

Fachbereich Mathematik, Naturwissenschaften und Datenverarbeitung  
Studiengang Business Mathematics

## Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Science

# Explainable Artificial Intelligence Modell-agnostische Erklärungsansätze im Vergleich

Vorgelegt von Büsra Karaoglan am 16. September 2020  
(Matrikelnummer : 5056046)

Referent Prof. Dr. Martin  
Korreferentin Prof. Dr. Lange

## **Abstract**

This thesis focuses on the explainability and better comprehensibility of black-box models. There are multiple approaches to explain the output of complex models, including model-agnostic techniques for post-hoc explainability. Currently, LIME and Kernel SHAP are among the most popular model-agnostic explainable AI methods. These methods explain predictions by local approximation of the black-box model with an interpretable model. Both methods use simplified input mapping in order to express the generated explanation in a form that human users understand. The goal is to obtain trustworthy model predictions while ensuring high prediction accuracy. The purpose of this thesis is to investigate and compare LIME and Kernel SHAP explanations. Therefore, firstly several black-box models are applied to default of credit card clients data set. Afterwards, the explanations techniques are used to explain the predictions of these complex models.

## **Zusammenfassung**

Die vorliegende Arbeit beschäftigt sich mit der Erklärbarkeit und daraus resultierenden besseren Verständlichkeit der Entscheidungen von Black-Box-Modellen. Es gibt zwar eine Vielzahl von Ansätzen, die zur Erklärung von Ergebnissen komplexer Modelle verwendet werden können, der Schwerpunkt liegt hier aber auf modellagnostischen Post-hoc-Erklärungsmethoden. Derzeit zählen LIME und Kernel SHAP zu den bekanntesten modellagnostischen Erklärungsmodellen, wobei beide Methoden für die Generierung der Erklärung das Black-Box-Modell mit einem interpretierbaren Modell lokal approximieren, um dann dieses einfache Modell zu analysieren. Um die Erklärung in einer für menschliche Benutzer verständlichen Form auszudrücken, verwenden die Methoden interpretierbare Datenrepräsentationen. Das angestrebte Ziel ist, vertrauenswürdige Modellvorhersagen zu erzielen und gleichzeitig eine hohe Vorhersagegenauigkeit zu gewährleisten. Das Hauptziel dieser Arbeit ist die Analyse und der Vergleich von LIME- und Kernel-SHAP-Erklärungen. Zu diesem Zweck werden zunächst verschiedene Black-Box Modelle auf den Datensatz Default Of Credit Card Clients angewendet und anschließend die Erklärungsmodelle verwendet, um die Entscheidungen der komplexen Vorhersagemodelle zu erklären.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Forschungsstand und Zielsetzung . . . . .	4
1.3 Aufbau der Arbeit . . . . .	5
<b>2 Methoden des maschinellen Lernens</b>	<b>7</b>
2.1 Terminologie und Notation . . . . .	7
2.2 Support Vector Machine . . . . .	7
2.3 Random Forest . . . . .	8
2.4 Künstliche neuronale Netze . . . . .	8
2.5 Klassifikationsmetriken . . . . .	9
<b>3 Grundlagen der Erklärbaren Künstlichen Intelligenz</b>	<b>11</b>
3.1 Ursprung der Erklärbarkeit . . . . .	11
3.2 Ethische, rechtliche und soziale Rahmenbedingungen . . . . .	12
3.3 Terminologie . . . . .	16
3.4 Arten von Transparenz . . . . .	17
3.5 Desiderate der Erklärbarkeit . . . . .	18
3.6 Taxonomie der Modellinterpretierbarkeit . . . . .	19
3.7 Erklärungsmethoden . . . . .	21
3.8 Erklärungsarten . . . . .	22
<b>4 Local Interpretable Model-Agnostic Explanations</b>	<b>23</b>
4.1 Einleitung . . . . .	23
4.2 Interpretierbare Repräsentation . . . . .	24
4.3 Der allgemeine LIME-Ansatz . . . . .	28
4.4 Adaption des LIME-Algorithmus an die Datentypen . . . . .	31
4.5 Submodular Pick . . . . .	38
4.6 Stärken und Schwächen des LIME-Algorithmus . . . . .	40
<b>5 Shapley Additive Explanations</b>	<b>42</b>
5.1 Einführung . . . . .	42
5.2 Shapley-Wert . . . . .	42
5.3 SHAP-Wert . . . . .	45
5.4 Kernel SHAP . . . . .	47
5.5 Vergleich von Kernel SHAP und LIME . . . . .	48
<b>6 Implementierung und Evaluation</b>	<b>49</b>
6.1 Problemstellung und Datenbeschreibung . . . . .	49
6.2 Vergleich von Black-Box Modellen . . . . .	54
6.3 Implementierung und Auswertung von Erklärungsmodellen . . . . .	56
6.3.1 Analyse von LIME- und Kernel-SHAP-Erklärungen . . . . .	57

6.3.2	Auswirkung von unterschiedlichen Kernel-Breiten auf den $R^2$ -Wert . . . . .	65
6.3.3	Globale Erklärung mit LIME und Kernel SHAP . . . . .	66

<b>7</b>	<b>Zusammenfassung und Ausblick</b>	<b>71</b>
----------	-------------------------------------	-----------

# 1 Einleitung

---

## 1.1 Motivation

In den letzten Jahren haben Systeme der Künstlichen Intelligenz einen immer größeren Einfluss auf unser tägliches Leben gewonnen, wobei die unterschiedlichen selbstlernenden Algorithmen Anwendung in zahlreichen Bereichen finden. Die Einsatzfelder der Künstlichen Intelligenz sind u.a. die Finanzdienstleistungsbranche, das Gesundheitswesen, Robotik und Produktion. Viele der eingesetzten Deep-Learning-Algorithmen sind „Black-Boxes“. Diese Algorithmen lernen in der Trainingsphase komplexe Zusammenhänge und bieten öfters sehr gute Vorhersagekraft. Nichtsdestotrotz können die Anwender den Lösungsweg nicht nachverfolgen und somit auch die Entscheidungsfindung nicht nachvollziehen. Dementsprechend wird die mangelnde Transparenz der Entscheidungsprozesse von vielen Anwendern aus Praxis und Forschung kritisiert [BB19].

Zwangsläufig wird der Einsatz der Künstlichen Intelligenz hinterfragt, und damit entstehen auch weitere Forschungsfragen. Darunter fallen Themen wie versteckter Bias oder fehlerhafte Schlussfolgerungen. Exemplarisch kann für den ersten Fall der Bankensektor betrachtet werden. Hier können „voreingenommene“ Machine-Learning-Algorithmen zu diskriminierenden Ergebnissen führen. Beispielsweise kann ein Kunde, der einen Kreditantrag stellt, aufgrund von Geschlecht oder Herkunft eine höhere Wahrscheinlichkeit eines Kreditausfalls als Vorhersage erhalten. Beim zweiten Fall, fehlerhaften Schlussfolgerungen, können in manchen Bereichen verheerende Folgen eintreten: Entscheidungsgrundlagen für z.B. medizinische Diagnostik oder autonomes Fahren können bei irrtümlichen Empfehlungen tödliche Resultate verursachen. Daher sollten die Empfehlungssysteme von Anwendern verstanden werden und vertraut werden können. Dies kann nur gelingen, wenn die Gründe für Machine-Learning-Entscheidungen transparent vermittelt werden [SM19].

Um dies zu ermöglichen, wurden bereits viele angemessene Schritte von verschiedenen Regierungs- und Aufsichtsstellen unternommen. Nicht nur die Bankenaufsicht stellt neben die Baseler Regelungen oder beispielsweise in [BF18] erste regulatorische Anforderungen, es werden auch allgemein solche Vorgaben durch die Datenschutz-Grundverordnung (DSGVO) gesetzt: Im Rahmen der neuen EU-DSGVO wird eine Art von Recht auf Auskunft und Erklärung auch über algorithmische Entscheidungen gefordert. Dies soll dazu dienen, die Relevanz der erklärbaren und interpretierbaren Algorithmen zu erkennen und hervorzuheben. Zudem führt die Forderung nach einer transparenten Informationspflicht zwangsläufig zur Frage, wie viel Informationen eine Person erhalten darf und ob dieses Wissen weitere Gefahren mit sich bringen könnte [WA19].

## 1.2 Forschungsstand und Zielsetzung

Der Wunsch nach Transparenz und Vertrauen in die Art und Weise, wie ein maschinelles Lernsystem seine Entscheidungen trifft, wird u.a. von der Forschungsbehörde des US-amerikanischen Verteidigungsministeriums DARPA (Defense Advanced Research Projects Agency) seit 2016 verstärkt untersucht. Dabei wurde erklärbare Künstliche Intelligenz (englisch: Explainable Artificial Intelligence, kurz XAI) als eins der Schwerpunktthemen gesetzt [GD16].

In den veröffentlichten Dokumenten von DARPA werden drei Hauptziele definiert. Die Entscheidungen der KI-Systeme sollen erklärt, die Stärken und Schwächen identifiziert und Verständnis für das Verhalten des Systems in Zukunft erlangt werden. Die folgende Abbildung 1.1 zeigt beispielhaft, wie so eine Erklärung und dazugehörige Vorteile aussehen könnten [GD16].

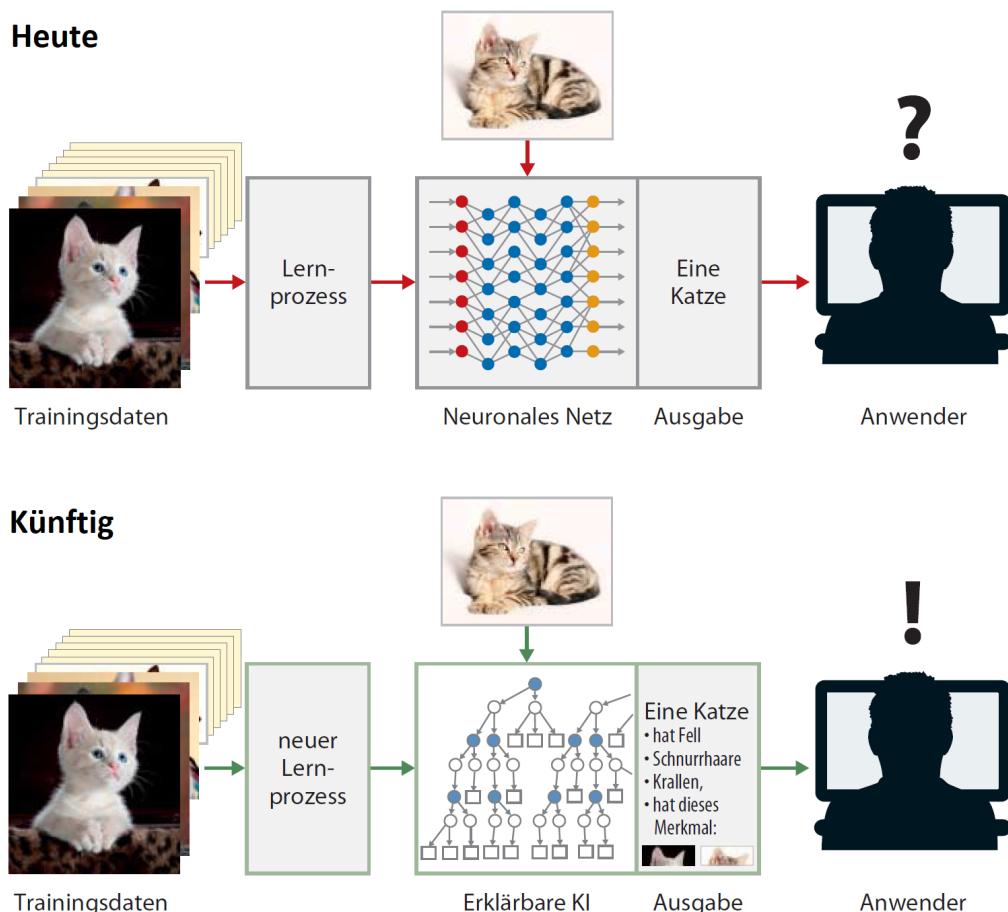


Abbildung 1.1: Konzept der erklärbaren Künstlichen Intelligenz [HA18]

Durch die Erklärbarkeit der Modelle lassen sich auch eventuelle „Clever-Hans“-Algorithmen entdecken. Dies sind Algorithmen, die zunächst einmal den Anschein von gut gelerntem Lernsystem erwecken. „Clever-Hans“, oder im Deutschen „Kluger Hans“, war ein vermeintlich mathematisch begabtes Pferd Anfang des 20. Jahrhunderts. Es konnte scheinbar die ihm von unterschiedlichen Menschen gestellten Rechenaufgaben durch Klopfen mit den Hufen mitteilen. In Wirklichkeit konnte das Pferd die Körpersprache und Mimik der Fragesteller deuten und so die Lösungen der Aufgaben finden. Demgemäß werden u.a. Machine-Learning-Ergebnisse mit ähnlichen Phänomenen, etwa dem unbemerkt Einfluss anderer Komponenten auf die Modellvorhersage, als Clever-Hans-Effekt bezeichnet [SM19].

Beispielsweise konnten Forscher mit Hilfe einiger Erklärungsmodelle für Black-Box-Modelle aus der Bilderkennung einen Clever-Hans-Effekt nachweisen. Eines der Lernverfahren hatte beim Bilderkennungssystem für die Klassen Boote und Pferde hohe Performance-Werte angezeigt. Infolgedessen wurden diese beiden Klassen näher mit geeigneten Erklärungsmodellen untersucht. Dabei konnte bewiesen werden, dass die Pferde nicht anhand der charakteristischen Merkmale klassifiziert wurden, z.B. ihrem Körperbau, sondern durch das Copyright-Zeichen der Pferde-Bilder. Zudem stellten die Forscher fest, dass der Lern-Algorithmus bei der Erkennung der Boote speziell das Merkmal Wasser berücksichtigt hatte. Mit Hilfe sogenannter Saliency Maps [SV13] konnten die Erkenntnisse auch visualisiert werden. Saliency Maps illustrieren, welche Eingabedaten bei der Bestimmung eines Resultates am einflussreichsten waren. Das Fehlen des Copyright-Vermerks bei Pferde-Bildern und Boote, die nicht in der Nähe von Wasser waren, führten bei den verwendeten Lernverfahren dementsprechend zu Fehlklassifikationen [LW19].

Das Ziel der Masterarbeit besteht zum einen darin, auf eine der aktuellsten Forschungsfragen der Künstlichen Intelligenz einzugehen. Im Zuge dieser Arbeit sollen die Lern- und Entscheidungsprozesse von ausgewählten Algorithmen der Künstlichen Intelligenz transparenter dargestellt und für Anwender erklärbar und vertrauenswürdig gemacht werden. Hierfür werden zunächst die Verfahren der Explainable AI eingegrenzt. Der Fokus dieser Arbeit liegt auf modellagnostischen – und damit auch Post-hoc – Erklärungsmodellen. Bei den modellagnostischen Erklärungsmodellen handelt es sich um erklärbare Ersatzmodelle, die auf jede Art von Machine-Learning-Modell angewendet werden können.

Eines der ersten und bekanntesten Erklärungsmodelle ist das von Marco Tulio Ribeiro, Sameer Singh und Carlos Guestrin im Jahre 2016 entwickelte LIME (Local Interpretable Model-Agnostic Explanations). Nachdem ein Vorhersagemodell trainiert wurde, generiert LIME lokale Erklärungsmodelle für einzelne Datenpunkte. Hierbei wird das Black-Box-Modell allein mit Informationen über Input- und Output-Daten lokal approximiert. Als lokale Erklärungsmodelle werden üblicherweise leicht interpretierbare Modelle, wie lineare Regression oder (kurze) Entscheidungsbäume, verwendet [RS16a].

Das zweite Erklärungsmodell ist das Kernel-SHAP-Verfahren (SHapley Additive exPlanations), das von Scott M. Lundberg und Su-In Lee im Jahr 2017 entworfen wurde. Ähnlich zu LIME wird auch hier eine lokale und interpretierbare Erklärung eines Machine-Learning-Modells erzeugt. Das Verfahren kombiniert LIME und die sogenannten Shapley-Werte mit dem Ziel, bessere Erklärungen zu liefern und Hyperparameter zu umgehen [LL17].

Die beiden Verfahren können nicht nur auf tabellenbasierte Daten, sondern auch auf Bild- und Text-Daten angewendet werden, wobei die Vorgehensweise je nach Anwendung angepasst wird. Im Rahmen dieser Masterarbeit werden die beiden Erklärungsmodelle ausführlich vorgestellt, miteinander verglichen und die jeweiligen Stärken und Schwächen analysiert. Anschließend werden die theoretisch beschriebenen Erklärungsmodelle auf den Datensatz Default Of Credit Card Clients von UCI Machine Learning Data Repository zur Vorhersage von Zahlungsausfällen bei Kreditkunden angewendet.

### 1.3 Aufbau der Arbeit

Einleitend werden in Kapitel 2 die Grundlagen von Machine Learning vorgestellt und die in dieser Ausarbeitung eingesetzten Algorithmen beschrieben. Kapitel 3 beschäftigt sich mit den theoretischen Aspekten der Arbeit. Hier werden die Begriffserklärungen für erklärbare Künstliche

Intelligenz eingeführt und detailliert erläutert. Kapitel 4 und 5 widmen sich den ausgewählten Erklärungsmodellen und ihren mathematischen Herleitungen. Im nachfolgenden Kapitel 6 werden diese Modelle dann auf einen Datensatz angewendet. Das Kapitel beinhaltet Erläuterungen bezüglich der Implementierung der Erklärungsmodelle, anschließend werden die Ergebnisse der Erklärungsmodelle umfangreich diskutiert. Im letzten Kapitel 7 werden die Erkenntnisse zusammengefasst und versucht, einen Ausblick über weiterführende Forschungsfragen zu geben.

## 2 Methoden des maschinellen Lernens

---

In diesem Kapitel werden die grundlegenden Konzepte vorgestellt, die zum Verständnis der in dieser Arbeit angewandten Methoden notwendig sind. In Abschnitt 2.1 wird zunächst die Notation von Klassifikationsproblemen im Bereich des überwachten Lernens aufgezeigt. Als Nächstes werden in den Abschnitten 2.2, 2.3 und 2.4 die im Rahmen dieser Arbeit verwendeten Algorithmen vorgestellt. Anschließend beschreibt Abschnitt 2.5 verschiedene Bewertungsmetriken für die zuvor erläuterten Algorithmen.

### 2.1 Terminologie und Notation

Maschinelles Lernen (englisch: Machine Learning) ist ein Teilgebiet der Künstlichen Intelligenz, kurz KI, und kann Muster in Daten anhand verschiedener Lernansätze entdecken. Einer von den Lernansätzen ist das Überwachte Lernen (englisch: Supervised Learning), bei dem nicht nur der Datensatz mit Beobachtungen bzw. Instanzen, sondern auch die entsprechenden Ergebnissen bereits bekannt sind. Das Ziel ist, einen mit Daten trainierten Lernalgorithmus auf unbekannte Daten anzuwenden und eine hohe Vorhersagekraft zu erzielen. Überwachtes Lernen wird in zwei Anwendungsbereiche, Regression und Klassifikation, unterteilt, wobei im Folgenden nur die Klassifikation weiter ausgeführt wird [PW19].

Ein Klassifikationsmodell ist eine Funktion  $f : X^d \rightarrow Y$  mit  $d \in \mathbb{N}$  die Dateninstanzen (Tupel)  $x$  aus einem Merkmalsraum  $X^d$  mit Eingabemerkmale auf eine Entscheidung  $y$  in einem Zielraum  $Y$  abbildet.  $X$  heißt auch Eingabeparameter oder unabhängige (Prädiktor-)Variablen,  $Y$  wird auch als Zielvariable oder abhängige (Antwort-)Variable bezeichnet. In einem Klassifikationsproblem mit  $K$  Klassen gilt  $Y \in \{1, \dots, K\}$  mit  $K \in \mathbb{N}$  [GM18].

In Bezug auf Künstliche Intelligenz und maschinelles Lernen lassen sich die Vorhersagemodelle in zwei Gruppen einteilen: White-Box-Modelle und Black-Box-Modelle. Bei White-Box-Modellen handelt es sich um klassische KI-Algorithmen, beispielsweise lineare Regression oder einfache Entscheidungsbäume, die interpretierbar sind und transparente Entscheidungsfindungen präsentieren. Im Gegenteil zu White-Box-Modelle sind die Black-Box-Modelle so konzipiert, dass sie zwar hohe Prognosegenauigkeit produzieren, aber keine direkten Erklärungen für die Entscheidung liefern können [MC20]. Diese Lernalgorithmen werden im Folgenden für die Klassifizierungsaufgaben vorgestellt und in Kapitel 6 angewandt.

### 2.2 Support Vector Machine

Bei einer binären Klassifikation mit Support Vector Machines werden Daten durch eine Hyperebene in zwei Klassen unterteilt. Anhand eines Trainingsdatensatzes  $\{(x_1, y_1), \dots, (x_N, y_N)\} | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}$ , wobei  $y_i$  die Klassenzugehörigkeit für  $x_i$  angibt, soll eine Hyperebene bestimmt werden, die beide Klassen möglichst eindeutig voneinander trennt. Hierfür werden die

Datenpunkte als Vektoren in einem Vektorraum dargestellt. Die Hyperebene wird als  $\{x : f(x) = x^T \beta + \beta_0 = 0\}$  und die Klassifizierungsregel als  $G(x) = \text{sign}(x^T \beta + \beta_0)$  definiert, wobei  $\text{sign}$  die Vorzeichenfunktion und  $\beta$  ein Normalenvektor mit  $\|\beta\| = 1$  ist. Somit soll nun die Hyperebene gefunden werden, die den größten Abstand zwischen den Trainingsdaten für die Klassen 1 und  $-1$  aufweist. Für dieses Ziel wird bei linear trennbaren Trainingsdaten das folgende Optimierungsproblem gelöst:

$$\min_{\beta, \beta_0} \|\beta\|$$

so dass die Nebenbedingung  $y_i(x_i^T \beta + \beta_0) \geq 1$  für alle  $i = 1, \dots, N$  gilt.

Da aber die Trainingsdaten nicht komplett linear trennbar sind, werden Fehlklassifikationen erlaubt und das Optimierungsproblem für nicht-linear separierbare Daten wie folgt umgeformt:

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \xi_i$$

so dass die Nebenbedingung  $\xi_i \geq 0$ ,  $y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i$  für alle  $i = 1, \dots, N$  gilt.

Dabei werden Schlupfvariablen  $\xi_i = (\xi_1, \xi_2, \dots, \xi_N)$  für die Verletzung der Nebenbedingung definiert, wobei die Summe der Schlupfvariablen mit einem Hyperparameter  $C$  multipliziert und zu der zu minimierenden Zielfunktion addiert wird. Das Optimierungsproblem wird in der dualen Form mit Hilfe der Lagrange-Multiplikatoren gelöst. Zusätzlich kann anhand des Kernel-Tricks eine nichtlineare Transformation des Vektorraums stattfinden, die in dieser Arbeit nicht verwendet und daher auch nicht weiter ausgeführt wird [HT09].

## 2.3 Random Forest

Random Forest zählt zu den Ensemble-Methoden und ist ein maschineller Lernalgorithmus, der auf Bagging (Bootstrap Aggregation) basiert. Zunächst findet eine Bootstrap-Stichprobe der Trainingsdaten statt, d.h. es werden  $B$  zufällige Stichproben mit Zurücklegen  $S_b$  für alle  $b = 1, \dots, B$  erzeugt. Anschließend wird auf dem neuen Trainingsdatensatz  $S_b$  ein Entscheidungsbaummodell  $f_b$  konstruiert, sodass am Ende insgesamt  $B$  Entscheidungsbäume entwickelt werden. Ein Entscheidungsbaum als Vorhersagemodell zeigt einen Entscheidungspfad, in dem jeder Knoten ein bestimmtes Merkmal  $j$  anhand einer Frage untersucht; die dazugehörige Antwort wird dann in einem äußeren Knoten bzw. Blatt dargestellt. Beispielsweise können unter Verwendung der Entropie Fragen gestellt werden, die den höchsten Informationsgewinn liefern. Die Vorhersage für eine neue Dateninstanz  $x$  findet am Ende im Fall einer Klassifikation durch eine Mehrheitsentscheidung statt [BA19].

## 2.4 Künstliche neuronale Netze

Ein künstliches neuronales Netz ist ein Vorhersagemode mit strukturellen Analogien zum Gehirn. Im Folgenden wird nur das Multi-Layer Perceptron (deutsch: mehrschichtiges Perzepron), auch bekannt als Feed-Forward-Netz, zur Klassifikation erläutert. Es besteht aus mehreren miteinander verbundenen Neuronen und ist in Schichten angeordnet, wobei jedes Neuron die Ausgabe anderer Neuronen erhält und eine gewichtete Summe berechnet. Ein Neuron wird erst aktiviert, wenn die Berechnung einen bestimmten Schwellenwert überschreitet. Das Neuron  $i$  hat somit die gewichtete Summe  $\sum_{j=1}^N w_{ij} x_j$ , worauf eine Aktivierungsfunktion  $f$  angewendet wird, wobei die Ausgabe  $f(\sum_{j=1}^N w_{ij} x_j)$  an andere Neuronen weitergegeben wird. Es existieren verschiedene

Aktivierungsfunktionen, darunter die ReLu-Funktion (Rectifier Linear Unit)  $f(x) = \max(0, x)$ . In der Eingabeschicht sind die Eingabewerte, die unverändert an eine oder mehrere verborgene Schichten bis zur Ausgabeschicht weitergereicht werden. Bei Klassifikationsaufgaben werden in der Ausgabeschicht die Wahrscheinlichkeiten der Klassen abgebildet. Da beim Trainieren des Netzes Abweichungen zwischen der gewünschten und der tatsächlichen Ausgabe entstehen, werden die Gewichte  $w_{ij}$  mittels Backpropagation-Algorithmus angepasst. Das Ziel ist hierbei den quadratischen Fehlerfunktion  $E = \frac{1}{2} \sum_{i=1}^N (t_i - o_i)^2$ , wobei  $N$  die Anzahl der Trainingsdaten,  $o_i$  der Ausgabewert des Neurons  $i$  und  $t_i$  der dazugehörige Zielausgabewert ist, zu minimieren. Hierfür können unterschiedliche Optimierungsverfahren, wie beispielsweise ADAM (Adaptive Moment Estimation), angewendet werden [EW16].

## 2.5 Klassifikationsmetriken

Die Ergebnisse der binären Klassifizierungsalgorithmen werden mithilfe der Wahrheitsmatrix in Tabelle 2.1 evaluiert. Hierfür wird das Vorhersagemodell erst anhand der Trainingsdaten trainiert, dann wird die Leistung des Modells an Testdaten bewertet. Bei True Positives und True Negatives handelt es sich um Beobachtungen, die richtig klassifiziert wurden, d.h. die positive Klasse wird auch richtig als positiv und die negative Klasse richtig als negativ vorhergesagt. Beobachtungen, die als positiv klassifiziert wurden, aber tatsächlich zur negativen Klasse angehören werden als False Positives bezeichnet. Umgekehrt werden Beobachtungen, die von einem Klassifizierungsalgorithmus als negativ vorhergesagt werden, aber tatsächlich positiv sind, False Negatives genannt.

	Vorhergesagte Klasse 0	Vorhergesagte Klasse 1
Tatsächliche Klasse 0	True Negatives (TN)	False Positives (FP)
Tatsächliche Klasse 1	False Negatives (FN)	True Positives (TP)

Tabelle 2.1: Wahrheitsmatrix

Mithilfe der Wahrheitsmatrix werden die folgenden Maßzahlen definiert:

- Erfolgsrate (Accuracy) =  $\frac{TP+TN}{TP+FN+FP+TN}$
- Sensitivität (Sensitivity) =  $\frac{TP}{TP+FN}$
- Spezifität (Specificity) =  $\frac{TN}{TN+FP}$
- Präzision (Precision) =  $\frac{TP}{TP+FP}$

Die Sensitivität wird auch True Positive Rate oder Recall-Wert genannt. Dabei gilt

$$\text{False Positive Rate} = 1 - \text{Spezifität} = \frac{FP}{TN + FP}$$

und die Informationen aus Präzision und Recall werden zum

$$\text{F1-Wert} = \frac{2 \cdot \text{Recall} \cdot \text{Przision}}{\text{Recall} + \text{Przision}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

kombiniert.

Um die Leistungsfähigkeit der Klassifizierungsalgorithmen zu prüfen und zu visualisieren, kann zusätzlich die ROC-Kurve (Receiver Operating Characteristics) verwendet werden. Dabei handelt es sich um einen Graph mit True Positive Rate, bei der auf der Y-Achse und False Positive Rate

auf der X-Achse aufgetragen ist und mit einer weiteren Maßzahl AUC (Area Under The ROC Curve), also die Fläche unter der ROC-Kurve, versehen ist. Ein AUC nahe 1 wird als zuverlässig angesehen, ein AUC von weniger als 0,5 dagegen als wertloser bzw. zufälliger Klassifikator. Dabei gilt, je höher der AUC-Wert, desto besser ist die Vorhersagekraft des Modells [PW19].

