experiments D.1 lassen sich aus Abbildung 5.11 entnehmen. Im gesamten Konzept 1 konnte DeepR schnell die höchste Genauigkeit erreichen. Beim Übergang zum Konzept 2 (Drift) verschlechtert sich die Genauigkeit von DeepR und HATR deutlich gegenüber Soft-MoE. Das ist überraschend, da HATR einen expliziten ADWIN Drift-Detektor verwendet, um gezielt auf Drift reagieren zu können. Das Soft-MoE-Modell konnte am schnellsten die Genauigkeit aus Konzept 1 wiederherstellen. Nach Rückkehr zum ersten Konzept konnten alle Modelle nach kurzer Zeit auf das vorherige MAE zurückkehren. Die Ergebnisse der letzten Instanz sind in Tabelle 5.9 zusammengefasst.

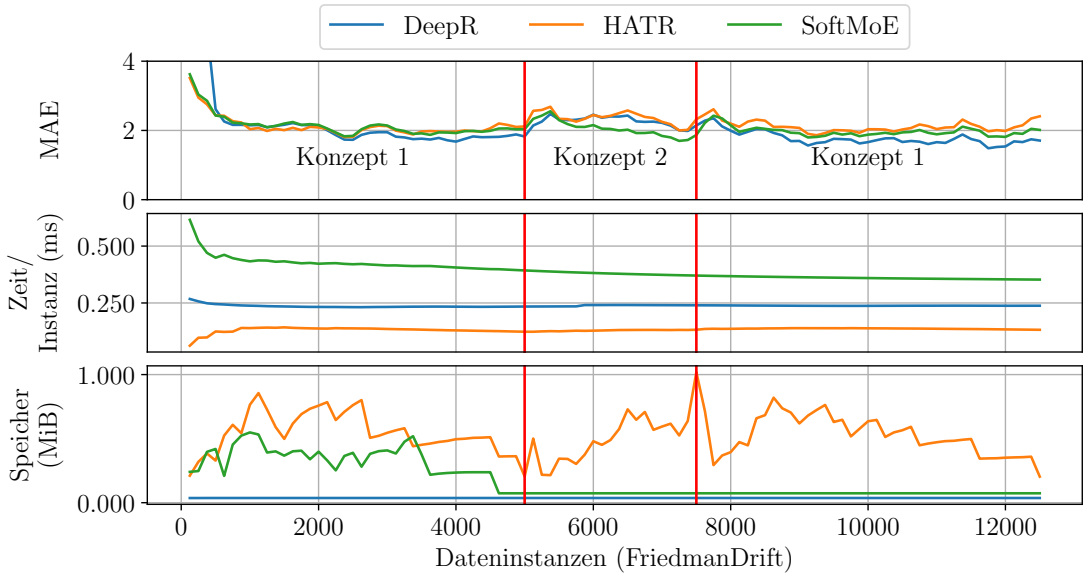


Abbildung 5.11.: Übersicht Metriken aller Modelle (Experiment D.1). GRA-Drift an den Stellen 5000 und 7500. Beste Ergebnisse **markiert**.

Durch die Grafik und Tabelle kann erkannt werden, dass der HATR deutlich schneller als die beiden anderen OML-Verfahren war. Dennoch ist der Speicherbedarf bei der Rückkehr zu Konzept 1 zwischenzeitlich sehr hoch. Das liegt vermutlich an den adaptiven Komponenten von HATR. Der Bedarf sinkt zwar wieder, aber die Größe der anderen Modelle ist deutlich kleiner. Auffällig ist, dass der Bedarf in der letzten Instanz für DeepR am geringsten ist, der in

den anderen Experimenten höher war. Der Drift hat keine signifikante Auswirkung auf die Trainings- und Inferenzzeit gehabt.

Modell	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
DeepR	2,974s	**0,037MiB**	0,238ms	**1,707**	**2,32**	**0,803**
HATR	**1,651s**	0,203MiB	**0,132ms**	2,408	3,066	0,655
SoftMoE	4,409s	0,074MiB	0,353ms	2,015	2,606	0,751

Tabelle 5.9.: Metriken aller Modelle (Experiment D.1). Beste Ergebnisse **markiert**.

Bei der endgültigen Modellgüte lässt sich das MoE zwischen DeepR und HATR einordnen. Am schlechtesten schnitt HATR ab, was überrascht. Insgesamt konnte das Soft-MoE-Modell mit einem MAE von 2,015 und einem RMSE von 2,606 ein deutlich besseres Ergebnis für das synthetische Problem erzielen als das nicht neuronale Modell. Dennoch hat in dieser Konstellation das DeepR-Modell die höchste Modellgüte. Alle untersuchten Verfahren konnten, mit den gewählten Parametern, recht schnell auf das wiederkehrende Konzept zurückfallen. Dabei kam es nicht zum katastrophalen Vergessen.

Experiment D.2 Im zweiten Drift-Experiment wurde das Label-Shift für Multiklassifizierung mit der neuartigen *SAMoE*-Variante mit Drift-Detektor aus Abbildung 4.1 untersucht. Nach 5000 Instanzen wurden die Trainingsdaten gewechselt – bei Vorhersagen wurde aber weiterhin nach allen Klassen gefragt. Die gemessenen Accuracy-Werte pro Modell können aus Abbildung 5.12 grafisch entnommen werden. Die durchschnittlichen Werte für Accuracy, Micro-Precision, Micro-Recall und Micro-F1-Score pro Modell vor und nach Anpassung der Aufgabe sind in Tabelle 5.10 zusammengefasst.

	Metrik	Accuracy		Micro-Precision		Micro-Recall		Micro-F1-Score	
Aufgabe	Modell	A	B	A	B	A	B	A	B
Aufgabe 1	DeepC	**0,515**	**0,125**	**0,505**	0,086	**0,515**	**0,125**	**0,423**	0,084
	HATC	0,243	0,104	0,250	**0,235**	0,243	0,104	0,236	**0,134**
	SAMoE	0,248	0,002	0,257	0,005	0,248	0,002	0,239	0,003
Aufgabe 2	DeepC	0,000	0,179	0,000	0,228	0,000	0,179	0,000	0,090
	HATC	0,000	0,150	0,000	0,298	0,000	0,150	0,000	0,168
	SAMoE	0,000	**0,340**	0,000	**0,330**	0,000	**0,340**	0,000	**0,261**

Tabelle 5.10.: Durchschnittliche Modellmetriken pro Aufgabe und Instanzbereich $A \in [0, 5000]$ und $B \in [5001, 10000]$ (Experiment D.2). Beste Ergebnisse **markiert**.

Nach dem Label-Shift hat das neuronale Netz die erste Aufgabe noch gut gelöst und dabei leicht angefangen, die zweite Aufgabe zu lernen. Ab ungefähr 6500 Instanzen hat das neuronale Netz verlernt, die erste Aufgabe zu lösen und konnte die zweite Aufgabe besser lösen. Dieses katastrophale Vergessen lässt sich auch bei der SAMoE-Variante beobachten. Der Driftdetektor von SAMoE hat korrekt die einzige Änderung bei Instanz 5152 erkannt, also 152 Instanzen nach dem Label-Shift. Die Erkennung der Änderung hat damit zuverlässig funktioniert. Dennoch hat das adaptive Hinzufügen des neuen Experten dazu geführt, dass der erste Experte komplett deaktiviert wurde. Dadurch konnte der neue HTC-Experte die neue Aufgabe deutlich besser lernen als die anderen Modelle, Aufgabe 1 konnte aber nicht mehr gelöst werden.

