Boosting bei Transferlernen für Regression

Anne-Sophie Fleck

Geboren am 05.04.2002 in Aachen

10. Januar 2024

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jochen Garcke

Zweitgutachter: Dr. Bastian Bohn

Institut für Numerische Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

Wir fassen beim zweiten Mal nicht mehr an den heißen Herd, nehmen bei Regen eine Jacke mit und setzen uns mit neuem Wissen auseinander: All dies lässt sich zu einer zentralen menschlichen Eigenschaft zusammenfassen, nämlich der Fähigkeit zu lernen. Dies ist die Kompetenz, das Verhalten als Folge von Erfahrungen, gegebenenfalls langfristig, zu verändern (vgl. [HD12]). Bisher ist es uns noch nicht gelungen, diese Fähigkeit komplett auf Maschinen zu übertragen. Jedoch ist das maschinelle Lernen in den letzten Jahren zu einem immer weiter wachsenden Themengebiet geworden, welches versucht, das Prinzip „Lernen“ auf technologische Systeme zu übertragen. Letztes Jahr, nach dem Aufschwung von Chat-GPT, einem Chatbot-Tool, wurde vom US-Präsidenten sogar die erste umfassende Executive Order zu künstlicher Intelligenz unterzeichnet, was verdeutlicht, wie stark das maschinelle Lernen bereits in der Gesellschaft angekommen ist (vgl. [Web23]).

Unser Ziel ist es, Algorithmen zu entwickeln, die Muster in Daten erkennen können. Wir nennen diese Algorithmen Lernalgorithmen. Die Eingabe solcher sind Trainingsdaten, die zum Lernen verwendet werden, um präzise Vorhersagen für neue Daten zu treffen. Um möglichst viele Informationen zu finden und genauere Prognosen zu erstellen, brauchen wir meist eine hohe Menge an Daten, um Lernalgorithmen zu trainieren.

In der Praxis ist es nicht immer gewährleistet, dass die Trainingsdaten eine repräsentative Abbildung der realen Situation darstellen, die bei der Vorhersage durch das trainierte Modell adressiert werden soll. Wir betrachten zwei Beispiele aus dem Bereich des überwachten Lernens, in dem den Trainingsdaten Werte oder Kategorien (= Labels) zugeteilt werden, die der Lernalgorithmus möglichst gut prognostizieren soll.

Beispiel 1.1. *Das erste Beispiel hierfür sind autonome Fahrzeuge, die Gefahrensituationen erkennen und einschätzen sollen. Durch eine höhere Dichte an Menschen und Fahrzeugen stehen vermehrt Daten aus Städten zu komplizierten Verkehrslagen zur Verfügung. Im ländlichen Raum sind dementsprechend weniger Daten vorhanden, zudem wird es Verkehrslagen geben, zu denen es keine Daten im ländlichen Raum gibt. Jedoch unterscheidet sich der städtische und ländliche Raum in seiner Verkehrssituation, weswegen das Lernen der autonomen Fahrzeuge in der Stadt für das Fahren im ländlichen Raum nicht ausreichen wird. Darüber hinaus kann sich die Datenlage über die Zeit hinweg verändern, falls zum Beispiel neue Verkehrsregeln eingeführt werden.*

Beispiel 1.2. *Als zweites Beispiel betrachten wir die Herausforderung der Gefühlserkennung von Texten. Wir nehmen an, dass wir bereits eine Kategorisierung von Gefühlen in politischen Reden haben. Die Gefühle, in die wir die Texte einordnen, sind unsere Labels. Jedoch wird die Hypothese, die vom Lernalgorithmus auf Grundlage dieser Kategorisierung von politischen Reden aufgestellt wurde, nicht erfolgreich auf Bewertungen von Büchern übertragen werden können. Selbst wenn wir auch für die Bücher eine Gefühlskategorisierung haben würden, müssten wir den Lernalgorithmus aufs Neue trainieren, um spezifische Merkmale der Bücherbewertungen zu erfassen.*

Zusammenfassend stehen wir in der realen Welt vor vier Problemen:

1. Wir haben nicht immer genügend Daten gegeben.
2. Wir haben nicht immer vollständige und repräsentative Daten gegeben.
3. Wir haben nicht immer gelabelte Daten gegeben.
4. Mit neuen Daten muss oft ein Modell neu trainiert werden.

Um diese Probleme zu lösen, möchten wir das vom Menschen genutzte Prinzip des Transferlernens, dem Übertragen von Informationen aus ähnlichen Kontexten auf technologische Systeme übernehmen. Statt jeden Lernalgorithmus von Grund auf neu zu trainieren oder umfangreiche, vollständige und gelabelte Daten kostenspielig erstellen zu müssen, kann man bereits vorhandenes Wissen nutzen und auf ähnliche Probleme anwenden. Transferlernen ermöglicht dementsprechend eine Effizienzsteigerung und Anpassungsfähigkeit von Lernalgorithmen durch das Recyceln von Datensätzen. Darüber hinaus können Transferlernalgorithmen die reale Welt der Daten besser simulieren.

Eine der Schwierigkeiten beim Transferlernen ist die sinnvolle Anwendung des Datensatzes, aus dem wir Informationen ziehen möchten (= Quelldatensatz), um diese möglichst gewinnbringend auf unseren eigentlichen Datensatz (= Zieldatensatz) anzuwenden. Gehen wir davon aus, dass unser Lernalgorithmus sowohl den Quelldatensatz als auch den Zieldatensatz erhält. Nun stellen sich die Fragen: Welche Informationen aus dem Quelldatensatz sollen übertragen werden? Wie übertragen wir die Informationen aus dem Quelldatensatz?

Diese Fragen lassen sich zusammenfassen zu der Frage, wie die einzelnen Quelldaten beim Einbringen in den Lernalgorithmus gewichtet werden sollen. Insbesondere den Ansatz des Boostings, eine iterative Aktualisierung der Gewichtung der Quelldaten aufgrund der Qualität der Prognose, werden wir genauer betrachten. Dabei probieren wir zudem das Sampling aus, eine Vorbereitung der Datensätze, bei der die Gewichtung der Quelldaten vereinfacht werden soll. Wir werden in dieser Bachelorarbeit das vorgestellte Verfahren von Shrey Gupta, Jianzhao Bi, Yang Liu und Avani Wildani ([Gup+22]) analysieren und versuchen, die numerischen Experimente und deren Ergebnisse zu reproduzieren.

Übersicht

Die Arbeit ist in sechs Kapitel aufgeteilt. Nach einer Einführung in die Grundlagen des maschinellen Lernens werden wir uns in *Kapitel 2* dem Transferlernen widmen. In den beiden darauffolgenden Kapiteln werden Regressionsverfahren für Transferlernen vorgestellt. In *Kapitel 3* stellen wir gängige Algorithmen, sowohl mit Kernelansätzen als auch mit Boostingansätzen, dar. Wir werden dabei unter anderem auf die Schwierigkeit des negativen Transfers eingehen und Boosting sowie Kernel näher erläutern. In *Kapitel 4* stellen wir das Verfahren von Gupta, Bi, Liu und Wildani ([Gup+22]) vor. Wir werden dafür zwei Konzepte des Samplings einführen und die daraus resultierenden theoretischen Vorteile schildern. In *Kapitel 5* prüfen wir das im vorherigen Kapitel vorgestellte Verfahren und vergleichen dieses mit den in *Kapitel 3* vorgestellten Verfahren. Wir beschreiben hierbei die Implementierungen sowie die durchgeführten Experimente und untersuchen die Notwendigkeit des Samplings. Die

Ergebnisse werden danach diskutiert. Zuletzt fassen wir in *Kapitel 6* die Ergebnisse kurz zusammen, ziehen ein Fazit und geben einen Ausblick.

Eigener Beitrag

- Aufarbeitung der Grundlagen des Transferlernens
- Präsentation einer Zusammenführung der Entwicklung von Boostingalgorithmen bis `S-TrAdaBoost.R2`
- Zusammenfassung weiterer Verfahren für Transferlernen
- Eigenständige Einarbeitung in den zur Verfügung gestellten Programmiercode von [Gup+22] sowie Untersuchung von Unterschieden zwischen deren Artikel und Programmiercode
- Anpassung des Programmiercodes an den Artikel, insbesondere eigene Implementierung von Variance Sampling (vgl. Kapitel 4.1.2)
- Experimente mit verschiedenen Arten von Samplings sowie Interpretation der Ergebnisse
- Analyse der Datensätze

Notation

Wir möchten an dieser Stelle eine Übersicht über die Konvention der Bennenung von Variablen zum Nachschlagen geben.

X	Merkmalsraum
Y	Labelraum
L	Lernalgorithmus
K	Kostenfunktion
(X^Q, Y^Q)	Quelldaten bestehend aus Datenpunkten mit Labels
(X^Z, Y^Z)	Zieldaten bestehend aus Datenpunkten mit Labels
p	Verteilung[1] der Daten
$p(y\|x)$	Verteilung von y bedingt x
$w(x, y)$	Gewichte für die Quelldaten
$[n]$	Menge $\{1, \ldots, n\}$

[1] An dieser Stelle sei angemerkt, dass im Bereich des maschinellen Lernens Verteilung und Dichte synonym genutzt werden (vgl. [Van16]) und wir dies übernehmen.

2 Maschinelles Lernen

In diesem Abschnitt werden wir uns mit den Grundlagen des Transferlernens beschäftigen. Dabei werden wir mit grundlegenden Definitionen aus dem maschinellen Lernen, insbesondere dem überwachten maschinellen Lernen, einsteigen. Außerdem werden wir uns dem Problem des Datenmangels widmen und uns mit einer der Lösungsideen, nämlich Transferlernen, beschäftigen.

2.1 Grundlegende Definitionen

Betrachten wir zunächst die Grundlagen, das Lernen von Algorithmen anhand von Daten. Die Daten können verschiedene Dimensionen haben und aus verschiedenen mathematischen Objekten bestehen. Der Raum, in dem sich die Datenpunkte befinden, nennen wir Merkmalsraum. Wir werden zur Vereinfachung in dieser Bachelorarbeit annehmen, dass Datenpunkte Vektoren im $\mathbb{R}^D$ entsprechen für eine beliebige Dimension D. Der Merkmalsraum ist im Allgemeinen nicht eindeutig, wie wir an dem nächsten Beispiel sehen können.

Beispiel 2.1. *Betrachten wir als Beispiel Abbildung 1. Die Datenpunkte lassen sich durch drei Eigenschaften unterscheiden: ihre beiden Koordinaten und die Farbe des Punktes. Die Farben können wir mit Werten in* $\mathbb{R}$ *darstellen, beispielsweise* -1 *und* 1*. Nehmen wir alle drei Eigenschaften, um die Datenpunkte zu beschreiben, so ist unser Merkmalsraum* $\mathbb{R}^3$*. Wir können beispielsweise auch nur die Koordinaten verwenden, um je nach Koordinaten die Farbe vorherzusagen. Dann entspräche der Merkmalsraum* $\mathbb{R}^2$*.*

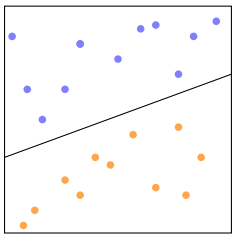

Abbildung 1: Beispiel von Datenpunkten

Im Allgemeinen werden die Datenpunkte aus einer Verteilung generiert. Eine formale Definition einer Verteilung werden wir hier nicht einführen, da dies einen längeren Exkurs in die Wahrscheinlichkeitstheorie benötigen würde. Eine solche formale Definition ist aber in [Mur22] zu finden. Stattdesseb reicht für unsere Zwecke die intuitive Vorstellung aus, dass eine Verteilung angibt, wie wahrscheinlich bestimmte Werte angenommen werden. In der Regel werden wir die genaue Verteilung der Daten nicht kennen. Wir nehmen jedoch grundlegend an, dass die Daten unabhängig und identisch verteilt gezogen werden, also unabhängig voneinander sind und aus derselben Verteilung stammen.

Wir unterscheiden im maschinellen Lernen zwischen verschiedenen Arten von Daten:

Definition 2.2 (Lernalgorithmus / Trainingsdaten / Testdaten / Merkmalsraum / Attribute (vgl. [MRT18]))**.**
Ein **Lernalgorithmus** ist ein Algorithmus, der Muster in den Daten „analysiert“ und Hypothesen entwickelt, die es ihm ermöglichen, Vorhersagen oder Entscheidungen für bisher unbenutzte Daten zu treffen. Die Daten, die für die Analyse verwendet werden, heißen **Trainingsdaten**. Unbetrachtete Daten, die dafür verwendet werden, eine Aussage über die Hypothese zu treffen, nennt man **Testdaten**. Der Raum, in dem sich die Daten befinden, heißt **Merkmalsraum**. **Attribute** sind die Eigenschaften, die verwendet werden, um einen Datenpunkt aus dem Trainings- oder Testdatenset zu beschreiben.

Die Hypothese kann als eine Abbildung aus dem Merkmalsraum in einen weiteren Raum betrachtet werden. Insbesondere entspricht die Anzahl an Attributen gerade der Dimension der Datenpunkte. Wir werden annehmen, dass die Menge an Test- und Trainingsdaten endlich ist.

Der Grundaufbau setzt sich wie folgt zusammen:

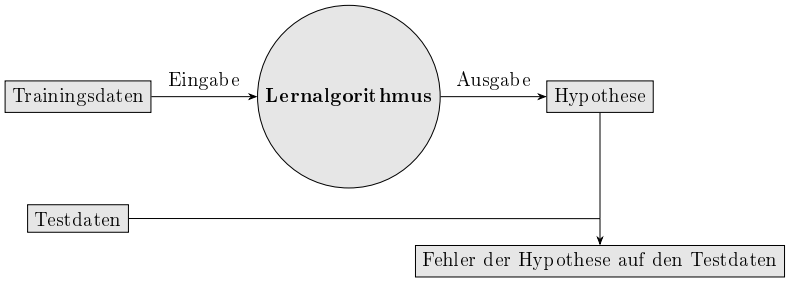

Abbildung 2: Modell eines Lernalgorithmus

Dabei gibt es unterschiedliche Arten des Lernens. Wir werden uns im Folgenden auf das überwachte Lernen einschränken. Dafür benötigen wir gelabelte Daten:

Definition 2.3 (Label (vgl. [MRT18]))**.**
Jedem Datenpunkt wird eine Ausgabe assoziiert, diese nennen wir **Label**.

Beispiel 2.4. *Wir möchten eine Aussage über den Verlauf von Eierstockkrebs anhand von Körpereigenschaften zum Zeitpunkt der Diagnose treffen. Dafür haben wir zwei gelabelte Datensätze über den Verlauf von an Eierstockkrebs erkrankten Frauen gegeben, die wir in Trainingsdaten und Testdaten unterteilen. Die Labels sind hierbei die Anzahl an Jahren, die die Frauen nach der Diagnose noch gelebt haben. Als Attribute wählen wir die Ausprägung des Eierstockkrebs, das Alter bei der Diagnose, die gewählte Therapieform und Vorerkrankungen. Nun erstellen wir anhand eines Lernalgorithmus, der von den Trainingsdaten lernt, eine Hypothese, wie der Verlauf allgemein bei erkrankten Frauen je nach ihren Werten bei den Attributen aussehen wird. Diese Hypothese testen wir auf dem zweiten Datensatz, den Testdaten, des Verlaufs von an Eierstockkrebs erkrankten Frauen.*

Bemerkung 2.5. Man kann ein Label auch als ein Art Attribut des Datenpunkts auffassen, da es eine Eigenschaft des Datenpunkts ist. Jedes Attribut kann zudem als Label gewählt werden, wobei je nach Datensatz dies mehr oder weniger sinnvoll sein kann. Betrachten kann man als Beispiel 2.1. Hierbei können die Koordinaten als Attribute dienen, um die Farben als Labels vorherzusagen. Beispielsweise könnten wir auch versuchen, eine der Koordinaten anhand der Farbe und der anderen Koor-

dinate vorherzusagen. Wir werden uns zur Vereinfachung darauf beschränken, dass das Label dem zu prognostizierenden Attribut entspricht.

Beim überwachten Lernen wird nun anhand gelabelter Trainingsdaten durch einen Algorithmus versucht, eine möglichst gute Prognose der Labels der Trainingsdaten zu finden, um diese auf andere Daten anwenden zu können. Wir messen dabei die Leistung des Lernalgorithmus anhand der Vorhersagegenauigkeit der Hypothese. Wir notieren dies wie folgt:

Notation 2.6. Gegeben einen Merkmalsraum $X \subseteq \mathbb{R}^D$, einen Labelraum $Y \subseteq \mathbb{R}$ und Trainingsdaten $(X_{train}, Y_{train}) = \{(x_1, y_1), ..., (x_m, y_m)\} \subseteq X \times Y$, erstellen wir mithilfe eines Lernalgorithmus L eine Hypothese $h_f : X \to Y$. Die Vorhersagegenauigkeit der Hypothese und somit auch der Qualität des Lernalgorithmus wird mithilfe der Testdaten $(X_{test}, Y_{test}) = \{(x_1^{test}, y_1^{test}), ..., (x_w^{test}, y_w^{test})\} \subseteq X \times Y$ berechnet.

Um die Leistung des Lernalgorithmus anhand der Hypothese messen zu können, vergleichen wir die prognostizierten Labels durch die Hypothese mit den tatsächlichen Labels der Testdaten. Dabei gibt es unterschiedliche Arten diese zu vergleichen. Dafür definieren wir den Begriff einer Kostenfunktion:

Definition 2.7 (Kostenfunktion (vgl. [MRT18]))**.**
Seien die Menge der Label der Testdaten durch Y_{test} und die Menge der vom Lernalgorithmus prognostizierten Label durch Y_{prog} gegeben. Die **Kostenfunktion** K ist eine Abbildung $K : Y_{test} \times Y_{prog} \to \mathbb{R}_+$ und misst den Unterschied zwischen dem tatsächlichen Label und dem prognostizierten Label eines Testdatenpunkts.

Beispiel 2.8. *Betrachte als Beispiel mit* $Y_{test}, Y_{prog} = \{0, 1\}$*. Eine mögliche Kostenfunktion ist* $K : Y_{test} \times Y_{prog} \to \mathbb{R}_+$ *mit* $K(y, y') = |y - y'|$*.*

Im Allgemeinen ist das Minimieren einer Kostenfunktion nicht immer das Ziel. In Kapitel 5 werden wir eine Kostenfunktion kennenlernen, bei der höhere Werte besser sind. Zur Vereinfachung werden wir jedoch in diesem Kapitel annehmen, dass eine besserer Wert der Kostenfunktion kleiner ist. Berechnen wir eine gewählte Kostenfunktion auf den tatsächlichen und den prognostizierten Labels der Testdaten auf Basis der erstellten Hypothese, so kann dies als Fehler der Hypothese gesehen werden. Unser Ziel ist es dementsprechend im Allgemeinen, eine Hypothese zu finden, die unsere Kostenfunktion möglichst auf den Trainingsdaten minimiert. Wir betrachten dabei Regressionsprobleme, bei denen die Labels eine beliebige reelle Zahl sind.

Definition 2.9 (Regression / Klassifikation (vgl. [MRT18]))**.**
Der Lernalgorithmus versucht bei **Regressionsproblemen**, gegeben ein Trainingsdatensatz mit Labels $Y \in \mathbb{R}$, eine möglichst gute Prognose von Y zu finden. **Klassifikation** ist dabei der Fall, wenn nur diskrete Ausgabewerte betrachtet werden.

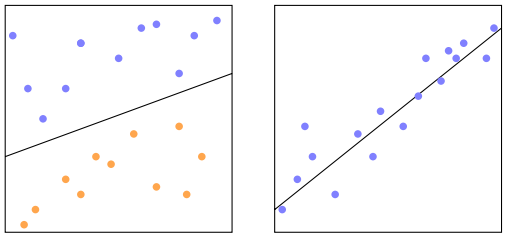

Abbildung 3: Beispiel für Klassifikation (links): Trennen der blauen und orangenen Punkte
Beispiel für lineare Regression (rechts): Versuch, die Punkte mithilfe einer linearen Abbildung möglichst gut zu approximieren

Dabei gibt es unterschiedliche Arten von Lernalgorithmen, die unterschiedlich gut mit komplexen Datensets umgehen können. Ein Beispiel für einen Lernalgorithmus sind Entscheidungsbäume, die eine Klassifizierung anhand einer Reihe von Wahlmöglichkeiten vornehmen.

Definition 2.10 (Entscheidungsbaum / Tiefe (vgl. [SB14])).
Ein **Entscheidungsbaum** entspricht einem Baum, bei dem beginnend mit der Wurzel in jedem Knoten eine Entscheidung getroffen wird und die Blätter des Baums aus den möglichen Labels bestehen. Die **Tiefe eines Entscheidungsbaums** ist definiert als die höchste Anzahl an Entscheidungen von der Wurzel bis zu einem Blatt.

Entscheidungsbäume entsprechen einer Art Partition des Raums wie im folgendem Beispiel zu sehen:

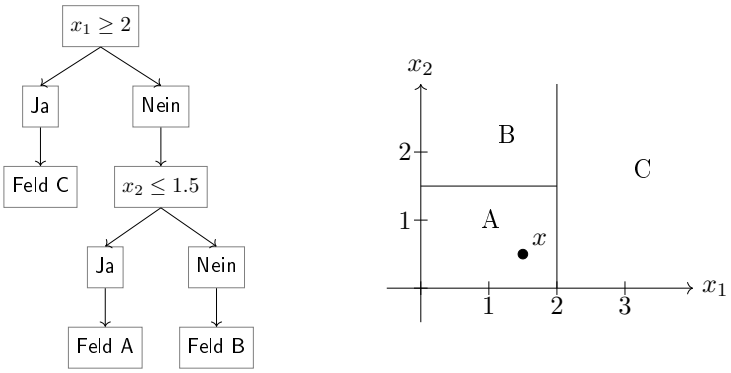

Abbildung 4: Beispiel für Entscheidungsbaum: In welchem Feld liegt x?

Wenn man nun bei Entscheidungsbäumen nur eine geringe Anzahl an Entscheidungen zulässt, lassen sich komplexe Abbildungen nur schwer approximieren. Lernalgorithmen, die geringfügig besser sind als zufälliges Raten, nennen wir im Folgenden schwache Lernalgorithmen.

Definition 2.11 (Schwacher Lernalgorithmus (vgl. [Kes20] und [SB14])).
Ein Lernalgorithmus heißt **schwach**, wenn er als Eingabe beliebige Gewichte $w_1, \cdots, w_m \geq 0$ mit $\sum_i w_i = 1$ und Trainingsdaten (X_{train}, Y_{train}) erhält und eine Hypothese h_w berechnet, sodass $\sum_{i:h_w(x_i)\neq y_i} w_i \leq \frac{1}{2} - \gamma$ für ein festes $\gamma > 0$.

Die Gewichte definieren insbesondere eine Wahrscheinlichkeitsverteilung über den Trainingsdaten.

Bemerkung 2.12. Definition 2.11 ist für den Fall der Klassifikation formuliert. Da wir die Definition für Boosting benutzen und Boosting zunächst für Klassifikation entstanden ist, reicht 2.11 für unsere Zwecke aus. Jedoch lässt sich die Definition ähnlich für Regression übertragen. Es gibt darüber hinaus allgemeinere Definitionen, die sowohl für Klassifikation als auch für Regression geeignet sind, bei denen die obere Schranke mit einer gewissen Wahrscheinlichkeit eingehalten werden soll. Solche finden sich zum Beispiel in [SB14] und [MRT18].