## 3 Grundlagen der Erklärbaren Künstlichen Intelligenz

---

Die Verfahren und Technologien der Künstlichen Intelligenz werden kontinuierlich weiterentwickelt und im Zuge dessen werden auch neue Forschungsgebiete identifiziert. Eines davon ist der Teilbereich der Erklärbaren Künstlichen Intelligenz (XAI). Dieses Kapitel dient als Grundlage für Erkenntnisse der XAI-Forschung und soll das erforderliche Grundwissen in diesem Bereich vermitteln. Zunächst werden in Abschnitt 3.1 der Ursprung und das Ziel der Erklärbarkeit im Zusammenhang mit Künstlicher Intelligenz vorgestellt. Als Nächstes wird in Abschnitt 3.2 der aktuelle Stand der Gesetzeslage wiedergegeben. In Abschnitt 3.3 werden in der XAI etablierte Begriffe und Definitionen erläutert. Ferner werden in den Abschnitten 3.4 und 3.5 die möglichen Arten von Transparenz und wünschenswerte Eigenschaften der Erklärbarkeit aufgeführt. In Abschnitt 3.6 wird die Taxonomie der Modellinterpretierbarkeit detailliert beschrieben. Zum Schluss werden in Abschnitt 3.7 und 3.8 weitverbreitete Methoden und Arten von Erklärungen charakterisiert.

### 3.1 Ursprung der Erklärbarkeit

Im Jahr 1983 führte Swartout [SW83] die ersten Ansätze in Richtung der Erklärbarkeit von Expertensystemen ein. Im Bereich der Wissensbasierten Systeme, einem Teilgebiet der Künstlichen Intelligenz, wurde ausführlich dazu geforscht. Mit weiteren Arbeiten [[SW85], [CT88], [CT89], [SM93]] folgten neue Erkenntnisse und Ziele zu Erklärungskomponenten und -anforderungen [HU92]. Van Lent, Fisher und Mancuso prägten 2004 erstmals den Begriff Erklärbare Künstliche Intelligenz, um die Fähigkeit ihres Systems zu beschreiben, das Verhalten von KI-gesteuerten Einheiten bei der Anwendung von Simulationsspielen zu erklären und Schlüsselereignisse zu kennzeichnen [VF04]. Im Zuge neuer und umfangreicher Angebote und Einsatzbereiche der KI-Technologien wurde das Forschungsgebiet aufgegriffen und konnte mit Hilfe der DARPA-Initiative [GD16] an Ansehen und Bekanntheit gewinnen.

Auf dem Gebiet der Erklärbaren Künstlichen Intelligenz wird fortlaufend geforscht und es gibt eine Vielzahl an Arbeiten, die sich nur auf das Teilgebiet der Erklärbarkeit von Deep Learning bzw. von Machine-Learning-Modellen spezialisiert haben [MK19]. Grundsätzlich liefern die Deep Learning Modelle zwar treffsichere Prognosen bezüglich der Datengrundlage, sie haben aber den Nachteil, dass die eigentlichen Entscheidungsprozesse des Algorithmus bei Black-Box-Verfahren aufgrund der Berechnung von komplexen Formeln und der Bestimmung mehrerer Parameter nicht transparent vermittelt werden können. Im Zusammenhang mit dieser Problematik erschließt sich die Wichtigkeit eines transparenten und verständlichen komplexen maschinellen Lernverfahrens je nach Anwendungsgebiet [GM18].

Dabei hängt der Umfang eines interpretierbaren bzw. erklärbaren Modells sehr oft von dessen Komplexität ab. Es gibt verschiedene Möglichkeiten, um die Komplexität eines Modells zu definieren und zu quantifizieren. Im Allgemeinen gilt, je komplexer das Modell, desto schwieriger ist es zu interpretieren und zu erklären. Verglichen mit einfachen und leicht verständlichen Modellen

sind komplexe ML-Modelle mit höherem Zeitaufwand verbunden und können sehr rechenaufwändig sein. Die Komplexität eines Modells kann durch die Berücksichtigung von Eigenschaften wie Linearität oder Monotonie bestimmt werden, aber auch durch die Anzahl der Gewichte oder Regeln in einem Modell. Zusätzlich können die Anzahl der Merkmale und die Interaktionen zwischen ihnen betrachtet werden. Niedrigere Komplexität kann erreicht werden, wenn ein Modell Vorhersagen nur anhand weniger Merkmale trifft, zwischen denen nur eine geringe Interaktion besteht [HG19].

Diesbezüglich wurden gängige Machine-Learning-Algorithmen und die dazugehörige Erklärbarkeit dieser Modelle in der erwähnten DARPA-Studie verglichen (siehe Abbildung 3.1).

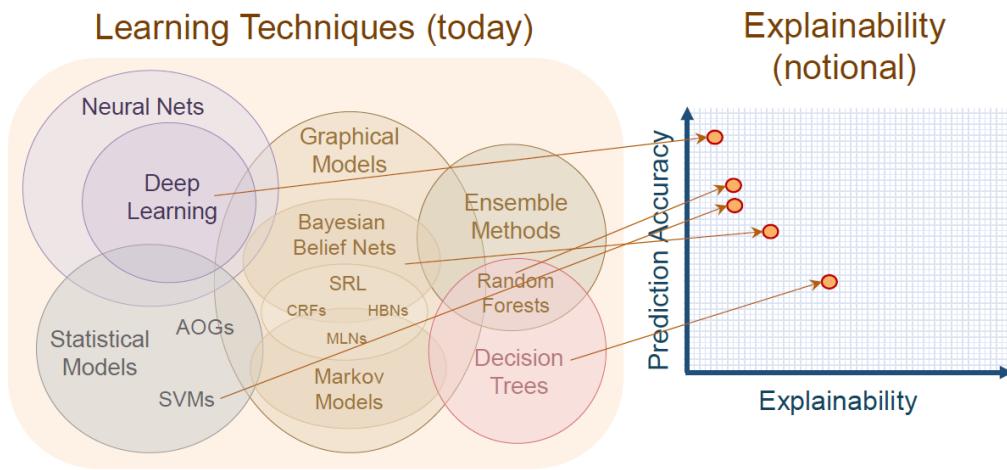


Abbildung 3.1: Der Kompromiss zwischen Genauigkeit und Erklärbarkeit [GD16]

Es ist erkennbar, dass Modelle wie Entscheidungsbäume (Decision Trees) zwar recht gut interpretierbar sind, aber im Vergleich zu den schwer nachvollziehbaren (tiefen) künstlichen neuronalen Netzen auch nur eine niedrigere, in Abschnitt 2.5 definierte, Erfolgsrate erreichen. Dabei hängt die potenzielle Genauigkeit eines Modells von seiner Komplexität ab. Ziel ist es, komplexe Modelle mit höherer Performance interpretierbar zu machen [GD16].

Zu diesem Zweck wurden zahlreiche Erklärungstechniken entwickelt, die das Zustandekommen der Entscheidungen und Prognosen von Black-Box-Modellen nachvollziehbar darstellen. Im Kontext der Erklärbarkeit der Black-Box-Modelle werden zunächst in Abschnitt 3.6 mit Hilfe unterschiedlicher Erklärungsansätze verschiedene Möglichkeiten zur Klassifizierung von Modellinterpretationen vorgestellt. In Abschnitt 3.7 und 3.8 werden je nach Erklärungsansatz passende Methoden und Erklärungsarten näher erläutert. Im Anschluss an diese Erklärungstechniken können dann die konkreten Erklärungsmodelle in Kapiteln 4 und 5 spezifiziert werden.

## 3.2 Ethische, rechtliche und soziale Rahmenbedingungen

Das Delegieren von Entscheidungen an komplexe, intransparente und nicht nachvollziehbare Algorithmen kann einen negativen Einfluss auf den gesellschaftlichen Zuspruch für das Einsatz von KI-Technologien ausüben. Hinsichtlich dieser Intransparenz können die unerkannten Auswirkungen von KI-Anwendung gegebenenfalls zur Verletzung von ethischen, rechtlichen oder sozialen Anforderungen führen [FG18].

Mit aktuellen Entwicklungen im Bereich der Erklärbaren Künstlichen Intelligenz sollen genau diese Schwachstellen ermittelt werden und für die Förderung der Forschung in Richtung transparenter und verantwortungsvoller KI geeignete Rahmenbedingungen geschaffen werden. Kenn-

zeichnend für die Folgen intransparenter automatisierter Entscheidungen sind nicht nur, wie in Abschnitt 1.2 erläutert, verborgene Clever-Hans-Effekte, sondern auch Fehlschlüsse durch *Scheinkausalitäten* [MM19].

Bei Scheinkausalität handelt es sich um einen statistischen Zusammenhang (Korrelation) zwischen zwei Variablen, die fälschlicherweise als Kausalzusammenhang (Kausalität) interpretiert wird. Dabei kann eine Scheinkausalität entstehen, wenn die Korrelation in den Daten zufällig ist oder bei beiden Größen eine indirekte Beziehung zu einer nicht einbezogenen Ursache vorliegt. Die Korrelation, das heißt hier die Stärke des Zusammenhangs zwischen den beiden Variablen, kann durch unterschiedliche Zusammenhangsmaße (Korrelationskoeffizienten) gemessen werden. Im Gegensatz dazu können kausale Schlüsse nicht vollständig mit statistischen Methoden nachgewiesen werden. Stattdessen kann dabei für die Entdeckung von Kausalzusammenhängen eine gemeinsame Untersuchung mehrerer Variablen und deren Zusammenhänge im Rahmen der multivariaten Statistik als Hilfsmittel genutzt werden. Die Erkenntnisse daraus können z.B. durch wiederholbare randomisierte Experimente überprüft werden [[MV11], [HS18]].

Als konkretes Beispiel kann die kriminelle Risikobewertung in den Vereinigten Staaten betrachtet werden. Viele Gerichte in den USA verwenden den Algorithmus COMPAS („Correctional Offender Management Profiling for Alternative Sanctions“) für die Berechnung der Rückfallwahrscheinlichkeit eines Angeklagten. Dabei soll das Vorhersagemodell einen Rückfälligkeitswert anhand von 137 Fragen ermitteln, die entweder vom Angeklagten selbst oder durch Auskunftserteilung aus dem Strafregister beantwortet werden. Die Fragen sind in verschiedene Kategorien unterteilt und es werden Merkmale wie Wohnort, Einkommen oder Familienverhältnisse betrachtet. Dieses Risikobewertungsinstrument wird dann vom Richter als Entscheidungshilfe bei der Frage einer möglichen Aussetzung der Freiheitsstrafe zur Bewährung eingesetzt [[AL16], [NI11]].

Der Algorithmus wurde mit der Kriminalitätsstatistik trainiert und dementsprechend beruhen die historischen Daten nicht auf kausalen Zusammenhängen, sondern auf statistischen Korrelationen. Infolgedessen wurde vom Vorhersagemodell auch keine tatsächliche Einflussfaktoren, sondern nur die Korrelationen von bestimmten Merkmalen für eine höhere kriminelle Rückfälligkeit ermittelt. Daher wurde u.a. bei Menschen mit schwarzer Hautfarbe eine höhere Punktzahl für die Rückfälligkeit vorhergesagt als bei weißen Menschen. Im Nachhinein wurde festgestellt, dass Schwarze, die doppelt so häufig als risikoreicher eingestuft wurden als Weiße, oft doch nicht erneut straffällig wurden. Dagegen haben viele weiße Menschen, die fast doppelt so häufig risikoarm im Vergleich zu schwarzen Menschen eingeschätzt wurden, tatsächlich wieder andere Straftaten begangen [AL16].

Dazu ist noch anzumerken, dass Angaben zur ethnischen Zugehörigkeit zwar nicht explizit abgefragt wurden, aber anhand bestimmter Fragen auf diese Variablen zurückgeschlossen werden konnte. Beispielsweise durch Faktoren wie Arbeitslosigkeit oder Kriminalität im eigenen Umfeld, bei denen der Anteil dunkelhäutiger Personen höher liegt [AL16]. Die Ergebnisse des Algorithmus sind von der Qualität der zugrundeliegenden Daten abhängig und das Vorhersagemodell kann in den Daten vorhandene eventuelle Diskriminierungen reproduzieren [KS19]. In diesem Beispiel können auch Fehler, die bei vergangenen Gerichtsentscheidungen bezüglich Angeklagten aus bestimmten Bevölkerungsgruppen gemacht wurden, in die Prognose einbezogen werden und so die Entscheidung einer vorzeitigen Haftentlassung beeinflussen.

Das hohe Diskriminierungspotential ist eine Folge von *algorithmischer Voreingenommenheit*, auf Englisch *Algorithmic Bias*. Dem Machine-Learning-Modell wird es vom Entwickler oder Anwender beim Trainieren bewusst oder unbewusst angeeignet oder, wie oben erörtert, aus den Daten durch bereits vorhandene Vorurteile übernommen. Daher können auch die Ergebnisse von KI-Anwendungen unethisch oder diskriminierend sein und ernsthafte Konsequenzen für bestimmten

Personengruppen oder Minderheiten haben [MM19].

Algorithmischer Voreingenommenheit kann unter Umständen durch Ausgewogenheit der Trainingsdaten und Eliminieren von diskriminierende Tendenzen vorgebeugt werden [KS19]. Jedoch gibt es auch KI-Systeme, bei denen die Entwickler die Daten nur bedingt im Voraus oder sogar erst in der Anwendung inspizieren können. Ein gutes Beispiel ist der Chatbot Tay von Microsoft aus dem Jahr 2016. Für den Chatbot wurden Profile bei mehreren Sozialen Netzwerken erstellt und er sollte lernen, wie eine 18- bis 24-jährige Frau aus den USA zu kommunizieren. Dafür wurden andere Nutzer der Plattformen aufgefordert, sich mit Tay auszutauschen und so beim Lernprozess mitzuhelpen. Dabei wurden dem Algorithmus vorweg mithilfe eines Filters obsszöne Äußerungen unterbunden. Allerdings brachten die Twitter-Nutzer dem an sich harmlosen Chatbot fremdenfeindliche und beleidigende Konversationen bei. Infolgedessen verfasste Tay innerhalb von 16 Stunden rassistische, antisemitische und sexistische Tweets und musste von den Entwicklern wieder abgeschaltet werden [[KS19], [BP16]].

An dieser Stelle treten nicht nur ethische, sondern auch rechtliche Herausforderungen an das lernende System auf. So waren die Microsoft-Entwickler verantwortlich für das diskriminierende Verhalten des Chatbots, obwohl die Diskriminierungen hier nicht auf den Entwickler selbst, sondern auf die Handlungen der Twitter-Nutzer zurückzuführen sind [KS19].

Vergleichsweise gibt es in der Automobilbranche schon seit einigen Jahren eine Debatte über die rechtliche Verantwortung und Haftung für potenzielle Schadensfälle. Denn es werden zeitgemäß in der Automobilbereich Künstliche Intelligente Systeme angewendet, insbesondere werden komplexe Deep-Learning-Algorithmen bei der Entwicklung von Fahrerassistenzsystemen eingesetzt. Mit der Verarbeitung von Sensordaten, automatisierter Objekterkennung und unter Berücksichtigung der jeweiligen Straßenverkehrsordnung werden ambitionierte Ideen und Konzepte in Richtung autonomes Fahren gefördert [KC16].

Zwar werden hier auf dem Weg zur Fahrzeugautomatisierung kontinuierlich technische Fortschritte erzielt, jedoch müssen zusätzlich *juristische und ethische Anforderungen* angesprochen werden [SW19].

Mit wachsendem KI-Einsatz in der Automobilindustrie drängt sich unweigerlich die Frage auf, wer bei einem Verkehrsunfall für resultierende Schäden haften muss. Es werden fünf Stufen im Bereich des automatisierten Fahrens unterschieden, wobei 2017 je nach technischem Automatisierungsgrad unterschiedliche haftungsrechtliche Regelungen in Kraft getreten sind. Prinzipiell wurde vom Bundestag beschlossen, dass bei herkömmlichen Fahrerassistenzsystemen oder auch vollständig autonomem Fahren der Fahrer die Fahrt überwachen und im Ernstfall auch die Steuerung übernehmen soll. Tatsächlich ist zurzeit die höchste Stufe die des voll autonomen Fahrens, das heißt ein Fahrzeug fährt ohne Fahrer bzw. dieser wird zum „reinen“ Passagier, in Deutschland noch unzulässig [DB18].

Um das Ziel der Vollautomatisierung zu erreichen, müssen neben der rechtlichen Verantwortung und Haftung für gefährliche Kollisionen auch die Entscheidungen des autonomen Systems näher betrachtet werden. Hierzu hat die Ethikkommission zum autonomen Fahren für das Bundesministerium für Verkehr und digitale Infrastruktur im Jahr 2017 notwendige ethischen Leitlinien konzipiert. Unter anderem wird der Schutz des Lebens als höchste Priorität angesehen, sodass ein Sachschaden stets einem Personenschaden vorzuziehen sein soll. Zudem ist bei einer Unfallsituation die Selektion von Menschen nach persönlichen Merkmalen unzulässig: Faktoren wie Alter, Geschlecht usw. dürfen keine Entscheidungskriterien für den Entscheidungsprozess sein [EK17].

Um diese Richtlinien erfüllen zu können, sollen daher auch bei der Nutzung von KI für autonom agierende Systeme die Nachvollziehbarkeit und Transparenz der Entscheidungsfindung sichergestellt werden. Dies ermöglicht nicht nur die Aufdeckung von eventuell diskriminierenden

Entscheidungsmerkmalen, sondern hilft auch die gesellschaftliche Akzeptanz von vollautomatisierten Fahrzeugen zu etablieren [SW19].

Dabei ergeben sich nicht nur im Hinblick auf Fahrerassistenzsysteme oder autonome Transportsysteme in der Mobilitätsbranche ethische Bedenken, sondern auch bei der Nutzung von komplexeren KI-Algorithmen mit entsprechender Intransparenz und Verantwortlichkeitslücken im Allgemeinen. Auch hierzu wurde im Mai 2018 die *Datenschutz-Grundverordnung (DSGVO) der Europäischen Union (EU)* vorgestellt. Die Verordnung reguliert generell den Daten- und Verbraucherschutz und ist für alle EU-Mitgliedstaaten verbindlich, wobei nach Artikel 84 Sanktionen bei Verstößen verordnet werden können. Neben Vorgaben wie Informationspflicht oder Auskunftsrecht wurden auch Ansätze in Richtung Transparenz und Nachvollziehbarkeit beim Einsatz von algorithmenbasierten Verfahren eingeführt [[EU16], [MM19]].

Gemäß Artikel 12 bis 15 müssen betroffene Personen u.a. „über das etwaige Bestehen einer automatisierten Entscheidungsfindung einschließlich Profiling mit aussagekräftigen Informationen über die dabei involvierte Logik sowie die Tragweite und die angestrebten Auswirkungen solcher Verfahren“ informiert werden. Laut Artikel 4 Nr. 4 wird Profiling als jede Art der automatisierten Verarbeitung personenbezogener Daten bezeichnet, um auf deren Grundlage bestimmte persönliche Aspekte zu bewerten und Nutzerinteressen vorherzusagen [EU16].

Zudem haben die betroffene Personen nach Artikel 22 das Recht, nicht ausschließlich Entscheidung unterworfen zu werden, die auf automatisierter Verarbeitung oder Profiling beruhen, wenn diese Entscheidung den Betroffenen „gegenüber rechtliche Wirkung entfaltet oder sie in ähnlicher Weise erheblich beeinträchtigt“. Nach Erwägungsgrund 71 haben somit betroffene Personen bei automatischer Ablehnung eines Online-Kreditantrags oder eines Online-Einstellungsverfahrens ohne jegliches menschliches Eingreifen Anspruch auf ein direktes Eingreifen einer Person und „auf Erläuterung der nach einer entsprechenden Bewertung getroffenen Entscheidung sowie [das Recht] auf Anfechtung der Entscheidung“ [EU16].

Grundsätzlich ist die Transparenzanforderung für die betroffene Personen sehr wichtig und es wird Artikel 13 und 14 zufolge eine „faire und transparente Verarbeitung“ von personenbezogenen Daten verlangt, nichtsdestotrotz sollen weiterhin Betriebs- und Geschäftsgeheimnisse bewahrt werden. Zudem würde ein durchschnittlicher Nutzer bei einer vollständigen Offenlegung der Algorithmen-Quellcodes nicht unbedingt die Entscheidungsfindung nachvollziehen können. Daher sollen auch die betroffenen Personen nachvollziehen können, auf welcher Basis die Entscheidungen getroffen wurden. Dazu wurde aber keine eindeutige Definition bezüglich der Erläuterungen für getroffene Entscheidungen oder Prognosen vorgestellt [MM19].

Ähnlich wie bei der Fahrzeugautomatisierung wurde hierzu von der EU-Kommission eine hochrangige Expertengruppe für Künstliche Intelligenz einberufen. Der am April 2019 vorgestellte Bericht über die *Ethik-Leitlinien für eine vertrauenswürdige KI* nennt die folgenden vier ethischen Grundsätze als Fundamente einer vertrauenswürdigen KI: Achtung der menschlichen Autonomie, Schadensverhütung, Fairness und Erklärbarkeit. Zusätzlich wurden diverse Anforderungen für die Umsetzung und Verwirklichung einer vertrauenswürdigen KI ausgearbeitet und abschließend eine Bewertungsliste vorgestellt [EK19a].

Die Bewertungsliste wurde entwickelt, um anhand spezifischer Fragen – die je nach Anwendungsfall angepasst werden können – die Entwicklung einer vertrauenswürdigen KI zu gewährleisten. Mit dem Ziel, die vorgestellten ethischen Leitlinien zu evaluieren und weiterzuentwickeln, haben sich hunderte Organisationen bereit erklärt, den Fragenkatalog in ihre bestehenden Verfahren einzubeziehen und Rückmeldungen über die Umsetzbarkeit und Vollständigkeit zu geben. Dabei sind die ausgearbeiteten Prinzipien vorerst rechtlich nicht bindend und dienen daher aktuell lediglich als Empfehlungen für die Entwicklung einer vertrauenswürdigen KI [EK19b].

Mit der Veröffentlichung von EU-DSGVO wurden nicht nur die Ethik-Leitlinien für eine vertrauenswürdige KI gefördert, sondern auch Forschungen von angesehenen Instituten und Verbänden auf dem Gebiet der Künstlichen Intelligenz [[BR18], [DE19], [FG18]]. Auch unterschiedliche Stiftungen und Organisationen förderten mehrere Publikationen, wie Algorithmenethik oder AlgorithmWatch, und es gab eine Vielzahl an Stellungnahmen zur Anwendung von Deep-Learning-Technologien und den Auswirkungen von Entscheidungen anhand algorithmenbasierter Prozesse [[JS17], [DS18]]. Dabei wurden zum einen Themen wie Erklärbarkeit, Transparenz und Rechenschaftspflicht ausführlich diskutiert, zum anderen Ansätze und Empfehlungen für politische Entscheidungsträger ausgearbeitet. Durch die Aufklärung und Sensibilisierung der Öffentlichkeit über die Folgen und Risiken der algorithmenbasierten Verfahren können konkrete Prinzipien und Standards für transparente und nachvollziehbare KI-Systeme weiterentwickelt und im Zuge eines verantwortungsvollen Einsatzes von KI etabliert werden.

Zusammenfassend wurden die sozialen, ethischen und rechtlichen Perspektiven vorgestellt und dabei zunächst die Risiken von Scheinkausalitäten und algorithmischer Voreingenommenheit näher erläutert. Ziel ist es nun, anhand geeigneter Erklärungsmodelle die Gründe für die Entscheidungen von KI-Anwendungen zu finden, sie für die Betroffenen oder Nutzer der KI-Systems auf verständliche Weise zu erläutern und die entscheidenden Faktoren für das Ergebnis der KI offenzulegen. Dadurch können Ungereimtheiten in den Daten oder dem genutzten Algorithmus aufgedeckt und nachträgliche Korrekturen vorgenommen werden. Angestrebt werden hier Fairness und Diskriminierungsfreiheit und schließlich eine Steigerung der Akzeptanz von KI-Technologien in der Gesellschaft. Um dies zu erreichen, werden nicht nur transparente Algorithmen und erklärbare Entscheidungen gefordert, sondern auch Debatten zum Thema ethische Richtlinien initiiert. Um den ethischen Anforderungen zu genügen, muss mit wachsendem Einsatz von KI, insbesondere von algorithmenbasierten Prognose- und Entscheidungssystemen, auch der Grad an Verantwortlichkeit und Haftung weiterentwickelt werden. Gemäß Bundesregierung [BR18] bietet der aktuelle Ordnungsrahmen schon „eine stabile Grundlage mit hohen Standards“, kann aber sofern erforderlich weiter angepasst werden. Im Folgenden werden ausschließlich die *technische Entwicklungen* im Mittelpunkt stehen. Um dies zu fundieren, werden zunächst die grundlegenden Informationen zu XAI-Systemen vorgestellt.

### 3.3 Terminologie

In Bezug auf die Begrifflichkeiten und selbst den Namen der Erklärbaren Künstlichen Intelligenz gibt es bisher keine einheitlich anerkannten Definitionen. Das übergreifende Forschungsfeld wird neben Explainable AI auch als Trustworthy AI [AK20] oder Responsible AI [DV19] bezeichnet. Dabei können erst durch erklärbare Darstellungen von KI-Entscheidungen und -Prognosen weitere Schritte in Richtung vertrauenswürdige und verantwortliche Künstliche Intelligenz eingeleitet werden. Zudem existieren unterschiedliche Ansichten und Darstellungen zu Bezeichnungen. Mit dem Ziel der Konsolidierung wurde im Auftrag von DARPA und deren XAI-Programm ein ausführlicher Literaturüberblick [MK19] veröffentlicht, der das gegenwärtige Elementarwissen und Ideen zur Erklärbaren Künstlichen Intelligenz darstellt.

Da in der Literatur gleiche Begriffe zum Teil unterschiedliche Bedeutungen haben [[DK17], [GB19], [LZ16]], folgt nun eine Auswahl an Fachwörtern, wobei zwischen Transparenz, Erklärbarkeit und Interpretierbarkeit unterschieden wird:

*Transparenz:*

Transparenz verweist auf das angewendete Verfahren und die dazugehörige weitgehende Nach-

vollziehbarkeit des Verhaltens. Die Anforderungen beziehen sich dabei auf die gesamte Modellstruktur, einzelne Modellelemente und den Lernalgorithmus. Konkret soll der Vorgang Trainieren des Modells mit Trainingsdaten, Anwendung auf Testdaten und anschließende Erzeugung von Prognosen dargestellt und belegt werden [RB20].

#### *Erklärbarkeit:*

Erklärbarkeit bedeutet, dass die Gründe für die erzeugten Prognosen und Entscheidungen auf eine bestimmte Art, z.B. durch Veranschaulichung der wichtigsten Einflussfaktoren, begründet werden [IE20].

#### *Interpretierbarkeit:*

Im Allgemeinen handelt es sich bei der Interpretierbarkeit von Machine-Learning-Modellen um die Fähigkeit, die Ergebnisse der Modelle in eine für Menschen verständliche und aufschlussreiche Form zu bringen. In diesem Zusammenhang sollen die Erklärungen die menschliche Auffassung berücksichtigen, einen Sinn ergeben und nachvollziehbar gestaltet sein [GM18].

### **3.4 Arten von Transparenz**

In Anbetracht der Transparenzanforderungen an KI-Systeme treten Fragen bezüglich der Motivation und des Nutzens der geforderten Informationen auf. Adrian Weller [WA19] unterscheidet acht Arten und Ziele von Transparenz, wobei die Erklärung entweder direkt aus dem originalen System abzulesen ist oder erst durch einen Erklärungsalgorithmus des ursprünglichen Systems erzeugt wird.

Hierbei werden verschiedene Gruppen von Menschen betrachtet: Der Entwickler (developer) gestaltet die Erklärung für das originale System, der Betreiber (deployer) ist der Inhaber und in der Lage, die Erklärung an die Öffentlichkeit oder für bestimmte Benutzergruppen freizugeben. Letztlich ist der Benutzer (user) ein Anwender des Systems. Gemäß der Gruppenzugehörigkeit werden unterschiedliche Typen von Erklärungen gefordert:

- Typ 1 Der Entwickler hat das Ziel, die Funktionsweise des Systems zu verstehen und das System zu verbessern, indem eventuell vorhandene Fehler beseitigt werden.
- Typ 2 Der Benutzer kann durch die gelieferte Erklärung das System, dessen Vorgehensweise und Handlungen bei möglichen Ausnahmefällen besser verstehen, was wiederum zur Verstärkung des Vertrauens in die Technologie führt.
- Typ 3 Im Hinblick auf die Gesellschaft können erst dann, wenn die Stärken und Schwächen bzw. die Grenzen des Systems erklärt werden, die Bedenken bezüglich des unbekannten Bereichs der KI eliminiert werden.
- Typ 4 Anhand der Erklärung kann der Benutzer nachvollziehen, warum eine bestimmte Entscheidung getroffen wurde und erwirbt zusätzlich, beispielsweise bei einer Überprüfung der Kreditwürdigkeit, das Recht auf eine Einspruchsmöglichkeit.
- Typ 5 Die Experten können ausführlich die einzelnen Entscheidungen überprüfen und zudem die Verantwortlichkeit und somit auch rechtliche Haftung zuweisen. Dies ist maßgebend bei folgenschweren Vorfällen, z.B. bei Unfällen durch autonomes Fahren.
- Typ 6 Mittels Erklärungen des KI-Systems kann die Überwachung und Prüfung der entsprechenden Sicherheitsnormen gewährleistet werden.

Typ 7 Wenn der Benutzer sich mit der Vorhersage bzw. Entscheidungsfindung vertraut macht, bewirkt dies voraussichtlich eine wiederkehrende Nutzung des Systems.

Typ 8 Der Benutzer wird aufgrund der vorhandenen Erklärung gezielt zu einer Handlung dirigiert, beispielsweise zum Kauf eines Artikels durch gezielte Produktempfehlung und die entsprechende Erklärung für diese Empfehlung.