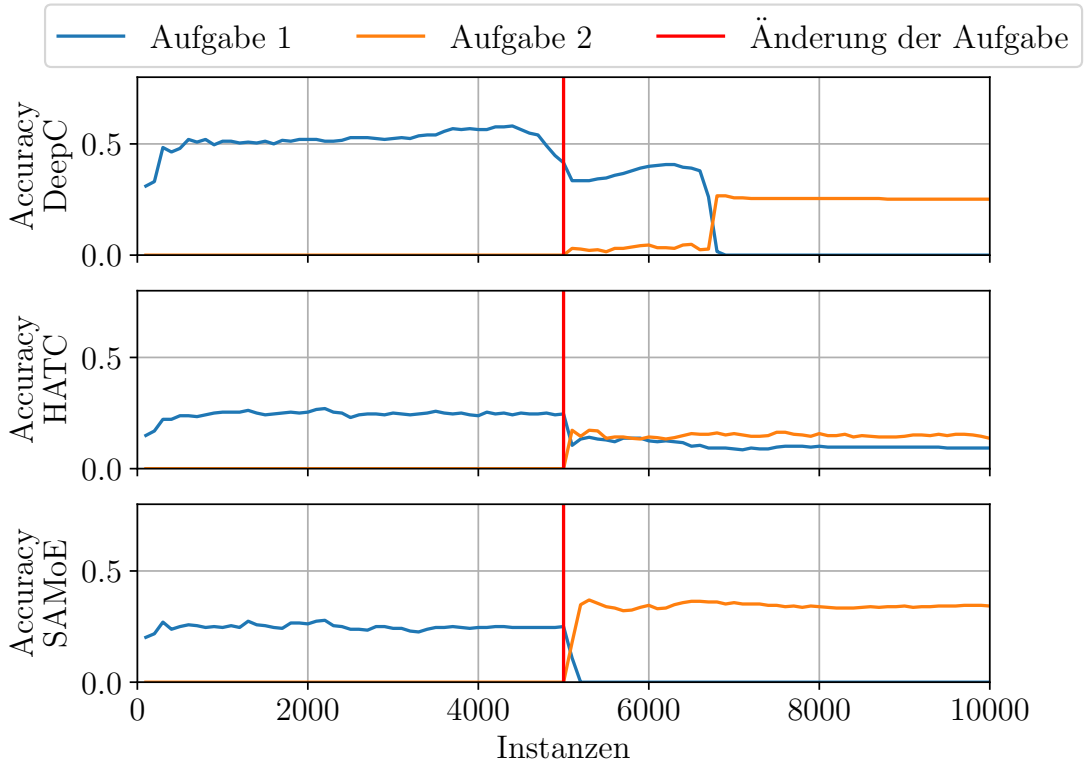


Abbildung 5.12.: Accuracy für Modelle mit Label-Shift nach 5000 Instanzen (Experiment D.2).

Mit dem Label-Shift ging der HATC mit ADWIN-Drift-Detektor am besten um. Die Änderungen in den Klassen wurden schneller als bei SAMoE erkannt und haben den Entscheidungsbaum adaptiv um eine Abzweigung ergänzt. Damit bleibt der Zugriff zum bereits gelernten bestehen. Durchschnittlich schneidet DeepC am besten für Aufgabe 1 im Instanzbereich ab. Bei Aufgabe 2 konnte das SAMoE durch das fälschlicherweise gezielte Lernen die höchsten Genauigkei-

ten erreichen. Langfristig wird das HATC-Modell die durchschnittlich besseren Leistungen erhalten, da kein katastrophales Vergessen eingetreten ist.

Die zweite Forschungsfrage dieser Arbeit, zum Thema der Anpassungsfähigkeit in dynamischen Umgebungen (RQ2), wurde durch die beiden Experimente unterschiedlich beantwortet. Ein normales Soft-MoE konnte, mit vorliegenden Parametern, stabiler mit dem wechselnden und wiederkehrenden Konzept umgehen als HATC oder dem neuronalen Netz. Ändert sich in der Multi-Klassifikation das Trainingsmaterial durch Label-Shift kann die neu eingeführte SAMoE-Variante den Drift gut erkennen. Dennoch wurden beim Hinzufügen des neuen Experten die Ausgangsneuronen schlecht initialisiert. Das bereits Gelernte vom ersten Experten ist damit nicht vergessen, sondern wird nur nicht abgerufen. Eine verbesserte Initialisierungsmethode kann diesem Problem entgegenwirken.

5.3. Limitationen

Die durchgeführten Experimente haben einige Limitationen, die in zukünftigen Arbeiten berücksichtigt werden sollten. Zunächst wurde nur eine sehr spezifische Parameterauswahl der einzelnen und äußerst unterschiedlichen Verfahren getroffen. Diese können, besonders bei komplexen Komponenten wie neuronalen Netzen, einen großen Einfluss auf die Leistung haben. Ein Vergleich kann damit sehr unterschiedlich ausfallen. Die durchgeführten Experimente wurden alle nur einmalig mit festem Zufallsstartwert (*Prequential Validation*) statt variierendem Zufallsstartwert durchgeführt. Bei nicht deterministischen Verfahren, wie sie in dieser Arbeit zum Einsatz kamen, hat das ebenfalls einen Einfluss auf die Modellleistung. Damit sind die Ergebnisse nur schwer generalisierbar.

Weiterhin wurde sich mit Soft-MoE und Sparse-MoE in dieser Arbeit auf grundlegende MoE-Architekturen konzentriert. Viele andere Variationen wurden nicht weiter untersucht. Diese können aber Einfluss im OML-Kontext haben. Der Fokus der Arbeit lag auf der grundlegenden, inkrementellen MoE-Architektur. Daher wurde die neuartige Variante *SAMoE* nur mit einem Experiment evaluiert, obwohl die dynamische Erweiterung des MoE-Systems ein vielversprechender Ansatz ist.

Der hier durchgeführte Umfang ist nicht ausreichend, um eine vollständige Beurteilung der Leistung von inkrementellen MoE-Systemen widerzuspiegeln. Weitere Untersuchungen und Experimente sind notwendig, um ein tieferes Verständnis der Potenziale und Herausforderungen solcher Systeme zu erlangen.

6. Schlussfolgerung

Im abschließenden Kapitel werden die wichtigsten Erkenntnisse zusammengefasst und kritisch reflektiert. Zudem wird ein Ausblick auf zukünftige Forschungsperspektiven und mögliche Weiterentwicklungen gegeben, um die Einsatzmöglichkeiten von MoE in dynamischen Umgebungen weiter zu optimieren.

6.1. Zusammenfassung

Die vorliegende Arbeit untersuchte den Einsatz von *Mixture of Experts* (MoE) im Kontext des *Online maschinelles Lernen* (OML) zur kontinuierlichen Verarbeitung von Streamingdaten. Durch den stetigen Anstieg an Internet of Things (IoT) Geräten wächst die Menge an generierten Daten exponentiell, sodass effiziente Methoden zur Echtzeitverarbeitung erforderlich sind. Die theoretischen Grundlagen zeigen, dass Online Machine Learning hierbei eine Lösung bietet, da es kontinuierlich neue Daten verarbeitet, ohne dass eine vollständige Speicherung notwendig ist. Eine weitere mögliche Lösung zur Verbesserung der Vorhersagegenauigkeit und Modellrobustheit ist der Einsatz von MoE, dessen Konzept ebenfalls theoretisch eingeführt wurde. Die systematische Literaturrecherche hat gezeigt, dass an verschiedenen (adaptiven) MoE-Variationen geforscht wird, jedoch bisher nur wenige Arbeiten die Kombination von MoE und OML untersuchen. Daraus wurden spezifische Forschungsfragen abgeleitet, die in der Arbeit beantwortet wurden.

Die Kombination von OML und MoE war die erste Forschungsfrage und wurde in dieser Arbeit durch das implementierte `river`MoE-Framework, das die Grundlage für MoE-Architekturen mit inkrementellem Lernen bereitstellt, beantwortet. Dabei wurde die einfache Schnittstelle des Basis-Framework `river` übernommen, um die Wiederverwendbarkeit beizubehalten. In durchgeführten Experimenten wurden unter gleichen Bedingungen verschiedene Datensätze für die Regression und Klassifikation genutzt, um die Modellgüte und Leistung mit anderen inkrementellen Verfahren zu vergleichen. In der Regression konnten marginal bessere Ergebnisse mit MoE gegenüber Modellen ohne MoE erzielt werden, während in der Klassifikation die Genauigkeit des Modells durch die Expertenmischung signifikant verbessert wurde.

Die Experimente zeigen, dass die inkrementellen MoE-Modelle in der Lage sind, kontinuierlich neue Daten zu integrieren und dabei die Vorhersagegenauigkeit zu verbessern (RQ1).

