technische universität
dortmund

Bachelorarbeit

Interaktive Visualisierung verschiedener Erklärbarkeitsverfahren für Neuronale Netze

Lisa Salewsky
10. Mai 2021

Gutachter:
Prof. Dr. Katharina Morik
Matthias Jakobs

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS VIII)
https://www-ai.cs.uni-dortmund.de/

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Die Zahl der Bereiche des täglichen Lebens, in denen Entscheidungen von Maschinen getroffen werden, wird stetig größer. Mit zunehmendem Einsatz von maschinellen Lernverfahren wächst zugleich auch der Bedarf an guten Erklärungen. Gleichzeitig erhöht sich die Komplexität der Probleme sowie die Anzahl der verwendeten Informationen. Hinzu kommt, dass die zur Entscheidungsfindung notwendigen Algorithmen (zum Teil durch die vorherigen Probleme bedingt) meist auf eine komplexe Weise und mit einer für Menschen unbeherrschbaren Menge von Daten und Eigenschaften arbeiten [44]. Je stärker einzelne Personen von solchen Entscheidungen betroffen sind, umso wichtiger ist es, das Vertrauen in ein maschinell bestimmtes Ergebnis zu festigen. Damit geht zugleich die Verbesserung des Verständnisses maschineller Entscheidungen einher [40].

Mittlerweile gibt es zahlreiche verschiedene Ansätze, die unterschiedlichste Lösungsvorschläge für besseres Verständnis des maschinellen Entscheidungsprozesses bieten. Sie lassen sich in fünf grobe Kategorien unterteilen, für die jeweils eine große Menge konkreter Algorithmen existieren. Dabei wird die zu erklärende Entscheidung und das zugehörige maschinelle Lernverfahren oft als Black Box bezeichnet, da es im Allgemeinen nicht möglich ist, die komplexe zugrundeliegende Funktion vollständig nachzuvollziehen. In dieser Arbeit werden mit der Bezeichnung „Ansatz“ die entsprechenden Grundideen zum Öffnen dieser Black Box bezeichnet. Algorithmus, (Erklärbarkeits-) Modell oder -Methode bezeichnen hingegen die konkrete Implementierung, die einem Ansatz zugeordnet werden kann.

Es ist möglich, **vollständig auf die Implementierung von nicht-interpretierbaren Algorithmen zu verzichten**. Stattdessen können Methoden verwendet werden, die lediglich eine begrenzte Anzahl von Eigenschaften betrachten und nur einfache Zusammenhänge beschreiben [44]. Auf diese Weise bleiben die Entscheidungen für Menschen greifbar und verständlich. Eine solche starke Einschränkung stößt jedoch im alltäglichen Leben oftmals

schnell an ihre Grenzen. Diese zeigen sich in verminderter Genauigkeit, weil zu wenig Eigenschaften ausgewählt oder komplexere Zusammenhänge nicht mehr darstellbar sind [19].

Werden jedoch komplexe Lernverfahren wie zum Beispiel Neuronale Netze verwendet, müssen diese erklärt werden [7], um Verständnis für das Ergebnis zu erzeugen. Durch Erklärbarkeitsmethoden wird die Black Box der komplexen Funktionen mit verschiedenen Methoden geöffnet [19].

Bei dem Ansatz der **Modell-Erklärung** bleibt die eigentliche Funktion im Allgemeinen eine Black Box. Nach dem erfolgreichen Trainieren des komplexen Modells wird zusätzlich ein interpretierbarer Ansatz, der als Surrogatmodell bezeichnet wird, so trainiert, dass er die gleichen Ergebnisse wie die Black Box Funktion liefert. Dieses weniger komplexe Modell kann so die Entscheidung für Menschen zugänglicher und verständlicher machen [19].

Wie hilfreich oder interpretierbar solche Erklärungen sind lässt sich jedoch nicht immer trivial bestimmen [44]. Oft sind zudem weitere Faktoren von Interesse, beispielsweise die Laufzeit oder der Speicherbedarf, wenn nur begrenzte Zeit oder Ressourcen zur Verfügung stehen. Des Weiteren ist das Verstehen von Erklärungen in vielen Fällen subjektiv und kann sich von Person zu Person unterscheiden, auch wenn die zugrundeliegende Erklärung unverändert bleibt [27].

Ein anderer Ansatz untersucht die einzelnen Teile eines Black Box Modells und macht die jeweiligen Auswirkungen auf eine Entscheidung erkennbar. Bei diesem Ansatz wird zwischen Methoden, die nur den In- und Output betrachten und solchen, die die gesamte Funktion einbeziehen, unterschieden. Erstere erklären die Ausgabe eines komplexen Modells indem sie sowohl eine Vorhersage als auch eine dazugehörige Erklärung liefern und stellen somit eine **Black Box Outcome Explanation** bereit [19].

Algorithmen, die nach dem Prinzip der Erklärung interner Vorgänge der Black Box aufgebaut sind, liefern Repräsentationen der Funktionalität des komplexeren Modells. Sie bilden eine Lösung für das **Black Box Inspection Problem** [19]. Somit werden Begründungen für getroffene Entscheidungen anhand der Veranschaulichung der Black Box Funktion selbst gegeben. Bei diesen kann es sich zum Beispiel um Balkendiagramme oder Heat-Maps handeln [26].

Eine weitere Methode bietet der Ansatz des **Transparent Box Design Problem**s. Dieser unterscheidet sich stärker von den anderen, da hier eine transparente Box erstellt wird, also eine Funktion, die direkt eine Erklärung für ihre Entscheidungen mit liefert. Auf diese Weise ist keine zusätzliche Erklärung notwendig [19]. Ein solcher Ansatz wird in dieser Arbeit jedoch nicht betrachtet, da er sich stark von den anderen unterscheidet und ein vollständig anderes Vorgehen erfordert.

Insgesamt zeigt sich, dass es zwar viele verschiedene Erklärbarkeitsmethoden gibt, diese aber nur schwierig zu vergleichen sind. Daher scheint es praktisch, diesen direkten Vergleich zwischen verschiedenen Erklärungsarten für bestimmte Daten zu visualisieren.

1.2 Ziel der Arbeit

Wie bereits erwähnt, werden maschinelle Entscheidungen immer mehr genutzt und haben im Laufe der Zeit einen immer größeren Einfluss auf das alltägliche Leben gewonnen. Dies führt zugleich zu einem wachsenden Bedürfnis an Erklärungen dieser Entscheidungen, die das Verständnis und damit die Akzeptanz erhöhen. Wenn Maschinen mit komplexen und uneinsichtigen Berechnungen darüber entscheiden, wer einen Kredit bekommt oder sogar vorhersagen, wer möglicherweise nach Entlassung aus dem Gefängnis eine erneute Straftat begeht, dann sind dies Entscheidungen, die für betroffene Personen große Lebensabschnitte beeinflussen. Je stärker dieser Einfluss ist, desto höher ist das Bedürfnis nach Erklärungen. Dies ist teilweise sogar gesetzlich vorgeschrieben [7, 44].

Aufgrund des stetig wachsenden Bedarfs an Erklärbarkeitsverfahren und der großen Vielfalt an verfügbaren Algorithmen liegt eine Kombination verschiedener Erklärungen in einer einzigen Oberfläche nahe. Eine solche Verbindung kann einem Nutzer so die Möglichkeit geben, einen Überblick über verschiedene Ansätze und konkrete Erklärungen zu erhalten und selbst die Entscheidung zu treffen, welche Erklärungen er als besonders hilfreich empfindet. Dabei wird von einem vorhandenen, bereits trainierten Neuronalen Netz ausgegangen, das die zu erklärende Black Box Funktion darstellt.

Das Ziel dieser Arbeit ist es, einem Nutzer eine Anwendung [1] zur Verfügung zu stellen, die diesen direkten Vergleich verschiedener Erklärbarkeitsansätze ermöglicht. Der Anwender kann mit der Oberfläche interagieren. Es ist ihm möglich zu entscheiden, welche Erklärungen er für seinen individuellen Anwendungsfall als hilfreich empfindet. Dabei kann der Schwerpunkt auf der Erhöhung des Verständnisses und somit dem Vertrauen in eine Entscheidung liegen, er könnte jedoch ebenso auf eine besonders kurze Laufzeit oder einen geringen Speicherverbrauch gesetzt werden.

Neben einer Visualisierung des Erklärbarkeitsansatzes werden somit auch Metadaten angezeigt. Dem Nutzer wird dabei ein Überblick über für ihn wichtige Informationen gegeben. Er hat zudem die Möglichkeit, bei einigen der Methoden zusätzliche Parameter einzustellen, mit denen die interne Ergebnisberechnung beeinflusst wird. Die Auswahl verschiedener Datenpunkte (siehe Definition 2.1.3) wird ebenfalls unterstützt.

Die verwendeten Erklärbarkeitsmethoden gehören verschiedenen Ansätzen an und bieten so Vergleiche zwischen diesen an. Gleichzeitig soll es dem Anwender möglich sein, durch unterschiedliche Erklärbarkeitsmethoden eines Ansatzes die Vergleichbarkeit innerhalb einer konkreten Grundidee zu erhöhen. Hierfür wurden sechs ausgewählte Algorithmen, je zwei pro Ansatz, realisiert.

Um eine Visualisierung zwecks eines Vergleichs verschiedener Erklärungsarten erstellen zu können, wird zunächst ein Klassifizierer benötigt, dessen Entscheidungen erklärt werden

[1] `https://github.com/LisaIrinaHanako/BA-VisualisierungErklaerbarkeitsverfahrenNN`

sollen. Hierbei handelt es sich um ein Neuronales Netz, das trainiert und für diese Arbeit bereitgestellt wird.

Neben dem Netz werden zudem Implementierungen verschiedener Erklärbarkeitsansätze und die Erstellung einer Visualisierung in Form einer interaktiv verwendbaren Webseite benötigt. Die Methoden werden anhand der Verfügbarkeit passender Bibliotheken ausgewählt. Ein Algorithmus muss dabei folgende Kriterien erfüllen, um als passend eingestuft zu werden:

- Er muss (hauptsächlich) in Python implementiert sein
- Der zu verwendende Programmcode muss frei verfügbar sein
- Der Code muss Erklärungen für neuronale Netze, sowie bestenfalls eine Umwandlung in eine interpretierbare Repräsentation liefern

In dieser Arbeit wird mithilfe von plotly und plotlys Dash (siehe Kapitel 4) eine Oberfläche implementiert, welche die sechs Algorithmen visualisiert. Des Weiteren wird für alle Erklärbarkeitsmethoden mindestens eine Interaktionskomponente bereitgestellt. Der konkrete Datenpunkt, der erklärt werden soll, ist auswählbar. Sofern die gewählten Bibliotheken es bereitstellen erhält der Nutzer außerdem weitere Optionen, die Erklärbarkeitsverfahren zu konfigurieren.

1.3 Vorgehen

In dieser Arbeit wird der German-Credit Kreditdatensatz [2], sowie ein darauf vor-trainiertes Neuronales Netz verwendet. Für diese Vorgaben werden anschließend bestehende Bibliotheken von fünf Algorithmen, sowie eine eigene Implementierung eines weiteren Ansatzes in einer Anwendung verbunden. Mit diesen werden Erklärungen für die Entscheidungen des gegebenen Neuronalen Netzes berechnet, die anschließend in passenden Visualisierungen aufbereitet und in einer Web-Oberfläche dargestellt werden. Das Verbinden der Bibliotheken mit dem Neuronalen Netz, die Erstellung der Visualisierungen und das Zusammenstellen in einer Oberfläche sind dabei Teile, die für diese Arbeit selbst implementiert werden. Zudem werden interaktive Komponenten für die einzelnen Erklärbarkeitsmodelle bereitgestellt, mit denen ein Nutzer die Berechnungen beeinflussen oder für andere Datenpunkte anzeigen lassen kann.

1.4 Aufbau der Arbeit

Im zweiten Kapitel werden die allgemeinen relevanten Grundlagen kurz dargestellt. Zu diesen gehört der Aufbau des Neuronalen Netzes, das die Basis für diese Arbeit bildet, sowie

[2] `http://archive.ics.uci.edu/ml`

die Erklärung des generellen Vorgehens beim Trainieren eines solchen Netzes anhand des Beispiels Backpropagation. Abschließend werden die den Erklärbarkeitsmethoden zugrundeliegenden Ansätze: Surrogatmodelle, Counterfactuals (CF) und Feature Contribution genauer beschrieben.

Das dritte Kapitel stellt die benutzten Erklärbarkeitsmodelle dar. Dabei werden jeweils die verwendeten theoretischen Konzepte aufgegriffen und erläutert. Zudem wird die Taxonomie, also Laufzeit und Speicherbedarf, für alle Algorithmen bestimmt. Bei den verwendeten Algorithmen handelt es sich um

- einen Entscheidungsbaum (sklearn-Bibliothek [3] [4] [5])
- logistische Regression (sklearn-Bibliothek [6])
- einen k-NearestNeighbors(kNN)- Klassifikator zur Berechnung der In-Sample-Counterfactuals (sklearn-Bibliothek [7])
- Diverse Counterfactual Explanations (DiCE-Bibliothek [8] [9])
- DeepSHAP (SHAP-Bibliothek [10] [11])
- Layerwise Relevance Propagation (LRP, selbst implementiert[12])

Das Design und die Umsetzung der interaktiven Benutzeroberfläche in Form einer Webseite bilden Kapitel vier. In diesem Abschnitt wird die Implementierung näher erläutert. Dies stellt die praktische Umsetzung der Arbeit dar. Die verwendete Software und benutzten Komponenten werden ebenfalls beschrieben.

Abschließend werden in Kapitel fünf die Ergebnisse der Arbeit zusammengefasst und ein Fazit gezogen. Unter anderem werden hier die aufgetretenen Probleme und deren Behebung beleuchtet. Des Weiteren findet eine Evaluierung der Praktikabilität und des Mehrwertes, sowie ein Ausblick auf mögliche Erweiterungen statt.

---

[3] https://github.com/scikit-learn/scikit-learn
[4] https://scikit-learn.org/stable/modules/classes.html
[5] https://scikit-learn.org/stable/modules/tree.html#tree
[6] https://scikit-learn.org/stable/modules/linear_model.html#linear-model
[7] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[8] https://github.com/interpretml/DiCE
[9] https://interpret.ml/DiCE/
[10] https://github.com/slundberg/shap
[11] https://shap.readthedocs.io/en/latest/index.html
[12] http://heatmapping.org/tutorial/

Kapitel 2

Grundlagen

2.1 Grundbegriffe des maschinellen Lernens

Bei maschinellem Lernen handelt es sich um einen Teilbereich der Informatik. Eine genaue, allgemeine Definition gibt es nicht. Der Begriff wird oft über die Darstellung bestimmter Szenarien erklärt. Dabei überschneidet sich ein entscheidender Teil: Es werden Computerprogramme entwickelt, die in der Lage sind, sich menschenähnlich zu verhalten. Dieses schließt zum Beispiel die Fähigkeit zu lernen oder zur Selbstverbesserung ein [22] und ermöglicht es intelligenten Systemen, Entscheidungen zu treffen. In vielen Situationen sind diese Klassifizierungen für Menschen schwierig zu entscheiden, zu erkennen oder sehr komplex.

Es ist nicht nur der Oberbegriff, der nicht eindeutig definiert ist. Auch viele andere Bezeichnungen sind mehrdeutig oder in Abhängigkeit von einem speziellen Problem unterschiedlich definiert. Es gibt aber auch Begrifflichkeiten, die etabliert sind. Im Folgenden werden die in dieser Arbeit verwendeten Fachbegriffe, Bezeichnungen und entsprechende Synonyme zur Klarheit definiert.

2.1.1 Definition. Maschinelles Lernen (ML, Machine Learning)

Mit dem Begriff maschinelles Lernen wird der gesamte Lernprozess für ein intelligentes System bezeichnet. Dieser beinhaltet mehrere verschiedene Schritte, die, abhängig von bestimmten Problemstellungen, in ihrer konkreten Realisierung variieren können. Er beschreibt das Entwerfen eines Algorithmus, der das gegebene Problem akkurat löst [1] [28, 34].

Die Grundlage für maschinelles Lernen bilden jedoch immer Daten, deren Art (Bilder, Texte, Tonaufzeichnungen, etc.) und Anzahl, sowie die Menge und Vielfalt deren einzelner vorhandener Eigenschaften vollkommen unterschiedlich sein können. Auf diesen werden dann verschiedenste Berechnungen ausgeführt. Heutzutage handelt es sich dabei oft um verschiedene Netze (siehe Definition 2.2.1).

[1] `https://machinelearning-blog.de/grundlagen/wie-maschinen-lernen/`

Im Allgemeinen gibt es für bestimmte Probleme mehrere Optionen, einen festgelegten Algorithmus mithilfe von Parametern zu modifizieren, sodass die vorgegebene Berechnung möglichst gut an die Daten angepasst ist.

2.1.2 Definition. Training/ Lernen

Ein intelligentes System wird trainiert, wenn die Parameter eines Algorithmus (z.B. Gewichte in einem NN) verändert werden, sodass die Ausgabe des Netzes (siehe Kapitel 2.2) schrittweise verbessert und an die Datengrundlage angepasst wird [42]. Eine genauere Erläuterung des Lernprozesses ist in Kapitel 2.3 angegeben.

2.1.3 Definition. Datenpunkt

Ein Datenpunkt ist hierbei ein einzelnes Objekt (für den Kreditdatensatz zum Beispiel die Daten für eine einzelne Person) aus allen vorhanden Daten, die für das Training oder das Testen einer Funktion verwendet werden. Dabei wird entsprechend zwischen Trainings- und Testdaten unterschieden [42, 51]. Ein Trainingsdatum wird zum Trainieren, also im Lernprozess eines Netzes (siehe Kapitel 2.2) verwendet [42, 51], ein Testdatum zum Testen der erlernten Klassifikationen des Netzes genutzt. Bei Letzterem handelt es sich um für das Netz noch unbekannte Daten, die dazu dienen, die Genauigkeit der Vorhersagen eines Netzes zu bestimmen [42, 51].

Eine **Eigenschaft** oder ein **Feature** beschreibt bestimmte Werte eines konkreten Datenpunktes. Für das „Objekt“ einer Person können Eigenschaften zum Beispiel der Name, das Geschlecht oder das Alter sein [21, 42].

Neben den vielen Unterschieden in Definition und Verwendung bestimmter Begriffe haben sich jedoch einige allgemein durchgesetzt. Zu diesen gehören grundlegende Konzepte wie Supervised und Unsupervised (und Semi-Supervised) Learning (siehe Definitionen 2.1.6 und 2.1.7) und die Unterscheidung in Klassifizierungs- und Regressionsprobleme (siehe Definitionen 2.1.5 und 2.1.4). Letztere basiert zum Teil auf der Unterscheidung von Eigenschaften in qualitative und quantitative Merkmale.

Eine Eigenschaft ist **qualitativ**, wenn sie nominal-skaliert (durch einen Begriff beschrieben) oder ordinal-skaliert (durch Ziffern beschrieben, ohne Aussage über eine Größe der Abstände) ist. Auf diesen Werten kann in der Regel nicht sinnvoll gerechnet werden, da sie keine einheitlichen Skalen und/oder Abstände einhalten. Des Weiteren sind qualitative Merkmale immer diskret [13, 46]. Typische Beispiele hierfür sind Namen, das Geschlecht oder Schulnoten. Name und Geschlecht sind dabei nominal-skaliert, Schulnoten ordinal.

Eine **quantitative** Eigenschaft lässt sich durch Zahlenwerte angeben. Die Werte können sowohl diskret als auch stetig sein, ermöglichen aber in jedem Fall sinnvolle Rechnungen. Abstände sind auf diesen Merkmalen fest durch bestimmte Einheiten (z.B. durch die Einheit 1 für natürliche Zahlen, cm oder Grad Celsius etc.) vorgegeben [13, 46]. Hierzu zählen zum Beispiel die Größe, das Alter oder andere Messungen, wie das Einkommen.

Auf dieser Unterscheidung basiert die Differenzierung in Regressions- und Klassifizierungsprobleme. Hierbei ist jedoch die Ausgabe (siehe Abbildung 2.2) entscheidend.

2.1.4 Definition. Regressionsprobleme

Handelt es sich bei der Ausgabe maschinellen Lernverfahrens um ein quantitatives Merkmal, liegt ein Regressionsproblem vor [21, 42].

2.1.5 Definition. Klassifikationsprobleme

Gibt ein maschinelles Lernverfahren ein qualitatives Merkmal aus, handelt es sich um ein Klassifikationsproblem [21, 42, 51].

Neben der Unterscheidung aufgrund der Ausgabeart eines Algorithmus wird auch die grundlegende Art des Lernens unterschieden in Supervised und Unsupervised Learning [2].

2.1.6 Definition. Supervised Learning / überwachtes Lernen

Beim Supervised Learning sind alle Trainingsdaten klassifiziert. Die Ausgabe, die vom Algorithmus errechnet werden soll, steht fest. Anhand dieses vorgegebenen korrekten Outputs kann somit das Ergebnis automatisch geprüft und eine Fehlerrate bestimmt werden [21, 51].

Zum Beispiel würde folgendes Szenario als supervised Learning kategorisiert werden: Die vorhandenen Daten sind Personendaten, anhand derer die Kreditwürdigkeit bestimmt wurde. Alle Personendaten wurden bereits beurteilt (Kredit erhalten/ Kredit nicht erhalten) und sind mit den entsprechenden korrekten Labeln versehen. Ein intelligentes System wird trainiert und lernt, Vorhersagen zu machen. Die Genauigkeit der Vorhersagen kann über die Fehlerrate anhand der abweichend klassifizierten Beispiele bestimmt werden.

2.1.7 Definition. Unsupervised Learning / unüberwachtes Lernen

Beim Unsupervised Learning sind die vorhandenen Trainingsdaten nicht vorher klassifiziert. Das heiß, sie haben keine Label und es gibt oft keine „korrekten“ Antworten, da die zugrundeliegenden Daten keine besondere vorgegebene Einordnung enthalten. Es sollen Strukturen in der Datenmenge erkannt werden [21, 51].

Zum unsupervised Learning gehört beispielsweise die Bestimmung von Ähnlichkeiten verschiedener Nutzer um Vorschläge (Filme, Produkte, etc.) zu generieren, die an die einzelnen Personen angepasst sind. Hierbei können die Nutzer nicht von vornherein in bestimmte Gruppen klassifiziert werden und eine Fehlerberechnung anhand falscher Ausgaben ist ebenfalls nicht möglich.

[2] https://machinelearning-blog.de/grundlagen/welche-arten-von-maschinellem-lernen-gibt-es/

2.2 Aufbau Neuronaler Netze

Für Berechnungen von Klassifikationen werden oftmals Neuronale Netze (NN) verwendet. Es kann verschiedenste Varianten dieser Netze geben wie zum Beispiel Tiefe Neuronale Netze (Deep Neural Networks, DNN), Convolutional Neural Networks (CNN) oder Reccurrent Neural Networks (RNN). Jeder dieser Typen hat besondere Eigenschaften, wie einzelne Schichten verbunden werden oder welche Schichtentypen verwendet werden [48, 51, 58]. Für diese Arbeit ist lediglich das verwendete Neuronale Netz wichtig und daher wird auch nur dieses definiert:

2.2.1 Definition. Neuronales Netz (NN)

Ein Neuronales Netz ist die Kombination vieler einzelner Funktionen. Es besteht aus Neuronen, die in Schichten angeordnet sind (siehe Abbildung 2.1). Dabei bestimmt die Anzahl der Schichten, ob es sich um ein Single-Layer NN (es gibt nur eine Schicht) oder Multi-Layer NN (es hat mehrere Schichten) handelt. Ein Netz hat immer eine Eingabe- und Ausgabeschicht (auch Input- und Output-Layer genannt). Diese sind im Falle eines Single-Layer NN identisch. Die Eingabeschicht enthält für jede Eigenschaften, die die Datenpunkte haben, ein Neuron. Die Ausgabeschicht enthält die Ausgaben des Netzes [16, 48]. Neben den essentiellen Schichten kann ein NN auch weitere, sogenannte versteckte bzw. innere Schichten (auch Hidden Layer) haben. Diese liegen zwischen der In- und Output-Schicht.

Jedes einzelne Neuron erhält seine Eingabe aus allen eingehenden Kanten. Diese sind die Ausgaben der vorherigen Neuronen, multipliziert mit den entsprechenden Gewichten (siehe 2.2 und 2.3). Es ist definiert über die Funktion $a_j^k = \sum_i a_i^{k-1} w_{ij}^k + b_j^k$. Ein **Neuron** wird im Folgenden mit a_j^k bezeichnet. Dabei bestimmt der Index k die Schicht in der sich ein Neuron befindet, j indiziert es innerhalb der Schicht k. a_i^{k-1} beschreibt alle Neuronen der vorherigen Schicht (näher am Input-Layer). Die Inputs für a_j^k sind, w_{ij}^k das **Gewicht**, das die beiden Neuronen (a_i^{k-1} und a_j^k) verbindet und b_j^k den **Bias** des Neurons a_j^k. Auch hier gibt k die entsprechende Schicht an, in der der Bias sich befindet, j bestimmt das Neuron, zu dem dieser Bias-Wert gehört. Ein Bias wird dazu genutzt, die Inputs zusätzlich zu modifizieren und anpassbar zu machen [32, 51]

Auf den so berechneten Eingaben jedes Neurons wird meist eine Aktivierungsfunktion (Definition 2.2.2) angewandt, welche die Ausgabe für das Neuron zurückgibt. Diese entspricht dem Wert des Neurons [32, 39, 51].

Neuronen sind in verschiedenen Schichten angeordnet. Dabei gibt jede Schicht immer eine bestimmte Aktivierungsfunktion (Definition 2.2.2) der Neuronen vor. Der Aufbau der Schicht ist ausschlaggebend für ihren Namen (z.B. Convolution, Pooling, Linear, etc.) [39, 42].

Innerhalb einer Schicht sind Neuronen nicht miteinander verbunden. Ein Neuron kann jedoch Verbindungen zu mehreren Neuronen aus anderen (vorherigen oder nachfolgenden) Schichten haben. Dabei sind vorherige Schichten solche, die näher an der Eingabeschicht liegen, nachfolgende analog solche, die näher an der Ausgabeschicht liegen. Diese Verbindungen laufen vom Input-Layer in Richtung des Output-Layers, nicht entgegengesetzt (siehe Abbildung 2.1).

Die Eingabe-Neuronen stellen die einzelnen Eigenschaften eines Objekts dar. Für ein Bild sind es die einzelnen Pixel, bei einer Person können es Merkmale wie zum Beispiel das Alter, die Größe und das Geschlecht sein [16]. Ausgabe-Neuronen bestimmen die Klassifikation und somit die Ausgabe, die das Netz errechnet hat. Es kann hier eine einzelne oder auch mehrere verschiedene Ausgaben geben [16].

Die einzelnen Neuronen sind über Kanten miteinander verbunden. Diese beschreiben Gewichte (siehe Abbildungen 2.1 und 2.2) [16]. Diese Gewichte werden mit w_{ij}^k bezeichnet und enthalten Zahlenwerte,welche die Auswirkung der Ausgabe auf das nachfolgende Neuron beeinflussen. Der Index k bestimmt hierbei die Schicht, die sich näher am Input befindet. Neuron i liegt in Schicht k, j in Schicht k+1.

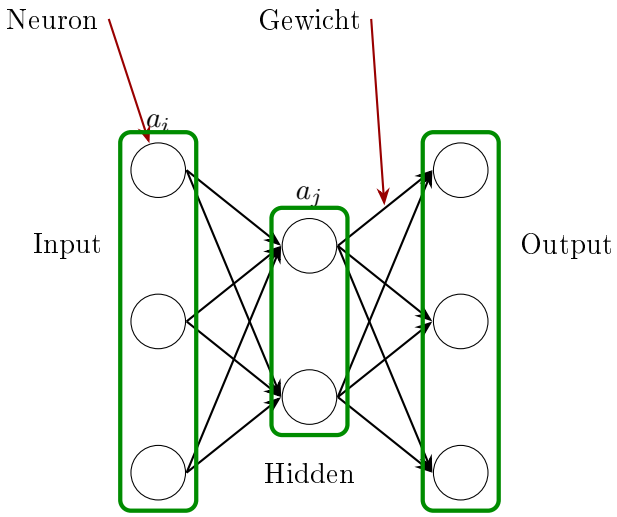


Abbildung 2.1: Es wird ein Neuronales Netz, das aus drei Schichten besteht, gezeigt. Links ist die Eingabeschicht mit drei Neuronen zu sehen. In der Mitte gibt es eine versteckte Schicht, die zwei Neuronen hat. Rechts ist eine Ausgabeschicht visualisiert, die erneut aus drei Neuronen besteht. Die grünen Kästen sollen die einzelnen Schichten deutlich machen. Zudem sind die Bezeichnungen „Neuron“ und „Gewicht“ zur Verdeutlichung beigefügt. a_i und a_j stellen beispielhafte Indizierungen der Neuronen, über denen sie stehen, dar.

Der Input eines Neurons setzt sich aus mehreren Teilen zusammen. Er wird im Folgenden mit z_j^k bezeichnet und besteht aus der Summe der Ausgaben vorheriger Neuronen, multipliziert mit den jeweiligen Kanten-Gewichten und einer eventuellen Addition eines

Bias: $z_j^k = \sum_i a_i \cdot w_{ij}^k + b_j^k$. Die Ausgabe eines Neurons (Neuronenaktivierung) errechnet sich aus der Anwendung einer Aktivierungsfunktion auf den Neuroneninput:

$$a_j = \sigma(z_j^k) = \sigma(\sum_i a_i \cdot w_{ij}^k + b^k) \tag{2.1}$$

2.2.2 Definition. Aktivierungsfunktion

Die Aktivierungsfunktion wird innerhalb eines Neurons auf den jeweiligen Input angewandt und errechnet die entsprechende Ausgabe. Diese Funktionen können verschiedenste Formen haben (z.B. Sigmoid, ReLU, etc.), müssen jedoch immer differenzierbar sein [51].

Wird eine Aktivierungsfunktion verwendet, so wird dies über die Notation σ gekennzeichnet.

Sigmoide Funktionen haben typischerweise eine S-Form. Sie sind beschränkt, differenzierbar und haben genau einen Wendepunkt [45].

Rectifier Linear Unit (ReLU)-Funktionen bilden das Maximum von 0 und einem weiteren Wert. Sie sorgen also dafür, alle negativen Eingaben auf 0 abzubilden. Alle anderen Werte werden auf positive Werte, z.B. die Identität abgebildet [1, 5, 51].