Haben wir jedoch einen Lernalgorithmus, der komplexere Hypothesen berechnet als für ein Datensatz benötigt, kann Overfitting entstehen.

Definition 2.13 (Overfitting (vgl. [Mur22]))**.**
Sei der Fehler einer Hypothese h einer Datenmenge (X_s, Y_s) definiert als $\frac{1}{s}\sum_{i=1}^{s} K(h(x_i), y_i)$ für eine Kostenfunktion K. Dann sprechen wir von **Overfitting**, wenn der Fehler auf den Trainingsdaten deutlich kleiner ist als der Fehler auf den Testdaten.

Beispiel 2.14. *Betrachten wir als Beispiel Abbildung 5. Wir möchten eine Hypothese finden, die die Datenpunkte (blau) möglichst gut approximiert und außerdem auf neuen Daten (orange) gute Werte liefern wird. Das zweite Bild versucht eine Gerade zu finden, die den Abstand zu allen Punkten minimiert. In der dritten Abbildung wurde eine Funktion gefunden, die durch alle Datenpunkte geht. Jedoch ist der Fehler der Testdaten bei dieser Funktion deutlich höher als der Fehler auf den Trainingsdaten, der* 0 *beträgt und somit ist dies ein Beispiel für Overfitting. Im Gegensatz dazu, erzeugt die Gerade weniger Diskrepanz zwischen dem Fehler auf den Trainingsdaten und dem auf den Testdaten.*

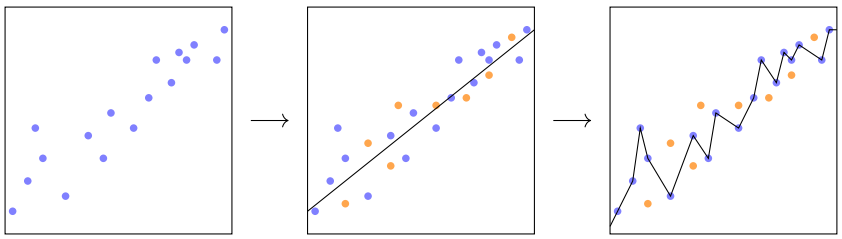

Abbildung 5: Beispiel für Overfitting: Trainingsdaten (blau) und Testdaten (orange)

Das Problem von Overfitting tritt häufig im Bereich des maschinellen Lernens auf, weswegen wir im weiteren Verlauf einen Mittelweg zwischen zu einfachen Algorithmen und Overfitting finden müssen.

2.2 Transferlernen

Damit unser Lernalgorithmus Hypothesen erstellt, die eine gewählte Kostenfunktion möglichst stark minimieren, brauchen wir eine hohe Menge an gelabelten Trainingsdaten, die möglichst gut die Verteilung, aus der sie gezogen sind, repräsentieren. In Beispiel 1.1 und Beispiel 1.2 haben wir bereits festgestellt, dass dies nicht immer der Fall ist und aufwendige Konsequenzen hat, wie das Neutrainieren eines Modells aufgrund von einer anderen Datenlage. Um dies zu verhindern, betrachten wir

das Konzept „Transferlernen“ aus der Psychologie, nämlich der Übertragung von Informationen von vergleichbaren Kontexten (vgl. [HD12]). Dies übernehmen wir im maschinellen Lernen, indem Informationen aus anderen bereits bestehenden Datensätzen genutzt werden, um bessere Prognosen zu erhalten.

Beispiel 2.15. *Wir möchten wie in Beispiel 2.4 eine Aussage darüber treffen, wie der Verlauf von Eierstockkrebs bei Frauen aussieht. Da jedoch die meisten erkrankten Frauen 40 Jahre oder älter sind (vgl. [Höw23]), haben wir nur wenig Daten über junge Frauen mit der Diagnose gegeben und können somit nur wenig Aussagen bezüglich jüngerer Frauen treffen. Jedoch könnten wir versuchen, Informationen über den Verlauf der älteren Frauen auf den Verlauf der jüngeren Frauen zu übertragen.*

Unser Ziel besteht nun darin, einen Lernalgorithmus zu finden, der als Eingabe mehrere Datensätze bekommt und hilfreiche Informationen aus diesen ziehen kann, um das Trainieren auf den Trainingsdaten und das Erstellen einer Hypothese zu unterstützen. Wir werden uns im weiteren Verlauf auf einen zusätzlichen Datensatz zu den Trainingsdaten und den Testdaten beschränken. Wir nutzen dabei die folgenden Bezeichnungen:

Definition 2.16 (Quell-und Zieldaten (vgl. [PY10]))**.**
Die Daten (X^Q, Y^Q), aus denen die Informationen stammen, bezeichnen wir als **Quelldaten**. Die Daten (X^Z, Y^Z), auf die das Wissen angewendet wird, heißen **Zieldaten**.
Es ergibt sich damit die folgende Definition:

Definition 2.17 (Transferlernen (vgl. [PY10]))**.**
Wir haben Zieldaten, gezogen aus einer Verteilung $p^Z(x, y)$, aus einem Merkmalsraum $X^Z \subseteq \mathbb{R}^{D^Z}$ sowie einem Labelraum Y^Z gegeben. Analog haben wir Quelldaten gegeben. **Transferlernen** versucht das Lernen auf den Zieldaten zu verbessern, wobei $X^Z \neq X^Q$, $p^Z(x^Z) \neq p^Q(x^Q)$, $Y^Z \neq Y^Q$ oder $p^Z(y^Z|x^Z) \neq p^Q(y^Q|x^Q)$ gilt.

Transferlernen bezieht sich dementsprechend auf den Fall, wenn mehr als ein Datensatz zum Trainieren verwendet wird, jedoch sich die Datensätze in den Verteilungen oder in den Räumen unterscheiden.

Unser Modell sieht nun wie folgt aus:

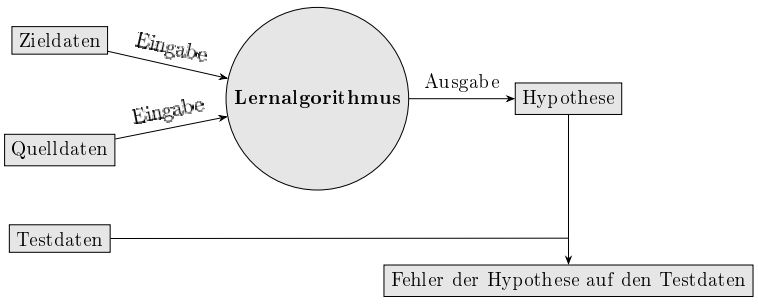

Abbildung 6: Modell von einem Lernalgorithmus mit Transferlernen

Anhand der Eigenschaft, in der sich die Zieldaten und Quelldaten unterscheiden, kann man zwischen verschiedenen Bereichen im Transferlernen unterscheiden: instanzbasiertes, attributsbasiertes und parameterbasiertes Transferlernen, sowie Transfer von relationalem Wissen [PY10]. Wir konzentrieren uns im weiteren Verlauf auf das instanzbasierte Transferlernen. Die Situation ist dabei die Folgende:

Definition 2.18 (Instanzbasiertes Transferlernen (vgl. [PY10]))**.**
Wir haben Quelldaten X^Q, gezogen aus der Verteilung $p^Q(x, y)$, und Zieldaten X^Z gezogen aus der Verteilung $p^Z(x, y)$, gegeben. Insbesondere sei $p^Q(x, y) \neq p^Z(x, y)$ für x aus einem Merkmalsraum X und einem Labelraum Y . Das Nutzen von einzelnen gewichteten Dateninstanzen aus dem Quelldatensatz zusätzlich zu den Zieldaten für die Erstellung einer Hypothese durch einen Lernalgorithmus nennt man **instanzbasiertes Transferlernen**.

Intuitiv beschreiben die unterschiedlichen Verteilungen den Unterschied zwischen den Datensätzen. Im Allgemeinen nehmen wir beim instanzbasierten Transferlernen an, dass wir deutlich mehr Quelldatenpunkte als Zieldatenpunkte gegeben haben. Das entspricht oft auch realen Situationen, in denen neue gelabelte und vollständige Daten teuer sind (vgl. [Gup+22], [PY10]). Auch wenn die Verteilungen beim instanzbasierten Transferlernen unterschiedlich sind, ist es wichtig, dass es eine Überschneidung von Quell- und Zieldaten gibt (vgl. [Van16]). Sonst können keine Informationen von den Quelldaten übernommen werden.

Unsere eigentlichen Trainingsdaten entsprechen beim instanzbasierten Transferlernen nun den Zieldaten und zusätzlich haben wir als Eingabe des Lernalgorithmus Quelldaten gegeben. Weiterhin haben wir einen Testdatensatz gegeben, auf welchem wir nach dem Lernen unsere Hypothese evaluieren.

Beispiel 2.19. *Wir betrachten Beispiel 2.15. Unser Zieldatensatz ist hierbei die Menge an Daten zu den jungen Frauen, die an Eierstockkrebs erkrankt sind. Wie oben erwähnt, haben wir hier nur eine geringe Datenmenge zu Verfügung. Die Menge an Daten von den erkrankten älteren Frauen sind unsere Quelldaten. Instanzbasiertes Transferlernen ist hierbei, wenn wir die Daten der erkrankten jungen Frauen und die Daten der erkrankten älteren Frauen als Eingabe in unseren Lernalgorithmus eingeben und dabei die Daten einer Person individuell betrachten. Wichtig ist dabei, dass zwischen den Daten der erkrankten älteren Frauen und der erkrankten jungen Frauen Überschneidungen gibt. Wenn beispielsweise alle erkrankten älteren Frauen eine Therapieform haben, jedoch alle jüngeren erkrankten Frauen eine andere, so wäre es schwierig, Informationen von den älteren Frauen zu übertragen. Außerdem sollten stärker die erkrankten älteren Frauen betrachtet werden, die ähnliche Verlaufe wie die jungen erkrankten Frauen haben. Die Frage ist dabei, wie man die ähnlichen Verläufe bestimmt und wie stark die Gewichte jeweils gewählt werden sollen.*

Eine Schwierigkeit beim instanzbasierten Transferlernen ist es, wie in Beispiel 2.19 erwähnt, geeignete Gewichte für den Quelldatensatz zu finden, um möglichst viele und hilfreiche Informationen aus dem Quelldatensatz zu ziehen. Insbesondere werden beim instanzbasierten Transferlernen die Quelldatenpunkte individuell betrachtet und ihnen wird ein individuelles Gewicht zugeteilt, wieviel Einfluss der einzelne Quelldatenpunkt haben soll. Das heißt, dass der Lernalgorithmus nicht nur eine Hypothese erstellt, sondern auch geeignete Gewichte für die Quelldaten finden muss.

Wir können die besten Gewichte für die Quelldatenpunkte und das Finden der besten Prognose wie folgt mathematisch beschreiben (vgl. [Van16] und [GV14]):

Sei X ein Merkmalsraum und Y ein Labelraum. Der Zieldatensatz ist gegeben durch $(X^Z, Y^Z) \subseteq X \times Y$ und der Quelldatensatz ist gegeben durch $(X^Q, Y^Q) \subseteq X \times Y$, die jeweils aus einer Verteilung $p^Z(x, y)$ bzw. $p^Q(x, y)$ gezogen worden sind. Die

Verteilung der Zieldaten lässt sich für ein $x \in X, y \in Y$ darstellen als

$$p^Z(x,y) = \frac{p^Z(x,y)}{p^Q(x,y)} p^Q(x,y).$$

Wir können die Gewichte der Quelldaten dementsprechend schreiben als

$$w(x,y) := \frac{p^Z(x,y)}{p^Q(x,y)}. \tag{2.1}$$

Da wir im Allgemeinen jedoch die Verteilungen der Datensätze nicht kennen, können wir die Gewichte nicht exakt bestimmen. Stattdessen versuchen wir $w(x,y)$ durch einen Ausdruck zu approximieren, der unabhängig von beiden Verteilungen ist. Eine Möglichkeit ist die Kullback-Leibler-Divergenz, die wir in Kapitel 3.1 kennenlernen werden.

Die beste mögliche Prognose y' für einen neuen Datenpunkt x' nach Definition einer Verteilung auf Basis der Trainingsdaten ist

$$y' = \arg\max_{y \in Y} \left((p^Z(x', y) \right) \tag{2.2}$$

Auch wenn wir die Verteilungen nicht kennen, können wir beide Verteilungen wie folgt schreiben:

$$p(x,y) = p(y|x)p(x).$$

Damit und mithilfe von Gleichung 2.1, lässt sich die Gleichung 2.2 umschreiben zu:

$$\begin{aligned} y' &= \arg\max_{y \in Y} \left((p^Z(x', y) \right) \\ &= \arg\max_{y \in Y} \left(\frac{p^Z(x', y)}{p^Q(x', y)} p^Q(y|x') p^Q(x') \right) \\ &= \arg\max_{y \in Y} \left(w(x', y) p^Q(y|x') p^Q(x') \right) \end{aligned}$$

Da die Wahl von y keinen Einfluss auf den Wert von $p^Q(x')$ hat, folgt

$$y' = \arg\max_{y \in Y} \left(w(x', y) p^Q(y|x') \right) \tag{2.3}$$

Finden wir dementsprechend geeignete Gewichte für die Quelldaten, können wir nur auf Basis der Quelldaten beziehungsweise der Verteilung der Quelldaten, die Prognosen für neue Daten finden, bei denen wir annehmen, dass sie aus der gleichen Verteilung stammen wie die Zieldaten. Jedoch müssen wir zudem Formulierungen finden, bei denen auch die Verteilung der Quelldaten verschwindet. Wir werden uns in Kapitel 3 und in Kapitel 4 mit verschiedenen Möglichkeiten beschäftigen geeignete Gewichte zu finden.

An dieser Stelle sei angemerkt, dass wir davon ausgehen, dass unsere Daten jeweils repräsentativ für ihre Verteilung sind. Repräsentativ bedeutet in dem Sinne, dass wir genügend Daten gegeben haben, um Aussagen über die Verteilung zu machen und insbesondere richtige Daten, das heißt Daten mit möglichst wenig Verzerrungen beispielsweise aufgrund von Messfehlern.

3 Andere Verfahren für Instanzbasiertes Transferlernen

Das Ziel von Kapitel 3 und Kapitel 4 ist die Herleitung und Entwicklung eines Lernalgorithmus, der in der Lage ist, auf Basis der Erfahrungen aus den Quelldaten zu lernen und dieses erlangte Wissen auf die Zieldaten anzuwenden, um die Genauigkeit der Prognose zu steigern. Dieser Ansatz kann als eine Art „Meta-Lernen“ betrachtet werden, da im Lernalgorithmus nicht nur eine Hypothese erstellt werden soll, sondern auch entschieden wird, wie das Wissen aus den Quelldaten sinnvoll genutzt werden kann. Zusammenfassend möchten wir einen Lernalgorithmus finden, der Transferlernen möglichst effektiv nutzt. Wichtig ist hierbei, dass kein negativer Transfer zustande kommt, also die Übertragung von Wissen von den Quelldaten, welches die Kostenfunktion vergrößert. Auf diesen Punkt werden wir in Kapitel 3.3 näher eingehen.

Wir werden uns in diesem Kapitel mit Verfahren zur Lösung des dargestellten Problems beschäftigen. Dabei werden wir zwischen zwei Herangehensweisen unterscheiden. Die eine Art aktualisiert jedes Gewicht in verschiedenen Iterationen einzeln, da dabei sogenanntes Boosting benutzt wird, die andere Art berechnet die Gewichte für alle Quelldatenpunkte gleichzeitig in einem Durchgang und nutzt dabei sogenannte Kernel. Wir betrachten zunächst die Ansätze mit Kernel.

3.1 Ansätze mit Kernel

Im Folgenden betrachten wir gängige Verfahren, die für Regressionsprobleme mit instanzbasiertem Transferlernen geeignet sind. In den folgenden Verfahren werden die Gewichte für alle Quelldatenpunkte gemeinsam bestimmt und daraufhin angewendet.

Zunächst werden wir die Begriffe Einbettung und Kernel einführen. Betrachten wir dazu folgendes Beispiel:

Beispiel 3.1. *Wir haben zwei Datensätze gegeben, die in Abbildung 7 visualisiert sind. Die Datensätze haben als Label blau und orange. Wir wollen nun eine lineare Klassifikation durchführen, also mithilfe einer linearen Abbildung die blauen von den orangenen Punkten trennen. Wie links in Abbildung 7 zu erkennen ist, wird dies nicht exakt möglich sein. Wir können nun jedoch die Datenpunkte in eine höhere Dimension abbilden. Bisher lassen sich alle Datenpunkte als* $(x_1, x_2) \in \mathbb{R}^2$ *beschreiben. Wir wählen in diesem Fall zur Abbildung in eine höhere Dimension die folgende Abbildung:* $\psi : \mathbb{R}^2 \to \mathbb{R}^3, \psi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$*. Da die blauen Punkte in der Nähe des Nullpunkts liegen, die orangen Punkte jedoch weiter entfernt sind, wird es nun in* $\mathbb{R}^3$ *möglich sein, die Punkte mithilfe einer Ebene zu trennen. Formal finden wir nun einen linearen Klassifikator* $h_\psi : \mathbb{R}^3 \to \{-1, 1\}$*. Wir können diesen linearen Klassifikator nun auf* $\mathbb{R}^2$ *übertragen, indem wir*

$$h(x) = \begin{cases} +1, & \textit{falls } \psi(x) \textit{ über oder in der Ebene liegt} \\ -1, & \textit{falls } \psi(x) \textit{ unter der Ebene liegt} \end{cases}$$

als Hypothese wählen. Das liefert uns die rechte Abbildung von Abbildung 7.

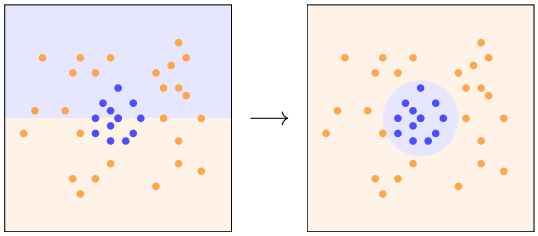

Abbildung 7: Beispiel des Nutzen einer Einbettung im Fall der linearen Klassifikation: davor (links) und danach(rechts)

Eine Abbildung von einem Merkmalsraum in einen höherdimensionalen Raum nennen wir Einbettung.

Da im Allgemeinen das Finden einer Hyperebene und die daraus entstehende Hypothese eine Vielzahl an Skalarprodukten erfordert und dies aufwändig sein kann, möchten wir nun Kernel, insbesondere symmetrisch und positiv semidefinite Kernel, einführen. Die Idee ist, dass der Kernel von zwei Datenpunkten gerade dem Skalarprodukt der zwei durch die Einbettung abgebildeten Punkte entspricht.

Definition 3.2 (Symmetrisch und positiv definiter Kernel (vgl.([MRT18]))**.**
Ein **Kernel** über X ist eine Abbildung $k : X \times X \to \mathbb{R}$ für einen beliebigen Merkmalsraum X. Wir nennen einen Kernel symmetrisch und positiv definit, wenn für alle Teilmengen $\{x_1, \ldots, x_m\} \subseteq X$ die Matrix $K_{kernel} = [k(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ symmetrisch und positiv semidefinit ist.

Zur besseren Vorstellung: Eine Matrix K_{kernel} ist genau dann positiv semidefinit, wenn $c^T K_{kernel} c = \sum_{i,j=1}^{m} c_i c_j K_{kernel}(x_i, x_j) \geq 0$ für alle $c \in \mathbb{R}^m$ gilt (vgl. [MRT18]). Wir betrachten im Folgenden nur symmetrisch und positiv definite Kernel, weswegen wir symmetrisch und positiv definit weglassen. Ein Beispiel für einen Kernel ist der Gauß-Kernel.

Beispiel 3.3 (Gauß-Kernel (vgl. [Van16]))**.** *Für eine beliebige Konstante $\sigma > 0$ ist der Gauß-Kernel definiert als $k_{rbf}(x_1, x_2) = \exp(-\frac{||x_1 - x_2||^2}{2\sigma^2})$ für alle x_1 und x_2 aus dem Merkmalsraum X.*

Der Gauß-Kernel misst hierbei die Ähnlichkeit zwischen x_1 und x_2, wobei die euklidische Distanz genutzt wird. Die Konstante σ trifft eine Aussage darüber, ab wann Distanzen zwischen den Vektoren berücksichtigt werden sollten (vgl. [Mur22]). Oft wird x_2 im Vorhinein festgelegt, um für eine Menge an Datenpunkten die Ähnlichkeit zu x_2 zu messen. Betrachtet man eine Linearkombination von Gauß-Kerneln, so kann x_2 beispielsweise jeweils als Punkt einer Untermenge an Quell- oder Zieldaten oder anderen Daten gewählt werden. Die Wahl von x_2 kann schwierig zu bestimmen sein (vgl. [Van16]). Eine Möglichkeit ist das Anwenden eines Kreuzvalidierungsverfahrens.

Definition 3.4 (F-faches Kreuzvalidierungsverfahren (vgl. [MRT18]))**.**
Bei einem **F-fachen Kreuzvalidierungsverfahren** werden die Daten zufällig in F Datenteilmengen geteilt. Für jede Datenteilmenge mit Index $i \in [F]$ wird durch einen Lernalgorithmus eine Prognose auf den verbleibenden Datenteilmengen mit Indizes $[F] \setminus i$ trainiert und auf der Datenteilmenge mit Index i getestet. Aus den entstanden Prognosen und deren Fehlern wird ein durchschnittlicher Fehler berechnet.

Jedoch muss an dieser Stelle angemerkt werden, dass durch die Vielzahl an Wahlmöglichkeiten ein Kreuzvalidierungsverfahren nicht immer ausreicht. Wir werden im

weiteren Verlauf nicht weiter darauf eingehen, eine Betrachtung dieses Problems befindet sich in [Van16].

Sei im Folgenden ein Merkmalsraum X und ein Labelraum Y gegeben. Wir schreiben x für einen beliebigen Punkt aus dem Merkmalsraum und y für ein beliebiges Label aus dem Labelraum.

3.1.1 `Kernel Mean Matching`

Die Inhalte dieses Unterkapitels basieren auf [SPH07] und [Mat+21].

Die Idee des Verfahren `Kernel Mean Matching` (`KMM`) ist es, die Quelldaten X^Q und Zieldaten X^Z in einen höherdimensionalen Raum abzubilden und dort die Distanz zwischen den abgebildeten Quell- und Zieldaten zu minimieren, sodass geeignete Gewichte für die Quelldaten gefunden und damit die Quelldaten an die Zieldaten angepasst werden.

Formal wollen wir für eine Einbettung ψ den folgenden Ausdruck

$$\left\| \frac{1}{n} \sum_{i=1}^{n} \beta_i \psi(x_i^Q) - \frac{1}{m} \sum_{i=1}^{m} \psi(x_i^Z) \right\|^2$$

nach β minimieren, wobei $\beta_i \in [0, B]$ und $|\frac{1}{n} \sum_{i=1}^{n} \beta_i - 1| \leq \varepsilon$ für festgelegte Parameter B und ε. Wir können den Zusammenhang zwischen Einbettungen und Kernel nutzen und das Minimierungsproblem zu

$$\min_{\beta} \frac{1}{2} \beta^T k \beta - \kappa^T \beta$$

umschreiben, sodass $\beta_i \in [0, B]$ und $|\sum_{i=1}^{n} \beta_i - n| \leq n\varepsilon$ gelten. Dabei ist k definiert als $k_{ij} := k(x_i^Q, x_j^Q)$, wobei k ein gewählter Kernel, beispielsweise der Gauß-Kernel, ist und κ definiert als $\kappa_i = \frac{m}{m'} \sum_{j=1}^{m'} k(x_i^Q, x_j^T)$.