Ein zentraler Aspekt dieser Arbeit war die Frage, inwiefern sich die Modellanpassung durch das dynamische Hinzufügen neuer Experten verbessern lässt. Die durchgeführten Experimente belegen, dass die Leistung des Modells durch eine dynamische Expertenselektion des Gates gesteigert werden kann. In Szenarien mit Konzeptdrift zeigte sich, dass die adaptive Expertenmischung robust auf Veränderungen reagieren kann. Die neu eingeführte *SAMoE*-Variante mit Drift-Detektor konnte den Drift schnell erkennen und einen Experten hinzufügen. Dabei wurde aber der bisherige Experte deaktiviert (RQ2).

Um die Leistung von MoE im Kontext kontinuierlich wachsender Streamingdaten angemessen zu bewerten, wurden verschiedene Evaluationsmetriken eingesetzt. Neben klassischen Fehlerraten wurde besonderes Augenmerk auf Gating-Diversität (Entropie), Rechengeschwindigkeit und Speicherverbrauch gelegt. Die Ergebnisse zeigen, dass offline-basierte Metriken alleine nicht ausreichen, um die Performance von MoE in einer Streaming-Umgebung umfassend zu bewerten. Stattdessen sind spezifische Methoden zur progressiven Validierung erforderlich, um die tatsächliche Güte des Modells realistisch abzubilden (RQ3).

6.2. Fazit

Zusammenfassend konnten die Forschungsfragen beantwortet werden. Die Kombination von MoE und OML ist ein vielversprechender Ansatz, um adaptive MoE-Modelle zu entwickeln, die kontinuierlich neue Daten integrieren können. Die durchgeführte systematische Literaturrecherche zeigt ebenfalls, wie aktuell und relevant die Expertenmischung für die Forschung ist. Die Forschung hat sich bisher größtenteils nur auf Offline-Verfahren konzentriert, obwohl inkrementelle Verfahren in der Praxis immer wichtiger werden.

Das umgesetzte Framework `riverMoE` bietet eine flexible und modulare Basis für die Implementierung von MoE-Architekturen, die einfach um neue Varianten erweitert werden können. Die durchgeführten Experimente zeigen, dass die adaptive Expertenmischung die Vorhersagegenauigkeit und Modellrobustheit verbessern kann. Dabei wurden nur lineare Experten verwendet, obwohl möglicherweise noch mehr Potenzial in der Wahl von neuronalen Experten steckt. In Szenarien mit Konzeptdrift zeigten die MoE-Modelle ein schlechteres Verhalten als erwartet. Auch wenn die Drifterkennung zuverlässig funktionierte, führt der deaktivierte Experte zu katastrophalen Vergessen – ein Kernproblem von inkrementellen Lernverfahren. Gerade durch die Nutzung von disjunkten Experten wurde erwartet, dass dieses Problem nicht auftritt.

Zusätzlich sind die entstandenen Verbesserungen in der Genauigkeit auf Kosten der höheren Rechenzeit und Speichernutzung entstanden. Aufgrund der komplexen neuronalen Komponente wird es schwierig werden, diesen Trade-Off zu verbessern. Die aufgelisteten Limitationen und Herausforderungen zeigen, dass es noch viel Forschungsbedarf gibt, um MoE-Modelle in dynamischen Umgebungen weiter zu etablieren. Der nächste Abschnitt gibt einen möglichen Ausblick auf zukünftige Forschungsperspektiven und Weiterentwicklungen.

6.3. Ausblick

Die vorliegende Arbeit bietet viel Potenzial für weitere Forschungsarbeiten. Zunächst sind weitere Experimente mit anderen Datensätzen, Gating- und Expertenkonfigurationen sinnvoll, um die Leistungsfähigkeit von MoE weiter zu untersuchen. Interessant sind dabei besonders spezielle Streaming-Datensätze für OML, die sich dynamisch verändern und Drift erzeugen. Das könnte den Vorteil der Kombination von MoE und OML verdeutlichen. Üblicherweise werden neuronale Experten für MoE verwendet, das wurde in dieser Arbeit nicht betrachtet. Die Verwendung von vortrainierten Experten könnte ebenfalls interessant sein, da das Gate möglicherweise schneller das optimale Routing ermittelt.

Durch die theoretischen Grundlagen und systematische Literaturrecherche wurde die Vielzahl der Variationen von MoE deutlich. Daher ist es naheliegend, das modulare `riverMoE`-Framework um weitere MoE-Varianten wie Adaptive-MoE, hierarchische MoE oder Deep-MoE zu erweitern. Interessant wäre auch die Ergänzung um Multi-Gate MoE, um Multi-Task-Lernen für kontinuierliches Lernen zu erforschen.

Die bereits vorgestellte *SAMoE* könnte erweitert werden, um das katastrophale Vergessen zu vermeiden. Dazu kann die Initialisierung der Ausgangsneuronen bei neuen Experten optimiert und mit dem vorgestellten *Temperature Scaling* kombiniert werden. Eine weitere mögliche Lösung wäre die *Replay Memory* Methode. Gleichzeitig könnte betrachtet werden, ob bestehende Experten, die nicht benötigt werden, auch entfernt werden können. Das würde den Speicherplatzbedarf reduzieren. Um SAMoE besser zu evaluieren, könnten weitere Experimente mit verschiedenen Driftszenarien durchgeführt werden.

Durch den Einsatz der vorgestellten MoE-Metriken und der nicht neuronalen Experten, wie den inkrementellen Entscheidungsbäumen, konnte in dieser Arbeit die Erklärbarkeit von inkrementellen MoE-Modellen gefördert werden. Im Rahmen der verantwortungsvollen Künstlichen Intelligenz ist es wichtig, die Gating- und Expertenentscheidungen von inkrementellen MoE-Modellen transparenter zu machen. Daher ist es sinnvoll, wie bereits in der FEAMoE-Arbeit, die Erklärbarkeit von MoE-Modellen weiterzuerforschen.

Literaturverzeichnis

Akidau, Tyler, Slava Chernyak und Reuven Lax (2018). *Streaming systems: the what, where, when, and how of large-scale data processing*. First edition. OCLC: ocn975362965. Sebastopol, CA: O,Reilly. ISBN: 978-1-4919-8387-4.

Bartz-Beielstein, Thomas und Eva Bartz, Hrsg. (2023). *Online Machine Learning: Eine praxisorientiere Einführung*. ger. 1. Auflage 2023. Wiesbaden: Springer Fachmedien Wiesbaden GmbH. ISBN: 978-3-658-42504-3.

Behnel, Stefan et al. (März 2011). "Cython: The Best of Both Worlds". In: *Comput. Sci. Eng.* 13.2, S. 31–39. ISSN: 1521-9615. DOI: 10.1109/MCSE.2010.118.

Bifet, Albert (2017). "Classifier Concept Drift Detection and the Illusion of Progress". en. In: *Artificial Intelligence and Soft Computing*. Hrsg. von Leszek Rutkowski et al. Bd. 10246. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, S. 715–725. ISBN: 978-3-319-59060-8. DOI: 10.1007/978-3-319-59060-8_64.

Bry, François und Nicola Henze (Juni 2005). "Personalisierung". de. In: *Informatik Spektrum* 28.3, S. 230–233. ISSN: 0170-6012, 1432-122X. DOI: 10.1007/s00287-005-0489-y.

Chen, Chonghao et al. (Sep. 2024). "BP-MoE: Behavior Pattern-aware Mixture-of-Experts for Temporal Graph Representation Learning". en. In: *Knowledge-Based Systems* 299, S. 112056. ISSN: 09507051. DOI: 10.1016/j.knosys.2024.112056.

Chen, Kuanhao, Zijie Yue und Miaojing Shi (Juni 2023). "Space-time video super-resolution using long-term temporal feature aggregation". en. In: *Auton. Intell. Syst.* 3.1, S. 5. ISSN: 2730-616X. DOI: 10.1007/s43684-023-00051-9.

Chen, Wuyang et al. (2023). "Lifelong language pretraining with distribution-specialized experts". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML' 23. Place: Honolulu, Hawaii, USA. JMLR.org, S. 5383–5395. DOI: 10.5555/3618408.