Diese Kategorien ermöglichen einen groben Überblick über die Interessenten und deren Nutzen aus transparenten algorithmischen Systemen. Anhand der jeweiligen Motivationshintergründe wird der Erklärungsberechtigte festgestellt und die Informationen können im legitimen Rahmen für die Nutzung bereitgestellt werden.

### 3.5 Desiderate der Erklärbarkeit

Anhand konkreter Attribute von Erklärungsmethoden kann die Angemessenheit und der Einsatz dieser Methoden beurteilt werden. Auch im Hinblick auf wünschenswerte Eigenschaften für die Erklärungen von KI-Systemen gibt es unterschiedliche Varianten zur Auswahl [[LZ16], [DK17], [AD19]]. Die Autoren von [HR19] verweisen auf die Erkenntnisse von Swartout und Moore [SM93] und deren Angaben zu erstrebenswerten Merkmalen für nützliche Erklärungen. Die folgenden allgemein definierten Ausführungen stammen aus dem Jahr 1993 und wurden für erklärbare Expertensysteme entwickelt:

- *Fidelity – Treue*: Das zugrunde liegende System sollte anhand vertrauensvoller Erklärung sinnvoll dargestellt werden.
- *Understandability – Verständlichkeit*: Umfasst mehrere Faktoren der Benutzerfreundlichkeit, einschließlich Terminologie, Benutzerkompetenzen, Abstraktionsniveau und Interaktivität.
- *Sufficiency – Adäquanz*: Es sollte die Funktion und die Terminologie erklären und detailliert genug sein, um eine Entscheidung zu begründen.
- *Low Construction Overhead – Geringer Mehraufwand*: Die Erklärung sollte den Aufwand für die Gestaltung der Künstlichen Intelligenz selbst im Regelfall nicht überwiegen. Falls doch, kann auf die Bereitstellung einer Erklärung verzichtet werden.
- *Efficiency – Effizienz*: Das Erklärungssystem sollte im Wesentlichen nicht die Leistungsfähigkeit der Ausführungszeit von Künstlicher Intelligenz verlangsamen.

Angesichts von Machine-Learning-Modellen wurden weitere Eigenschaften für Erklärungsmethoden definiert, wobei verschiedene Vorstellungen und Präferenzen im Fokus stehen [[GM18], [DK17]]. Dazu gehören im Zusammenhang mit der zeitgemäßen ethischen Auffassung die Faktoren *Gerechtigkeit - Fairness* und *Datenschutz – Privacy*. Zum einen wurden Unvoreingenommenheit und Schutz vor Diskriminierung gefordert, zum anderen sollen vertrauliche Daten sorgfältig behandelt werden [[GM18], [RR14], [AS15]].

Ein weiterer erwähnenswerter Punkt ist, dass die Ergebnisse der verwendeten Algorithmen auf *Kausalität – Causality* basieren sollen. Somit wird die Verwendung von kausalen Beziehungen zwischen den Variablen im Modell gewährleistet und die unmittelbaren Erkenntnisse können korrekt interpretiert werden[ [CP19], [HL19]].

Abschließend wird auf die Besonderheit der *Konsistenz – Consistency* von Erklärungsmethoden hingewiesen. Hierbei sollten für ein festes Modell die Erklärungen bei ähnlichen Datenpunkten

ähnlich sein, außerdem für einen festen Datenpunkt zwischen verschiedenen Vorhersagemodellen oder dasselbe Modell mit verschiedenen Trainingsläufen (mit denselben Daten trainiert) vergleichbar sein [[SF20a], [MC20], [CP19]].

## 3.6 Taxonomie der Modellinterpretierbarkeit

Um die Taxonomie der Erklärbarkeit von Künstlicher Intelligenz einheitlich vorzustellen, werden in diesem Abschnitt die Erklärungsansätze als Methoden der Modellinterpretierbarkeit aufgefasst. Es wurde schon in Abschnitt 3.3 darauf hingewiesen, dass in den meisten Arbeiten, die sich mit der Erklärbarkeit von ML-Algorithmen befassen, der Begriff Interpretierbarkeit oftmals synonym mit anderen Begriffen wie Transparenz und Erklärbarkeit verwendet wird. Die Analyse eines maschinellen Lernmodells hinsichtlich seiner Interpretierbarkeit anhand der nachfolgend betrachteten Erklärungsansätze ist ein wichtiger Baustein, um den Stand der Forschung umfassend abzubilden. Die Ansätze bieten nicht nur einen Überblick über die bestehende Modellinterpretierbarkeit, sondern dienen auch als Hilfsmittel für einen Vergleich oder eine Gegenüberstellung unterschiedlicher Erklärungsmethoden. Gemäß [MC20] und [HG19] können die Erklärungsansätze für die Interpretierbarkeit von ML nach folgenden Kriterien klassifiziert werden.

### Intrinsische und Post-hoc-Interpretierbarkeit

Die Erklärungsansätze werden prinzipiell in intrinsische und Post-hoc-Ansätze unterteilt und danach unterschieden, ob die Interpretierbarkeit durch Einschränkung der Komplexität des maschinellen Lernmodells oder durch die Anwendung von Methoden zur Analyse nach dem Training erreicht wird. Dabei bezieht sich *intrinsische* Interpretierbarkeit auf Modelle, die aufgrund ihrer einfachen Struktur und geringen algorithmischen Komplexität als interpretierbar gelten. Beispielsweise sind die gelernten Aufteilungen von kurzen Entscheidungsbäumen oder die Gewichte eines einfachen linearen Modells leichter zu interpretieren, womit sie das Verständnis für die Entscheidungen und Resultate des Modells fördern. Im Gegensatz dazu verweist die *Post-hoc* Interpretierbarkeit auf Erklärungsmethoden, die erst nach dem Modelltraining angewendet werden. Zudem werden Post-hoc-Erklärungen verwendet, um die Vorhersagen von komplexen Modellen, beispielsweise künstlicher neuronaler Netze oder Random Forest, zu erklären. Bei der nachträglich hergestellten Interpretierbarkeit ist es nicht möglich die genaue interne Funktionsweise des Modells aufzuzeigen. Stattdessen werden die maßgeblichen Faktoren des Entscheidungsprozesses identifiziert und nachvollziehbar dargestellt. Zusätzlich können Post-hoc eingesetzte Erklärungsmethoden auch auf intrinsisch interpretierbare Modelle angewendet werden.

### Modellspezifische und modellagnostische Interpretierbarkeit

Ein weiteres wichtiges Kriterium ist die Unterscheidung zwischen modellspezifischen und modellagnostischen Erklärungsansätzen: Die Methoden für die Interpretierbarkeit umfassen im Allgemeinen nur eine bestimmte Modellklasse oder agieren modellunabhängig, u.a. also auch für Black-Box-Modelle. Die *modellspezifischen* Erklärungsansätze sind Interpretationsmethoden, die nur bei bestimmten Modelltypen verwendet werden können. Die Ansätze werden speziell für die jeweiligen Modelle entwickelt. Somit basieren die Erklärungen auf den jeweils internen Abläufen des Modells, wobei die modellinternen Strukturen, Gewichte oder Parameter untersucht werden. Funktioniert z.B. ein Ansatz ausschließlich für die Interpretation von künstlichen neuronalen Netzen, so wird diese Interpretationsmethode als modellspezifisch angesehen. Des Weiteren gelten Interpretationen von intrinsisch interpretierbaren Modellen, beispielsweise von Gewichten in

einem linearen Regressionsmodell, immer als modellspezifisch. Demgegenüber sind *modellagnostische* Erklärungsansätze nicht an bestimmte Arten von Algorithmen gebunden, sondern können auf jedes Modell angewendet werden. Die Interpretationsmethoden werden hier erst nach dem Training des Modells, also Post-hoc, verwendet. Üblicherweise arbeiten diese Ansätze durch Analysieren von Eingabe- und Ausgabewerten und können das Verhalten des Modells entschlüsseln, ohne seine Interna, z.B. Informationen über Gewichte oder Strukturen, zu kennen. Können Methoden verallgemeinert werden, z.B. durch Erstellung einer Erklärung für unterschiedliche Modelltypen wie lineare Regression oder künstliche neuronale Netze, so gelten sie als modelagnostisch.

### Globale und lokale Interpretierbarkeit

Bei der Erzeugung von Erklärungen wird schließlich noch der Geltungsbereich als Charakterisierungsmerkmal betrachtet. Dabei werden die Interpretationsmethoden danach klassifiziert, ob das Modell auf globaler Ebene zu verstehen ist und so versucht wird, das gesamte Modellverhalten zu erklären, oder ob nur in der lokalen Region agiert wird und daher auch das Verständnis auf einer einzelnen Vorhersage, auch Instanz genannt, beschränkt ist. Eine *globale* Erklärung bietet nicht nur einen weitreichenden Einblick in das Gesamtbild, sondern auch in die Funktionsweise des Modells. Mit diesem Erklärungsansatz wird versucht, die Modellentscheidungen auf Grundlage von gelernten Beziehungen zwischen den Eingabeparametern und den Zielvariablen über den gesamten Datensatz hinweg zu verstehen. Die *lokale* Interpretierbarkeit bezieht sich hingegen auf Erklärungen einzelner Vorhersagen oder Gruppen von Vorhersagen. Folglich wird beim Generieren der Gründe für die Entstehung von Prognosen die Beziehung zwischen den Eingabeparametern und Zielvariablen speziell nur auf diesen Datenpunkt oder eine Gruppe ähnlicher Datenpunkte bezogen betrachtet. Somit identifizieren die lokalen Erklärungen spezifische Eingabeparameter, die zu einer bestimmten Entscheidung für eine einzelne Instanz oder eine Gruppe von Instanzen beigetragen haben. Hierzu kann die folgende Abbildung 3.2 betrachtet werden, wobei nur beispielhaft die globale und lokale Interpretierbarkeit eines Modells dargestellt wird. Die Abbildung stammt aus [CP18] und wurde auf Deutsch übersetzt.

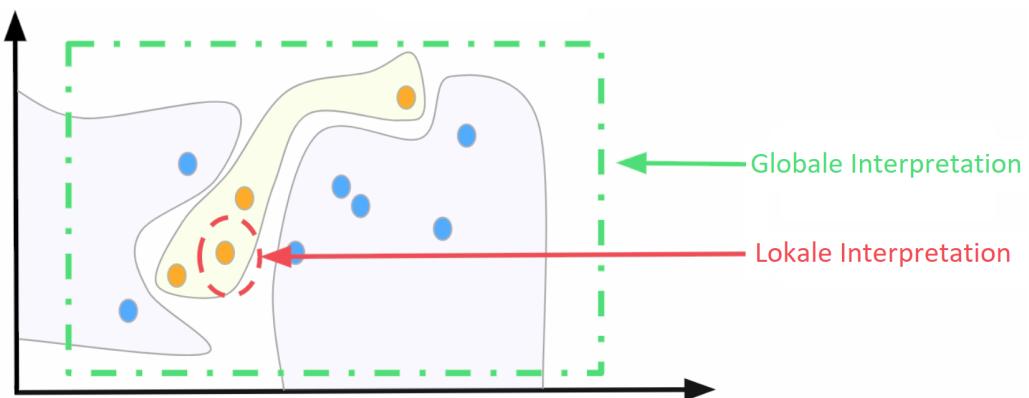


Abbildung 3.2: Globale und lokale Modellinterpretierbarkeit [CP18]

Im Gegensatz zu lokalen Erklärungen gestaltet sich das Erreichen einer globalen Erklärbarkeit in der Praxis als äußerst schwere Aufgabe. Selbst für lineare Modelle, also Algorithmen, die eine intrinsische Interpretationsfähigkeit haben, wird die Visualisierung oder Analyse der Interaktionen von Merkmalen bei mehr als drei Dimensionen sehr schwierig. Das führt dazu, dass die gesamte Modellstruktur von Menschen nicht vollständig erfasst werden kann. Es ist jedoch möglich, bei einigen Modellen nur Teilbereiche zu berücksichtigen, wodurch sie zumindest auf modularer Ebene global interpretierbar werden. Je nach Modell sind die interpretierbaren Bestandteile un-

terschiedlich, beispielsweise Gewichte für lineare Modelle oder Aufteilungen und Blattknoten für Entscheidungsbäume [[MC20], [HG19]].

In Bezug auf die in 3.1 vorgestellte Modelkomplexität fördern die lokalen Interpretationen im Rahmen von einzelnen Vorhersagen das Verständnis für das Verhalten des ansonsten komplexen Modells. Da die lokal generierte Erklärung entsprechend aus einer kleinen Region stammt, hängt die Vorhersage möglicherweise nur linear oder monoton von einigen Merkmalen ab – auch wenn es sich im Allgemeinen um ein nichtlineares, nichtmonotones Modell mit komplexen Abhängigkeiten zwischen den Merkmalen handelt. Linearität bedeutet hier, dass sich der Wert der Antwortvariablen für jede Änderung des Wertes der Prädiktorvariablen um einen konstanten Betrag ändert. Durch die Monotonie wird garantiert, dass sich, falls der Wert der Prädiktorvariablen in eine bestimmte Richtung ändert, auch der Wert der Antwortvariablen in die gleiche bzw. in die entgegengesetzte Richtung ändern wird. Aus diesem Grund können die lokalen Erklärungen im Vergleich zu globalen Erklärungen genauer sein, aber auch weil globale Interpretationen in einigen Fällen approximativ sein können oder auf Durchschnittswerten basieren. Daher wird auch die beste Analyse eines Modells durch die Kombination der Ergebnisse globaler und lokaler Interpretationstechniken erreicht [[MC20], [HG19]].

## 3.7 Erklärungsmethoden

Mittlerweile wurde schon eine Vielzahl an Arbeiten publiziert, die durch wissenschaftliche Grundlagenforschung einen großen Teil zur Vereinheitlichung im Bereich der Erklärbaren Künstlichen Intelligenz beigetragen haben. Es gibt eine Reihe von Erklärungsmethoden und trotz der Bemühungen zum Erreichen besserer Übersichtlichkeit der Theorie werden die Methoden zum Teil unterschiedlich gruppiert [[AB18], [DL18], [SM19], [GB19]]. Demgemäß dient auch die folgende Auflistung nicht als vollständige Übersicht der aktuell vorgestellten Erklärungsmethoden, sondern es werden nur die gebräuchlichsten modellagnostischen Methoden aufgeführt. Wie bereits bekannt, werden Post-hoc modellagnostische Analysen bei Modellen angewendet, die selbst nicht transparent sind, bereits trainiert wurden und nicht auf einen bestimmten Lernalgorithmus beschränkt sind. Die Erklärungsmethoden lassen sich grob in drei Kategorien einteilen:

### *Visualisierung:*

Um das Modellverhalten zu erklären, werden bei der visuellen Interpretation Diagramme verwendet. Dabei wird mithilfe einer grafischen Darstellung die Beziehung zwischen einer oder mehreren Eingabevariablen und den Vorhersagen eines Black-Box-Modells visualisiert. Damit werden dem Endbenutzer Informationen bereitgestellt, die die Bedeutung von Merkmalen für die endgültigen Entscheidungen des Modells aufzeigen [NS19].

### *Ersatzmodell:*

Ein Ersatz- oder Stellvertretermodell ist ein einfaches Modell, das darauf trainiert ist, die Vorhersagen eines Black-Box-Modells zu approximieren. Das Ersatzmodell ist häufig ein intrinsisch interpretierbares Modell, beispielsweise eine lineare Regression oder ein Entscheidungsbaum, welches auf die ursprünglichen Eingaben und Vorhersagen eines komplexen Modells trainiert wird. Die Ergebnisse des Ersatzmodells sind leichter zu verstehen und tragen dazu bei, durch dessen Interpretation Schlussfolgerungen über das Black-Box-Modell zu ziehen. Diese Methode wird zwar auf Englisch hauptsächlich *Surrogate Models* genannt, es gibt aber auch Arbeiten, die diese Technik als *Proxy Models*, *Shadow Models*, *Model Simplification* oder *Mimic Learning* bezeichnen [[GB19], [HP18], [AD19], [DL18]].

### *Merkmalsrelevanz:*

Diese Erklärungsmethode zielt darauf ab, die Erklärung eines Modells auf Grundlage von Be-

rechnungen zu Relevanz, Einfluss oder Wichtigkeit von Merkmalen zu liefern. Durch den zugewiesenen Wert kann aufgezeigt werden, welche der Eingabemerkmale des Modells einen positiven oder negativen Einfluss auf das Ergebnis hatten. Dabei bestehen mehrere Verfahren zur Bestimmung der Merkmalsbeiträge, und es gilt stets: je höher die Bewertung, desto bedeutender ist das Merkmal und desto mehr dient es auch als wichtiger Erklärungsfaktor. Auch hier existieren unterschiedliche Namen für dieselbe Technik: Die *Feature Relevance* Erklärungsmethode ist auch unter Bezeichnungen wie *Feature Attribution*, *Feature Contribution* oder *Feature Importance* zu finden [[AC19], [NS19], [BX20]].

Es ist noch erwähnenswert, dass die Erklärungsmethoden grundsätzlich sowohl einen globalen als auch lokalen Geltungsbereich haben können. Nachdem nun die Methoden zur Generierung von modellagnostischen Erklärungen eingeführt sind, werden anschließend in den nächsten Kapiteln 4 und 5 die ausgewählten Erklärungsmodelle vorgestellt, die jeweils zu einer bestimmten Kategorie gehören. Dabei werden die Zuverlässigkeit und die Grenzen dieser Modelle ausführlich diskutiert. Bevor die Erklärungsmodelle im Detail betrachtet werden können, ist es notwendig, sie unter die bestehenden Erklärungsarten einzuordnen.

## 3.8 Erklärungsarten

Dieser Abschnitt befasst sich mit der Art der erzeugten Erklärungen, wobei auch hier die Aufmerksamkeit nur auf modellagnostische Erklärungstechniken gerichtet wird. Im Allgemeinen lassen sich die Erklärungsarten laut der wissenschaftlichen Publikationen von Lipton [LZ16] und Arrieta et al. [AD19] einteilen in folgende Typen:

### *Textuelle Erklärung:*

Bei den textuellen Erklärungen wird die Erklärung in Form einer Beschreibung in natürlicher Sprache generiert. Diese Repräsentation der Erklärung gehört zu einer der gängigsten Erklärungsarten, da auch Menschen die getroffenen Entscheidungen oft verbal begründen.

### *Visuelle Erklärung:*

Im Gegensatz dazu können die generierten Erklärungen visuell mittels unterschiedlicher Darstellungen präsentiert werden. Die einfache Veranschaulichung durch Diagramme oder Grafiken ermöglicht es, den Einfluss von Merkmalen für Menschen verständlich und interpretierbar zu demonstrieren. Neben der grafischen Visualisierung existieren noch verschiedene Verfahren, die bei bildbasierten ML-Modellen verwendet werden. Bei dieser Form der Visualisierung werden zunächst die Bilder als Eingabedaten in bestimmte Bereiche eingeteilt. Dann wird durch Hervorhebung wichtiger Regionen in einem Bild der Einfluss von für die Vorhersage wichtigen Pixeln illustriert.

### *Beispielgebende Erklärung:*

Die beispielgebenden Erklärungen konzentrieren sich dagegen hauptsächlich auf die Extraktion repräsentativer Beispiele. Um die Entscheidungen des Modells zu erklären, wird hier nach ähnlichen Fällen in den Eingabedaten gesucht. Das Auswählen von bestimmten Instanzen des Datensatzes für die Erklärung ist das Äquivalent für eine menschliche Begründung aufgrund von beispielhaften Erfahrungen.

Im Englischen werden diese Erklärungsarten als *text explanation*, *visual explanation* und *explanation by example* bezeichnet und treten auch häufig in Kombinationen auf.

## 4 Local Interpretable Model-Agnostic Explanations

---

In diesem Kapitel wird die Methode der Local Interpretable Model-Agnostic Explanation, auch bekannt als LIME, vorgestellt. Anfänglich wird in Abschnitt 4.1 die wesentliche Idee der LIME-Methode erläutert. Als Nächstes findet in Abschnitt 4.2 eine Einführung in die von LIME verwendete interpretierbare Repräsentation statt. Danach wird in Abschnitt 4.3 das allgemeine LIME-Verfahren vorgeführt. Dies dient als Basis für den Abschnitt 4.4 mit einer Vorstellung des LIME-Algorithmus basierend auf dem Datentyp. Im Anschluss daran wird in Abschnitt 4.5 der Submodular-Pick-Algorithmus, häufig auch als SP-LIME bezeichnet, behandelt. Zum Ende des Kapitels werden in Abschnitt 4.6 die Stärken und Schwächen der LIME-Methode ausführlich untersucht.

### 4.1 Einleitung

Local Interpretable Model-Agnostic Explanation (LIME) wurde von Marco Tulio Ribeiro, Sameer Singh, und Carlos Guestrin entwickelt und erstmals im Jahr 2016 in einem Artikel mit dem Titel „Why Should I Trust You? Explaining the Predictions of Any Classifier“ veröffentlicht [RS16a]. Wie der Name impliziert, handelt es sich beim LIME-Algorithmus um ein modellagnostische Erklärungsmodell. Dementsprechend kann das Ergebnis eines beliebigen Vorhersagemodells auf eine interpretierbare und zuverlässige Weise erklärt werden. Zudem handelt es sich hier um ein lokales Ersatzmodell, sodass ein komplexes Modell lokal durch ein einfaches Modell approximiert wird, das leicht zu interpretieren ist. Dabei konzentriert sich LIME nur auf eine einzige Vorhersage und findet dafür eine lokal treue Erklärung, wobei die lokale Treue verständlicherweise nicht die globale Treue – für das gesamte Modellverhalten – impliziert. In mehreren Arbeiten wurde schon gezeigt, dass die Entscheidungsgrenze eines Black-Box-Modells in einer lokalen Region viel einfacher bzw. sogar linear sein kann [[BS10], [LR18], [SK10]]. Daher beruht beim LIME-Algorithmus die lokale Erklärbarkeit auch auf der Annahme, dass die Entscheidungsgrenze eines Vorhersagemodells – egal wie komplex das Modell auf globaler Ebene ist – um die zu erklärende Instanz herum nahezu linear ist.

Der Hauptbestandteil der LIME-Erklärung besteht darin, jedem Merkmal der Eingabedaten eine Zahl zuweisen, mit deren Hilfe der Einfluss des Merkmals auf die Vorhersage quantifiziert werden kann. Es wird eine Liste von Erklärungen erzeugt, die den Beitrag jedes Merkmals zur Vorhersage einer Instanz widerspiegelt. Diese Form der Erklärung, ob also ein Merkmal einen positiven oder negativen Einfluss auf das Ergebnis der lokalen Ersatzmodell hat, hängt vom Datentyp der zu erklärenden Instanz ab. Die Ausgabe von LIME beruht dabei auf der sogenannten interpretierbaren Repräsentation von Daten, mit der verständliche Erklärungen generiert werden können. Es sind mehrere Implementierungen von LIME in Python und R verfügbar. Falls in diesem Kapitel Kommentare zum Implementierung gemacht werden, beziehen sie sich auf die von den ursprünglichen Autoren entwickelte Python-Implementierung. Allgemein bezieht sich das Kapitel hauptsächlich auf die Arbeit von Ribeiro et al. aus [RS16a] und Beiträge aus anderen Arbeiten werden entsprechend gekennzeichnet. Für ein besseres Verständnis der Funktionsweise

von LIME zeigt die folgende Abbildung ein Beispiel, wie ein Prozess für die Erklärung einzelner Vorhersagen bei tabellarischen Daten veranschaulicht werden kann.

### Beispiel 4.1.1

Ein Modell sagt anhand von Daten aus einem vorab ausgefüllten Fragebogen voraus, ob ein Patient an Grippe leidet oder nicht. Die Eingabeparameter für das Vorhersagemodell können beispielsweise numerische Eingabedaten wie „Alter“ oder „Gewicht“ und kategoriale Eingabedaten wie „Niesen“ oder „Kopfschmerzen“ (1 für Ja und 0 für Nein) sein. Anschließend werden mithilfe von LIME in der Anamnese des Patienten die Symptome hervorgehoben, die in diesem Fall zur Vorhersage „Patient hat die Grippe“ geführt haben. LIME liefert eine kleine Liste von Symptomen mit relativen Gewichten, hier die drei wichtigsten Merkmale, die zur „Grippe“-Vorhersage geführt haben. „Niesen“ und „Kopfschmerzen“ sind Symptome, die zur Vorhersage beigetragen haben (in grün), wobei „keine Müdigkeit“ gegen die Vorhersage spricht (in rot). Mit diesen Informationen und einem besseren Verständnis des Entscheidungsprozesses des Modells ist der Entscheidungsträger, hier ein Mediziner, in der Lage zu überprüfen, ob er der Vorhersage des Modells vertrauen kann [RS16a]. In Abbildung 4.1 wird die Vorgehensweise im Prozess zur LIME-Erklärung einzelner Vorhersagen dargestellt.

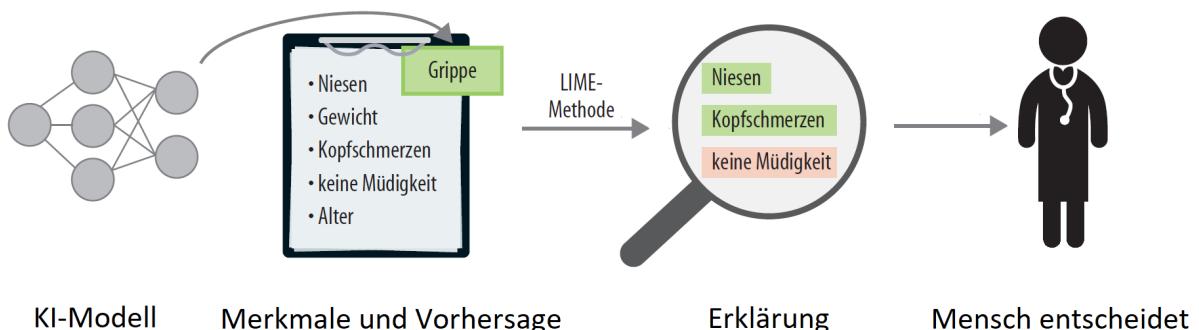


Abbildung 4.1: Generierte LIME-Erklärung einer einzelnen Modellvorhersage [HA18]

## 4.2 Interpretierbare Repräsentation

Um die Interpretierbarkeit in den generierten Erklärungen aufrechtzuerhalten, verwendet LIME die interpretierbare Repräsentation von Daten. Dabei ist diese Form der Datendarstellung für Menschen leicht verständlich und wird dementsprechend als interpretierbar bezeichnet. Die folgenden Ausführungen stammen von Sokol et al. aus [SH19] und [SF20b]. Im Allgemeinen wird ein Black-Box-Modell als Funktion  $f : X \rightarrow Y$  betrachtet, wobei  $X$  die Eingabedaten bzw. der Merkmalsraum und  $Y$  der Ausgaberaum mit Vorhersagen des Black-Box-Modells  $f$  ist. Um die Schreibweise von Ribeiro et al. beizubehalten, wird hier bei der Betrachtung für die Entscheidung einer Instanz nicht  $x^{(i)} \in X$  sondern  $x \in X$  benutzt. Somit gilt von nun an  $x \in X$  als die originale Darstellung des zu erklärenden Datenpunktes.

Für das LIME-Verfahren gilt  $X = \mathbb{R}^d$  als  $d$ -dimensionaler Merkmalsraum und  $Y = \mathbb{R}$  als Ausgaberaum. Zusammenfassend gilt  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  mit  $x \in \mathbb{R}^d$  als die ursprüngliche Darstellung einer zu erklärenden Instanz. Mithilfe der Transformation auf einen anderen  $d'$ -dimensionalen Merkmalsraum durch die Abbildungsfunktion  $IR : X \rightarrow X'$  wird aus  $x \in X$  nun  $x \in X'$  bzw.  $x \in \mathbb{R}^{d'}$ , wobei die Funktion  $IR$  eine Abkürzung für interpretierbare Repräsentation darstellt. Um sicherzustellen, dass die von LIME generierte Erklärung nachvollziehbar und interpretierbar für Menschen ist, muss die Dimension der interpretierbaren Darstellung nicht notwendigerweise die

gleiche wie die des ursprünglichen Merkmalsraums sein. Wie diese interpretierbare Darstellung  $x \in \mathbb{R}^{d'}$  nun aussehen soll, hängt von der Art der Daten ab. Hierbei muss zudem die Datentransformation von der ursprünglichen Domäne  $X$  in eine interpretierbare Darstellung  $X'$  bijektiv sein und eine entsprechende, eindeutig definierte Umkehrfunktion  $IR^{-1} : X' \rightarrow X$  haben. Genau diese beiden Anforderungen werden sowohl für Text- als auch für Bilddaten erfüllt:

*Textdaten:*

Für eine interpretierbare Darstellung der Textdaten verwendet LIME den Bag-of-Words-Ansatz. Dabei kann die zu erklärende Instanz  $x \in X$ , beispielsweise ein Satz, als ein binärer Vektor dargestellt werden. Mit den zwei Zuständen wird durch 1 das Vorhandensein und 0 das Fehlen von eindeutigen Wörtern in diesem Satz angezeigt. Die in eine interpretierbare Datendarstellung umgewandelte Instanz gilt nun formal als  $X' = \{0, 1\}^{d'}$  mit  $d'$  als Anzahl der Wörter ist, die die zu erklärende Instanz enthält. Bei der Textklassifizierung gilt somit  $x' \in \{0, 1\}^{d'}$  als die interpretierbare Repräsentation für einen ausgewählten Datenpunkt.