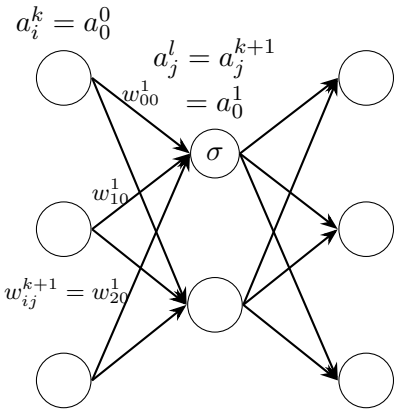


Abbildung 2.2: Hier wird eine Visualisierung der aufgelisteten Notationen dargestellt. Ein Neuron wird über einen hochgestellten Index, der die Schicht bestimmt (hier k und $l = k + 1$), und einen tiefgestellten Index, der das Neuron in der entsprechenden Schicht angibt (hier i und j) indiziert. Auch die Gewichtskanten erhalten hochgestellte und tiefgestellte Indizes. Der Hochgestellte gibt hier die Schicht an, in der die Gewichtskante „ankommt“. Die tiefgestellte Indizierung besteht aus den konkreten Indizes der Neuronen, die die Kante verbindet. σ beschreibt eine beliebige Aktivierungsfunktion.

2.3 Der Lernprozess Neuronaler Netzwerke

Damit Neuronale Netze gute Ergebnisse liefern können, müssen sie zunächst trainiert werden. Dieses Training wird auch als Lernprozess bezeichnet. Die grundlegende Idee eines Lernprozesses für den Fall eines Regressionsproblems mit Supervised Learning wird im Folgenden anhand von Beispielen mit reellen Werten (wie für Personendaten, Daten zur Kreditwürdigkeit oder viele Weitere) als Eingabe erläutert.

Zu Beginn wird ein Netz erstellt. Dies beinhaltet die Entscheidung über eine Anzahl von Schichten, die jeweiligen Schichten(-typen) und das Festlegen der Reihenfolge, in der die Schichten vom Input- zum Output-Layer durchlaufen werden sollen. Anschließend kann das erstellte Netz mit zufälligen Werten für Gewichtskanten und Bias der Neuronen initialisiert werden [42].

Darüber hinaus ist es wichtig, eine geeignete Fehlerfunktion zu bestimmen. Anhand der Ergebnisse dieser kann der Lernprozess stattfinden.

2.3.1 Definition. Fehlerfunktion/ Fehler/ Kostenfunktion

Eine Fehlerfunktion C gibt anhand eines sogenannten Loss-Wertes an, wie gut oder schlecht eine Klassifizierung des Neuronalen Netzes war [51]. Sie beschreibt die „Distanz“ des vom Netz berechneten zum erwarteten Ergebnis. Daher wird sie auch als Kostenfunktion bezeichnet [42, 51].

Für die Fehlerfunktion wird in dieser Arbeit die kategorische Kreuzentropie (für Softmax-Layer, siehe Kapitel 2.2) verwendet:

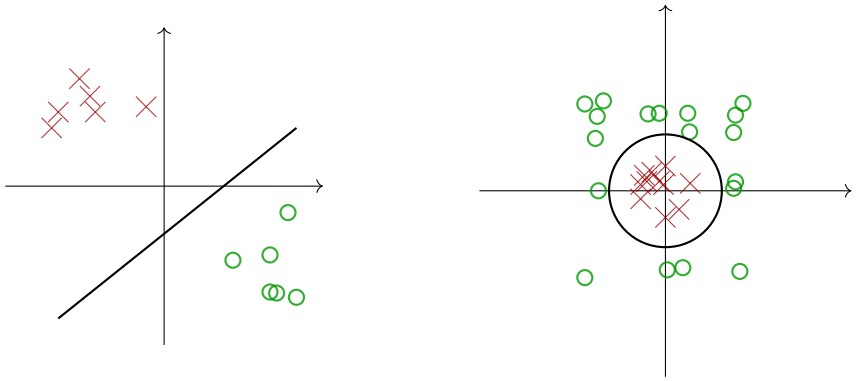
$$H(P,Q) = -\sum_{x \in X} P(x) \cdot log(Q(x)) \tag{2.2}$$

Hierbei sind P und Q zwei Wahrscheinlichkeitsverteilungen, wobei gilt, dass P(x) die Wahrscheinlichkeit des Ereignisses x in P und Q(x) analog die Wahrscheinlichkeit des Ereignisses x in Q beschreibt. Das Ergebnis gibt den Unterschied zwischen P und Q in "Bits"(bzw. der Anzahl an Bits, die nötig wäre, um ein Ereignis von P in Q zu repräsentieren) [3] [25].

Zum Beispiel können Datenpunkte mit den Werten 0 und 1 als Labeln gegeben sein. Das Netz hat die Aufgabe, diese Punkte so zu unterteilen, dass alle Datenpunkte, die einen Bereich teilen, die gleiche Klasse bzw. das gleiche Label haben. Da die Datenpunkte meist vieldimensional sind, ist es oft schwierig, solche Bereiche zu visualisieren. Im Zweidimensionalen Raum kann Abbildung 2.3 eine beispielhafte Aufteilung in zwei Klassen veranschaulichen. Hier werden nur zwei Merkmale der Datenpunkte betrachtet und die Beispiele entsprechend ihrer Werte für Eigenschaft x auf der x-Achse (und analog für Eigenschaft y) eingeordnet. An dieser Stelle beschreibt die Fehlerfunktion die Distanz vom korrekten Ergebnis anhand der falsch klassifizierten Datenpunkte.

[3] `https://machinelearningmastery.com/cross-entropy-for-machine-learning/`

Es muss beachtet werden, dass die Loss-Funktion meist nicht ausschließlich die *Anzahl* falsch klassifizierter Daten ausgibt, sondern diese zusätzlich „modifiziert“. Eine Möglichkeit der Modifizierung stellt die Berechnung der euklidischen Distanz jedes Datenpunktes zur Trennung der bestimmten Bereiche dar. Punkte, die korrekt eingeordnet wurden, können eine positive Ausgabe erhalten, fehlerhaft zugeordnete hingegen eine negative. So kann die Fehlerrate des Netzes stetig und gewichtet bestimmt werden. Analog zum zweidimensionalen Beispiel geschieht die Berechnung auch für höherdimensionale Datenpunkte.



(a) Aufteilung durch eine lineare Funktion **(b)** Aufteilung durch eine polynomielle Funktion

Abbildung 2.3: Es sind zwei verschiedene Aufteilungen von Datenpunkten in je zwei unterschiedliche Klassen abgebildet. In Abbildung 2.3a ist als Trennung eine lineare Funktion angegeben. Diese ist nur beispielhaft gewählt und soll verdeutlichen, wie eine lineare Entscheidungsgrenze visualisiert werden kann. 2.3b zeigt ein ähnliches Beispiel für eine kreisförmige Entscheidungsgrenze, die durch eine polynomielle Funktion entstanden ist.

Nach der ersten Erstellung und Initialisierung des Netzes beginnt der eigentliche Lernprozess. Dabei wird für jeden Trainingsdatenpunkt der gleiche Prozess durchgeführt: Ein Trainingsbeispiel wird vom Netz klassifiziert und der entsprechende zugehörige Fehler wird berechnet. Um diesen zu verringern muss nun das Netz rückwärts durchlaufen und die Parameter der einzelnen Neuronen angepasst werden, sodass der Fehler minimiert wird [16, 42].

2.3.1 Gradientenabstieg

Der Gradientenabstieg kann genutzt werden, um den Fehler zu minimieren. Hierbei wird je Neuron für alle Inputs (z_j) ein Gradient bestimmt. Dieser ergibt sich aus der partiellen Ableitung der Fehlerfunktion (C) über alle Neuroneninputs (z) nach dem jeweiligen Gewicht (w_j):

$$\forall_j \frac{\partial C}{\partial w_j} \tag{2.3}$$

Durch diese partiellen Ableitungen kann für jedes Gewicht die Richtung des größten Anstiegs bestimmt werden und so durch Subtraktion dieses maximalen Anstiegs von dem

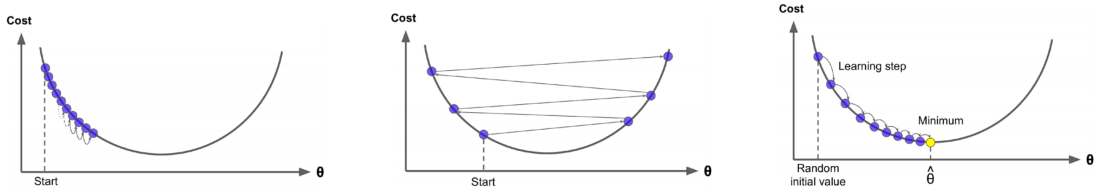
Gewicht eine maximale Verkleinerung des Fehlers erzeugt werden [42, 51]. Ein Gewicht wird dann immer um den Wert der jeweiligen partiellen Ableitung angepasst und auf diese Weise für die aktuellen Iteration optimiert.

Das Vorgehen bei der Berechnung der partiellen Ableitungen und des Neusetzens des Gewichts kann wie folgt verbildlicht werden: Angenommen, eine Person steht auf einem Berg und sucht den schnellsten Weg nach unten. Sie kann nur eine gewisse Distanz weit sehen, wie steil es in diesem Radius nach unten geht. Diese Steigung entspricht der berechneten partiellen Ableitung für jede Richtung (im NN für jedes Gewicht). Die stärkste Steigung wird ausgewählt und die Person begibt sich zum Punkt am Rande ihres vorherigen Sichtfeldes (das Neusetzen des Gewichts als die Differenz des alten Gewichts und der partiellen Ableitung). An dieser Position beginnt das Ganze von vorn und es wird wieder nach der stärksten Steigung Ausschau gehalten.

Dabei ist für Neuronale Netze zusätzlich durch die sogenannte Schrittweite einstellbar, wie stark die Optimierungen sich auswirken. Im Beispiel entspricht dies der Veränderung der Sichtweite der Person.

2.3.2 Definition. Schrittweite

Bei der Schrittweite α handelt es sich um einen Parameter, der in der Optimierung wichtig ist, um zu verhindern, dass sich der Extremstelle in zu kleinen (siehe Grafik 2.4a) oder zu großen Schritten genähert wird (siehe Grafik 2.4b). Zu kleine Schritte bedeuten, dass der Lernprozess sehr lange dauert, zu große Schritte können dafür sorgen, dass das Optimum im schlimmsten Fall nie erreicht werden kann (siehe Grafik 2.4b)[18, 51].



(a) Zu kleine Schrittweite [18] **(b)** Zu große Schrittweite [18] **(c)** Optimale Schrittweite [18]

Abbildung 2.4: In den drei Abbildungen sind verschiedene Schrittweiten für einen Gradientenabstieg visualisiert. Links ist eine zu kleine Schrittweite gewählt, sodass die Annäherung an die Extremstelle sehr langsam voranschreitet. In diesem Fall dauert der Gradientenabstieg sehr lange, erreicht aber schlussendlich das Minimum. In der Mitte ist der Ablauf für eine zu große Schrittweite visualisiert. Hier ist der Extremfall dargestellt, in dem aufgrund der fehlerhaften Schrittweite niemals das Optimum erreicht wird, weil die Funktion niemals konvergiert. Es ist jedoch auch möglich, mit einer zu großen Schrittweite weiterhin das Minimum zu erreichen. Die rechte Abbildung zeigt den optimalen Ablauf mit passend gewählter Schrittweite an. Die Funktion nähert sich dem Minimum mit passenden Schritten, die weder zu klein noch zu groß sind. Sie konvergiert in optimaler Zeit.

Insgesamt ergibt sich hieraus also der Gradient

$$w_j^k = w_j^{k-1} - \alpha \frac{\partial C}{\partial w_j^{k-1}} \tag{2.4}$$

2.3.2 Backpropagation

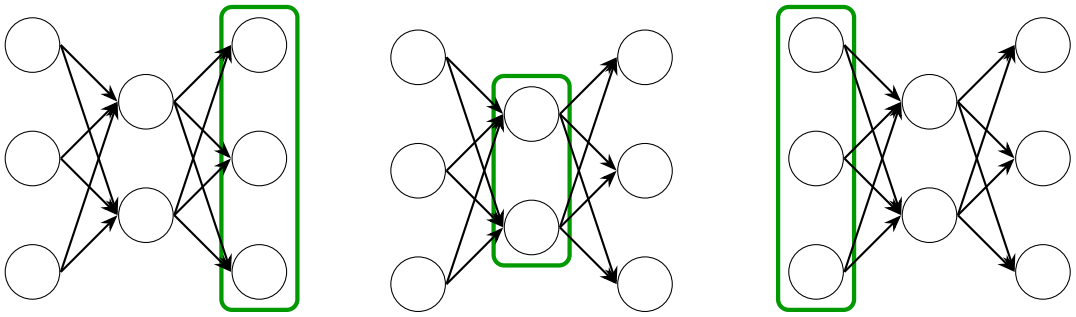
Um nun diese maximale Verkleinerung des Fehlers durch das gesamte Netz zu propagieren kann Backpropagation genutzt werden. Diese nutzt den Gradientenabstieg und wendet ihn schichtweise für alle Neuronen an. So werden an jeder Stelle des Netzes Optimierungen vorgenommen, die zu einer Minimierung des Fehlers führen [42, 51].

Hierbei wird durch das Modifizieren von Neuronenoutputs aus vorherigen Schichten (z_j^{k-1}) die Berechnung des optimalen Gradienten je Neuron eine Schicht näher zum Input (in Schicht $k-1$) verlagert.

$$\forall_j \frac{\partial C}{\partial w_j^{k-1}} = \forall_j \frac{\partial C}{\partial a_j^k} \cdot \frac{\partial a_j^k}{\partial z_j^{k-1}} \cdot \frac{\partial z_j^{k-1}}{\partial w_j^{k-1}} \tag{2.5}$$

Hierbei ist a_j^k: Output von Neuron a_j in Schicht k, z_j^{k-1}: Input für Neuron a_j in Schicht k und w_j^{k-1}: Gewicht, das zum Input z_j^{k-1} gehört. Da in der partiellen Ableitung nicht nur die Gewichte, sondern zusätzlich auch die Inputs für das entsprechende Neuron betrachtet werden, wird durch die Gradientenberechnung die Rückpropagation angestoßen. Aufgrund der jeweiligen partiellen Ableitungen wird jedoch für jedes Neuron der vorherigen Schicht (k-1) nur der Teil betrachtet, dessen partielle Ableitung nicht 0 ist.

Diese Ableitungen werden gebildet, bis die Eingabeschicht erreicht ist und die Neuronenoutputs nicht mehr verändert werden können, da es die Eingabewerte sind. Auf diese Weise werden die Werte der einzelnen Neuronen und die Gewichte im Netz Schicht für Schicht so verändert, dass sich der errechnete Fehler verringert [16, 42, 51].



(a) Backpropagation Schritt 1 **(b)** Backpropagation Schritt 2 **(c)** Backpropagation Schritt 3

Abbildung 2.5: Hier wird schichtenweise der Ablauf der Backpropagation für das Beispiel-NN gezeigt. In Schritt 1 (linke Abbildung) wird zunächst nur die Ausgabeschicht betrachtet. Von hier beginnend werden die Gradienten bestimmt und in die nächst-frühere Schicht (Schritt 2, in der Mitte) in Richtung der Eingabeschicht propagiert. Anschließend werden auch dort die Gradienten berechnet und diese in die erste Schicht (Abbildung rechts) weitergegeben. Hier kann nicht weiter propagiert werden, da es die Eingabeschicht ist. Die Berechnung endet.

Anschließend wird dieser Prozess für alle weiteren Trainingsdaten durchgeführt.

Nachdem der Lernprozess abgeschlossen wurde sind alle Gewichte fest und dürfen nicht mehr angepasst werden [16].

Für die Validierung werden dann die Testdaten genutzt. Diese werden ebenfalls dem Netz als Eingabe übergeben und von ihm klassifiziert. Für alle Testdaten werden die jeweiligen Fehlerraten bestimmt und ein gesamter Fehler z.B. über die Berechnung des Durchschnitts gebildet. Anhand dieses Fehlers kann die Genauigkeit der Klassifizierung des Netzes für fremde, vom Netz zuvor nicht gesehene Daten bestimmt werden [42, 51].

Für die Validierung gibt es mehrere Ansätze, mit denen eine möglichst genaue Aussage über die Genauigkeit eines Intelligenten Systems getroffen werden kann. Zu diesen zählt zum Beispiel Cross-Validation. Es ist jedoch auch möglich, mit je einem Trainings- und einem Testset das Netz zu validieren [42].

2.3.3 Definition. Cross-Validation

Cross-Validation wird für das validieren von Trainingsergebnissen verwendet. Hierbei werden die vorhandenen Daten mehrfach in verschiedene Trainings- und Testsets (Mengen aus Trainings- und Testdaten) eingeteilt. Anschließend wird das Netz mit allen bestimmten Trainingssets trainiert und mit den zugehörigen Testsets validiert. Von den bestimmten Genauigkeitswerten wird anschließend der Durchschnitt als repräsentative Genauigkeit des Netzes berechnet [42].

2.4 Das gegebene Neuronale Netz

Für diese Arbeit wurde ein Neuronales Netz zur Verfügung gestellt. Dieses wurde auf dem German Credit Datensatz trainiert. Es besteht aus den folgenden drei Schichten in

der aufgezählten Reihenfolge: Linear (mit ReLU Aktivierung), Dropout und Linear (mit Softmax Aktivierung). Dabei werden ReLU, Dropout und Softmax hier als eigene Schichten gezählt, da sie in der Implementierung als separate Schicht umgesetzt wurden.

Ein **Linear Layer** ist eine Schicht, in der alle Neuronen der aktuellen Schicht jeweils mit allen Neuronen der nachfolgenden Schicht verbunden sind [38, 42, 51]. Außerdem enthält sie meist eine zusätzliche Aktivierungsfunktion. In der ersten Schicht wird in diesem NN eine ReLU-Funktion verwendet [1, 42, 51]. Der **Dropout** Layer wird genutzt, um Overfitting vorzubeugen. Dabei werden im Trainingsprozess zufällig Knoten der Schicht auf Null gesetzt, um so Informationen auszuschließen [12, 51]. **Overfitting** bedeutet dabei, dass die beim Training bestimmte Funktion sich zu sehr an den Trainingsbeispielen orientiert. Die Daten, die für den Lernprozess verwendet wurden, können (nahezu) fehlerfrei bestimmt werden, auf unbekannten Datenpunkten erzielt die Funktion dagegen eher schlechte Ergebnisse und macht viele Fehler [10, 42, 51] . Der **Softmax** Layer kann entweder ein eigener Schichtentyp, oder wie hier, ein Linear Layer mit einer Softmax-Aktivierungsfunktion sein. Er wird oft bei Klassifikationsproblemen mit mehreren möglichen Ausgaben verwendet und bestimmt die Wahrscheinlichkeiten für alle Ausgabeknoten. Aus diesen wird dann derjenige mit der höchsten Wahrscheinlichkeit als Klassifizierung des Netzes ausgewählt [2, 14, 42, 51].

Das Netz stellt zudem eine Vorverarbeitung zur Verfügung, mithilfe dessen ordinale Werte normalisiert und nominale Eigenschaften codiert werden. Die Normalisierung geschieht mithilfe des sklearn *StandardScalers* [4]. Dieser nutzt die Formel $z = \frac{x-u}{s}$, wobei x den entsprechenden Datenpunktvektor, u den Durchschnitt aller Trainingsbeispiele und s die Standardabweichung der Trainingsbeispiele beschreiben.

Die Transformation der kategorischen Variablen geschieht über *One-Hot Codierung* [5]:

2.4.1 Definition. One-Hot-Codierung

Bei der One-Hot-Codierung werden qualitative Eigenschaften in mehrere Vektoren aufgeteilt. Für jede mögliche Ausprägung eines nominalen Wertes wird ein eigener Vektor erstellt, der ebendiese Ausprägung repräsentiert [51]. Eine qualitative Eigenschaft wird nun über alle diese Vektoren zusammen angegeben. Dabei kann nur genau einer den Wert **1** erhalten (der Vektor, der zur entsprechenden Ausprägung gehört, die das Feature im nicht-codierten Zustand hatte). Alle anderen haben den Wert **0**.

Beispielsweise würde der Beziehungsstatus, der die Ausprägungen ledig, verheiratet, geschieden und verwitwet haben kann, zu einer Erzeugung von vier neuen Eigenschaften führen. Von diesen kann immer nur genau eine den Wert 1 haben, da ein Datenpunkt

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder`

nur genau eine der Ausprägungen enthalten kann. Ist eine Person verheiratet, erhalten die neuen Eigenschaften Beziehungsstatus: ledig, Beziehungsstatus: geschieden und Beziehungsstatus: verwitwet alle den Wert 0 und Beziehungsstatus: verheiratet wird auf 1 gesetzt.

Diese Codierung ist Teil der Vorverarbeitung des Netzes und kann zurückgerechnet werden. Die Umkehrung der Codierung ist für eine sinnvolle und verständliche Visualisierung der Werte notwendig. Hierfür stellt der verwendete Transformierer eine Methode zum invertieren der Vorverarbeitung bereit.

2.5 Ansätze

Aus den anfangs erwähnten fünf Ansätzen können drei (die Modellerklärung, die Black Box Outcome Explanation und das Black Box Inspection Problem) gut als erklärende Ansätze verwendet werden. Die anderen Ansätze gehen von der Erstellung einer eigenen Lösung aus, die anstelle einer Black Box Funktion verwendet wird, auf dieser Herangehensweise liegt jedoch in dieser Arbeit kein Fokus. Die gewählten Ansätze entsprechen jeweils einem der drei verwendeten Grundideen zum Öffnen der Black Box, die von komplexen Neuronalen Netzen gebildet wird. Eine Erklärung mithilfe von Surrogatmodellen passt zu der Idee der Modell-Erklärung. Counterfactuals entsprechen einer Erklärung der Entscheidung lediglich anhand der Ausgabe und sind damit Black Box Outcome Explanations. Der Ansatz der Erklärung durch die Bestimmung von Feature Contributions erklärt die innere Funktionalität des Netzes und bietet daher eine mögliche Lösung für das Black Box Inspection Problem [19].

Im Folgenden werden die genauen Grundideen der drei Ansätze näher erläutert. In Kapitel 3 werden zudem die Theorie und in Kapitel 4 die Implementierung für die konkreten Algorithmen beschrieben.

2.5.1 Surrogatmodelle

Surrogatmodelle sind einfache, leicht interpretierbare, Approximationen von komplexen Funktionen, die bestenfalls eine kurze Laufzeit haben. Ein solches Modell wird so trainiert, dass es genau wie die Black Box-Funktion klassifiziert, inklusive der Fehler. Auf diese Weise können sie für Erklärungen von Black Box Funktionen genutzt werden. Bei der Approximation handelt es sich immer um ein leicht interpretierbares Modell, das sich weiterhin wie die zu erklärende Funktion verhält [29, 54].

Hierfür können alle interpretierbaren Modelle genutzt werden, zum Beispiel Entscheidungsbäume [15, 53] oder lineare Modelle [21, 29]. Für diese Arbeit werden diese beiden Methoden implementiert.

Welche Modelle tatsächlich als leicht interpretierbar eingestuft werden können ist jedoch eine subjektive Entscheidung. Im Allgemeinen können hierzu aber diejenigen Algorithmen

gezählt werden, die zu einem gewissen Grad oder vollständig transparent sind [44]. Die Aufstellung einer allgemeinen, alles umfassenden Definition für leicht interpretierbare Modelle ist nicht möglich, da je nach Anwendungsgebiet verschiedene Eigenschaften greifen [44]. Generell gelten sowohl Bäume als auch lineare Modelle als interpretierbar. Hier ist die Verständlichkeit jedoch stark von der jeweiligen Größe abhängig. Je tiefer ein Baum wird, desto weniger greifbar und interpretierbar wird er für einen Nutzer. Ähnliches gilt auch für lineare Modelle. Je mehr Variablen betrachtet werden, desto unübersichtlicher wird die Erklärung.

Alle entsprechenden Modelle haben jedoch gemein, dass sie für einen Nutzer Sinn ergeben, sich also in einer für Menschen verständlichen Domäne befinden [32].

2.5.2 Counterfactuals

Counterfactuals liefern Erklärungen anhand von beispielhaften Änderungen und sind damit ein beispielbasierter Ansatz [29]. Die Erklärung entsteht aus einer Wahl gut passender Attribute und einer Beschreibung der möglichen Wertänderung. Gut passend bedeutet hier, dass durch die Wahl der veränderten Attribute ein noch möglichst ähnlicher Datenpunkt entstehen oder gefunden werden soll. Die Veränderung einzelner Eigenschaften des Objekts, sowie die Anzahl der gewählten Eigenschaften muss dafür so gering wie möglich sein. Zugleich soll durch diese Attribute aber eine Änderung der vorhergesagten Klasse erreicht werden [55].

Diese beispielbasierten Erklärungen sind oftmals von folgender Form: Wäre Attribut $X = x_i$ und $Z = z_i$ gewesen, dann wäre die Entscheidung als y_i anstelle von y_j ausgefallen. Konkret könnte im Fall von Kreditwürdigkeitsbeurteilung die Aussage wie folgt lauten: Wäre das Einkommen in Höhe von $x_i = 5000$ € monatlich und die Schulden nicht höher als $z_i = 10000$ € gewesen, dann wäre die Kreditwürdigkeit als hoch genug eingestuft worden (y_i = Kredit wird bewilligt statt y_j = Kredit wird abgelehnt).

Auf diese Weise ist es möglich, ausschlaggebende Eingabe-Eigenschaften, sowie Handlungshinweise und Ähnliches bereitzustellen. Counterfactuals können so einen Einblick in die für eine Entscheidung wichtigen Eigenschaften liefern und zugleich Probleme bei der Entscheidungsfindung aufdecken [55].

2.5.3 Feature Contribution

Die Erklärung durch das Bestimmen von Feature Contributions beschreibt einen Ansatz, bei dem der Beitrag einzelner Komponenten, wie zum Beispiel Neuronen in versteckten Schichten, zu einer Klassifizierung betrachtet wird [36, 52]. Hierfür wird anhand der Eingabe-Eigenschaften eine Erklärung bestimmt, welche die Wichtigkeit einzelner Eigenschaften für eine Entscheidung und somit den Beitrag dieser Inputs für das Ergebnis errechnet [32].

Auf diese Weise ist es möglich, auch einen Einblick in die inneren Schichten des Modells zu geben, da Contributions nicht nur auf die Eingabeschicht beschränkt sein müssen. So kann mehr Transparenz geschaffen und die Funktionsweise der Black Box verständlich gemacht werden. Auch dieser Ansatz eröffnet zudem die Option, mögliche Probleme des Modells anhand der bestimmten Wichtigkeiten für die Entscheidung zu erkennen [32].

Der Unterschied zu den vorherigen Ansätzen ist hier, dass auch alle inneren Schichten des neuronalen Netzes, die Hidden Layer, Teil der Erklärung sind. Das Verständnis wird nicht, wie bei Counterfactuals, ausschließlich aus Ein- und Ausgaben bestimmt und hat im Gegensatz zur Erstellung vollständig neuer Modelle, wie bei der Verwendung von Surrogatmodellen, direkten Bezug zur zugrundeliegenden Funktion [29].

Zu Algorithmen für diesen Ansatz gehören zum Beispiel die Layerwise Relevance Propagation (LRP) [32] oder eine Kombination aus Shapley Values und dem DeepLIFT Verfahren durch DeepSHAP [11].

Kapitel 3

Erklärungen in der Praxis

3.1 Surrogatmodelle

Wie bereits in Kapitel 2.5.1 beschrieben, bestimmen Surrogatmodelle eine Annäherung an das eigentliche Neuronale Netz. Ein solches Modell verwendet die Datenpunkte, die von dem NN klassifiziert werden und die vom Netz berechneten Ausgaben. Auf Basis dieser werden hier ein Entscheidungsbaum bzw. eine logistische Regression berechnet, mit denen dann weitere Datenpunkte erklärbar sind.

3.1.1 Entscheidungsbaum

Für das Surrogatmodell wird in dieser Arbeit ein Entscheidungsbaum verwendet, da dieser zu den leichter interpretierbaren Ansätzen [29] gezählt wird. Ein Entscheidungsbaum besteht aus inneren Knoten, Blättern und diese Knoten verbindenden Zweigen. Die Blätter enthalten konkrete Klassifizierungen, die inneren Knoten repräsentieren Verzweigungen, die bestimmte Abfragen (auch Tests genannt [41]) enthalten. Diese Tests prüfen für einen Datenpunkt jeweils, ob die entsprechende Eigenschaft einen bestimmten Grenzwert über- bzw. unterschreitet. Anhand dieser Abfragen kann der entsprechende Zweig eines zu klassifizierenden Datenpunktes verfolgt werden, indem der passende Pfad gewählt wird [41]. Ein Knoten kann dabei beispielsweise abfragen, ob das Alter größer als 30 ist. Er hat dann zwei Kanten, die jeweils für „ja“ (das Alter ist größer als 30) bzw. „nein“ (das Alter ist kleiner oder gleich 30) stehen. Eine Erklärung für konkrete Entscheidungen entspricht der Ausgabe des Pfades, den der zu erklärende Datenpunkt genommen hat [35]. Die Kanten dieses Pfades ergeben eine Konjunktion verschiedener Eigenschaften, die zum Blatt und somit zur entsprechenden Entscheidung geführt haben.

Für diese Arbeit wird ein Entscheidungsbaum trainiert, der das Neuronale Netz approximiert und auf den Trainingsdaten die gleichen Klassifizierungen vornimmt wie das Netz.

Der implementierte Entscheidungsbaum nutzt den DecisionTreeClassifier und zugehörige Methoden von sklearn [9, 37]. Dieser Klassifizierer nutzt eine modifizierte Version des CART (Classification and regression trees) Algorithmus [29] zur Erstellung der Entscheidungsbäume. Die Implementierung verfügt über mehrere einstellbare Hyperparameter. Beispielsweise ist eine Unterscheidung zwischen der Verwendung der Gini-Unreinheit (siehe Definition 3.1.1) oder Entropie (siehe Definition 3.1.2) als Maß der Unreinheit der einzelnen Knoten möglich [29].