Chen, Zixiang et al. (2024). "Towards understanding the mixture-of-experts layer in deep learning". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. event-place: New Orleans, LA, USA. Red Hook, NY, USA: Curran Associates Inc. ISBN: 978-1-71387-108-8.

Dietterich, Thomas G. (2000). "Ensemble Methods in Machine Learning". In: *Multiple Classifier Systems*. Hrsg. von Gerhard Goos, Juris Hartmanis und Jan Van Leeuwen. Bd. 1857. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–15. ISBN: 978-3-540-67704-8. DOI: 10.1007/3-540-45014-9_1.

Domingos, Pedro und Geoff Hulten (Aug. 2000). "Mining high-speed data streams". en. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. Boston Massachusetts USA: ACM, S. 71–80. ISBN: 978-1-58113-233-5. DOI: 10.1145/347090.347107.

Du, Jing et al. (Juli 2022). "Hierarchical Task-aware Multi-Head Attention Network". en. In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Madrid Spain: ACM, S. 1933–1937. ISBN: 978-1-4503-8732-3. DOI: 10.1145/3477495.3531781.

Du, Nan et al. (Juli 2022). "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts". In: *Proceedings of the 39th International Conference on Machine Learning*. Hrsg. von Kamalika Chaudhuri et al. Bd. 162. Proceedings of Machine Learning Research. PMLR, S. 5547–5569.

Eigen, David, Marc' Aurelio Ranzato und Ilya Sutskever (März 2014). *Learning Factored Representations in a Deep Mixture of Experts*. arXiv:1312.4314 [cs]. DOI: 10.48550/arXiv.1312.4314.

Ernst, Hartmut (2000). *Grundlagen und Konzepte der Informatik: Eine Einführung in die Informatik ausgehend von den fundamentalen Grundlagen*. ger. 2., überarbeitete und verbesserte Auflage. Springer eBook Collection Computer Science and Engineering. Wiesbaden s.l: Vieweg+Teubner Verlag. ISBN: 978-3-663-10229-8. DOI: 10.1007/978-3-663-10229-8.

Fedus, William, Barret Zoph und Noam Shazeer (Jan. 2022). "Switch transformers: scaling to trillion parameter models with simple and efficient sparsity". In: *J. Mach. Learn. Res.* 23.1. Publisher: JMLR.org. ISSN: 1532-4435.

Finch, Tony (2009). "Incremental calculation of weighted mean and variance". In: *University of Cambridge* 4.11-5. Publisher: Citeseer, S. 41–42.

Friedman, Jerome H. (März 1991). "Multivariate Adaptive Regression Splines". In: *Ann. Statist.* 19.1. ISSN: 0090-5364. DOI: 10.1214/aos/1176347963.

Fritz, Jürgen (2022). *Datenbasierte Optimierung des Business Management Systems: Geschäftsprozesse verbessern mit Data Analytics, Industrie 4.0, KI, Chatbots und Co.* ger. Hanser eLibrary. München: Hanser. ISBN: 978-3-446-47131-3. DOI: 10.3139/9783446472549.

Gulcan, Ege Berkay und Fazli Can (März 2023). "Unsupervised concept drift detection for multi-label data streams". en. In: *Artif Intell Rev* 56.3, S. 2401–2434. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-022-10232-2.

Gumm, Heinz-Peter, Manfred Sommer und Wolfgang Hesse (2011). *Einführung in die Informatik*. ger. 9., vollst. überarb. Aufl. München: Oldenbourg. ISBN: 978-3-486-59711-0. DOI: 10.1524/9783486704587.

Halford, Max (2016). *OpenBikes Challenge - A benchmark for online machine learning on bike sharing data.*

Halford, Max et al. (Juni 2020). *creme, a Python library for online machine learning.*

Harries, Michael, New South Wales et al. (1999). "Splice-2 comparative evaluation: Electricity pricing". In: Publisher: University of New South Wales, School of Computer Science and Engineering.

Hihn, Heinke und Daniel A. Braun (Feb. 2023). "Hierarchically structured task-agnostic continual learning". en. In: *Mach Learn* 112.2, S. 655–686. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-022-06283-9.

Hihn, Heinke und Daniel A. Braun (Apr. 2024). "Online continual learning through unsupervised mutual information maximization". en. In: *Neurocomputing* 578, S. 127422. ISSN: 09252312. DOI: 10.1016/j.neucom.2024.127422.

Hu, Songhua et al. (Sep. 2024). "Graph transformer embedded deep learning for short-term passenger flow prediction in urban rail transit systems: A multi-gate mixture-of-experts model". en. In: *Information Sciences* 679, S. 121095. ISSN: 00200255. DOI: 10.1016/j.ins.2024.121095.

Huang, Tao et al. (Aug. 2024). "Pull together: Option-weighting-enhanced mixture-of-experts knowledge tracing". en. In: *Expert Systems with Applications* 248, S. 123419. ISSN: 09574174. DOI: 10.1016/j.eswa.2024.123419.

Jacobs, Robert A. et al. (Feb. 1991). "Adaptive Mixtures of Local Experts". en. In: *Neural Computation* 3.1, S. 79–87. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1991.3.1.79.

Jiang, Albert Q. et al. (Jan. 2024). *Mixtral of Experts*. arXiv:2401.04088 [cs].

Jiang, Shen et al. (Aug. 2024). "Automatic Multi-Task Learning Framework with Neural Architecture Search in Recommendations". en. In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Barcelona Spain: ACM, S. 1290–1300. DOI: 10.1145/3637528.3671715.

Jordan, Michael und Robert Jacobs (1991). "Hierarchies of adaptive experts". In: *Advances in Neural Information Processing Systems*. Hrsg. von J. Moody, S. Hanson und R. P. Lippmann. Bd. 4. Morgan-Kaufmann, S. 985–992. DOI: 10.5555/2986916.2987037.

Jordan, Michael I. und Robert A. Jacobs (März 1994). "Hierarchical Mixtures of Experts and the EM Algorithm". en. In: *Neural Computation* 6.2, S. 181–214. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1994.6.2.181.

Khan, Nawsher et al. (März 2018). "The 10 Vs, Issues and Challenges of Big Data". en. In: *Proceedings of the 2018 International Conference on Big Data and Education*. Honolulu HI USA: ACM, S. 52–56. ISBN: 978-1-4503-6358-7. DOI: 10.1145/3206157.3206166.

Kobayashi, Shusuke und Susumu Shirayama (Mai 2021). "Selecting data adaptive learner from multiple deep learners using Bayesian networks". en. In: *Neural Comput & Applic* 33.9, S. 4229–4241. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-020-05234-6.

Kulbach, Cedric et al. (2025). "DeepRiver: A Deep Learning Library for Data Streams". In: *Journal of Open Source Software* 10.105. Publisher: The Open Journal, S. 7226. DOI: 10.21105/joss.07226.

Kullback, S. und R. A. Leibler (März 1951). "On Information and Sufficiency". en. In: *Ann. Math. Statist.* 22.1, S. 79–86. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729694.

Laney, Douglas (Feb. 2001). "3D Data Management: Controlling Data Volume, Velocity, and Variety". In: *Application Delivery Strategies*. Techn. Ber. META Group, S. 1–4.

Li, Danwei et al. (Aug. 2023). "AdaTT: Adaptive Task-to-Task Fusion Network for Multitask Learning in Recommendations". en. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Long Beach CA USA: ACM, S. 4370–4379. DOI: 10.1145/3580305.3599769.

Li, Jiamin et al. (2023). “Adaptive Gating in Mixture-of-Experts based Language Models”. en. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, S. 3577–3587. DOI: 10.18653/v1/2023.emnlp-main.217.

Liu, Zhuang et al. (Juni 2022). “A ConvNet for the 2020s”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, S. 11966–11976. ISBN: 978-1-66546-946-3. DOI: 10.1109/CVPR52688.2022.01167.

Ma, Jiaqi et al. (Juli 2018). “Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts”. en. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London United Kingdom: ACM, S. 1930–1939. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220007.

Miller, Damian, Hrsg. (2005). *E-Learning: eine multiperspektivische Standortbestimmung*. ger. Bern: Haupt. ISBN: 978-3-258-06898-5.