Das beschriebene Minimierungsproblem ist ein quadratisches Optimierungsproblem. Lösungsverfahren lassen sich in [SPH07] nachlesen. Das resultierende β wird als Gewicht für die Quelldatenpunkte verwendet und alle Datenpunkte werden einem Lernalgorithmus L übergeben, mit dem eine Hypothese berechnet wird.

Bei den beiden folgenden Verfahren ist unser Ziel weiterhin, $w(x, y)$ (vgl. Gleichung 2.1) als Gewicht für die Quelldaten zu approximieren. Wir machen Gebrauch von der üblichen Annahme, dass sie sich als Linearkombination eines Kernels darstellen lassen: $w(x, y) = \sum_{1 \leq i \leq t} \alpha_i k((x, y), (x_i, y_i))$ für ein $t \in \mathbb{N}$ und einen gewählten Kernel k. Dies ist möglich, da sich mit Kerneln viele verschiedene Abbildungen beschreiben lassen (vgl. [Van16]). Beim ersten nun folgenden Verfahren betrachten wir für (x_i, y_i) die Zieldaten. Beim zweiten folgenden Verfahren nutzen wir die Quelldaten und nutzen als Kernel den Gauß-Kernel.

3.1.2 `Kullback-Leibler Importance Esimation Procedure`

Die Inhalte dieses Unterkapitels basieren auf [Sug+08], [MRT18], [Mur23] und [Mat+21].

Die `Kullback-Leibler Importance Esimation Procedure` (`KLIEP`) basiert auf der Idee, die Kullback-Leibler-Divergenz zu minimieren. Diese ist wie folgt definiert:

Definition 3.5 (Kullback-Leibler-Divergenz (vgl. [Mur22]))**.**
Die **Kullback-Leibler-Divergenz** ist definiert als $D_{KL}(p||q) = \int p(x) \log(\frac{p(x)}{q(x)})dx$ für zwei stetige Verteilungen $p(x), q(x)$ und ein beliebiges x.

Bemerkung 3.6. Eine Definition für stetige Verteilungen findet sich in [Mur22]. Dort findet sich zudem eine analoge Definition der Kullback-Leibler-Divergenz für diskrete Verteilungen.

Vorstellen kann man sie sich als Maß für Unterschiede zwischen zwei Verteilungen. Möchten wir dementsprechend Gewichte $w(x, y)$ finden, sodass $w(x, y)p^Q(x, y)$ möglichst die Verteilung der Trainingsdaten ist, ist dies gleichbedeutend mit dem Ziel, die Kullback-Leibler-Divergenz zu minimieren:

$$\begin{aligned} w(x,y)^* &= \arg\min_w D_{KL}(p^Z||w \cdot p^Q) \\ &= \arg\min_w \int p^Z(x,y) \log\left(\frac{p^Z(x,y)}{w(x,y)p^Q(x,y)}\right) d(x,y) \\ &= \arg\min_w \int p^Z(x,y) \log\left(\frac{p^Z(x,y)}{p^Q(x,y)}\right) d(x,y) - \int p^Z(x,y) \log(w(x,y))\, d(x,y) \end{aligned}$$

Da der erste Term nicht von den Gewichten abhängt, können wir dies wie folgt umschreiben:

$$\begin{aligned} w(x,y)^* &= \arg\max \int p^Z(x,y) \log(w(x,y))\, d(x,y) \\ &\approx \frac{1}{m} \sum_{1 \le i \le m} \log(w(x_i^Z, y_i^Z)) \\ &= \frac{1}{m} \sum_{1 \le i \le m} \log\left(\sum_{1 \le i \le m} \alpha_i k((x_j^Z, y_j^Z), (x_i^Z, y_i^Z))\right) \end{aligned}$$

Statt die gesamte Verteilung der Zieldaten zu betrachten, approximieren wir das Integral mit dem Durchschnitt des Logarithmus der Gewichtsfunktion auf den Zieldaten. Außerdem setzen wir die Darstellung der Gewichte ein. Zusammengefasst möchten wir ein α^* finden, welches

$$\alpha^* = \max_\alpha \sum_{1 \le j \le m} \log\left(\sum_{1 \le i \le m} \alpha_i k((x_j^Z, y_j^Z), (x_i^Z, y_i^Z))\right)$$

erfüllt, und können damit dann die Gewichte ausrechnen. Aus Normierungsgründen möchten wir dabei, dass

$$1 \approx \frac{1}{n} \sum_{1 \le i \le n} w(x_i^Q, y_i^Q) \Leftrightarrow n \approx \sum_{1 \le j \le n} \sum_{1 \le i \le m} \alpha_i k((x_j^Q, y_j^Q), (x_i^Z, y_i^Z))$$

Wir werden zum Finden von α^* annehmen, dass bei der zweiten Approximation Gleichheit gilt. Das Finden einer Lösung mit der genannten Nebenbedingung ist ein konvexes Optimierungsproplem, weswegen wir auch hier eine eindeutige Lösung finden.

3.1.3 `Instance weighting kernel ridge regression`

Die Inhalte dieses Unterkapitels basieren auf [GV14], [MRT18] und [Van16].

Betrachten wir zunächst Ridge regression. Dabei möchten wir die Funktion

$$\sum_{i=1}^{n}(\theta^t x_i - y_i)^2 + \lambda||\theta||^2 \tag{3.1}$$

nach θ minimieren mit y_i als die tatsächlichen Labels, x_i als die mitgegeben Trainingsdatenpunkte und λ als Regularisierungsparameter. Der erste Term ist gerade eine lineare Regression (siehe als Beispiel Abbildung 3) mit dem Ziel, eine lineare Abbildung zu finden. Der zweite Term ist eine Regularisierung, um Overfitting zu vermeiden und damit allgemein gute Hypothesen zu erhalten.

Damit auch komplexere Hypothesen erstellt werden können, möchten wir hier Kernel nutzen. Wir wollen nun dementsprechend

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{n}\theta_j k(x_i, x_j))^2 + \lambda||\theta||^2$$

nach θ minimieren für einen ausgewählten Kernel k. Das Minimieren von diesem Term nennt sich Kernel ridge regression.

Das Verfahren `Instance weighting kernel ridge regression` (`IWKRR`) basiert auf dem Ziel, eine umgeformte Kernel ridge regression

$$\frac{1}{2}\left(\sum_{i=1}^{m}\left(\theta^t\phi(x_i^T) - y_i^T\right)^2 + \sum_{j=1}^{n} w_j\left(\theta^t\phi(x_j)^S - y_j^S\right)^2\right) + \frac{\lambda}{2}\sum_{d=1}^{D}\theta_d^2 \tag{3.2}$$

nach θ zu minimieren. Durch die Gewichte w_i sollen die Quelldaten an die Zieldaten angepasst werden. Betrachten wir Gleichung 3.2 genauer, entspricht dies der Formel für Kernel ridge regression. Die Frage ist nun, wie w_i gewählt werden soll.

`IWKRR` möchte $||y-y'||^2$ für alle tatsächlichen Labels y nach y', wobei dies dem prognostizierten Label ist, minimieren. Das können wir zu folgendem Minimierungsproblem mit Gleichung 2.3 umschreiben:

$$\min_w \sum_{i=1}^{m}(y_i^Z - \arg\max_y(w(x_i^Z, y)p^Q(y|x_i^Z)))^2.$$

Kennen wir die Verteilung der Quelldaten, können wir leicht die optimalen Gewichte ausrechen. Um dementsprechend $p^Q(y|x)$ aus dem Problem zu approximieren, nutzen wir eine Kernel ridge regression, denn $\arg\max_y p^Q(y|x)$ entspricht gerade dem durch eine perfekte Hypothese prognostizierten Label, erstellt auf den Quelldaten. Wir approximieren dieses Label mit einer Kernel ridge regression

$$\arg\max_y p^Q(y|x) \approx a^t k(x)$$

für einen Kernel $k(x) = (k(x_1^Q, x), \cdots, k(x_n^Q, x))$. Wir nehmen an, dass wir a durch eine Kernel ridge regression gefunden haben. Wir müssen noch die Gewichte einfügen. Durch ein ähnliches Vorgehen erhalten wir (genauer in [Van16] dargestellt) mit $W(x, y) = \text{diag}((x_i, y_i), \ldots, (x_i, y_i))$

$$\arg\max_y w(x, y)p^Q(y|x) \approx a^t W(x, y)k(x)$$

und damit

$$\min_{w} \sum_{i=1}^{m} (y_i^Z - a^t W(x_i^Z, y_i^Z) k(x_i^Z))^2.$$

Da die Gewichte nur von α abhängen und wir wie bei `KMM` zusätzlich noch einen Regularisierungsterm hinzufügen möchten, ergibt sich:

$$\min_{\alpha} \sum_{i=1}^{m} (y_i^Z - a^t W(x_i^Z, y_i^Z) k(x_i^Z))^2 + \gamma ||\alpha||^2. \tag{3.3}$$

Damit erhalten wir die folgenden Schritte für `IWKRR`:

1. Zunächst berechnen wir auf den Quelldaten eine Kernel ridge regression, um ein a zu erhalten und die Verteilung der Quelldaten zu approximieren.
2. Daraufhin können wir mithilfe von a ein α finden, welches Gleichung 3.3 minimiert.
3. Das α setzen wir in die Gewichte ein und können damit Gleichung 3.2 nach θ minimieren und erhalten damit unsere Endhypothese.

3.2 Neuronale Netzwerke

Zudem besteht die Möglichkeit des Trainierens eines neuronales Netzwerks. Obgleich diese oft sehr gute Ergebnisse liefern, werden wir sie im weiteren Verlauf der Bachelorarbeit nicht weiter betrachten, da viele grundlegende Mechanismen von neuronalen Netzwerken noch nicht gut verstanden sind (vgl. [Ber+23]). Trotzdem gibt es auch zu neuronalen Netzwerken und Transferlernen aktuelle Forschung, beispielsweise [Asi+23].

3.3 Ansatz mit Boosting

Ein weiterer Ansatz zum Finden der Gewichte basiert auf dem Prinzip Boosting. Boosting ist eine etablierte Methode zur Verbesserung der Vorhersagegenauigkeit von Klassifikationsalgorithmen. Boosting basiert auf der Idee, schrittweise einen schwachen Lernalgorithmus zu kombinieren, um einen stärkeren, leistungsfähigeren Lernalgorithmus zu generieren.

Definition 3.7 (Boosting (vgl. [MRT18]))**.**
Boosting ist eine Methode, bei der ein schwacher Lernalgorithmen mehrfach so kombiniert werden, dass die resultierende Prognose die Kostenfunktion stärker minimiert als der schwache Lernalgorithmus alleine.

Um das Prinzip Boosting besser nachvollziehen zu können, schauen wir uns im Folgenden einen der bekanntesten Boostingalgorithmen an: `AdaBoost` (vgl. [Gup+22]).

3.3.1 `AdaBoost`

Die Grundlage, auf der die meisten Boostingalgorithmen aufbauen, ist `AdaBoost`, ein Klassifikationsalgorithmus (vgl. [MRT18]). In `AdaBoost` aus [FS97] wird ein Trainingsdatensatz mit Labels aus $Y = \{0, 1\}$ und ein schwacher Lernalgorithmus verwendet. Der Algorithmus basiert auf der schrittweisen Anpassung der Gewichtung der Datenpunkte.

Beispiel 3.8. *Wir betrachten Abbildung 8. Wir möchten hierbei die blauen von den orangen Punkten trennen und haben dafür jedoch nur eine Gerade zu Verfügung. Dieses Ziel ist bei diesem Beispiel nicht erreichbar. Deswegen versuchen wir zunächst eine Gerade zu finden, die möglichst viele Punkte nach Farbe trennt. Wir stellen fest, dass ein blauer Punkt (umkreist) jedoch auf der orangen Seite liegt. Durch Fokussierung auf diesen Punkt, können wir eine Gerade finden, die diesen Punkt und möglichst viele weitere Punkte richtig klassifiziert. Die Kombination aus den beiden Geraden liefert uns eine Trennlinie, die eine Separierung in einen orangenen und blauen Bereich ermöglicht.*

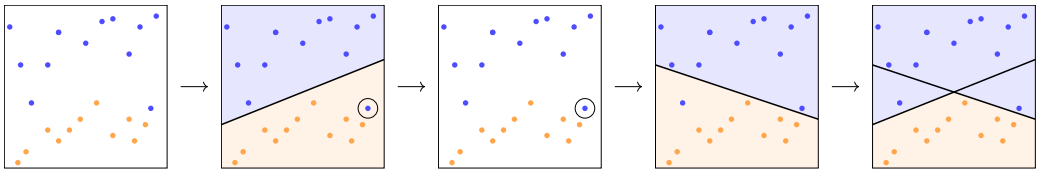

Abbildung 8: Beispiel von `AdaBoost`

Formal werden zunächst die Gewichte initialisiert und normiert (Schritt 0 und 1). Nachdem eine vorläufige Prognose durch den schwachen Lernalgorithmus erstellt wurde (Schritt 2), wird der Fehler, also die Kostenfunktion, auf den Trainingsdaten berechnet (Schritt 3). Bei `AdaBoost` werden die Gewichte der falsch klassifizierten Punkte aufaddiert. Daraus wird ein Gewichtungsfaktor errechnet, der vom Fehler der Prognose abhängt (Schritt 4). Die Gewichtung der Eingabewerte wird entsprechend der Korrektheit der Vorhersagen angepasst (Schritt 5). Wenn die Vorhersagen korrekt sind, erhalten diese Datenpunkte weniger Gewicht. Im Gegensatz dazu werden Datenpunkte, bei denen die Vorhersagen von den eigentlichen Labels abweichen, stärker gewichtet. Diese Schritte werden mehrfach wiederholt. Ausgegeben wird eine Hypothese, die je nach Größe des Fehlers der einzelnen Prognosen aus den einzelnen gewichteten Prognosen von jedem Schritt bestehen. Die Herleitung von `AdaBoost` lässt sich in [FS97] nachlesen.

Algorithm 1: AdaBoost

Eingabe: Trainingsdaten X^Z und Y^Z der Größe m, Anzahl an Iterationen N, schwacher Lernalgorihtmus L

Initialisiere $w_i^{(1)} = 1$ für alle $1 \leq i \leq m$ `// Schritt 0`

Ausgabe: Hypothese h_{t^*}

for $t = 1, ..., N$ **do**

- Initialisierung $Z^{(t)} = \sum_{i=1}^{m} w_i^{(t)}$, $p_i^{(t)} = \frac{w_i^{(t)}}{Z^{(t)}}$ `// Schritt 1`
- Erhalte Hypothese h_t durch L mit $p_i^{(t)}$ `// Schritt 2`
- Berechne den Fehler $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} p_i^{(t)}$ `// Schritt 3`
- Berechne den Gewichtungsfaktor $\eta_t = \frac{1}{2}\ln(\frac{1}{\varepsilon_t} - 1)$ `// Schritt 4`
- Aktualisiere die Gewichte $w_i^{(t+1)} = w_i^{(t)} \mathrm{e}^{-\eta_t y_i h_t(x_i)}$ `// Schritt 5`

return $h_{t^*}(x) = sgn(\sum_{t=1}^{N} \eta_t h_t(x))$

Wir betrachten das Anpassen der Gewichte in jedem Schritt in Algorithmus 1 genauer. Wir stellen zunächst fest, dass sich der Gewichtungsfaktor antiproportional verhält zum Fehler. Zudem lassen sich die neuen Gewichte durch Einsetzen der De-

finition von η_t umschreiben, $w_i^{(t+1)} = w_i^{(t)} \mathrm{e}^{-\eta_t y_i h_t(x_i)}$ wird zu $w_i^{(t+1)} = w_i^{(t)} (\frac{1}{\varepsilon_t} - 1)^{-\frac{1}{2} y_i h_t(x_i)}$. Es lässt sich außerdem erkennen, da $y_i h_t(x_i)$ bei falscher Zuordnung negativ ist, dass die Gewichte bei falscher Zuordnung größer werden. Anders herum, werden sie bei richtiger Zuordnung kleiner. Außerdem misst η_t die Wichtigkeit der einzelnen Prognosen für die Gesamtprognose. Je kleiner der Fehler, also je größer η_t, umso mehr Einfluss wird der Prognose aus der Iteration t auf die Gesamtprognose gegeben. Je höher die Anzahl an Iterationen, desto kleiner wird der Fehler der Hypothese sein. Jedoch besteht das Risiko von Overfitting.

Es lässt sich mathematisch beweisen, dass sich durch die Verwendung von `AdaBoost` (Algorithmus 1) ein starker Lernalgorithmus erzeugen lässt (vgl. [FS97]). Obwohl wir einen starken Lernalgorithmus erzeugen, der komplexe Situationen meistern kann, ist die Gefahr für Overfitting niedriger als bei vielen anderen starken Lernalgorithmen (vgl. [Gup+22]). Dies liegt daran, dass Ausreißer, im Vergleich zu den restlichen Daten, zwar betrachtet werden, aber dies in der Endhypothese durch den Gewichtungsfaktor miteinbezogen wird. Dadurch werden die Vorhersagen weniger anfällig für Ausreißer.

3.3.2 `AdaBoost.R2`

Bevor wir erläutern, inwieweit Boosting für das Finden der Gewichte der Quelldaten beim Transferlernen hilfreich sein kann, erweitern wir `AdaBoost` zu einem Regressionsalgorithmus. Dafür betrachten wir insbesondere die Berechnung des Fehlers und das Anpassen der Gewichte. Der Fehler kann nun beliebig groß werden, da die Labels beliebige reelle Werte annehmen können. Um jedoch weiterhin die Fehler der Prognosen vergleichen zu können, betrachten wir den angepassten Fehler:

Definition 3.9 (Angepasster Fehler (vgl. [Dru97]))**.**
Gegeben sei ein Datenset X der Größe m mit Labels Y und prognostizierten Labels Y'. Der Fehler eines Datenpunkts ist definiert als $err_i := |y_i - y_i'|$. Wir definieren den **angepassten Fehler** eines Datenpunkts als $e_i := \frac{err_i}{\max_{j \in \{1,\cdots,m\}} err_j}$ für alle $i \in \{1, \cdots, m\}$.

Insbesondere stellen wir fest, dass der angepasste Fehler zwischen 0 und 1 liegt. Statt nun wie in `AdaBoost` nur die aufaddierten Gewichte der Punkte mit falsch prognostizierten Labels als Fehler zu definieren, skalieren wir diese mit dem angepassten Fehler des jeweiligen Punkts. Dadurch haben Punkte mit vergleichsweise großem Fehler in der Prognose des Labels größeren Einfluss als Punkte mit vergleichsweise kleinem Fehler, weiterhin jedoch abhängig von den Gewichten. Wir betrachten nun den Algorithmus `AdaBoost.R2` (vgl. [PS10]).

Da in Algorithmus 2 $\varepsilon_t < \frac{1}{2}$, weil sonst abgebrochen wird, und somit $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} = (\frac{1}{\varepsilon_t} - 1)^{-1} < 1$ ist, aktualisieren wir die Gewichte exakt wie in `AdaBoost`, jedoch ist der Exponent von dem angepassten Fehler des Datenpunkts abhängig. Der Exponent ist also nicht mehr $-\frac{1}{2}$ oder 0, sondern $-(1 - e_i^{(t)}) \in [0, 1]$.

Die Ausgabe ist der gewichteter Median der Hypothesen mit dem Gewichtungsfaktor $\ln(\frac{1}{\beta_t})$. Er gibt dabei das kleinste Label aus, sodass die Summe der Gewichte der Hypothesen, bei denen das prognostizierte Label kleiner gleich ist als das Label, größer ist als die Hälfte der Summe aller Gewichte von allen Hypothesen. Das bedeutet, dass hierbei das prognostizierte Label am nächsten am tatsächlichen Label liegt, bei

Algorithm 2: AdaBoost.R2

Eingabe: Trainingsdaten X^Z und Y^Z der Größe m, Anzahl an Iterationen N, schwacher Lernalgorithmus L

Initialisiere $w_i^{(1)} = \frac{1}{m}$ für alle $1 \leq i \leq m$

Ausgabe: Hypothese h_{t^*}

for $t = 1$ *to* N **do**

- Erhalte Hypothese h_t durch L mit $w_i^{(t)}$
- Berechne den angepassten Fehler $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{J}$ mit $J = \max_{i=1}^{n} |y_i - h_t(x_i)|$
- Berechne den Gesamtfehler $\varepsilon_t = \sum_{i=1}^{m} e_i^{(t)} w_i^{(t)}$
- **if** $\varepsilon_t \geq \frac{1}{2}$ **then**
  - **return** $h_f(x) = \inf\{y \in Y : \sum_{t:h_t(x) \leq y} \ln\left(\frac{1}{\beta_t}\right) \geq \frac{1}{2} \sum_t \ln\left(\frac{1}{\beta_t}\right)\}$
- Berechne den Gewichtungsfaktor $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
- Aktualisiere die Gewichte $w_i^{(t+1)} = \frac{w_i^{(t)} \beta_t^{1-e_i^{(t)}}}{Z_t}$ mit Z_t als Normalisierungskonstante

return $h_{t^*}(x) = \inf\{y \in Y : \sum_{t:h_t(x) \leq y} \ln\left(\frac{1}{\beta_t}\right) \geq \frac{1}{2} \sum_t \ln\left(\frac{1}{\beta_t}\right)\}$

über der Hälfte der gewichteten Hypothesen.

Diesen Algorithmus können wir nun verwenden, um geeignete Gewichte für die Quelldaten beim Transferlernen zu finden.

3.3.3 `Two-stage TrAdaBoost.R2`

Die Zieldaten entsprechen gerade den Trainingsdaten, dementsprechend werden wir die Vorgehensweise auf diesen zunächst beibehalten. Die Idee ist, dass wir über die Iterationen hinweg geeignete Gewichte für die Quelldaten finden und gleichzeitig Boosting nutzen, um eine geeignete Hypothese auf den Zieldaten zu finden. Betrachten wir `AdaBoost.R2` (Algorithmus 2) im Hinblick auf die Quelldaten genauer. Wurde einem Quelldatenpunkt ein falsches Label zugeordnet, liefert dieser vermutlich keine relevanten Informationen für das Zieldatenset. Wir möchten also, dass dessen Gewichte reduziert werden, um den Einfluss des Quelldatenpunkts zu reduzieren.

Wir möchten weiterhin die Gewichte der Zieldaten anpassen, denn auch bei den Zieldaten können Ausreißer für schlechtere Hypothesen sorgen. Zusammenfassend bedeutet dies, dass bei Richtigkeit ihres Labels die Gewichte der Zieldaten sich gegensätzlich zu den Gewichten der Quelldaten verhalten. Eine mögliche formale Aktualisierung der Gewichte für einen Gewichtungsfaktor β_t basierend auf den angepassten Fehlern e_i^t in der Iteration t (analog zu [Dai+07]) wäre:

$$w_i^{t+1} = w_i^t \beta_t^{1-e_i^t} \qquad \text{für die Zieldaten,}$$

$$w_i^{t+1} = w_i^t \beta_t^{e_i^t} \qquad \text{für die Quelldaten.}$$

Jedoch haben sich nach [PS10] bei einer solchen Aktualisierung wie dieser hier zwei Probleme ergeben: Zunächst gehen Gewichte von Quelldaten, die ähnlich zu Zieldaten

sind, auch gegen 0, sodass nur wenig Informationen aus den Quelldaten gezogen werden konnten. Das liegt an der Nutzung des angepassten Fehlers, da dadurch selbst kleine Fehler zu einer Reduzierung der Gewichte führen. Das zweite Problem besteht darin, dass die Summe der Gewichte der Quelldaten vergleichsweise hoch zu der Summe der Gewichte der Zieldaten bei einer höheren Anzahl an Quelldaten ist. Das führt dazu, dass von den Quelldaten weiter entfernte Zieldaten höhere Gewichte erhalten als naheliegende Zieldaten, da der Fokus zu stark auf den Quelldaten liegt und diese den Fehler dominieren.