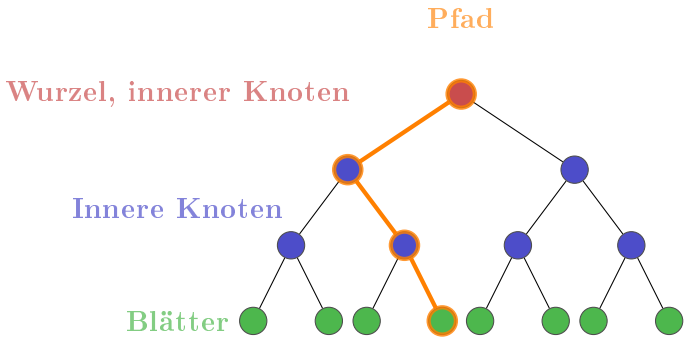


Abbildung 3.1: Hier sind ein beispielhafter Entscheidungsbaum und die verwendeten Bezeichnungen visualisiert. Der oberste rote Knoten ist die Wurzel, die jedoch hier nicht von gewöhnlichen inneren Knoten (in blau dargestellt) unterschieden werden muss. In grün sind die Blätter des Baumes markiert. Ein Entscheidungsbaum muss weder immer balanciert noch vollständig sein. In orange ist ein beispielhafter Pfad von der Wurzel zu einem Blatt angegeben.

Im weiteren Verlauf gelten für die Beschreibung von Entscheidungsbäumen folgende Namenskonventionen:

Ein Baum besteht aus Knoten und Kanten. Wie in Abbildung 3.1 dargestellt sind alle Knoten, die mindestens einen Nachfolger (auch Kinder) haben, innere Knoten. Diejenigen ohne Nachfolger werden Blätter genannt. Der Knoten, der keinen Vorgänger (auch Eltern) hat, wird als Wurzel bezeichnet. Ein Pfad beschreibt eine Menge von Knoten und die Kanten, durch die diese miteinander verbunden sind. Er beginnt immer in der Wurzel und endet immer in einem Blatt [21, 41].

Knoten des Entscheidungsbaums (sowohl innere als auch Blätter) werden mit $\mathbf{m}$ indiziert, Trainingsvektoren haben die Bezeichnung $\mathbf{x_j^i} \in \mathbf{R^m}$ mit $i \in \{1, ..., n\}$, n $\hat{=}$ Anzahl der Datenpunkte, $j \in \{1, ..., m\}$, m $\hat{=}$ Anzahl der Feature des Datenpunktes und $\mathbf{y^i} \in \mathbf{R^l}$ beschreiben die zu x^i zugehörigen Labelvektoren (Vektoren, die Werte für jedes Ausgabeneuron enthalten). Der hochgestellte Index i beschreibt also, um welchen Datenpunkt bzw. um welches Label es sich handelt. Der tiefgestellte Index gibt die entsprechende Eigenschaft an [21].

Ein Split bezeichnet das Aufteilen der Datenpunkte in einem Knoten in zwei Kind-Knoten. Hierfür muss ein Split-Kandidat $\boldsymbol{\theta(j, t_m)}$ mit Feature $\mathbf{j}$ und Threshold $\mathbf{t_m}$ für die entsprechende Eigenschaft in Knoten $\mathbf{m}$ gewählt werden. Für alle Datenpunkte in dem Knoten wird dann der Wert des Features mit dem Threshold verglichen und so ein Split bestimmt [8, 21, 37].

$\mathbf{N_m} = |Q_m|$ bestimmt die Gesamtzahl aller Beispiele in der Menge $\mathbf{Q_m}$ in Knoten m. Die Teilmengen dieser Datenpunkte werden jeweils mit $\mathbf{Q_m^{left}} = \{(x, y)|x_j^i \leq t_m\}$ für diejenigen, die nicht größer als der gewählte Threshold an Knoten m sind, und $\mathbf{Q_m^{right}} = Q_m \backslash Q_m^{left} = \{(x, y)|x_j^i > t_m\}$ für die verbleibenden Datenpunkte, bezeichnet. Die Gesamtzahl der Feature in Q_m^{left} und Q_m^{right} werden analog zur Bezeichnung von N_m mit $N_m^{left} = |Q_m^{left}|$ und $N_m^{right} = |Q_m^{right}|$ bezeichnet [21, 37, 41]

Eine Funktion $\mathbf{H(Q_m)}$ ist die Unreinheits- bzw. Loss-Funktion, mithilfe derer die Qualität $\boldsymbol{G(Q_m, \theta)} = \frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta))$ eines Splits bestimmt wird [8, 29, 37].

3.1.1 Definition. Gini-Unreinheit

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{3.1}$$

Hierbei entspricht $\mathbf{p_{mk}}$ der Proportion der Observationen in Knoten m, die Klasse $\mathbf{k}$ angehören [21].

$$p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k) \tag{3.2}$$

I ist die Indikatorfunktion [21]:

$$I(y = k) = \begin{cases} 0 & \text{falls } y \neq k \\ 1 & \text{falls } y = k \end{cases} \tag{3.3}$$

Die zweite Funktion zur Berechnung der Unreinheit eines Knotens ist die Entropie:

3.1.2 Definition. Entropie [21]

$$H(Q_m) = -\sum_k p_{mk} log(p_{mk}) \tag{3.4}$$

Der Algorithmus bestimmt dann mit der gewählten Unreinheitsfunktion (Gini-Unreinheit oder Entropie) als Maß eine optimale Lösung, indem $\theta^\star$ rekursiv für $Q_m^{left}(\theta^\star)$ und $Q_m^{right}(\theta^\star)$ berechnet wird, bis ein Abbruchkriterium (zum Beispiel die maximale Tiefe) erreicht wurde.

$$\theta^\star = \arg\min_\theta G(Q_m, \theta)$$

Auch diese Abbruchkriterien sind Hyperparameter, die in sklearn der Funktion übergeben werden können [8, 29, 37].

Insgesamt bestimmt der Algorithmus also zunächst für alle Datenpunkte mögliche Split-Kandidaten $\theta(j, t_m)$. Aus diesen wird mithilfe der gewählten Unreinheitsfunktion die jeweilige Qualität $G(Q_m, \theta)$ berechnet. Anschließend kann derjenige Split ausgewählt werden, der eine maximale Qualität bei minimalem θ erzeugt [8, 29, 37].

Nach der Erstellung kann der Baum zudem zurückgeschnitten werden. Dieser Vorgang wird als Pruning bezeichnet und kann im Allgemeinen helfen, das Verständnis des Nutzers zu erhöhen, indem ein Baum wieder verkleinert wird. Als Pruning Funktion wird bei sklearn das Minimal Cost-Complexity Pruning [8, 29, 37] verwendet.

Hierbei wird für einen gegebenen Baum $\mathbf{T}$ mithilfe des Komplexitätsparameters α zunächst das Cost-Complexity-Maß $R_\alpha(T) = R(T) + \alpha|\tilde{T}|$ bestimmt. $R(T)$ ist die gesamte, gewichtete Unreinheit der Blätter und $|\tilde{T}|$ beschreibt die Anzahl der Blätter des Baumes. Für das Pruning wird anschließend

$$\alpha_{eff}(t) = \frac{R(t) - R(T_t)}{|T| - 1} \tag{3.5}$$

bestimmt und das Blatt, das den kleinsten α_{eff}-Wert enthält, zurückgeschnitten. $R(t)$ beschreibt hierbei das Cost-Complexity-Maß für einen einzelnen Knoten, $R(T_t)$ das Cost-Complexity-Maß für einen Pfad mit t als Wurzel. Der Prozess des Berechnens von $\alpha_{eff}(t)$ für alle Blätter und des Zurückschneidens des schwächsten wird fortgesetzt, bis für alle Blätter t gilt $\alpha_{eff}(t) > \alpha$. Dieser Hyperparameter ist eine weitere vom Nutzer einstellbare Variable, die in der Web-Oberfläche übergeben werden kann. Standardmäßig ist er auf 0 gesetzt, sodass kein Pruning vorgenommen wird [8, 29, 37].

3.1.2 Lineare Modelle (Logistische Regression)

Als zweite Methode im Bereich Surrogatmodelle wird ein lineares Modell eingesetzt. Ähnlich wie Entscheidungsbäume können auch lineare Modelle so trainiert werden, dass sie zur Erklärung des Black Box Algorithmus exakt die gleichen Klassifikationen bestimmen, wie das zu erklärende Modell.

Um eine Darstellung zu errechnen, werden verschiedene Prädiktoren und Koeffizienten in eine (meist lineare) Gleichung der Form $h_W(x^i) = w_0 + w_1 x_1^i + w_2 x_2^i + ... + w_p x_p^i + \epsilon$ verbunden. $h_W(x^i)$ beschreibt die sogenannte Entscheidungsgrenze (auch Decision-Boundary) des Datenpunktes x^i mit den einzelnen Eigenschaften (Prädiktoren) x_j^i. Diese Grenze besteht aus den entsprechenden Koeffizienten $W = \{w_j\}$ mit $j \in \{1, ...p\}$, und den Prädiktoren x_j^i, die durch die Koeffizienten modifiziert werden. w_0 ist ein von den Prädiktoren unabhän-

giger Koeffizient. Die Entscheidungsgrenze, also die Grenze, anhand derer zwischen zwei Entscheidungen unterschieden werden kann, ist in Abbildung 2.3a visualisiert [21].

Die einzelnen Prädiktoren des Modells können kombiniert und zu Polynomen verbunden werden. So ist es auch möglich, quadratische, kubische oder höhere polynomielle Funktionen in linearen Modellen umzusetzen. Ein Beispiel für die Entscheidungsgrenze eines linearen Modells mit polynomiellen Koeffizienten zeigt Abbildung 2.3b [21].

Anhand der erstellten Gleichung lassen sich aus den Koeffizienten die Gewichtungen der Eigenschaften oder Eigenschaftskombinationen, welche die Prädiktoren bilden, ablesen [21]. Diese Gewichtungen können beispielsweise durch ein Balkendiagramm visualisiert werden.

Um eine gute Interpretierbarkeit zu gewährleisten sollten sowohl die Anzahl der verwendeten Feature als auch die Anzahl der Kombinationen und somit die Höhe des Funktionsgrades möglichst gering gehalten werden [21, 44].

Für die Umsetzung eines linearen Modells wird in dieser Arbeit die logistische Regression zur Klassifizierung verwendet die das verwendete Neuronale Netz approximiert und auf den Trainingsdaten die gleichen Klassifizierungen vornimmt, wie das Netz. Grundlage hierfür bilden die vorhandenen sklearn Implementierungen.

Bei der logistischen Regression handelt es sich um ein Modell, das mithilfe der logistischen Funktion $f(x) = \frac{L}{1+e^{-k(x-x_0)}}$ [21] eine Klassifikation bestimmt. Diese gibt die Wahrscheinlichkeiten für bestimmte Ergebnisse an. L beschreibt hierbei den maximalen Funktionswert(standardmäßig 1) k die Steigung der Kurve (ebenfalls im Allgemeinen 1) und x_0 den Wendepunkt auf der x-Achse (für die Standard-Funktion 0). Diese logistische Funktion wird mit der linearen Gleichung, die für lineare Regression verwendet wird, zur logistischen Regression:

3.1.3 Definition. Logistische Regression

$$f(h(x^i)) = \frac{1}{1 + exp(h(x^i))} = \frac{1}{1 + exp(w_0 + w_1 x_1^i + w_2 x_2^i + ... + w_p x_p^i + \epsilon)} \tag{3.6}$$

mit $i \in \{1, ...p\}$ [21, 42]

Es ist weiterhin möglich, aus der Funktion $h(x^i)$ die Entscheidungsgrenze abzulesen [21, 29], auch wenn sie mit der logistischen Funktion kombiniert wird. Für eine Interpretation der Ergebnisse bietet die logistische Regression jedoch weitere Möglichkeiten. $f(h(x^i))$ liefert einen Wert im Bereich $[0, 1]$. Dieser gibt die bedingte Wahrscheinlichkeit $P(y = 1|x^i; w)$ an, mit welcher der Datenpunkt x^i der Klasse **1** (der Kredit wird genehmigt) zugeordnet wird. P ist abhängig von x^i und wird von den Koeffizienten w parametrisiert. Die Koeffizienten selbst werden im Lernprozess der logistischen Regression optimiert. Sie repräsentieren die Gewichtungen der einzelnen Prädiktoren x_j [21, 29, 42].

Um $f(h(x^i))$ zu optimieren wird der Stochastic Average Gradient (SAG) als Solver verwendet. Dieser kombiniert den Stochastic Gradient [42] mit dem Full (bzw. Batch) Gradient [43]. Hierbei werden die Gewichte mithilfe der Funktion

$$\min_w g(w) = \frac{1}{n} \sum_{i=1}^{n} v_i(w) \tag{3.7}$$

optimiert. Dazu gilt $\nabla f_i(w) = h'(w^T x^i) \cdot x^i$ mit $h(z) = log(1 + exp(-y^i z))$ für logistische Regression (ohne zusätzliche Regularisierung). Bei der Berechnung des SAG wird jedoch nicht für alle Beispiele die Ableitung berechnet. Es wird bei jeder Iteration zufällig bestimmt, für welchen Datenpunkt dies geschieht:

$$w^{k+1} = w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} v_i^k \tag{3.8}$$

mit

$$\begin{aligned} v_i^k &= \begin{cases} f_i'(w^k) & \text{für } i = i_k \\ v_i^{k-1} & \text{sonst} \end{cases} \\ &= \begin{cases} w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} h'((w^k)^T x^i) \cdot x^i & \text{für } i = i_k \\ v_i^{k-1} & \text{sonst} \end{cases} \\ &= \begin{cases} w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} log(1 + exp(-y^i \cdot ((w^k)^T x^i)) \cdot x^i & \text{für } i = i_k \\ v_i^{k-1} & \text{sonst} \end{cases} \end{aligned}$$

wobei w^k den Vektor der Koeffizienten in der k-ten Iteration, α_k die Schrittweite, n die Anzahl der Beispiele und y^i das zum Datenpunkt x^i gehörige Label beschreibt [37, 43, 49].

Die zufällige Auswahl geschieht hier durch den Parameter i_k, der für jede Iteration beliebig aus der Menge $\{1, ..., n\}$ gewählt wird. Er bestimmt, für welchen Datenpunkt die Ableitung gebildet wird. Für alle anderen Beispiele wird der vorherige Wert v_i^{k-1} verwendet. Insgesamt wird somit in jeder Iteration zunächst ein Gewicht des Koeffizientenvektors neu gesetzt und anschließend ein Schritt in Richtung des Durchschnitts der Summe aller Koeffizienten w gemacht. Dieser Schritt ist ähnlich wie in 2.3.1 beschrieben, das Annähern an das Minimum [1] [2] [49].

Zusätzlich werden weitere Parameter zur Verfügung gestellt, mit denen verschiedenste Eigenschaften der Regression verändert werden können. Zu diesen zählt auch die Auswahlmöglichkeit, l_2 Regularisierung zu verwenden, oder keine Regularisierung zu nutzen.

[1] https://www.cs.ubc.ca/~schmidtm/Courses/540-W18/L11.pdf
[2] https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L12.pdf

3.1.4 Definition. l_2-Regularisierung [21, 29, 37, 42, 49, 51]

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(x_i^T w + c)) + 1) \tag{3.9}$$

Hier beschreibt w den Koeffizienten-Vektor, C die inverse Stärke der Regularisierung, n die Anzahl der Datenpunkte, y_i mit $i \in R^m$ das jeweilige Label des Datenpunktes x_i und c eine Konstante, die den y-Achsenabschnitt bzw. den Bias des Modells bestimmt. C modifiziert dabei nicht den eigentlichen l_2 Summanden der Gleichung ($\frac{1}{2} w^T w$), sondern die Basis-Optimierungsfunktion für logistische Regression ($\log(\exp(-y_i(x_i^T w + c)) + 1)$) [9, 21, 42]. Je größer der Wert von C ist, desto größer wird der Wert, und somit der Einfluss, der Basisfunktion. Auf diese Weise hat die Regularisierung verhältnismäßig weniger Einfluss als die Basisfunktion und es wird invers zum Wert von C regularisiert. Eine alternative Darstellung, die den inversen Charakter des Parameters mehr verdeutlicht, ist die äquivalente Form $\min_{w,c} \frac{1}{2C} w^T w + \sum_{i=1}^{n} \log(\exp(-y_i(x_i^T w + c)) + 1)$ [21, 29, 37, 42, 49].

3.2 Counterfactuals (CFs)

Counterfactuals (CF) bieten dem Nutzer konkrete Handlungsempfehlungen bzw. Änderungshinweise. Sie zeigen, welche minimale Änderungen an den Eigenschaften eines Datenpunktes durchgeführt werden müssten, um eine andere Klassifizierung zu erhalten. Auf diese Weise stellen sie mögliche „Welten“ dar, die zu einer anderen Entscheidung geführt hätten [55].

Je nach Ansatz kann es sich bei CFs um Datenpunkte des eigentlichen Datensatzes (In-Sample-Counterfactuals oder In-Sample-CFs) oder um neu erzeugte Beispiele handeln. Eine Besonderheit der Verwendung von beiden Arten der Counterfactuals ist, dass diese nicht unbedingt realisierbare Werte enthalten. Beispielsweise könnte für die Genehmigung eines Kredits das Alter eine Rolle spielen. Wenn der Kredit für eine Person im Alter von 70 Jahren abgelehnt wird, könnte ein denkbarer anderer/ neuer Datenpunkt (ein Counterfactual) in allen Werten gleich bleiben und lediglich für das Alter den geänderten Wert von 35 haben. Diese Änderung führt zur Gewährung des Kredits. Somit beinhaltet dieses Counterfactual einen unerreichbaren Wert, der dennoch zugleich einen minimalen Abstand zum zu erklärenden Datenpunkt hat (siehe Abbildung 3.2). Für dieses Beispiel ist dabei die Änderung „minimal“, da nur eine einzelne Eigenschaft verändert ist. An dieser Stelle entsteht eine Unterscheidung zwischen In-Sample- und berechneten CFs: Für Ansätze, bei denen Nachbarn mit einer anderen Klassifizierung gesucht wurden, ist der nächste Datenpunkt derjenige, mit dem Alter von 35. Es ist jedoch weiterhin möglich, dass ein Alter von 40 noch immer zur Genehmigung des Kredites führen würde, es einen solchen Datenpunkt in der verwendeten Menge lediglich nicht gibt. Bei berechneten CFs bedeutet ein solcher Datenpunkt, dass ein Alter von 36 Jahren mit sonst gleichen Werten die ursprüngliche Entscheidung (Ablehnung des Kredites) nicht verändert hätte. Gleichzeitig kann es

in beiden Fällen weitere CFs geben, die gleich weit oder nur wenig weiter vom originalen Datenpunkt entfernt sind, für die das Alter keine Rolle spielt, sondern ein oder mehrere andere Eigenschaften wichtig sind [33]. Welche Eigenschaften dies im Einzelnen sind und wie genau die Nähe eines CFs zu einem Datenpunkt bestimmt wird, ist sehr stark von der eigentlichen Umsetzung abhängig.

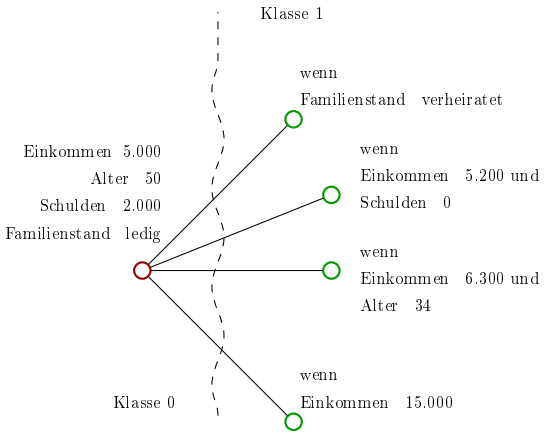


Abbildung 3.2: Es sind verschiedene Counterfactual Datenpunkte für den links angegebenen roten Punkt visuell dargestellt. Dieser stellt ein Datum dar, das vom NN als Klasse 0 (für den Kreditdatensatz z.B. der Kredit wird abgelehnt) bestimmt wurde. Die angegebenen Eigenschaften sind beispielhaft gewählt. Auch die Werte der Counterfactual-Datenpunkte (grün) stellen lediglich Beispiele dar. Bei diesen Punkten kann es sich um Daten aus dem verwendeten Datensatz handeln, es können jedoch auch neugenerierte Punkte sein. Sie alle werden vom NN als Klasse 1 (für den Kreditdatensatz z.B. der Kredit wird genehmigt) bestimmt. Die jeweils bei den Counterfactuals aufgelisteten Eigenschaftswerte stellen dar, für welches Feature des roten Datenpunktes sich Ausprägungen ändern müssen (auf die angegebenen Werte), um eine andere Klassifizierung zu erhalten. Diese Änderungen müssen nicht zwingend realistisch sein.

Counterfactuals geben somit eine Erklärung für eine Klassifizierung an und können ein hilfreiches Werkzeug sein, um schlechte Trainingsdaten, die einen gewissen Bias haben, zu erkennen. [55] Sie können jedoch nicht unbedingt realisierbare Hinweise enthalten und stellen somit nicht immer tatsächliche Handlungsempfehlungen dar. Wichtig ist hierbei die Unterscheidung von nicht realisierbaren Werten für einen konkreten Anwendungsfall gegenüber allgemein nicht möglichen Werten [29]. Das Alter von 70 auf 35 zu ändern ist nicht realisierbar. Der Wert 35 für das Alter im Allgemeinen dagegen schon. Allgemein nicht realisierbar wären hier zum Beispiel negative Werte. Diese werden für die Berechnung von Counterfactuals ausgeschlossen [29].

Für diese Arbeit werden Counterfactuals verwendet, um mit Hilfe nächster möglicher Welten eine Klassifizierung zu erklären. Hierfür werden zum einen In-Sample Beispiele bestimmt, bei denen das CF, das einen Datenpunkt erklären soll, ebenfalls aus der gege-

benen Datenmenge stammt. Zum anderen werden neue CFs berechnet, die nicht in den Eingabedaten enthalten sind.

3.2.1 In-Sample Counterfactuals

Für diese Arbeit werden In-Sample Counterfactuals mithilfe des k-Nearest-Neighbors-Algorithmus (kNN) bestimmt. Hierbei werden Datenpunkte mit geringstem Abstand zum ursprünglichen Datenpunkt ermittelt [21, 37]. Von dem kNN werden mithilfe der Euklidischen- und der Gower-Distanz die k Nachbarn berechnet und aus diesen die In-Sample-Counterfactuals bestimmt und ausgegeben. Counterfactuals sind dabei nur diejenigen Nachbarn, die eine abweichende Klassifizierung haben.

Der verwendete kNN-Algorithmus ist Teil der sklearn Bibliothek und wird dort mit verschiedenen weiteren Methoden bereitgestellt. Dabei stellt das „k“ im Namen die Anzahl der Nachbarn eines Datenpunktes, die berücksichtigt werden, dar. Für ein gegebenes k und ein Beispiel x^i werden dann die ersten k Datenpunkte, die x^i am nächsten sind, berechnet. Die Bestimmung der Abstände ist dabei abhängig von der gewählten Entfernungsmetrik. Hier wird standardmäßig die euklidische Distanz, mit $i_a, i_b \in \{1, ..., n\}$, $n \hat{=}$ Anzahl der Datenpunkte und $j \in \{1, ..., m\}$, $m \hat{=}$ Anzahl der Eigenschaften der Datenpunkte [21, 37], verwendet:

3.2.1 Definition. Euklidische Distanz

$$d(x^{i_a}, x^{i_b}) = \sqrt{\sum_j ((x_j^{i_a} - x_j^{i_b})^2))} \tag{3.10}$$

x^{i_a} und x^{i_b} sind zwei Beispiele aus der Datenmenge [21].

Alternativ steht als Abstandsmetrik auch die Gower-Distanz zur Auswahl:

3.2.2 Definition. Gower-Distanz [17]

$$S_{ij} = \frac{\sum_{k=1}^{v} s_{ijk}\delta_{ijk}}{\sum_{k=1}^{v} \delta_{ijk}} \tag{3.11}$$

Diese musste zusätzlich für den kNN konfiguriert und damit verbunden werden (siehe Kapitel 4). Hierbei bezeichnen i und j die Indizes von zwei Datenpunkten. δ_{ijk} beschreibt die Möglichkeit, die Eigenschaft k der Datenpunkte miteinander zu vergleichen. Sie ist **1**, wenn beide Punkte in Eigenschaft k miteinander vergleichbar sind, in allen anderen Fällen hat sie den Wert **0**. Für den vorhandenen Datensatz sind alle Datenpunkte immer je Eigenschaft miteinander vergleichbar, da es keinen Datenpunkt mit fehlenden Werten gibt. Die Berechnung von s_{ijk} ist davon abhängig, ob eine nominale oder ordinale Eigenschaft betrachtet wird. Für ordinale Werte gilt

$$s_{ijk} = 1 - \frac{|x^i - x^j|}{R_k} \tag{3.12}$$

wobei R_k der größte Wert des jeweiligen Wertebereichs ist [17]. Für nominale Werte ist s_{ijk} wie folgt definiert [17]:

$$s_{ijk} = \begin{cases} 1 & \text{falls } x^i = x^j \\ 0 & \text{sonst} \end{cases} \tag{3.13}$$

Eine Brute-Force-Berechnung bestimmt für jeden Datenpunkt die Distanzen zu allen anderen Beispielen und wählt anschließend aus der Matrix aller berechneten Werte für jeden Datenpunkt die k nächsten Nachbarn aus. Je mehr sich k der Anzahl aller Datenpunkte annähert, desto weniger schwer wirkt sich die Berechnung aller Entfernungen auf die Laufzeit aus (eine ausführlichere Laufzeitanalyse ist im Unterkapitel 3.4 angegeben).

Für eine Verbesserung der Performanz des Algorithmus könnte mit verschiedenen Ansätzen die Gesamtzahl der Entfernungsberechnungen verringert werden. Bei der geringen Anzahl der Datenpunkte des Kreditdatensatzes ist eine solche Optimierung nicht erforderlich [6, 56]. Für diese Arbeit reicht die Bestimmung der Distanzen der k nächsten Datenpunkte aus. Hier muss lediglich geprüft werden, welche der Nachbarn eine andere Klassifizierung als der zu erklärende Datenpunkt haben. Diese können dann als In-Sample Counterfactuals ausgegeben werden.

3.2.2 DiCE

Als zweite Methode zur Realisierung von Counterfactuals wird in dieser Arbeit der DiCE (Diverse Counterfactual Explanations) Ansatz verwendet. Dieser erstellt keine In-Sample-CFs sondern berechnet neue Datenpunkte, die nicht Teil der vorhandenen Datenmenge sein müssen. Dabei sollen die ermittelten Punkte divers sein und gleichzeitig so nah wie möglich am zu erklärenden Datenpunkt liegen [33]. Um diese mehrfache Vorgabe zu erfüllen muss im Optimierungsproblem sowohl die Diversität als auch die Nähe zum Input berücksichtigt werden. Dabei stellt das Erzeugen einer diversen Menge von Erklärungspunkten einen der größte Unterschiede von DiCE zu vielen einfachen Counterfactual-Berechnungen dar [33].

Für diese Arbeit wird ein DiCE Explainer verwendet, der eine Menge diverser CFs berechnet und diese als Tabelle ausgibt. Die Umsetzung von DiCE wird mit der zugehörigen Implementierung der interpretML-Bibliothek realisiert. Diese nutzt für die Berechnung der diversen Counterfactuals eine Erweiterung der Formel $\arg\min_c yloss(f(c), y) + |x - c|$, die ein einzelnes CF berechnet. Hierbei ist $\mathbf{c}$ das Counterfactual selbst, die Funktion *yloss* sorgt für die Berechnung eines Wertes, der sich möglichst stark in seiner Klassifizierung $\mathbf{y}$ von der des zu erklärenden Datenpunktes unterscheidet und $|\mathbf{x} - \mathbf{c}|$ sorgt für eine möglichst kleine Distanz zum ursprünglichen Beispiel [33]. Ein möglichst starker Unterschied in der Klassifizierung entsteht durch die Wahrscheinlichkeit, mit welcher der Algorithmus eine bestimmte Klasse ausgibt. Ist der originale Datenpunkt als Klasse 0 (der Kredit wird

abgelehnt) bestimmt, sollte ein Counterfactual so sicher wie möglich als Klasse 1 (der Kredit wird genehmigt) bestimmt werden.

Die Optimierung, die beim DiCE-Algorithmus vorgenommen wird, minimiert folgende Formel [33]:

$$C(x^j) = \arg\min_{c_1,...,c_k} \frac{1}{k}\sum_{i=1}^{k} yloss(f(c_i), y) + \frac{\lambda_1}{k}\sum_{i=1}^{k} dist(c_i, x^j) - \lambda_2\ dpp\_diversity(c_1, ..., c_k) \tag{3.14}$$

Hierbei beschreibt $C(x^j)$ die kombinierte Kostenfunktion, bei der über k Counterfactuals, die generiert werden sollen, optimiert wird. c_i mit $i \in \{1, ..., k\}$ bestimmt ein einzelnes CF-Beispiel. Die *yloss*-Funktion minimiert den Abstand der Vorhersage $f(c_i)$ des Netzes zur gewünschten Klassifizierung y (diejenige Klassifizierung die nicht dem Label des Datenpunktes x^j entspricht). Mit der *dist*-Funktion wird die Entfernung von Counterfactual c_i und dem ursprünglichen Datenpunkt x^j bestimmt. Die Diversität der einzelnen CFs wird durch den Summanden $dpp\_diversity(c_1, ..., c_k) = \det(K)$ mit $K_{ij} = \frac{1}{1+dist(c_i,c_j)}$ ausgedrückt, wobei *dist* wieder die Entfernung bestimmt, hier jedoch zwischen zwei CFs c_i und c_j. Dabei wird durch die Berechnung von K_{ij} sichergestellt, dass sich alle Werte im Bereich $[0, 1]$ befinden. So ergibt sich für exakt identische Datenpunkte eine Determinante von 0, für sehr verschiedene eine Determinante von 1. Auf diese Weise wird die Diversität beschrieben, bleibt jedoch in einem vorgegebenen Rahmen und es wird verhindert, dass sie durch ihre Berechnung stärker gewichtet wird, je diverser die Punkte sind. Bei λ_1 und λ_2 handelt es sich um Hyperparameter, mit denen die Gewichtung von *yloss*, *dist* und *dpp_diversity* angepasst werden kann [33].

Für den Algorithmus selbst sind die Wahl dieser drei Funktionen ausschlaggebend. Die in der Implementierung verwendete Definition für *dpp_diversity*, den determinantal point processes (DPP), wurde bereits angegeben. Diese Funktion bestimmt die Determinante einer Distanzenmatrix, in deren Zellen jeweils die Abstände von zwei CFs zueinander berechnet wurden [33].