Montiel, Jacob, Max Halford et al. (2021). “River: machine learning for streaming data in Python”. In: *Journal of Machine Learning Research* 22.110, S. 1–8.

Montiel, Jacob, Jesse Read et al. (2018). “Scikit-Multiflow: A Multi-output Streaming Framework”. In: *Journal of Machine Learning Research* 19.72, S. 1–5.

Moreno-Torres, Jose G. et al. (Jan. 2012). “A unifying view on dataset shift in classification”. en. In: *Pattern Recognition* 45.1, S. 521–530. ISSN: 00313203. DOI: 10.1016/j.patcog.2011.06.019.

Nguyen, Huy, Pedram Akbarian und Nhat Ho (2024). “Is temperature sample efficient for softmax gaussian mixture of experts?” In: *Proceedings of the 41st International Conference on Machine Learning*. ICML,24. Place: Vienna, Austria. JMLR.org.

Page, Matthew J et al. (März 2021). “PRISMA 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews”. en. In: *BMJ*, n160. ISSN: 1756-1833. DOI: 10.1136/bmj.n160.

Park, Chung et al. (Juli 2024). “Pacer and Runner: Cooperative Learning Framework between Single- and Cross-Domain Sequential Recommendation”. en. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Washington DC USA: ACM, S. 2071–2080. DOI: 10.1145/3626772.3657710.

Raab, Christoph, Moritz Heusinger und Frank-Michael Schleif (Nov. 2020). "Reactive Soft Prototype Computing for Concept Drift Streams". en. In: *Neurocomputing* 416, S. 340–351. ISSN: 09252312. DOI: 10.1016/j.neucom.2019.11.111.

Rahman, Md Hishamur et al. (Dez. 2024). "Gated ensemble of spatio-temporal mixture of experts for multi-task learning in ride-hailing system". en. In: *Multimodal Transportation* 3.4, S. 100166. ISSN: 27725863. DOI: 10.1016/j.multra.2024.100166.

Rahul, Kumar, Rohitash Kumar Banyal und Neeraj Arora (Aug. 2023). "A systematic review on big data applications and scope for industrial processing and healthcare sectors". en. In: *J Big Data* 10.1, S. 133. ISSN: 2196-1115. DOI: 10.1186/s40537-023-00808-2.

Raibulet, Claudia (2008). "Facets of Adaptivity". en. In: *Software Architecture*. Hrsg. von David Hutchison et al. Bd. 5292. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 342–345. ISBN: 978-3-540-88029-5. DOI: 10.1007/978-3-540-88030-1_33.

Raschka, Sebastian, Vahid Mirjalili und Knut Lorenzen (2018). *Machine Learning mit Python und Scikit-learn und TensorFlow: das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. ger. 2., aktualisierte und erweiterte Auflage. mitp Professional. Frechen: mitp. ISBN: 978-3-95845-733-1.

Reinsel, David, John Gantz und John Rydning (Nov. 2018). "The Digitization of the World From Edge to Core". en. In: *Framingham: International Data Corporation* 16. Publisher: IDC Corporate Framingham, MA, USA, S. 1–28.

Shapley, Lloyd S. (Okt. 1988). "A value for n -person games". In: *The Shapley Value*. Hrsg. von Alvin E. Roth. 1. Aufl. Cambridge University Press, S. 31–40. ISBN: 978-0-521-36177-4. DOI: 10.1017/CBO9780511528446.003.

Sharma, Shubham, Jette Henderson und Joydeep Ghosh (Aug. 2023). "FEAMOE: Fair, Explainable and Adaptive Mixture of Experts". en. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. Macau, SAR China: International Joint Conferences on Artificial Intelligence Organization, S. 492–500. ISBN: 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/55.

Shazeer, Noam et al. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. Version Number: 1. DOI: 10.48550/ARXIV.1701.06538.

Statlog (Image Segmentation) (1990). DOI: 10.24432/C5P01G.

Tang, Hongyan et al. (Sep. 2020). "Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations". en. In: *Fourteenth ACM Conference on Recommender Systems*. Virtual Event Brazil: ACM, S. 269–278. ISBN: 978-1-4503-7583-2. DOI: 10.1145/3383313.3412236.

Team, NLLB et al. (Aug. 2022). *No Language Left Behind: Scaling Human-Centered Machine Translation*. arXiv:2207.04672 [cs].

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS' 17. event-place: Long Beach, California, USA. Red Hook, NY, USA: Curran Associates Inc., S. 6000–6010. ISBN: 978-1-5108-6096-4.

Wang, Xin et al. (Mai 2021). "Gated neural network framework for interactive character control". en. In: *Multimed Tools Appl* 80.11, S. 16229–16246. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-020-08792-y.

Warren, James (2015). *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. eng. New York: Manning Publications Co. LLC. ISBN: 978-1-61729-034-3.

Wright, Adam (Mai 2024). *Worldwide IDC Global DataSphere Forecast, 2024–2028: AI Everywhere, But Upsurge in Data Will Take Time*. en. Tech Supplier US52076424. International Data Corporation.

Zhao, Zhe et al. (Sep. 2019). "Recommending what video to watch next: a multitask ranking system". en. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. Copenhagen Denmark: ACM, S. 43–51. ISBN: 978-1-4503-6243-6. DOI: 10.1145/3298689.3346997.

A. Ergebnisse

A.1. Klassendiagramm

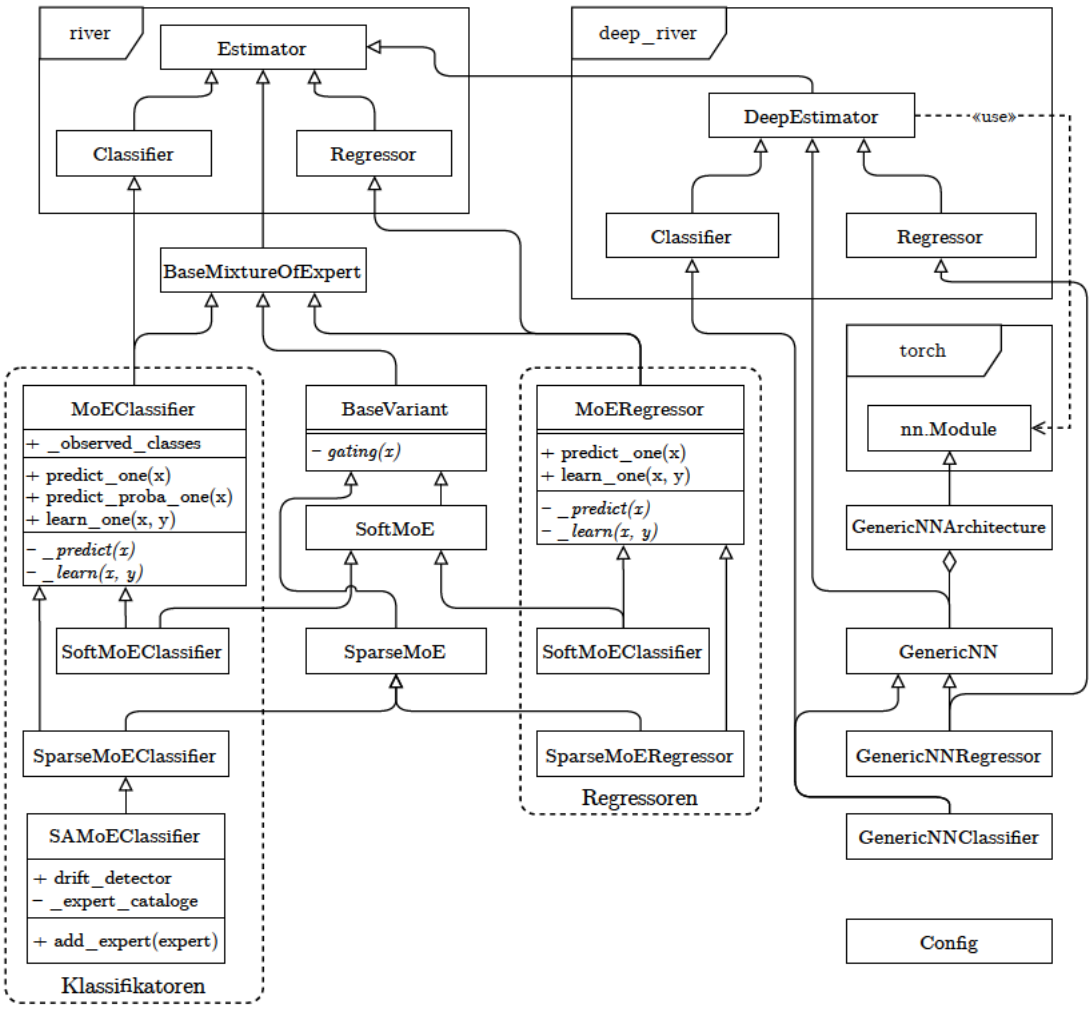


Abbildung A.1.: Vereinfachtes Klassendiagramm des `river`MoE-Framework.