### Beispiel 4.2.1

Als Beispiel wird die binäre Klassifikation von Katastrophen-Tweets betrachtet. Der Datensatz entstammt einem Kaggle-Wettbewerb [KDDT], wobei die Daten vom Unternehmen Appen [ADRM] erstellt und zur Verfügung gestellt wurden. Es beinhaltet 10.000 Tweets, die von Menschen klassifiziert wurden. Die Zielvariable gibt mit 1 oder 0 an, ob es sich bei einem Tweet um eine echte Katastrophe handelt oder nicht. Beispiele für Keine-Katastrophen-Tweets sind „I love fruits“ oder „What a nice hat?“, während Tweets wie „We're shaking... It's an earthquake“ oder „Forest fire near La Ronge Sask. Canada“ als Katastrophen-Tweets klassifiziert werden. Das Ziel ist hier, die korrekten Katastrophen-Wörter aufzugreifen und als „relevant“ zu klassifizieren. In Abbildung 4.2 wird der ausgewählte Tweet bzw. die Instanz  $x$  oben im Bild dargestellt. Die entsprechend interpretierbare Bag-of-Words-Darstellung des Tweets  $x'$  mit  $d' = 14$  als Anzahl der Wörter wird unten im Bild illustriert.

hiroshima one of history's worst examples of mass murder hiroshima war atombomb japan

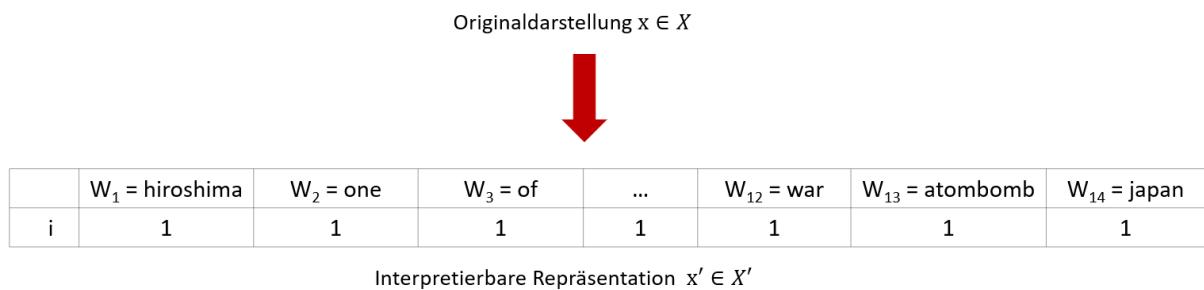


Abbildung 4.2: Originaldarstellung und interpretierbare Repräsentation der Textdaten

*Bilddaten:*

Eine ähnliche Begründung mit einem binären Vektor gilt auch für Bilddaten, wobei hier die interpretierbare Datendarstellung durch die Superpixel-Segmentierung illustriert wird. Superpixel [RM03] sind auf Ähnlichkeit basierende Erfassung von Pixeln, wobei es zum Beispiel in Form einer Gruppe von verbundenen Pixeln mit ähnlichen Farben auftreten kann Superpixel. Für die LIME-Methode stehen zwar mehrere Segmentierungsalgorithmen zur Verfügung, die Autoren haben aber für die Python Standardimplementierung das Segmentierungsverfahren Quick-Shift [VS08] mit den Parametern `kernel_size=4`, `max_dist=200` und `ratio=0.2` ausgewählt. Entsprechend dieser Parametrisierung werden die Superpixel ermittelt und ihre Anzahl hier als

$d'$  deklariert. Wiederum wird mit 1 und 0 auf die Existenz oder Abwesenheit eines Merkmals, hier Superpixels, hingewiesen. Dabei zeigt 1 das Vorhandensein bzw. Belassen des Superpixels wie im Originalbild und 0 das Fehlen bzw. Ausgrauen des Superpixels an. Somit definiert jedes Superpixel ein interpretierbares Merkmal und es gilt  $X' = \{0, 1\}^{d'}$ . Infolgedessen gibt die interpretierbare Repräsentation eines Bildes  $x' \in \{0, 1\}^{d'}$  die Anzahl der betrachteten Superpixel als  $d'$  an.

### Beispiel 4.2.2

Dieses Beispiel stammt aus einem Artikel der Autoren der LIME-Methode und wurde auf der Website O'Reilly [RS16b] veröffentlicht. Um die Funktionsweise von LIME bei der Bildklassifizierung vorzustellen, wird zunächst ein Bild, hier ein Baumfrosch, ausgewählt und die ursprünglichen Darstellung als 3-dimensionale Matrix von Farbkanälen (in Abbildung 4.3 links) in eine interpretierbare Repräsentation aus Superpixeln (in Abbildung 4.3 rechts) umgewandelt. Die Verwendung von Superpixel reduziert somit die Anzahl der zu berücksichtigenden Merkmale. Das heißt, aus dem originalen Merkmalsraum, in dem jedes Pixel als  $(x, y, r, g, b)$  Tupel repräsentiert ist, wird auf den interpretierbaren Merkmalsraum abgebildet, in dem das Originalbild in eingeblendet Superpixeln (den gelb umrandeten Regionen im rechten Bild) dargestellt wird. Eingeblendet bedeutet hier, dass alle Pixelwerte aus dem Originalbild beibehalten wird und alle interpretierbare Merkmale  $SP_1, \dots, SP_{d'}$  mit 1 versehen sind.

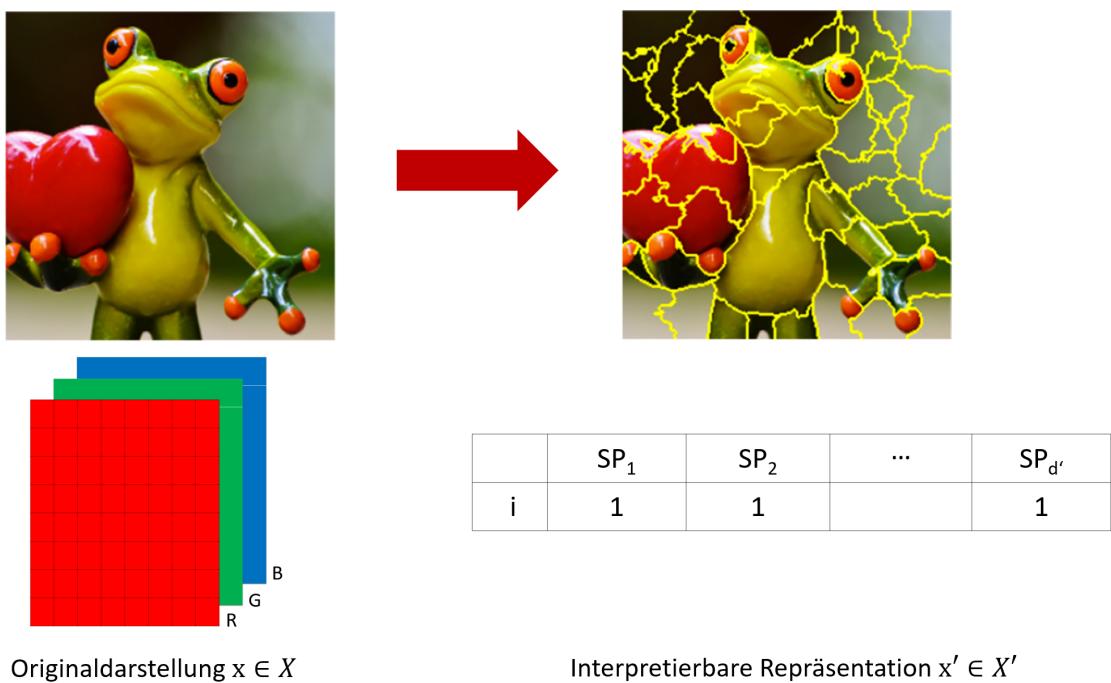


Abbildung 4.3: Transformation eines Bildes in eine interpretierbare Datendarstellung [RS16b]

Neben Text- und Bilddaten lassen sich auch tabellarische Daten in ein interpretierbares Format konvertieren. Die Generierung einer interpretierbaren Datendarstellung hängt bei tabellarischen Daten zusätzlich von der Art des Merkmals ab, wobei LIME sowohl mit kategorialen als auch mit numerischen Merkmalen funktioniert:

*Tabellendaten:*

Um für beide Merkmalstypen eine interpretierbare Datenrepräsentation erzeugen zu können, wird hier – im Gegensatz zu Text- und Bilddaten – Zugriff auf Trainingsdaten benötigt. Das Ziel ist,

für kategoriale Daten eine binäre Variable zu erzeugen, indem die 1 auf die originale Klasse der Instanz und 0 auf eine andere Klasse, die nicht mit der erklärten Instanz übereinstimmt, hinweist. Die Binärisierung der kategorialen Merkmale basiert auf dem Verfahren One-Hot Encoding. Dabei ist die Binärisierung kategorialer Merkmale invertierbar, da die ursprüngliche Darstellung der Instanz aus der binärisierten Form eindeutig wiederhergestellt werden kann. Im Gegensatz dazu ist es schwierig, eine Transformationsfunktion  $IR$  für die interpretierbare Repräsentation der numerischen Merkmale zu definieren, die bijektiv ist und eine eindeutige Umkehrung aufweisen soll. Der beliebteste und auch von LIME verwendete Ansatz ist zunächst die Diskretisierung des Merkmals und anschließend die Binärisierung durch One-Hot Encoding. Aufgrund der standardmäßigen Python-Implementierung werden die numerischen Merkmale in Quartile diskretisiert, wobei noch optional in Dezile oder basierend auf Entropie diskretisiert werden kann. Anhand des untenstehenden Beispiels fällt auf, dass bei diesem Ansatz zwar ein Merkmal eindeutig einer Kategorie zugeordnet werden kann, die umgekehrte Prozedur jedoch nicht eindeutig definiert ist. Daher ist auch hier die Transformationsfunktion  $IR$  zwar surjektiv, aber nicht injektiv und somit auch nicht bijektiv, was zur Folge hat, dass die numerischen Merkmale, die diskretisiert (und binärisiert) wurden, nicht eindeutig in ihre ursprüngliche Darstellung  $X$  rekonstruiert werden können. Hierauf wird mehr in 4.4 eingegangen. Auch hier wird am Ende die interpretierbare Darstellung der numerischen Daten anhand einer binären Variable erstellt, wobei die 1 darauf hinweist, dass das Merkmal sich in der gleichen Kategorie wie die zu erklärende Instanz befindet. Zudem basiert die Dimension  $d'$  der interpretierbaren Darstellung  $x'$  auf derselben Dimension  $d$  wie im Originaldatensatz, das heißt, im Vergleich zu anderen Datentypen wird hier die gleiche Anzahl von Merkmalen für die interpretierbare Repräsentation, also  $d' = d$ , verwendet.

### **Beispiel 4.2.3**

*Es folgt ein einfaches Beispiel, das versucht, tabellarische Daten in interpretierbare Teile zu unterteilen. Hier sind nicht nur kategoriale, sondern auch numerische Daten vorhanden. Beispielsweise wird ein numerisches Merkmal „Kontostand“  $x_i$  mit dem Wert 850 betrachtet. Es kann dann z.B. in vier Kategorien diskretisiert werden:  $(-\infty, 500]$ ,  $(500, 1200]$ ,  $(1200, 5600]$  und  $(5600, \infty)$ , die als  $[0, 1, 0, 0]$  binärisiert werden, was anzeigt, dass  $x_i = 850$  in die zweite Kategorie fällt. Der Grund für die Diskretisierung der numerischen Daten besteht darin, die Erklärbarkeit der Ausgabe zu verbessern, indem die Merkmale zusammengefasst beschrieben werden, z.B. in Form von „ $500 < \text{Kontostand} < 1200$ “. Die daraus resultierende Darstellung ist in Abbildung 4.4 illustriert. Für dieses Beispiel wird die interpretierbare Darstellung  $x'$  später in Abschnitt 4.4 ausführlich erläutert.*

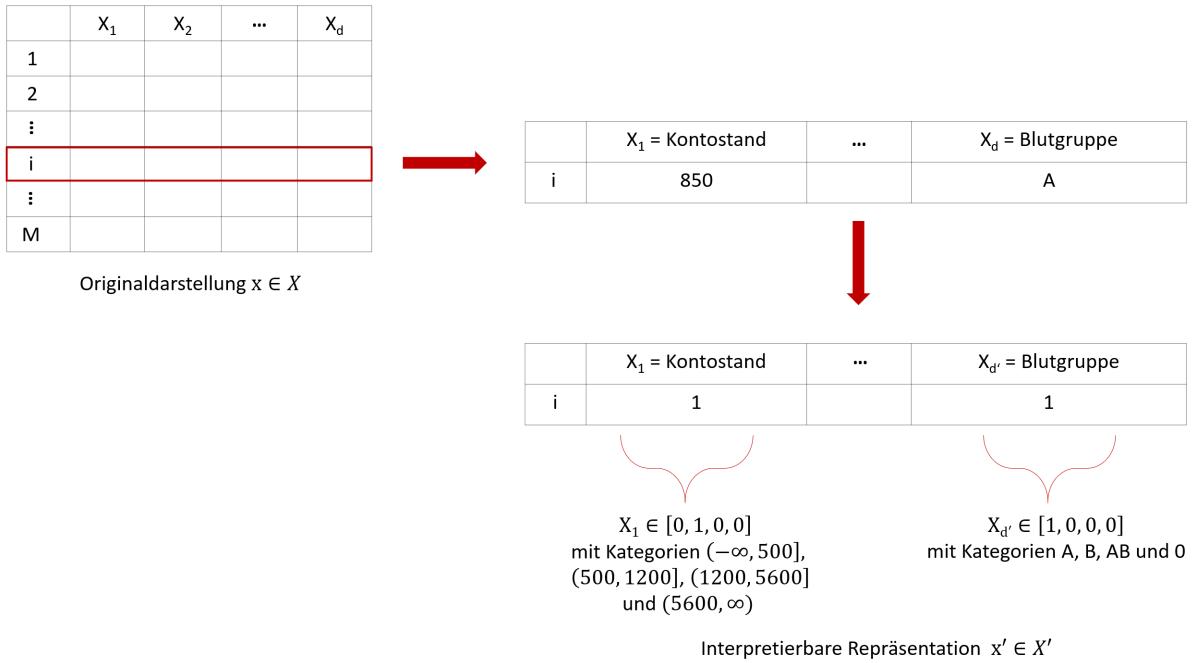


Abbildung 4.4: Zuordnung der Tabellendaten von originaler zu interpretierbarer Datendarstellung

### 4.3 Der allgemeine LIME-Ansatz

Wie schon angeführt, sei  $X = \mathbb{R}^d$  der Merkmalsraum,  $x \in \mathbb{R}^d$  die originale Darstellung einer zu erklärenden Instanz und  $x' \in \mathbb{R}^{d'}$  die dazugehörige interpretierbare Präsentation. Zudem sei  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  das zu erklärende Black-Box-Modell, wobei  $f$  bei Klassifikationsaufgaben entweder anhand der Wahrscheinlichkeitswerte oder einer binären Variable die Zugehörigkeit von  $x$  zu einer bestimmten Klasse anzeigt und bei Regressionsaufgaben die Zielvariablen darstellt. Es sollte erwähnt werden, dass LIME bei einer Mehrklassen-Klassifizierung jede Klasse separat erklärt, so dass  $f(x)$  die Vorhersage der relevanten Klasse ist. Somit wird auch nur eine Erklärung für den Einfluss von Merkmalen im Hinblick auf die vorhergesagte Klasse zur Verfügung gestellt.

Zusätzlich wird mit  $X' = \mathbb{R}^{d'}$  eine Erklärung als ein Modell  $g : \mathbb{R}^{d'} \rightarrow \mathbb{R}$  bzw.  $g : \{0, 1\}^{d'} \rightarrow \mathbb{R}$  definiert. Hierbei gilt insbesondere für das Modell  $g \in G$ , dass  $G$  eine Klasse von interpretierbaren Modellen, z.B. lineare Modelle oder Entscheidungsbäume, ist. Wird beispielsweise als  $g$  ein lineares Modell verwendet, so kann die Vorhersage von  $f$  für einen bestimmten Datenpunkt  $x$  durch die Analyse von Koeffizienten des lokalen linearen Modells erklärt werden.

Um sicherzustellen, dass das interpretierbare Modell selbst nicht zu komplex wird, wird für das Modell  $g \in G$  ein Strafterm eingeführt. Dabei gilt  $\Omega(g)$  als ein Maß für die Komplexität der Erklärung  $g$  und wird daher als das Gegenteil von Interpretierbarkeit betrachtet. Darüber hinaus findet die Regularisierung statt, um einfachere Modelle aus der Klasse  $G$  zu bevorzugen und folglich das interpretierbare Modell so einfach wie möglich zu gestalten. Das Komplexitätsmaß kann beispielsweise die Baumtiefe für Entscheidungsbäume oder die Anzahl der Gewichte ungleich Null für lineare Modelle sein.

Des Weiteren wird, um eine Erklärung für eine einzelne Vorhersage von  $f$  durch das Anpassen eines einfacheren, interpretierbaren Erklärungsmodells  $g$  zu erhalten, noch ein neuer Datensatz lokal um den Datenpunkt  $x$  durch Permutieren der interpretierbaren Darstellung des Datenpunktes  $x'$  erstellt. Diese interpretierbare Stichprobe an Daten  $z' \in X'$  wird dann für die Definition der

Lokalität um  $x$  verwendet. Mithilfe eines Ähnlichkeitsmaßes  $\pi_x(z)$  mit  $\pi : X' \times X' \rightarrow \mathbb{R}$  werden den permutierten Instanzen  $z'$  Gewichte entsprechend ihrer Nähe zum interessierenden Punkt  $x'$  zugewiesen. Um die Schreibweise von Ribeiro et al. weiterhin beizubehalten, wird im weiteren Verlauf  $\pi_x(z)$  anstatt  $\pi_{x'}(z')$  verwendet. Für die Berechnung von  $\pi_x(z)$  werden noch Distanzfunktion und Kernel-Breite definiert. Es wird hier entsprechend dem vorhandenen Datensatz sowohl die lokale synthetische Stichprobe  $z$  unterschiedlich erzeugt als auch das Ähnlichkeitsmaß  $\pi_x(z)$  auf unterschiedliche Weise berechnet.

Letztendlich soll  $L(f, g, \pi_x(z))$  ein Maß dafür sein, wie untreu das interpretierbare Modell  $g$  bei der Annäherung an das Vorhersagemodell  $f$  in der durch  $\pi_x(z)$  definierten lokalen Region ist. Eine Möglichkeit, die lokale Treue des lokalen Ersatzmodells zu quantifizieren, ist die Analyse von  $R^2$ -Wert (näheres dazu in Abschnitt 6.3.2). Die Verlustfunktion  $L$  kann z.B. über die Funktion mittlere quadratische Abweichung (Mean Squared Error – MSE) berechnen, wie nahe die Erklärung bzw. Vorhersage des interpretierbaren Modells  $g$  an der Vorhersage des Black-Box-Modells  $f$  liegt. Mathematisch ausgedrückt, muss schließlich die Verlustfunktion für die Untreue  $L(f, g, \pi_x(z))$  minimiert (bzw. die lokale Treue maximiert) werden, wobei die Komplexität  $\Omega(g)$  niedrig genug ist, um von Menschen interpretiert zu werden. Die LIME-Erklärung  $\xi(x)$  ergibt sich daher aus der Lösung des Optimierungsproblems:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} L(f, g, \pi_x(z)) + \Omega(g) . \quad (4.1)$$

Diese Formel stellt mittels des ersten und zweiten Terms einen Kompromiss zwischen Treue und Interpretierbarkeit (Fidelity-Interpretability Trade-off) dar und kann mit verschiedenen Klassen von potenziell interpretierbaren Modellen  $G$ , zu minimierenden Verlustfunktionen  $L(f, g, \pi_x(z))$  und niedrig zu haltenden Komplexitätsmaßen  $\Omega(g)$  verwendet werden. Bei der tatsächlichen Anwendung wird aber nur der erste Term optimiert, da das Kriterium für das Komplexitätsmaß  $\Omega(g)$  meist nur vereinfacht dargestellt werden kann. Beispielsweise wird bei einem linearen Modell  $g$  nur eine begrenzte Anzahl  $K$  von Merkmalen ausgewählt, wodurch die Erklärung nur noch auf die  $K$  wichtigsten Merkmale beschränkt wird, die die vorhergesagten Ergebnisse am besten beschreiben.

Für eine bessere Übersichtlichkeit werden im Folgenden die wesentlichen Schritte des verallgemeinerten LIME-Algorithmus aufgeführt:

1. Mithilfe der Abbildung  $IR : X \rightarrow X'$  wird eine interpretierbare Darstellung  $x' \in X'$  des Datenpunkts  $x$  bestimmt, die einen Datenpunkt aus seinem originalen Merkmalsraum  $X$  in die interpretierbare Darstellung  $X'$  transformiert. Wie bereits in Abschnitt 4.2 erläutert, wird die interpretierbare Repräsentation durch einen nur aus Einsen bestehenden binären Vektor  $x' = (1, \dots, 1)$  dargestellt.
2. Es werden  $N$  permutierte Stichproben  $z'_i \in X'$  mit  $i = 1, \dots, N$  der interpretierbaren Version der zu erklärenden Instanz  $x'$  erzeugt; dies kann in Form einer binären Matrix  $N \times d'$  aufgezeigt werden. Diese Form der Datengenerierung unterscheidet sich je nach Datentyp und wird in Abschnitt 4.4 detaillierter ausgearbeitet.
3. Die interpretierbare Darstellung der generierten Stichproben  $z'_i$  wird unter Verwendung der Umkehrfunktion  $IR^{-1} : X' \rightarrow X$  zurück in die ursprüngliche Darstellung  $X$  als  $z_i \in X$  mit  $i = 1, \dots, N$  invertiert.
4. Das komplexe Modell  $f$  wird auf die permutierten Beobachtungen  $z_i$  angewendet und es werden Vorhersagen  $f(z_i) \in \mathbb{R}$  anhand von Stichprobendaten getroffen.

5. Die Abstände zwischen den Stichproben  $z'_i$  und dem zu erklärenden Datenpunkt  $x'$  werden mit einer Distanzfunktion  $D$  gemessen und anschließend mit einem exponentiellen Kernel als Ähnlichkeitsmaß  $\pi_x(z)$  und  $\sigma$  als Kernel-Breite berechnet. Je näher die Stichproben an der originalen Instanz liegen, desto höhere Gewichtung erhalten diese Beobachtungen.
6. Die interpretierbaren Stichproben  $z'_i$  und dazugehörigen Black-Box-Modell-Prognosen  $f(z_i)$  werden als Trainingsdaten für ein gewichtetes, interpretierbares Modell  $g$  verwendet. Zuletzt werden aus der Analyse des einfacheren Modells Schlussfolgerungen für das lokale Verhalten des komplexeren Modells gezogen.

Obwohl die Formulierung (4.1) mit unterschiedlichen interpretierbaren Modellen als  $g$  verwendet werden kann, wird eine sparsame Modellierung angestrebt, um eine hohe Komplexität zu vermeiden. Hierfür wählen die Autoren ein spärliches lineares Modell als erklärbare Modell  $g$  aus. Da in der Praxis der Benutzer die Komplexität  $\Omega(g)$  durch Auswahl von  $K$  bestimmt, konzentriert sich die folgende Untersuchung allein auf die Treue des Modells. Hier sei  $G$  die Klasse der linearen Modelle mit  $g(z') = w_g z'$ . Zudem sei das Ähnlichkeitsmaß  $\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$  ein exponentieller Kernel, der auf einer Distanzfunktion  $D$  mit der Kernel-Breite  $\sigma$  definiert ist. Als  $L$  wird die folgende lokal gewichtete quadratische Verlustfunktion verwendet:

$$L(f, g, \pi_x(z)) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2 \quad (4.2)$$

Für die Komplexität  $\Omega(g)$  wird eine Obergrenze  $K$  für die Anzahl der Nicht-Null-Elemente in  $w_g$  durch

$$\Omega(g) = \infty 1[\|w_g\|_0 > K] \quad (4.3)$$

festgelegt. Dabei ist  $\|w_g\|_0$  die Anzahl der Nicht-Null-Elemente des Vektors  $w_g$  und es gilt  $K \in \mathbb{N}$ . Diese Gleichung kann aber nicht direkt gelöst werden und wird daher unter Verwendung von einer gewichteten Merkmalsauswahltechnik für die  $K$  approximiert. Es gibt mehrere Ansätze und dazugehörige Implementierungen, die zur Auswahl der  $K$  wichtigen Merkmale führen. Hier wird die  $L1$ -Regularisierung ausgewählt und durch eine gewichtete LASSO-Methode [TR96] mit  $K$  Koeffizienten ungleich Null durchgeführt. Die Funktionen (4.2) und (4.3) werden dann in Gleichung (4.1) eingesetzt, wobei durch die Festsetzung von  $\Omega(g) = 0$  wegen  $\|w_g\|_0 \leq K$  die Analyse vereinfacht wird [GL20]:

$$\xi(x) = \operatorname{argmin}_{w_g} \sum_{i=1}^N \exp\left(-\frac{D(x, z_i)^2}{\sigma^2}\right) (f(z_i) - w_g z'_i)^2 \quad (4.4)$$

Damit wird die Anzahl der zu berücksichtigenden Variablen der LASSO-Regression bestimmt, die z.B. für Textdaten die Anzahl der Wörter, für Bilddaten die Anzahl der Superpixel und für Tabellendaten die Anzahl der Spalten bzw. Merkmale sind. Anschließend werden die von LASSO ausgewählten  $K$  Koeffizienten  $w_g$  als die wichtigsten Koeffizienten des linearen Modells  $g(z')$  zum Trainieren eingesetzt. Die Python- und R-Implementierungen verwenden beide als interpretierbares Standard-Modell  $g$  die Ridge Regression [HK70], also  $L2$ -Regularisierung, mit einem Regularisierungsparameter von 1. Zudem wurde in Python standardmäßig die Stichprobengröße  $N$  bei Tabellen- und Textdaten 5.000 und bei Bilddaten 1.000 festgelegt. Schließlich werden die

endgültigen Koeffizienten  $w_g$  über gewichtete kleinste Quadrate geschätzt, sodass LIME die ausgewählten  $K$  Merkmale und die dazugehörigen Gewichte im lokalen Modell  $g(z')$  als Erklärung für  $f(x)$  zurückgibt. Der folgende Algorithmus 1 veranschaulicht solch eine Prozedur.

---

**Algorithm 1** Spärliche lineare Erklärungen mit LIME

---

**Require:** Black-Box Modell  $f$ , Stichprobengröße  $N$   
**Require:** Instanz  $x$ , dazugehörige interpretierbare Darstellung  $x'$   
**Require:** Ähnlichkeitsmaß  $\pi_x(z)$ , Länge der Erklärung  $K$

- 1:  $\mathcal{Z} \leftarrow \{\}$
- 2: **for**  $i \in \{1, 2, 3, \dots, N\}$  **do**
- 3:      $z'_i \leftarrow \text{sample\_around}(x')$
- 4:      $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
- 5: **end for**
- 6:  $w \leftarrow \text{K-LASSO}(\mathcal{Z}, K)$      ▷ mit  $z'_i$  als Merkmale,  $f(z)$  als Zielvariable
- 7: **return**  $w$

---

Hier sollte beachtet werden, dass das Trainieren eines gewichteten interpretierbaren linearen Modells  $g$  letztlich nur auf den Daten der durch LASSO ausgewählten  $K$  Merkmale  $z_i$  als Eingabevervariablen und den entsprechenden Black-Box-Modell-Vorhersagen  $f(z_i)$  als Zielvariablen beruht. Die Verlustfunktion gibt an, wie gut das interpretierbare Modell die Black-Box-Vorhersagen annähert, wobei  $L(f, g, \pi_x(z))$  dann minimiert wird, wenn die Ausgabe des interpretierbaren Modells  $g(z'_i)$  in Verbindung mit der Umgebung der zu erklärenden Instanz  $x$  den geringsten Unterschied zur Ausgabe des Black-Box-Modells  $f(z_i)$  aufweist. Dabei wird die Verlustfunktion (4.2) als eine Art Treuemaß betrachtet, mit dem die Qualität von Erklärungen gemessen werden kann. Zum Schluss kann dann die Vorhersage der Black-Box-Modell erklärt werden, indem die Koeffizienten  $w_g$  des lokalen linearen Modells analysiert werden.

Darüber hinaus bedeutet der Schritt `sample_around(x')` im Algorithmus (1) Stichprobenneherhebungen um die interpretierbare Instanz  $x'$  herum. Da diese künstlichen Stichproben durch die exponentielle Kernfunktion  $\pi_x(z)$  in Bezug auf  $x$  gewichtet werden, wird für die Berechnung des Kernels, der die Lokalität definiert, noch eine Distanzfunktion  $D$  benötigt. Wie bei der interpretierbaren Repräsentation in Abschnitt 4.2 hängt auch die Erzeugung der Stichproben und die Auswahl der Distanzfunktion von den verwendeten Daten ab.

## 4.4 Adaption des LIME-Algorithmus an die Datentypen

In diesem Abschnitt werden die lokal synthetisch generierten Stichproben und die verwendeten Distanzfunktionen, basierend auf den Eingabedaten, vorgestellt. Zusätzlich werden die vorgestellten Beispiele aus Abschnitt 4.2 hier bis zur Erstellung der LIME-Erklärung fortgesetzt.