Die Wahl der *yloss*-Funktion basiert auf der Erfüllung mehrerer Voraussetzungen: Wenn die gewünschte Klassifizierung y **1** (der Kredit wird genehmigt) ist und die Ausgabe des NN $f(c)$ den Threshold von 0.5 überschreitet, also auch die Klasse **1** vorhersagt, soll die Entfernung des *Vorhersage-Wertes* vom Wert 1 nicht bestraft werden (analog soll, falls die Klassifizierung **0** (der Kredit wird abgelehnt) gewünscht ist und der Threshold von 0.5 nicht überschritten wird, auch die Entfernung der Vorhersage von 0 nicht bestraft werden). Konkret bedeutet das: Wird eine Klassifizierung von 1 gefordert und ein Wert von 0.7 errechnet, ist die Differenz dieser Werte „egal“. Somit wird verhindert, dass Datenpunkte bestimmt werden, die besonders nah an einer der Klassifizierungen (0 oder 1) liegen.

Gleichzeitig soll eine proportionale Bestrafung in Form des Abstandes von $f(c)$ zu 0.5 vergeben werden, wenn die Vorhersage dem gewünschten y entspricht, jedoch eine stärkere

Bestrafung verwenden, wenn die Vorhersage nicht dem gewünschten y entspricht. Diese Bedingungen erfüllt die Hinge-Loss-Funktion:

3.2.3 Definition. Hinge-Loss [33]

$$Hinge - Loss = \max(0, 1 - z \cdot logit(f(c))) \tag{3.15}$$

Hierbei wird

$$z = \begin{cases} -1 & \text{wenn } y = 0 \\ 1 & \text{wenn } y = 1 \end{cases} \tag{3.16}$$

gewählt. $logit(f(c))$ gibt die unskalierte Ausgabe des NN an [33].

Bei der Distanzfunktion muss, wie für die Gower-Distanz, zwischen nominalen und ordinalen Eigenschaften unterschieden werden. Entfernungen ordinaler Werte werden mithilfe der Manhattan-Distanz zwischen dem CF-Beispiel und dem originalen Datenpunkt berechnet [33]:

$$dist\_cont(c^{i_a}, x^{i_b}) = \frac{1}{d_{cont}} \sum_{j=1}^{d_{cont}} \frac{|c_j^{i_a} - x_j^{i_b}|}{MAD_j} \tag{3.17}$$

Hierbei gibt d_{cont} die Anzahl der ordinalen (continuous) Feature an, c^{i_a} ist ein bestimmtes Counterfactual und x^{i_b} ist der zu erklärende Datenpunkt. MAD_j ist die mediane absolute Abweichung (median absolute deviation) der Werte für das jeweilige Feature j. $MAD_j = \frac{1}{k} \sum_{i=1}^{k} |c_j^i - x_j^{i_b}|$, wobei k die Anzahl der CFs bestimmt [33].

Nominale Daten erhalten ein Distanzmaß, bei dem ein Abstand von 1 immer dann zugewiesen wird, wenn sich die Ausprägung eines kategorischen Features im CF von der des Datenpunktes unterscheidet (0, wenn sie gleich sind).

$$dist\_cat(c^{i_a}, x^{i_b}) = \frac{1}{d_{cat}} \sum_{j=1}^{d_{cat}} I(c_j^{i_a} \neq x_j^{i_b}) \tag{3.18}$$

d_{cat} gibt die Anzahl der nominalen (categorical) Feature an [33], I ist die Indikatorfunktion (siehe Definition 3.3).

Die Optimierung der kombinierten Loss-Funktion 3.14 wird dann mit dem Verfahren des Gradientenabstiegs (siehe 2.3.1) berechnet und bestimmt so diejenigen k Counterfactuals, die gleichzeitig möglichst divers sind und nah am zu erklärenden Datenpunkt liegen [33].

3.3 Feature Contribution

Für die Erklärung durch Feature Contribution werden im Rahmen dieser Arbeit die beiden Methoden LRP und DeepSHAP verwendet. Bei diesen handelt es sich um Algorithmen, die ein Neuronales Netz rückwärts, also entgegen der Klassifizierungsrichtung, durchlaufen

und anhand der gelernten Gewichte bzw. der Neuroneninputs eine Erklärung der Netz-Eingabe erzeugen [36, 52]. Diese Eingabe, also die Neuronen der Input-Schicht, entspricht den jeweiligen Eigenschaften der Datenpunkte.

Für die gegebenen Kredit-Daten wird die Erklärung mit Hilfe eines Balkendiagramms dargestellt. Durch die Höhe der Balken wird die Wichtigkeit der bestimmten Eigenschaften für die Klassifizierung angegeben. Positive Werte zeigen, dass die Eigenschaft eine Klassifizierung als Klasse 1 (der Kredit wird genehmigt) unterstützt, negative Werte sind eine Beeinträchtigung für eine Entscheidung für die gleiche Klasse. Analog dazu, jedoch mit umgekehrtem Vorzeichen, wird die Klasse 0 (der Kredit wird abgelehnt) beeinflusst. Dies entspricht der allgemeinen Grundidee der Feature Contribution: zur Erklärung Wichtigkeiten (oder auch Gewichte) pro Eingabedimension auszugeben. Diese Gewichte geben für jeden konkreten Datenpunkt an, wie stark die entsprechende Belegung für bzw. gegen die Entscheidung des Modells spricht [32]. Hierfür erhalten die Neuronen der Eingabeschicht, die den Eigenschaften der Datenpunkte entsprechen, diese Gewichte „zugewiesen“. Aus diesen Werten kann dann ein Balkendiagramm erstellt werden, das visualisiert, wie wichtig (positiv oder auch negativ) eine Eigenschaft für die Entscheidung des NN war.

3.3.1 DeepSHAP

Im Bereich der Feature Contriubtion wird für diese Arbeit der DeepSHAP Algorithmus [3] verwendet. Der SHAP (SHapley Additive exPlanations) Deep Explainer [24] basiert auf dem DeepLIFT (Deep Learing Important FeaTures) Algorithmus [50].

DeepLIFT erklärt die Beteiligung einzelner Neuronen, gemessen an ihren Inputs. Um diesen Beteiligungswert errechnen zu können werden für jeden Knoten Attributregeln aufgestellt, die sich aus der Aktivierung auf dem Referenzinput ergeben. Mithilfe dieser Regeln und der Verbindung mit der Kettenregel [50] lassen sich die Beteiligungen mittels Backpropagation (siehe 2.3.2) bestimmen. [50]

In dieser Arbeit wird für die Umsetzung des Deep Explainers die SHAP Bibliothek verwendet [4]. Diese stellt die Implementation eines Algorithmus bereit, der DeepSHAP auf neuronalen Netzen anwendet und der für die Erklärung von Klassifizierungen genutzt werden kann [5].

Für DeepSHAP wird eine Kombination von DeepLIFT und den Shapley Values verwendet, um eine verbesserte Laufzeit zu erreichen. Diese Beschleunigung entsteht, wenn aus den DeepLIFT Attribut-Regeln Approximationen von Shapley Values bestimmt werden.

[3] `https://shap.readthedocs.io/en/latest/generated/shap.DeepExplainer.html#shap.DeepExplainer`

[4] `https://shap.readthedocs.io/en/latest/index.html`

[5] `https://shap.readthedocs.io/en/latest/generated/shap.DeepExplainer.html`

Die für jeden Knoten bestimmten Attribut-Regeln können dabei so gewählt werden, dass sie eine passende Abschätzung bilden. Durch das Aggregieren vieler Hintergrundbeispiele wird eine Annäherung ermittelt, sodass die Summe der SHAP Values die Differenz zwischen dem erwarteten ($E(f(x^i))$) und dem aktuellen ($f(x^i)$) Output des Modells ergeben: $f(x^i) - E(f(x^i))$ [24].

Shapley Values sind ein Lösungskonzept, das für die Spieltheorie entwickelt wurde. Bei diesem werden alle Feature als „Spieler“, die Vorhersagen oder Klassifizierungen als „Gewinn“ angesehen. Aus dem Zusammenspiel der einzelnen Variablen kann dann die Beteiligung am Output errechnet und somit eine Erklärung für Klassifizierungen bestimmt werden [29].

Es werden zunächst die Schritte und Grundlagen von DeepLIFT und anschließend die Kombination der beiden Algorithmen erläutert: DeepLIFT berechnet die Beteiligung von Neuronen a_i^k an einem Output y gemessen an den jeweiligen Neuroneninputs. i und k indizieren hierbei das entsprechende Neuron im Netz (siehe Abbildung 2.2). Dafür werden folgende Schritte durchgeführt:

1. Für alle Neuronen des Netzes werden Referenzwerte r_i^k angegeben

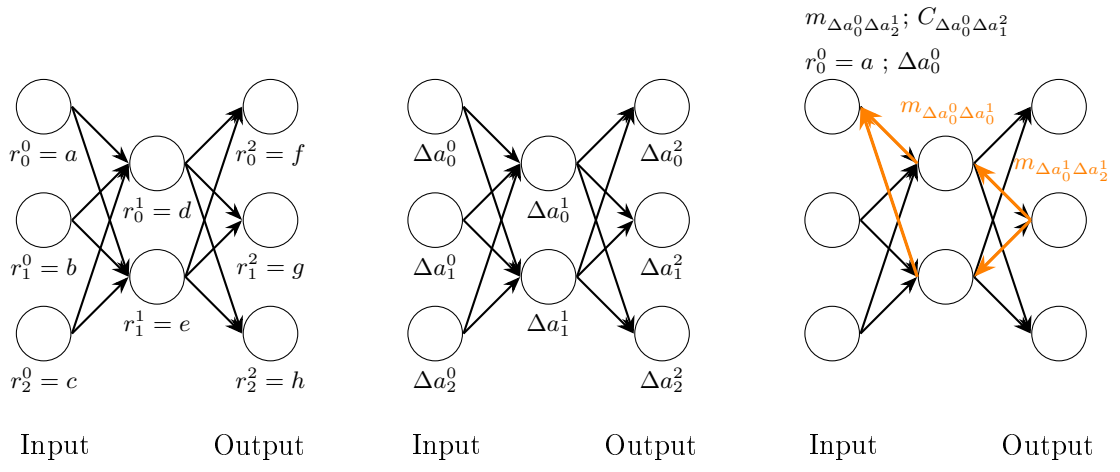
2. Für den zu erklärenden Datenpunkt x^n werden dann die Neuronenaktivierungen a_i^k und aus diesen die Differenzen $\Delta a_i^k = a_i^k - r_i^k$ berechnet

3. Für jede Schicht wird, abhängig von der Art der Schicht, die Contribution-Zuweisungsregel bestimmt (linear: LinearRule, nicht-linear: RescaleRule, siehe Definitionen 3.3.2, 3.3.3)

4. Von der Output bis hin zur Input-Schicht werden alle Multiplizierer für direkt verbundene Neuronen berechnet

5. Mit der Kettenregel werden zunächst die kombinierten Multiplizierer für die jeweiligen Input-Neuronen bestimmt und aus diesen dann die entsprechenden Contributions berechnet

[50]



(a) Schritt 1 **(b)** Schritt 2 **(c)** Schritt 3-5

Abbildung 3.3: Hier ist der Ablauf des DeepLIFT Algorithmus grob dargestellt. In der linken Abbildung (Schritt 1) ist das Wählen der Referenzwerte dargestellt. Schritt 2 (in der Mitte) visualisiert die berechneten Differenzen der Neuronenaktivierung zur Referenz. Rechts sind die Berechnungen der Multiplizierer beispielhaft für einen Pfad vom oberen Input zum mittleren Output visualisiert. Alle orangenen Kanten ergeben summiert den gesamten Multiplizierer $m_{\Delta a_0^0 \Delta a_1^2}$, aus dem dann die Beteiligung $C_{\Delta a_0^0 \Delta a_1^2}$ von Neuron a_0^0 an Ausgabe a_1^2 berechnet werden kann.

Für eine verbesserte Lesbarkeit der Regeln werden im Folgenden Neuronen mit x bzw. y und Zielneuronen mit t (target) bezeichnet. Bei konkretem Bezug auf die Visualisierungen wird die entsprechende, dort verwendete, Notation genutzt. Ein Multiplizierer ist hierbei von der Form $m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x}$ und erfüllt für alle Kanten j, die das Neuron x_i mit Neuronen y_j höherer Schichten (näher am Output) verbinden, die Kettenregel:

3.3.1 Definition. Kettenregel [50]

$$m_{\Delta x_i \Delta t} = \sum_j m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta t} \tag{3.19}$$

Diese ermöglicht das Rückpropagieren der Multiplizierer durch das Netz [50].

Abbildung 3.3 zeigt beispielhaft den Ablauf des Algorithmus. In Schritt 1 (3.3a) ist das Festlegen von Referenzen für jedes Neuron visualisiert. Schritt 2 (3.3b) zeigt die berechneten Differenzen. Die Berechnungsschritte der Multiplizierer mit einem Beispiel für die Verwendung der LinearRule für die Ausgabe- und versteckte Schicht sind in 3.3c durch die orangenen Pfeile in Rückrichtung für $m_{\Delta a_0^0 \Delta a_2^1}$ und damit anschließend auch für $C_{\Delta a_0^0 \Delta a_2^1}$ verbildlicht [50].

Welche Regel für die Zuweisung von Contributions $C_{\Delta x \Delta t}$ verwendet wird, hängt von der Art der Schicht ab. Für lineare Schichten, wie zum Beispiel eine Lineare- oder Convolution-Schicht kann die LinearRule verwendet werden [50].

Hierbei wird von einer linearen Aktivierungsfunktion σ (siehe 2.2.2) mit

$$y = b + \sum_i w_i x_i$$

und damit $\Delta y = \sum_i w_i \Delta x_i$ ausgegangen (bei Berechnung der Differenzen heben sich die Biaswerte b auf) [50].

Um zwischen positiven und negativen Anteilen unterscheiden zu können, werden $\Delta y = \Delta y^+ + \Delta y^-$, $\Delta s = \Delta x^+ + \Delta x^-$ und $C_{\Delta x \Delta t} = C_{\Delta y^+ \Delta t} + C_{\Delta y^- \Delta t}$ jeweils als Summe dieser beiden Anteile definiert. Aus diesen können Δy^+ bzw. Δy^- weiterhin als

$$\begin{aligned} \Delta y^+ &= \sum_i 1\{w_i \Delta x_i > 0\} w_i \Delta x_i & \Delta y^- &= \sum_i 1\{w_i \Delta x_i < 0\} w_i \Delta x_i \\ &= \sum_i 1\{w_i \Delta x_i > 0\} w_i (\Delta x_i^+ \Delta x_i^-) & &= \sum_i 1\{w_i \Delta x_i > 0\} w_i (\Delta x_i^+ \Delta x_i^-) \end{aligned}$$

definiert werden und hängen nun nur noch von bereits bekannten Werten w_i (den Gewichten) und x_i (den Neuronenoutputs) ab [50]. $1\{w_i \Delta x_i > 0\}$ bedeutet hierbei, die Multiplikation von $w_i \Delta x_i$ mit 1 (der Wert wird berücksichtigt), wenn die Bedingung $w_i \Delta x_i > 0$ erfüllt ist, bzw. die Multiplikation mit 0 (der Wert wird nicht berücksichtigt), wenn sie nicht erfüllt ist. Dies gilt analog für die anderen drei Bedingungen. Aus diesen Definitionen folgen für C und m [50]:

$$\begin{aligned} C_{\Delta x_i^+ \Delta y^+} &= 1\{w_i \Delta x_i > 0\} w_i \Delta x_i^+ & C_{\Delta x_i^+ \Delta y^-} &= 1\{w_i \Delta x_i < 0\} w_i \Delta x_i^+ \\ C_{\Delta x_i^- \Delta y^+} &= 1\{w_i \Delta x_i > 0\} w_i \Delta x_i^- & C_{\Delta x_i^- \Delta y^-} &= 1\{w_i \Delta x_i < 0\} w_i \Delta x_i^- \end{aligned}$$

3.3.2 Definition. LinearRule [50]

$$\begin{aligned} m_{\Delta x_i^+ \Delta y^+} &= m_{\Delta x_i^- \Delta y^+} \\ &= 1\{w_i \Delta x_i > 0\} w_i \frac{\Delta x_i}{\Delta x_i} \\ &= 1\{w_i \Delta x_i > 0\} w_i \end{aligned} \tag{3.20}$$

$$\begin{aligned} m_{\Delta x_i^+ \Delta y^-} &= m_{\Delta x_i^- \Delta y^+} \\ &= 1\{w_i \Delta x_i < 0\} w_i \frac{\Delta x_i}{\Delta x_i} \\ &= 1\{w_i \Delta x_i < 0\} w_i \end{aligned} \tag{3.21}$$

Für Schichten mit nicht-linearen Aktivierungsfunktionen, wie z.B. der ReLU-Funktion, wird die RescaleRule genutzt. Diese geht von einer Funktion $y = f(x)$ mit nur einem Input aus, die eine nichtlineare Transformation durchführt. Aus der „Summation-to-delta"-Eigenschaft $\Delta t = \sum_{i=1}^{n} C_{\Delta x_i \Delta t}$ ergibt sich in diesem Fall $\Delta y = C_{\Delta x_i \Delta t}$ [50].

Ein Einsetzen in die Definition für Multiplizierer ergibt sofort $m_{\Delta x \Delta y} = \frac{C \Delta y}{\Delta x}$. Mit der definierten Aufteilung in positive und negative Anteile folgen:

$$\Delta y^+ = \frac{\Delta y}{\Delta x}\Delta x^+ \qquad \Delta y^- = \frac{\Delta y}{\Delta x}\Delta x^-$$
$$= C_{\Delta x^+ \Delta y^+} \qquad = C_{\Delta x^- \Delta y^-}$$

Daraus kann wiederum die RescaleRule definiert werden:

3.3.3 Definition. RescaleRule [50]

$$m_{\Delta x^+ \Delta y^+} = m_{\Delta x^- \Delta y^-} = m_{\Delta x \Delta y} = \frac{\Delta x}{\Delta y} \tag{3.22}$$

Die wesentlichen Grundlagen des DeepLIFT Algorithmus sind durch die vorgestellten Gleichungen und Definitionen beschrieben. Für die Kombination fehlt die Definition der Shapley Values und eine anschließende Verbindung zu DeepSHAP [24]. Die Formel zur Bestimmung von Shapley Values setzt sich aus mehreren Komponenten zusammen:

3.3.4 Definition. Shapley Values

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!}[f_x(z') - f(z' \backslash i)] \tag{3.23}$$

Dabei wird als Input eines Erklärungsmodells ein sogenannter „simplified Input“ [24] x' verwendet. Bei diesem handelt es sich um einen Vektor, der eine vereinfachte Darstellung der originalen Werte ist. Eine Vereinfachung entsteht dabei, indem bestimmte Werte des originalen Inputs durch 0 ersetzt, also ausgelassen, werden. Welche Werte dies sind ist dabei anwendungsspezifisch. Mit einer spezifizierten Funktion $h_x(x') = x$ muss dieser vereinfachte Vektor jedoch wieder auf den originalen Datenpunkt abgebildet werden können. Die Vereinfachung wird hier also durch das Setzen bestimmter Eigenschaften als „fehlend“ mit dem Wert 0 beschrieben. So haben sie keinen Einfluss auf die Berechnung der Shapley Values [24]. $z' \in \{0, 1\}^M$ beschreibt nun diejenigen Vektoren, in denen alle Eigenschaften, deren zugehörige Einträge einen Wert ungleich 0 besitzen, eine Teilmenge der Eigenschaften von x' mit der gleichen Bedingung sind. x' gibt dabei mit allen nicht-0-Werten diejenigen Eigenschaften an, die für die Berechnung betrachtet werden. z' bildet die Menge aller Vektoren, die davon eine Teilmenge darstellen.

Zum Beispiel könnte es im originalen Modell die fünf Eigenschaften Alter, Beziehungsstatus, Einkommen, Bildung und Anzahl Kinder geben. Von diesen sollen jedoch nur die zwei Feature Alter und Einkommen betrachtet werden. In einem vereinfachten Vektor für den Datenpunkt $x_1 = (25, \mathit{ledig}, 2000, \mathit{Abitur}, 1)^T$ ergibt sich $x' = (25, -, 2000, -, -)$. Da die Werte im Beispiel einfacher lesbar sind, wenn sie nicht codiert dargestellt werden, wird hier ein Querstrich eingesetzt für das Ausschließen von Eigenschaften verwendet. In z' sind nun neben dem Vektor x' selbst alle Teilmengen dieses Vektors enthalten, die sich durch

entfernen eines oder mehrerer weiterer Feature ergeben:
$z' = \{(25, -, 2000, -, -), (-, -, 2000, -, -), (25, -, -, -, -), (-, -, -, -, -)\}$.

Für diese Vektoren werden dann die jeweiligen Ergebnisse der Shapley Values (siehe Formel 3.23) entsprechend berechnet. M beschreibt die Gesamtzahl der Feature und damit die Größe von z', $|z'|$ die Anzahl der nicht-0-Einträge von z' (im Beispiel die Einträge, die nicht durch Querstriche ersetzt wurden). Die Funktion f ist die zu erklärende Blackbox und wird hier verwendet, um einmal die Ausgabe für ganz z' und einmal für $z' \backslash i$ (also z' ohne die Eigenschaft i) zu berechnen und deren Differenz zu bestimmen. Insgesamt wird so durch die Summierung über alle möglichen Teilmengen von x' ein gewichtetes Mittel über alle Differenzen ausgegeben [24]. Die Shapley Values beschreiben also mit ϕ_i die Beteiligung von Eigenschaft i an der Vorhersage des Netzes im Vergleich zur mittleren Vorhersage des NN.

Um den abschließenden Schritt der Kombination ausführen zu können fehlt eine letzte wichtige Definition: Es gilt $f(h_x(z')) = f_x(z') = E[f(z)|z_S]$, wobei S die Menge aller nicht-0-Indizes von z' angibt und in z_S somit alle Variablen „fehlen“, die nicht in S enthalten sind. Die SHAP Werte geben für jede Eigenschaft an, um wie viel sie die erwartete Vorhersage des Modells verändern. Die Grafik 3.4 visualisiert für einen konkreten Wert für i, wie durch die einzelnen Ergebnisse für ϕ_i von $E[f(z)]$ (dem eigentlichen Erwartungswert) zu $f(x)$ (der Ausgabe des NN) gelangt werden kann. Ein allgemeines Ergebnis entsteht durch das Bilden eines Mittelwerts aller möglichen Kombinationen und Anordnungen [24].

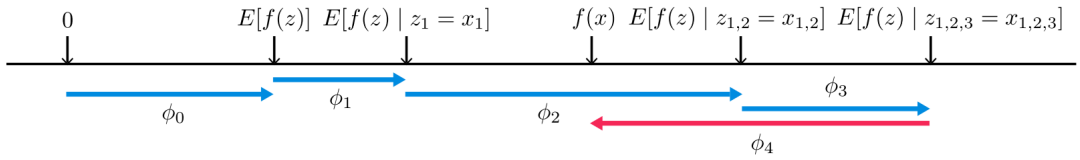


Abbildung 3.4: Es ist eine Verbildlichung der Shapley Values für eine mögliche Kombination von Eigenschaften und deren Auswahlreihenfolge visualisiert. Hierbei wird durch ϕ jeweils die Beteiligung des zugehörigen Features an der Abweichung vom Erwartungswert der Vorhersage angegeben. ϕ_0 beschreibt hierbei $E[f(z)]$ ohne die Wahl einer bestimmten Eigenschaft. Für alle folgenden ϕ_i wird jeweils eine zusätzliche Eigenschaft gewählt und aus der Berechnung ausgeschlossen. Wurden alle Eigenschaften entfernt, wird der Wert $f(x)$ erreicht. [24]

$\phi_i(f, x)$ wird für die Berechnung des eigentlichen DeepSHAP Algorithmus über die Contributions $C_{\Delta x_j \Delta y}$ approximiert. Hierfür muss jedoch für die Referenzwerte der Erwartungswert $E[x_j]$ des jeweiligen Neurons verwendet werden, woraus sich dann $m_{\Delta x_j \Delta y} = \frac{C_{\Delta x_j \Delta y}}{\Delta x_j} = \frac{C_{\Delta x_j \Delta y}}{x_j - E[x_j]}$ ergibt [24].

Dies führt zur Approximation und somit zur Kombination beider Ansätze:

$$\phi_j(y, x) \approx m_{\Delta x \Delta y}(x_j - E[x_j]) = C_{\Delta x_j \Delta y} \tag{3.24}$$

Ein Beispiel dafür ist im Paper [24] (Kapitel 4.2, Abbildung 2, verbunden mit den zugehörigen Gleichungen 13-16) zu finden.

3.3.2 Layerwise Relevance Propagation (LRP)

Als zweite Methode im Bereich Feature Contribution wird Layerwise Relevance Propagation (LPR) implementiert. Für LRP werden sogenannte Relevanzen berechnet und rückwärts durch das Netz propagiert. Da diese in der Summe für jede Schicht gleich bleiben, kann für alle Layer eine Erklärung durch die Relevanzen angegeben werden. Die Ergebnisse können als Einfluss des jeweiligen Neurons auf den entsprechenden Output gesehen werden.

Im Rahmen dieser Arbeit wird LRP anhand eines Tutorials selbst implementiert. Bei diesem Tutorial handelt es sich um das zum Paper von Montavon et al. [32] Zugehörige. Darin wird der notwendige Code für Bilddaten beschrieben [6].

Für die Verwendung von LRP werden im Folgenden Relevanzneuronen erstellt und alle Relevanzwerte entgegen der Klassifizierungsrichtung durch das Netz propagiert. Diese Relevanzneuronen sind mit R_k bezeichnet und repräsentieren in der Ausgabeschicht zunächst die tatsächliche Neuronenaktivierung a_k [32].

Das Vorgehen ist ähnlich wie bei der Backpropagation im Lernprozess (siehe 2.3.2). Den Startpunkt stellen ebenfalls die Output-Neuronen dar, deren Wert hier jedoch zugleich die Klassifikation sowie die jeweilige Relevanz in der Ausgabeschicht beschreibt. Von diesen Neuronen aus muss die Relevanz zurück zur Eingabeschicht propagiert werden. Dafür wird das Netz in seine relevanten Teile aufgebrochen und die vorhandene Struktur ausgenutzt. Ähnlich der Backpropagation werden über die Gewichtskanten rückwärts durch das Netz die jeweiligen Relevanz-Werte errechnet. Bei der Relevanzpropagation werden jedoch keine Gewichte mehr verändert. Stattdessen sollen Neuronen mit einer hohen Relevanz R_k, also hohen Inputs a_j und hohen Gewichten w_{jk}, identifiziert werden [32].

[6] `http://heatmapping.org/tutorial/`

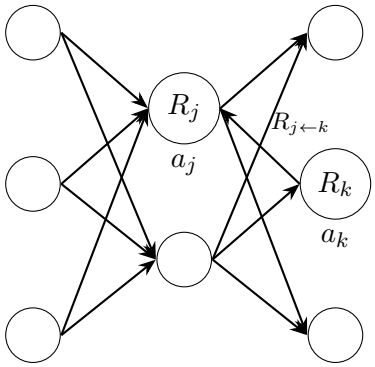


Abbildung 3.5: Hier wird beispielhaft für zwei konkrete Neuronen dargestellt, wie Relevanzen durch das Netz propagiert werden. Dabei wird in der Ausgabeschicht R_k über die Ausgabe selbst bestimmt. Diese Relevanz wird mit der entsprechend verwendeten LRP-Regel eine Schicht nach vorn zu R_j propagiert.

Für die eigentliche Berechnung der Relevanzen können zahlreiche verschiedene Methoden verwendet werden. Es gibt nicht eine Grundregel, die für alles verwendet wird, sondern vielmehr eine Sammlung verschiedener LRP-Regeln, die spezifisch für bestimmte Anwendungen sind. In dieser Arbeit werden dem Nutzer LRP-0, LRP-γ und LRPϵ zur Auswahl bereitgestellt. Diese sind alle für die gegebenen Daten geeignet (beispielsweise im Gegensatz zur z^B-Regel, die sich eher zur Berechnung von Pixel-Werten eignet) [30, 32].

Die zur Wahl angebotenen Regeln sind der LRP$\alpha_1 - \beta_0$-Regel (3.29) sehr ähnlich, beziehungsweise können aus dieser abgeleitet werden:

Die LRP-0-Regel ist dabei gewissermaßen die „Basis“ Regel. Hier wird das Gewicht w_{jk} als Ganzes und nicht aufgeteilt in positive und negative Anteile mit einbezogen.

3.3.5 Definition. LRP-0-Regel [30]

$$R_j = \sum_k \left(\frac{a_j w_{jk}}{\sum\limits_j a_j w_{jk}} R_k \right) \tag{3.25}$$

Bei der LRP-γ-Regel wird nun zusätzlich der positive Anteil w_{jk}^+, der mit einem festen Vorfaktor γ multipliziert wird, auf das Gewicht w_{jk} aufsummiert. So wird der Einfluss der positiven Anteile der Gewichte erhöht. Damit wird ein stabileres Ergebnis erzeugt.

3.3.6 Definition. LRP-γ-Regel [30]

$$R_j = \sum_k \left(\frac{a_j w_{jk} + \gamma w_{jk}^+}{\sum\limits_j a_j (w_{jk} + \gamma w_{jk}^+)} R_k \right) \tag{3.26}$$

Die LRPϵ-Regel verwendet ebenfalls das Gewicht als Ganzes, vergrößert aber den Zähler des Quotienten um einen festen Wert ϵ. Es sorgt dafür, dass ein Teil der Relevanzen „absorbiert“ werden. Auf diese Weise verringert es den Einfluss von kleinen Störungen. Gleichzeitig werden die Erklärungen damit weniger zahlreich.

3.3.7 Definition. LRP-ϵ-Regel [30]

$$R_j = \sum_k \left(\frac{a_j w_{jk}}{\epsilon + \sum_j a_j w_{jk}} R_k \right) \tag{3.27}$$

Diese angebotenen Regeln können nicht für die Eingabeschicht verwendet werden, da sie auf Neuroneninputs einer vorherigen Schicht (näher am Input-Layer) basieren, die für die erste Schicht nicht vorhanden sind. Daher wird für diese Schicht die w^2-Regel genutzt. Bei der w^2-Regel werden die Neuroneninputs a_j nicht mehr miteinbezogen und stattdessen die Gewichte w_{jk} quadriert.

3.3.8 Definition. LRP-w^2-Regel

$$R_j = \sum_k \left(\frac{w_{jk}^2}{\sum_j w_{jk}^2} R_k \right) \tag{3.28}$$

[30]

Für die zuvor erwähnte LRP-$\alpha_1 - \beta_0$-Regel kann (im Gegensatz zu den anderen, einfacheren Regeln) eine Verbindung zur Deep Taylor Decomposition hergestellt werden [32]. Diese ist nötig, um die durchgeführten Schritte und das Funktionieren der Relevanzpropagation zu begründen.