A.2. Aktivitätsdiagramm

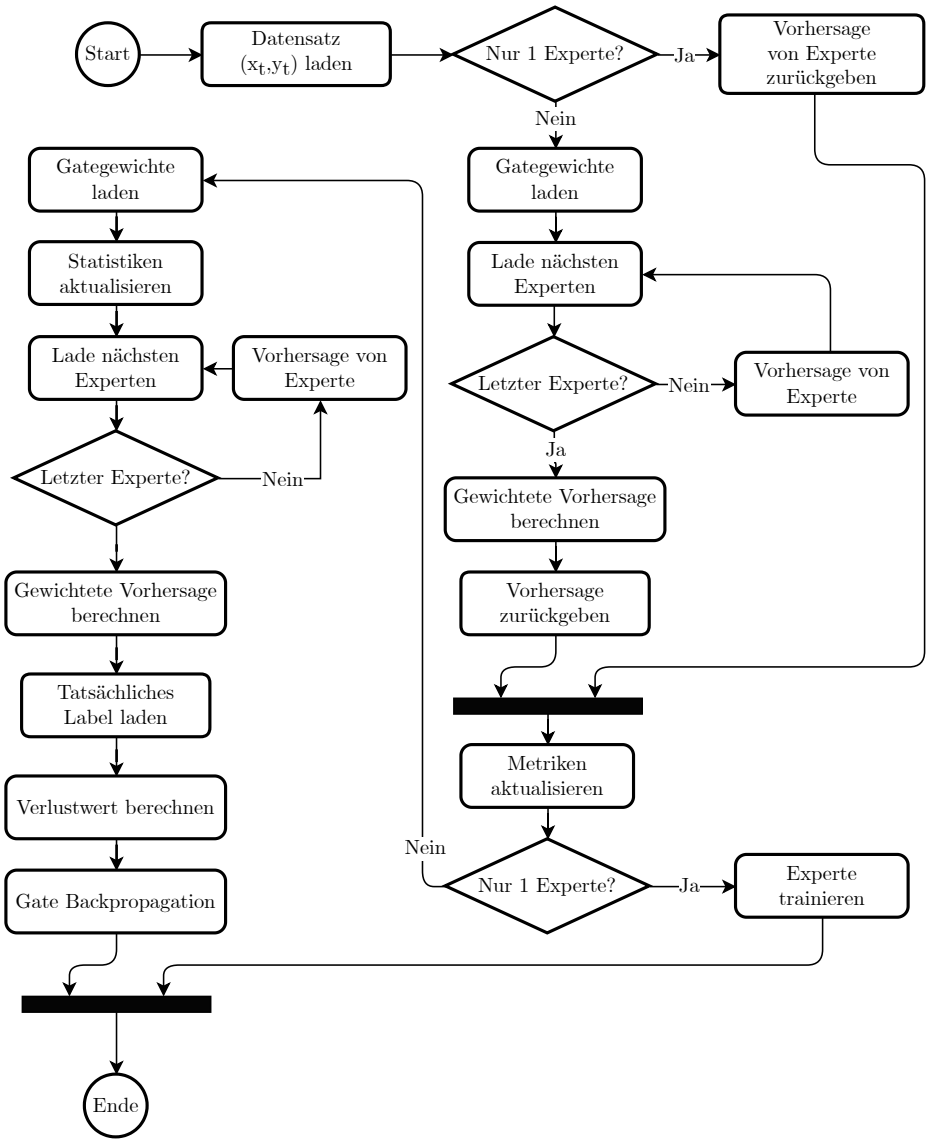


Abbildung A.2.: `river`MoE Aktivitätsdiagramm des Inferenz- und Trainingsschritts.

A.3. Modellparameter

Modell	Parameter	Standardwert	Aktueller Wert
DeepC/MoE	loss_fn	binary_cross_entropy	cross_entropy
	optimizer_fn	sgd	sgd
	lr	0,001	0,01
	output_is_logit	True	True
	device	cpu	cpu
LogR	optimizer		SGD
	loss		Log
	l2	0,0	0,0
	l1	0,0	0,0
	intercept_init	0,0	0,0
	intercept_lr	0,01	Constant
	clip_gradient	1000000000000,0	1000000000000,0
	initializer		Zeros
HTC	grace_period	200	200
	max_depth		3
	split_criterion	info_gain	info_gain
	delta	1e-07	1e-07
	tau	0,05	0,05
	leaf_prediction	nba	nba
	nb_threshold	0	0
	nominal_attributes		
	splitter		GaussianSplitter
	binary_split	False	False
	min_branch_fraction	0,01	0,01
	max_share_to_split	0,99	0,99
	max_size	100,0	100,0
	memory_estimate_period	1000000	1000000
	stop_mem_management	False	False
	remove_poor_attrs	False	False
	merit_preprune	True	True
+ HATC	drift_window_threshold	300	300
	drift_detector		ADWIN
HATC/MoE/DeepC	seed		42

Tabelle A.1.: Parameter für die Klassifikationsmodelle.

Modell	Parameter	Standardwert	Aktueller Wert
MoE Gate	loss_fn	binary_cross_entropy	mse
DeepR	loss_fn	mse	mse
DeepR/	optimizer_fn	sgd	sgd
MoE Gate	lr	0,001	0,001
	output_is_logit	True	True
	device	cpu	cpu
LogR	optimizer		SGD
	loss		Squared
	l2	0,0	0,0
	l1	0,0	0,0
	intercept_init	0,0	0,0
	intercept_lr	0,01	Constant
	clip_gradient	1000000000000,0	1000000000000,0
	initializer		Zeros
HTR	grace_period	200	200
	max_depth		3
	delta	1e-07	1e-07
	tau	0,05	0,05
	leaf_prediction	adaptive	adaptive
	leaf_model		LinearRegression
	model_selector_decay	0,95	0,95
	nominal_attributes		
	splitter		TEBSTSplitter
	min_samples_split	5	5
	binary_split	False	False
	max_size	500,0	500,0
	memory_estimate_period	1000000	1000000
	stop_mem_management	False	False
	remove_poor_attrs	False	False
	merit_preprune	True	True
+ HATR	drift_window_threshold	300	300
	drift_detector		ADWIN
BaseR	statistic		Mean
HATR/MoE/DeepR	seed		42

Tabelle A.2.: Parameter für die Regressionsmodelle.