*Textdaten:*

Für die Textklassifizierung wird eine gleichverteilte zufällige Ziehung von Nicht-Null-Elementen der Instanz  $x'$  durchgeführt. Das bedeutet, es werden Wörter nach dem Zufallsprinzip aus der zu erklärenden Instanz entfernt und die restlichen Wörter als  $z'_i$  erfasst. Um die Ähnlichkeit  $\pi_x(z)$  zwischen den Datenpunkten  $x'$  und  $z'_i$  zu berechnen, wird für Textdaten die Kosinus-Distanz als  $D$  verwendet, wobei es als

$$\text{Kosinus-Distanz} = 1 - \text{Kosinus-Ähnlichkeit} = 1 - \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$$

definiert ist [VM15].

#### Beispiel 4.4.1

Um mit dem Beispiel der Katastrophen-Tweets fortfahren, werden nun die permutierten Beobachtungen  $z'_i$  in Abbildung 4.5 dargestellt.

	$W_1 = \text{hiroshima}$	$W_2 = \text{one}$	$W_3 = \text{of}$	...	$W_{12} = \text{war}$	$W_{13} = \text{atombomb}$	$W_{14} = \text{japan}$
1	1	0	0	...	1	1	0
2	0	1	1	...	0	1	1
3	1	1	0	...	1	0	1
:	:	:	:	:	:	:	:
N	1	0	1	...	1	0	0

Abbildung 4.5: Stichprobenerhebung der Textdaten

Um die Wahrscheinlichkeit für die Auswahl einer Klasse mit dem Black-Box-Modell  $f$  prognostizieren zu können, werden die Stichproben aus der interpretierbaren Darstellung  $X'$  in die Originaldarstellung  $X$  invertiert. Dabei wird beispielsweise die erste Stichprobe

$$z'_1 = [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0]$$

als

$$z_1 = \text{"hiroshima worst of war atombomb"}$$

dargestellt und in  $f$  eingesetzt. Bei dem Exemplar aus Abbildung 4.6 macht das logistische Regressionsmodell  $f$  durch die Klassifizierung des Tweets als „relevant“ eine richtige Vorhersage. Bei der Textklassifizierung werden die Wörter mit den höchsten oder niedrigsten Gewichtungen  $w_g$  angegeben, die einen positiven bzw. negativen Einfluss auf das Klassifizierungsergebnis haben. Zusätzlich werden diese Wörter im Satz durch farbliche Markierung in orange bzw. blau hervorgehoben. Mit  $K = 6$  wird dann nur eine begrenzte Anzahl der lokal einflussreichsten Wörter ausgegeben. Dies ist ein Fall, in dem das verwendete Vorhersagemodell hochrelevante Wörter aufgreift und daher auch verständliche Entscheidungen zu treffen scheint. Wird beispielsweise das Wort japan entfernt, so würde die Vorhersage mit 0,13 weniger Zuversicht einen Katastrophen Tweet vorhersagen (Näheres dazu in Abschnitt 6.3.1).

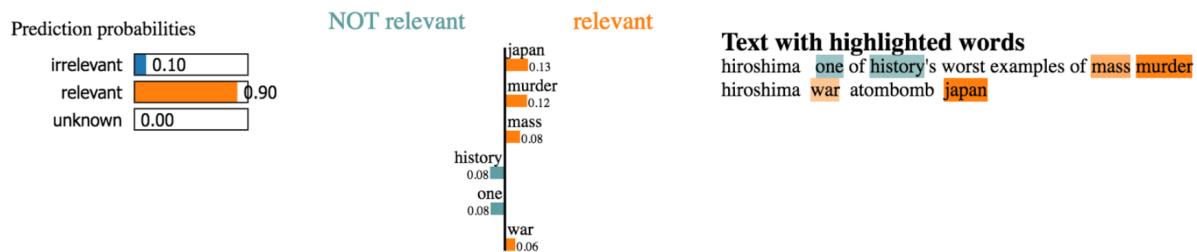


Abbildung 4.6: LIME-Erklärung eines Textklassifizierungsmodells [AE18]

### Bilddaten:

Bei der Erzeugung von Erklärungen für die Klassifizierung von Bildern verhält sich der LIME-Algorithmus ähnlich wie bei Texten. Auch hier wird die Stichprobenerhebung durch eine gleichverteilte zufällige Ziehung von Nicht-Null-Elementen der Beobachtung  $x'$  konstruiert. Somit werden die  $z'_i$  durch Ein- und Ausblenden verschiedener Superpixel dargestellt, wobei 1 das ursprüngliche und 0 ein ausgegrautes Superpixel angibt. Ferner ist es möglich, als Distanzfunktion für die Bildklassifikation die euklidische Distanz oder die Kosinus-Distanz zu verwenden.

### Beispiel 4.4.2

Für dieses Beispiel wird das vortrainierte Bilderkennungsmodell Inception Neural Network Modell von Google [SL15] mit 1000 verschiedenen Klassen als Zielvariable verwendet. Hier sagt das Klassifizierungsmodell „Baumfrosch“ mit  $p = 0.54$  als wahrscheinlichste Klasse voraus. Die interpretierbaren Stichproben  $z'_i$  werden mittels Ein- und Ausblenden der Superpixel, also  $N$  Variationen des Bildes  $x'$ , erstellt. Um einzelne Prognosen durch das Vorhersagemodell  $f$  zu erstellen, werden die Stichproben  $z'_i$  aus  $X'$  in den Originalzustand  $z_i$  aus  $X$  zurückgeführt. Die Umkehrfunktion  $IR^{-1}$  hängt hier vom Zustand der binären Vektoren ab. Ist ein Superpixel eingebendet, so werden alle Pixelwerte aus dem Originalbild beibehalten; falls nicht, wird das ausgeblendete bzw. ausgegraute Superpixel durch die jeweiligen segmentspezifischen Durchschnittsfarben ersetzt [SF20b]. In der Mitte der Abbildung 4.7 sind zwar durch Perturbed instances die Stichproben  $z'_i$  dargestellt, jedoch werden die  $z_i$  für das Black-Box-Modell  $f$  verwendet und die Wahrscheinlichkeiten  $P(\text{tree frog})$  dementsprechend berechnet. Es wird dann für jede Stichprobe die Wahrscheinlichkeit ausgegeben, dass sich ein Baumfrosch im Bild befindet. Anschließend wird ein lokal gewichtetes lineares Modell  $g$  trainiert und die Superpixel mit den höchsten positiven Gewichten, die für die Klassentscheidung „Baumfrosch“ relevant waren, als Erklärung ausgegeben. Durch das Ausblenden von Superpixeln, die das Gesicht des Frosches beinhalten, wurde anscheinend die Identifizierung des Bildes als Baumfrosch für das Modell erschwert. Folglich gibt die LIME-Erklärung an, dass ein Großteil der ursprünglichen Klassifizierungsentscheidung  $f$  auf dem Gesicht des Frosches basierte. Abbildung 4.7 zeigt, wie ein solches LIME-Verfahren beispielsweise aussehen kann.

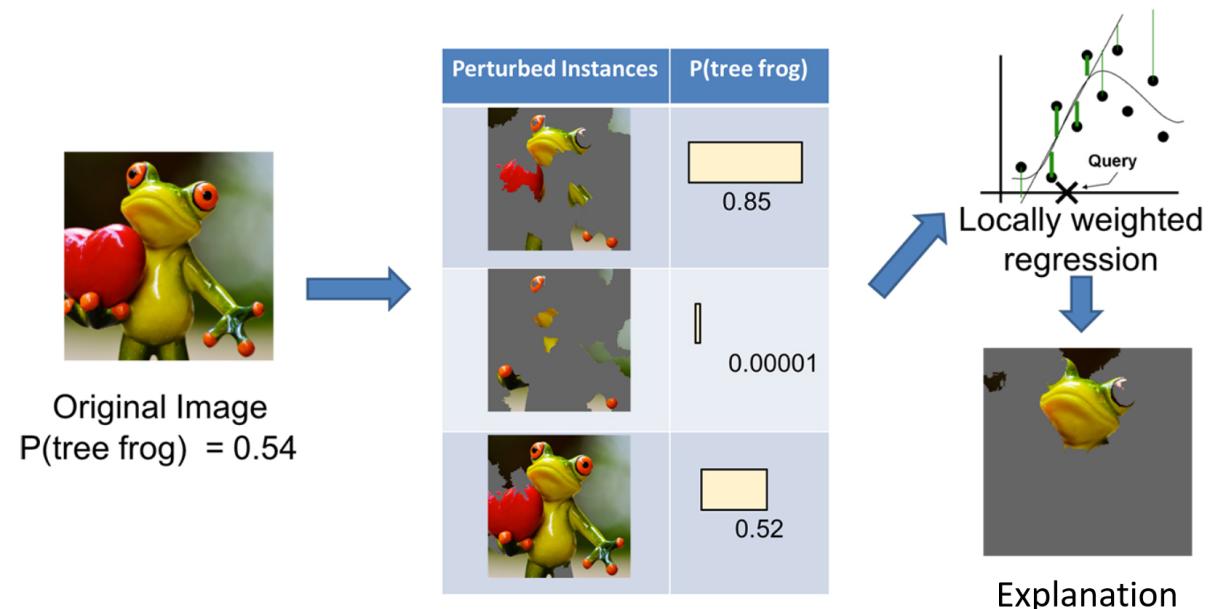


Abbildung 4.7: LIME-Erklärung zur Bildklassifizierung [RS16b]

Als nächster Schritt können optional die anderen, mit niedrigeren Wahrscheinlichkeiten vorhergesagten Klassen betrachtet werden. Wie in Abbildung 4.8 dargestellt, hat das Vorhersagemodell  $f$  das Bild  $x$  nicht nur mit  $p = 0.54$  als „Baumfrosch“, sondern auch mit  $p = 0.07$  als „Billardtisch“ und mit  $p = 0.05$  als „Heißluftballon“ klassifiziert. Die LIME-Erklärung zeigt an, dass die Klasse „Billardtisch“ vorhergesagt wurde aufgrund des grünen Hintergrunds und den Händen und Augen des Frosches, die Billardkugeln ähneln. Ebenso hat das Herz im Bild eine Ähnlichkeit mit einem roten Ballon, wodurch die Klasse „Heißluftballon“ mit einer Wahrscheinlichkeit ungleich Null vorhergesagt.

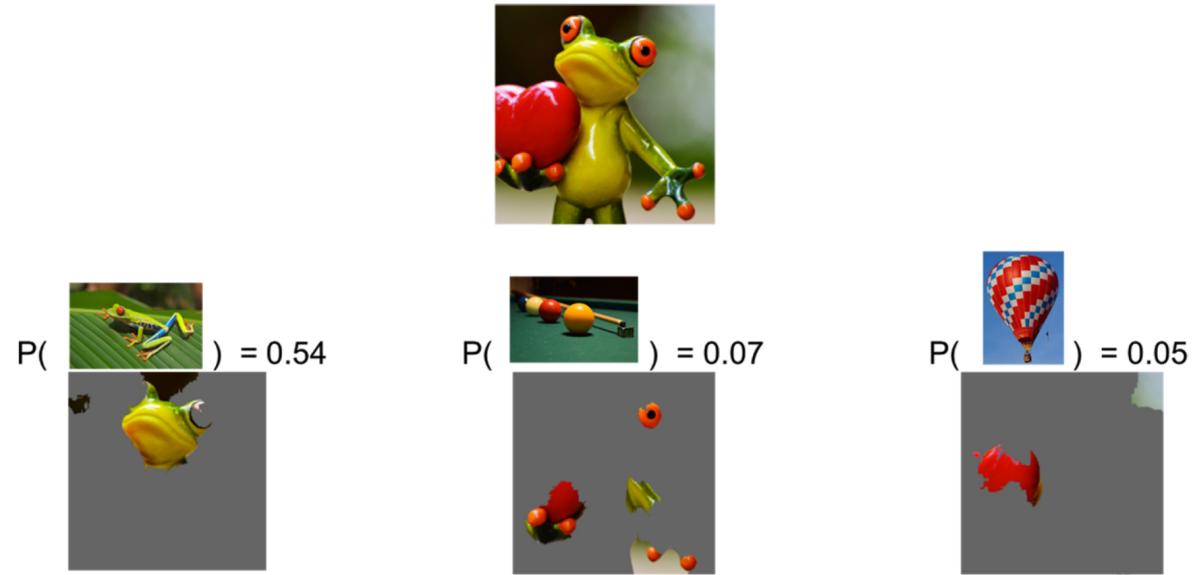


Abbildung 4.8: LIME-Erklärungen für die drei vorhergesagten Klassen zur Bildklassifizierung [RS16b]

#### Tabellendaten:

Im Gegensatz zu Text- und Bilddaten wird für tabellarische Daten eine andere Vorgehensweise eingeführt. Da die numerischen Merkmale erst diskretisiert und dann binärisiert wurden, kann aus ihnen – wie bereits in Abschnitt 4.2 erwähnt – nicht eindeutig ihre ursprüngliche Repräsentation  $X$  rekonstruiert werden. Um dies zu umgehen, werden für die Bestimmung der Vorhersage durch  $f$  die Stichproben im Originalzustand  $z_i$  und für die Berechnung der Distanz die interpretierbaren Stichproben  $z'_i$  verwendet. Somit findet keine Umkehrung von  $z'_i$  in  $z_i$  statt, da beide Darstellungen schon zur Verfügung stehen. Dementsprechend kann der Algorithmus 1 eingesetzt werden. Von nun an wird mit  $z_j$  und  $z'_j$  auf die Stichprobenwerte des Merkmals  $j$  für alle  $1 \leq j \leq d$  und mit  $\hat{z}$  auf den diskretisierten Zustand der Stichprobe hingewiesen [GL20].

Als erstes werden anhand einer festen Anzahl von Quantilsbereichen (Bins)  $p$  für jedes Merkmal  $j$  die empirischen Quantilen  $q_{j,0}, \dots, q_{j,p}$  berechnet. Die Standardeinstellung ist die Diskretisierung auf Grundlage der Quartile mit  $p = 4$ . Infolgedessen gibt es entlang jeder Dimension eine Abbildung  $\phi_j : \mathbb{R} \rightarrow \{1, \dots, p\} \forall j \in \{1, \dots, d\}$ , indem jede reelle Zahl, also der Wert des Merkmals  $j$ , den Index des Quantils zugeordnet bekommt, zu dem sie gehört. Dabei gilt, wenn  $x_j$  auf der Intervallgrenze der Quantilen  $q_k$  und  $q_{k+1}$  liegt, erhält es den Wert  $k$ . Um die endgültigen Stichprobendaten  $z_i$  für  $i \in \{1, \dots, N\}$  herzustellen, wird als nächstes die Umkehrung der Diskretisierung eingeführt. Hierfür findet zunächst die Stichprobenerhebung für jedes Merkmal  $j \in \{1, \dots, d\}$   $d$ -mal innerhalb der diskretisierten Darstellung  $\{1, \dots, p\}$  durch gleichverteilte zufällige Ziehung statt [GL20].

### Beispiel 4.4.3

Das Beispiel für Tabellendaten aus Abschnitt 4.2 wird hier weitergeführt. Beispielsweise sehen die Stichproben für die Merkmale  $j \in \{1, \dots, d\}$  wie folgt aus:  $\hat{z}_1 = (1, 2, \dots, 3)^T, \dots, \hat{z}_d = (2, 3, \dots, 1)^T$ . Die transponierten Tupel entsprechen den Spalten in der Abbildung 4.9. Ausgeschrieben bedeutet dies z.B. für das letzte Element von  $\hat{z}_1$ , dass die Stichprobe  $\hat{z}_N$  bei dem Merkmal  $X_1 = \text{"Kontostand"}$  zufällig eine 1 erhalten hat und somit in den ersten Quantilsbereich  $q_1$  mit  $(-\infty, 500]$  angehört.

	$X_1 = \text{Kontostand}$	...	$X_d = \text{Blutgruppe}$
$x$	2		1
$z_1$	1		2
$z_2$	2		3
:			
$z_N$	3		1

$\hat{z}_j \quad \forall j \in \{1, \dots, d\}$

$\hat{z}_i \quad \forall i \in \{1, \dots, N\}$

Abbildung 4.9: Stichprobenerhebung in diskretisierter Darstellung

Anschließend wird mit  $\psi : \{1, \dots, p\} \rightarrow \mathbb{R}$  die Umkehrung der Diskretisierung der Stichproben eingeleitet. Hierfür wird unter Verwendung von Mittelwerten und Standardabweichungen der Trainingsdaten eine Normalverteilung angewendet, die sich nur auf die entsprechenden Quantilsbereiche beschränkt. Somit wird mit  $\psi_j(k)$  für eine Koordinate  $j$  und einen Bin-Index  $k$  eine einzelne Stichprobe aus der an den Bin-Grenzen abgeschnittenen Normalverteilung entnommen [GL20].

### Beispiel 4.4.4

Wird nun  $\psi_j(k) \quad \forall j \in \{1, \dots, d\}$ , angewendet, so wird beispielsweise aus der diskreten Stichprobe  $\hat{z}_1 = (1, 2, \dots, 3)^T$  die numerische Stichprobe  $z_1 = (200, 945, \dots, 1552)^T$ . Somit stammt z.B. die 200 aus dem Quantilsbereich  $q_1$  mit  $(-\infty, 500]$ . Anschließend werden die numerischen Stichproben  $z_1, \dots, z_N \in \mathbb{R}^d$  in das Black-Box-Modell  $f$  eingesetzt (siehe Abbildung 4.10).

	$X_1 = \text{Kontostand}$	...	$X_d = \text{Blutgruppe}$		$f() = \text{Klasse } 1$	$f() = \text{Klasse } 0$
$x$	850		A		0	1
$z_1$	200		B		0.6	0.4
$z_2$	945		AB		0.2	0.8
:	:				:	:
$z_N$	1552		A		0.3	0.7

Abbildung 4.10: Ermittlung der Klassenzugehörigkeit mittels originaler Darstellung der Stichproben

Als letzter Schritt werden für die Berechnung der Distanz die numerischen Stichproben  $z_i$  mit  $i \in \{1, \dots, N\}$  in die interpretierbare Darstellung  $z'_i$  umgewandelt. Für alle  $i \in \{1, \dots, N\}$  gilt  $z'_j = 1_{\phi_j(z)=\phi_j(x)}$  für alle  $1 \leq j \leq d$ . Dies weist darauf hin: Falls die diskrete Stichprobe  $\phi_j(z) = \hat{z}_j$  in denselben Quantilsbereich wie  $x$  fällt, dann wird es mit einer 1, sonst mit einer 0 ausgedrückt. Für die Tabellendaten wird die euklidische Distanz als Distanzfunktion  $D$  verwendet [GL20].

### Beispiel 4.4.5

Für die Berechnung der Distanz werden die in der Abbildung 4.9 vorgestellte Instanz und die Stichproben aus der diskretisierten Form  $\phi_j(x) = (2, \dots, 1)$  und  $\hat{z}_1 = (1, \dots, 2), \hat{z}_2 = (2, \dots, 3), \dots, \hat{z}_N = (3, \dots, 1)$  in die unten abgebildete 4.11 binärisierte Anordnung  $z'_i$  umgewandelt.

The diagram illustrates the transformation of data from a continuous form to a binary form for distance calculation. On the left, a table shows the original data points  $x'$ ,  $z'_1$ ,  $z'_2$ ,  $\vdots$ , and  $z'_N$  with columns for Kontostand ( $X_1$ ) and Blutgruppe ( $X_d'$ ). A red arrow points to the right, where another table shows the transformed data  $z'_i$  with columns for Kontostand ( $X_1$ ) and Blutgruppe ( $X_d'$ ). The right table also includes a header row for the Euclidean distance  $D$ .

	$X_1 = \text{Kontostand}$	...	$X_{d'} = \text{Blutgruppe}$
$x'$	1		1
$z'_1$	0		0
$z'_2$	1		0
$\vdots$			
$z'_N$	0		1

	$D = \text{Euklidische Distanz}$
$x'$	$\sqrt{(1-1)^2 + \dots + (1-1)^2}$
$z'_1$	$\sqrt{(0-1)^2 + \dots + (0-1)^2}$
$z'_2$	$\sqrt{(1-1)^2 + \dots + (0-1)^2}$
$\vdots$	$\vdots$
$z'_N$	$\sqrt{(0-1)^2 + \dots + (1-1)^2}$

Abbildung 4.11: Berechnung der Distanz anhand der interpretierbaren Stichproben

Um die Vorgehensweise der Stichprobenerhebung für Tabellendaten besser zu verstehen, folgt nun eine visuelle Darstellung des LIME-Algorithmus. Erwähnenswert ist noch, dass bei der Python-Implementierung standardmäßig `sample_around_instance` auf `false` gesetzt. Das bedeutet, dass die Parameter für die Normalverteilung, Mittelwert und Standardabweichung auf den Merkmalsdaten basieren. Falls aber `sample_around_instance` als `true` festgelegt werden sollte, dann werden die Stichproben einer abgeschnittenen Normalverteilung entnommen, die auf die zu erklärende Instanz zentriert ist.

### Beispiel 4.4.6

Das folgende Beispiel stammt von Christoph Molnar aus [MC20]. Die Entscheidungsregionen für ein komplexes binäres Klassifikationsmodell werden durch den grauen und blauen Hintergrund dargestellt. Diese Bereiche sind zwei Klassen, die durch das Black-Box-Modell  $f$  vorhergesagt werden. Der große gelbe Punkt ist die zu erklärende Instanz  $x$ , die kleinen schwarzen Punkte sind die entnommenen Stichproben. Anhand der Abbildung 4.12 links wird deutlich, dass in diesem Beispiel gegen eine Stichprobenerhebung um die ausgewählte Instanz herum entschieden wurde, also `sample_around_instance = false`. Diese Stichproben werden entsprechend ihrer Nähe zu  $x$  gewichtet und das Gewicht wird hier durch die Größe der schwarzen Punkte dargestellt, wobei gilt, je kleiner die euklidische Distanz, desto höher wird das Ähnlichkeitsmaß. Wie rechts in Abbildung 4.12 dargestellt, wird anschließend für jede Stichprobe die Vorhersage anhand des komplexen Modells  $f$  bestimmt und zuletzt ein lokales lineares Modell  $g$  basierend auf gewichteten Stichproben erlernt. Die weiße Linie stellt die Entscheidungsfunktion des einfachen linearen Modells dar, und nun kann die Vorhersage des komplexen Modells für die ausgewählte Instanz erklärt werden.

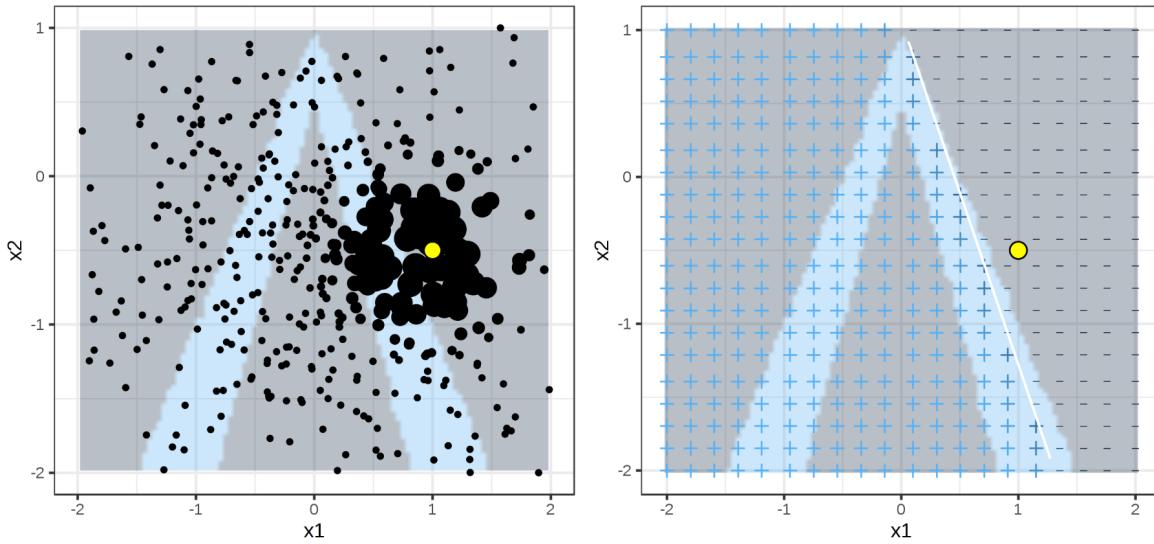


Abbildung 4.12: LIME-Algorithmus für tabellarische Daten [MC20]

#### Beispiel 4.4.7

Nun wird ein konkretes Beispiel für die Erstellung einer LIME-Erklärung für Tabellendaten vorgestellt. Die Python-Implementierung von Riberio et al. kann in [RMGH] nachgeschlagen werden. Für dieses Beispiel wird der Adult-Datensatz aus dem UCI Machine-Learning-Repository [KB96] verwendet, wobei dieser Datensatz sowohl numerische als auch kategoriale Merkmale umfasst. Der Datensatz umfasst 14 Merkmale, von denen 6 kontinuierlich und 8 kategorial sind, und eine Zielvariable mit zwei Klassen. Die numerischen Eingabedaten sind beispielsweise „Age“, „Capital Gain“ und „Hours per week“, die kategorialen sind unter anderem „Race“, „Sex“ und „Marital Status“. Ziel ist es vorherzusagen, ob eine Person mehr als 50.000 Dollar pro Jahr verdient. Zuerst wird ein XGBoost-Modell als  $f$  trainiert und im Anschluss die LIME-Erklärung für eine Instanz, hier eine Person  $x$ , erstellt. Insgesamt hat das Modell entschieden, dass es sehr wahrscheinlich ist, dass diese Person über 50.000 Dollar pro Jahr verdient. Abbildung 4.13 zeigt an, dass die Modellentscheidung  $f$  vor allem auf Merkmale wie Kapitalgewinn (Capital Gain) und Familienstand (Marital Status) zurückzuführen ist. Hier wird also die Frage beantwortet, ob für die vorhergesagte Klassifikation z.B. der Kapitalgewinn über Null ( $\text{Capital Gain} > 0$ ) verglichen mit einem Kapitalgewinn außerhalb dieses Bereichs einen positiven oder negativen Effekt hat [RMGH].

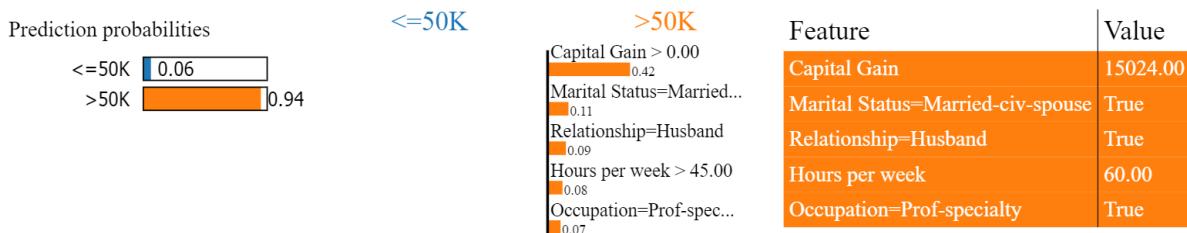


Abbildung 4.13: LIME-Erklärung für Tabellendaten [RMGH]

## 4.5 Submodular Pick

Zwar vermittelt die Erklärung einer einzelnen Vorhersage dem Benutzer ein gewisses Verständnis für die Zuverlässigkeit des Vorhersagemodells. Dies allein reicht jedoch nicht aus, um das Vertrauen in ein Modell als Ganzes zu bewerten. Mit dem Ziel, einen allgemeinen Überblick über das Verhalten eines Modells zu erhalten, stellen die Autoren Ribeiro et al. eine weitere Technik vor. Submodular Pick (SP-LIME) ist eine Methode, die eine Menge repräsentativer Instanzen mit ihren entsprechenden lokalen LIME-Erklärungen auswählt. Somit kann das Modell durch Aggregation der lokalen Erklärungen global erklärt werden. Hier soll eine Menge von nicht-redundanten Erklärungen ausgewählt werden, das heißt, die Auswahl von Instanzen mit ähnlichen Erklärungen muss vermieden werden.

Der Benutzer hat möglicherweise nicht die Zeit bzw. Geduld, bei einer großen Anzahl von Erklärungen jede einzelne durchzugehen und zu überprüfen. Daher wird ein Budget  $B$  als die Anzahl der Erklärungen definiert, die der Benutzer zu überprüfen bereit ist. Bei einer Menge von  $X$ -Instanzen ist der Auswahlschritt die Aufgabe, ein Maximum von  $B$ -Instanzen auszuwählen, die der Benutzer untersuchen soll. Um mit der Konstruktion einer Erklärungsmatrix  $\mathcal{W}$  beginnen zu können, muss zunächst die Anzahl der zu erklärenden Instanzen mit  $n = |X|$  festgelegt werden. Entweder werden Erklärungen für die gesamten Daten generiert oder es wird mit `sample_size` ein bestimmter Prozentsatz für die Wahl der gleichverteilten Stichproben aus dem Datensatz entschieden. Standardmäßig wird bei der Auswahl der Menge von Instanzen  $X$  für das Letztere entschieden.