3.3.9 Definition. LRP-$\alpha - \beta$-Regel [32]

$$R_j = \sum_k \left(\alpha \frac{a_j w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k - \beta \frac{a_j w_{jk}^-}{\sum_j a_j w_{jk}^-} R_k \right) \tag{3.29}$$

Ein Neuron sei definiert als

3.3.10 Definition. LRP-Neuron

$$a_k = \max(0, \sum_j a_j w_{jk} + b_k), \text{ mit } b_k \geq 0 \tag{3.30}$$

und die Relevanz eines Neurons a_k gegeben durch

$$\begin{aligned} R_k &= a_k c_k \\ &= \max(0, \sum_j a_j w'_{jk} + b'_k) \\ &= \max(0, \sum_j a_j w_{jk} \cdot c_k + b_k \cdot c_k) \\ &= \max(0, \sum_j a_j w_{jk} + b_k) \cdot c_k \end{aligned} \tag{3.31}$$

Zudem sei c_k hierbei konstant und positiv [32].

Mithilfe der Deep Taylor Decomposition kann aus dieser Relevanz R_k die Relevanz R_j für Neuronen der tieferen Schichten bestimmt werden. Hierfür muss zunächst der nächste Entwicklungspunkt $(\tilde{a}_j)_j$ bestimmt werden, für den $f(\tilde{a}_j) = 0$ gilt. Dieser stammt aus dem Segment $[(a_j 1_{w'_{jk} \leq 0})_j, (a_j)_j]$ und sorgt dafür, dass die Deep Taylor Decomposition nur Terme erster Ordnung enthält. Somit können in R_k alle Terme mit allen a_j, nach denen nicht abgeleitet wird, auf 0 gesetzt werden. R_k lässt sich also beschreiben als

$$R_k = \sum_j \underbrace{\frac{\partial R_k}{\partial a_j}\Big|_{(\tilde{a}_j)_j} \cdot (a_j - \tilde{a}_j)}_{R_{j \leftarrow k}} \tag{3.32}$$

wobei die Dekomposition von R_k für diesen Fall eine geschlossene Form hat:

$$R_{j \leftarrow k} = \frac{a_j w_{jk}^+}{\sum\limits_j a_j w_{jk}^+} R_k \tag{3.33}$$

Aus der LRP $\alpha_1 - \beta_0$-Regel, für die in der Basis- $\alpha - \beta$-Regel (Gleichung 3.29) $\alpha = 1$ und $\beta = 0$ gesetzt werden ($R_j = \sum\limits_k (1 \cdot \frac{a_j w_{jk}^+}{\sum\limits_j a_j w_{jk}^+} R_k - 0 \cdot \frac{a_j w_{jk}^-}{\sum\limits_j a_j w_{jk}^-} R_k) = \sum\limits_k (\frac{a_j w_{jk}^+}{\sum\limits_j a_j w_{jk}^+} R_k)$) kann nun die identische Gleichung erhalten werden, die durch Summierung über alle R_k der nachfolgenden Schicht ebenfalls erhalten werden:

$$\begin{aligned} R_j &= \sum_k R_{j \leftarrow k} \\ &= \sum_k \left(\frac{a_j w_{jk}^+}{\sum\limits_j a_j w_{jk}^+} R_k \right) \end{aligned} \tag{3.34}$$

Eine detaillierte Herleitung ist im Paper von Montavon et al. [31] zu finden.

Somit ist also der erste Schritt der Begründung anhand der Deep Taylor Decomposition, der den Aufbau der LRP$\alpha_1 - \beta_0$-Regel ergibt, erfolgreich. Es bleibt zu zeigen, dass diese Form auch auf näher am Input liegende Schichten übertragbar ist. Für alle Neuronen R_j

muss eine Form $R_j = a_j c_j$ erstellbar sein, bei der c_j quasi-konstant und positiv ist. Es gilt

$$\begin{aligned} c_j &= \sum_k \left(\frac{w_{jk}^+}{\sum_j a_j w_{jk}^+} R_k \right) \\ &= \sum_k \left(\frac{w_{jk}^+}{\sum_j a_j w_{jk}^+} \max(0, \sum_j a_j w'_{jk} + b'_k) \right) \\ &= \sum_k \left(\frac{w_{jk}^+}{\sum_j a_j w_{jk}^+} \max(0, \sum_j a_j w_{jk} \cdot c_k + b_k \cdot c_k) \right) \\ &= \sum_k \left(\frac{w_{jk}^+}{\sum_j a_j w_{jk}^+} \max(0, \sum_j a_j w_{jk} + b_k) \cdot c_k \right) \end{aligned} \tag{3.35}$$

[32]

c_j ist somit ausschließlich von w_{jk} bzw. w_{jk}^+, b_k, c_k und über zwei verschachtelte Summen von a_j abhängig. w_{jk} und w_{jk}^+ sowie b_k sind konstant. Laut Definition ist außerdem w_{jk}^+ positiv. Auch für den Zähler des Bruchs ergibt sich durch die Maximums-Funktion ein Wert größer oder gleich 0, da nur positive Ergebnisse von $\sum_j a_j w_{jk} + b_k$ in Betracht gezogen werden. Für ReLU-Netze ist die Neuronenaktivierung und damit der Nenner des Bruchs ebenfalls immer positiv. Somit muss lediglich für c_k analysiert werden, ob es (quasi-) konstant und positiv ist [30, 32].

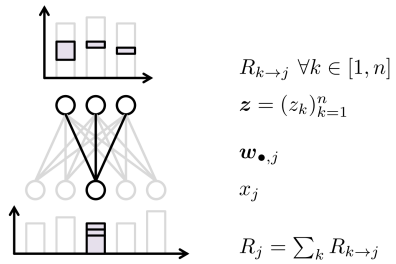
Für die Ausgabeschicht gilt dies per Definition von c_k. Für alle vorhergehenden Schichten folgt somit, dass das entsprechende c_k quasi-konstant und positiv ist und somit auch in $R_j = a_j c_j$ diese Bedingung für c_j erfüllt ist. Somit ist die Form auf alle Paare aufeinanderfolgender Schichten übertragbar und anwendbar. Es folgt, dass die Relevanzen für das gesamte Netz mit dieser Formel bestimmt werden können [30, 32].

Eine essenzielle Anforderung an die Relevanzen ist die Erfüllung der Relevance Conservation Property (Relevanzerhaltung), die dafür sorgt, dass eine aussagekräftige Ausgabe erhalten werden kann, von der durch das Propagieren nichts „verloren“ geht [32]. Dabei kann die Relevanzerhaltung ähnlich wie die Spannungserhaltung in einem Stromnetz angesehen werden. Sie sorgt dafür, dass die Summe über alle Relevanzen für jede Schicht des Netzes identisch ist:

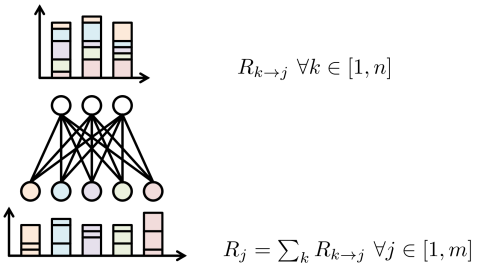
3.3.11 Definition. Relevanzerhaltung

$$\sum_{i=1}^{d} R_i = ... = \sum_j R_{j \leftarrow k} = \sum_k R_{j \leftarrow k} = ... = f(a) \tag{3.36}$$

Eine andere Verbildlichung zeigen Abbildungen 3.6a und 3.6b. Der untere Teil der Abbildung zeigt jeweils eine Schicht, die näher an der Ausgabeschicht liegt, der obere Teil eine Schicht, die näher an der Eingabeschicht liegt. Die Klassifizierungsrichtung ist von oben nach unten, die Propagationsrichtung entsprechend von unten nach oben [32].



(a) Sammeln aller Relevanzen (Klassifizierungsrichtung) [59]



(b) Rückverteilung aller Relevanzen (Propagationsrichtung) [59]

Abbildung 3.6: In den beiden Abbildungen ist die Relevanzerhaltung visualisiert. Dabei wird links dargestellt, wie sich ein Neuroneninput zusammensetzt. Die oberen Neuronen geben dabei bildlich einen Teil ihrer Aktivierung an das Neuron der Folgeschicht weiter, dessen Aktivierung sich somit aus diesen Werten zusammensetzt. Rechts wird die Propagationsrichtung dargestellt, in der jedes Neuron rückwärts in seine Bestandteile aufgeteilt wird und Neuronen der Vorschicht diese „zurückerhalten".

Die Bedeutsamkeit dieser Eigenschaft lässt sich leicht schlussfolgern: Ist die Summe aller Relevanzen in jeder Schicht identisch, gilt automatisch, dass durch die Relevanzen jeder Schicht der Anteil des entsprechend zugehörigen Neurons für eine Entscheidung bestimmt wird. Die Relevanzen lassen sich dank der Relevanzerhaltung also direkt in ein Balkendiagramm übertragen.

3.4 Taxonomie

Abhängig vom konkreten Anwendungsfall sind nicht nur die eigentlichen Erklärungen und ein Vergleich dieser interessant, sondern auch die Laufzeit und der Speicherverbrauch können wichtig werden. Daher werden theoretische Abschätzungen dieser beiden Werte bestimmt, die eine Einordnung der verwendeten Algorithmen ermöglichen sollen.

In den Tabellen 3.1, 3.2 und 3.3 sind jeweils verschiedene Metadaten angegeben. Für die Surrogatmodelle muss im Gegensatz zu den anderen Ansätzen zwischen einer (einmaligen) Laufzeit zur Modellerstellung (Tabelle 3.1) und der eigentlichen Zugriffszeit (Tabelle 3.2) nach der initialen Erzeugung unterschieden werden.

Es gelten folgende Beizeichnungskonventionen:

- $n_{Features}$ bezeichnet die Anzahl der Eigenschaften der einzelnen Datenpunkte
- $n_{Samples}$ beschreibt die Anzahl der Datenpunkte, die Betrachtet werden (für das Training des Baumes sind es die Trainingsdaten, sonst die Testdaten)
- $n_{BatchSize}$ bezieht sich auf die Anzahl der Datenpunkte, die für eine Iteration bei Berechnung der logistischen Regression verwendet werden
- $n_{Iterationen}$ gibt die Anzahl der Iterationen des Gradientenabstiegs an
- h_{Tree} beschreibt die Höhe des Entscheidungsbaumes
- k ist die Anzahl der Nachbarn, die betrachtet werden (kNN) bzw. der Counterfactuals, die berechnet werden (DiCE)
- $f(x)$ bezieht sich auf die Dauer eines Durchlaufs des NN

Methode	Theoretische Laufzeit (Modelltraining)
Entscheidungsbaum	$\mathcal{O}(n_{Features} \cdot n_{Samples} \cdot h_{Tree} + n_{Samples}^2)$ [47]
Logistische Regression	$\mathcal{O}(n_{Iterationen}(n_{features} \cdot n_{BatchSize} + n_{Outputs} \cdot n_{BatchSize}))$ [7]

Tabelle 3.1: Tabelle der theoretischen Worst-Case Laufzeiten für die Modellerstellung für den Entscheidungsbaum und die logistische Regression

[7] https://www.stat.cmu.edu/~ryantibs/convexopt-F18/lectures/stochastic-gd.pdf

Methode	Theoretische Laufzeit (Zugriff)
Entscheidungsbaum	$\mathcal{O}(h_{Tree})$
Logistische Regression	$\mathcal{O}(n_{Features})$ [4, 20]
KNN	$\mathcal{O}(k \cdot n_{Samples} \cdot n_{Features})$
DiCE	$\mathcal{O}(n_{Iterationen}(k^2 \cdot n_{Features} \cdot (f(x) + k^2)))$ [57]
DeepSHAP	$\mathcal{O}(f(x))$ [3, 23]
LRP	$\mathcal{O}(f(x))$ [23]

Tabelle 3.2: Tabelle der theoretischen Worst-Case Laufzeiten für die einzelnen Algorithmen

Methode	Theoretischer Speicherverbrauch
Entscheidungsbaum	$\mathcal{O}(2^{h_{Tree}})$
Logistische Regression	$\mathcal{O}(n_{Features} \cdot n_{Samples})$
KNN	$\mathcal{O}(k \cdot n_{Features})$ [6]
DiCE	$\mathcal{O}(k \cdot n_{Features})$
DeepSHAP	$\mathcal{O}(n_{Neuronen} \cdot n_{Outputs})$ [8] [3, 23]
LRP	$\mathcal{O}(n_{Neuronen} \cdot n_{Outputs})$ [23]

Tabelle 3.3: Tabelle des theoretischen Worst-Case Speicherbedarfs für die einzelnen Algorithmen

Die Laufzeit für das Erstellen des Entscheidungsbaumes ergibt sich aus der Berechnung aller möglichen Splits für alle Datenpunkte $\mathcal{O}(n_{Features} \cdot n_{Samples})$ und einer (abgeschätzten) Höhe des finalen Baumes $\mathcal{O}(h_{Tree})$. Diese Höhe kann zwischen $\mathcal{O}(log(n_{Samples}))$ und $\mathcal{O}(n_{Samples})$ liegen, wobei für die logarithmische Höhe angenommen wird, dass jeder Split zu einer Aufspaltung in zwei annähernd gleichgroße Teile führt, für die lineare Höhe dagegen wird eine Trennung in 1 und $n_{SamplesRest} - 1$ Feature angenommen, bei der je Knoten genau ein Datenpunkt klassifiziert wird [47].

Hierbei gilt zu beachten, dass die Höhe des Baumes zusätzlich für den Nutzer einstellbar ist und nicht ausschließlich durch den Aufbau des Baumes sondern des Weiteren durch diese vorgegebene Maximalhöhe begrenzt wird. In diesem Fall wird der kleinere der beiden Werte (eigentliche Höhe/ Maximalhöhe)verwendet. Die Berechnung der Split-Möglichkeiten wird als konstant angenommen. Hinzu kommt die Laufzeit, die für das Minimal Cost-Complexity Pruning benötigt wird. Diese berechnet für alle vorhandenen Blätter $\mathcal{O}(n_{Samples})$ einen festen Wert und kann maximal alle Blätter des Baumes in dieser Berechnung betrachten. Es kann zudem maximal der gesamte Baum zurückgeschnitten werden. In dieser wird also für einen Knoten (o.B.d.A.) einmal die Berechnung ausgeführt, sodass dieser Knoten entfernt wird. Daraus ergeben sich $\mathcal{O}(n_{Samples})$ Berechnungen, die je $\mathcal{O}(n_{Samples})$ viele Knoten betrachten, also eine Laufzeit von $\mathcal{O}(n^2_{Samples})$. Werden diese kombiniert, überwiegt jeweils

[8] `https://captum.ai/docs/algorithms_comparison_matrix`

die größere als gesamte Laufzeitschranke: $\mathcal{O}(n_{Features} \cdot n_{Samples} \cdot h_{Tree} + n^2_{Samples})$. Alle weiteren Parameter, die dem Nutzer zur Verfügung gestellt werden, beschränken die Laufzeit weiterhin je nach konkretem Wert, weil es sich bei diesen um Abbruchkriterien handelt. Da die Tiefe des Baumes ohnehin für die Laufzeit bestimmt werden muss, wird dieses Abbruchkriterium gesondert miteinbezogen. Alle Weiteren zu berücksichtigen wäre theoretisch denkbar, würde jedoch zu einer starken und nicht notwendigen Verkomplizierung führen.

Der Speicherbedarf wird durch die Anzahl der Knoten des Baumes $\mathcal{O}(2^{h_{Tree}})$ und die Zugriffszeit als Länge des längsten Pfades $\mathcal{O}(h_{Tree})$ bestimmt. Für die Verwendung des Entscheidungsbaumes wird für jeden Knoten, also jede Entscheidung, Speicherplatz benötigt. Die bestimmte Ausgabe des Bauems verbraucht zusätzlich für die Knoten des zugehörigen Pfades Speicherplatz von $\mathcal{O}(h_{Tree})$, dieser ist jedoch immer geringer als die Schranke für den gesamten Baum. Ein Zugriff auf einen Datenpunkt nach Erstellung des Baumes geschieht über das Durchlaufen genau eines Pfades des Entscheidungsbaumes mit Länge $\mathcal{O}(h_{Tree})$ bestimmt. Für einen balancierten Baum ergibt sich eine Höhe von $h_{Tree} = log(n_{Samples})$, für einen maximal unbalancierten Baum dagegen $h_{Tree} = N$. Auch der Speicherverbrauch wird durch die angegebene Maximalhöhe des Baumes beeinflusst. Diese ergibt sich auch an dieser Stelle aus dem Minimum der eigentlichen Baumhöhe und der vom Nutzer gewählten maximalen Höhe.

Das Bestimmen einer logistischen Regression hängt von der Laufzeit für die Berechnung des Gradientenabstiegs $\mathcal{O}(n_{features} \cdot n_{BatchSize} + n_{Outputs} \cdot n_{BatchSize})$ [9] [4] und der Zahl der Iterationen $n_{Iterationen}$ ab. Dabei wird für den Gradienten der Eingaberaum ($\mathcal{O}(n_{features} \cdot n_{BatchSize})$) bzw. der Ausgaberaum ($\mathcal{O}(n_{Outputs} \cdot n_{BatchSize})$) betrachtet, abhängig davon, welcher Wert größer ist. Wird zusätzlich regularisiert, kommt im Gradientenabstieg die Berechnung des l_2-Summanden (elementweises quadrieren der Koeffizienten) in $\mathcal{O}(n_{Features})$ hinzu, diese ist jedoch immer asymptotisch geringer als die Laufzeit der restlichen Berechnungen und es ergibt sich insgesamt $\mathcal{O}(n_{features} \cdot n_{BatchSize} + n_{Outputs} \cdot n_{BatchSize})$ pro Iteration. Alle änderbaren Hyperparameter beeinflussen das konkrete Ergebnis, jedoch nicht die Laufzeit.

Der Speicherbedarf für logistische Regression ohne Betrachtung der Optimierungsfunktion ergibt sich aus der Anzahl der Koeffizienten, also $\mathcal{O}(n_{Features})$. Wird jedoch die zur Berechnung des Gradientenabstiegs verwendete SAG-Funktion miteinbezogen, bestimmt diese die Schranke für den Speicherbedarf mit $\mathcal{O}(n_{features} \cdot n_{Samples})$ [20]. Der anschließende Zugriff für einen Datenpunkt ergibt sich aus dem Summieren der Multiplikation (als konstant angenommen) aller Eigenschaftswerte des jeweiligen Datenpunktes mit den entsprechenden Koeffizienten: $\mathcal{O}(n_{Features})$.

[9] `https://www.stat.cmu.edu/~ryantibs/convexopt/lectures/stochastic-gd.pdf`

Um die Laufzeit von kNN für den Zugriff zu bestimmen, müssen die Anzahl der zu berechnenden nächsten Nachbarn k, die Anzahl der Datenpunkte, zu denen Distanzen berechnet werden müssen $\mathcal{O}(n_{Samples})$ und die Anzahl der Eigenschaften der Datenpunkte $\mathcal{O}(n_{Features})$ betrachtet werden. Der Algorithmus muss zunächst die Distanz des jeweiligen Datenpunktes zu einem anderen Punkt bestimmen ($\mathcal{O}(n_{Features})$). Diese Berechnung wird für alle Datenpunkte ($\mathcal{O}(n_{Samples})$) durchgeführt und aus diesen anschließend die k nächsten Nachbarn herausgesucht. Die Distanzberechnungen sind dabei beide (Euklidische- und Gower-Distanz) je Datenpunkt in Zeit $\mathcal{O}(n_{Features})$ möglich.

Da es sich bei den Nachbarn um Punkte des Datensatzes handelt reicht es aus, die k Referenzen auf die entsprechenden Punkte abzulegen. Da in dieser Arbeit der sklearn-kNN-Algorithmus benutzt wird, wird während der Berechnungen ein kd-Baum erstellt. Dieser benötigt Speicherplatz in Größe $\mathcal{O}(k{\cdot}n_{Features})$ [6]. Es gibt weitere Alternativen zur Berechnung, die für diese Arbeit jedoch nicht relevant sind. Somit ergibt sich ein Speicherbedarf von $\mathcal{O}(k \cdot n_{Features})$.

Der DiCE Algorithmus optimiert die Funktion 3.14 mithilfe des Gradientenabstiegs über k Counterfactuals, die jeweils$n_{Feature}$-groß sind. Die gesamte Laufzeit wird hierbei durch die Berechnung der Determinante zur Optimierung der Diversität mit $\mathcal{O}(k^3)$ [10] [57] oder das Berechnen des Hinge-Losses (siehe 3.15), das wiederum von der Laufzeit des Neuronalen Netzes $\mathcal{O}(f(x))$ abhängt, dominiert. Dabei ergibt sich für die Gradientenberechnung über den k Counterfactuals $\mathcal{O}(k \cdot n_{Features} \cdot (k^3 + k \cdot f(x)))$. Die Berechnung der Distanzen aller k CFs zum zu erklärenden Datenpunkt kann in $\mathcal{O}(n_{Features})$ durchgeführt werden und liegt daher insgesamt in $\mathcal{O}(k{\cdot}n_{Features})$. Dieser Wert wird immer asymptotisch geringer sein als die Laufzeiten der anderen beiden Summanden und fällt daher aus der gesamten Laufzeit ($\mathcal{O}(k \cdot n_{Features} \cdot (k \cdot f(x) + k^3 + k \cdot n_{Featues}))$) heraus. Insgesamt kann die Laufzeit als $\mathcal{O}(k^2 \cdot n_{Features} \cdot (f(x) + k^2)) = \mathcal{O}(k \cdot n_{Features} \cdot (k \cdot f(x) + k^3))$ zusammengefasst werden. Auch hier beeinflussen die einstellbaren Hyperparameter lediglich das Ergebnis jedoch nicht die Laufzeit.

Der Speicherbedarf ergibt sich für DiCE aus der Anzahl der Counterfactuals k und deren Größe $n_{Features}$, da es sich bei diesen Datenpunkten nicht bloß um vorhandene Beispiele aus der Datenmenge handelt, sondern sie vollständig neu berechnet werden. Für jeden der Nachbarn wird somit ein eigener Datenpunkt mit allen Eigenschaften abgelegt und der gesamte Speicherbedarf liegt in $\mathcal{O}(k \cdot n_{Features})$

[10] http://people.mpi-inf.mpg.de/~msagralo/LR_Zerlegung.pdf

Um die Laufzeit des DeepSHAP Algorithmus zu bestimmen müssen der Vorwärtslauf, in dem der Datenpunkt vom Netz klassifiziert und die Werte aller Neuronen bestimmt werden, die Bestimmung der Referenzen und Differenzen für jedes Neuron und die anschließende Berechnung der Multiplizierer berücksichtigt werden. Der Vorwärtslauf wird durch die Laufzeit des Netzes $\mathcal{O}(f(x))$ bestimmt. Die Referenzwerte und anschließend die Differenzen zu diesen werden für alle Neuronen $\mathcal{O}(n_{Neuronen})$ in konstanter Zeit berechnet. Werden hier die Erwartungswerte zur Kombination von DeepLIFT und Shapley Values zu DeepSHAP 3.24 verwendet, muss diese Berechnung mit berücksichtigt werden: Die Erwartungswerte können in $\mathcal{O}(n_{Featuers})$ berechnet werden, da jede Eigenschaft dafür zu betrachten ist. Für den Rückwärtsschritt muss für jede Kante ($\mathcal{O}(n_{Kanten})$) ein Multiplizierer errechnet werden. Aus diesen kann dann mithilfe der Kettenregel 3.19 in konstanter Zeit die Summe gebildet werden, die den Multiplizierer für ein Input-Feature und einen Output bestimmt. Da die Multiplizierer jeweils vom entsprechenden Output abhängen, müssen diese Berechnungsschritte je Ausgabe ($\mathcal{O}(n_{Outputs})$) durchgeführt werden, was zu einer Laufzeit von $\mathcal{O}(n_{Kanten} \cdot n_{Outputs})$ führt. Da die Berechnung von DeepLIFT der Berechnung von LRP-ϵ entspricht [3, 23], kann der Rückwärtsschritt jedoch wie für LRP durch $\mathcal{O}(f(x))$ abgeschätzt werden [23], woraus sich eine gesamte Laufzeit von ebenfalls $\mathcal{O}(f(x))$ ergibt.

Auch der Speicherbedarf bestimmt sich aus der Gleichheit von DeepLIFT zu LRP-ϵ [3, 23]. Werden alle errechneten Multiplizierer der Feature $\mathcal{O}(n_{Features})$ für jede mögliche Ausgabe $\mathcal{O}(n_{Outputs})$ für die Berechnung abgespeichert, führt dies zu einem gesamten Speicherverbrauch von $\mathcal{O}(n_{Neuronen} \cdot n_{Outputs})$. Dabei wird für diese Arbeit angenommen, dass für eine Verbesserung der Laufzeit alle errechneten Multiplizierer für ihr jeweiliges Neuron (als Summe) abgespeichert werden.

Für LRP wird die Laufzeit mit $\mathcal{O}(f(x))$ als die Laufzeit des Netzes angegeben [23]. Diese ergibt sich aus dem zu errechnenden Vorwärtslauf für den zu erklärenden Datenpunkt und der anschließenden Rückwärtsberechnung der Relevanzen (jeweils in $\mathcal{O}(f(x))$).

Der Speicherbedarf hängt von der Zahl der berechneten Relevanzen $n_{Neuronen}$ ab. Diese werden je Ausgabe bestimmt und müssen daher für jeden Output $\mathcal{O}(n_{Outputs})$ bestimmt werden, was einen gesamten Speicherverbrauch von $\mathcal{O}(n_{Neuronen} \cdot n_{Outputs})$ ergibt. Dabei existiert ein Trade-off zwischen dem Speicherverbrauch und der Laufzeit [23]. Dieser ergibt sich aus der Zahl der abgespeicherten Relevanzen und wird in dieser Arbeit als Speicherung aller Relevanzen für das jeweils zugehörige Neuron ($\mathcal{O}(n_{Neuronen})$) je Output angenommen.

Kapitel 4

Implementierung

Die Implementierung der Oberfläche stellt den Hauptteil dieser Arbeit dar. Hierfür wurde als Programmiersprache Python gewählt. Die erstellten Visualisierungen wurden mithilfe der Funktionalitäten, die durch die plotly- [1] und plotlys Dash-Bibliothek [2] angeboten werden implementiert. Zur Oberflächenerstellung gehören folgende Bestandteile:

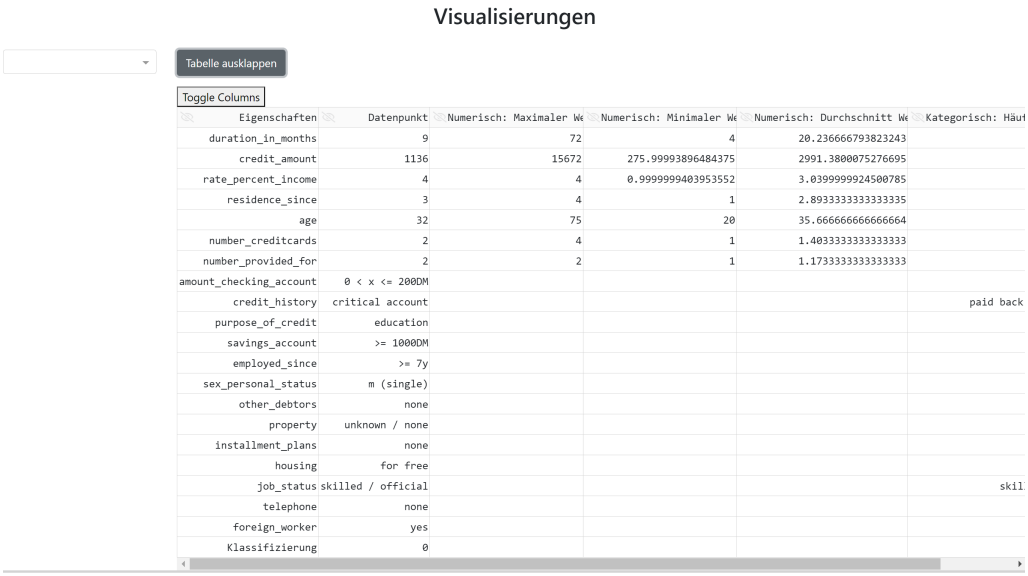
- das Einbinden der fünf bereitgestellten und des selbst implementierten Algorithmus
- die Herstellung einer Verbindung zwischen den Algorithmen und dem bereitgestellten Neuronalen Netz
- das Bestimmen und Aufbereiten der entsprechenden für die Verfahren passenden Visualisierungen
- das Einbinden interaktiver Komponenten

Durch den letzten Punkt wird dem Nutzer für alle Modelle ermöglicht, einen bestimmten Datenpunkt zu wählen und für diesen Erklärungen zu erhalten. Für diejenigen Modelle, die weitere Eingabeparameter besitzen wurden zudem einige Interaktionsmöglichkeiten für Hyperparameter, die Einfluss auf die verfahrensinternen Berechnungen (anstatt nur auf die Inputvariablen) haben, angeboten. Diese veränderbaren Hyperparameter sind im jeweiligen **Code**-Abschnitt der Algorithmen aufgezählt. An diesen Stellen werden auch die Auswirkungen der Änderungen auf das Modell erörtert.

[1] `https://plotly.com/`
[2] `https://plotly.com/dash/`

4.1 Grundidee und Aufbau

Die erstellte Oberfläche zeigt neben den verschiedenen Visualisierungen, welche die Ausgaben der sechs Algorithmen in unterschiedlichen Darstellungsformen anzeigen, eine Übersichtstabelle der Datenpunkte. In dieser sind jedoch nicht alle Beispiele des Datensatzes enthalten. Stattdessen zeigt sie die Werte des aktuell gewählten Datums, sowie den maximalen, minimalen und durchschnittlichen Wert für quantitative Eigenschaften und die Ausprägung, die in der Mehrheit der Datenpunkte vorherrscht, für qualitative Eigenschaften. Auf diese Weise kann ein Nutzer direkt einordnen, in welchen Bereichen er Werte eines Datenpunktes erwarten kann und ob es sich um eher gewöhnliche oder eher ungewöhnliche Ausprägungen handelt. Für alle Visualisierungen musste zunächst die Vorverarbeitung der Datenpunkte wieder rückgängig gemacht werden. Warum sowohl die Vorverarbeitung als auch die Invertierung dieser notwendig waren und wie das Umkehren umgesetzt wurde, wird genauer in Kapitel 4.3 erläutert.