Um die beiden Probleme anzugehen, wird in [PS10] der Algorithmus `Two-Stage TrAdaBoost.R2` (`TTR2`) vorgestellt. Dieser ist aufgeteilt in zwei Phasen. Nach der Initialisierung und der Normierung der Gewichte wird in der ersten Phase `AdaBoost.R2'` aufgerufen mit den Gewichten, einer Anzahl an Iterationen, dem schwachen Lernalgorithmus und den Ziel- und Quelldaten. In `AdaBoost.R2'` wird eine Hypothese wie in `AdaBoost.R2` berechnet, wobei die Gewichte der Quelldaten in diesem Schritt nicht verändert werden. Das bedeutet, dass `AdaBoost.R2'` in einer gewissen Anzahl an Iterationen verschiedene Gewichtungen der Zieldaten mit den festgesetzten Gewichten der Quelldaten ausprobiert. Dies ermöglicht einen stärkeren Fokus auf die Zieldaten. Mit einem Kreuzvalidierungsverfahren wird für die Endhypothese der Iteration ein Fehler berechnet. Der kleinste Fehler von allen Iterationen in `TTR2` bestimmt, welche Hypothese am Ende des Algorithmus ausgeben wird. Dadurch wählen wir die Hypothese, die auf den Zieldaten am besten funktioniert hat.

In der zweiten Phase von `TTR2` werden nun auch die Quelldaten verändert. Es wird zunächst auf den Ziel- und Quelldaten mit dem schwachen Lernalgorithmus eine Hypothese und der angepasste Fehler auf den Daten berechnet (analog zu `AdaBoost.R2`). Dabei werden die veränderten Gewichte der Trainingsdaten aus der vorherigen Phase übernommen. Mithilfe des angepassten Fehlers werden die Gewichte der Quelldaten aktualisiert. Dabei bleibt die Idee bestehen, dass die Gewichte von falsch zugeordneten Labels von Quelldaten reduziert werden sollten. Der Gewichtungsfaktor hängt hierbei von der Iteration ab, in der wir uns befinden. Mit jeder Iteration wird der Gesamteinfluss der Quelldaten reduziert. Der Gewichtungsfaktor wird dabei so gewählt, dass das Gesamtgewicht der Zieldaten in der Anzahl an Iterationen von ihrer Gesamtgröße im Vergleich zu der Gesamtzahl an Daten zu 1 steigt. Dadurch wird sichergestellt, dass `AdaBoost.R2'` auf verschiedene Verhältnisse der Gewichte der Quell- und Zieldaten Hypothesen berechnet und damit das beste Verhältnis herausfindet. Jedoch wird weiterhin die Größe eines Fehlers eines Punkts entscheidend sein für die Aktualisierung der Gewichte.

Durch die beiden Phasen wird sichergestellt, dass trotz einer deutlich größeren Menge an Quelldaten im Vergleich zu den Zieldaten der Fokus auf den Zieldaten liegt und relevantere Quelldaten für die Zieldaten höheres Gewicht haben und nicht gegen 0 gehen (vgl. [PS10]). Außerdem reduziert das 2-Phasen-Modell die durch die Quelldaten verursachte Verzerrung, falls es im Vergleich zu den Zieldaten deutlich mehr Quelldaten gibt (vgl. [Gup+22]).

Bevor wir den Algorithmen betrachten, führen wir Notation ein:

Notation 3.10 (Quell-und Zieldaten).
Wir notieren die Quelldaten wie folgt:

- $X^Q = (x_1, \ldots, x_n)$

- $Y^Q = (y_1, \ldots, y_n)$

Wir notieren die Zieldaten wie folgt:

- $X^Z = (x_{n+1}, \ldots, x_{m+n})$
- $Y^Z = (y_{n+1}, \ldots, y_{m+n})$

Algorithm 3: Two-stage TrAdaBoost.R2

Eingabe: Zieldaten X^Z mit Y^Z der Größe m, Quelldaten X^Q mit Y^Q der Größe n, Anzahl an Iterationen N, Anzahl an Boostingiterationen N_{Boost}, Anzahl an Kreuzvalidierungen F, Lernalgorithmus L

Ausgabe: Hypothese h_{t^*}

Initialisiere $w_i^{(1)} = \frac{1}{n+m}$ für alle $1 \leq i \leq n+m$

for $t = 1$ *to* N **do**

  Rufe AdaBoost.R2' auf und erhalte Fehler $error_t$ **von** $model_t$**:**

    for $s = 1$ *to* N_{Boost} **do**

      Erhalte Hypothese h_s durch L mit $w_i^{(s)}$

      Berechne den angepassten Fehler $e_i'^{(s)} = \frac{|y_i - h_s(x_i)|}{Y}$ mit $Y = \max_{i=1}^n |y_i - h_s(x_i)|$

      Berechne den Gesamtfehler $\varepsilon_s = \sum_{i=1}^m e_i'^{(s)} w_i^{(s)}$

      if $\varepsilon_s \geq \frac{1}{2}$ **then**

        springe zu dem Berechnen von $model_t$

      Berechne den Gewichtungsfaktor $\beta_t = \frac{\varepsilon_s}{1-\varepsilon_s}$

      Aktualisiere die Gewichte der Zieldaten $w_i^{(s+1)} = \frac{w_i^{(s)} \beta_s^{1-e_i^{(s)}}}{Z_s}$ für $n+1 \leq i \leq n+m$ mit Z_t als Normalisierungskonstante

    Berechne $model_t = \inf\{y \in Y : \sum_{s:h_s \leq y} \ln\left(\frac{1}{\beta_s}\right) \geq \frac{1}{2} \sum_s \ln\left(\frac{1}{\beta_s}\right)\}$ und Fehler $error_t$ von $model_t$ anhand eines F-fachen Kreuzvalidierungsverfahren auf den Zieldaten

  Erhalte Hypothese h_t durch L

  Berechne den angepassten Fehler $e_i^{(t)} = \frac{|y_i - h(x_i)|}{J}$ mit $J = \max_{i=1}^n |y_i - h_t(x_i)|$

  Aktualisiere die Gewichte

$$w_i^{(t+1)} = \begin{cases} \frac{w_i^{(t)} \beta_t^{e_i^{(t)}}}{Z_t}, & 1 \leq i \leq n \\ \frac{w_i^{(t)}}{Z_t}, & n+1 \leq i \leq n+m \end{cases}$$

  mit β_t so, dass $\sum_{i=n+1}^{n+m} w_i^{(t)} = \frac{m}{(n+m)} + \frac{t}{(N-1)}(1 - \frac{m}{(n+m)})$

return h_{t^*} mit $t^* = \text{argmin}_t \text{error}_t$

Wir stellen fest, dass trotz der Lösung der vorher vorgestellten Probleme der Algorithmus 3 noch Optimierungsmöglichkeiten hat:

Zunächst muss angemerkt werden, dass der Algorithmus durch die zwei Phasen und die unveränderten Quelldaten in der ersten Phase komplex aufgebaut ist. Obwohl die Aufteilung den Fokus auf die Zieldaten legt, birgt es die Gefahr in der ersten

Phase die Generalisierbarkeit des Algorithmus zu reduzieren. Das liegt daran, dass die Quelldaten nicht aktualisiert werden. Dies hat die Auswirkung, dass keine weiteren Informationen aus dem Quelldatensatz übertragen werden, keine Anpassung an die neuen Gewichte der Zieldaten stattfindet und zu einer Überanpassung an die Quelldaten führt. Daraus kann, wie oben erwähnt, eine fehlende Generalisierbarkeit folgen.

Darüber hinaus ist `TTR2` (Algorithmus 3) anfällig für negativen Transfer (vgl. [Men22] und [Gup+22]). Wir werden negativen Transfer über eine Beschreibung definieren, eine mathematische Definition findet sich in [Wan+18].

Definition 3.11 (Negativer Transfer (vgl. [Wan+18]))**.**
Negativer Transfer bedeutet, dass die Übertragung von Information von Quelldaten negative Auswirkungen auf das Lernen auf den Zieldaten hat.

Wir möchten ein Beispiel zu negativem Transfer geben.

Beispiel 3.12. *Führen wir Beispiel 2.19 weiter. Unsere Quelldaten bestehen aus an Eierstockkrebs erkrankten älteren Frauen und unsere Zieldaten, wobei wir deutlich weniger Zieldaten als Quelldaten gegeben haben, sind Daten von jüngeren erkrankten Frauen. Wir wollen eine Prognose zur Überlebenschance von jungen erkrankten Frauen vorhersagen. Obwohl die Überlebenschancen bei* 20% *(vgl. [Höw23]) liegen, haben wir jedoch nur Daten von erkrankten älteren Frauen gegeben, die an Eierstockkrebs gestorben sind. Übertragen wir nun diese Information auf unsere Zieldaten, so werden wir die Überlebenschancen stark reduzieren, insbesondere da wir mehr Quelldaten als Zieldaten gegeben haben. Diese Verzerrung der Überlebenschancen ist ein Beispiel für negativen Transfer.*

Wie in Beispiel 3.12 gesehen, hängt Transferlernen von der Größe des Zieldatensatzes und dem Unterschied zwischen den Verteilungen der Quell- und Zieldaten ab. Darüber hinaus sollte zum Vergleichen negativer Transfer wie folgt verstanden werden: Durch das Hinzufügen von Quelldaten ist die Kostenfunktion auf den Testdaten schlechter als ohne das Hinzufügen der Quelldaten und nur durch das Trainieren auf den Trainingsdaten mit dem gleichen Lernalgorithmus. ([Wan+18]). Es wäre beispielsweise kontraproduktiv, einen schwachen Lernalgorithmus für Transferlernen mit einem deutlich stärkeren Lernalgorithmus, nur auf den Zieldaten angewendeten, zu vergleichen. Da dementsprechend die Wahl des Lernalgorithmus Einfluss auf negativen Transfer hat, versuchen Gupta et al. `TTR2` in Hinblick auf reduzierten negativen Transfer weiterzuentwickeln. Wir werden im nächsten Kapitel ein Verfahren präsentieren, das probiert, `TTR2` zu verbessern.

4 Algorithmus `S-TrAdaBoost.R2`

Im Kapitel 4 betrachten wir nun den Algorithmus `S-TrAdaBoost.R2` aus [Gup+22], eine Weiterentwicklung von `TTR2`. Wir werden zunächst Lösungsideen präsentieren zu den Optimierungsmöglichkeiten in Kapitel 3.3 und diese zu einem Algorithmus zusammenfügen. Wir haben zwar Unstimmigkeiten zwischen dem bereitgestellten Code von [Gup+22] und dem im Artikel vorgestellten Algorithmus gefunden, jedoch werden wir uns in diesem Kapitel zunächst nur auf den im Artikel vorgestellten Algorithmus `S-TrAdaBoost.R2` konzentrieren. Wir werden in Kapitel 5 die Unstimmigkeiten erläutern und die verschiedenen Versionen in der Praxis vergleichen.

Sofern nicht anders angegeben, basieren alle Inhalte des Kapitels auf [Gup+22].

4.1 Sampling

Um den Aufwand unseres Algorithmus zu reduzieren und dabei trotzdem Generalisierbarkeit zu garantieren, möchten wir zunächst unsere gegebenen Quell- und Zieldaten bearbeiten. Außerdem möchten wir die in Kapitel 3.3 beschriebenen Schwächen von `TTR2` verbessern. Als erstes konzentrieren wir uns auf die Reduktion des negativen Transfers. Experimente zeigen, dass Sampling eine Lösung sein könnte ([Men22]), da das Verwenden von den gesamten Quelldaten zum negativen Transfer beitragen kann und Sampling in Boosting gleiche oder teilweise bessere Ergebnisse liefert als reines Umgewichten in Boosting (vgl. [Sei+08]). Auch mit anderen Algorithmen erzeugt Sampling eine Verbesserung in der Minimierung der Kostenfunktion (vgl. [KF19] und [ZZ15]).

4.1.1 Importance Sampling

Die Idee hinter Importance Sampling ist, dass einige Quelldaten den Zieldaten mehr ähneln als andere. Daher wollen wir im weiteren Verlauf die „schlechteren“ Daten herausfinden und diese im Vorfeld entfernen, um ein größeres Augenmerk auf die „besseren' Daten zu legen. „Schlechtere“ Daten bedeutet, dass bei denen nach Gleichung 2.3 die Gewichte $w(x, y)$ im perfekten Fall nahe an 0 gewesen wären.

Beispiel 4.1. *Betrachten wir weiterhin Beispiel 2.19. Wir haben nach wie vor die Daten der älteren erkrankten Frauen als Quelldaten und weniger Daten von jüngeren erkrankten Frauen als Zieldaten gegeben. Im Allgemeinen gibt es drei verschiedene evidenzbasierte Therapieformen für Eierstockkrebs: eine Operation, eine Chemotherapie und eine Antikörpertherapie (vgl. [Höw23]). Wir nehmen an, dass alle jüngeren erkrankten Frauen eine Operation gewählt haben, um den Eierstockkrebs zu entfernen und möchten die Chancen auf Heilung bei einer Operation prognostizieren. Die Daten der älteren erkrankten Frauen, die keine Operation gewählt haben, werden uns vermutlich keine Informationen zur Vorhersage der Chancen geben können.*

Die Idee ist nun, dass wir als Vorbereitung die Daten der älteren erkrankten Frauen, die keine Operaton als Therapieform gemacht haben, aus unserem Quelldatensatz entfernen, damit sich der Algorithmus von Anfang an auf die informativeren Daten konzentriert.

Wir möchten dementsprechend herausfinden, welche Quelldaten repräsentativer für unsere Zieldaten sind als andere und die beim Boosting ohnehin geringe Gewichte

erhalten würden. Diese Vorgehensweise ermöglicht eine Effizienzsteigerung, da die Bestimmung der Gewichte an dieser Stelle wegfällt.

Beispiel 4.2. *Wir betrachten Abbildung 9 als Beispiel. Wir berechnen zunächst die Entfernung von allen Quelldatenpunkten zum Durchschnitt der Zieldatenpunkte. Dann wählen wir die Quelldaten aus, die am weitesten entfernt sind. Diese löschen wir aus unserem Quelldatensatz.*

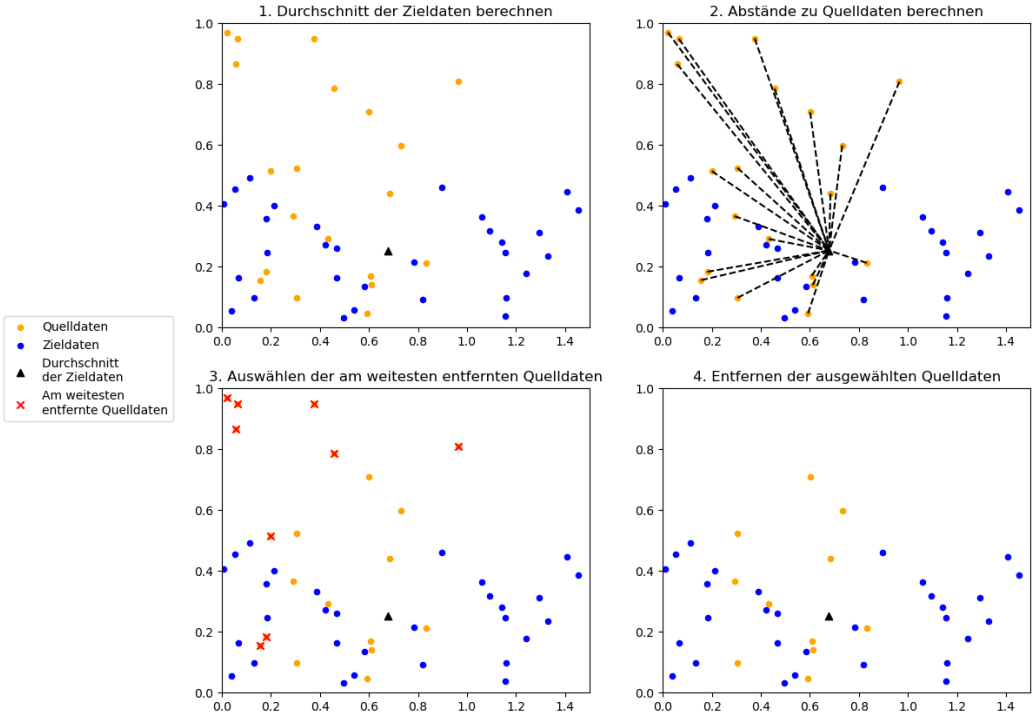

Abbildung 9: Idee von Importance Sampling

Um die weniger relevanten Daten zu bestimmen, benutzen wir wie in Beispiel 4.2 die Distanz zwischen Quell- und Zieldaten. Wir werden hier die Manhattandistanz nutzen, da [Gup+22] diese in ihrem bereitgestellten Code benutzen. An dieser Stelle besteht die Möglichkeit, verschiedene Distanzen, sowie weitere Methoden wie z.B. gradienbasierte (vgl. [Elv+23]), auszuprobieren. Formal funktioniert Importance Sampling wie folgt:

Algorithm 4: Importance Sampling

Eingabe: Zieldaten X^Z mit Y^Z der Größe m, Quelldaten X^Q mit Y^Q, Anzahl q
Ausgabe: Neuer Quelldatensatz X^{NQ} bestehend aus den näheren Quelldaten
Berechne $\overline{x}^Z = \frac{\sum_{i=1}^{m} x_i^T}{m}$
Berechne den Abstand $x_i^K = \left| |x_i^Q - \overline{x}^Z| \right|_1$ für alle $x_i \in X^Q$
$X^{NQ} = \emptyset$
for q kleinste x_i^K **do**
  $X^{NQ} \leftarrow X^{NQ} \cup \{x_i^Q\}$
return X^{NQ}

In Algorithmus 4 werden die Quelldatenpunkte mit höherer Distanz zu dem Durch-

schnitt der Zieldatenpunkte hierbei aus dem Quelldatensatz entfernt. Damit erhöhen wir die Vorhersagegenauigkeit in den Fällen, in denen Quell- und Zieldaten sich stark voneinander unterscheiden, da die Datenpunkte mit den größten Unterschieden entfernt wurden. Die Anzahl an Quelldatenpunkten, die wir entfernen, übernehmen wir aus [Gup+22], um Vergleichbarkeit zu schaffen. Jedoch wäre es hier interessant, noch weitere Experimente zu der Anzahl durchzuführen, beispielsweise mit einem Kreuzvalidierungsverfahren für jeden Datensatz. Zusammengefasst kann sich der Algorithmus 4 durch die vorzeitige Entfernung einiger für die Zieldaten weniger repräsentativen Daten auf die informativeren Quelldaten konzentrieren.

4.1.2 Variance Sampling

Wir möchten zudem, dass unser Algorithmus auch in der Allgemeinheit und nicht nur auf den Trainingsdaten gute Ergebnisse liefert. Das vorherige Verfahren `TTR2` versucht dies mit dem 2-Phasen-Modell, wie in Unterkapitel 3.3 beschrieben. Jedoch kann dies zu einer Verzerrung durch die Quelldaten führen. Deswegen erzeugen wir stattdessen ein wenig Streuung in den Trainingsdaten durch das Hinzufügen von ähnlichen Quelldaten, um allgemein bessere Vorhersagen auf den Testdaten machen zu können. Wir nutzen hierbei eine Idee aus [Sal+19]. Betrachten wir das folgende Beispiel:

Beispiel 4.3. *Wir betrachten Abbildung 10. Wir finden zunächst* 2 *Zentren der Quelldaten. Zentren sind dabei die beiden Punkte, zu denen jeweils eine von zwei Datenteilmengen der Quelldaten den kleinsten Abstand hat. Zu den Zentren suchen wir jeweils den Zieldatenpunkt, der am nächsten an dem Quelldatenzentrum liegt. Von den ausgewählten Zieldatenpunkten suchen wir nun jeweils einen Quelldatenpunkt, der zu dem Zieldatenpunkt den kleinsten Abstand hat. Diese fügen wir im weiteren Schritt der Zieldatenmenge hinzu.*

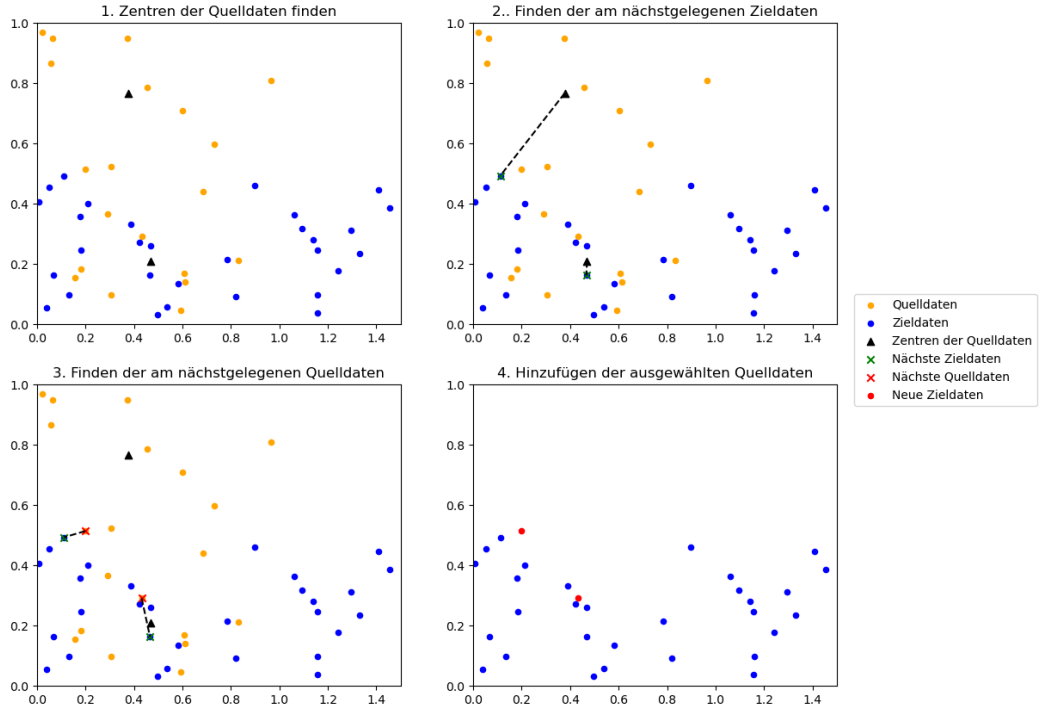

Abbildung 10: Beispiel von Variance Sampling

Insbesondere erzeugen wir damit eine höhere Anzahl an Trainingsdaten. Diese Vorgehensweise verallgemeinern wir im Folgenden:

Algorithm 5: Variance Sampling

Eingabe: Zieldaten X^Z mit Y^Z, Quelldaten X^Q mit Y^Q, Anzahl k
Ausgabe: Neuer Zieldatensatz X^{NZ} mit hinzugefügter Streuung
Finde k Zentren $X^C \subset X^Q$ mit `k-means-Clustering`, sodass
$X^C = \{x_1^C, x_2^C, \ldots, x_k^C\}$
$X^E = \emptyset$, $X^{NZ} = \emptyset$
for $x^C \in X^C$ **do**
  Finde x^Z, welches $||x^C - x^Z||$ minimiert
  $X^E \leftarrow X^E \cup \{x^Z\}$
for $x^Z \in X^E$ **do**
  Finde x^Q, welches $||x^Z - x^Q||$ minimiert
  $X^{NZ} \leftarrow X^{NZ} \cup \{x^Q\}$
return X^{NZ}

Die Zentren in Algorithmus 5 finden wir mit `k-means-Clustering`. Dafür wählen wir zunächst k beliebige Datenpunkte aus unserem gegebenen Datensatz als Zentren aus. Wir erstellen anhand dieser Zentren k Cluster, Gruppen aus Datenpunkten, indem wir für jeden Datenpunkt den Abstand zu den Zentren berechnen und ihn dem Zentrum zuordnen, welches am nächsten ist. Für jedes erstellte Cluster berechnen wir daraufhin den Schwerpunkt und wählen diesen als neues Zentrum. Dies wiederholen wir so oft, bis die Clusterzuweisung der Datenpunkte konstant bleibt. Das Ergebnis, das heißt die zuletzt gewählten Zentren, hängen stark von den initialen Zentren ab. Deswegen möchten wir eine `k-means++`-Initialisierung nutzen, wodurch wir anders als zuvor ein nachweislich gutes Ergebnis erhalten. Die Initialisierung funktioniert ähnlich zu `k-means-Clustering`. Anders als bei diesem wird bei `k-means++` das nächste Clusterzentrum aus den verbleibenden Datenpunkten mit einer Wahrscheinlichkeit, die proportional zu den zuvor berechneten Distanzen ist, gewählt. Das bedeutet, dass Punkte, die weiter von den bisherigen Zentren entfernt sind, eine höhere Wahrscheinlichkeit haben, ausgewählt zu werden. Mehr dazu findet sich in [SB14] und [AV07].