Jede Zeile in der Matrix  $\mathcal{W}$  der Größe  $n \times d'$  repräsentiert die lokale Bedeutung von Merkmalen einer bestimmten Instanz, wobei  $d'$  die Anzahl der Merkmale ist, mit denen Instanzen in  $X$  erklärt werden. Im Falle der Verwendung linearer Modelle als Erklärungen wird für eine Instanz  $x^{(i)}$  und eine Erklärung  $g_i = \xi(x^{(i)})$  die Erklärungsmatrix mit  $\mathcal{W}_{ij} = |w_{g_{ij}}|$  angegeben. Ferner wird für jedes Merkmal  $j$  in  $\mathcal{W}$ , also die Spalten von  $\mathcal{W}$ , anhand  $I_j$  die globale Wichtigkeit dieser Merkmale im Erklärungsraum beschrieben. Hier wird die globale Wichtigkeit der Merkmale durch  $I_j = \sqrt{\sum_{i=1}^n |\mathcal{W}_{ij}|}$  definiert, wobei Merkmalen, die viele verschiedene Instanzen erklären, höhere Wichtigkeitswerte zugewiesen werden. Da bei der Auswahl Instanzen mit ähnlichen Erklärungen vermieden werden sollen, wird die Anforderung einer nicht-redundanten Abdeckung mit Gleichung 4.5 definiert. Die Abdeckung wird als die Mengenfunktion  $c$  beschrieben, die bei gegebenen  $\mathcal{W}$  und  $I$  die gesamte Wichtigkeit der Merkmale berechnet, die zumindest in einer Instanz in einer Menge  $V$  auftreten.

$$c(V, \mathcal{W}, I) = \sum_{j=1}^{d'} \mathbf{1}_{[\exists i \in V : \mathcal{W}_{ij} > 0]} I_j \quad (4.5)$$

Das Hauptziel des Auswahlschritts in Gleichung 4.6 besteht darin, die Menge  $V, |V| \leq B$  zu finden, die die höchste Abdeckung erzielt.

$$\text{Pick}(\mathcal{W}, I) = \underset{V, |V| \leq B}{\operatorname{argmax}} c(V, \mathcal{W}, I) \quad (4.6)$$

Ribeiro hat in seinem Blog [RMBG] gezeigt, dass es sich bei  $c$  um eine monotone, submodulare Funktion handelt. Daher kann auch diese Aufgabe, also der Auswahlschritt in 4.6, in dem eine gewichtete Abdeckungsfunktion maximiert wird, als submodulares Optimierungsproblem angesehen werden. Aufgrund der Submodularität wird in der Praxis ein Greedy-Algorithmus

verwendet. Zu diesem Zweck wird  $c(V \cup \{i\}, \mathcal{W}, I) - c(V, \mathcal{W}, I)$  als marginaler Abdeckungsgewinn beim Hinzufügen einer Instanz  $i$  zu einer Menge  $V$  definiert. Um nun die Menge  $V$  mit der höchsten nicht-redundanten Abdeckung zu finden, wird iterativ die Instanz mit dem höchsten marginalen Abdeckungsgewinn zur Menge  $V$  hinzugefügt. Dieser Annäherungsprozess wird in Algorithmus 2 beschrieben und als submodulare Auswahl (Submodular Pick) bezeichnet.

---

**Algorithm 2** Submodular Pick (SP)

---

**Require:** Instanzen  $X$ , Budget  $B$

```

1: for all  $x^{(i)} \in X$  do
2:    $\mathcal{W}_i \leftarrow \text{explain}(x^{(i)}, x'^{(i)})$        $\triangleright$  Verwende Algorithmus 1
3: end for
4: for  $j \in \{1, \dots, d'\}$  do
5:    $I_j \leftarrow \sqrt{\sum_{i=1}^n |\mathcal{W}_{ij}|}$            $\triangleright$  Berechne Wichtigkeit der Merkmale
6: end for
7:  $V \leftarrow \{\}$ 
8: while  $|V| < B$  do                   $\triangleright$  Greedy Optimierung der Gleichung 4.6
9:    $V \leftarrow V \cup \text{argmax}_i c(V \cup \{i\}, \mathcal{W}, I)$ 
10: end while
11: return  $V$ 

```

---

Es sollte noch erwähnt werden, dass die SP-LIME-Methode zurzeit nur auf Text- und Tabellen-daten angewendet werden kann. Eine sinnvolle Definition für  $I_j$  für Bilder wurde bisher nicht aufgestellt, da hier etwas gemessen werden muss, beispielsweise Farbhistogramme oder andere Merkmale von Superpixeln, das zwischen den Superpixelen in verschiedenen Bildern  $x^{(i)}$  vergleichbar ist.

### Beispiel 4.5.1

Die Vorgehensweise von SP-LIME wird anhand eines leicht verständlichen Beispiels vorgestellt. Es handelt sich um eine Textklassifikation, wobei die Zeilen die Instanzen  $x^{(i)}$  (Dokumente) und die Spalten die Merkmale  $x_j^{(i)}$  (Wörter) repräsentieren. Abbildung 4.14 zeigt, wie die Erklärungsmatrix  $W$  mit  $n = d' = 5$  in vereinfachter Form durch binäre Gewichte für die Merkmale dargestellt wird. Zunächst wird die globale Wichtigkeit  $I_j$  für die Merkmale  $j \in \{1, \dots, 5\}$  berechnet:

$$\begin{aligned}
I_1 &= \sqrt{\sum_{i=1}^5 \mathcal{W}_{i1}} = \sqrt{1+0+0+0+0} = 1, \quad I_2 = \sqrt{\sum_{i=1}^5 \mathcal{W}_{i2}} = \sqrt{1+1+1+1+0} = 2, \\
I_3 &= \sqrt{\sum_{i=1}^5 \mathcal{W}_{i3}} = \sqrt{0+1+1+0+0} \approx 1,41, \quad I_4 = \sqrt{\sum_{i=1}^5 \mathcal{W}_{i4}} = \sqrt{0+0+0+1+1} \approx 1,41, \\
I_5 &= \sqrt{\sum_{i=1}^5 \mathcal{W}_{i5}} = \sqrt{0+0+0+0+1} = 1.
\end{aligned}$$

Es ist deutlich, dass  $f2$  das wichtigste Merkmal mit einer globalen Wichtigkeit von  $I_2 = 2$  (die blau gestrichelte Linie) ist. Verglichen mit anderen Merkmalen, wie z.B.  $f1$  (nur in  $x^{(1)}$  vertreten), wurde das Merkmal  $f2$  in den meisten Erklärungen ( $x^{(1)}, x^{(2)}, x^{(3)}$  und  $x^{(4)}$ ) verwendet. Infolgedessen hat  $I_j$  für das Merkmal  $f2$  einen höheren Wert als für  $f1$ , das heißt  $I_2 > I_1$ , und Merkmal  $f2$  wird verwendet, um mehr Instanzen zu erklären. Schließlich wird das Auswahlverfahren mit dem vom Benutzer angegebenen Budget  $B = 2$  gestartet. Das Ziel, repräsentative und nicht-redundante Instanzen zu finden, wurde durch die Auswahl von Zeile 2 und 5 (in rot), also

die Instanzen  $x^{(2)}$  und  $x^{(5)}$ , erfüllt. Beispielsweise fügt die dritte Zeile nach Auswahl der zweiten Zeile keinen Wert hinzu und wäre daher redundant. Zudem decken die beiden ausgewählten Instanzen alle Merkmale außer  $f_1$  ab.

	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$
1	■	■	■	■	■
2	■	■	■	■	■
3	■	■	■	■	■
4	■	■	■	■	■
5	■	■	■	■	■

Abbildung 4.14: Musterbeispiel zum Submodular-Pick-Algorithmus

## 4.6 Stärken und Schwächen des LIME-Algorithmus

Abschließend werden die Stärken und die potenziellen Limitationen der LIME-Methode benannt. Ein großer Vorteil ist, dass LIME modellagnostisch ist und somit auf jedes Black-Box-Modell angewendet werden kann. Darüber hinaus ist LIME in Python und R implementiert und generiert Erklärungen für die Vorhersage von Text-, Bild- und Tabellendaten [MC20]. Zudem ist LIME eine beliebte Technik, die aufgrund ihrer Einfachheit bereits in vielen Anwendungen eingesetzt wurde. Die Anwendbarkeit von LIME konnte z.B. auf Music Content Analysis durch SLIME [MS17] erweitert werden, auf Survival Analysis durch SurvLIME [KU20], auf Bayesian Predictive Models durch KL-LIME [PT18] oder auf Graph Neural Networks durch GraphLIME [HY20]. Auch im Bereich des maschinellen Lernens, beispielsweise für induktive logische Programmierung, wurden unterschiedliche Versionen von LIME vorgestellt [[SG18], [SF19], [RD20]]. Außerdem haben die Autoren Ribeiro et al. neben dem originalen LIME auch die Anchors-Methode entwickelt. Sie basiert auf LIME und wurde um Entscheidungsregeln für präzisere Erklärungen erweitert [RS18].

Es gibt einige weitere Gründe, die zur Modifikation der aktuellen Implementierung von LIME geführt haben. Bereits Ribeiro et al. selbst wiesen in [RS16a] insbesondere auf Folgendes hin: Falls das zugrunde liegende Modell  $f$  in der Lokalität der Vorhersage stark nichtlinear ist, wird die Wahl von  $g$  als linearen Modell möglicherweise keine zuverlässige Leistung bezüglich plausibler Erklärungen erbringen. Im Zuge dessen wurde von unterschiedlichen Forschern ein entscheidungsbaumbasierter LIME-Ansatz implementiert, wobei es unterschiedliche Varianten von Entscheidungsbäumen als interpretierbares Modell  $g$  verwendet wurden [[LF19], [SZ19], [SF20b]]. Mit dem Ziel, das globale Verhalten des Modells zuverlässig wiederzugeben, wurden auch in Hinblick auf das von SP-LIME konstruierte globale Verständnis verschiedene Optionen für die Aggregation der lokalen Erklärungen vorgeschlagen [[ES19], [AN19], [VH19]].

Die größte Schwäche der LIME-Algorithmus ist aber laut [MC20], eine angemessene Definition der lokalen Region, insbesondere bei tabellarischen Daten, zu finden. Hierzu verwenden die aktuellen Python- und R-Implementierungen für das Ähnlichkeitsmaß den exponentiellen Kernel  $\pi_x(z) = \sqrt{\exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)}$  mit dem Hyperparameter  $\sigma$ , der die Kernel-Breite darstellt. Dabei wurde standardmäßig für Textdaten  $\sigma = 25$ , für Bilddaten  $\sigma = 0,25$  und für Tabellendaten  $\sigma = 0,75 \cdot \sqrt{K}$  mit  $K$  als die Anzahl der Merkmale bzw. Spalten definiert. Weder die Auswahl von  $\pi$  noch von  $\sigma$  wurde von Ribeiro et al. begründet. Es wurde bereits in [LR18] gezeigt, dass

der Wert von  $\sigma$  die Genauigkeit bzw. Treue der generierten Erklärung beeinflusst. Um die richtige Kernel-Breite zu finden, wird angeraten, verschiedene Kernel-Einstellungen auszuprobieren und die erzeugten Erklärungen danach zu bewerten, ob sie sinnvoll erscheinen oder nicht.

### Beispiel 4.6.1

Ein Beispiel für die Suche nach einer passenden Kernel-Breite wurde in [BC20] visualisiert und hier auf Deutsch übersetzt. In Abbildung 4.15 wurden die Stichproben durch rote Dreiecke in negative Klassen und durch gelbe Punkte in positive Klassen eingeteilt. Wird die Kernel-Breite  $\sigma$  zu klein ausgewählt, so wird nur wenigen Stichproben ein hohes Gewicht zugewiesen. Im Gegensatz dazu werden bei der Auswahl von zu großem  $\sigma$  alle Stichproben gleich gewichtet. Entweder wird also eine kleine Reichweite erreicht oder es wird schwierig ein lineares Modell abzuleiten. Wird aber ein optimales  $\sigma$  gefunden, so kann die lokale Treue des linearen Modells gewährleistet werden.

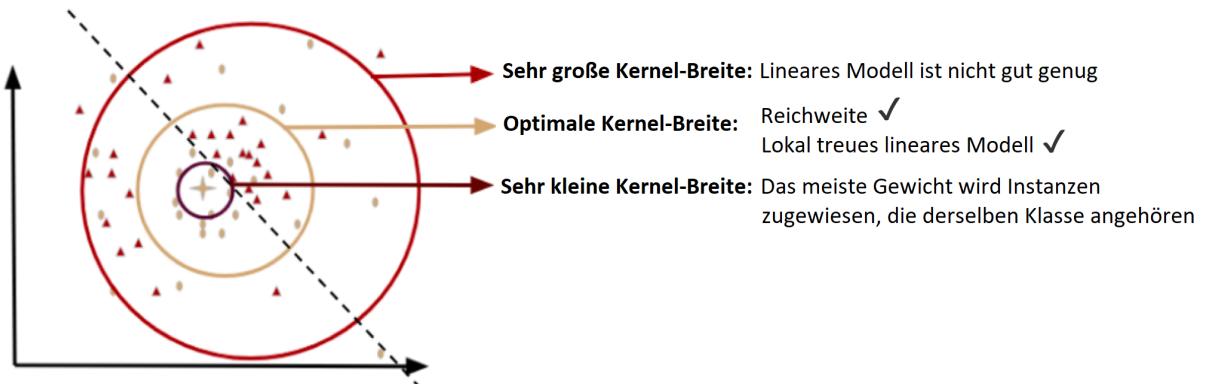


Abbildung 4.15: Kernel-Breiten im Vergleich [BC20]

Ein weiterer Nachteil bei der Verwendung von LIME besteht darin, dass die Instabilität der generierten Erklärungen erwiesen ist. Das heißt, die LIME-Erklärung für zwei sehr nahe beieinander liegende Datenpunkte, also ähnliche Instanzen, kann unterschiedlich ausfallen. Sogar die wiederholte Anwendung der LIME-Methode für eine bestimmte Instanz kann unter den gleichen Bedingungen zu unterschiedlichen Erklärungen führen [[AJ18], [ZS19]]. Hier ist das Erzeugen von permutierten Stichproben der Grund, warum die Technik unter mangelnder Stabilität leidet, da für dieselbe Instanz unterschiedliche Merkmale ausgewählt und gewichtet werden und mit ihnen die Entscheidung der Black-Box-Modell erklärt wird [MC20]. Wie bereits in 4.4 erläutert, erfolgt die Stichprobenerfassung für Text- und Bilddaten durch gleichverteilte zufällige Ziehung und für Tabellendaten entweder unter Verwendung der Normalverteilung oder der Verteilung von Trainingsdaten. Dabei kann die Vernachlässigung der Interaktion zwischen den Merkmalen zu unrealistischen Stichproben führen, wobei diese Datenpunkte dann für die Erstellung der Erklärungsmodell  $g$  verwendet werden [MC20]. Auch hier wurden zur Bewältigung der genannten Schwächen der LIME-Methode verschiedene Ansätze zur Datengenerierung vorgeschlagen. Diese modifizierten LIME-Methoden zielen darauf ab, die Instabilität in der erzeugten Erklärung, die durch den zufällig erzeugten Datensatz verursacht wird, durch eine sinnvolle Stichprobenerhebung zu beheben [[RY19], [ZK19], [SZ20], [SD20]].

Ferner ist zu erwähnen, dass die von LIME erzeugte Ausgabe für Bilddaten zwar den positiven Einfluss (grüne Farbe) und den negativen Einfluss (rote Farbe) des Superpixels auf die Klassifizierung einer bestimmten Instanz hervorhebt, es gibt aber nicht die zugewiesenen Gewichte für diese Regionen aus [SH18].

## 5 Shapley Additive Explanations

---

Das Ziel dieses Kapitels ist es, die SHAP-Methode (ein Akronym für Shapley Additive Explanations) herzuleiten und ausführlich zu erläutern. In Abschnitt 5.1 erfolgt ein grober Überblick der SHAP-Methode und der Verlauf des Kapitels wird konkretisiert. Um das Konzept der SHAP-Methode vorzustellen, werden zunächst die Shapley-Werte in Abschnitt 5.2 eingeführt. Anschließend wird in Abschnitt 5.3 die SHAP-Methode detailliert vorgestellt. In Abschnitt 5.4 liegt der Fokus auf Kernel SHAP, einem modellagnostischen Ansatz zur SHAP-Methode. Schließlich werden die Methoden LIME und Kernel SHAP in Abschnitt 5.5 verglichen und dabei ihre Unterschiede hervorgehoben.

### 5.1 Einführung

Shapley Additive Explanations (SHAP) ist ein lokales Erklärungsmodell, das von Scott M. Lundberg und Su-In Lee zum ersten Mal 2016 vorgestellt wurde [LL16]. Die SHAP-Methode basiert auf den Shapley-Werten [SL53], einem Konzept aus der Spieltheorie, das von Lloyd S. Shapley zur Berechnung des Beitrags jedes Spielers in einem kooperativen Spiel eingeführt wurde. Die Idee, die Shapley-Werte für die Interpretation von Black-Box-Modellen zu verwenden, wurde zuerst von Štrumbelj und Kononenko [SK10] vorgeschlagen. Die Erklärungsmethode von Lundberg et al. ermittelt, wie stark unterschiedliche Eingabemerkmale die Ausgabe eines beliebigen Machine-Learning-Modells beeinflussen, und gehört daher zur Kategorie der Merkmalsrelevanz. Die Vorhersage für eine beliebige Instanz  $x$  wird dabei als die Summe der Beiträge der einzelnen Eingabemerkmale erklärt, wobei diese Beiträge auf Grundlage der Merkmalskombinationen berechnet werden. Aufgrund der hohen Anzahl möglicher Kombinationen von Merkmalen stellten Lundberg et al. eine Schätzung des Shapley-Wertes vor. Zu diesem Zweck wurde ein einheitlicher Ansatz für die Generierung von Erklärungen in Form von additiven Merkmalszuordnungen vorgeschlagen, der sieben modellspezifische und modellagnostische Erklärungsmodelle vereinigt. Dazu gehört die in Kapitel 4 vorgestellte LIME-Methode. Die Autoren kombinieren die LIME-Implementierung von Ribeiro et al. mit Shapley-Werten und nennen es Kernel SHAP. Dabei wird ein spezieller Kernel für die Gewichtung des lokalen linearen Ersatzmodells definiert, anhand dessen dann die Koeffizienten des LIME-Modells die SHAP-Werte approximieren.

### 5.2 Shapley-Wert

Der Shapley-Wert gehört zu den wichtigsten Lösungskonzepten in der kooperativen Spieltheorie und wurde im Jahre 1953 von Lloyd S. Shapley [SL53] formuliert. Die kooperative Spieltheorie ist ein Teilgebiet der Spieltheorie und befasst sich mit Koalitionen von Spielern und der dazugehörigen Bestimmung des Auszahlungsvektors durch eine Koalitionsfunktion [WH05]. Die Ausführungen in diesem Abschnitt stammen von Lloyd S. Shapley [SL53] und Štrumbelj und Kononenko [SK10] und beginnen mit der Definition eines Koalitionsspiels.

**Definition 5.2.1** Ein Koalitionsspiel ist ein Tupel  $\langle N, v \rangle$  bestehend aus einer endlichen Menge  $N = \{1, 2, \dots, n\}$  von  $n$  Spielern und  $v : 2^N \rightarrow \mathbb{R}$  eine charakteristische Funktion mit  $v(\emptyset) = 0$  ist.

Die Teilmengen von  $N$  sind Koalitionen und  $N$  wird als große Koalition bezeichnet. Dabei wird die charakteristische Funktion auch als Koalitionsfunktion bezeichnet, wobei  $v(S)$  der Wert der Koalition  $S \subset N$  ist. Somit wird der Wert einer Koalition von Spielern als die Auszahlung betrachtet, die durch die Bildung dieser Koalition erzeugt wird. Für jedes Koalitionsspiel findet noch eine Aufteilung der Auszahlungen unter den Spielern statt. Mit dem Ziel, eine „faire“ Verteilung der Gesamtauszahlung  $v(N)$  zu finden, werden die Axiome der Effizienz, der Symmetrie, des Nullspielers und der Additivität definiert. Dabei ist der Shapley-Wert die einzige Auszahlungsfunktion, die genau diese Axiome erfüllt:

**Definition 5.2.2** Für das Spiel  $\langle N, v \rangle$  gibt es eine eindeutige Lösung  $\phi$ , die die Axiome der Effizienz, Symmetrie, Nullspieler und Additivität erfüllt und als Shapley-Wert definiert wird:

$$\begin{aligned}\phi_j(v) &= \frac{1}{N} \sum_{S \subseteq N \setminus \{j\}} \binom{N-1}{|S|}^{-1} (v(S \cup \{j\}) - v(S)) \\ &= \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!(N-|S|-1)!}{N!} (v(S \cup \{j\}) - v(S))\end{aligned}\tag{5.1}$$

Bei Gleichung 5.1 für den Shapley-Wert wird durch  $v(S \cup \{j\}) - v(S)$  der marginale Beitrag des Spielers  $j$  zur Koalition  $S$  berechnet. Zusätzlich wird die Wahrscheinlichkeit für die Zusammenkunft einer Koalition bestimmt und so die marginalen Beiträge gewichtet. Der Shapley-Wert des Spiels wird in Form eines Auszahlungsvektors  $\phi(v) = (\phi_1(v), \phi_2(v), \dots, \phi_N(v))$  angegeben, wobei die Erfüllung der folgende Axiome durch den Shapley-Wert zu einer fairen Auszahlung führt.

#### Effizienz:

Die Summe der Shapley-Werte aller Spieler entspricht dem Wert der großen Koalition:

$$\sum_{j \in N} \phi_j(v) = v(N).$$

#### Symmetrie:

Wenn für zwei Spieler  $j$  und  $k$  die  $v(S \cup \{j\}) = v(S \cup \{k\})$  für jedes  $S$  gilt, wobei  $S \subset N$  und  $j, k \notin S$ , dann ist  $\phi_j(v) = \phi_k(v)$ .

#### Nullspieler:

Wenn  $v(S \cup \{j\}) = v(S)$  für jedes  $S$  gilt, wobei  $S \subset N$  und  $j \notin S$ , dann leistet der Spieler  $j$  keinen Beitrag zu den Koalitionen und erhält den Shapley-Wert Null  $\phi_j(v) = 0$ .

#### Additivität:

Für die Kombination von zwei Koalitionsspielen  $v$  und  $w$  gilt:  $\phi_j(v + w) = \phi_j(v) + \phi_j(w)$ , wobei die Summe von  $v$  und  $w$  definiert ist durch:  $(v + w)(S) = v(S) + w(S)$ .

**Beispiel 5.2.1** Anhand eines einfachen Beispiels wird ein Koalitionsspiel und das Lösungskonzept der Shapley-Werte illustriert. Hierfür wird das Handschuh-Spiel aus [WH05] für  $N = \{1, 2, 3\}$  verwendet, bei dem die Spieler entweder linke oder rechte Handschuhe besitzen. In diesem Fall besitzen die Spieler 1 und 2 einen linken Handschuh und der Spieler 3 einen rechten Handschuh. Mit dem Ziel, Handschuhpaare zu bilden, wird die Koalitionsfunktion wie folgt definiert:

$$v(S) = \begin{cases} 1 & \text{falls } S \in \{\{1, 3\}, \{2, 3\}, \{1, 2, 3\}\} \\ 0 & \text{sonst} \end{cases}$$

Die Menge aller Koalitionen wird durch  $|2^N| = 2^{|N|}$  mit  $N = \{1, 2, 3\}$ ,  $|N| = 3$  bestimmt. Es ist möglich  $2^3 = 8$  Koalitionen durch  $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}$  und  $\{1, 2, 3\}$  zu bilden. Mit  $N! = 3! = 6$  wird für die Berechnung der Shapley-Werte die unten abgebildete Tabellen für die Spieler 1, 2 und 3 ausgefüllt.

$S \subseteq N \setminus \{j\}$	$ S !(N -  S  - 1)!$	$v(S \cup \{j\}) - v(S)$
$\emptyset$	$0! \cdot (3 - 0 - 1)! = 2$	$v(\{1\}) - v(\emptyset) = 0 - 0 = 0$
$\{2\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{1, 2\}) - v(\{2\}) = 0 - 0 = 0$
$\{3\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{1, 3\}) - v(\{3\}) = 1 - 0 = 1$
$\{2, 3\}$	$2! \cdot (3 - 2 - 1)! = 2$	$v(\{1, 2, 3\}) - v(\{2, 3\}) = 1 - 1 = 0$

Tabelle 5.1: Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 1

$S \subseteq N \setminus \{j\}$	$ S !(N -  S  - 1)!$	$v(S \cup \{j\}) - v(S)$
$\emptyset$	$0! \cdot (3 - 0 - 1)! = 2$	$v(\{2\}) - v(\emptyset) = 0 - 0 = 0$
$\{1\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{1, 2\}) - v(\{1\}) = 0 - 0 = 0$
$\{3\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{2, 3\}) - v(\{3\}) = 1 - 0 = 1$
$\{1, 3\}$	$2! \cdot (3 - 2 - 1)! = 2$	$v(\{1, 2, 3\}) - v(\{1, 3\}) = 1 - 1 = 0$

Tabelle 5.2: Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 2

$S \subseteq N \setminus \{j\}$	$ S !(N -  S  - 1)!$	$v(S \cup \{j\}) - v(S)$
$\emptyset$	$0! \cdot (3 - 0 - 1)! = 2$	$v(\{3\}) - v(\emptyset) = 0 - 0 = 0$
$\{1\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{1, 3\}) - v(\{1\}) = 1 - 0 = 1$
$\{2\}$	$1! \cdot (3 - 1 - 1)! = 1$	$v(\{2, 3\}) - v(\{2\}) = 1 - 0 = 1$
$\{1, 2\}$	$2! \cdot (3 - 2 - 1)! = 2$	$v(\{1, 2, 3\}) - v(\{1, 2\}) = 1 - 0 = 1$

Tabelle 5.3: Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 3

Die Anwendung des Shapley-Wertes aus 5.1 ergibt folgende Auszahlungen für die Spieler 1, 2 und 3:

$$\begin{aligned} \phi_1(v) &= \frac{1}{6} \cdot (2 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 2 \cdot 0) = \frac{1}{6} \\ \phi_2(v) &= \frac{1}{6} \cdot (2 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 2 \cdot 0) = \frac{1}{6} \\ \phi_3(v) &= \frac{1}{6} \cdot (2 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 2 \cdot 1) = \frac{4}{6} = \frac{2}{3} \end{aligned}$$

Der Auszahlungsvektor für das Handschuh-Spiel mit drei Spielern sieht wie folgt aus:

$$\phi(v) = (\phi_1(v), \phi_2(v), \phi_3(v)) = \left( \frac{1}{6}, \frac{1}{6}, \frac{2}{3} \right)$$

Es ist erkennbar, dass der Shapley-Wert den Wert der großen Koalition  $v(N) = v(\{1, 2, 3\}) = 1$  auf die drei Spieler verteilt und letztlich die Beiträge der Spieler zur endgültigen Auszahlung  $\phi_1(v) + \phi_2(v) + \phi_3(v) = 1$  aufsummieren lässt.

### 5.3 SHAP-Wert

Nachdem die Terminologie und der Shapley-Wert für kooperative Spiele eingeführt wurde, wird nun die von Štrumbelj und Kononenko in den Arbeiten [SK10] und [SK14] ausgearbeitete Analogie der Shapley-Werte für erklärbare maschinelles Lernen vorgestellt. Der Idee der Shapley-Wert folgend, sei die Koalition  $S$  eine Menge von interpretierbaren Eingabemerkmalen  $j$ , die Auszahlung der Koalition sei der Vorhersagewert des Modells und der marginale Beitrag eines Spielers wird hier als die Auswirkung eines Merkmals definiert. Dabei wird hier auf einen spezifischen Instanz  $x$  bzw.  $x^{(i)}$  fokussiert. Der aus 5.1 bekannte Shapley-Wert wird somit als der lokale Shapley-Wert formuliert:

$$\phi_j = \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{j\}) - f_x(S)] \quad (5.2)$$

Im Gegensatz zu den Spielern aus der Spieltheorie können hier die Merkmale nicht entfernt werden, da einem bereits trainierten Vorhersagemodell Werte für jedes Merkmal übergeben werden müssen. Das heißt, es werden keine Koalitionen mit und ohne Merkmal  $j$  berechnet. Vielmehr werden die Merkmale, die nicht zur aktuellen Merkmalskombination angehören, anhand des Durchschnittswertes über mögliche Werte des Merkmals aus dem Datensatz bestimmt. Zudem wächst die Anzahl der möglichen Merkmalskombinationen exponentiell und somit ist es in der Praxis schwierig, die exakten Shapley -Werte zu berechnen. Hierfür können verschiedene Approximationsmethoden verwendet werden, wobei Štrumbelj und Kononenko die Shapely-Werte mittels Monte-Carlo-Approximation berechnet haben. Es sollte noch erwähnt werden, dass auch im Kontext der Vorhersagemodelle der Shapley -Wert 5.2 die in Abschnitt 5.2 dargestellten Eigenschaften der Effizienz, der Symmetrie, des Nullspielers und der Additivität erfüllt. Die genauen Formulierungen zur effizienten Annäherung und die Eigenschaften der Shapley-Werte können in [SK14] nachgelesen werden.