Eigenschaften	Datenpunkt	Numerisch: Maximaler We	Numerisch: Minimaler We	Numerisch: Durchschnitt We	Kategorisch: Häuf
duration_in_months	9	72	4	20.236666793823243	
credit_amount	1136	15672	275.99993896484375	2991.3800075276695	
rate_percent_income	4	4	0.9999999403953552	3.0399999924500785	
residence_since	3	4	1	2.8933333333333335	
age	32	75	20	35.666666666666664	
number_creditcards	2	4	1	1.4033333333333333	
number_provided_for	2	2	1	1.1733333333333333	
amount_checking_account	0 < x <= 200DM				
credit_history	critical account				paid back
purpose_of_credit	education				
savings_account	>= 1000DM				
employed_since	>= 7y				
sex_personal_status	m (single)				
other_debtors	none				
property	unknown / none				
installment_plans	none				
housing	for free				
job_status	skilled / official				skil
telephone	none				
foreign_worker	yes				
Klassifizierung	0				

Abbildung 4.1: Im Bild ist die ausklappbare Übersichtstabelle des gewählten Datenpunktes, sowie das Dropdown-Menü, mit dem ein spezieller Datenpunkt aus der Menge gewählt werden kann, zu sehen. Ist die Tabelle eingeklappt, sind nur das Dropdown-Menu und der Knopf sichtbar. Durch einen Klick auf „Tabelle ausklappen“ kann die Übersichtstabelle geöffnet und, wie im Bild dargestellt, angezeigt werden. Es ist möglich, in der Tabelle horizontal zu scrollen.

Jeder Algorithmus hat eine Visualisierung erhalten, die mit der jeweiligen Ausgabe korrespondiert und diese in angemessener Weise darstellt. Zudem werden Einschätzungen von Laufzeit und theoretischem Speicherbedarf in Form einer Farbkodierung angegeben. Diese basieren auf den Berechnungen in Unterkapitel 3.4. Der Entscheidungsbaum wurde sowohl bildlich, als Pfad des Entscheidungsbaumes mit Knoten und Kanten, als auch durch eine textuelle Beschreibung jeweils des Ergebnispfades umgesetzt. Die Wahl der textuellen

Ausgabe ermöglicht ein Zusammenfassen von Eigenschaften, die in einem Ergebnispfad mehrfach vorkommen. Im Rahmen dieser Arbeit wurde die Möglichkeit nicht genutzt, da zunächst eine Ähnlichkeit zwischen der textuellen und der bildlichen Ausgabe erkennbar sein sollte.

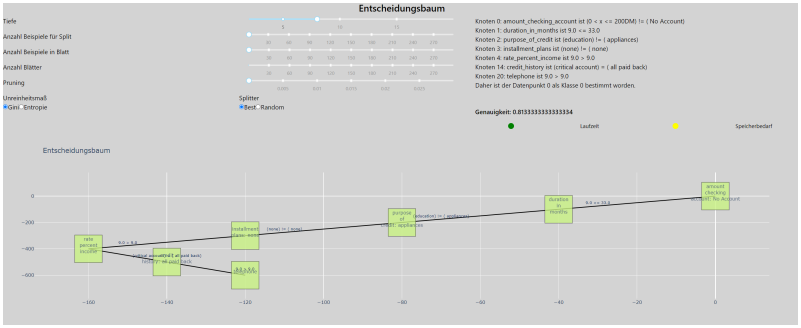


Abbildung 4.2: Hier wird ein Entscheidungsbaum als Pfad (unten) und als Text (oben rechts), sowie die einstellbaren Hyperparameter (oben links) visualisiert. Des Weiteren sind die Genauigkeit des jeweiligen Baumes sowie die theoretische Laufzeit und der theoretische Speicherbedarf angegeben.

Als Konfigurationsmöglichkeiten wurden verschiedene (diskrete) Eingabewerte für quantitative Hyperparameter sowie einige Optionen für qualitative Hyperparameter bereitgestellt. Eine Veränderung dieser bewirkt automatisch eine vollständige neue Berechnung des Entscheidungsbaumes und damit eine Aktualisierung der Visualisierungen. Nähere Details zur Implementierung des automatischen Aktualisierens sind im Kapitel 4.2 beschreiben.

Für die Visualisierung der logistischen Regression wurde ein Balkendiagramm gewählt. Dieses zeigt sowohl die durch die logistische Regression berechnete (und aufbereitete) Ausgabe für das gesamte Modell als auch eine selbst errechnete Ausgabe für den aktuellen Datenpunkt. Dabei ergeben sich die konkreten, datenpunktspezifischen Werte aus der Multiplikation der Eigenschaftswerte des Datums mit den errechneten gesamten Koeffizienten. Für beide Ergebnismengen wurde ein eigener Balken zum Diagramm hinzugefügt, sodass diese entsprechend für jeden Koeffizienten nebeneinander dargestellt werden und somit leicht miteinander vergleichbar sind. Die x-Achse hat als Beschriftung die Namen der entsprechend visualisierten Eigenschaften erhalten. Für die y-Achse werden dynamisch der größte und kleinste Wert für die jeweilige berechnete Ausgabe bestimmt und aus diesen eine Skalierung und Beschriftung erstellt. Dadurch kann das Diagramm jederzeit so groß wie möglich angezeigt werden ohne über den dargestellten Bereich hinauszulaufen.

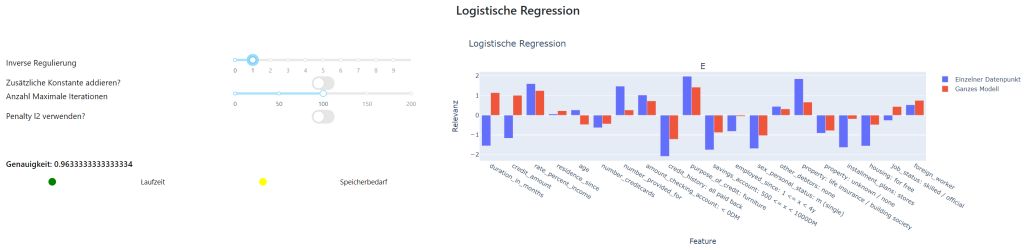
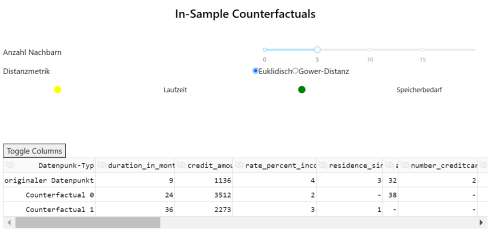


Abbildung 4.3: Es ist das Balkendiagramm einer logistischen Regression (rechts) und die einstellbaren Hyperparameter gezeigt. Zudem sind die Genauigkeit, die theoretische Laufzeit und der theoretische Speicherverbrauch visualisiert. Das Diagramm zeigt in blau die Ergebnisse für den gewählten Datenpunkt, in rot die für das gesamte Modell. An der x-Achse sind die jeweiligen Eigenschaften angegeben, an der y-Achse die Höhe der Werte der zugehörigen Koeffizienten.

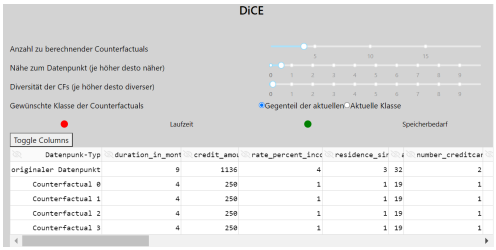
Auch für die Hyperparameter dieses Algorithmus werden Auswahlmöglichkeiten für qualitative und Einstelloptionen für quantitative Attribute angeboten. Die Auswahl der Regularisierung unterscheidet nur zwischen der Verwendung oder dem Auslassen dieser. Daher wurde ein Schalter, der die Optionen *an*: regularisieren und *aus*: nicht regularisieren bietet, verwendet. Soll regularisiert werden, wird hierfür die l_2-Regularisierung genutzt. Die Wahl des Nutzers wird anschließend in die passende Eingabe für die Berechnung der logistischen Regression umgewandelt. Auch die Auswahl, ob eine zusätzliche Konstante (y-Achsenabschnitt) zum Modell hinzuaddiert werden soll, kann über einen Schalter umgesetzt werden. Hier war keine Konvertierung der Werte notwendig, da direkt die Wahrheitswerte als Eingabe für den Algorithmus fungieren.

Eine Veränderung der verschiedenen Eingabeoptionen führt sowohl für den Entscheidungsbaum (bei Änderung der Entscheidungsbaum-Hyperparameter) als auch für die logistische Regression (bei Änderung der Hyperparameter für logistische Regression) zu einer Neuberechnung des jeweiligen Modells. Dadurch verändert sich automatisch die Genauigkeit der Entscheidungen, die auf dem Modell beruht. Aus diesem Grund wurde für beide Surrogatmodelle ein zusätzliches Feld ergänzt, in dem die Genauigkeit des aktuell verwendeten Modells angezeigt wird. So kann der Nutzer jederzeit entscheiden, wie nah die Erklärungen tatsächlich an den Entscheidungen des Netzes liegen sollen.

Für das Darstellen der Counterfactuals eignet sich eine Tabelle, in der zusätzlich der zu erklärende Datenpunkt mit seinen entsprechenden Werten angezeigt wird. Hier unterscheidet sich die Visualisierung der In-Sample-CFs und der DiCE Ergebnisse nur durch die eigentlichen Werte, nicht jedoch in der Art der Visualisierung.



(a) kNN Tabelle

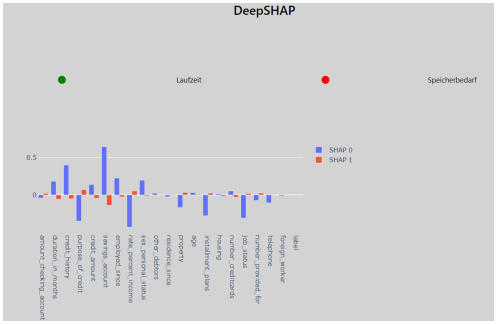


(b) DiCE Tabelle

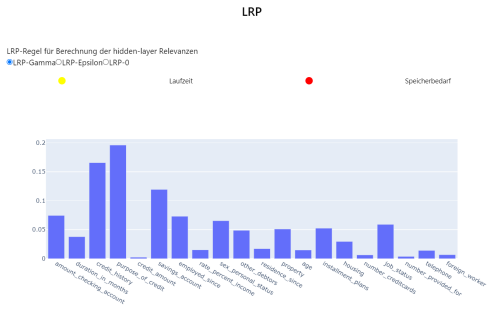
Abbildung 4.4: Es sind die tabellarischen Ergebnisse für In-Sample Counterfactuals mithilfe des kNN-Klassifizierers (links) und für DiCE (rechts) abgebildet. Beide Tabellen zeigen zusätzlich den gewählten Datenpunkt in der ersten Zeile an. Darunter sind die bestimmten CFs angegeben. Des Weiteren sind die verschiedenen Konfigurationsoptionen dargestellt.

Auch hier werden für die einstellbaren Hyperparameter Optionen für quantitative und qualitative Konfigurationswerte zur Verfügung gestellt.

Ähnlich verhält es sich mit der Visualisierung von den DeepSHAP-Ergebnissen und den LRP-Ausgaben. Für beide wurden Balkendiagramme gewählt und diese ähnlich umgesetzt wie für die logistische Regression. DeepSHAP hat hierbei ebenfalls zwei zu visualisierende Datenmengen, die in zwei Balken je Eigenschaft resultieren. Hier beschreiben diese Balken jedoch Entscheidungen, die für Klasse **0** (der Kredit wurde abgelehnt) bzw. Klasse **1** (der Kredit wurde genehmigt) sprechen. LRP zeigt nur eine Ergebnismenge, welche die Wichtigkeit der einzelnen Eigenschaften für eine Entscheidung für Klasse **1** angeben.



(a) DeepSHAP Balkendiagramm



(b) LRP Balkendiagramm

Abbildung 4.5: Die Bilder zeigen die Visualisierungen der DeepSHAP (links) und LRP (rechts) Ergebnisse in Form von Balkendiagrammen. Links sind die Werte für die Klasse 0 (hier SHAP 0 genannt) in blau, für die Klasse 1 (hier SHAP 1 genannt) in rot visualisiert. Rechts sind die LRP Werte für Klasse 1 gezeigt. Beide Graphen sind an der x-Achse mit den zugehörigen Eigenschaften und an der y-Achse mit der Höhe des Ergebniswertes beschriftet. Für LRP steht zudem die Konfiguration der zu nutzenden LRP-Regel zur Verfügung.

4.2 Die Oberfläche mit Dash und plotly

Zur Umsetzung der Visualisierung wurden verschiedene Pakete von plotly, Dash und igraph sowie pandas für die Erstellung von Dataframes verwendet.

Hierbei stellt Dash das eigentliche Herzstück - die sogenannte **Dash-App**, in der alle Visualisierungen kombiniert und angeordnet werden - zur Verfügung. Diese muss ein Layout erhalten, das alle Objekte, die angezeigt werden sollen, beschreibt. Es enthält verschiedene Bereiche, in die html-Komponenten und Visualisierungen eingebunden sind. Auch hierfür wurden eingebaute Funktionen verwendet.

Die Auswahl des Datenpunktes geschieht mithilfe eines **html.Dropdowns**, das von Dash direkt bereitgestellt wird.

Die verbleibenden Plots wurden als **Dash.Graph**-Objekte im Layout eingefügt. In diesen wiederum befinden sich die Plots der einzelnen Modelle, die mit plotly erzeugt wurden.

Plotly bildet mit den Graph-Objects die Grundlage für die einzelnen Visualisierungen. Die meisten verwendeten Diagrammtypen werden von dieser Bibliothek bereitgestellt. Für die Erstellung des Entscheidungsbaumes wurden zusätzlich Methoden aus der **igraph** Bibliothek (für die Erstellung des Baumes als igraph.Tree aus Knoten und Kanten) verwendet. Die verschiedenen Eingabevisualisierungen sind ebenfalls Dash-Komponenten.

Die Dash-App benötigt jedoch nicht nur das erstellte Layout. Sie muss zudem gestartet werden und bestimmte Callbacks erhalten um zu einer interaktiven Oberfläche zu werden. Diese Callbacks wurden für jeden Visualisierungsbereich erstellt. Bei diesen handelt es sich um

- die Übersichtstabelle, die den Datenpunkt und Modell-Metadaten zeigt
- die Visualisierung des Entscheidungsbaumes (als Scatterplot und Text)
- den Barplot der logistischen Regression
- die Tabelle, die die In-Sample Counterfactuals visualisiert
- die Ausgabetabelle von DiCE
- den Barplot für die Darstellung der DeepSHAP-Werte
- die Visualisierung der LRP-Ergebnisse in einem Barplot

Alle Bereiche erhalten einen (individuellen) Callback-Aufruf, wenn der gewählte Datenpunkt sich ändert. Doch nicht nur das Wählen eines anderen Datenpunktes, sondern auch das Anpassen eines beliebigen Hyperparameters sorgt für einen Callback-Aufruf. Dieser ist immer abhängig vom geänderten Hyperparameter und aktualisiert ausschließlich die Visualisierung des zugehörigen Erklärbarkeitsverfahrens. Wird zum Beispiel die Tiefe

des Entscheidungsbaumes verändert, so sorgt ein individueller Callback dafür, dass sich lediglich der Scatterplot, die Textausgabe und das Genauigkeit-Ausgabefeld für den Entscheidungsbaum ändern. Wird die Wahl der LRP-Regel verändert, so wird ein individueller Callback für ein Update des LRP-Barplots aufgerufen.

Jeder dieser verschiedenen Callbacks sorgt somit für die neue Berechnung mit angepassten Hyperparametern des jeweiligen Modells. Es konnten jedoch einige Laufzeitoptimierungen vorgenommen werden: Die Ergebnisse des DeepSHAP-Explainers werden einmalig für alle Datenpunkte bestimmt und anschließend nur noch ausgelesen. Ebenso wird die Gower-Distanzmatrix einmalig bei Programmstart für alle Datenpunkte untereinander bestimmt. Für alle anderen Methoden kann keine solche Optimierung vorgenommen werden, da sich hier die internen Berechnungen abhängig von den gewählten Hyperparametern ändern und jedes Mal neu errechnet werden müssen. Entsprechend den jeweiligen Laufzeiten und dem Speicherbedarf ist für jeden Algorithmus eine „Ampel“ eingefügt worden. Diese signalisiert über grün (schnelle Laufzeit bzw. geringer Speicherbedarf), gelb (mittlere Laufzeit bzw. mittlerer Speicherbedarf) und rot (hohe Laufzeit bzw. großer Speicherbedarf) wie die jeweiligen Algorithmen einzuordnen sind. Dabei ist für DeepSHAP die Laufzeit mit der vorherigen Berechnung aller Werte angezeigt.

4.2.1 Visualisierung des Entscheidungsbaumes

Für die Umsetzung des Entscheidungsbaumes im Code wurde die Grundlage von sklearn [9, 37] verwendet. Diese implementiert grundlegende Funktionen zur Erstellung eines Entscheidungsbaumes mit mehreren Variablen. In der realisierten Visualisierung wurden von diesen Hyperparametern sieben zur freien Konfiguration bereitgestellt. Dabei handelt es sich um folgende Attribute:

- die Veränderung der Tiefe des Baumes
- das Anpassen der Mindestanzahl der anders-klassifizierten Beispiele, die für einen Split eines Blattes notwendig sind
- die Bestimmung der Mindestanzahl an Beispielen in einem Blatt
- das Verändern der Anzahl der Blätter des Baumes
- eine Konfiguration der Stärke des Zurückschneidens des Baumes
- die Wahl zwischen Gini-Unreinheit und Entropie als Maß für die Unreinheit im Blatt
- die Möglichkeit zwischen bestem oder zufälligem Aufteilen eines Knotens zu wählen

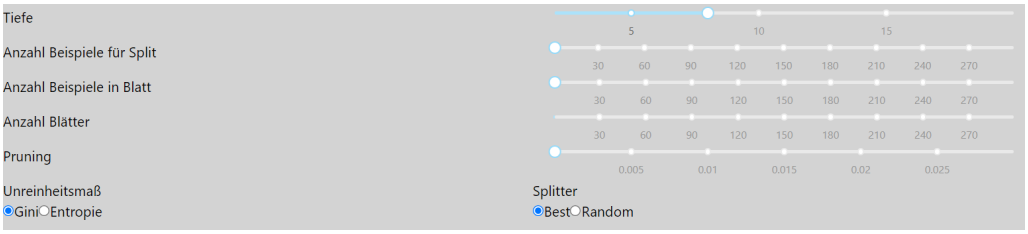


Abbildung 4.6: Es sind die aufgelisteten Hyperparameter für den Entscheidungsbaum in ihrer jeweiligen realisierten Umsetzung in der Oberfläche angezeigt. Die Tiefe, die Anzahl der Beispiele für einen Split und die Anzahl der Beispiele in einem Blatt sind als Slider dargestellt und in 1-er Schritten zwischen 0 und 100 auswählbar. Das Pruning wurde auch über einen Slider realisiert, kann jedoch nur im Bereich 0 bis 0,03 liegen, ohne dass der Baum zu stark zurückgeschnitten wird. Für Unreinheitsmaß und Splitter stehen jeweils zwei Optionen in Form von Radio-Items zur Verfügung.

Als Konfigurationsoptionen wurden mithilfe von **Slidern** verschiedene (diskrete) Eingabewerte für quantitative Hyperparameter und durch die Nutzung von **RadioItems** mit nur einer aktiven Auswahl die Optionen für qualitative Hyperparameter bereitgestellt. Eine Veränderung dieser bewirkt automatisch eine vollständige neue Berechnung des Entscheidungsbaumes und damit eine Aktualisierung der Ausgaben im Scatterplot-Baum und der Textausgabe.

Die Slider haben je nach möglichen Eingabegrößenordnungen verschiedene Skalierungen und Markierungen erhalten, sodass deutlich wird, welchen Wert der Nutzer gewählt hat.

Als Tiefe des Baumes wird die Anzahl der Ebenen, die maximal erzeugt werden dürfen bezeichnet. Sie gibt gleichzeitig die maximale Länge des ausgegebenen Pfades an. Hier wurde als Wert

Die Wahl zwischen Entropie oder Gini-Unreinheit bestimmt, welche der Funktionen als Unreinheitsmaß verwendet wird. Es kann zwischen der Suche des besten oder der Verwendung eines zufälligen Split-Kandidaten ausgewählt werden.

Bei der Mindestanzahl an Datenpunkten in einem Blatt handelt es sich um ein Abbruchkriterium. Wird dieser Mindestwerte bei einem Split nicht erreicht, ist die Berechnung für den jeweiligen Pfad beendet. Ähnlich verhält es sich mit der Maximalanzahl der Blätter des Baumes: Ist diese erreicht, bricht die Berechnung ab.

Von dem erstellten Entscheidungsbaum wird anschließend der entsprechende Pfad, der die Erklärung für den gewählten Datenpunkt enthält, ausgegeben. Dies geschieht sowohl in Text-Form als auch durch Visualisierung des Wegs als Pfad mit Konten und Kanten. Um diesen zu ermitteln, können durch bereitgestellte Methoden der Pfad eines Datenpunktes und von diesem die zugehörigen Knoten-Indizes abgefragt werden.

Da sklearn keine kategorischen Feature unterstützt, muss zur Modellerstellung die Vorverarbeitung des Netzes verwendet werden. Dies führt dazu, dass für die Visualisierung

die Ergebnisse anschließend wieder zurückgerechnet werden müssen, sodass für den Nutzer verständliche Ergebnisse zur Verfügung stehen (siehe Kapitel 4.3).

Der Text enthält die Knotennummer (im Baum) und die entsprechende im Knoten betrachtete Eigenschaft. Diese Werte können als Index-Liste über den erstellten DecisionTreeClassifier abgefragt werden. Abhängig vom Wert der Eigenschaft wird für nominale Variablen bestimmt und angegeben, ob die Ausprägung im Datenpunkt aufgrund der Gleichheit oder Ungleichheit zu einem konkreten Wert zur Entscheidung beigetragen hat. Beispielsweise könnte der Beziehungsstatus durch Gleichheit zu einer Entscheidung beitragen, wenn im Baum der Wert verheiratet gefragt war und der Datenpunkt ebenfalls die Ausprägung verheiratet enthält. Durch Ungleichheit kann es immer dann zur Entscheidung beitragen, wenn der Baum den Wert verheiratet fordert, der Datenpunkt aber eine andere Ausprägung (hier zum Beispiel ledig, geschieden, verwitwet) enthält. Für ordinale Werte wird bestimmt und angegeben, ob sie größer gleich oder kleiner als ein bestimmter Grenzwert waren und so zur Entscheidung beitrugen. Dies wurde durch die Verwendung einzelner html.Divs pro Zeile umgesetzt.

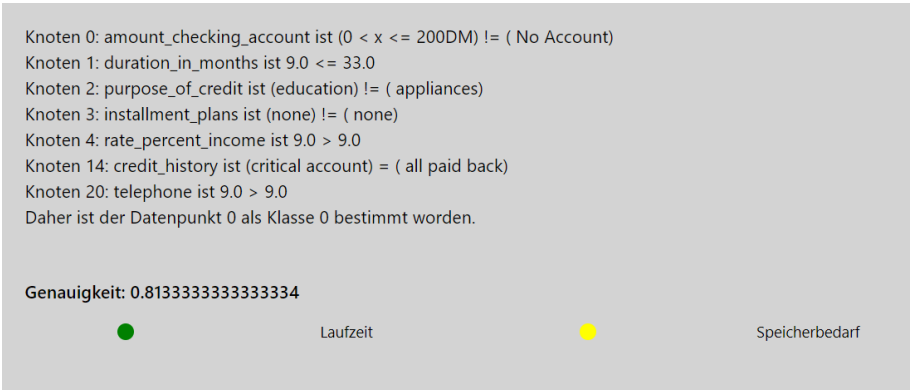


Abbildung 4.7: Hier ist eine beispielhafte Textausgabe für den Datenpunkt 0 und einen berechneten Entscheidungsbaum dargestellt. Der Pfad ist in Textform angegeben, sodass für jeden Knoten des Pfades die entsprechende Nummer sowie die Entscheidung mit Angabe des Thresholds angezeigt werden. Die letzte Zeile gibt zudem die Entscheidung an. Mit etwas Abstand sind außerdem die Genauigkeit sowie für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

Für die Erstellung eines Entscheidungsbaumes wurde ein **Scatterplot** von plotlys graph_objects erstellt. Zur Erstellung der passenden Knoten und Kanten wurde anschließend ein igraph **Graph.Tree** mit der Gesamtzahl an Knoten und maximal zwei Kindern pro Knoten erstellt. Von der sklearn Bibliothek wird ein Ergebnispfad mit den zugehörigen Knotenindizes zurückgegeben. Da aus diesen ein Pfad erstellt werden soll, der bestenfalls Abzweigungen in verschiedene Richtungen je nach Ergebnis der Abfrage hat, mussten also doppelt so viele Knoten zur Verfügung gestellt werden, wie die höchste ausgegebene Knotennummer anzeigt. So konnte ein vollständiger Binärbaum erstellt werden, aus dem dann

ein zu den Entscheidungen passender Pfad von Knoten gewählt und visualisiert wurde. Die Knoten enthalten jeweils die zum Split zugehörigen Abfragen, die Kanten sind mit den Werten und Thresholds beschriftet, die zur Entscheidung geführt haben.

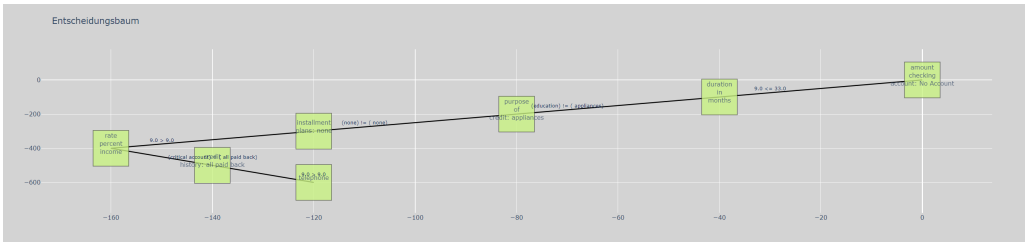


Abbildung 4.8: Es ist ein Pfad, der für den ausgewählten Datenpunkt eine Erklärung liefert, visualisiert. Dieser besitzt die gleichen Knoten und Abfragen wie die textuelle Ausgabe, lediglich in anderer Darstellungsform.

4.2.2 Visualisierung der logistischen Regression

Für die Implementierung des Linearen Modells, genauer einer logistischen Regression, wurde ebenfalls die Grundlage der sklearn-Bibliothek [9, 37] verwendet. Sie stellt Funktionen zur Erstellung eines linearen Modells mit mehreren Hyperparametern zur Verfügung. In der Web-Oberfläche wurde von diesen Hyperparametern eine Auswahl zur freien Konfiguration bereitgestellt. Es sind folgende Veränderungsoptionen einstellbar:

- die Kalibrierung der Inversen Regulierung
- die Auswahl der Regularisierung als l_2 oder None
- das optionale Anpassen der Größe einer Konstante (y-Achsenabschnitt), die hinzuaddiert werden kann
- die Festlegung einer maximalen Anzahl an Iterationen, welche die Optimierungsfunktion durchlaufen darf

Auch für die Hyperparameter dieses Algorithmus wurden **Slider** und **RadioItems** verwendet um die entsprechenden Einstellmöglichkeiten für qualitative und quantitative Attribute zu bieten. Die beiden Schalter wurden als **BooleanSwitch**es realisiert.

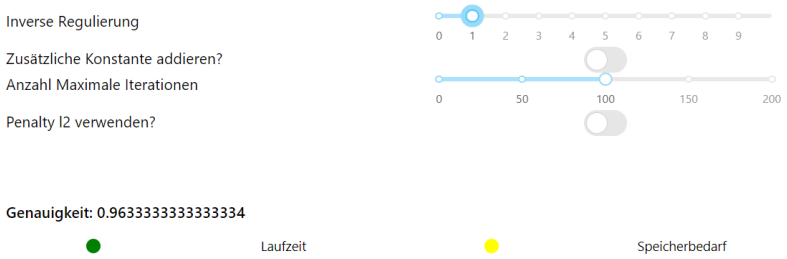


Abbildung 4.9: Hier sind die aufgelisteten Hyperparameter für logistische Regression in ihrer jeweiligen realisierten Umsetzung in der Oberfläche dargestellt. Die inverse Regulierung ist ein Slider im Bereich 0 bis 10, die maximale Anzahl der Iterationen wird ebenfalls über einen Slider angegeben, hier liegt der Bereich jedoch zwischen 0 und 200. Die Abfragen, ob eine Konstante addiert werden bzw. eine Penalty verwendet werden soll, sind über Boolean Switches umgesetzt. Mit etwas Abstand sind außerdem die Genauigkeit, sowie für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

C ist der inverse Regularisierungsparameter und beeinflusst - wenn für die Regularisierung l_2 gewählt wurde - wie stark diese sich auswirkt (siehe 3.1.4). Bei der Konstante, die hinzuaddiert wird, handelt es sich um c aus der Basisfunktion der logistischen Regression (3.1.3).

Anhand der eingestellten Hyperparameter wird dann eine logistische Regression auf den Testdaten ausgeführt. Diese bestimmt für alle Eigenschaften die Werte der Koeffizienten in $h(x)$. Durch diese Koeffizienten wird das lineare Surrogatmodell beschrieben. Eine Visualisierung als Balkendiagramm kann dem Nutzer somit für alle Feature die Werte der jeweiligen Koeffizienten (jeweils für das gesamte Modell und für einen einzelnen Datenpunkt) anzeigen. Diese Annäherung ermöglicht dem Nutzer, aus den Balken Einflüsse der einzelnen Eigenschaften und deren Gewichtung abzulesen.

Die verwendete Bibliothek berechnet für die Erstellung der logistischen Regression zunächst eine Annäherung an die Klassifizierung des neuronalen Netzes und optimiert hierbei die Koeffizienten. Die einstellbaren Hyperparameter beeinflussen, wie gut dieses Surrogatmodell bestimmt wird. Mithilfe einer bereitgestellten Methode können für die Datenpunkte die Vorhersagen des linearen Modells berechnet und aus diesen dann die Genauigkeit bestimmt werden.