Durch das Finden der Zentren stellen wir sicher, dass in einer Umgebung Quelldaten liegen. Die Anzahl der Zentren übernehmen wir von [Gup+22] mit etwa 5% der Größe der Zieldaten.

Durch das Hinzufügen von geeigneten Quelldaten zu dem Zieldatensatz erhöhen wir die Varianz, die durch die durchschnittliche quadratische Abweichung der einzelnen Datenpunkte vom Mittelwert gemessen wird. Eine formale Definition findet sich in [MRT18]. Durch das Hinzufügen von geringem Rauschen reduzieren wir die Gefahr für Overfitting und erhöhen die Anpassungsfähigkeit unserer Hypothese, was zu einer höheren Robustheit führt.

Wir erhöhen zusammenfassend beim Variance Sampling (Algorithmus 5) durch das Hinzufügen von kleinen Verzerrungen im Zieldatensatz die Generalisierbarkeit unseres Algorithmus. Unter anderem ermöglicht uns dies, beim Erstellen der Gewichte den Aufwand des Algorithmus von `TTR2` zu reduzieren, wie wir im Folgenden genauer sehen werden.

4.2 Beschreibung des Algorithmus

Wir betrachten erneut die in Kapitel 3.3 beschriebenen Schwachpunkte von `TTR2`. Das Problem des negativen Transfers sind wir durch das Importance Sampling (Algorithmus 4) angegangen. Wir möchten uns im Folgenden die Aufteilung in zwei Phasen, insbesondere die Aktualisierung der Gewichte der Quell- und Zieldaten an unterschiedlichen Punkten, und die darausfolgende Komplexität genauer ansehen.

Durch das Importance Sampling haben wir weniger informative Quelldaten, die unter Umständen zu negativem Transfer führen können, entfernt. Dadurch ist eine Aufteilung in verschiedene Phasen des Gewichten und die darausfolgende stärkere Anpassung der Gewichte der Zieldaten, nicht notwendig. Anders als bei `TTR2`, können wir also im Folgenden die Gewichte gleichzeitig aktualisieren. Wir updaten dabei weiterhin die Zieldaten, um eine möglichst gute Hypothese zu finden.

Die Idee ist nun, dass durch den Aufruf von `AdaBoost.R2` mit den gewichteten Daten eine Hypothese berechnet wird. Die angepassten Fehler der Prognose durch die Hypothese auf den Daten werden dann benutzt, um die Gewichtungen zu aktualisieren. Dabei soll jedoch die Aktualisierung der Quelldaten stärker von den Fehlern abhängen, um weitere Verzerrungen durch die Quelldaten zu reduzieren.

Der Algorithmus 6 kann man in mehrere Schritte unterteilen:

1. **Vorbereitung:** Zunächst führen wir Importance und Variance Sampling durch und betrachten nun nur noch die aktualisierten Datensätze. Außerdem erhält jeder Datenpunkt ein Anfangsgewicht, wir setzen dieses auf $\frac{1}{q+z}$. Nun führen wir in jeder Iteration die folgenden Schritte durch:

2. **Erstellen einer Hypothese:** Wir rufen nun `AdaBoost.R2` auf, um eine Hypothese zu erhalten. Anders als bei `TTR2` werden hier, sowohl die gewichteten Ziel- als auch die gewichteten Quelldaten betrachtet, um eine Hypothese zu erstellen.

3. **Berechnen des zugeordneten Fehlers der Hypothese:** Am Ende des Algorithmus, nach N Iterationen, wird die Hypothese h_t ausgegeben, die den kleinsten zugeordneten Fehler hat. Die naheliegende Option wäre es hier, die Summe des gewichteten angepassten Fehler der Zieldaten als zugeordneten Fehler zu nehmen. Um möglichst allgemeingültige Hypothesen zu haben, möchten wir jedoch, dass die Hypothese nicht auf den Daten trainiert wird, auf denen der Fehler ausgerechnet wird. Deswegen nutzen wir ein F-faches Kreuzvalididerungsverfahren. Wir rufen in jedem Durchgang `AdaBoost.R2` auf, um eine Hypothese h_{t_j} auf den gewichteten $F-1$ Datenteilmengen der Zieldaten und den gewichteten Quelldaten zu berechnen. Danach speichern wir jeweils den Fehler $error_{t_j}$ der Hypothese h_{t_j}, um den Durchschnitt der Fehler der Kreuzvalidierungsverfahren als Fehler zugeordnet zur Hypothese h_t zu speichern. Dieser zugeordnete Fehler ist der Fehler, den wir am Ende des Algorithmus betrachten und insbesondere wird dieser nicht auf den gleichen Daten trainiert.

Algorithm 6: S-TrAdaBoost.R2

Eingabe: Zieldaten X^Z mit Y^Z der Größe m, Quelldaten X^Q mit Y^Q der Größe n, Anzahl an Iterationen N, Anzahl an Boostingiterationen N_{Boost}, Anzahl an Kreuzvalidierungen F, Lernalgorithmus L, Lernrate α

Ausgabe: Hypothese h_{t^*}

Erhalte neuen Quelldatensatz $(X^{Q'}, Y^{Q'})$ der Größe q mit Importance Sampling

Erhalte neuen Zieldatensatz $(X^{Z'}, Y^{Z'})$ der Größe z mit Variance Sampling

Sei $(X^A, Y^A) = (X^{Q'} \cup X^{Z'}, Y^{Q'} \cup Y^{Z'})$ die Aneinanderreihung der neuen Datensätze mit Gesamtgröße $q + z$

Initialisiere die Gewichte $w_i^{(1)} = \frac{1}{q+z}$ für alle $1 \leq i \leq q + z$

for $t = 1$ *to* N **do**

- Rufe `AdaBoost.R2` mit (X^A, Y^A) als Eingabedaten, N_{Boost} und L auf und erhalte Hypothese h_t
- **Berechne Fehler** $error_t$ **zugeordnet zur Hypothese** h_t **mithilfe eines Kreuzvalidierungsverfahren:**
  - Teile den Zieldatensatz in F Datenteilmengen $(X_f^{Z'}, Y_f^{Z'})$ für $f \in [F]$
  - **for** $j = 1$ *to* F **do**
    - Erhalte neue Trainingsdaten $(X_{train}, Y_{train}) = (X^A \setminus X_j^{Z'}, Y^A \setminus Y_j^{Z'})$
    - Rufe `AdaBoost.R2` mit N_{Boost}, (X_{train}, Y_{train}) und L auf und erhalte Hypothese h_{t_j}
    - Berechne den Fehler $error_{t_j}$ von h_{t_j} auf den Testdaten $(X_j^{Z'}, Y_j^{Z'})$
  - Berechne den h_t zugeordneten Fehler $error_t = \frac{1}{F} \sum_{j=1}^{F} error_{t_j}$
- Berechne den angepassten Fehler $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{\max_{j=1}^{(q+z)} |y_j - h_t(x_j)|}$ für $1 \leq i \leq q + z$
- Berechne den Gewichtungsfaktor der Quelldaten $\overline{\beta}_t = \frac{\sum_{i=1}^{q+z} w_i^{(t)} e_i^{(t)}}{1 - \sum_{i=1}^{q+z} w_i^{(t)} e_i^{(t)}}$
- Berechne den Gewichtungsfaktor der Zieldaten $\beta_t = \frac{z}{(q+z)} + \frac{t}{N-1}(1 - \frac{z}{q+z})$
- Aktualisiere die Gewichte

$$w_i^{(t+1)} = \begin{cases} \frac{w_i^{(t)} \overline{\beta}_t^{e_i^{(t)}}}{Z_t} \alpha, & 1 \leq i \leq q \\ \frac{w_i^{(t)} \beta_t^{1 - e_i^{(t)}}}{Z_t} \alpha, & q + 1 \leq i \leq q + z \end{cases}$$

- mit Z_t als Summe der Gewichte

return h_{t^*} mit $t^* = \arg\min_t \text{error}_t$

4. **Aktualisieren der Gewichte:** Wir berechnen zunächst den angepassten Fehler der Datenpunkte durch die Hypothese h_t. Weiterhin sind die Aktualisieren der Gewichte von dem angepassten Fehler im Exponenten der Gewichtungsfaktoren abhängig. Jedoch gibt es Unterscheidungen zu den Gewichtungsfaktoren aus `TTR2`.

Zieldaten: Die Gewichte der Zieldaten werden nun nicht vom Erstellen der Hypothese übernommen. Wir aktualisieren diese in demselben Schritt wie die Gewichte der Quelldaten. Anders als bei `TTR2`, bei dem die Gewichte der Zieldaten auch vom Gesamtfehler abhängig waren, ändern wir nun die Gewichte abhängig von einer Konstante. Die Konstante beträgt zunächst ungefähr das Verhältnis der Quell- und Zieldaten und wird mit jeder Iteration erhöht, sodass sie gegen 1 strebt. Die Zieldaten

werden dementsprechend, abhängig vom angepassten Fehler im Exponenten, mit jeder Iteration mit einem höheren Gewichtungsfaktor multipliziert. Da die Verzerrung durch die Quelldaten durch das Sampling reduziert ist, muss bei `S-TrAdaBoost.R2` (Algorithmus 6) im Gegensatz zu `TTR2` die Gesamtsumme der Zieldaten nicht mehr einem bestimmten Wert entsprechen, was das Berechnen der Gewichte der Quelldaten weniger komplex macht.

Quelldaten: Im Unterschied zu `TTR2` sind die Gewichte nicht mehr abhängig von der Gesamtgröße der Gewichte der Zieldaten, sondern von dem aufsummierten gewichteten angepassten Fehler der Hypothese auf beiden Datensätzen. Dadurch erhalten Quelldaten bei großem Fehler eine stärkere Strafe. Somit wird auch hier gegen die Verzerrung der Quelldaten gearbeitet ohne das Zwei-Phasen-Modell nutzen zu müssen.

Wie anfangs erwähnt, gab es Unterschiede zwischen dem bereitgestellten Code und dem Artikel. Einer davon bezieht sich auf die unterschiedlichen Gewichtungsfaktoren. Im Code wurde auch für die Zieldaten der Gewichtungsfaktor, der vom angepassten Gesamtfehler abhängt, verwendet. Experimente, die in Kapitel 5 näher erläutert werden, haben gezeigt, dass beide Versionen ähnliche Ergebnisse liefern.

Außerdem fügen wir eine Lernrate ein. Mit dieser können wir festlegen, wie stark die Gewichte aktualisiert werden. Jedoch sei vorausgreifend erwähnt, dass die bisherigen Experimente in Kapitel 5 nahelegen, dass die Lernrate keinen großen Einfluss auf die Hypothese hat. Aber die Anzahl der Experimente ist nicht ausreichend, um dies abschließend zu beantworten.

Insbesondere haben wir durch das Sampling und die daraus resultierende mögliche gleichzeitige Aktualisierung der Gewichte die Komplexität gegenüber `TTR2` reduziert.

Betrachten wir die theoretische Laufzeit von `S-TrAdaBoost.R2` (vgl. [Gup+22]):

Sei d die Dimension der Daten und S die Anzahl an Daten, das heißt $S = n + m$. Betrachten wir zunächst die Gesamtlaufzeit der `for`-Schleife. Entscheidend hierfür ist die Laufzeit des Erstellen eines Entscheidungsbaums. Dies dauert $O(dS^2log(S))$ (vgl. [Gup+22]). Pro Iteration erstellen wir $N_{Boost}F$ Entscheidungsbäume und aktualisierung die Gewichte in $O(q + z) = O(S)$. Nehmen wir an, dass N_{Boost} und F Konstanten sind, erhalten wir eine Laufzeit von $O(NdS^2log(S))$ für N Iterationen.

Bei der Vorbereitung hat das Variance Sampling eine Laufzeit von $O(dSk)$ durch das `k-means-Clustering` (vgl. [AV07]). Die Laufzeit des Importance Sampling und der Initialisierung ist darin enthalten.

Da $k \leq S$ ist, dominiert die Laufzeit der Entscheidungsbäume und wir erhalten insgesamt eine Laufzeit von $O(NdS^2log(S))$.

Wichtig ist für das Verfahren, eine gute Parameteranpassung und Größe der neuen Datensätze zu finden. Wir werden uns in Kapitel 5 praktischen Experimenten widmen, um `S-TrAdaBoost.R2` zu testen. Es sei an dieser Stelle jedoch angemerkt, dass wir Diskrepanzen zwischen dem zur Verfügung gestellten Code von [Gup+22] und dem hier aufgeschriebenen Algorithmus basierend auf dem Artikel [Gup+22] gefunden haben.

5 Numerische Experimente

In diesem Kapitel führen wir numerischen Experimente von `S-TrAdaBoost.R2` durch. Nach dem Vorstellen von Kostenfunktionen, vergleichen wir das Verfahren mit den in Kapitel 3 vorgestellten: `TTR2`, `KMM`, `KLIEP` und `IWKRR`. Dabei führen wir unterschiedliche Experimente durch, die die Notwendigkeit von Sampling sowie die Perfomance von `S-TrAdaBoost.R2` überprüfen.

5.1 Vorbereitung

5.1.1 Datensätze

In unseren Experimenten nutzen wir fünf „UCI Regression Datasets“, heruntergeladen von [Tor]. Die verwendeten Datensätze sind dabei in der nun folgenden Tabelle beschrieben. Von den Datensätzen *Ailerons* und *Elevators* war bereits eine Einteilung in Trainings- und Testdaten gegeben.

Datensätze	Beschreibung	(Anzahl, Dimension)
Ailerons	Der Datensatz beschreibt das Fliegen eines F-16-Flugzeugs. Die Kategorien bestehen aus Beschreibungen zum Status des Flugzeugs und es ist das Ziel, eine Steueraktion an den Querrudern (= *Ailerons*) des Flugzeugs vorherzusagen.	Training: (7154,41) Test: (6596,41)
Elevators	Dieser Datensatz beschreibt ebenfalls das Fliegen eines F-16-Flugzeugs, wobei hier das Ziel ist, eine Aktion an den Höhenrudern (= *Elevators*) des Flugzeugs vorherzusagen.	Training: (8752,19) Test: (7847,19)
Abalone	Anhand von physikalischen Messungen wird hierbei die Anzahl an Ringen auf den Schalen von Seeohren (= *Abalone*) vorhergesagt, womit das Alter vorhergesagt werden kann.	(4177,9)
Kinematics	Der folgende Datensatz beinhaltet Informationen über die Gelenkwinkel eines Roboters mit 8 Gliedern. Das Ziel ist hierbei die Vorhersage der Entfernung eines Glieds, das eine Aufgabe ausführt, von einem zuvor festgelegten Ort.	(8192,9)
Computer Activity	Der letzte Datensatz besteht aus einer Sammlung von Daten zur Computeraktivität und versucht den Anteil der Zeit vorherzusagen, in dem die CPUs im Benutzermodus ausgeführt werden.	(8192,22)

Standardisierung: Zunächst berechnen wir den Mittelwert und die Standardabweichung der Attribute bis auf die Labels. Anschließend skalieren und zentrieren wir die Attribute auf einen Mittelwert von 0 und eine Standardabweichung von 1. Dadurch werden unterschiedliche Einheiten oder Skalen einheitlicher, sodass einzelne Attribute nicht mehr aufgrund ihrer Größenordnung dominieren (vgl. [Ped+11]).

Aufteilung in Quell- und Zieldaten: Wir übernehmen hier das Verfahren von [Gup+22] und [PS10]. Da bei den Datensätzen *Ailerons* und *Elevators* bereits eine Aufteilung in Trainings- und Testdaten gegeben war, betrachten wir hier nun zwei verschiedene Vorgehensweisen:

Bei den Datensätzen *Ailerons* und *Elevators* nutzen wir die gegebene Aufteilung in Trainings- und Testdaten. Für die Quell- und Zieldaten teilen wir die Trainingsdaten in zwei zufällige Datenteilmengen, wobei die Zieldaten 5% der gegebenen Trainingsdaten ausmachen sollen.

Bei den drei anderen Datensätzen war anfänglich keine Aufteilung gegeben. Wir splitten zunächst die Daten in Quelldaten und verbleibende Daten auf, die später in Trainings- (= Zieldaten) und Testdaten unterteilt werden. Für die Abspaltung von Quelldaten nutzen wir ein moderat mit den Labels korellierendes Attribut, welches wir von [Gup+22] übernehmen und spalten anhand dessen die Daten in Quell- und weitere Daten. Die Aufteilung in Trainings- und Testdaten wird anhand eines Kreuzvalididerungsverfahrens gemacht. Wir trennen in erster Instanz die verbleibenden Daten in 20 Datenteilmengen und wählen pro Iteration eine Datenteilmenge aus, welche als Testdatensatz fungiert. Alle anderen werden zu unseren Trainingsdaten.

5.1.2 Kostenfunktionen

Um die Verfahren vergleichen zu können, betrachten wir zwei Kostenfunktionen, die wir auf unsere Testdaten anwenden. Wir nutzen jeweils einen Datensatz der Größe w, tatsächliche Labels y des Datensatzes und die prognostizierten Labels y' der berechneten Hypothese.

Definition 5.1 (Root mean sqared error ($RMSE$) (vgl. [MRT18])).
Der $RMSE$ eines gelabelten Datensatzes ist definiert als

$$RMSE(y, y') = \sqrt{\frac{1}{w}\sum_{i=1}^{w}(y_i - y_i')^2}$$

Der $RMSE$ liegt in $[0, 1]$ und ein kleinerer Wert entspricht einer besseren Vorhersage.

Definition 5.2 (Quadrat des Korrelationskoeffizienten nach Pearson (r^2) (vgl. [Vir+20])).
Wir definieren hierbei r^2 eines gelabelten Datensatzes als

$$r^2(y, y') = \left(\frac{\sum_{i=1}^{w}(y_i - \overline{y})(y_i' - \overline{y}')}{\sqrt{\sum_{i=1}^{w}(y_i - \overline{y})^2 \sum_{i=1}^{w}(y_i' - \overline{y}')^2}}\right)^2$$

mit $\overline{y}$ als den Mittelwert der tatsächlichen Label und $\overline{y}'$ entsprechend der prognostizierten Label.

Auch r^2 liegt in $[0, 1]$ und misst, wie gut die zu beobachtende Verbreitung, genannt Variabilität, der prognostizierten Label durch lineare Abhängigkeit zu den tatsächlichen Label erklärt werden kann. Hierbei entspricht ein größerer Wert einer höher erklärbaren Verbreitung.

5.1.3 Implementierung der Verfahren

Um Vergleichbarkeit zu schaffen, nutzen wir hierbei die gleichen Implementierungen wie [Gup+22]. Wir werden außerdem in jedem Experiment zunächst eine dann beschriebene Art des Samplings ausführen. Daraufhin werden auf den veränderten Quell- und Zieldaten in 20 Durchgängen die ausgewählten Verfahren ausgeführt, wobei wir aus jeder Iteration den $RMSE$ und r^2 speichern. Die Ergebniswerte sind jeweils die Durchschnitte von $RMSE$ und r^2 aus den 20 Durchgängen. Bei den Datensätzen *Abalone*, *Kinematics* und *Computer Activity* sind dies gerade die Durchgänge, in denen wir ein Kreuzvalidierungsverfahren anwenden. Das heißt, diese Datensätze werden zunächst in 20 Datenteilmengen geteilt und in jedem der Durchgänge wird auf einer anderen Datenteilmenge getestet. Die anderen werden dabei als Trainingsdaten verwendet. Als Lernalgorithmus nutzen wir einen Entscheidungsbaum mit maximaler Tiefe von sechs. Als Kernel wird der Gauß-Kernel verwendet.

`KLIEP`, `KMM` und `TTR2`: Wir nutzen hierbei die Implementierungen von [Mat+21] mit den voreingestellten Parametern bis auf die Boostingiterationen bei `TTR2`, die wir wie [Gup+22] auf 100 erhöhen.

`IWKRR`: Hier nutzen wir die Implementierung von [Rui19] mit den empfohlenen Parametern von [Rui19]. Beim Durchführen von `IWKRR` sind wir auf das Problem gestoßen, dass die Endgewichte w_j aus Gleichung 3.2 negativ werden können, da α_i negativ werden kann. Da jedoch von den Gewichten die Wurzel genommen wird, erhalten wir not-a-number-Werte, mit denen eine weitere Berechnung von Gleichung 3.2 nicht möglich ist. Deswegen haben wir folgendes Experiment durchgeführt:

Die erste Zeile von 1 ist das Aufrufen von `IWKRR`. Wir mussten feststellen, dass bei den *Elevators*, *Abalone* und *Kinematics* Datensätzen negative Gewichte auftreten und der Code abbricht. Deswegen führen wir vier Veränderungen durch, die das Problem beheben sollen. Dabei verändern wir in den ersten drei Schritten die Gewichte, bevor wir die Wurzel von diesen ziehen: Bei den ersten beiden Versuchen nehmen wir den Betrag von den Gewichten, bei dem zweiten Versuch multiplizieren wir die Gewichte nach dem Wurzel ziehen noch mit dem eigentlichen Vorzeichen und bei dem dritten Versuch verschieben wir die Gewichte um $|\min(sw)| + 1$ und teilen zur Normierung noch durch den maximalen Gewichtswert. Die 1 ist hierbei willkürlich gewählt gewesen. Beim letzten Versuch lassen wir das Ziehen der Wurzel weg.