Die von Lundberg et al. [[LL16], [LL17]] vorgestellte SHAP-Modell verwendet *Additive Feature Attribution Methods*. Solche Methoden haben ein Erklärungsmodell  $g$ , das eine lineare Funktion von binären Variablen ist:

$$g(z') = \phi_0 + \sum_{j=1}^M \phi_j z'_j \quad (5.3)$$

wobei  $z' \in \{0, 1\}^M$ ,  $M$  die Anzahl der vereinfachten Eingabemerkmale und  $\phi_j \in \mathbb{R}$  ist. Dabei weist  $z'$  jedem Merkmal den Wert 1 oder 0 zu, was anzeigt, ob es in der Merkmalskombination auftritt oder nicht. Zudem wird die Prognose des originalen Vorhersagemodells  $f$  für eine bestimmte Instanz  $x$  durch einen lokalen Erklärungsmodell  $g$  verständlich erklärt. Hierbei wird mittels einer Transformationsfunktion  $x = h_x(x')$  eine vereinfachte Darstellung der Eingabemerkmale erreicht. Das Erklärungsmodell  $g$  soll in der Nähe von  $x'$  das Vorhersagemodell  $f$  lokal approximieren, das heißt: Immer dann, wenn  $z' \approx x'$ , sollte  $g(z') \approx f(h_x(z'))$  gelten. Den Autoren zufolge gibt es eine einzige Lösung für  $\phi_j$ , die die folgenden drei wünschenswerten Eigenschaften erfüllt:

- *Local Accuracy*:

Die lokale Genauigkeit erfordert, dass die Ausgabe des Erklärungsmodells  $g$  mit dem ursprünglichen Modell  $f$  für die zu erklärende Vorhersage übereinstimmt:

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^M \phi_j x'_j$$

- *Missingness:*

Eingabemerkmale, bei denen  $x'_j = 0$  ist, d.h. die Merkmale bereits im originalen Datenpunkt  $x_j$  fehlen, haben keine Auswirkung:

$$x'_j = 0 \implies \phi_j = 0$$

- *Consistency/Monotonicity:*

Es sei  $f_x(z') = f(h_x(z'))$  und  $z' \setminus j$  als  $z'_j = 0$ . Wenn für zwei beliebige Modelle  $f$  und  $f'$ :

$$f'_x(z') - f'_x(z' \setminus j) \geq f_x(z') - f_x(z' \setminus j)$$

für alle Eingaben  $z' \{0, 1\}^M$  gilt, dann ist:

$$\phi_j(f', x) \geq \phi_j(f, x)$$

Die SHAP-Formel, die genau diese Eigenschaften erfüllt, ist definiert als:

$$\phi_j(f, x) = \sum_{z' \subseteq x'} \frac{|z'|! (M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus j)] \quad (5.4)$$

wobei  $|z'|$  die Anzahl der Nicht-Null-Elementen in  $z'$  ist. Zudem werden durch  $z' \subseteq x'$  alle  $z'$ -Vektoren dargestellt, indem die Nicht-Null-Elemente eine Teilmenge der Nicht-Null-Elemente in  $x'$  sind. Hierzu wurden von Lundberg et al. als einheitliches Maß für die Auswirkung bzw. Wichtigkeit von Merkmalen die SHAP-Werte vorgestellt. Diese werden als Shapley-Werte einer bedingten Erwartungsfunktion des ursprünglichen Modells aufgefasst. Somit sind die SHAP-Werte die Lösungen für die Gleichung 5.4. Um sie zu berechnen, wird  $f_x(z') = f(h_x(z')) = E[f(z)|z_S]$  definiert, wobei  $S$  die Menge von Nicht-Null-Elementen in  $z'$  ist.

Die folgende Abbildung 5.1 illustriert das Konzept der SHAP-Werte. Der Basiswert  $E[f(z)]$  ist die erwartete Modellvorhersage, d.h. wenn alle Merkmale fehlen, wird die durchschnittliche Vorhersage über den gesamten Datensatz als Basis verwendet. Die Auswirkung eines Merkmals wird als die Änderung des erwarteten Wertes der Modellausgabe beim Beobachten des Merkmals im Vergleich zu keiner Beobachtung des Merkmals definiert. Demgemäß werden vom Basiswert bis zu den Modellausbewerten  $f(x)$  die SHAP-Werte gemäß 5.4 berechnet und jedem Merkmal die jeweilige Änderung der erwarteten Modellvorhersage zugewiesen. Die SHAP-Werte, die den Vorhersagewert nach oben treiben, werden mit roten Pfeilen dargestellt und diejenigen, die den Vorhersagewert nach unten treiben, mit blauen Pfeilen.

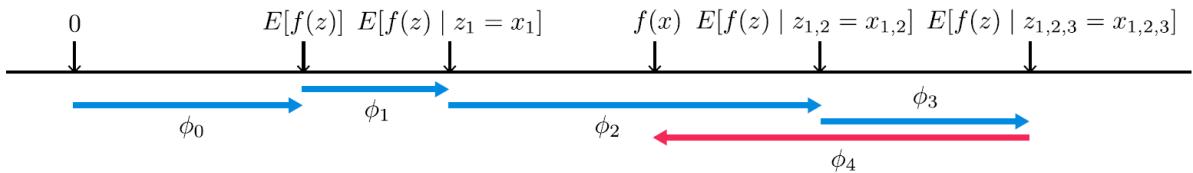


Abbildung 5.1: Schematische Darstellung der SHAP-Werte [LL17]

Abgesehen von lokalen Erklärungen für eine bestimmte Instanz  $x$  kann SHAP auch dazu verwendet werden, um die Wichtigkeit von Merkmalen auf globaler Ebene zu vergleichen. Hierfür wird entweder der vollständige Testdatensatz oder eine Stichprobenmenge betrachtet. Die Merkmale, die den größten Einfluss auf das Gesamtergebnis des Modells haben, werden durch die Berechnung des Durchschnitts der absoluten SHAP-Werte über alle Stichproben identifiziert [LE18]:

$$I_j = \sum_{i=1}^n |\phi_j^{(i)}| \quad (5.5)$$

Mithilfe der von Lundberg et al. entwickelten Python-Implementierung von SHAP kann neben der lokalen Erklärung auch die aggregierte Auswirkung von Merkmalen auf Modellvorhersagen als globale Erklärung veranschaulicht werden. Insgesamt bietet die SHAP-Bibliothek sowohl für modellspezifische als auch modellagnostische Erklärungsmodelle viele verschiedene Visualisierungen für die berechneten SHAP-Werten [LSGH].

## 5.4 Kernel SHAP

Wie bereits in 5.1 erwähnt, beruht auch LIME auf der additiven Merkmalszuordnung 5.3. Für *Additive Feature Attribution Methods* haben Lundberg et al. gezeigt, dass die Shapley-Werte die einzige Lösung sind, die die in 5.3 vorgestellten wünschenswerten Eigenschaften erfüllt. Dabei hängt diese Lösung von der Verlustfunktion  $L$ , dem zugehörigen Ähnlichkeitsmaß bzw. der Gewichtung  $\pi_{x'}(z')$  und dem Regularisierungsterm  $\Omega(g)$  ab, wobei der wesentliche Unterschied zu LIME in der Wahl der Gewichte  $\pi_{x'}(z')$  des Modells liegt. Erst durch folgende spezifische Parameter wird die Approximation der SHAP-Werte ermöglicht:

$$\begin{aligned} \Omega(g) &= 0 \\ \pi_{x'}(z') &= \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)} \\ L(f, g, \pi_{x'}) &= \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi_{x'}(z') \end{aligned} \quad (5.6)$$

Dabei ist  $M$  die Anzahl der Merkmale bzw. die maximale Koalitionsgröße und  $|z'|$  die Anzahl der Nicht-Null-Elemente in  $z'$ . Der Kernel SHAP trainiert somit ein nach der Spieltheorie gewichtetes lineares Regressionsmodell als lokales Ersatzmodell und bestimmt die Koeffizienten des Ersatzmodells als die SHAP-Werte.

Der Ansatz von Kernel SHAP erfolgt im Wesentlichen in folgenden Schritten [MC20]:

1. Generiere Stichproben aus verschiedenen Koalitionen  $z'_k \in \{0, 1\}^M$  für  $k \in \{1, \dots, K\}$ , wobei  $M$  die Anzahl der Merkmale ist. Die 1 bedeutet, das Merkmal ist in der Koalition vorhanden, 0 demgemäß nicht.
2. Verwende die Transformationsfunktion  $h_x$ , um jede Stichprobe  $z'_k$  in die ursprüngliche Merkmalsraum zu konvertieren. Anschließend wird das komplexe Modell  $f$  angewendet:  $f(h_x(z'_k))$ . Die 1 wird durch den tatsächlichen Wert des Merkmals der zu erklärenden Instanz  $x$  dargestellt, und die 0 unterscheidet sich je nach Datentyp. Beispielsweise wird die 0 bei tabellarischen Daten durch einen zufällig ausgewählten Wert aus den jeweiligen Merkmalsdaten ersetzt.
3. Für jede  $z'_k$  wird das Gewicht mit dem Kernel SHAP  $\pi_{x'}(z')$  nach 5.6 berechnet.
4. Die interpretierbaren Stichproben  $z'_k$  und dazugehörigen Modellvorhersagen  $f(h_x(z'_k))$  werden als Trainingsdaten für ein gewichtetes lineares Regressionsmodell  $g$  verwendet.
5. Die Koeffizienten aus dem linearen Modell werden als Shapley-Werte  $\phi_j$  zurückgegeben.

## 5.5 Vergleich von Kernel SHAP und LIME

Die in Kapitel 4 und 5 vorgestellten Methoden für die Erklärbarkeit eines Black-Box-Modells werden nun auf Gemeinsamkeiten und Unterschiede hin untersucht.

Die Methoden Kernel SHAP und LIME sind beide modellagnostisch und können auf Text-, Bild- und Tabellendaten angewendet werden. Beide Methoden verwenden ein lineares Modell  $g$ , um das originale Vorhersagemodell  $f$  lokal zu approximieren und so die Vorhersage für eine Instanz  $x$  zu erklären. Hierbei liefern beide Methoden durch die Aggregation der lokalen Erklärungen eine Möglichkeit, das Modell global zu interpretieren. Zusätzlich werden die Eingabedaten bei beiden Methoden durch eine Transformationsfunktion in ein leicht interpretierbares Datenformat konvertiert. Es sollte noch beachtet werden, dass die Transformationsfunktion und dementsprechend auch die Dimension der interpretierbaren Daten bei Kernel SHAP von LIME unterscheiden. Beispielsweise wird bei Bilddaten Saliency Maps anstatt Superpixel verwendet, bei Tabellendaten in Kernel SHAP werden die numerischen Merkmale nicht binärisiert. Darüber hinaus müssen bei Kernel SHAP zwar für alle Merkmale die SHAP-Werte zurückgegeben werden, aber dafür wurde in Python die Verwendung des linearen Modells  $g$  mit LASSO-Regularisierung implementiert. Demnach werden durch die Merkmalsauswahltechnik aus den  $M$  Merkmalen diejenigen Merkmale bestimmt, die den stärksten Einfluss auf die Vorhersagen haben; die restlichen Merkmale erhalten einen SHAP-Wert, der auf Null geschätzt ist [LSGH].

Beide Methoden beruhen auf additiver Merkmalszuordnung, d.h. die Summe der Koeffizienten aller Merkmale ergibt die endgültige Vorhersage für eine bestimmte Instanz. Allerdings werden die vom lokalen Erklärungsmodell generierten Koeffizienten unterschiedlich aufgefasst. Bei LIME handelt es sich bei diesen Koeffizienten um die Differenz zwischen der Vorhersage mit und ohne das jeweilige Merkmal, bei Kernel SHAP um den approximierten SHAP-Wert als Beitrag eines Merkmals zur Differenz zwischen der tatsächlichen und der mittleren Vorhersage.

Der Hauptunterschied zwischen Kernel SHAP und LIME liegt in der Wahl von  $\Omega(g)$  und  $\pi_{x'}(z')$ . Bei LIME werden die Parameter heuristisch definiert. Im Gegensatz dazu werden sie bei Kernel SHAP mit aus der Spieltheorie abgeleiteten SHAP-Werten begründet [LL16]. Der Hyperparameter Kernel-Breite  $\sigma$ , der für die Berechnung der  $\pi_{x'}(z')$  verwendet wird, ist u.a. der Grund, warum die LIME-Methode instabile Erklärungen generiert. Die aus LIME als Ähnlichkeitsmaß bekannte  $\pi_{x'}(z')$  wird bei Kernel SHAP als Gewichte für die Koalitionen betrachtet. In LIME werden die Stichproben  $z'_i$ , die am meisten Nicht-Null-Elemente haben, als sehr ähnliche Datenpunkte zur Instanz  $x'$  angesehen und dementsprechend auch höher gewichtet. Dagegen erfolgt bei Kernel SHAP die Gewichtung  $\pi_{x'}(z')$  nach der Anzahl der Merkmale in der Koalition  $|z'|$ , wobei kleine Koalitionen (wenige Nicht-Null-Elemente in  $z'_k$ ) und große Koalitionen (viele Nicht-Null-Elemente in  $z'_k$ ) die größten Gewichte erhalten. Dies liegt daran, dass Merkmale erst durch die isolierte Betrachtung auf ihre Auswirkungen auf die Vorhersage hin besser untersucht werden können [MC20]. Im Vergleich zu LIME bietet SHAP aufgrund der mathematisch hergeleiteten Eigenschaften eine hohe Konsistenz und lokale Genauigkeit. Auch wenn es sich bei den generierten Erklärungen von Kernel SHAP um die geschätzten SHAP-Werte handelt, kann damit die theoretisch optimale Lösung approximiert werden.

Nichtsdestotrotz hat auch die Kernel SHAP Methode eine Schwäche, und zwar die hohe erforderliche Rechenzeit. Dabei müssen nicht wie bei der SHAP-Methode  $2^M$  mögliche Kombinationen von Merkmalen beachtet werden, allerdings ist bei Kernel SHAP die Approximation dieser Merkmalskombinationen trotzdem rechenintensiv. Hierfür wurde in Python die Anzahl der Stichproben  $z'_k$  standardmäßig auf  $2M + 2^{11}$  festgelegt, wobei auch diese alternative Formulierung aller möglichen Merkmalskombinationen viel Rechenzeit erfordert [LSGH]. Der wesentliche Nachteil von Kernel SHAP gegenüber LIME ist also die zeitaufwändige Generierung von Erklärungen.

# 6 Implementierung und Evaluation

---

In diesem Kapitel werden die zuvor erläuterten Erklärungsmodelle implementiert und bewertet. Zunächst werden in Abschnitt 6.1 die Daten vorgestellt. Im Anschluss werden in Abschnitt 6.2 die Vorhersagen von drei Black-Box-Modellen verwendet. Zum Schluss werden in Abschnitt 6.3 die Entscheidungen der Black-Box-Modelle mithilfe der Erklärungsmodelle untersucht.

## 6.1 Problemstellung und Datenbeschreibung

Die Grundlage der folgenden Ausführungen sind die Daten eines in Taiwan ansässigen Kreditkartenherausgebers, der die Wahrscheinlichkeit eines Zahlungsausfalls bei Kreditkartenkunden bestimmen möchte. Der aus dem UCI Machine Learning Repository stammende Datensatz enthält Zahlungsinformationen von 30.000 Kunden von April bis September 2005. Das Ziel liegt darin, zum einen Black-Box-Modelle zur Vorhersage des Zahlungsausfalls von Kreditkartenkunden zu verwenden, zum anderen sollen die Faktoren der Modelle bestimmt werden, die zum Zahlungsausfall von Kreditkartenkunden beigetragen haben. Der Datensatz besteht aus 24 unabhängigen Variablen und eine abhängige Variable:

- ID: Kundennummer
- LIMIT\_BAL: Betrag des gewährten Kredits in NT-Dollar (einschließlich Einzel-, Familien- und Zusatzkredit)
- SEX: Geschlecht (1 für männlich und 2 für weiblich)
- EDUCATION: Bildungsstand (1 für einen höheren akademischen Grad, 2 für Universitätsabschluss, 3 für Hochschulreife, 4 für sonstige, 5 und 6 für unbekannt)
- MARRIAGE: Familienstand (1 für verheiratet, 2 für ledig und 3 für sonstige)
- AGE: Alter in Jahren
- PAY\_0 - PAY\_6: Rückzahlungsstatus (-1 für ordnungsgemäße Zahlung, 1 für Zahlungsverzug von einem Monat, 2 für Zahlungsverzug von zwei Monaten, ..., 8 für Zahlungsverzug von acht Monaten und 9 für Zahlungsverzug von neun Monaten und länger); PAY\_0: Rückzahlungsstatus im September 2005, PAY\_2: Rückzahlungsstatus im August 2005, ..., PAY\_6: Rückzahlungsstatus im April 2005
- BILL\_AMT1 - BILL\_AMT6: Höhe des Rechnungsbetrags (NT-Dollar); BILL\_AMT1: Höhe des Rechnungsbetrags im September 2005, BILL\_AMT2: Höhe des Rechnungsbetrags im August 2005, ..., BILL\_AMT6: Höhe des Rechnungsbetrags im April 2005
- PAY\_AMT1 - PAY\_AMT6: Betrag der vorherigen Zahlung (NT-Dollar); PAY\_AMT1: Betrag der vorherigen Zahlung im September 2005, PAY\_AMT2: Betrag der vorherigen Zahlung im August 2005, ..., PAY\_AMT6: Betrag der vorherigen Zahlung im April 2005

- default payment next month: Zahlungsausfall (1 für Ja und 0 für Nein)

Die Implementierung findet in der Jupyter Notebook-Umgebung Google Colab statt. Um die Daten in Python einlesen zu können, müssen zunächst die benötigten Pakete importiert werden. Anschließend kann der Datensatz für die Analyse bereinigt werden. Zu Beginn wird die Spalte der Kundennummer aus dem Datensatz entfernt. Zusätzlich werden *PAY\_0* und die binäre Antwortvariable für die Zahlungsausfall *default payment next month* in *PAY\_1* und *Default Payment* umbenannt. Um die Daten analysieren zu können, müssen sie zunächst bereinigt werden. Optional können weibliche Kunden mit 0 anstatt 2 kodiert werden. Der Familienstand sollte laut Beschreibung der Daten nur die Werte 1, 2 und 3 beinhalten, daher werden Kunden mit einem Wert 0 für den Familienstand den Wert 3 für sonstige erhalten. Zudem kann die Variable für Bildung in die Werte 1 bis 4 gruppiert werden; daher werden die zusätzlichen Werte 0, 5 und 6 der Kategorie 4 als sonstige zugewiesen. Die Variablen *PAY\_1* bis *PAY\_6* haben die nicht erfassten Kategorien -2 und 0. Mit der Annahme, dass diese Werte eine planmäßige Zahlung oder keinen Gebrauch der Kreditkarte symbolisieren, werden die Kategorien -2, -1 und 0 als Kategorie 0 aufgefasst. Nun können diese 23 unabhängigen Variablen als Merkmale für die Anpassung eines Black-Box-Modells zur Prognose der Zahlungsausfalls (Abschnitt 6.2) verwendet werden.

#### section: Vergleich von Black-Box Modellen

Zunächst wird die Anzahl der Zahlungsausfall bezüglich des Geschlechts betrachtet, siehe Abbildung 6.1. Es gibt anscheinend mehr weibliche Kreditnehmer als männliche, und die Frauen haben insgesamt weniger Zahlungsausfälle als Männer.

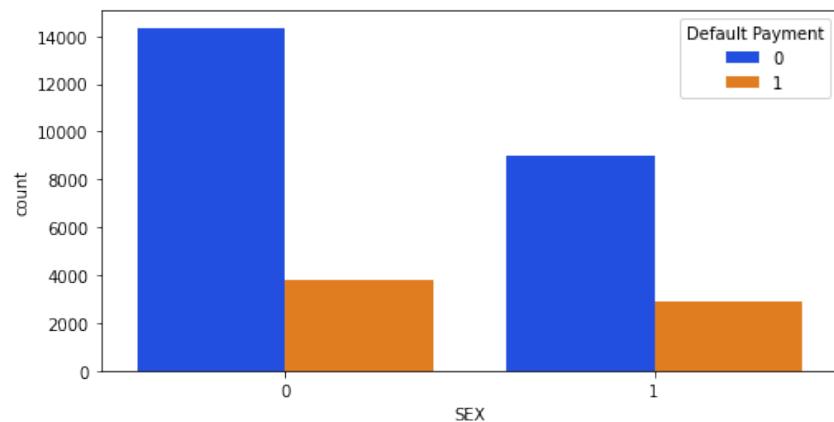


Abbildung 6.1: Anzahl Zahlungsausfall nach Geschlecht

Als Nächstes wird der Familienstand des Kreditnehmers in Abhängigkeit vom Zahlungsausfall visualisiert. Abbildung 6.2 zeigt, dass ledige Kreditnehmer weniger Zahlungsausfälle haben als verheiratete.

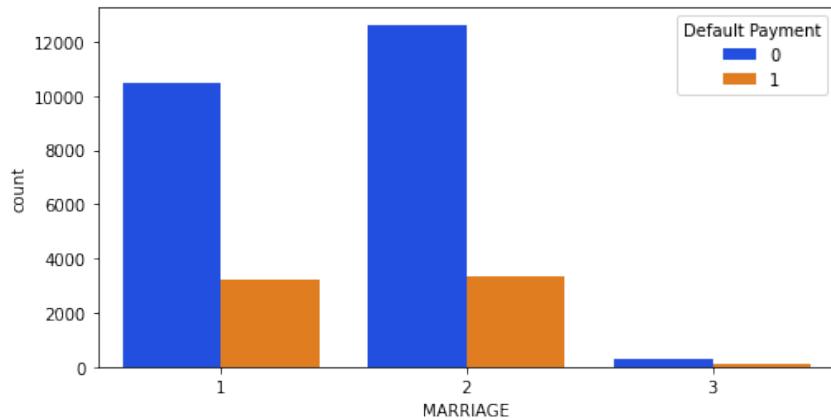


Abbildung 6.2: Anzahl Zahlungsausfall nach Familienstand

Zusätzlich kann das Bildungsniveau des Kreditnehmers betrachtet werden. Anhand von Abbildung 6.3 wird deutlich, dass die meisten Kreditnehmer einen Universitätsabschluss haben. Zudem haben die Kreditnehmer mit einem Universitätsabschluss weniger Zahlungsausfälle als Kreditnehmer mit einem höheren akademischen Grad und Hochschulreife.

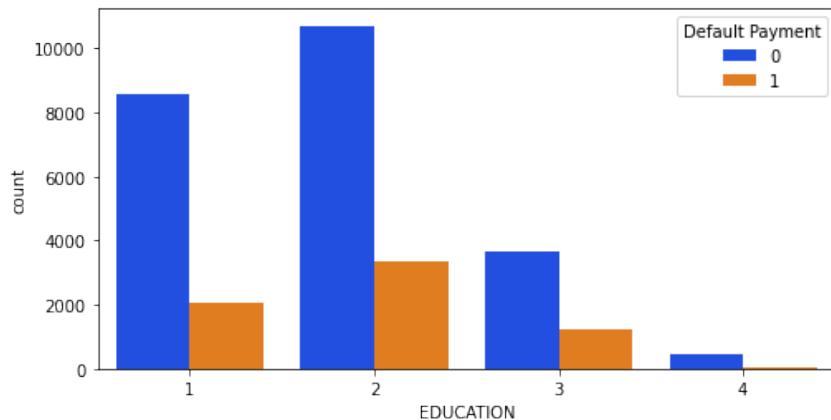


Abbildung 6.3: Anzahl Zahlungsausfall nach Bildungsniveau

In Abbildung 6.4 wird der monatliche Rückzahlungsstatus in Abhängigkeit vom Zahlungsausfall dargestellt.

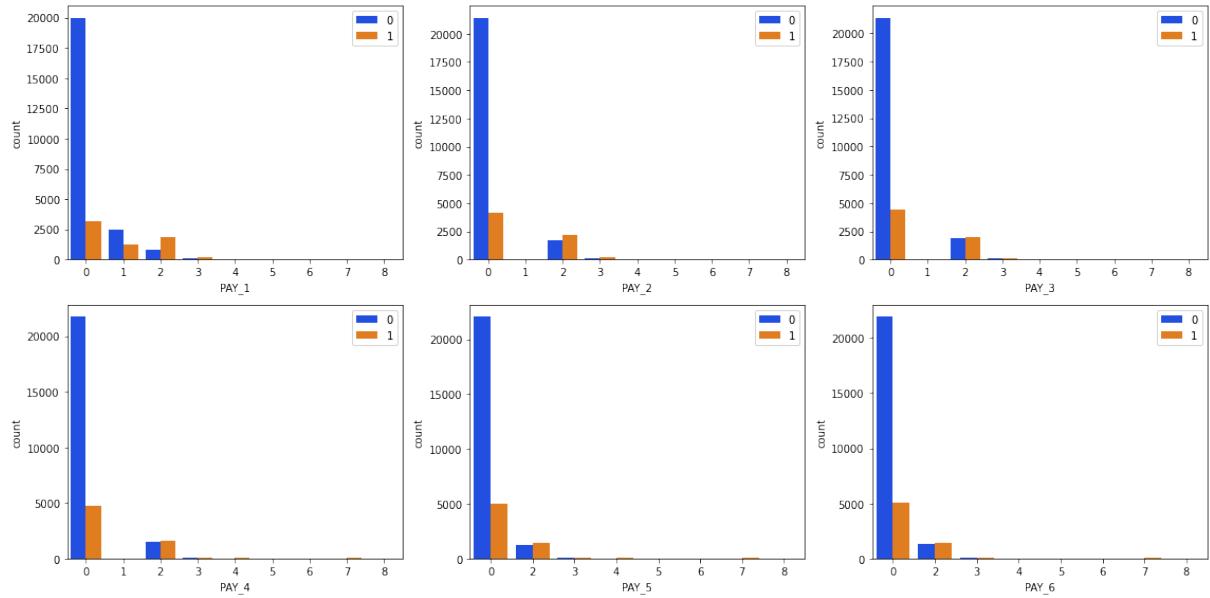


Abbildung 6.4: Anzahl Zahlungsausfall nach der monatlichen Rückzahlungsstatus

Als letztes wird, um die Interaktion zwischen den Variablen anzuzeigen, Pearson-Korrelationskoeffizienten für sie erfasst. Die Korrelationskoeffizienten werden hier als Korrelationsmatrix zwischen den Variablen dargestellt. Dabei gibt jedes Kästchen die Korrelation zwischen den zwei Variablen in der entsprechenden Zeile und Spalte an. Die Korrelationskoeffizienten können Werte zwischen 1 und -1 annehmen, wobei 0 keine Beziehung, -1 eine sehr starke negative Korrelation und 1 eine sehr starke positive Korrelation zwischen den Variablen anzeigt. Aus Abbildung 6.5 wird ersichtlich, dass es wie zu erwarten starke Korrelationen zwischen dem monatlichen Rückzahlungsstatus und zwischen den monatlichen Rechnungsbeträgen gibt. In Abhängigkeit von *Default Payment* weist neben den Variablen *PAY\_1* bis *PAY\_6* auch die Variable *LIMIT\_BAL* eine Korrelation auf. Bei den restlichen Variablen scheinen keine Korrelationen untereinander aufzutreten.

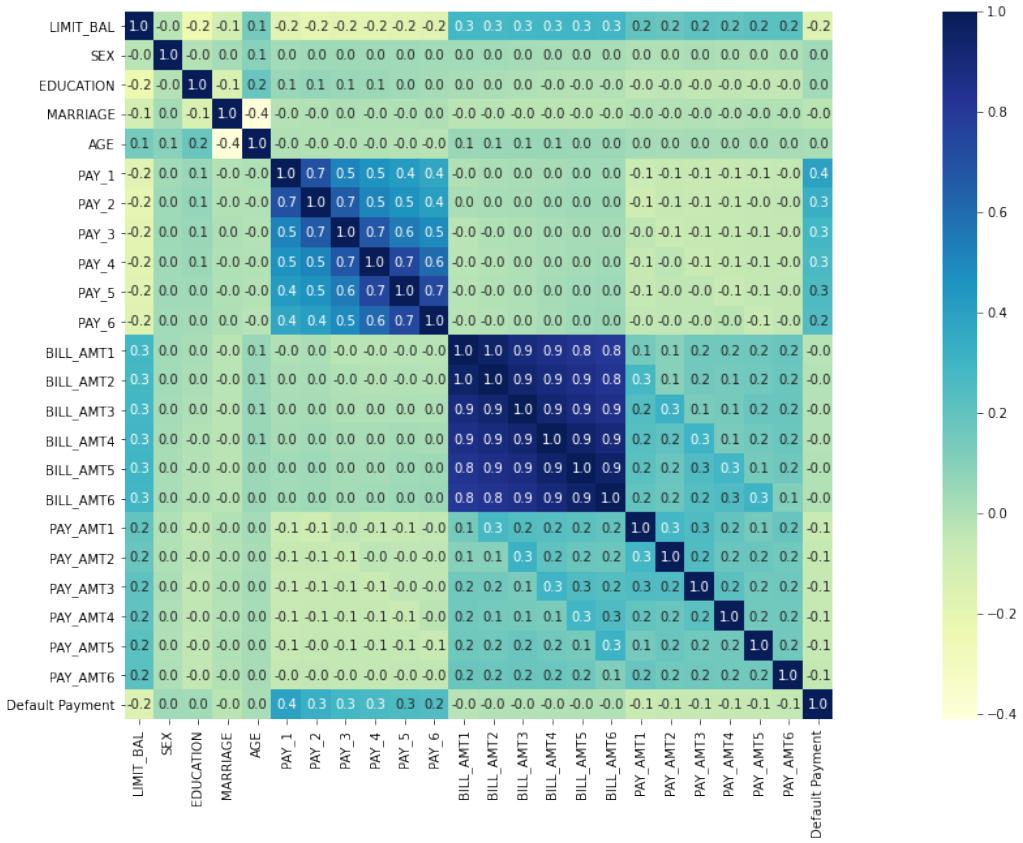


Abbildung 6.5: Korrelationsmatrix

Im Rahmen der Datenaufbereitung werden die numerischen Variablen auf einen Bereich von 0 bis 1 skaliert und die Daten in Test- und Trainingsdaten unterteilt. Mit Hilfe des Trainingsdatensatzes werden die Black-Box-Modelle in Abschnitt 6.2 trainiert und anschließend wird durch den Testdatensatz überprüft, wie gut das Modell mit neuen Daten funktioniert. Beziehungsweise ist das Ziel hier, eine Vorhersage über den Zahlungsausfall der Kreditnehmer im Testdatensatz zu treffen. Zudem wird die absolute und die relative Häufigkeit der Verteilung von 0 und 1 für die Antwortvariable ausgegeben. Dabei fällt auf, dass der Datensatz unausgeglichen ist, da die Klassen für Zahlungsausfall (*Default Payment = 1*) und keinen Zahlungsausfall (*Default Payment = 0*) nicht gleich groß repräsentiert sind. Hierbei kommt *Default Payment = 0* bei 23.364 Beobachtungen bzw. 77.88% und *Default Payment = 1* nur bei 6.636 Beobachtungen bzw. 22.12% vor. Somit kann 0 als Mehrheitsklasse und 1 als Minderheitsklasse deklariert werden. Für das Ziel, die Anzahl von *Default Payment = 1* auf ausgeglichene Weise zu erhöhen, um damit bessere Ergebnisse der Black-Box-Modelle zu erzielen, kann die Resampling-Methode SMOTE (Synthetic Minority Over-sampling Technique) [CB02] angewendet werden. Das Verfahren erzeugt synthetische Datenpunkte der Minderheitsklasse und verwendet hierfür die k-nächste-Nachbarn-Methode. Nach Anwendung von SMOTE ist der Datensatz balanciert und die Anzahl der Beobachtungen in jeder Klasse ausgeglichen. Aus den Trainingsdaten mit 18.669 Beobachtungen für Klasse 0 und 5.331 Beobachtungen für Klasse 1 werden so jeweils 18.669 Beobachtungen für Klasse 0 und Klasse 1.