Die Koeffizienten der Funktion können von dem erstellten Objekt, das die Hyperparameter der logistischen Regression enthält, abgefragt und anschließend in einem Diagramm visualisiert werden. Hierbei ist zu beachten, dass nur die Werte für das gesamte Modell zurückgegeben werden. Für einen einzelnen Datenpunkt muss je Eigenschaft das Produkt der Ausprägung im Datenpunkt und dem entsprechenden Koeffizienten gebildet werden. Beide Ergebnisse werden im Balkendiagramm angezeigt.

Die Höhe eines Balkens gibt dabei den Einfluss eines Wertes auf die gesamte Klassifizierung an. Negative Werte sind dabei als Indiz für einen negativen Einfluss zur Klassifizierung **1** (der Kredit wurde genehmigt) und gleichzeitig für einen positiven Einfluss zur Klassifizierung **0** (der Kredit wurde abgelehnt) zu verstehen. Beispielsweise zeigt ein negativer Balken für die Eigenschaft „Alter “ folgendes: Der Koeffizient für das Alter ist negativ, die tatsächlichen Ausprägungen dagegen immer positiv. Daher wird eine Multiplikation, wie sie in $h(x)$ (siehe Unterkapitel 3.1.2) aus Koeffizienten und Datenpunkten berechnet wird, immer ein negatives Ergebnis haben. Hier wird diese Berechnung nicht summiert sondern lediglich für einen konkreten Koeffizienten und die entsprechende Ausprägung im Datenpunkt betrachtet. Je größer das Alter, desto stärker wird der Einfluss des gesamten Produkts auf die gesamte Summe von $h(x)$. Das Alter drückt in diesem Beispiel das Ergebnis in die negative Richtung und somit näher zur Klasse 0, weg von Klasse 1. Das bedeutet, je größer das Alte ist, desto geringer die Chance, einen Kredit zu erhalten. Analog sind positive Balken als Indiz für einen positiven Einfluss zur Klassifizierung **1** und gleichzeitig für einen negativen Einfluss zur Klassifizierung **0** zu verstehen

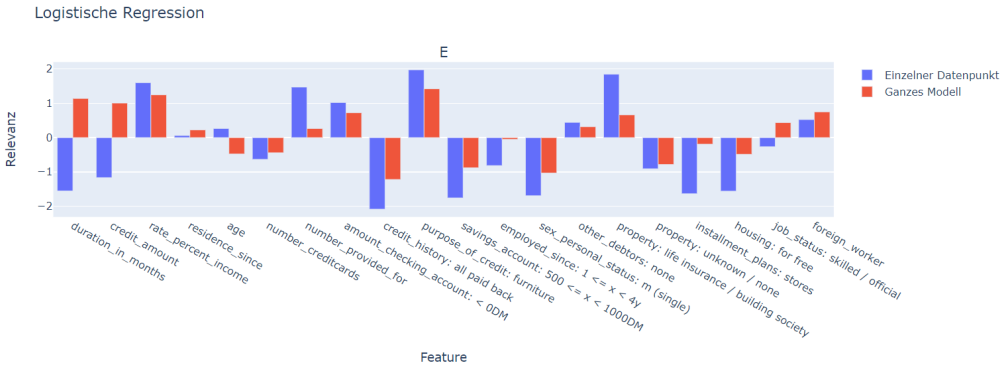


Abbildung 4.10: Die Abbildung zeigt ein Balkendiagramm für die bestimmte logistische Regression. Die blauen Balken beschreiben einen einzelnen Datenpunkt, die roten das gesamte Modell. An der x-Achse sind die einzelnen Eigenschaften, die den Koeffizienten zuzuordnen sind, an der y-Achse die jeweiligen Höhen angegeben.

Für die Darstellung der Ergebnisse der logistischen Regression wurde ein **Barplot** gewählt. Dieser zeigt die zwei verschiedenen Plots (gesamtes Modell und gewählter Datenpunkt) für jedes Feature an, sodass die entsprechenden Balken für jeden Koeffizienten nebeneinander dargestellt werden und dadurch leicht miteinander vergleichbar sind.

4.2.3 Visualisierung der In-Sample Counterfactuals

Für die Implementierung der In-Sample CFs wird der kNN-Algorithmus aus der sklearn Bibliothek verwendet [9, 37]. Hier wird eine Funktion zur Berechnung der k nächsten Nachbarn direkt bereitgestellt. Der Nutzer erhält in der Oberfläche die Möglichkeit, die Anzahl

der Nachbarn (den Wert für **k**), die berechnet werden sollen, anzugeben. Zudem kann zwischen Euklidischer- und Gower-Distanz gewählt werden. Dabei mussten - für die euklidische Distanzberechnung - die nominalen Werte durch die Vorverarbeitung in One-Hot Codierung (siehe Definition 2.4.1) angegeben werden, sodass die Berechnung der Distanzen für diese sinnvolle Ergebnisse liefern konnte.

Die Hyperparameter wurden mit den gleichen Komponenten umgesetzt wie für den Entscheidungsbaum und die logistische Regression.

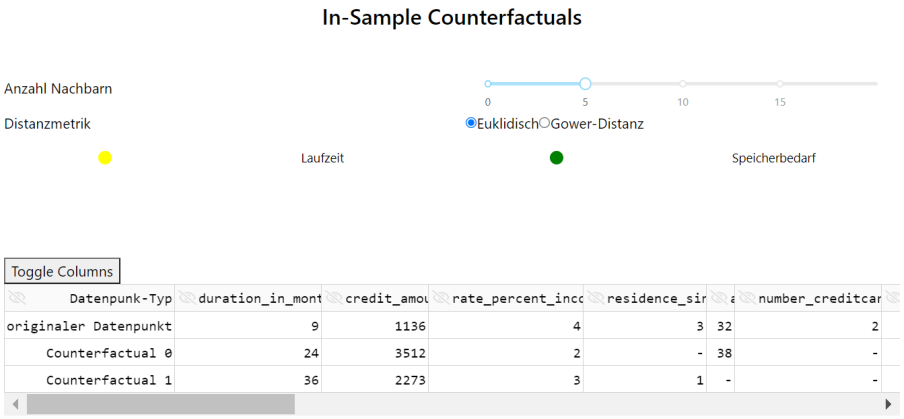


Abbildung 4.11: Hier sind die in der Oberfläche verstellbaren Hyperparameter für die Berechnung der In-Sample CFs mithilfe des kNN gezeigt. Die Anzahl der Nachbarn kann aus einem Bereich von 0 bis 100 mit einem Slider gewählt werden, die Distanzmetrik als eine von zwei Optionen über Radio Items. Unten ist zudem die Ergebnistabelle gezeigt, die den gewählten Datenpunkt und alle CFs, die unter den Nachbarn gefunden wurden, ausgibt. Es sind außerdem für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

Der erstellte kNN-Klassifizierer stellt eine Methode bereit, mit der unter Verwendung der jeweiligen Entfernungsmetrik alle Nachbarn und ihre Abstände zum zu erklärenden Datenpunkt bestimmt werden können. Aus diesen werden dann diejenigen Nachbarn ausgesucht, deren Klasse sich von der des zu erklärenden Punktes unterscheidet. Der Datenpunkt selbst und alle gefundenen und passenden Counterfactuals werden in einer Tabelle untereinander ausgegeben, sodass ein direkter Vergleich möglich ist. In den Zellen, deren Werte sich nicht von denen des originalen Beispiels unterscheiden, werden diese Werte bei den entsprechenden CFs durch einen Querstrich ersetzt. Falls sich für eine Eigenschaft keinerlei Unterschiede in allen CFs zum Original feststellen lassen, wird die Eigenschaft vollständig aus der Visualisierung entfernt, da sie für diesen Fall keinen Einfluss auf die Klassifizierung hatte. So kann der Nutzer direkt ablesen, welche Eigenschaften bzw. Eigenschaftskombinationen in diesen Counterfactuals zu einer veränderten Klassifizierung geführt haben.

Die Ergebnisse der Berechnungen wurden als **Dash DataTable** realisiert. Diese müssen als Eingabe **pandas.DataFrame**s erhalten. Bei diesen DataFrames handelt es sich um eine

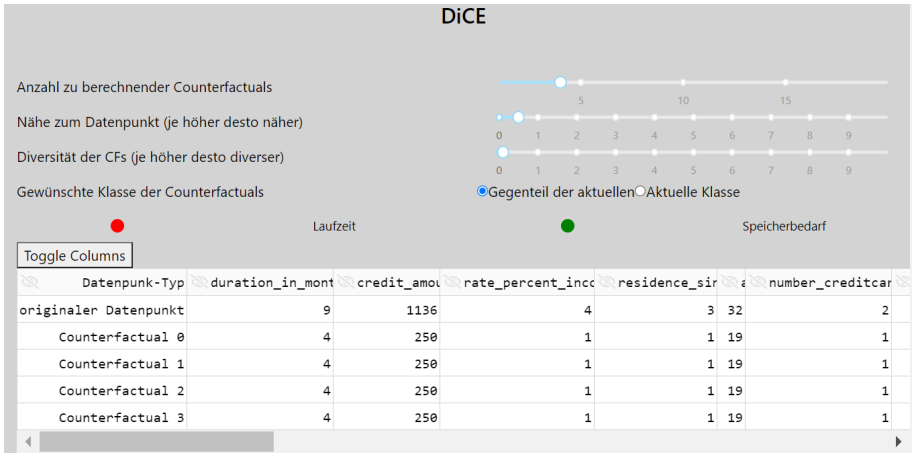
Art **Dictionary**, das die Informationen für einzelne Zeilen und Spalten der Tabelle enthält. Hierfür wurde für jedes bestimmte Counterfactual ein neuer Eintrag erstellt und dieser an den Dataframe angehängt. Auf diese Weise wurde je CF eine Zeile ergänzt. Anschließend an die Berechnung wurden für einen leichteren Vergleich mit dem gewählten Datenpunkt dessen Werte in die erste Zeile der Tabelle eingefügt und alle Zeilen beschriftet.

Eine Besonderheit stellt die Verwendung der Gower-Distanz als Distanzmetrik dar. Diese ist kein von sklearn bereitgestelltes Maß sondern musste gesondert eingebunden werden. Hierfür war zu beachten, dass die Metrik-Funktion in Ein- und Ausgabe mit den vom kNN erwarteten Werten übereinstimmten. Die verwendete Bibliothek, welche eine Implementierung der Gower-Distanz bereitstellt, berechnet eine Matrix und gibt diese zurück. Dies hat dazu geführt, dass eine neue Funktion erstellt werden musste und dass sowohl die Ein- als auch die Ausgaben an die Erwartungen von kNN an eine Metrik-Funktion angepasst werden mussten. Erwartet werden zwei Datenpunkte als Eingabe und die Distanz zwischen diesen beiden. Daher wird zunächst die gesamte Matrix, welche die Distanzen zwischen allen Datenpunkten untereinander bestimmt, berechnet und anschließend der entsprechende Abstand für die beiden Datenpunkte selektiert und ausgegeben.

4.2.4 Visualisierung der DiCE Ausgaben

Für die Implementierung des DiCE-Algorithmus wurde die interpretML-DiCE-Bibliothek verwendet. Diese stellt Funktionen zur Erstellung einer „DiCE-Erklärungsinstanz“ bereit, welche die Datenmenge und das zur erklärende Modell benötigt. Aus den übergebenen Daten kann dann für einen bestimmten Datenpunkt eine vom Nutzer wählbare Anzahl an CFs generiert werden. Zudem ist es möglich, die Gewichtung der einzelnen Teile der Optimierungsfunktion (Nähe zum zu erklärenden Datenpunkt bzw. Diversität der generierten CFs) einzustellen. Diese korrespondieren direkt mit den Hyperparametern λ_1 (gewichtet die Nähe zum Beispiel) und λ_2 (gewichtet die Diversität). Je höher die Werte sind, desto stärker beeinflussen die einzelnen Summanden die kombinierte Optimierungsfunktion (siehe 3.14).

Anhand dieser Werte können die Counterfactuals berechnet und ausgegeben werden. Anschließend wird die ursprüngliche Ausgabe in eine Tabelle umgewandelt. Eigenschaften, die sich nicht von denen des originalen Beispiels unterscheiden, werden auch hier mit einem Querstrich markiert. Falls sich für eine Eigenschaft keinerlei Unterschiede in allen CFs zum Original feststellen lassen, wird die Eigenschaft vollständig aus der Visualisierung entfernt, da sie für diesen Fall keinen Einfluss auf die Klassifizierung hatte. Die Tabelle enthält außerdem auch den Ausgangsdatenpunkt, sodass der Nutzer alle Eigenschaften der CFs und des Beispiels miteinander vergleichen kann.



Datenpunk-Typ	duration_in_mont	credit_amou	rate_percent_incc	residence_sir	a	number_creditcar
originaler Datenpunkt	9	1136	4	3	32	2
Counterfactual 0	4	250	1	1	19	1
Counterfactual 1	4	250	1	1	19	1
Counterfactual 2	4	250	1	1	19	1
Counterfactual 3	4	250	1	1	19	1

Abbildung 4.12: Es sind die verstellbaren Hyperparameter für die Berechnung der DiCE Cfs dargestellt. Die Anzahl der zu berechnenden CFs kann aus einem Bereich von 0 bis 20 mit einem Slider gewählt werden, die Nähe zum Datenpunkt sowie die Diversität haben eine Auswahl zwischen 0 und 10. Die gewünschte Klasse (Gegenteil entspricht CFs, aktuelle Klasse entspricht allen anderen Datenpunkten, die dieselbe Klasse haben, damit aber keine CFs darstellen) kann aus zwei Radio Items gewählt werden. Unten ist zudem die Ergebnistabelle gezeigt, die den gewählten Datenpunkt und alle berechneten CFs ausgibt. Es sind außerdem für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

Um die DiCE Ausgaben in der Oberfläche verwenden zu können, musste eine neue Methode implementiert werden. Die interne Methode, auf der diese basiert, unterstützt ausschließlich Ausgaben in die Konsole und konnte daher nicht verwendet werden. Die verschiedenen Fälle der originalen Methode wurden beibehalten und jeweils an Stelle der prints durch Verwendung der **append**-Methode eine **pandas.DataFrame** Ausgabe zusammengestellt. Mit dieser konnte anschließend ebenso verfahren werden wie für die Erstellung der Tabelle der In-Sample CFs.

4.2.5 Visualisierung der DeepSHAP Ergebnisse

Die Implementierung des DeepSHAP Algorithmus nutzt die SHAP Bibliothek [24] und die von dieser bereitgestellten Funktionen. Mithilfe der Bibliothek kann der DeepExplainer für PyTorch Modelle erstellt werden. Dieser benötigt das zu erklärende Netz und die Datenmenge, für die Erklärungen berechnet werden sollen. Anschließend können von diesem Objekt mithilfe einer Funktion die SHAP-Values bestimmt und so eine Erklärung für alle Datenpunkte sowohl für die Klassifizierung als Klasse **0** (der Kredit wurde abgelehnt) als auch als Klasse **1** (der Kredit wurde genehmigt) berechnet werden. Da diese Ausgabe automatisch Erklärungen für alle Datenpunkte enthält, ist es ausreichend, sie nur beim initialen Laden aller Algorithmen einmalig auszuführen. Anschließend kann auf die SHAP Values (je für Klasse 0 und Klasse 1) eines Datenpunktes mit seinem entsprechendem Index

zugegriffen werden. Auf diese Weise wird der Zugriff zur Laufzeit schneller und es entstehen kaum Wartezeiten für den Nutzer.

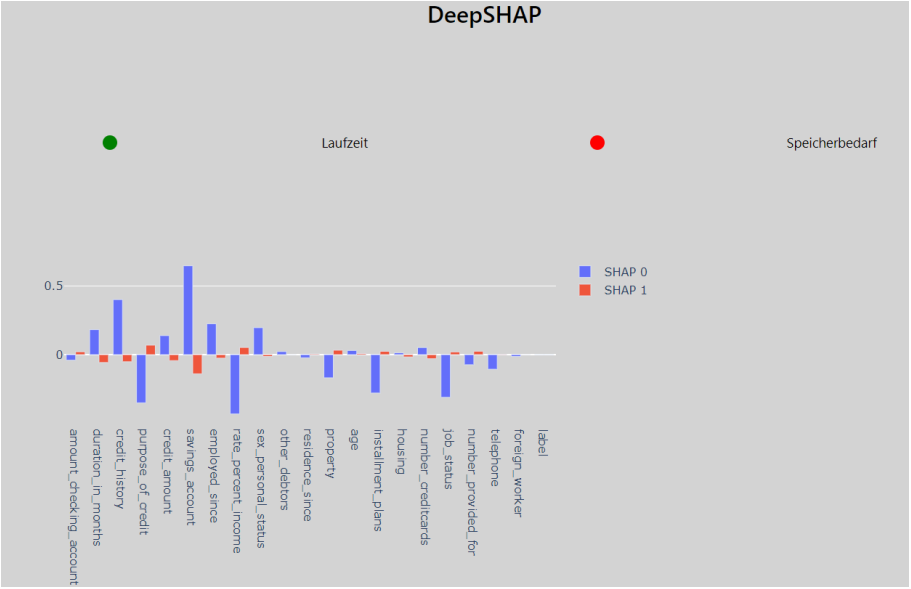


Abbildung 4.13: Es ist das Balkendiagramm der berechneten DeepSHAP-Ergebnisse visualisiert. In blau sind die Ergebnisse für Klasse 0, in rot die für Klasse 1 angegeben. Diese sind jeweils mit SHAP0 bzw. SHAP 1 bezeichnet. Die x-Achse ist mit den entsprechenden Eigenschaften, die y-Achse mit den Werten dieser beschriftet. Es sind außerdem für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

Für Shapley Values gibt es außer der Wahl des Datenpunktes, der erklärt werden soll, keine weiteren einstellbaren Variablen. Die bereitgestellte Methode kann als Argumente lediglich die zur Berechnung notwendigen Datenpunkte sowie optionale Hyperparameter, welche die Sortierung der Ausgabe verändern, erhalten. Da die verwendete Basisimplementation keine Variablen übergeben bekommt, mit denen die eigentliche Berechnung beeinflusst werden kann, können dem Nutzer keine weiteren Interaktionsoptionen geboten werden.

Die berechneten Werte werden für den gewählten Datenpunkt als Balkendiagramm mit den beiden Klassen und den entsprechenden berechneten Wichtigkeiten für alle Eigenschaften dargestellt. Dabei sind hier die Balken in ihrem Vorzeichen fast immer verschieden. Dies liegt an der Annäherung an die tatsächlichen Shapley Values. Aus exakten Berechnungen würde sich ein immer genau gegenteiliges Ergebnis für beide Klassen ergeben. Je stärker ein Wert (bei Verwendung der Annäherung) positiv ist, desto wahrscheinlicher ist es, das der für die entgegengesetzte Klasse berechnete Wert, ein anderes Vorzeichen hat. Sind die SHAP Werte nur sehr klein, kann es jedoch auch vorkommen, dass beide positiv oder beide negativ sind.

Für die Darstellung der Ausgaben des Deep Explainers wurde ein **Barplot** gewählt. Dieser zeigt die jeweils errechneten Wichtigkeiten für eine Entscheidung für Klasse **0** und für eine Entscheidung für Klasse **1** durch zwei Balken je Eigenschaft in verschiedenen Farben an.

4.2.6 Visualisierung der LRP Ausgaben

Für LRP wurde auf Basis des zum Paper von Montavon et al. [32] zugehörigen Tutorials [3] eine eigene Implementierung erstellt. Diese ist in der grundlegenden Struktur an die des Beispiels angelehnt: Zunächst wird die Berechnung in Klassifizierungsrichtung angestoßen um für alle Neuronen die jeweiligen Werte für das zu erklärende Beispiel zu erhalten. Anschließend kann der Rückwärtsschritt mit der jeweiligen gewählten Regel ausgeführt werden. Hierbei ist zu beachten, dass dieser Schritt zum einen nicht bis zur Eingabeschicht ausgeführt werden kann, der eine eigene abschließende Berechnung benötigt. Dies liegt daran, dass die Eingabeschicht keine Inputs mehr hat, da die Neuronen die Ausgangswerte des Datenpunktes enthalten. Daher ist an dieser Stelle keine Regel verwendbar, die unter Berücksichtigung der Neuroneninputs die Relevanzen berechnet. Zudem können die zur Wahl gestellten Regeln nur für bestimmte Schichtentypen verwendet werden. Da das verwendete Netz eine Softmax- und eine Dropout-Schicht besitzt, muss für diese eine gesonderte Regel, die Übernahme der Ergebnisse der vorherigen Schicht, bestimmt werden. Nach der erfolgreichen Berechnung des Rückwärtsschritts kann als Letztes die Berechnung für die Eingabeschicht durchgeführt werden. Dieser setzt die w^2-Regel um, die für die Berechnung auf der Input-Schicht verwendet wird. Der letzte Schritt gibt die Ergebnisse für die einzelnen Eigenschaften aus, die anschließend als Balkendiagramm dargestellt werden.

Die gesamte LRP-Berechnung wurde für diese Arbeit selbst implementiert. Hierfür wurden im Wesentlichen drei Methoden umgesetzt:

- der Vorwärtslauf, in dem für jede Schicht des Netzes die zugehörigen Ergebnisse der **Autograd**-Berechnung mit abgespeichert werden
- der Rückwärtsschritt mit Ausnahme der Input-Schicht, in dem die eigentlichen Relevanzen berechnet werden
- der letzte Schritt, der die Relevanzen der Eingabeschicht gesondert bestimmt

Der Vorwärtsschritt ist hierbei lediglich der Aufruf der **forward**-Methode des Netzes. Dabei muss zusätzlich die integrierte Autograd-Berechnung eingeschaltet werden.

Für den Rückwärtsschritt wird über alle Schichten des Netzes mit Ausnahme der ersten (rückwärts) iteriert. In dieser Schleife wird geprüft, um was für eine Schicht es sich bei der aktuell betrachteten handelt (hier muss zum Beispiel die Dropout-Schicht übersprungen

[3] `heatmapping.org`

werden). Entsprechend der jeweiligen Schicht wird die ausgewählte LRP-Regel verwendet und die berechneten Relevanzen gespeichert. Wird eine Schicht „übersprungen“, wird der Wert der vorherigen Schicht übernommen. Ist der Rückwärtsschritt beendet, existieren für alle Neuronen aller Schichten die entsprechenden Relevanzen, mit Ausnahme der Eingabeschicht. Für diese muss gesondert eine andere LRP-Regel verwendet werden. In dieser Implementierung wird die w^2-Regel verwendet.

Die einzelnen Schritte zur Relevanzberechnung entsprechen dabei denen, die schon in der Theorie für LRP (3.3.2) erklärt wurden. Sie wurden in Anlehnung an die Berechnungsschritte der Theorie wie folgt bestimmt:

Zunächst wird $z_k = \sum_j a_j w_{jk}$ als Ergebnis des Vorwärts-Schritts bestimmt. Anschließend kann der gesamte Quotienten $\frac{R_k}{z_k}$ berechnet und das Ergebnis abgelegt werden. Dieses wird im dritten Schritt mit z_k multipliziert, wobei es sich hier um eine Elementenweise Multiplikation mit anschließender Summierung handelt. Mit diesem Ergebnis wird die **backward**-Methode aufgerufen und so die Relevanz eine Schicht in Richtung der Eingabeschicht propagiert. Dank der Abspeicherung des Gradienten aus dem Vorwärtsschritt kann c (siehe 3.25-3.27) über den Aufruf von **grad** aus den errechneten Werten ausgelesen werden. Die gesamten Relevanzen ergeben sich nun durch multiplizieren der propagierten Werte mit c. Die gewählten LRP-Regeln haben in dieser Berechnung Einfluss auf die Berechnung von z, wie sich bereits aus den Formeln schließen lässt.

Für die erste Schicht des Netzes ist die Grundidee ähnlich, jedoch wird an dieser Stelle eine andere Regel verwendet. Statt der Verwendung der Variable z reicht es hier aus, die Gewichte w zu betrachten, die lediglich quadriert werden müssen. Alle weiteren Schritte bleiben essentiell gleich.

Das Balkendiagramm der berechneten Ausgaben kann anschließend ähnlich wie für die DeepSHAP-Erklärung verstanden werden. Bei LRP wird jedoch nur die Wichtigkeit der einzelnen Eigenschaften für eine Klassifizierung als Klasse **1** (der Kredit wurde genehmigt) angegeben, da die Ergebnisse für Klasse **0** (der Kredit wurde abgelehnt) die gleichen, lediglich mit einem anderen Vorzeichen sind. Die Höhe eines Balkens gibt dabei an, wie wichtig eine Eigenschaft für die Entscheidung als Klasse **1** war. Negative Werte zeigen hier ebenfalls einen negativen Einfluss der Eigenschaft auf die Entscheidung an.

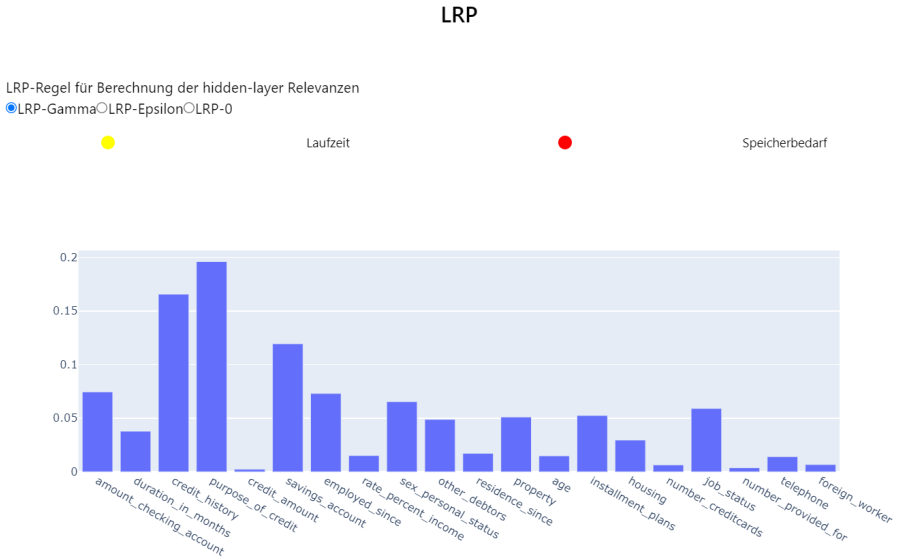


Abbildung 4.14: Hier sind die verstellbaren Hyperparameter für die Berechnung der LRP-Werte angezeigt. Die LRP-Regel, die genutzt werden soll, kann aus drei Radio Items ausgewählt werden. Die Ergebnisse der mit LRP berechneten Relevanzen sind darunter als Balkendiagramm angezeigt. Die x-Achse ist mit den entsprechenden Eigenschaften, die y-Achse mit den Werten beschriftet. Es sind außerdem für die theoretische Laufzeit und den theoretischen Speicherbedarf Einschätzungen angegeben.

4.3 Konvertierungen für bessere Visualisierung

Für die Visualisierungen im Allgemeinen musste zunächst aus den codierten und vorverarbeiteten Werten des NNs wieder lesbare Werte erzeugt werden. Hierfür musste die Vorverarbeitung durch Invertieren rückgängig gemacht werden. Der dafür verwendete Transformierer setzt sich aus einem **StandardScaler** für quantitative und einem **OneHotEncoder** für qualitative Eigenschaften zusammen. Beide werden von der sklearn Bibliothek bereitgestellt und besitzen eine Methode zum Invertieren der Codierungen. Mit dieser können sie sowohl für bekannte transformierte Datenpunkte als auch für neue Werte die Vorverarbeitung umkehren und für einen Nutzer verständliche Werte zurückberechnen. Hierfür musste lediglich darauf geachtet werden, dass die korrekten Eigenschaften an den passenden Transformierer übergeben werden.

Die Erzeugung der DataFrames sowie die anschließende Tabellenerstellung bot ebenfalls einige Herausforderungen: Die Datenpunkte sollten hierfür so aufgearbeitet werden, dass nicht nur lesbare und verständliche Werte angezeigt werden, sondern zusätzlich unwichtige (als zum originalen Datenpunkt *nicht* unterschiedliche) Werte durch einen Querstrich ersetzt werden. Eigenschaften, bei denen sich für keins der CFs eine Änderung zum originalen Wert ergab, wurden vollständig entfernt.

Kapitel 5

Diskussion, Ausblick und Fazit

5.1 Zusammenfassung

In dieser Arbeit wurden sechs verschiedene Erklärbarkeitsmethoden, von denen je zwei einem generellen Ansatz angehören, implementiert und in einer Oberfläche visualisiert. Die Implementierung ist auf einen festen Datensatz mit nominalen und reellwertigen Daten beschränkt und zeigt, dass ein allgemeiner Vergleich in einer Anwendung realisierbar ist. Dabei ist die Einschränkung auf einen Datensatz nicht zur Umsetzung notwendig, sondern ist dem Zeitrahmen der Arbeit geschuldet. Zugleich ermöglichen die Einfachheit des Datensatzes einen Einblick und Überblick über die verwendete Datengrundlage. Mögliche Erweiterungen sind Teil des Ausblicks im folgenden Unterkapitel.

Bei der Wahl der Algorithmen wurde darauf geachtet, dass diese zwar verschiedenen Ansätzen angehören, jedoch die verwendeten Datentypen vergleichen können. Des Weiteren sollten alle verwendeten Bibliotheken in Python implementiert und relativ gut in die Dash-Oberfläche einzubinden sein.

Die realisierte Oberfläche setzt die gestellten Anforderungen um. Sie visualisiert sechs Erklärbarkeitsverfahren und stellt mindestens eine , in den meisten Fällen jedoch mehrere Interaktionsmöglichkeiten bereit. Die Auswirkungen von Anpassungen an Parametern werden an der Oberfläche unmittelbar dargestellt. Durch das Anzeigen der Visualisierungen aller sechs Algorithmen erhält der Nutzer außerdem die angedachten Vergleichsmöglichkeiten.

Das Design und der Aufbau der Oberfläche wurden mit Hinblick auf den Fokus der Arbeit - das Vergleichen der verschiedenen Erklärungen - erstellt. Ursprünglich sollten, zur individuellen Anpassbarkeit, neben der Auswahl der Datenpunkte noch die Erklärbarkeitsverfahren separat aktivierbar sein. Zugunsten der Vergleichbarkeit und Steigerung der Interaktion wurde dieser Ansatz allerdings verworfen.

Beibehalten wurde die Datenpunktauswahl. Es wurde eine zusätzliche Übersicht ergänzt, die neben der Darstellung der Werte des gewählten Punktes zusätzliche Metadaten

anzeigt. Es werden der größte, der kleinste und der Durchschnittswert für alle quantitativen und der häufigste Wert für alle qualitativen Eigenschaften dargestellt. Diese Anzeige soll dem Benutzer die Möglichkeit geben, den Datenpunkt selbst in der Datenmenge einzuordnen, ohne dafür die Erklärbarkeitsverfahren zu betrachten. Sie soll so die Fähigkeit des Nutzers, Erklärungen einzuordnen, erhöhen.