Wir verwenden hierbei zunächst Importance Sampling und Variance Sampling wie in [Gup+22] beschrieben und testen auf den veränderten Quell- und Zieldaten in 20 Durchgängen die verschiedenen Implementierungen von `IWKRR`.

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
`IWKRR`	0.00027	0.58	error	error	error	error	error	error	8.50	0.63
Betrag	0.00027	0.58	0.0046	0.52	1.60	0.44	0.11	0.82	8.50	0.63
Vorzeichen	0.00027	0.58	0.0046	0.52	1.60	0.44	0.11	0.82	8.50	0.63
Verschiebung	0.00026	0.62	0.0041	0.62	1.58	0.44	0.10	0.83	7.76	0.67
ohne Wurzel	0.00027	0.58	0.0048	0.50	1.61	0.44	0.11	0.82	8.50	0.63

Tabelle 1: `IWKRR`: Testen von Möglichkeiten zur Fehlerbehebung. Die Werte sind jeweils der Durchschnitt von 20 Durchgängen und vorher wurde Sampling (vgl. Kapitel 4) durchgeführt.

Wir stellen fest, dass die Ergebnisse in allen Versuchen außer bei der Veränderung,

bei der die Gewichte verschoben werden, übereinstimmen, auch bei den Datensätzen, bei denen keine negativen Gewichte auftreten. Insbesondere sind die Ergebnisse beim Verschieben besser als bei den anderen Versuchen. Interessant wäre es, da 1 als Beispiel gewählt worden ist, hier noch weitere Experimente durchzuführen unter anderem mit der Weite der Verschiebung. Aus Zeitgründen und dem Fokus auf `S-TrAdaBoost.R2` werden wir dies hier nicht weiter verfolgen und die Wurzel in weiteren Aufrufen weglassen.

`S-TrAdaBoost.R2`**:** Wir übernehmen hier den Algorithmus des bereitgestellten Codes von [Gup+22]. Die Parameter wurden hierbei von [Dai+07] übernommen.

Wir führen außerdem zwei Experimente mit `S-TrAdaBoost.R2` durch, die im Folgenden beschrieben werden. Wir verwenden dabei zunächst Importance Sampling und Variance Sampling wie in [Gup+22] beschrieben und testen auf den veränderten Quell- und Zieldaten in 20 Durchgängen die verschiedenen Implementierungen von `S-TrAdaBoost.R2`.

Beim Durchgehen des von [Gup+22] bereitgestellten Code ist uns aufgefallen, dass bei der Aktualisierung der Gewichte sowohl für die Quelldaten als auch für die Zieldaten der vom angepassten Fehler abhängige Gewichtungsfaktor $\overline{\beta}_t$ verwendet wurde. Die Autoren haben den folgenden Schritt

---

Algorithm 7: Aktualisierung der Gewichte nach [Gup+22]

Berechne den Gewichtungsfaktor $\overline{\beta}_t = \frac{\eta_t}{1-\eta_t}$ mit $\eta_t = \sum_{i=1}^{p+q} w_i^{(t)} e_i^{(t)}$ und
$\beta_t = \frac{q}{(p+q)} + \frac{t}{N-1}(1 - \frac{q}{p+q})$
Aktualisiere die Gewichte

$$w_i^{(t+1)} = \begin{cases} \frac{w_i^{(t)} \overline{\beta}_t^{e_i^{(t)}}}{Z_t} \alpha, & 1 \leq i \leq p \\ \frac{w_i^{(t)} \beta_t^{1-e_i^{(t)}}}{Z_t} \alpha, & p \leq i \leq p+q \end{cases}$$

---

durch

Algorithm 8: Aktualisierung der Gewichte nach dem bereitgestellten Code

---

Berechne den Gewichtungsfaktor $\overline{\beta}_t = \frac{\eta_t}{1-\eta_t}$ mit $\eta_t = \sum_{i=1}^{p+q} w_i^{(t)} e_i^{(t)}$
Aktualisiere die Gewichte

$$w_i^{(t+1)} = \begin{cases} \frac{w_i^{(t)} \overline{\beta}_t^{e_i^{(t)}}}{Z_t} \alpha, & 1 \leq i \leq p \\ \frac{w_i^{(t)} \overline{\beta}_t^{1-e_i^{(t)}}}{Z_t} \alpha, & p \leq i \leq p+q \end{cases}$$

---

in `S-TrAdaBoost.R2` ersetzt.

Um die Auswirkung dessen zu erkennen, testen wir nun auf allen fünf Datensätzen den bereitgestellten Code mit dem gleichen Gewichtungsfaktor für Quell- und Zieldaten gegen den Code wie in [Gup+22] (in Kapitel 4.2 erläutert) mit zwei verschiedenen Gewichtungsfaktoren.

Wir stellen fest, dass die Ergebnisse fast identisch sind, wir sehen beim *Kinematics* und *Computer Activity* Datensatz eine minimale Verbesserung bei der Version, die

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
Nach Code	0.00019	0.79	0.0039	0.67	1.65	0.46	0.17	0.57	2.38	0.92
Nach Artikel	0.00019	0.70	0.0039	0.67	1.66	0.45	0.15	0.66	2.34	0.91

Tabelle 2: `S-TrAdaBoost.R2`: Testen von zwei verschiedenen Arten von Gewichtungsfaktoren. Die Werte sind jeweils der Durchschnitt von 20 Durchgängen und vorher wurde Sampling (vgl. Kapitel 4) durchgeführt.

in Kapitel 4 präsentiert wurde. Da es nach dem Experiment keinen großen Einfluss hat, welche Art wir verwenden, werden wir im Folgenden den bereitgestellten Code verwenden. Jedoch wäre es spannend noch weiter zu untersuchen, warum unterschiedliche Gewichtungsfaktoren auf den ausgewählten Datensätzen einen kleineren Einfluss als angenommen haben.

Außerdem sind uns Unklarheiten bezüglich der Gewichtungen aufgefallen. Im Artikel [Gup+22] wird geschrieben, dass sie Z_t jeweils als die Summe der gesamten Summe wählen. Dabei ist uneindeutig, ob sich die Summe auf die vorherigen Gewichte oder die aktuellen Gewichte bezieht. In `TTR2` ist dies gerade die Summe der aktuellen Gewichte und damit werden die Gewichte normalisiert. In dem bereitgestellten Code ist Z_t die Summe der vorherigen Gewichte. Da die Lernrate im Algorithmus mit den Gewichten multipliziert wird, vergleichen wir deswegen drei verschiedene Arten: den bereitgestellten Code, das Normalisieren der Gewichte und das Normalisieren der Gewichte multipliziert mit der Lernrate. Dies sind die Ergebnisse unseres Experiments:

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
Vorherige Gewichte	0.00020	0.77	0.0039	0.69	1.66	0.45	0.17	0.58	2.41	0.92
Normalisieren mit $\alpha = 0.1$	0.00020	0.77	0.0039	0.68	1.66	0.45	0.17	0.58	2.39	0.92
Normalisieren	0.00020	0.77	0.0039	0.69	1.65	0.46	0.17	0.57	2.40	0.92

Tabelle 3: `S-TrAdaBoost.R2`: Testen von verschiedenen Arten von Z_t. Die Werte sind jeweils der Durchschnitt von 20 Durchgängen und vorher wurde Sampling (vgl. Kapitel 4) durchgeführt.

Wir können insgesamt keinen signifikanten Unterschied zwischen den Experimenten festmachen. Es scheint, als würden die Wahl von Z_t keinen großen Unterschied machen, solange alle Gewichte durch diese geteilt werden, zumindest für die Wahl der Summe der vorherigen Gewichte und der derzeitigen Gewichte. Auch wenn wir nun das Normalisieren mit und ohne Lernrate vergleichen, stellen wir fest, dass dies in den Kostenwerten keinen signifikanten Unterschied macht. Die beiden Versuche mit und ohne Lernrate deuten darauf hin, dass diese gegebenfalls keinen Einfluss hat. Jedoch sollten hier noch weitere Experimente durchgeführt werden mit anderen gewählten Lernraten. Spannend wäre es dabei, unterschiedliche Lernraten für Quell- und Zieldaten zu testen mit der Hoffnung das Übergewicht von Quelldaten und die daraus möglich resultierende Verzerrung reduzieren.

Wir werden im weiteren Verlauf aufgrund der Ähnlichkeit der Ergebnisse weiterhin den bereitgestellten Code verwenden.

5.2 Durchführung

Auch hier werden wir in jedem Experiment zunächst eine jeweils beschriebene Art von Sampling ausführen und daraufhin auf den veränderten Quell- und Zieldaten in 20 Durchgängen die ausgewählten Verfahren durchführen, wobei wir aus jeder Iteration den $RMSE$ und r^2 speichern. Die Werte in der Tabelle sind jeweils die Durchschnitte von $RMSE$ und r^2 aus den 20 Durchgängen. Bei den Datensätzen *Abalone*, *Kinematics* und *Computer Activity* sind dies wie oben beschrieben gerade die Durchgänge, in denen wir ein Kreuzvalidierungsverfahren anwenden.

In [Gup+22] werden zwei verschiedene Experimente durchgeführt. Das erste besteht darin, dass die Verfahren wie in Kapitel 3 und Kapitel 4 getestet werden. Da Sampling auf jedem Datensatz und vor jedem Verfahren durchgeführt werden kann, führen sie als zweites Experiment alle Verfahren mit vorherigem Sampling (vgl. Kapitel 4) durch. Unser Ziel besteht in diesem Kapitel darin, die Experimente zu reproduzieren. Dabei konzentrieren wir uns insbesondere auf das zweite Experiment, da dies nach [Gup+22] gleiche bzw. bessere Ergebnisse geliefert hat.

Beim Analysieren des bereitgestellten Codes wurden Unstimmigkeiten zwischen dem bereitgestellten Code von [Gup+22] und dem Verfahren erklärt in Kapitel 4 gefunden. Beim Variance Sampling wurden mehrfach Quell- und Zieldaten vertauscht. Darüber hinaus war die Reihenfolge von Importance Sampling und Variance Sampling vertauscht. Deswegen möchten wir im ersten Schritt das Sampling von dem bereitgestellten Code von [Gup+22] mit einem selbstgeschriebenen Code nach dem beschriebenen Verfahren in Kapitel 4 vergleichen. Da das Sampling vom bereitgestellten Code zufällig erscheint, stellen sich die Fragen, ob die beiden Schritte Variance Sampling und Importance Sampling und Sampling allgemein notwendig ist, Wir machen deswegen im weiteren Verlauf sieben Experimente:

1. Ausführen der Verfahren mit Sampling von dem bereitgestellten Code
2. Ausführen der Verfahren mit eigenem Code nach dem beschriebenen Sampling in Kapitel 4 (Importance Sampling und Variance Sampling)
3. Ausführen der Verfahren mit eigenem Code (vgl. Kapitel 4), jedoch mit umgekehrter Reihenfolge von Variance Sampling und Importance Sampling
4. Ausführen der Verfahren ohne Sampling
5. Ausführen der Verfahren mit uniformen zufälligen Sampling
6. Ausführen der Verfahren mit Variance Sampling (vgl. Kapitel 4)
7. Ausführen der Verfahren mit Importance Sampling (vgl. Kapitel 4)

Das Sampling wird immer vor den Verfahren durchgeführt. Bei dem fünften Experiment wählen wir Daten zufällig mit einer uniformen Verteilung aus den Quelldaten aus, die zu dem Zieldatensatz hinzugefügt werden und entfernen analog Daten. Die Anzahl der Daten übernehmen wir dabei aus dem bereitgestellten Code von [Gup+22].

Schreiben wir nun im Folgenden `S-TrAdaBoost.R2`, so sprechen wir nur von dem Algorithmus nach dem Sampling, da das Sampling sich in den folgenden Experimenten unterscheidet und bei einem Experiment sogar weggelassen wird.

5.3 Ergebnisse

Vergleichen wir nun die Ergebnisse. Hierbei sind die jeweils besten Werte, d.h. die höchsten Werte von r^2 und niedrigsten Werte vom $RMSE$, fettgedruckt und wir kürzen `S-TrAdaBoost.R2` mit `S-TR` ab.

Ergebnisse mit dem bereitgestellten Code und mit Sampling aus dem Artikel:

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
`S-TR`	**0.00019**	**0.78**	**0.0036**	**0.77**	**1.70**	**0.48**	0.19	0.50	**3.38**	**0.92**
`TTR2`	0.00022	0.70	0.0043	0.56	1.73	0.42	0.20	0.41	3.59	0.90
`IWKRR`	0.00027	0.60	0.0042	0.60	1.75	0.47	**0.11**	**0.80**	9.05	0.67
`KMM`	0.00029	0.49	0.0062	0.37	2.51	0.04	0.27	0.08	12.18	0.16
`KLIEP`	0.00026	0.60	0.0053	0.35	2.66	0.07	0.26	0.12	11.95	0.19

Tabelle 4: Ergebnisse mit dem Sampling des bereitgestellten Codes von [Gup+22]: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
`S-TR`	**0.00019**	**0.78**	**0.0039**	**0.67**	1.63	**0.45**	0.17	0.58	**2.39**	**0.92**
`TTR2`	0.00022	0.70	0.0048	0.46	1.67	0.41	0.20	0.40	2.63	0.91
`IWKRR`	0.00027	0.60	0.0041	0.61	**1.61**	0.44	**0.11**	**0.82**	8.50	0.63
`KMM`	0.00029	0.49	0.0064	0.15	2.43	0.07	0.26	0.13	10.74	0.19
`KLIEP`	0.00026	0.60	0.0066	0.14	2.61	0.09	0.26	0.13	10.65	0.21

Tabelle 5: Ergebnisse mit Sampling wie in Kapitel 4 beschrieben: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

Beim Vergleichen der beiden Experimente sehen wir, dass die Ergebnisse ähnlich sind. Bei dem Sampling nach Kapitel 4 erkennen wir für mehrere Verfahren eine leichte Verbesserung des $RMSE$ bei den Datensätzen *Elevators*, *Abalone* und *Kinematics* und eine deutlichere Verbesserung beim Datensatz *Computer Activity*. Für r^2 ist keines der beiden Experimente eindeutig besser.

Vergleichen wir dies nun mit den Ergebnissen, die Gupta et al. in ihrem Artikel [Gup+22] präsentieren, Für die Datensätze *Ailerons* und *Kinematics* stimmen unsere Ergebnisse bis auf minimale Abweichungen mit den Ergebnissen aus dem Artikel [Gup+22] überein. Bei dem *Elevators*-Datensatz und dem *Computer Activity* Datensatz beim 1. Experiment sehen wir eine deutliche Verschlechterung in unseren Ergebnissen. Im Gegensatz dazu sind unsere Ergebnisse beim *Abalone*-Datensatz deutlich besser. Dies kann daran liegen, dass die Aufteilung in Quell- und Zieldaten bei den *Elevators* zufällig gewählt wird, genauso wie die Aufteilung der Ziel- und Quelldaten bei den Datensätzen *Abalone* und *Computer Activity*. Jedoch können wir bestätigen, dass `S-TrAdaBoost.R2` in den meisten Fällen die besten Fehlerwerte und immer bessere Werte als `TTR2` hat.

Außerdem können wir die Verteilung der Ergebnisse, die [Gup+22] angeben, bestätigen (siehe Abbildung 23 in Anhang A). Die Datensätze *Computer Activity* und *Abalone* haben auch bei uns höhere Schwankungen in den Ergebnissen.

Vergleichen wir dies nun mit den Verfahren ohne vorheriges Sampling.

Ergebnisse ohne Sampling:

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
`S-TR`	**0.00018**	**0.82**	**0.0038**	**0.68**	2.39	0.43	0.21	0.38	**3.85**	**0.94**
`TTR2`	0.00022	0.71	0.0046	0.50	2.19	0.31	0.22	0.33	5.69	0.88
`IWKRR`	0.00026	0.60	0.0044	0.57	**1.56**	**0.45**	**0.10**	**0.85**	8.78	0.66
`KMM`	0.00022	0.72	0.0064	0.15	4.48	0.08	0.27	0.15	8.16	0.85
`KLIEP`	0.00021	0.73	0.0064	0.16	4.47	0.10	0.26	0.17	7.62	0.87

Tabelle 6: Ergebnisse ohne vorheriges Sampling: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

Bemerkung 5.3. Beim Aufruf von `KLIEP` auf dem *Abalone*-Datensatz ist für ein Experiment $r^2 = \textit{not a number}$ herausgekommen. Es scheint, als wäre die Eingabe von r^2 konstant und das führt zu der Rechnung $0/0$. Da wir uns auf `S-TrAdaBoost.R2` konzentrieren, haben wir diese Iteration entfernt und nur den Durchschnitt der anderen 19 Experimente hier eingetragen.

Insgesamt sind die Werte zu unseren Ergebnissen mit dem Code aus dem Artikel ähnlich. Wir sehen bei dem *Abalone*-Datensatz eine leichte Verbesserung von `IWKRR`, jedoch eine deutliche Verschlechterung von `S-TrAdaBoost.R2` und `TTR2`, welche auch auf dem *Computer-Acitivity*-Datensatz zu sehen ist. Bei diesem Datensatz hat sich auch `IWKRR` verschlechtert. Bei `KLIEP` und `KMM` sehen wir auf verschiedenen Datensätzen sowohl Verbesserungen als auch Verschlechterungen.

Aufgrund der uneindeutigen Ergebnisse stellt sich die Frage, inwiefern Sampling die Verfahren im Allgemeinen verbessert. Wir können bisher nur feststellen, dass `S-TrAdaBoost.R2` vom Sampling profitiert. Es stellt sich trotzdem die Frage, aufgrund der ähnlichen Ergebnisse trotz unterschiedliches Sampling, inwiefern Sampling die Verfahren verbessert. Dafür führen wir zwei Experimente mit zufälligem Sampling durch.

Ergebnisse mit uniform zufälligem Sampling:

	Ailerons		*Elevators*		*Abalone*		*Kinematics*		*C. Activity*	
Fehler	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2	*RMSE*	r^2
`S-TR`	**0.00018**	**0.79**	**0.0038**	**0.68**	2.00	**0.43**	0.19	0.50	3.32	**0.94**
	0.00019	**0.79**	**0.0039**	**0.67**	1.88	0.46	0.19	0.51	3.26	**0.94**
`TTR2`	0.00022	0.70	0.0047	0.49	2.15	0.37	0.21	0.37	**2.98**	0.93
	0.00022	0.70	0.0048	0.47	2.08	0.41	0.21	0.38	**2.92**	0.92
`IWKRR`	0.00026	0.61	0.0044	0.57	**1.77**	**0.43**	**0.11**	**0.81**	8.93	0.67
	0.00026	0.60	0.0044	0.57	**1.74**	**0.43**	**0.11**	**0.81**	8.92	0.67
`KMM`	0.00025	0.62	0.0066	0.12	4.82	0.02	0.28	0.15	8.23	0.84
	0.00025	0.64	0.0066	0.14	3.46	0.09	0.27	0.17	7.23	0.84
`KLIEP`	0.00023	0.68	0.0064	0.15	4.55	0.05	0.27	0.16	8.28	0.86
	0.00023	0.69	0.0067	0.14	3.90	0.08	0.26	0.19	7.82	0.86

Tabelle 7: Ergebnisse mit zufälligem Sampling mit zwei Durchläufen: jeweils Durchschnitt von 20 Durchgängen.

Bemerkung 5.4. Beim Aufruf von `KLIEP` auf dem *Computer-Activity*-Datensatz ist für das erste Experiment $r^2 = not\,a\,number$ herausgekommen. Wir sind analog zu Bemerkung 5.3 vorgegangen.

Bis auf den Datensatz *Abalone* sehen wir eine Verbesserung der Verfahren `KLIEP` und `KMM`, wobei die Ergebnisse ähnlich zu denen ohne Sampling sind. Spannend ist jedoch zu sehen, dass bei beiden zufälligen Experimenten bei dem Datensatz *Computer Activity* `TTR2` bessere Ergebnisse liefert als `S-TrAdaBoost.R2`. Bei *Abalone* sehen wir eine Verschlechterung im Vergleich zu dem Experiment mit Sampling und bis auf `KLIEP` und `KMM` in einem Experiment eine Verbesserung im Vergleich zu ohne Sampling.

Insgesamt können wir feststellen, dass je nach Verfahren ohne Sampling oder wie im Artikel beschrieben fast bei allen Durchläufen besser ist als mit zufälligem Sampling. Wir möchten nun weitere Experimente machen, in denen wir das beschriebene Sampling genauer analysieren:

Ergebnisse von weiteren Kombinationen aus Sampling:

Wir vergleichen hier zusätzlich die umgekehrte Reihenfolge von dem im Artikel beschriebenen Sampling sowie nur Variance Sampling und nur Importance Sampling.

Zunächst ist in Abbildung 11 zu erkennen, dass das Ändern der Reihenfolge von Importance Sampling und Variance Sampling keinen signifikanten Einfluss hat. Die Ergebnisse für die ersten drei Verfahren sind meisten dann am besten, wenn im Vorhinein Importance Sampling durchgeführt wurde. Das könnte erklären, weshalb das Sampling vom bereitgestellten Code ähnliche Ergebnisse geliefert hat wie das Sampling nach Kapitel 4, da bei dem bereitgestellten Code fürs Sampling nur beim Variance Sampling Fehler waren. Wir sehen darüber hinaus bei *Kinematics*, dass `IWKRR` mit Abstand die besten Erebnisse liefert. Bei *Computer Activity* lässt sich erkennen, dass für `S-TrAdaBoost.R2` und `TTR2` besonders Importance Sampling wichtig ist, jedoch für `KLIEP` und `KMM` ohne Sampling oder nur Variance Sampling. Das letzte sieht man auch bei dem Datensatz *Ailerons*. Spannend ist bei *Kinematics*, dass hierbei die Kombination aus Variance Sampling und Importance Sampling wichtig ist.

Zusammengefasst scheint die Reihenfolge keinen großen Einfluss zu haben und je nach Datensatz verschiedene Arten von Sampling sinnvoll zu sein.