A.4. Ergebnisse der Evaluationen

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,051s	0,005MiB	0,028ms	210,19	2708,872	-216065,538
21900	0,626s	0,005MiB	0,029ms	22,363	782,014	-8443,816
41975	1,243s	0,005MiB	0,03ms	14,409	564,883	-4180,359
62050	1,818s	0,005MiB	0,029ms	11,472	464,621	-2830,396
82125	2,389s	0,005MiB	0,029ms	10,031	403,878	-2112,995
102200	2,961s	0,005MiB	0,029ms	9,171	362,059	-1693,292
122275	3,537s	0,005MiB	0,029ms	8,656	331,021	-1392,648
142350	4,113s	0,005MiB	0,029ms	8,316	306,808	-1181,444
162425	4,687s	0,005MiB	0,029ms	7,978	287,235	-1030,817
182470	5,262s	0,005MiB	0,029ms	7,646	271,008	-919,481

Tabelle A.3.: Ergebnisse des Regressions-Experiments R.1, Modell: LinR.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,116s	0,044MiB	0,063ms	4,075	5,179	0,21
21900	0,845s	0,057MiB	0,039ms	5,205	7,013	0,321
41975	1,533s	0,036MiB	0,037ms	5,469	7,183	0,324
62050	2,208s	0,036MiB	0,036ms	5,421	7,136	0,332
82125	2,888s	0,036MiB	0,035ms	5,471	7,207	0,327
102200	3,567s	0,036MiB	0,035ms	5,51	7,246	0,321
122275	4,245s	0,036MiB	0,035ms	5,596	7,323	0,318
142350	4,926s	0,036MiB	0,035ms	5,689	7,42	0,308
162425	5,605s	0,036MiB	0,035ms	5,675	7,425	0,31
182470	6,333s	0,036MiB	0,035ms	5,597	7,369	0,32

Tabelle A.4.: Ergebnisse des Regressions-Experiments R.1, Modell: HTR.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,042s	0,004MiB	0,023ms	4,565	5,834	-0,002
21900	0,49s	0,004MiB	0,022ms	6,519	8,511	-0
41975	0,98s	0,004MiB	0,023ms	6,839	8,736	-0
62050	1,427s	0,004MiB	0,023ms	6,926	8,732	-0
82125	1,886s	0,004MiB	0,023ms	7,011	8,784	-0
102200	2,375s	0,004MiB	0,023ms	7,039	8,796	-0
122275	2,814s	0,004MiB	0,023ms	7,116	8,867	-0
142350	3,257s	0,004MiB	0,023ms	7,184	8,922	-0
162425	3,697s	0,004MiB	0,023ms	7,222	8,942	-0
182470	4,136s	0,004MiB	0,023ms	7,247	8,933	-0

Tabelle A.5.: Ergebnisse des Regressions-Experiments R.1, Modell: BaseR.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,632s	0,079MiB	0,346ms	4,06	5,19	0,207
21900	6,973s	0,092MiB	0,318ms	5,203	7,014	0,321
41975	13,161s	0,071MiB	0,314ms	5,468	7,183	0,324
62050	19,313s	0,071MiB	0,311ms	5,42	7,137	0,332
82125	25,564s	0,071MiB	0,311ms	5,46	7,195	0,329
102200	31,785s	0,071MiB	0,311ms	5,499	7,228	0,325
122275	37,882s	0,071MiB	0,31ms	5,586	7,304	0,322
142350	44,219s	0,071MiB	0,311ms	5,679	7,397	0,313
162425	50,553s	0,071MiB	0,311ms	5,666	7,406	0,314
182470	56,855s	0,071MiB	0,312ms	5,589	7,349	0,323

Tabelle A.6.: Ergebnisse des Regressions-Experiments R.1, Modell: SoftMoE.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,992s	0,127MiB	0,544ms	4,643	7,262	-0,553
21900	10,952s	0,098MiB	0,5ms	5,715	7,731	0,175
41975	20,541s	0,071MiB	0,489ms	5,991	7,869	0,188
62050	30,518s	0,071MiB	0,492ms	5,997	7,827	0,196
82125	40,754s	0,071MiB	0,496ms	6,064	7,885	0,194
102200	50,415s	0,071MiB	0,493ms	6,118	7,92	0,189
122275	60,418s	0,071MiB	0,494ms	6,207	7,996	0,187
142350	70,53s	0,071MiB	0,495ms	6,295	8,075	0,181
162425	80,123s	0,071MiB	0,493ms	6,303	8,086	0,182
182470	89,837s	0,071MiB	0,492ms	6,267	8,049	0,188

Tabelle A.7.: Ergebnisse des Regressions-Experiments R.2, Modell: SparseMoE Top($k = 1$).

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	0,935s	0,1MiB	0,512ms	11,516	202,016	-1200,659
21900	11,235s	0,061MiB	0,513ms	5,814	58,718	-46,61
41975	21,394s	0,061MiB	0,51ms	5,801	42,72	-22,914
62050	31,932s	0,061MiB	0,515ms	5,672	35,368	-15,407
82125	42,263s	0,061MiB	0,515ms	5,671	30,963	-11,425
102200	52,746s	0,061MiB	0,516ms	5,681	27,951	-9,098
122275	63,16s	0,061MiB	0,517ms	5,75	25,746	-7,431
142350	73,249s	0,061MiB	0,515ms	5,827	24,05	-6,266
162425	83,529s	0,061MiB	0,514ms	5,81	22,672	-5,428
182470	94,356s	0,061MiB	0,517ms	5,728	21,515	-4,802

Tabelle A.8.: Ergebnisse des Regressions-Experiments R.2, Modell: SparseMoE Top($k = 2$).

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
1825	1,09s	0,079MiB	0,597ms	32,284	788,809	-18320,206
21900	11,295s	0,093MiB	0,516ms	7,596	227,814	-715,672
41975	21,494s	0,071MiB	0,512ms	6,737	164,633	-354,167
62050	31,833s	0,071MiB	0,513ms	6,293	135,467	-239,695
82125	41,566s	0,071MiB	0,506ms	6,137	117,809	-178,87
102200	51,386s	0,071MiB	0,503ms	6,051	105,658	-143,288
122275	61,532s	0,071MiB	0,503ms	6,053	96,647	-117,799
142350	71,835s	0,071MiB	0,505ms	6,084	89,623	-99,899
162425	82,031s	0,071MiB	0,505ms	6,028	83,943	-87,126
182470	92,114s	0,071MiB	0,505ms	5,917	79,232	-77,677

Tabelle A.9.: Ergebnisse des Regressions-Experiments R.2, Modell: SparseMoE Top($k = 3$).

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0.013s	0.005MiB	0.029ms	0.79	0.786	0.744	0.764
5436	0.163s	0.005MiB	0.03ms	0.841	0.847	0.729	0.784
10419	0.313s	0.005MiB	0.03ms	0.853	0.857	0.799	0.827
15402	0.466s	0.005MiB	0.03ms	0.86	0.865	0.813	0.838
20385	0.615s	0.005MiB	0.03ms	0.856	0.867	0.794	0.829
25368	0.765s	0.005MiB	0.03ms	0.845	0.851	0.771	0.809
30351	0.915s	0.005MiB	0.03ms	0.84	0.844	0.762	0.801
35334	1.063s	0.005MiB	0.03ms	0.831	0.834	0.747	0.788
40317	1.261s	0.005MiB	0.031ms	0.834	0.834	0.754	0.792
45312	1.411s	0.005MiB	0.031ms	0.837	0.84	0.762	0.799

Tabelle A.10.: Ergebnisse des Klassifikations-Experiments C.1, Modell: LogR.

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,025s	0,127MiB	0,055ms	0,834	0,802	0,845	0,823
5436	0,207s	0,072MiB	0,038ms	0,826	0,792	0,758	0,775
10419	0,368s	0,072MiB	0,035ms	0,826	0,777	0,847	0,811
15402	0,53s	0,072MiB	0,034ms	0,819	0,764	0,861	0,809
20385	0,685s	0,072MiB	0,034ms	0,803	0,78	0,766	0,773
25368	0,838s	0,072MiB	0,033ms	0,784	0,787	0,678	0,729
30351	0,988s	0,072MiB	0,033ms	0,767	0,793	0,607	0,688
35334	1,139s	0,072MiB	0,032ms	0,75	0,797	0,543	0,646
40317	1,309s	0,073MiB	0,032ms	0,745	0,754	0,584	0,658
45312	1,471s	0,106MiB	0,032ms	0,747	0,742	0,619	0,675

Tabelle A.11.: Ergebnisse des Klassifikations-Experiments C.1, Modell: HTC.