Bei den ausgewählten veränderbaren Hyperparametern wurden hauptsächlich solche gewählt, die leicht verständlich und gut in die zum Algorithmus zugehörigen Formeln einzuordnen sind. So werden einem Nutzer Optionen geboten, tiefere Einsicht in die einzelnen Algorithmen zu nehmen. Für den Entscheidungsbaum und die logistische Regression werden die Modelle bei Veränderung der Hyperparameter neu berechnet und somit die Genauigkeit jedes Mal neu bestimmt. Da die Hyperparameter das Modell beeinflussen, dieses aber nicht direkt dargestellt wird, sind die direkten Auswirkungen für diese beiden Ansätze nicht so leicht ersichtlich. Lediglich die Veränderung des tatsächlichen Ergebnisses und der Genauigkeit kann eingesehen werden.

Für die In-Sample Counterfactuals und DiCE werden zwar auch neue Berechnungen angestoßen und die Hyperparameter beeinflussen diese, es kann jedoch keine Genauigkeit angegeben werden. Dafür sind die Einflüsse der Hyperparameter leichter am Ergebnis ablesbar. Das Erhöhen der Zahl der Nachbarn bzw. CFs erhöht die Anzahl der ausgegebenen Counterfactuals. Lediglich die Wahl der Distanzmetriken ist ebenfalls weniger einsehbar. Die DiCE-Hyperparameter bestimmen, wie wichtig Diversität und Nähe zum ursprünglichen Datenpunkt sind und können direkt aus den Werten der berechneten CFs im Vergleich zum Datenpunkt bzw. anderen Counterfactuals abgelesen werden.

Bei DeepSHAP und LRP verhält es sich ähnlich wie bei den Surrogatmodellen. Die interne Berechnung ist schlecht für den Nutzer einsehbar und nur die Ergebnisse werden angezeigt. Da für DeepSHAP keine Parameter zur Verfügung stehen, ist hier nicht viel Interaktivität vorhanden. Für LRP kann die verwendete LRP-Regel angepasst werden. Da jedoch nur die Relevanzen der Eingabeschicht visualisiert sind, sind die internen Änderungen weiterhin eher schlecht einsehbar.

5.2 Probleme

Bei der Implementierung bestand die Herausforderung vor allem darin, für jeden Algorithmus aufs Neue die tatsächliche Verbindung mit dem Datensatz und dem verwendeten Netz herzustellen. Die bereitgestellten und ausgegebenen Daten mussten für jeden Algorithmus individuell so angepasst werden, dass diese den jeweiligen Anforderungen entsprachen. Für die von sklearn bereitgestellten Bibliotheken war ein einfaches Verändern der *shape*, sowie das Einbinden der Klassifizierungen des Netzes an Stelle der tatsächlichen Label

ausreichend. Für das Einbauen der Gower-Distanz musste zusätzlich eine eigene Funktion implementiert werden, die die Ein- und Ausgabekonventionen des kNN berücksichtigt. Der DiCE Algorithmus benötigte ein Dictionary, das die Eigenschaftsnamen und die jeweiligen Werte direkt miteinander in Verbindung setzt. Für den DeepSHAP Algorithmus musste neben den Datenpunkten zudem das trainierte Modell übergeben werden. Hierfür war es nötig, eine separate forward-Funktion zu implementieren, da das ursprüngliche Modell ausschließlich auf Tensoren rechnete. Mit der neuen Methode können jedoch auch die von DeepSHAP verwendeten numpy Arrays als Eingaben genutzt werden. LRP wurde vollständig selbst implementiert, hatte jedoch keine besonderen Anforderungen an die benötigten Eingaben.

Eine weitere sehr große Hürde stellte das Aufbauen der Visualisierungen und vor allem das Invertieren der Vorverarbeitung dar. Hierfür musste je nach Modell eine sehr individuelle Lösung und Aufbereitung implementiert werden, sodass die Ausgaben und Ergebnisse im resultierenden Graph korrekt dargestellt und verständlich sind.

Zudem wurden für diese Arbeit zwei zusätzliche Counterfactual-Algorithmen in Betracht gezogen, die jedoch wieder verworfen werden mussten. Dabei handelt es sich um die Counterfactual-Berechnung von Fat Forensics, deren Implementierung zu stark auf einem Brute-Force Verfahren und einer hart eingestellten Grenze beruhten, sowie Alibi Anchors, die jedoch ausschließlich Ausgaben für die gleiche Klasse bestimmen (anstatt CFs, die eine gegenteilige Klasse haben). Zur besseren Vergleichbarkeit wurden daher anschließend In-Sample-Counterfactuals ausgewählt. Diese stellen einen Ansatz dar, der passende Ergebnisse (der gegenteiligen Klassifizierung) ausgibt, sich jedoch in der grundlegenden Bestimmung stark vom gewählten DiCE-Algorithmus unterscheidet (siehe Unterkapitel 3.2.1 und 3.2.2).

5.2.1 Diskussion der Ergebnisse

Das Programm stellt die gewünschten Funktionalitäten zur Verfügung und visualisiert berechnete Erklärungen in einer für einen Nutzer verständlichen Aufbereitung.

Die implementierte Anwendung ermöglicht den Vergleich verschiedener Erklärbarkeitsverfahren und bietet Interaktionsmöglichkeiten. Es ist möglich, einen Datenpunkt aus einer gegebenen Menge auszuwählen. Die Auswahl dieses Datums aktualisiert alle Visualisierungen, sodass sie für diesen die entsprechende Erklärung anzeigen. Dabei muss jedoch beachtet werden, dass die Surrogatmodelle, DeepSHAP und LRP jeweils nur Annäherungen berechnen. Diese geben keine feststehende, unveränderliche Erklärung an, sondern ändern sich mit jeder Berechnung. Für den Entscheidungsbaum und die logistische Regression wird eine neue Berechnung erst mit der Veränderung eines Hyperparameters, nicht jedoch durch Auswahl eines neuen Datenpunktes angestoßen. Die Counterfactual Algorithmen

sind beispielbasierte Erklärungen und liefern daher inhärent keine feststehenden Erklärungen, sondern lediglich Beispiele.

Die verschiedenen Erklärungen werden angezeigt und können auf einer Seite miteinander verglichen werden. Wie viel Nutzen ein solcher Vergleich einem Anwender bringt kann jedoch nicht pauschal bestimmt werden. Vor allem für einen Informationsgewinn durch die Kombination mehrerer Algorithmen ist das subjektive Empfinden, welche Erklärungen hilfreich sind, ausschlaggebend: Ein Anwender hat keinen Benefit, wenn nicht mindestens eine nützliche Erklärung unter den verschiedenen gezeigten ist. Die Chance, dass dies der Fall ist, sinkt mit der gebotenen Auswahl. Dies führt jedoch nicht zwangsläufig dazu, dass jeder Nutzer eine hilfreiche Erklärung erhält. Das Finden einer einzelnen hilfreichen Erklärung genügt jedoch für tatsächlichen Informationsgewinn aus einer Kombination der Erklärungen nicht. Um einen Vergleich vornehmen zu können müssen mindestens zwei der Algorithmen als verständlich und hilfreich eingestuft werden. Erst dann kann eine Erklärung durch Ergebnisse anderer Algorithmen zusätzlich validiert oder sogar invalidiert werden.

Hinzu kommt, dass die Erklärungsalgorithmen selbst eine Black Box darstellen. In diese kann durch die Option, Hyperparameter zu verändern, ein kleiner Einblick geschaffen werden. Dieser sorgt jedoch nicht für ein vollständiges Verständnis der Erklärungsgewinnung selbst. Zudem ist auch hierfür notwendig, dass ein Anwender aus dem Interagieren mit den Hyperparametern für ihn hilfreiche Informationen gewinnen kann. Dies setzt jedoch teilweise (zum Beispiel bei LRP) tiefer greifendes Vorwissen voraus.

Auch die inhärente Ungleichheit der Ansätze erschwert die Möglichkeit, Erklärungen verschiedener Grundideen miteinander zu vergleichen. Ebenso führen die generierten Ausgabenarten dazu, dass die Ergebnisse grob in zwei Gruppierungen eingeteilt werden können, was ihre Vergleichbarkeit betrifft: Die Balkendiagramme der logistischen Regression, von DeepSHAP und von LRP sind leichter miteinander vergleichbar als mit den anderen Typen. Analog dazu sind die Tabellen der CFs untereinander leichter vergleichbar. Der Entscheidungsbaum weicht stärker ab, kann jedoch eher den Tabellen mit konkreten Counterfactual Beispielen zugeordnet werden.

Da ein Pfad mit bestimmten Entscheidungen angegeben wird, kann zwar die konkrete Klassifizierung nachvollzogen werden, die Auswirkungen bestimmter Eigenschaften festzustellen ist jedoch nicht auf einen Blick möglich. Eine Kombination mit Counterfactual Ergebnissen kann mit dem Entscheidungsbaum zu einem Informationsgewinn führen. Wird ein Pfad ausgegeben, können die Eigenschaften und entsprechenden Thresholds mit den CFs verglichen werden. Dies kann dazu führen, dass aus den Konkreten Beispielen in Kombination mit dem Entscheidungsbaum nicht nur eine Erklärung für die Klassifizierung sondern auch Änderungsvorschläge gewonnen werden können, die einander gegenseitig validieren.

Erst durch das Auswählen verschiedener Datenpunkte und das Vergleichen vieler Pfade kann ein Muster entstehen, dass bestimmte Eigenschaften häufiger enthält. Dieses kann anschließend mit den Balkendiagrammen verglichen werden. Ähnlich verhält es sich mit den CF-Tabellen. Auch hier kann nach der Generierung mehrerer Erklärungen für verschiedene Datenpunkte ein besserer Einblick in bestimmte Eigenschaften gewonnen werden, der anschließend in Bezug zu Balkendiagrammen gesetzt werden kann.

Generell kann die Möglichkeit, den Datenpunkt auszuwählen und diesen einzusehen, das allgemeine Verständnis für jeden einzelnen Algorithmus erhöhen. Auch die Auswirkungen der gegebenen Hyperparameter sind gut einsehbar. Dabei bleiben interne Vorgänge der einzelnen Algorithmen jedoch eher selbst eine Black Box. Dieses Problem tritt aber generell bei Verwendung von Erklärbarkeitsmethoden auf. Durch die Kombination verschiedener Ansätze wird dem Nutzer in dieser Anwendung mehr geboten als durch eine einzelne Erklärung. Die Kombination mehrerer Algorithmen eröffnet dem Nutzer die Wahl, einer für ihn verständlichen Erklärung zu vertrauen. Zugleich ist er jedoch nicht auf die Wahl einer einzelnen Ausgabe beschränkt sondern hat auch die Option, andere Ergebnisse mit zu betrachten und sich aus allen oder mehreren gemeinsam ein Urteil zu bilden.

Das Einstellen der Hyperparameter bietet verschiedene Möglichkeiten, die Ergebnisse zu konfigurieren und zu verändern. Dies versetzt den Nutzer in die Lage, Auswirkungen einzelner Werte auf das Ergebnis direkt zu betrachten. Ob dies jedoch einen Mehrwert für das Verständnis der Entscheidung bietet, hängt stark vom Anwender und seinem Vorwissen ab. Für Personen, die lediglich bessere Einsicht in eine bestimmte Entscheidung suchen, sind Hyperparameter, welche die Berechnung verändern nicht zwingend für jeden Anwender Verständnis-fördernd.

In der aktuellen Umsetzung ist diese Oberfläche ausschließlich für den bereitgestellten Datensatz und das vor-trainierte NN verwendbar. Dieser Anwendungsfall birgt wenig allgemeinen Mehrwert, ist jedoch so weit erweiterbar, dass die Einbindung realer Anwendungsfälle denkbar ist. Zudem sind die verwendeten Verfahren auch für größere Datensätze und komplexere Netze mit anderen Schichten anwendbar.

Für die Verwendung in der Praxis zur Erklärung maschineller Entscheidungen kann eine vergleichende Oberfläche durchaus Vorteile bieten. Dabei sind jedoch weitere Aspekte, wie zum Beispiel der Datenschutz, zu beachten. Dies ist ein Problem, das durch entsprechende Anpassungen (zum Beispiel das Entfernen von In-Sample-Counterfactuals und der Datenpunktwahl) behoben werden kann. Hinzu kommt, dass ähnliche Anpassungen für die Umsetzung jeder Erklärung ohnehin beachtet werden muss und somit auch direkt für mehrere Erklärbarkeitsmethoden umgesetzt werden kann.

5.3 Fazit

Insgesamt stellt die implementierte Oberfläche eine interaktive Vergleichsmöglichkeit für die verschiedenen Algorithmen dar. Die Option, verschiedene Datenpunkte zu wählen, ermöglicht einem Benutzer bessere Einsichten in die Entscheidung als eine statische, nicht veränderbare Erklärung.

Die aktuell realisierte Anwendung stellt alle geforderten Funktionen zur Verfügung, bietet jedoch in ihrer derzeitigen Implementierung wenig allgemeinen Nährwert. Sie zeigt trotzdem, dass eine Kombination verschiedener Erklärbarkeitsverfahren Vorteile bringen kann. Gleichzeitig wird aber auch sichtbar, dass in vielen Fällen, selbst bei kleinen und übersichtlichen Datensätzen, stark vom Anwender abhängt, welcher Nutzen aus Erklärungen gezogen werden kann.

5.3.1 Ausblick

Für die Zukunft sind zahlreiche verschiedene Erweiterungen der Oberfläche vorstellbar.

Von den einzelnen Erklärbarkeitsverfahren wurde bisher jeweils eine Auswahl an Hyperparametern zur Verfügung gestellt, die größtenteils leicht verständliche Werte der Erklärungsbestimmung (zum Beispiel die Höhe des Erklärungsbaumes) verändert haben. An dieser Stelle ist eine Erweiterung auf alle verfügbaren Hyperparameter denkbar. Hier könnte zudem eine Unterscheidung zwischen einfachen „Basis-Hyperparametern“, die jedem Nutzer immer für Interaktionen angezeigt werden, und „erweiterten Hyperparameter“, die nur dann gezeigt werden, wenn dies zusätzlich aktiviert wird, erfolgen.

Auch eine Erweiterung der verwendeten Erklärbarkeitsverfahren ist gut vorstellbar. Dies wäre mit größeren und aufwändigeren Änderungen verbunden als das bloße Ergänzen weiterer Hyperparameter, ist jedoch nicht ausgeschlossen. Wenn mehr Visualisierungen hinzukommen wird der verworfene Ansatz aus der Grundidee wieder relevanter, verschiedene Visualisierungen auch ausblendbar zu machen. Ansonsten könnte sich ein Nutzer leicht von der Fülle der Visualisierungen erschlagen fühlen.

Das Ergänzen weiterer Verfahren bewirkt gleichzeitig, dass beachtet werden muss, dass der aktuell verwendete Datensatz nur tabellarische Daten beinhaltet. Neue Verfahren müssen daher entweder Daten dieses Typs verarbeiten können oder es muss zusätzliche Ergänzungen geben. Dies führt zu einer weiteren Erweiterung, dem Hinzufügen neuer nutzbarer Datensätze und somit auch anderer Datentypen (zum Beipsiel Bilder oder Audiodaten). Gibt es mehr verschiedene Daten-Arten können mehr Algorithmen eingebunden werden. Hierbei muss jedoch immer beachtet werden, dass Algorithmus und Datentyp kompatibel sind. Dies könnte durch eine Abfrage geregelt und dem Nutzer durch „Ausgrauen“ der nicht zum gewählten Datentyp passenden Algorithmen angezeigt werden.

Eine Erweiterung der verwendeten Datensätze und mit diesen der Art der Daten sorgt zugleich dafür, dass auch die Visualisierung des gewählten Datums und einer eventuellen Übersicht mit Meta- und Randdaten entsprechend angepasst werden muss.

Das allgemeine Design der Oberfläche ist stark mit den Erweiterungen verwoben. Je mehr Modelle und damit Visualisierungen implementiert werden, desto wichtiger wird die Option, diese ein- und auszublenden. Auch eine Möglichkeit zur individuellen Umsortierung ist denkbar. Ebenso kann das Einführen mehrerer Seiten eine sinnvolle Ergänzung darstellen, zum Beispiel mit einer Seite je Ansatz, sodass Vergleiche unter verschiedenen Algorithmen eines Ansatzes auf einer Seite, Vergleiche unter verschiedenen Ansätzen auf mehreren Seiten möglich sind. Je mehr Datensätze eingefügt werden, desto wichtiger wird eine Übersicht über den gewählten Datensatz. Auch diese könnte auf einer eigenen Seite umgesetzt werden.

Es bieten sich zahlreiche verschiedene Optionen, die Arbeit in Zukunft auszuweiten und für verschiedene größere Spektren anwendbar zu machen. Dabei sollte jedoch berücksichtigt werden, dass vor allem die Interaktivität, aber auch die Verbesserung des Verständnisses des Nutzers weiterhin im Fokus bleiben sollten und dieses Ziel nicht in einer größeren Auswahl von Erklärbarkeitsmodellen untergehen sollte. Alle erläuterten Erweiterungen sind dabei generell praktikabel und können in die Oberfläche integriert werden. Sie erfordern jedoch sehr unterschiedlichen Aufwand. Vor allem eine tiefgreifende Veränderung des Aufbaus der Oberfläche sowie die Erweiterung der verwendbaren Datentypen erzeugen einen sehr hohen zusätzlichen Aufwand. Das Einbinden weiterer Erklärbarkeitsmodelle hingegen ist eher weniger aufwändig und in kürzerer Zeit umsetzbar.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Agarap, Abien Fred: *Deep Learning using Rectified Linear Units (ReLU).* arXiv:1803.08375, Februar 2019.

[2] Alabassy, Bassma, Mona Safar und M. Watheq El-Kharashi: *A High-Accuracy Implementation for Softmax Layer in Deep Neural Networks.* In: *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, Seiten 1–6, April 2020.

[3] Ancona, Marco, Enea Ceolini, Cengiz Öztireli und Markus Gross: *Towards better understanding of gradient-based attribution methods for Deep Neural Networks.* arXiv:1711.06104, März 2018. arXiv: 1711.06104.

[4] Bach, Francis: *Stochastic gradient methods for machine learning*, 2013. Verfügbar unter `https://www.di.ens.fr/~fbach/fbach_sgd_online.pdf`, letzer Zugriff 06.05.2021.

[5] Banerjee, Chaity, Tathagata Mukherjee und Eduardo Pasiliao: *The Multiphase ReLU Activation Function.* In: *Proceedings of the 2020 ACM Southeast Conference*, Seiten 239–242, Tampa FL USA, April 2020. ACM.

[6] Bentley, Jon Louis: *Multidimensional binary search trees used for associative searching.* Communications of the ACM, 18(9):509–517, September 1975.

[7] Bibal, Adrien, Michael Lognoul, Alexandre de Streel und Benoît Frénay: *Legal requirements on explainability in machine learning.* Artificial Intelligence and Law, Juli 2020.

[8] Breiman, L.: *Classification and regression trees.* CRC Press LLC, 2017.

[9] Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt und Gaël Varoquaux: *API design for machine learning software: experiences from the scikit-learn project.* arXiv:1309.0238, September 2013.

[10] Burnham, Kenneth P. und David Raymond Anderson: *Model selection and multimodel inference: a practical information-theoretic approach.* Springer, New York, 2nd ed Auflage, 2002.

[11] Castro, Javier, Daniel Gómez und Juan Tejada: *Polynomial calculation of the Shapley value based on sampling.* Operations Research, 36(5):1726–1730, 2009.

[12] Choe, Junsuk und Hyunjung Shim: *Attention-Based Dropout Layer for Weakly Supervised Object Localization.* Seiten 2219–2228, 2019.

[13] Cramer, E. und U. Kamps: *Grundlagen der wahrscheinlichkeitsrechnung und statistik: Ein skript für studierende der informatik, der ingenieur- und wirtschaftswissenschaften.* Springer-lehrbuch. Springer Berlin Heidelberg, 2014.

[14] Du, Gaoming, Chao Tian, Zhenmin Li, Duoli Zhang, Yongsheng Yin und Yiming Ouyang: *Efficient Softmax Hardware Architecture for Deep Neural Networks.* In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, Seiten 75–80, Tysons Corner VA USA, Mai 2019. ACM.

[15] Frosst, Nicholas und Geoffrey Hinton: *Distilling a Neural Network Into a Soft Decision Tree.* Bari, Italy, November 2017.

[16] Goodfellow, Ian, Yoshua Bengio und Aaron Courville: *Deep Learning*, 2016. Verfügbar unter `https://www.deeplearningbook.org/`, letzter Zugriff 06.05.2021.

[17] Gower, J. C.: *A General Coefficient of Similarity and Some of Its Properties.* Biometrics, 27(4):857–871, 1971. Publisher: Wiley, International Biometric Society.

[18] Géron, Aurélien: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc., September 2019.

[19] Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti und Dino Pedreschi: *A Survey of Methods for Explaining Black Box Models.* ACM Computing Surveys, 51(5):93:1–93:42, August 2018.

[20] Hsieh, Tsung-Yu, Yasser EL-Manzalawy, Yiwei Sun und Vasant Honavar: *Compositional Stochastic Average Gradient for Machine Learning and Related Applications.* arXiv:1809.01225, September 2018. arXiv: 1809.01225.

[21] James, Gareth, Daniela Witten, Trevor Hastie und Robert Tibshirani: *ISLR Seventh Printing An Introduction to Statistical Learning: with Applications in R.* Springer Texts in Statistics. Springer, New York, NY, 2013.

[22] Jameson, Anthony und John Riedl: *Introduction to the Transactions on Interactive Intelligent Systems.* ACM Transactions on Interactive Intelligent Systems, 1(1):1–6, Oktober 2011.

[23] Lapuschkin, Sebastian: *Opening the machine learning black box with Layer-wise Relevance Propagation.* 2019. Accepted: 2019-01-30T10:00:47Z.

[24] Lundberg, Scott M. und Su-In Lee: *A Unified Approach to Interpreting Model Predictions.* In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Band 30 der Reihe *NIPS'17*, Seiten 4765–4774, Long Beach, California, USA, 2017.

[25] Martinez, Manuel und Rainer Stiefelhagen: *Taming the Cross Entropy Loss.* In: Brox, Thomas, Andrés Bruhn und Mario Fritz (Herausgeber): *Pattern Recognition*, Lecture Notes in Computer Science, Seiten 628–637, Cham, 2019. Springer International Publishing.

[26] Maurus, Michael, Jan Hendrik Hammer und Jürgen Beyerer: *Realistic heatmap visualization for interactive analysis of 3D gaze data.* In: *Proceedings of the Symposium on Eye Tracking Research and Applications*, Seiten 295–298, Safety Harbor Florida, März 2014. ACM.

[27] Möller, Kornelia: *Konstruktivistische Sichtweisen für das Lernen in der Grundschule?* In: Rossbach, Hans-Günther, Karin Nölle und Kurt Czerwenka (Herausgeber): *Forschungen zu Lehr- und Lernkonzepten für die Grundschule*, Jahrbuch Grundschulforschung, Seiten 16–31. VS Verlag für Sozialwissenschaften, Wiesbaden, 2001.

[28] Mohri, Mehryar, Afshin Rostamizadeh und Ameet Talwalkar: *Foundations of Machine Learning, second edition.* MIT Press, Dezember 2018.

[29] Molnar, Christoph: *Interpretable Machine Learning.* 2019. Verfügbar unter `https://christophm.github.io/interpretable-ml-book/`, letzter Zugriff 06.05.2021.

[30] Montavon, Grégoire, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek und Klaus-Robert Müller: *Layer-Wise Relevance Propagation: An Overview.* In: Samek, Wojciech, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen und Klaus-Robert Müller (Herausgeber): *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, Band 11700, Seiten 193–209. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.

[31] Montavon, Grégoire, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek und Klaus-Robert Müller: *Explaining nonlinear classification decisions with deep Taylor decomposition.* Pattern Recognition, 65:211–222, Mai 2017.

[32] Montavon, Grégoire, Wojciech Samek und Klaus-Robert Müller: *Methods for interpreting and understanding deep neural networks.* Digital Signal Processing, 73:1–15, Februar 2018.

[33] Mothilal, Ramaravind Kommiya, Amit Sharma und Chenhao Tan: *Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations.* arXiv:1905.07697, Dezember 2019.

[34] Naqa, Issam El, Ruijiang Li und Martin J. Murphy: *Machine Learning in Radiation Oncology: Theory and Applications.* Springer, Juni 2015.

[35] Nguyen, Tung D., Kathryn E. Kasmarik und Hussein A. Abbass: *Towards Interpretable Deep Neural Networks: An Exact Transformation to Multi-Class Multivariate Decision Trees.* ArXiv, März 2020.

[36] Palczewska, Anna, Jan Palczewski, Richard Marchese Robinson und Daniel Neagu: *Interpreting Random Forest Classification Models Using a Feature Contribution Method.* In: Bouabana-Tebibel, Thouraya und Stuart H. Rubin (Herausgeber): *Integration of Reusable Systems*, Advances in Intelligent Systems and Computing, Seiten 193–218. Springer International Publishing, Cham, 2014.

[37] Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos und David Cournapeau: *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12(85):2825–2830, 2011.

[38] Rajaguru, Harikumar und Sunil Kumar Prabhakar: *E-health Design with Spectral Analysis, Linear Layer Neural Networks and Adaboost Classifier for Epilepsy Classification from EEG Signals.* In: Hemanth, D. Jude und S. Smys (Herausgeber): *Computational Vision and Bio Inspired Computing*, Lecture Notes in Computational Vision and Biomechanics, Seiten 634–640, Cham, 2018. Springer International Publishing.

[39] Ramachandran, Prajit, Barret Zoph und Quoc V. Le: *Searching for Activation Functions.* arXiv:1710.05941, Oktober 2017.

[40] Ras, Gabriëlle, Marcel van Gerven und Pim Haselager: *Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges.* In: Escalante, Hugo Jair, Sergio Escalera, Isabelle Guyon, Xavier Baró, Yağmur

Güçlütürk, Umut Güçlü und Marcel van Gerven (Herausgeber): *Explainable and Interpretable Models in Computer Vision and Machine Learning*, The Springer Series on Challenges in Machine Learning, Seiten 19–36. Springer International Publishing, Cham, 2018.

[41] Rastogi, Rajeev und Kyuseok Shim: *PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning.* Data Mining and Knowledge Discovery, 4(4):315–344, Oktober 2000.

[42] Rebala, Gopinath, Ajay Ravi und Sanjay Churiwala: *Machine Learning Definition and Basics.* In: Rebala, Gopinath, Ajay Ravi und Sanjay Churiwala (Herausgeber): *An Introduction to Machine Learning*, Seiten 1–17. Springer International Publishing, Cham, 2019.

[43] Ruder, Sebastian: *An overview of gradient descent optimization algorithms.* arXiv:1609.04747, Juni 2017. arXiv: 1609.04747.

[44] Rudin, Cynthia: *Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead.* Nature Machine Intelligence, 1(5):206–215, September 2019.

[45] Ryong, Kim, Park Kyung-Hye und Young-Kuk Kim: *Improving Similarity Measurement of User's Rating Value using Sigmoid Function in Personalization System.* In: *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, Seiten 1–4, Danang Viet Nam, Januar 2016. ACM.

[46] Sachs, Michael: *Wahrscheinlichkeitsrechnung und Statistik: für Ingenieurstudierende an Hochschulen: mit 35 Bildern, 93 Beispielen und 71 Aufgaben.* Mathematik-Studienhilfen. Fachbuchverlag Leipzig im Carl Hanser Verlag, München, 5. Auflage Auflage, 2018.

[47] Sani, Habiba Muhammad, Ci Lei und Daniel Neagu: *Computational Complexity Analysis of Decision Tree Algorithms.* In: Bramer, Max und Miltos Petridis (Herausgeber): *Artificial Intelligence XXXV*, Lecture Notes in Computer Science, Seiten 191–197, Cham, 2018. Springer International Publishing.

[48] Sazli, Murat H: *A BRIEF REVIEW OF FEED-FORWARD NEURAL NETWORKS.* Communications Series A2-A3: Physical Sciences and Engineering, 50:11–17, 2006.

[49] Schmidt, Mark, Reza Babanezhad, Mohamed Osama Ahmed, Aaron Defazio, Ann Clifton und Anoop Sarkar: *Non-Uniform Stochastic Average Gradient Method for Training Conditional Random Fields.* arXiv:1504.04406, April 2015. arXiv: 1504.04406.

[50] Shrikumar, Avanti, Peyton Greenside und Anshul Kundaje: *Learning Important Features Through Propagating Activation Differences.* arXiv:1704.02685, Oktober 2019.

[51] Skansi, Sandro: *Machine Learning Basics.* In: Skansi, Sandro (Herausgeber): *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*, Undergraduate Topics in Computer Science, Seiten 51–77. Springer International Publishing, Cham, 2018.

[52] Strumbelj, Erik und Igor Kononenko: *An Efficient Explanation of Individual Classifications using Game Theory.* The Journal of Machine Learning Research, 11, 2010.

[53] Tan, Sarah, Rich Caruana, Giles Hooker, Paul Koch und Albert Gordo: *Learning Global Additive Explanations for Neural Nets Using Model Distillation.* Dezember 2018.

[54] Tripathy, Rohit K. und Ilias Bilionis: *Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification.* Journal of Computational Physics, 375:565–588, Dezember 2018.

[55] Wachter, Sandra, Brent Mittelstadt und Chris Russell: *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR.* SSRN Electronic Journal, März 2018.

[56] Wald, Ingo und Vlastimil Havran: *On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N).* In: *2006 IEEE Symposium on Interactive Ray Tracing*, Seiten 61–69, Salt Lake City, UT, USA, September 2006. IEEE.

[57] Williams, Virginia Vassilevska: *Multiplying matrices faster than coppersmith-winograd.* In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, STOC '12, Seiten 887–898, New York, NY, USA, Mai 2012. Association for Computing Machinery.

[58] Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do und Kaori Togashi: *Convolutional neural networks: an overview and application in radiology.* Insights into Imaging, 9(4):611–629, August 2018.

[59] Yang, Yinchong, Volker Tresp, Marius Wunderle und Peter A Fasching: *Explaining Therapy Predictions with Layer-Wise Relevance Propagation in Neural Networks.* 2018 IEEE International Conference on Healthcare Informatics (ICHI), Seiten 152–162, 2018.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 10. Mai 2021

Lisa Salewsky