Gesamtvergleich:

Da wir jeweils die gleichen Quell- und Zieldaten verwendet haben, können wir pro Datensatz die besten Ergebnisse insgesamt begutachten:

Ailerons	`S-TrAdaBoost.R2` ohne Sampling oder nur Variance Sampling
Elevators	`S-TrAdaBoost.R2` mit bereitgestelltem Code
Abalone	`IWKRR` ohne Sampling oder nur Importance Sampling
Kinematics	`IWKRR` ohne Sampling
Computer Activity	`S-TrAdaBoost.R2` mit Sampling wie im Artikel

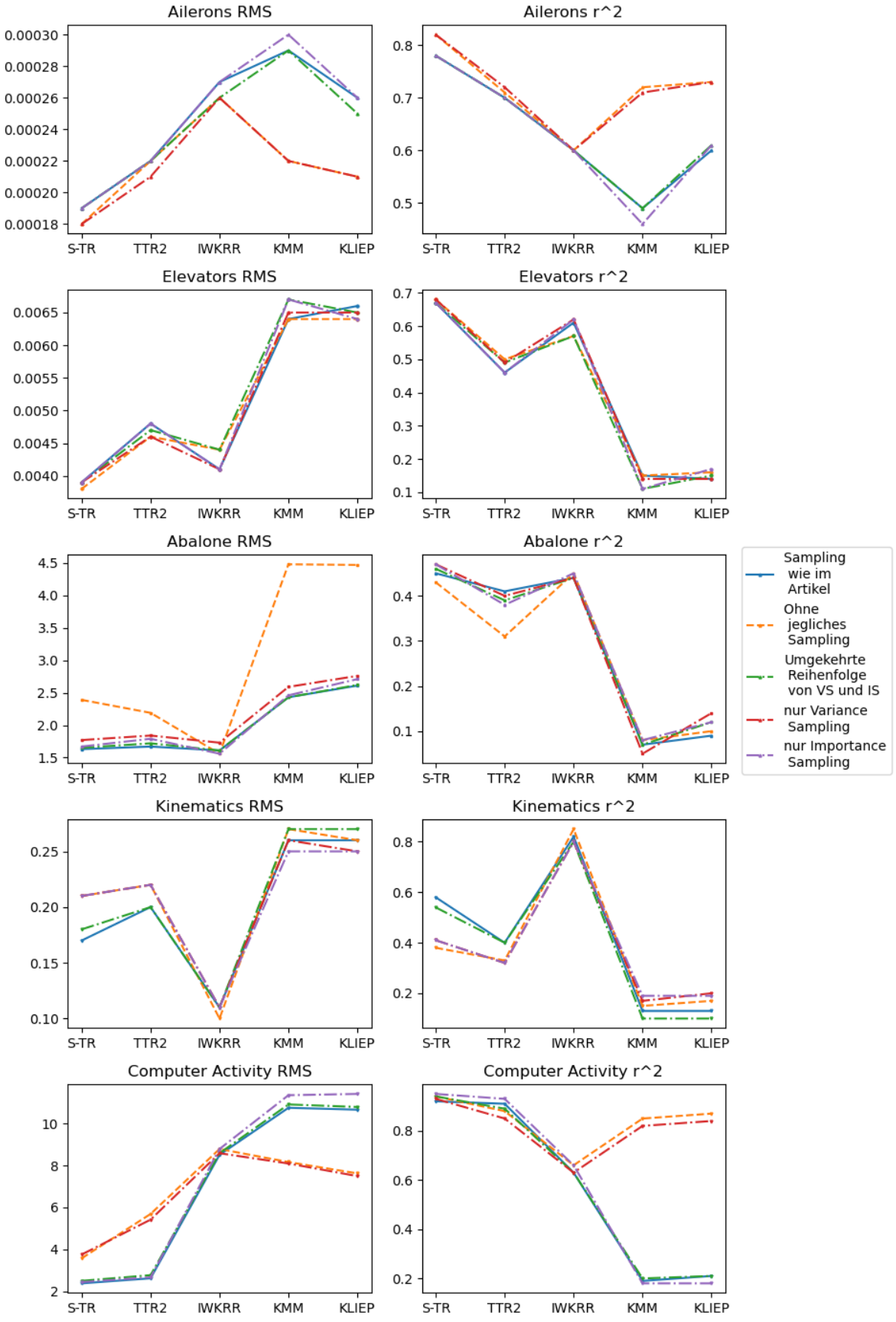

Abbildung 11: Ergebnisse von verschiedenen Kombinationen von Sampling

Wir müssen an dieser Stelle jedoch anmerken, dass fast alle Ergebnisse von *Ailerons* und *Elevators* ähnlich zueinander sind. Bei *Abalone* hat zwar `IWKRR` durchweg die besten Ergebnisse, jedoch liefert `S-TrAdaBoost.R2` mit dem wie im Artikel beschriebenen Sampling nahe Ergebnisse. Wir können dennoch feststellen, dass `S-TrAdaBoost.R2` fast immer gleichwertige oder bessere Ergebnisse als `TTR2` liefert und nur `IWKRR` je nach Datensatz bei den Ergebnissen von `S-TrAdaBoost.R2` mithalten kann.

Aus den Experimenten wird zudem deutlich, dass `S-TrAdaBoost.R2` im Durchschnitt am besten mit dem wie im Artikel beschrieben Sampling funktioniert, wobei häufig vor allem Importance Sampling ausschlaggebend ist. Dasselbe gilt für `TTR2`. Das Verfahren `IWKRR` läuft am besten ohne vorheriges Sampling, `KLIEP` und `KMM` mit vorherigem Variance Sampling. Bei `KLIEP` und `KMM` möchten wir eine Vermutung anstellen, woran das liegen könnte: In [Van16] wird erwähnt, dass die beiden Verfahren von einer weiteren Annahme ausgehen, nämlich $p^Z(y|x) = p^Q(y|x)$. Haben wir nun mehr Quell- und Zieldaten gegeben, für die diese Annahme gilt, so werden `KLIEP` und `KMM` vermutlich bessere Ergebnisse liefern. Durch das Variance Sampling haben wir gerade eine höhere Überschneidung von Quell- und Zieldaten gegeben. Wegen der hohen Anzahl an ähnlichen Ergebnissen und keiner eindeutigen Art von Sampling, möchten wir die Datensätze genauer betrachten:

5.4 Analyse der Aufteilung der Daten

Für die ersten beiden Datensätze *Ailerons* und *Elevators* betrachten wir die Aufteilung in Test-, Ziel- und Quelldaten. Da wir bei den drei anderen Datensätzen ein Kreuzvalidierungsverfahren anwenden und dabei unterschiedliche Ziel- und Testdaten erhalten, betrachten wir hier nur die Aufteilung in Quelldaten und den Rest, der dann in Ziel- und Testdaten geteilt wird. Zur Vereinfachung werden wir bei diesen drei Datensätzen alle Daten, die nicht als Quelldaten verwendet werden, im weiteren Verlauf als Zieldaten bezeichnen. Wir werden zusätzlich zu der Analyse der Aufteilung eine Visualisierung der Daten nach dem Sampling aus [Gup+22] machen.

Um die Daten mit [Hun07] zu visualisieren, nutzen wir abseits von der Funktion *parallel_coordinates* von [McK10] Verfahren zur Dimensionsreduktion. Wir verwenden Implementierungen von `PCA` und `t-SNE` aus [Ped+11].

5.4.1 Verfahren zur Dimensionsreduktion

Dieses Unterkapitel basiert auf [Mur22] und [Ped+11].

`PCA`: Die `Principal Component Analysis` (`PCA`) basiert auf der Idee, die Achsen nach der maximalen Varianz zu rotieren und die Daten auf die Achsen zu projizieren, die am meisten Varianz beschreiben. Diese nennen wir Hauptkomponenten.

Beispiel 5.5. *Betrachten wir Abbildung 12. Die beiden Achsen, die die Varianz der Daten beschreiben, sind eingezeichnet. Die Achse, die nach rechts oben geht, beschreibt den größten Anteil der Varianz, da die Punkte in diese Richtung die größten Abweichungen zum Durchschnitt haben. Die zweite Achse beschreibt den restlichen Anteil der Varianz. Möchten wir nun eine Dimensionsreduktion durchführen, können wir die erste Hauptkomponenente als Basis für die Datenpunkte wählen.*

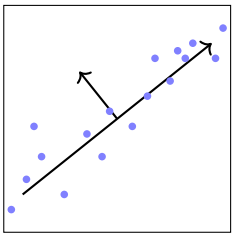

Abbildung 12: Beispiel für die Achsen, die die Varianz beschreiben

Das Verfahren `PCA` versucht dabei, möglichst die globale Struktur der Daten bei der Reduktion zu erhalten. Im Gegensatz dazu konzentiert sich `t-SNE` mehr auf die lokalen Strukturen.

`t-SNE`**:** versucht die Ähnlichkeiten der Datenpunkte während der Abbildung in einen niedrigdimensionalen Raum beizubehalten. Dafür werden die Ähnlichkeiten in Wahrscheinlichkeiten umgewandelt und versucht, die Kullback-Leibler-Divergenz zwischen den Wahrscheinlichkeiten der niedrigdimensionalen und den höherdimensionalen Daten zu minimieren.

Bemerkung 5.6. Da die Kostenfunktion, die bei `t-SNE` minimiert wird, nicht konvex ist, haben wir keine optimale Lösung, weswegen sich die Bilder im Folgenden leicht unterscheiden können, obgleich es derselbe Datensatz ist.

Wir werden im Folgenden bei der Dimensionsreduktion die Daten immer auf einen zweidimensionalen Raum abbilden.

5.4.2 Ergebnisse der Analyse zur Aufteilung

Im Allgemeinen werden die Zieldaten gelb, Quelldaten rot und Testdaten, falls vorhanden, grün sein. Zu besseren Sichtbarkeit stellen wir jeden Plot mehrfach da mit verschiedenen Teilen unseres Datensatzes im Vordergrund.

Ailerons Wir visualisieren die Daten zunächst nach ihren Koordinaten:

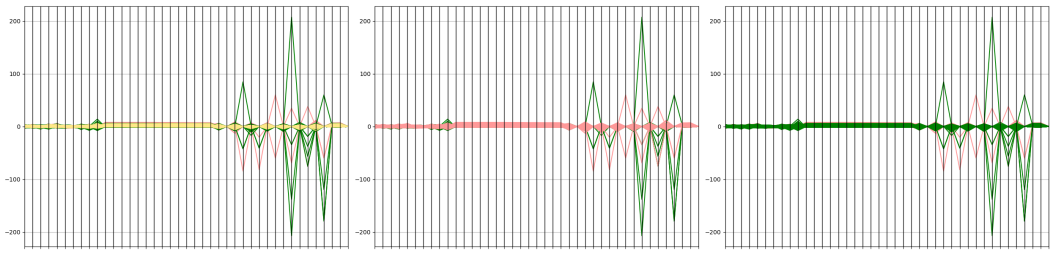

Abbildung 13: Visualisierung des Datensatzes *Ailerons* aufgeteilt in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

Wir sehen, dass die Quell- und Testdaten mehr Schwankungen in ihren Werten bei den letzten Attributen haben. Betrachten wir dies nun nach dem Importance Sampling und Variance Sampling:

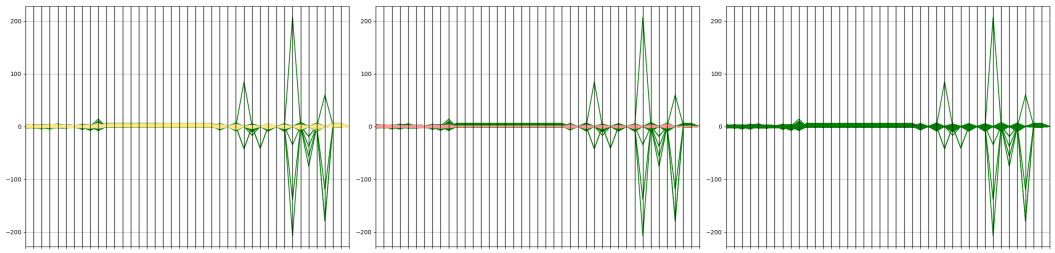

Abbildung 14: Visualisierung des gesampelten Datensatzes *Ailerons* aufgeteilt in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün)
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

Nun sind die Schwankungen der Quelldaten verschwunden. Dies ergibt Sinn, da wir die vom Durchschnitt der Zieldatenpunkte am weitesten entfernten Quelldatenpunkte entfernt haben. Wir wissen nun, dass alle Daten, bis auf wenige Testdatenpunkte, in den einzelnen Attributen ähnliche Werte annehmen. Jedoch sagt uns dies noch nicht, wie nah sich Quell-, Ziel- und Testdaten sind. Da bei dem Datensatz die Varianz auf fast alle Achsen gleichwertig aufgeteilt ist und damit eine Dimensionsreduktion mit `PCA` nicht hilfreich ist, verwenden wir `t-SNE`:

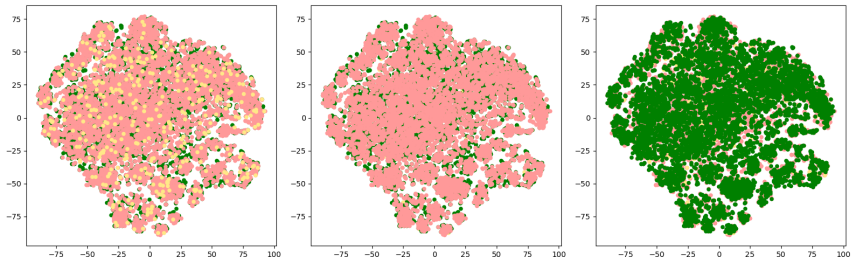

Abbildung 15: Visualisierung des Datensatzes *Ailerons* nach dem Anwenden von `t-SNE` aufgeteilt in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün): Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

Wir sehen hier, dass die Datenpunkte sehr nah aneinander liegen. Das könnte die nur geringen Schwankungen in unseren Ergebnissen erklären. Da es jedoch bei den Verfahren `KLIEP` und `KMM` zu Verbesserungen kommt bei den Experimenten ohne Sampling bzw. nur mit Variance Sampling, möchten wir uns zudem den reduzierten Datensatz nach dem Sampling anschauen:

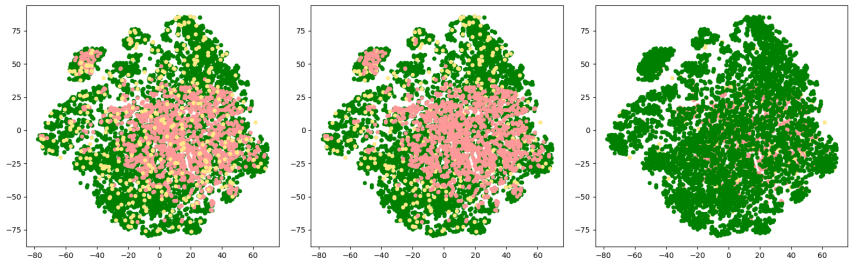

Abbildung 16: Visualisierung des gesampelten Datensatzes *Ailerons* nach dem Anwenden von `t-SNE` aufgeteilt in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

Wir erkennen hier nur, dass sich die Anzahl der Quelldaten deutlich verringert hat, wobei eher die äußeren Quelldaten entfernt worden sind durch das Importance Sampling. Für `KLIEP` und `KMM` sind hier gegebenfalls zu viele Quelldaten entfernt worden. Da hierbei ähnliche Ergebnisse mit und ohne Sampling für die anderen Verfahren herausgekommen sind, stellt sich die Frage, ob nicht auch das Weglassen aller Quelldaten zu ähnlichen Ergebnisse führen würde. Insbesondere da die Zieldaten nach dem Bild die Testdaten alleine abdecken und die anderen Verfahren weniger Quelldaten benötigen.

Elevators Wir haben ähnliche Ergebnisse wie bei *Ailerons* für den Datensatz *Elevators* erhalten. Wir werden deswegen dies nicht weiter ausführen. Die relevanten Visualisierungen sind in Kapitel C zu finden.

Abalone

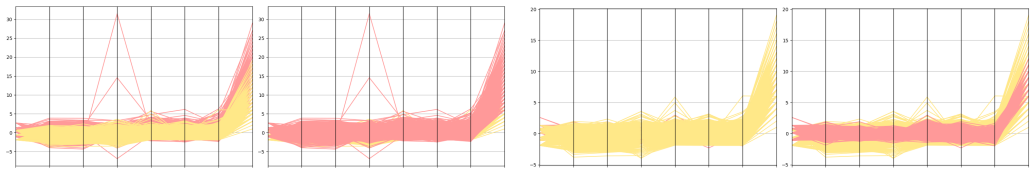

Abbildung 17: Visualisierung des Datensatzes *Abalone* aufgeteilt in Zieldaten (gelb) und Quelldaten (rot) vor (links) und nach dem Sampeln (rechts):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten und dritten Plot sind die Zieldaten und im zweiten und vierten die Quelldaten im Vordergrund.

Ohne vorheriges Sampling sehen wir, dass die Quelldaten mehr Werte abdecken und ein paar Ausreißer haben. Nach dem Sampling wurden die Ausreißer und die weiter vom Durchschnitt der Zieldaten entfernte Quelldaten entfernt, was die Verbesserung nach dem Sampling bei den meisten Verfahren erklärt. Betrachten wir zusätzlich die reduzierten Daten. Wir wenden dafür `PCA` an, da die ersten beiden Hauptkomponenten ca. 80% der Varianz erklären.

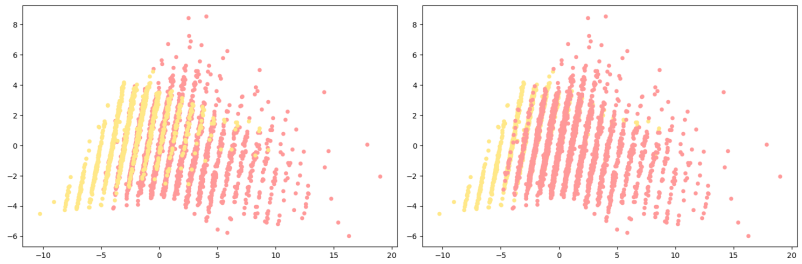

Abbildung 18: Visualisierung des Datensatzes *Abalone* nach dem Anwenden von `PCA` aufgeteilt in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

Wir erkennen, dass es zwischen den Quell- und Zieldaten Überschneidungen gibt, jedoch ein Anteil an Quelldaten weiter entfernt von den Zieldaten verteilt ist. Führen wir nun zusätzlich Sampling durch:

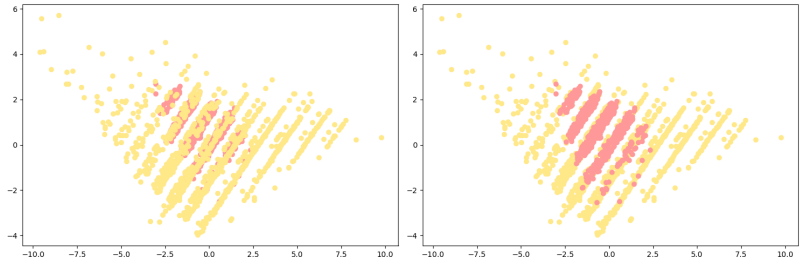

Abbildung 19: Visualisierung des Datensatzes *Abalone* nach dem Anwenden von `PCA` in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

Wir sehen, dass wir deutlich weniger Quelldaten erhalten und diese sich in der Nähe der Mitte der Zieldaten befinden. Zusätzlich lässt sich erkennen, dass die Zieldaten teilweise etwas voneinander entfernt liegen. Werden nun als Testdaten weiter entfernt liegende Datenpunkte ausgewählt, erklärt dies die vergleichsweise zu den anderen Verfahren höheren $RMSE$-Werte. Gegebenfalls wäre es hier sinnvoll, weniger Quelldaten zu entfernen, da sie vorher einen größeren Bereich an Zieldaten abgedeckt haben.

Kinematics Auch bei diesem Datensatz war die Varianz zu stark auf alle Achsen verteilt, weswegen wir auch hier `t-SNE` genutzt haben. Wir erhalten hier ähnliche Ergebnisse wie bei *Ailerons*, deswegen werden wir nicht weiter darauf eingehen. Die Visualisierungen finden sich auch in Kapitel C.

Computer Activity Bei diesem Datensatz hatten wir die größten Differenzen bei den $RMSE$-Werten, was sich auch in der Verteilung der Daten wiederspiegelt:

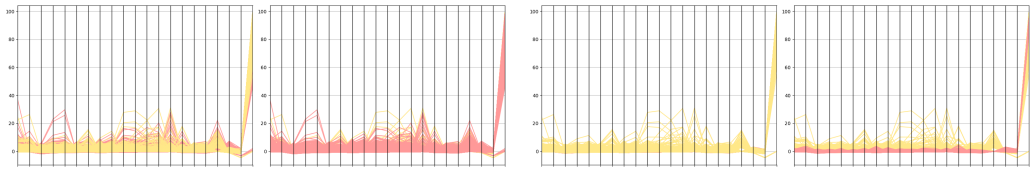

Abbildung 20: Visualisierung des Datensatzes *Computer Activity* aufgeteilt in Zieldaten (gelb) und Quelldaten (rot) vor (links) und nach dem Sampeln (rechts)
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten und dritten Plot sind die Zieldaten und im zweiten und vierten die Quelldaten im Vordergrund.

Wir sehen eine hohe Anzahl an Datenpunkten, die sich bei einzelnen Attributen wie Ausreißer verhalten. Es muss jedoch gesagt werden, dass dies theoretisch auch dieselben Datenpunkte sein können. Sampling führt bei Quelldaten dazu, dass diese Datenpunkte entfernt werden. Wir reduzieren nun den Datensatz mit `PCA`, da sich ca. 95% mit den ersten beiden Hauptkomponenten beschreiben lassen:

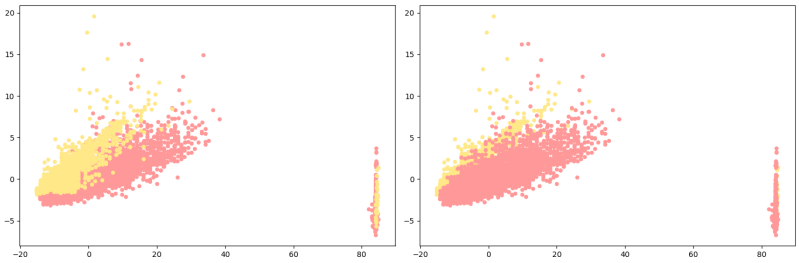

Abbildung 21: Visualisierung des Datensatzes *Computer Activity* nach dem Anwenden von `PCA` aufgeteilt in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

Wir können erkennen, dass es Überschneidungen von Ziel- und Quelldaten gibt. Außerdem sehen wir zwei verschiedene Häufungen an Datenpunkten. Der kleinere rechts könnte erklären, weshalb wir hier die größten Unterschiede in den $RMSE$-Werten haben. Legt man zu viel Gewicht auf diese weiter entfernten Daten, so könnte man eine Verzerrung des Datensatzes erzeugen. Für solche Fälle sollten Ansätze mit Boosting im Allgemeinen besser funktionieren, da sie auch innerhalb der Zieldaten Gewichte verteilen und Datenpunkte, die sich teilweise wie Ausreißer verhalten, weniger in Betracht ziehen, was unsere Ergebnisse erklären kann. Führen wir nun noch Sampling durch:

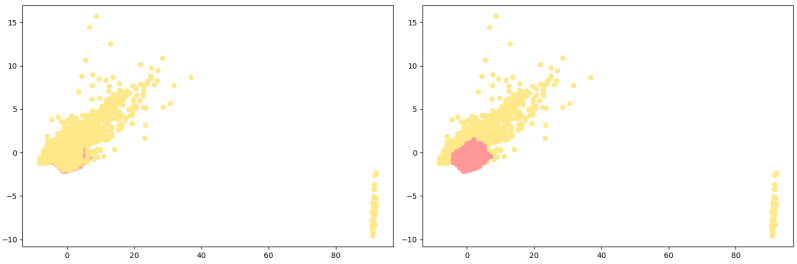

Abbildung 22: Visualisierung des Datensatzes *Computer Activity* nach dem Anwenden von `PCA` aufgeteilt in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

Hierbei erkennen wir weiterhin, dass der Quelldatensatz erfolgreich reduziert wurde und nah am Durchschnitt der Zieldaten liegt. Insbesondere wird die weiter entfernte Häufung nicht mehr von den Quelldaten abgedeckt.

Insgesamt können die Visualisierungen der Datensätze Teile unsere Ergebnisse erklären. Jedoch ist damit nicht klar geworden, wieso `IWKRR` insbesondere bei dem *Kinematics*-Datensatz deutlich bessere Ergebnisse zeigt.

6 Fazit

Das Ziel dieser Arbeit bestand darin, die Ergebnisse von [Gup+22] zu untersuchen.