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,01s	0,003MiB	0,022ms	0,874	0,86	0,864	0,862
5436	0,119s	0,003MiB	0,022ms	0,841	0,8	0,8	0,8
10419	0,227s	0,003MiB	0,022ms	0,838	0,815	0,815	0,815
15402	0,335s	0,003MiB	0,022ms	0,846	0,827	0,827	0,827
20385	0,444s	0,003MiB	0,022ms	0,847	0,825	0,825	0,825
25368	0,553s	0,003MiB	0,022ms	0,846	0,82	0,82	0,82
30351	0,663s	0,003MiB	0,022ms	0,854	0,827	0,827	0,827
35334	0,774s	0,003MiB	0,022ms	0,856	0,828	0,828	0,828
40317	0,884s	0,003MiB	0,022ms	0,855	0,827	0,827	0,827
45312	0,996s	0,003MiB	0,022ms	0,853	0,827	0,827	0,827

Tabelle A.12.: Ergebnisse des Klassifikations-Experiments C.1, Modell: BaseC.

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,178s	0,166MiB	0,393ms	0,885	0,838	0,928	0,881
5436	1,986s	0,111MiB	0,365ms	0,88	0,888	0,796	0,84
10419	3,761s	0,111MiB	0,361ms	0,885	0,891	0,842	0,866
15402	5,504s	0,111MiB	0,357ms	0,89	0,895	0,852	0,873
20385	7,245s	0,111MiB	0,355ms	0,885	0,895	0,836	0,864
25368	8,981s	0,111MiB	0,354ms	0,879	0,888	0,82	0,853
30351	10,77s	0,111MiB	0,355ms	0,881	0,886	0,825	0,855
35334	12,501s	0,111MiB	0,354ms	0,879	0,879	0,827	0,852
40317	14,216s	0,113MiB	0,353ms	0,876	0,871	0,829	0,849
45312	16,008s	0,146MiB	0,353ms	0,876	0,867	0,836	0,852

Tabelle A.13.: Ergebnisse des Klassifikations-Experiments C.1, Modell: SoftMoE.

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,295s	0,106MiB	0,65ms	0,766	0,726	0,783	0,753
5436	3,201s	0,226MiB	0,589ms	0,806	0,776	0,719	0,746
10419	6,092s	0,169MiB	0,585ms	0,81	0,801	0,755	0,777
15402	8,936s	0,169MiB	0,58ms	0,812	0,813	0,749	0,78
20385	11,723s	0,169MiB	0,575ms	0,797	0,809	0,704	0,753
25368	14,542s	0,112MiB	0,573ms	0,786	0,8	0,666	0,727
30351	17,371s	0,112MiB	0,572ms	0,784	0,798	0,655	0,719
35334	20,207s	0,112MiB	0,572ms	0,776	0,793	0,633	0,704
40317	22,88s	0,112MiB	0,568ms	0,773	0,771	0,653	0,707
45312	25,596s	0,112MiB	0,565ms	0,777	0,776	0,667	0,718

Tabelle A.14.: Ergebnisse des Klassifikations-Experiments C.2, Modell: SparseMoE Top($k = 1$).

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,263s	0,14MiB	0,581ms	0,828	0,782	0,865	0,821
5436	3,131s	0,169MiB	0,576ms	0,829	0,8	0,757	0,778
10419	5,979s	0,169MiB	0,574ms	0,831	0,806	0,812	0,809
15402	8,773s	0,111MiB	0,57ms	0,835	0,809	0,825	0,817
20385	11,846s	0,112MiB	0,581ms	0,829	0,814	0,791	0,802
25368	14,859s	0,112MiB	0,586ms	0,819	0,809	0,754	0,781
30351	17,831s	0,112MiB	0,587ms	0,816	0,811	0,737	0,772
35334	20,688s	0,112MiB	0,585ms	0,81	0,808	0,719	0,761
40317	23,568s	0,112MiB	0,585ms	0,811	0,801	0,732	0,765
45312	26,52s	0,112MiB	0,585ms	0,813	0,801	0,746	0,772

Tabelle A.15.: Ergebnisse des Klassifikations-Experiments C.2, Modell: SparseMoE Top($k = 2$).

Instanz	Zeit	Speicher	Zeit/Instanz	Accuracy	Precision	Recall	F1
453	0,275s	0,166MiB	0,607ms	0,857	0,826	0,87	0,847
5436	3,256s	0,111MiB	0,599ms	0,85	0,83	0,779	0,804
10419	6,409s	0,111MiB	0,615ms	0,857	0,841	0,831	0,836
15402	9,354s	0,111MiB	0,607ms	0,86	0,844	0,842	0,843
20385	12,266s	0,112MiB	0,602ms	0,851	0,847	0,806	0,826
25368	15,454s	0,112MiB	0,609ms	0,842	0,844	0,773	0,807
30351	18,548s	0,112MiB	0,611ms	0,842	0,847	0,763	0,803
35334	21,419s	0,112MiB	0,606ms	0,837	0,845	0,752	0,796
40317	24,529s	0,113MiB	0,608ms	0,838	0,838	0,762	0,798
45312	27,605s	0,146MiB	0,609ms	0,84	0,838	0,773	0,804

Tabelle A.16.: Ergebnisse des Klassifikations-Experiments C.2, Modell: SparseMoE Top($k = 3$).

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
125	0,033s	0,036MiB	0,268ms	9,824	10,948	-4,692
1500	0,352s	0,037MiB	0,235ms	2,164	2,747	0,721
2875	0,668s	0,037MiB	0,232ms	1,949	2,565	0,734
4250	0,991s	0,037MiB	0,233ms	1,827	2,394	0,77
5625	1,322s	0,037MiB	0,235ms	2,296	2,977	0,652
7000	1,682s	0,037MiB	0,24ms	2,219	2,871	0,625
8375	1,998s	0,037MiB	0,239ms	2,042	2,635	0,686
9750	2,313s	0,037MiB	0,237ms	1,735	2,347	0,782
11125	2,65s	0,037MiB	0,238ms	1,639	2,083	0,826
12500	2,974s	0,037MiB	0,238ms	1,707	2,32	0,803

Tabelle A.17.: Ergebnisse des Drift-Experiments D.1, Modell: DeepR.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
125	0,008s	0,211MiB	0,062ms	3,515	4,516	0,031
1500	0,214s	0,498MiB	0,143ms	2,008	2,536	0,762
2875	0,393s	0,522MiB	0,137ms	2,136	2,726	0,7
4250	0,545s	0,506MiB	0,128ms	1,978	2,563	0,737
5625	0,713s	0,342MiB	0,127ms	2,326	2,985	0,65
7000	0,919s	0,594MiB	0,131ms	2,249	2,905	0,616
8375	1,15s	0,659MiB	0,137ms	2,313	3,021	0,587
9750	1,358s	0,517MiB	0,139ms	1,991	2,588	0,735
11125	1,517s	0,481MiB	0,136ms	2,084	2,649	0,719
12500	1,651s	0,203MiB	0,132ms	2,408	3,066	0,655

Tabelle A.18.: Ergebnisse des Drift-Experiments D.1, Modell: HATR.

Instanz	Zeit	Speicher	Zeit/Instanz	MAE	RMSE	R2
125	0,077s	0,241MiB	0,615ms	3,623	4,539	0,021
1500	0,65s	0,367MiB	0,433ms	2,2	2,758	0,718
2875	1,199s	0,402MiB	0,417ms	2,174	2,75	0,694
4250	1,709s	0,238MiB	0,402ms	1,988	2,526	0,744
5625	2,168s	0,074MiB	0,385ms	2,183	2,788	0,695
7000	2,616s	0,074MiB	0,374ms	1,841	2,419	0,734
8375	3,064s	0,074MiB	0,366ms	2,078	2,683	0,675
9750	3,513s	0,074MiB	0,36ms	1,918	2,51	0,75
11125	3,961s	0,074MiB	0,356ms	1,921	2,448	0,76
12500	4,409s	0,074MiB	0,353ms	2,015	2,606	0,751

Tabelle A.19.: Ergebnisse des Drift-Experiments D.1, Modell: SoftMoE.

B. Hilfsmittel

Name	Beschreibung
Draw.io	Zeichnenprogramm, genutzt zur Erstellung aller Grafiken
LanguageTool	Prüfung von Rechtschreibung und Grammatik
Github Copilot	KI-Unterstützung für das Schreiben von Code (zum Beispiel Kommentare)

Tabelle B.1.: Übersicht der eingesetzten Hilfsmittel in dieser Arbeit

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg, 28. März 2025 Finn V. Dohrn