Wir haben uns dafür zunächst in die Grundlagen des Transferlernens eingearbeitet und die Herausforderung der Gewichte analysiert. Daraufhin haben wir verschiedene Verfahren betrachtet, die Gewichte für die Quelldaten berechnen und dann eine Hypothese erstellen. Darunter waren Ansätze mit Kernel und Boosting, auf dem `S-TrAdaBoost.R2` aufbaut. Bevor wir uns jedoch dies näher angeschaut haben, haben wir den Ansatz von Boosting erklärt und `TTR2` kennengelernt. Auf den ersten Blick wirkt `S-TrAdaBoost.R2` wie ein vereinfachter und verbesserter Algorithmus. Jedoch sind wir bei den numerischen Experimenten, bei denen wir die Verfahren getestet und verglichen haben, auf Differenzen gestoßen zwischen dem beschriebenen Algorithmus im Artikel und dem bereitgestellten Code, die diese Annahme zunächst wackeln ließen. Wir konnten trotzdem bestätigen, dass `S-TrAdaBoost.R2` in fast allen Fällen bessere Ergebnisse liefert als `TTR2` und je nach Datensatz nur `IWKRR` bessere Ergebnisse liefert. Dabei haben wir ähnliche Fehlerwerte wie [Gup+22] erhalten.

Außerdem haben wir gesehen, dass die Wahl des Samplings Einfluss auf die Ergebnisse hat. Dabei ist jedoch nicht klar geworden, welche die beste Art des Samplings ist. Dies variiert je nach Verfahren und nach Datensatz. Wir können dementsprechend keine klare Aussage darüber treffen. Da wir beim Sampling die Anzahl an neuen Quell- und Zieldaten von [Gup+22] übernommen haben, wäre es hier spannend, unterschiedliche Größen auszuprobieren. Überdies sollten auch andere Samplingmethoden ausprobiert werden, wie z.B. aus [Elv+23]. Auch könnte man weitere Experimente mit verschiedenen Metriken durchführen und vergleichen.

Wir müssen an dieser Stelle jedoch anmerken, dass erste Experimente uns vermuten lassen, dass die unterschiedliche Wahl der Gewichtungsfaktoren aus Kapitel 4 und das Normalisieren der Gewichte bei `S-TrAdaBoost.R2` keine Rolle spielen. Im Allgemeinen ist das Anpassen der Gewichtungsfaktoren dennoch eine Möglichkeit zur Weiterentwicklung des Algorithmus.

Des Weiteren haben wir bei `IWKRR` gesehen, dass durch Verschieben der Gewichte bessere Ergebnisse erzielt werden können. Hier wäre es spannend weitere Experimente durchzuführen und diese mit `S-TrAdaBoost.R2` zu vergleichen.

Wir sind außerdem in dieser Bachlorarbeit nicht auf Laufzeiten eingegangen. Für die Praxis wäre hier eine vergleichende Untersuchung sinnvoll. Bei unseren Experimenten war `S-TrAdaboost.R2` mit Abstand das langsamste Verfahren.

Zuletzt haben wir Datensätze gesehen, in denen es wenig Veränderung in den Ergebnissen gab. Durch die Visualisierung der Daten konnten wir oftmals sehen, dass hierbei die Ziel- und Quelldaten große Überschneidungen haben. Das führt zu der Frage, ob diese Aufteilung für Transferlernen geeignet ist und ob die Quelldaten überhaupt noch weitere Informationen liefern. Spannend wäre es hier zu sehen, ob `AdaBoost.R2` ähnliche Ergebnisse nur auf den Zieldaten liefert, was ein Hinweis wäre, dass eine andere Aufteilung oder ein anderer Datensatz verwendet werden sollte.

Zusammenfassend können wir zwar den Erfolg von `S-TrAdaBoost.R2` größtenteils bestätigen, jedoch bleiben weitere Fragen zu diesem Feld, insbesondere Sampling, offen. Vermutlich hat Sampling das Potenzial zu weiteren Verbesserungen in der Prognose zu führen.

A Visualisierung der Verteilung der Ergebnisse

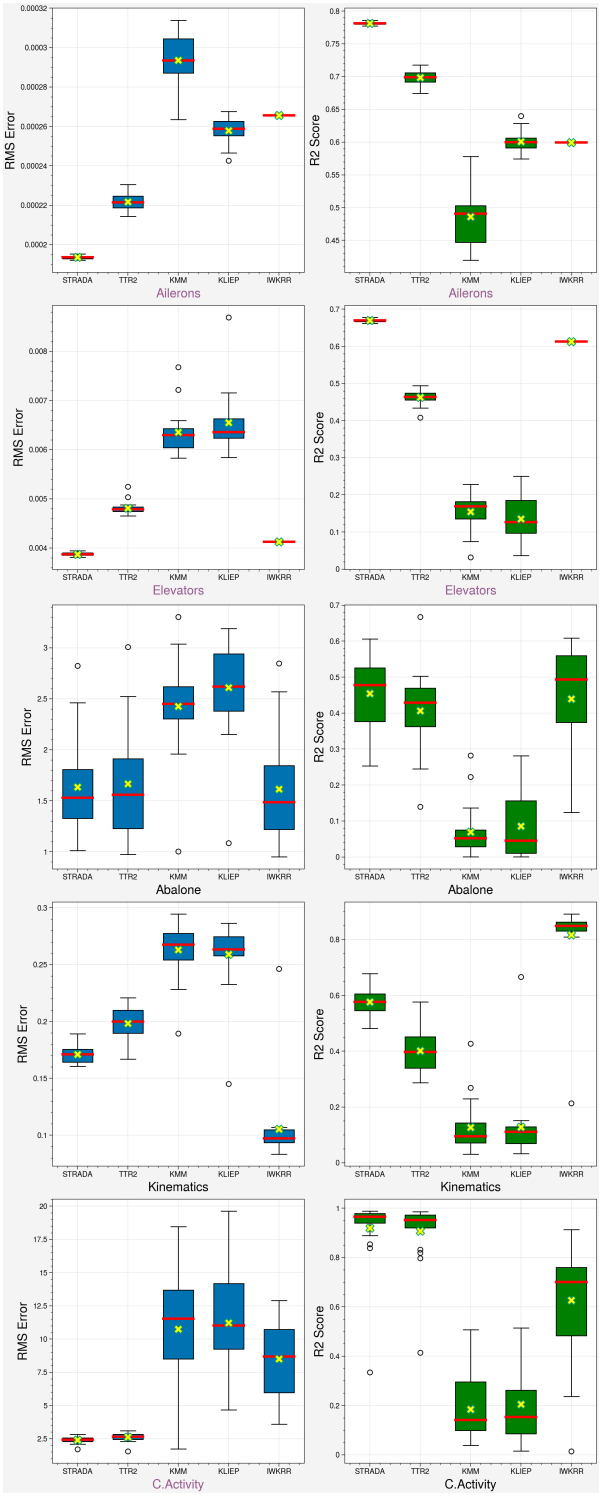

Abbildung 23: Visualisierung der Ergebnisse des $RMSE$ und r^2 von 20 Durchgängen der Verfahren `S-TrAdaBoost` (`STRADA`), `TTR2`, `KMM`, `KLIEP` und `IWKRR` mit vorherigem Sampling (vgl. Kapitel 4). Der Mittelwert (Marker: gelbes "X") und der Median (Marker: rote Linie) sind hervorgehoben. Die Datensätze, für die S-TrAdaBoost.R2 besonders gut abschneidet, sind ebenfalls markiert (Marker: lila).

B Ergebnisse mit verschiedenen Arten von Sampling

Ergebnisse wie im Artikel mit umgekehrter Reihenfolge

	ailerons		elevators		abalone		kinematics		c.activity	
Fehler	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2
S-TR	**0.00019**	**0.78**	**0.0039**	**0.67**	1.65	**0.46**	0.18	0.54	**2.51**	**0.94**
TTR2	0.00022	0.70	0.0047	0.49	1.72	0.39	0.20	0.40	2.78	0.89
IWKRR	0.00026	0.60	0.0044	0.57	**1.61**	0.44	**0.11**	**0.80**	8.58	0.63
KMM	0.00029	0.49	0.0067	0.11	2.43	0.07	0.27	0.10	10.90	0.20
KLIEP	0.00025	0.61	0.0065	0.15	2.62	0.12	0.27	0.10	10.78	0.21

Tabelle 8: Ergebnisse mit Sampling beschrieben wie in [Gup+22] mit umgekehrter Reihenfolge: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

Variance Sampling

	ailerons		elevators		abalone		kinematics		c.activity	
Fehler	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2
S-TR	**0.00018**	**0.82**	**0.0039**	**0.68**	1.77	**0.47**	0.21	0.41	**3.76**	**0.93**
TTR2	0.00021	0.72	0.0046	0.49	1.84	0.40	0.22	0.32	5.42	0.85
IWKRR	0.00026	0.60	0.0041	0.62	**1.73**	0.44	**0.11**	**0.80**	8.58	0.63
KMM	0.00022	0.71	0.0065	0.14	2.59	0.05	0.26	0.17	8.09	0.82
KLIEP	0.00021	0.73	0.0065	0.14	2.76	0.14	0.25	0.20	7.49	0.84

Tabelle 9: Ergebnisse mit vorherigem Variance Sampling: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

Importance Sampling

	ailerons		elevators		abalone		kinematics		c.activity	
Fehler	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2	*RMS*	r^2
S-TR	**0.00019**	**0.78**	**0.0039**	**0.67**	1.67	**0.47**	0.21	0.41	**2.45**	**0.95**
TTR2	0.00022	0.70	0.0048	0.46	1.79	0.38	0.22	0.32	2.68	0.93
IWKRR	0.00027	0.60	0.0041	0.62	**1.56**	0.45	**0.11**	**0.80**	8.78	0.66
KMM	0.00030	0.46	0.0067	0.11	2.46	0.08	0.25	0.19	11.34	0.18
KLIEP	0.00026	0.61	0.0064	0.17	2.71	0.12	0.25	0.19	11.40	0.18

Tabelle 10: Ergebnisse mit vorherigem Importance Sampling: Die Werte sind jeweils der Durchschnitt von 20 Durchgängen.

C Weitere Plots zur Analyse der Datensätze

Plots zum Datensatzes *Elevators*

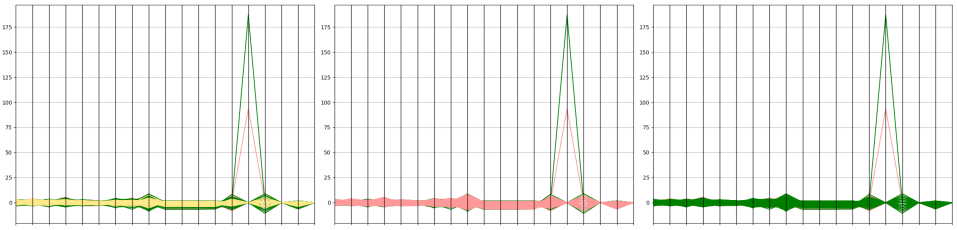

Abbildung 24: Plotten des Datensatzes *Elevators* in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

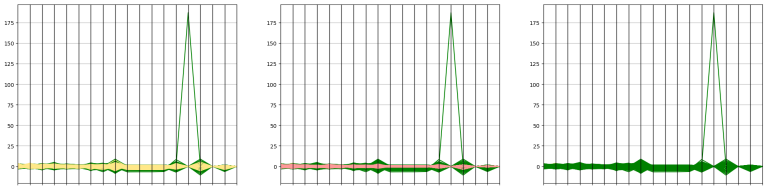

Abbildung 25: Plotten des gesampelten Datensatzes *Elevators* in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

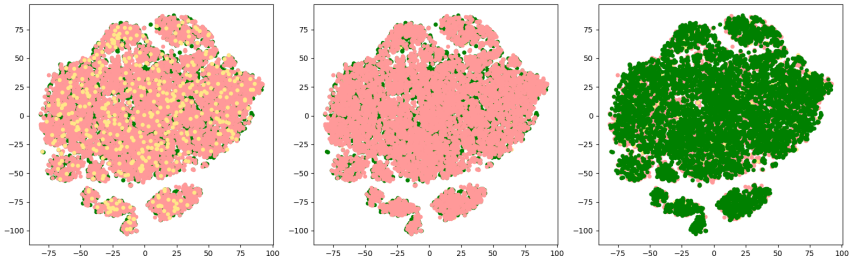

Abbildung 26: Plotten des Datensatzes *Elevators* nach dem Anwenden von `t-SNE` in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

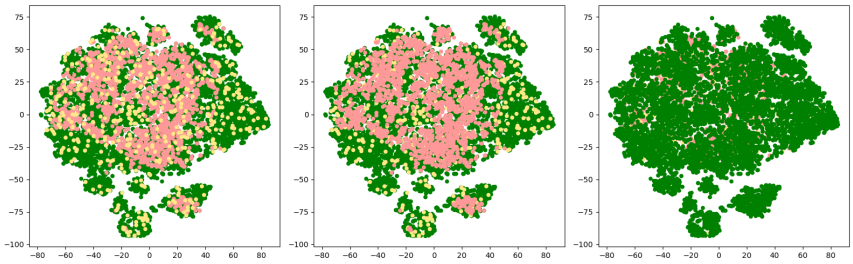

Abbildung 27: Plotten des gesampelten Datensatzes *Elevators* nach dem Anwenden von `t-SNE` in Zieldaten (gelb), Quelldaten (rot) und Testdaten (grün):
Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten, im zweiten die Quelldaten und im dritten die Testdaten im Vordergrund.

Plots zum Datensatz *Kinematics*

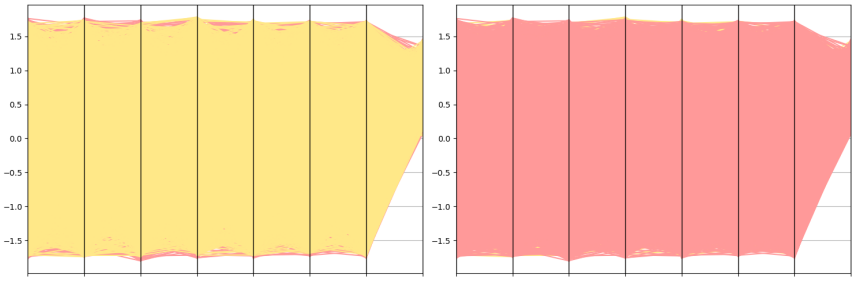

Abbildung 28: Plotten des Datensatzes *Kinematics* aufgeteilt in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

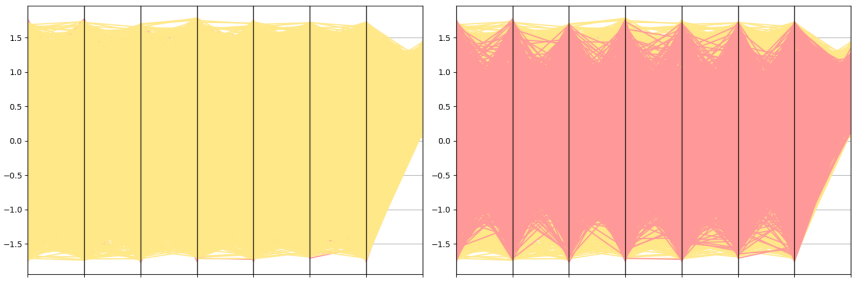

Abbildung 29: Plotten des gesampelten Datensatzes *Kinematics* aufgeteilt in Zieldaten (gelb) und Quelldaten (rot):
Die x-Achse entspricht den Attributen des Datensatzes und die y-Achse dem Wertebereich, den die Werte der Datenpunkte in den Attributen annehmen.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

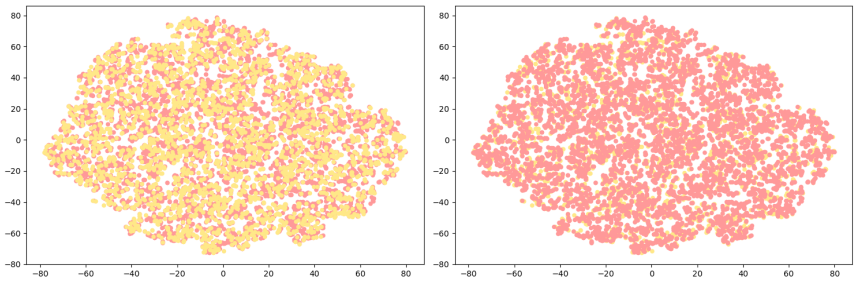

Abbildung 30: Plotten des Datensatzes *Kinematics* nach dem Anwenden von `t-SNE` aufgeteilt in Zieldaten (gelb) und Quelldaten (rot): Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

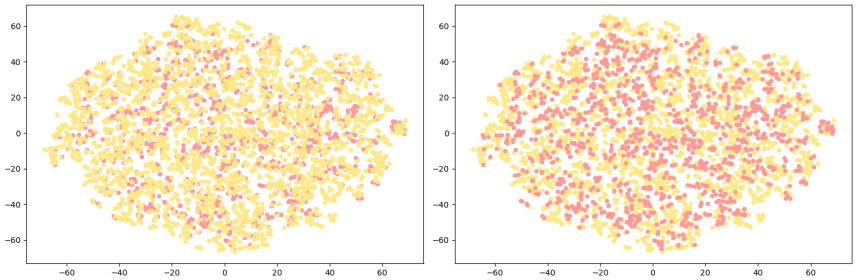

Abbildung 31: Plotten des gesampelten Datensatzes *Kinematics* nach dem Anwenden von `t-SNE` aufgeteilt in Zieldaten (gelb) und Quelldaten (rot): Die x-Achse und die y-Achse entsprechen den Koordinaten der reduzierten Datenpunkte.
Im ersten Plot sind die Zieldaten und im zweiten die Quelldaten im Vordergrund.

Literatur

[Asi+23] Sohaib Asif, Yi Wenhui, Kamran Amjad, Hou Jin, Yi Tao und Si Jinhai. „Detection of COVID-19 from chest X-ray images: Boosting the performance with convolutional neural network and transfer learning“. eng. In: *Expert systems* 40.1 (2023).

[AV07] David Arthur und Sergei Vassilvitskii. „k-means: the advantages of careful seeding“. eng. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms*. Society for Industrial und Applied Mathematics, 2007, S. 1027–1035.

[Ber+23] Julius Berner, Philipp Grohs, Gitta Kutyniok und Philipp Petersen. „The Modern Mathematics of Deep Learning“. eng. In: *arXiv.org* (2023).

[Dai+07] Wenyuan Dai, Qiang Yang, Gui-Rong Xue und Yong Yu. „Boosting for transfer learning“. eng. In: *ACM International Conference Proceeding Series; Vol. 227: Proceedings of the 24th international conference on Machine learning; 20-24 June 2007*. ACM, 2007, S. 193–200.

[Dru97] Harris Drucker. „Improving regressors using boosting techniques“. In: *Proceedings of the 14th International Conference on Machine Learning*. Association for Computing Machinery, 1997, S. 107–115.

[Elv+23] Víctor Elvira, Emilie Chouzenoux, Ömer Deniz Akyildiz und Luca Martino. *Gradient-based Adaptive Importance Samplers*. 2023.

[FS97] Yoav Freund und Robert E. Schapire. „A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting“. eng. In: *Journal of computer and system sciences* 55.1 (1997), S. 119–139.

[Gup+22] Shrey Gupta, Jianzhao Bi, Yang Liu und Avani Wildani. „Boosting for regression transfer via importance sampling“. In: *International Journal of Data Science and Analytics* (2022).

[GV14] Jochen Garcke und Thomas Vanck. „Importance Weighted Inductive Transfer Learning for Regression“. eng. In: *Machine Learning and Knowledge Discovery in Databases*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 466–481.

[HD12] Gernot Horstmann und Gesine Dreisbach. *Allgemeine Psychologie 2 kompakt: Lernen, Emotion, Motivation, Gedächtnis. Mit Online-Materialien*. ger. Beltz Verlagsgruppe, 2012.

[Höw23] Elisabeth Höwler. „Oft zu spät erkannt: Eierstockkrebs“. ger. In: *Die Heilberufe* 75.11 (2023), S. 10–13.

[Hun07] John D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95.

[Kes20] Thomas Kesselheim. *Skript: Algorithmische Grundlagen des maschinellen Lernens*. unveröffentlicht, 2020.

[KF19] Angelos Katharopoulos und François Fleuret. *Not All Samples Are Created Equal: Deep Learning with Importance Sampling*. 2019.

[Mat+21] Antoine de Mathelin, François Deheeger, Guillaume Richard, Mathilde Mougeot und Nicolas Vayatis. „ADAPT: Awesome Domain Adaptation Python Toolbox“. In: *arXiv preprint arXiv:2107.03049* (2021).

[McK10] Wes McKinney. „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman. 2010, S. 56–61.

[Men22] Zhaorui Meng. „Bagging Based Multi-Source Learning and Transfer Regression for Electricity Load Forecasting“. eng. In: *IAENG international journal of computer science* 49.2 (2022), S. 335–.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh und Ameet Talwalkar. *Foundations of machine learning*. eng. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts ; London, England: The MIT Press, 2018.

[Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot und Édouard Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[PS10] David Pardoe und Peter Stone. „Improving regressors using boosting technique“. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Association for Computing Machinery, 2010, S. 1863–870.

[PY10] Sinno Jialin Pan und Qiang Yang. „A Survey on Transfer Learning“. eng. In: *IEEE transactions on knowledge and data engineering* 22.10 (2010), S. 1345–1359.

[Rui19] Ruibzhan. *Demo for traditional transfer learning techniques*. 2019. URL: `https://github.com/Ribosome25/Traditional_Transfer_Learning/blob/master/README.md` (besucht am 10. 12. 2023).

[Sal+19] Syed Moshfeq Salaken, Abbas Khosravi, Thanh Nguyen und Saeid Nahavandi. „Seeded transfer learning for regression problems with deep learning“. In: *Expert Systems with Applications* 115 (2019), S. 565–577.

[SB14] Shai Shalev-Shwartz und Shai Ben-David. *Understanding machine learning from theory to algorithms*. eng. New York: Cambrige Univ. Press, 2014.

[Sei+08] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse und Amri Napolitano. „Resampling or Reweighting: A Comparison of Boosting Implementations“. In: *2008 20th IEEE International Conference on Tools with Artificial Intelligence*. Bd. 1. 2008, S. 445–451.

[SPH07] Bernhard Schölkopf, John Platt und Thomas Hofmann. „Correcting Sample Selection Bias by Unlabeled Data“. eng. In: *Advances in Neural Information Processing Systems 19*. United States: MIT Press, 2007.

[Sug+08] Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul von Bünau und Motoaki Kawanabe. „Direct importance estimation for covariate shift adaptation“. eng. In: *Annals of the Institute of Statistical Mathematics* 60.4 (2008), S. 699–746.

[Tor] Luis Torgo. *Description and Download of UCI Regression datasets*. URL: `https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html` (besucht am 06. 11. 2023).

[Van16] Thomas Vanck. „New Importance Sampling Based Algorithms for Compensating Dataset Shifts“. Doktorarbeit. Institut für Mathematik, TU Berlin, 2016.

[Vir+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt und SciPy 1.0 Contributors. „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python“. In: *Nature Methods* 17 (2020), S. 261–272.

[Wan+18] Zirui Wang, Zihang Dai, Barnabás Póczos und Jaime G. Carbonell. „Characterizing and Avoiding Negative Transfer“. In: *CoRR* abs/1811.09751 (2018).

[Web23] Nina Weber. „ChatGPT – Ein Jahr, das die Welt veränderte“. In: *Futuriq.de* (2023).

[ZZ15] Peilin Zhao und Tong Zhang. *Stochastic Optimization with Importance Sampling*. 2015.