## 6.2 Vergleich von Black-Box Modellen

In diesem Abschnitt werden nun die ausgewählten Black-Box-Modelle für die Vorhersage des Zahlungsausfalls eines Kreditnehmers vorgestellt. Da es sich um eine Klassifikationsaufgabe handelt, werden Support Vector Machine Classifier, Random Forest Classifier und Multi-Layer Perceptron Classifier auf die Daten angewendet. Der folgende Codeausschnitt zeigt die Implementierung der Support Vector Machine Classifier.

```
svc = SVC(kernel='linear', probability=True)
%time svc.fit(X_train, Y_train)
svc_pred = svc.predict(X_test)
```

Es wurde eine lineare Support Vector Machine Classifier mit in Python standardmäßig implementiertem  $C = 1$  gewählt, wobei die benötigte Rechenzeit für dieses Prozess 5 Minuten und 45 Sekunden beträgt. Es sollte erwähnt werden, dass die standardmäßig auf `False` gesetzte `probability` für die anschließende Verwendung in LIME und SHAP als `probability=True` implementiert werden musste und dass dieser Vorgang einen erheblichen Einfluss auf die benötigte Rechenzeit hat. Zur Quantifizierung der Ergebnisse des Vorhersagemodells bietet es sich an, die Wahrheitsmatrix und die wichtigsten Klassifizierungsmetriken ausgeben zu lassen. In der untenstehenden Abbildung 6.6 ist erkennbar, dass die Erfolgsrate des Modells 79% beträgt, wobei Klasse 0, also kein Zahlungsausfall mit *Default Payment = 0*, 4.073 Mal und Klasse 1, also ein Zahlungsausfall mit *Default Payment = 1*, 677 Mal richtig vorhergesagt wird. Das Ziel besteht darin, Kreditnehmer mit einem Zahlungsausfall, also die positive Klasse mit *Default Payment = 1*, besser vorherzusagen. Dabei sollen die Falsch-negativ-Klassifikationen, also kein Zahlungsausfall vorhergesagt, obwohl tatsächlich ein Zahlungsausfall stattfindet, minimiert und somit ein hoher Recall-Wert erlangt werden. Außerdem sollte, auch wenn bereits das SMOTE Verfahren angewendet wurde, der F1-Wert genauer betrachtet werden, da er eine bessere Metrik bei unausgewogener Klassenverteilung ist.

				precision	recall	f1-score	support
		0	1				
Predicted	0	4073	622	0	0.87	0.87	0.87
	1	628	677	1	0.52	0.52	0.52
Actual				accuracy		0.79	6000
	0			macro avg	0.69	0.69	6000
	1			weighted avg	0.79	0.79	0.79

Abbildung 6.6: Wahrheitsmatrix und Klassifizierungsmetriken für Support Vector Machine Classifier

Mit dem nächsten Codeausschnitt wird nun ein Random Forest Classifier eingesetzt mit 150 als Anzahl der Bäume, die der Algorithmus erstellen soll, wobei hier das Entropie-Kriterium verwendet wird. Zudem beträgt die Rechenzeit hier lediglich 29 Sekunden.

```
rfc = RandomForestClassifier(n_estimators=150, criterion='entropy')
%time rfc.fit(X_train, Y_train)
rfc_pred = rfc.predict(X_test)
```

Anhand der zugehörigen Wahrheitsmatrix und den Klassifizierungsmetriken in Abbildung 6.7 ist erkennbar, dass im Vergleich zu Support Vector Machine Classifier gleich viele Richtig-negativ-Klassifikationen und ein wenig mehr Richtig-positiv-Klassifikationen auftreten.

Predicted			precision	recall	f1-score	support	
	0	1					
Actual			0	0.87	0.87	0.87	4695
0	4073	622	1	0.52	0.51	0.52	1305
			accuracy			0.79	6000
1	635	670	macro avg	0.69	0.69	0.69	6000
			weighted avg	0.79	0.79	0.79	6000

Abbildung 6.7: Wahrheitsmatrix und Klassifizierungsmetriken für Random Forest Classifier

Als letztes wird der Multi-Layer Perceptron Classifier implementiert. Im folgenden Codeausschnitt werden standardmäßig die Optimierungsmethode ADAM und die Aktivierungsfunktion ReLu verwendet. Für die Konstruktion des Netzes werden 1.000 Epochen, drei Schichten und jeweils zehn Knoten definiert. Epoche ist ein kompletter Durchlauf eines Datensatzes, und dementsprechend werden hier die kompletten Daten insgesamt 1.000-mal durch das neuronale Netz geschoben. Hierbei stellt jedes Element im Tupel `hidden_layer_sizes=(10, 10, 10)` mit  $i$  als dessen Index die Anzahl der Knoten an der  $i$ -ten Position dar.

```
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
%time mlp.fit(X_train, Y_train.values.ravel())
mlp_pred = mlp.predict(X_test)
```

Für die Erstellung von Prognosen für einen Zahlungsausfall durch den Multi-Layer Perceptron Classifier wurden 54 Sekunden Rechenzeit benötigt. Abbildung 6.8 zeigt, dass die Erfolgsrate mit 73% deutlich kleiner ist als beim Support Vector Machine Classifier (79%) und beim Random Forest Classifier (79%). Durch die Wahrheitsmatrix wird wiederum deutlich, dass die Anzahl der richtig vorhergesagten Zahlungsausfälle, also  $\text{Default Payment} = 1$ , auf 826 gestiegen ist, wobei die Anzahl der Falsch-negativ-Klassifikationen nur noch 479 beträgt. Verglichen mit den beiden vorherigen Modellen werden also durch Multi-Layer Perceptron Classifier bessere Vorhersagen für das Ereignis eines Zahlungsausfalls gemacht.

Predicted			precision	recall	f1-score	support	
	0	1					
Actual			0	0.88	0.76	0.82	4695
0	3566	1129	1	0.42	0.63	0.51	1305
			accuracy			0.73	6000
1	479	826	macro avg	0.65	0.70	0.66	6000
			weighted avg	0.78	0.73	0.75	6000

Abbildung 6.8: Wahrheitsmatrix und Klassifizierungsmetriken für Multi-Layer Perceptron Classifier

In Tabelle 6.1 sind die Ergebnisse der Klassifizierungsmetriken für die vorgestellten Vorhersagemodelle zusammengefasst. Dabei fällt, wie schon erwähnt, die Verbesserung von Recall- und F1-Wert des Multi-Layer Perceptron Classifiers besonders auf, wobei dieser Klassifikator lediglich mit einer Erfolgsrate von 73% vorhersagen kann, ob die Zahlung eines Kreditkartenkunden im nächsten Monat ausfallen wird.

Zusätzlich kann die Leistung der Modelle durch eine ROC-Kurve visualisiert werden. Abbildung ?? zeigt die ROC-Kurven mit den dazugehörigen AUC-Wertes. Auch hier wird festgestellt, dass der AUC-Wert von Multi-Layer Perceptron Classifier besser ist als der der anderen Modelle.

Black-Box Modell	Precision	Recall-Wert	F1-Wert	Accuracy	ROC
Support Vector Machine Classifier	0.52	0.52	0.52	0.79	0.69
Random Forest Classifier	0.52	0.51	0.52	0.79	0.69
Multi-Layer Perceptron Classifier	0.42	0.63	0.53	0.73	0.70

Tabelle 6.1: Übersicht der Klassifizierungsmetriken

Dabei ist der AUC-Wert von 0,696 der ROC-Kurve zwar nicht sehr nah an 1, aber dennoch größer als 0,5.

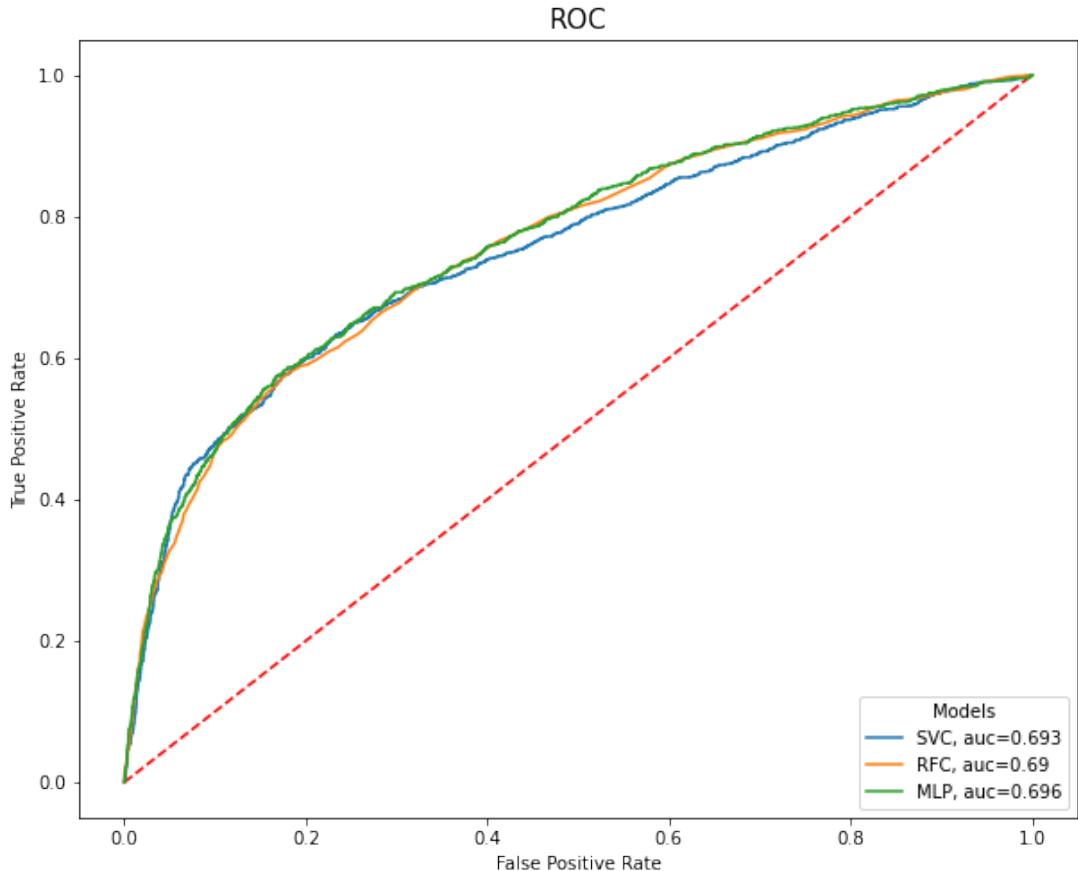


Abbildung 6.9: ROC-Kurven

### 6.3 Implementierung und Auswertung von Erklärungsmodellen

Nun können die Erklärungen von LIME und Kernel SHAP für die vorgestellten Black-Box-Modelle generiert werden. Hierfür werden in Unterabschnitt 6.3.1 zunächst die Erklärungsmodelle implementiert und anschließend mithilfe von vier Beispielen für eine Erklärung der Vorhersage einer Instanz analysiert. Zusätzlich wird in Unterabschnitt 6.3.2 anhand eines Beispiels die Wirkung der unterschiedlichen Kernel-Breiten für die LIME-Erklärungen untersucht. Anschließend werden in Unterabschnitt 6.3.3 die jeweiligen globalen Erklärungen der Modelle gegenübergestellt.

### 6.3.1 Analyse von LIME- und Kernel-SHAP-Erklärungen

In diesem Unterabschnitt werden Merkmale identifiziert, die zur Vorhersage beitragen, und ihre Auswirkungen auf die Vorhersage erläutert. Die folgenden Codeausschnitte beziehen sich nur auf die Implementierung von LIME und Kernel SHAP für den Multi-Layer Perceptron Classifier, wobei der Ablauf für die beiden anderen Modelle identisch ist. Die Ausgabe der Black-Box-Modelle muss zunächst für die LIME- und Kernel-SHAP-Eingabe modifiziert werden.

```
predict_mlp = lambda x: mlp.predict_proba(x).astype(float)
```

Als Nächstes wird das LIME-Erklärungsmodell für tabellarische Daten eingerichtet. Hierbei werden nur die Trainingsdaten, Merkmalsnamen und die beiden Klassennamen definiert: Klasse 0 für kein Zahlungsausfall und Klasse 1 für Zahlungsausfall. Der Rest wird von der standardmäßigen Implementierung von LIME für Tabellendaten übernommen, darunter 5.000 als Umfang der Stichprobe für jede Instanz, eine Kernel-Breite  $\sigma = \sqrt{23} \cdot 0,75 = 3.597$  und die euklidische Distanz als Distanzfunktion  $D$ .

```
explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,
class_names=Y_train.unique(), feature_names=X_train.columns,
categorical_features=categorical_names, verbose=True)
```

Die Ausgabe der LIME-Erklärung soll nur die Auswirkung der fünf wichtigsten Merkmale auf die Prognose veranschaulichen.

```
i = 35
%time exp = explainer_lime.explain_instance(X_test.iloc[i],
predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Auch bei Kernel SHAP muss zunächst die Initialisierung des Erklärungsmodells stattfinden.

```
explainer = shap.KernelExplainer(predict_mlp, X_train)
```

Anschließend kann das Kernel-SHAP-Erklärungsmodell aufgerufen werden. Hierbei muss vor der Verwendung das Hilfsmittel `shap.initjs()` zur Visualisierung initialisiert werden. Zudem werden hier nur 100 ausgewählte Stichproben als Datensatz übergeben, denn je mehr Daten ausgewählt werden, umso länger dauert die Rechenzeit.

```
shap.initjs()
i = 35
%time shap_values = explainer.shap_values(X_test.iloc[i], nsamples=100)
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc[i],
link="identity")
```

#### Instanz 1

Zunächst wird der Datenpunkt 35 als erste Instanz untersucht, wobei für diesen Kreditnehmer die Wahrscheinlichkeit für einen Zahlungsausfall von allen Modellen richtig als Klasse 0 prognostiziert wurde. Damit gab es eine richtig negative Klassifizierung dieser Instanz mit folgenden Werten:  $SEX=1$ ,  $EDUCATION=2$ ,  $MARRIAGE=2$ ,  $PAY\_1=0$ ,  $PAY\_2=0$ ,  $PAY\_3=0$ ,  $PAY\_4=0$ ,  $PAY\_5=0$ ,  $PAY\_6=0$ ,  $LIMIT\_BAL=120.000$ ,  $AGE=32$ ,  $BILL\_AMT1=114.476$ ,  $BILL\_AMT2=113.995$ ,  $BILL\_AMT3=116.574$ ,  $BILL\_AMT4=117.041$ ,  $BILL\_AMT5=50.980$ ,  $BILL\_AMT6=51.285$ ,  $PAY\_AMT1=4.100$ ,  $PAY\_AMT2=6.000$ ,  $PAY\_AMT3=5.000$ ,  $PAY\_AMT4=5.000$ ,  $PAY\_AMT5=58.895$ ,  $PAY\_AMT6=5.000$ .

Zuerst wird LIME auf den Support Vector Machine Classifier angewendet, wobei die ausgegebene LIME-Erklärung immer aus vier Teilen besteht. Der obere linke Teil gibt als erstes die Konstante (Intercept) des lokalen linearen Modells an, danach werden die Vorhersage des LIME-Modells und die originale Vorhersage des Black-Box-Modells ausgegeben. Zudem wird hier die Rechenzeit für die Ausführung des LIME-Modells und der  $R^2$ -Wert angezeigt (näheres dazu in Unterabschnitt 6.3.2). Der untere linke Teil gibt zusätzlich die vom Black-Box-Modell berechneten Wahrscheinlichkeiten für die Vorhersage von Klasse 0, kein Zahlungsausfall, und Klasse 1, Zahlungsausfall, an. Der mittlere Teil gibt in diesem Fall die fünf wichtigsten Merkmale an. Die orangefarbenen Merkmale unterstützen die Klasse 1, die blauen Merkmale Klasse 0. Der rechte Teil zeigt die gleichen Farbkennzeichnungen wie der linke und mittlere Teil und enthält die tatsächlichen Werte für die 5 wichtigsten Merkmale.

In Abbildung 6.10 ist erkennbar, dass die Vorhersage des Support Vector Machine Classifier eine Wahrscheinlichkeit von 36% für Klasse 1 ergibt, womit mit einer Wahrscheinlichkeiten von 64% diese Instanz als Klasse 0 vorhergesagt wird.  $PAY\_1 = 0$  und  $PAY\_2 = 0$  sind die am stärksten beeinflussenden Merkmale, die die Vorhersage in Richtung Klasse 0 antreiben. Die Vorhersage des LIME-Modells 0,369 wird dabei durch die Summe der Konstante und der Koeffizienten ermittelt. Hierfür können die Koeffizienten der abgebildeten Merkmale als Liste ausgegeben werden und wie folgt nachgerechnet werden:

$$0,869 - 0,311 - 0,201 + 0,004 + 0,004 + 0,004 = 0,369$$

Würde z.B. das Merkmal  $PAY\_1 = 0$  entfernt, würde die Vorhersage mit 0,311 weniger Zuversicht keinen Zahlungsausfall klassifizieren.

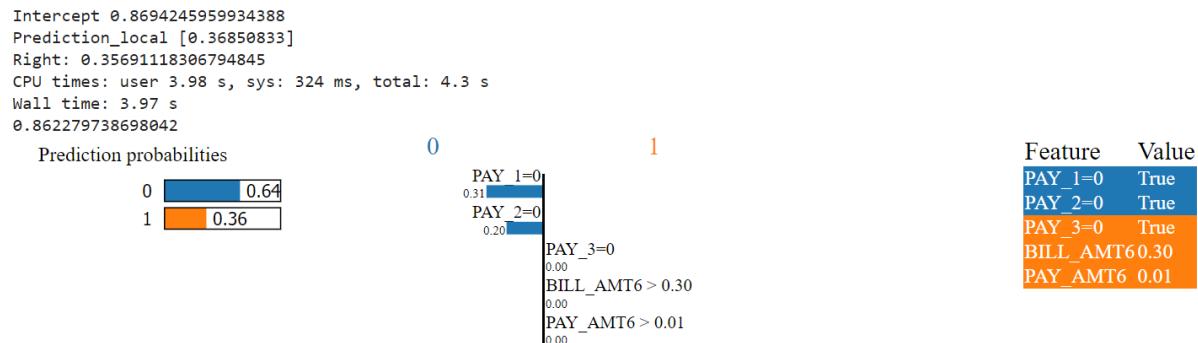


Abbildung 6.10: LIME-Erklärung zum Support Vector Machine Classifier für Instanz 1

In Abbildung 6.11 ist die LIME-Erklärung für Random Forest Classifier dargestellt. Hierzu sollte noch erwähnt werden, dass der endgültige Vorhersagewert von 0,81 das Modell in Bezug auf die Vorhersage der Klasse 0 sehr zuversichtlich macht.

Als Nächstes wird die LIME-Erklärung für den Multi-Layer Perceptron Classifier generiert. In Abbildung 6.12 ist zu sehen, dass auch hier v.a. die Merkmale  $PAY\_1 = 0$  und  $PAY\_2 = 0$  zum Vorhersagewert von 0,79 beigetragen haben.

Da bei den Ergebnissen für den Multi-Layer Perceptron Classifier eine höhere Leistung in Bezug auf die Recall- und F1-Wert erzielt wurde, wird im Folgenden die dazugehörige Kernel-SHAP-Erklärung generiert. Hierbei unterscheidet sich die Interpretation von Kernel SHAP und LIME. Zum einen liefert Kernel SHAP kein Vorhersagemodell wie bei LIME, weshalb auch kein  $R^2$ -Wert berechnet werden kann. Zum anderen werden SHAP-Werte für alle Merkmale berechnet, wobei durch die in Abschnitt 5.5 erläuterte L1-Regularisierung nur die Merkmale, die den stärksten Einfluss auf die Vorhersage haben, visualisiert werden. Die restlichen Merkmale erhalten einen

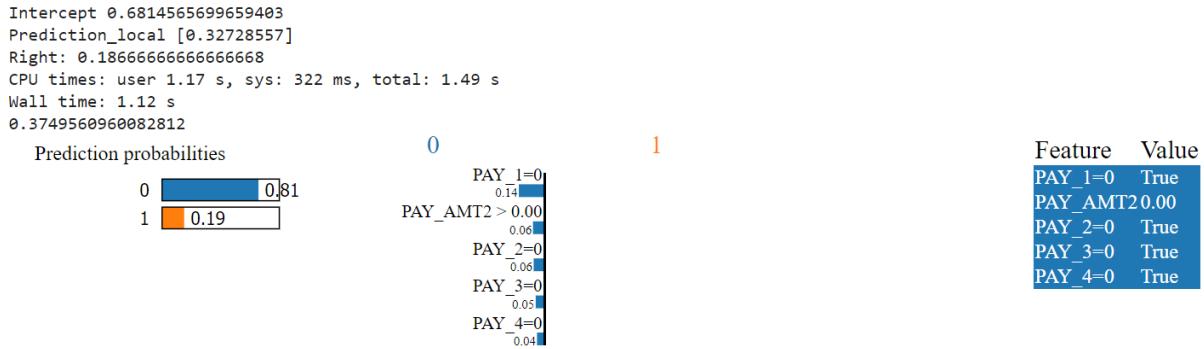


Abbildung 6.11: LIME-Erklärung zum Random Forest Classifier für Instanz 1

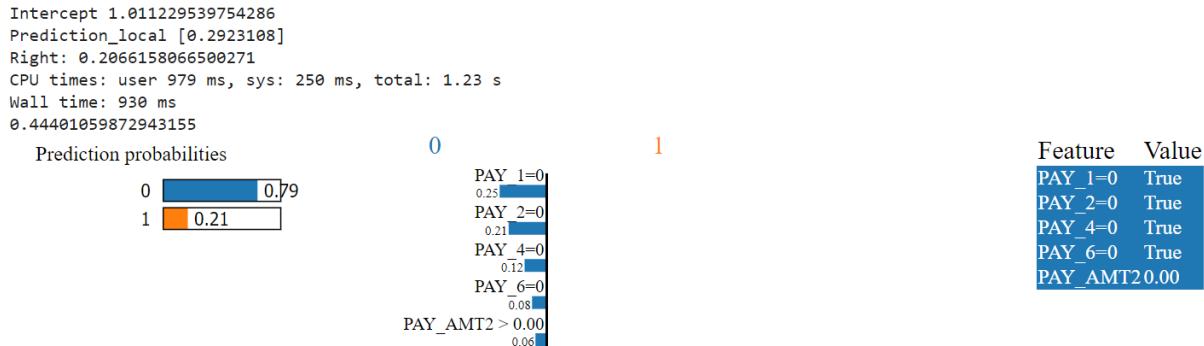


Abbildung 6.12: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1

SHAP-Wert von Null. Dabei ist der SHAP-Wert der durchschnittliche marginale Beitrag eines Merkmalswerts über alle möglichen Merkmalskombinationen.

In Abbildung 6.13 ist zu erkennen, dass die Erklärung von Kernel SHAP für diese Instanz einen Basiswert, also den Erwartungswert  $E[f(z)]$ , von 0,515 berechnet, wobei die vorhergesagte Wahrscheinlichkeit von Multi-Layer Perceptron Classifier gleich 0,21 ist. Normalerweise zeigen rote Balken die Merkmale, die die Vorhersage nach rechts (höher) treiben, blaue Balken dement sprechend Merkmale, die die Vorhersage nach links (niedriger) treiben. Die Größe der Pfeile ist dabei proportional zur Größe des Beitrags. In diesem Fall wurden aber nur Merkmale mit einem negativen Beitragswert, die also zur Verringerung der Modellvorhersage beitragen, identifiziert. Der Merkmal  $PAY\_AMT5=0,1381$  (skaliert) bzw.  $PAY\_AMT5=58.895$  hat am meisten dazu beigetragen, vom Basiswert zum Ausgabewert zu gelangen. Genau wie die LIME-Erklärung von anderen Black-Box-Modellen hat auch die von Kernel SHAP generierte Erklärung des Multi-Layer Perceptron Classifier die Merkmale  $PAY\_1 = 0$  und  $PAY\_2 = 0$  als großen Einfluss für die Vorhersage der Klasse 0 ausgegeben. Zum ersten Mal tritt das Merkmal  $MARRIAGE = 2$  als Einflussfaktor auf, wobei es hier als dritt wichtigster Beitrag identifiziert wurde.

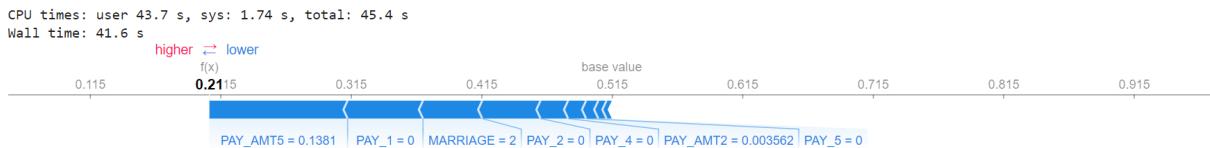


Abbildung 6.13: Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1

Zudem ist es möglich, die berechneten SHAP-Werte ausgeben zu lassen (vgl. Abbildung 6.14). Zum Beispiel handelt es sich bei dem vorletzten, 22. Merkmal um den einflussreichsten Faktor

$PAY\_AMT5$  mit einem SHAP-Wert von  $\phi_{22} \approx 0,106$ .

```
array([-0.00403602,  0.          , -0.04542741, -0.05777129, -0.04465046,
       -0.00629676, -0.02115615, -0.00933667,  0.          ,  0.          ,
       0.          ,  0.          ,  0.          ,  0.          ,  0.          ,
       0.          ,  0.          ,  0.          , -0.01378795,  0.          ,
       0.          , -0.10593468,  0.          ])
```

Abbildung 6.14: Ausgabe der Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1

Der Basiswert und alle SHAP-Werte summieren sich am Ende auf die vorhergesagte Wahrscheinlichkeit des Multi-Layer Perceptron Classifier:

$$0,515 - 0,004 - 0,045 - 0,058 - 0,045 - 0,006 - 0,021 - 0,009 - 0,014 - 0,106 = 0,207 \approx 0,21$$

## Instanz 2

Als zweite Instanz wird der Datenpunkt 2675 analysiert. Für diesen Kreditnehmer wurde von allen Modellen richtig Klasse 1 vorhergesagt, d.h. es liegt eine richtig positive Klassifizierung vor. Es folgt die Beschreibung der Instanz:  $SEX=0$ ,  $EDUCATION=2$ ,  $MARRIAGE=1$ ,  $PAY\_1=2$ ,  $PAY\_2=0$ ,  $PAY\_3=0$ ,  $PAY\_4=0$ ,  $PAY\_5=0$ ,  $PAY\_6=0$ ,  $LIMIT\_BAL=30.000$ ,  $AGE=42$ ,  $BILL\_AMT1=5.288$ ,  $BILL\_AMT2=6.310$ ,  $BILL\_AMT3=8.693$ ,  $BILL\_AMT4=11.541$ ,  $BILL\_AMT5=13.344$ ,  $BILL\_AMT6=15.118$ ,  $PAY\_AMT1=1.117$ ,  $PAY\_AMT2=2.500$ ,  $PAY\_AMT3=3.000$ ,  $PAY\_AMT4=2.000$ ,  $PAY\_AMT5=2.000$ ,  $PAY\_AMT6=553$ .

In Abbildung 6.15 ist erkennbar, dass der Wert von  $PAY\_1 = 2$  die Wahrscheinlichkeit für einen Zahlungsausfall am stärksten erhöht, während  $PAY\_2 = 0$  eher darauf hinweist, dass kein Zahlungsausfall stattfinden wird.

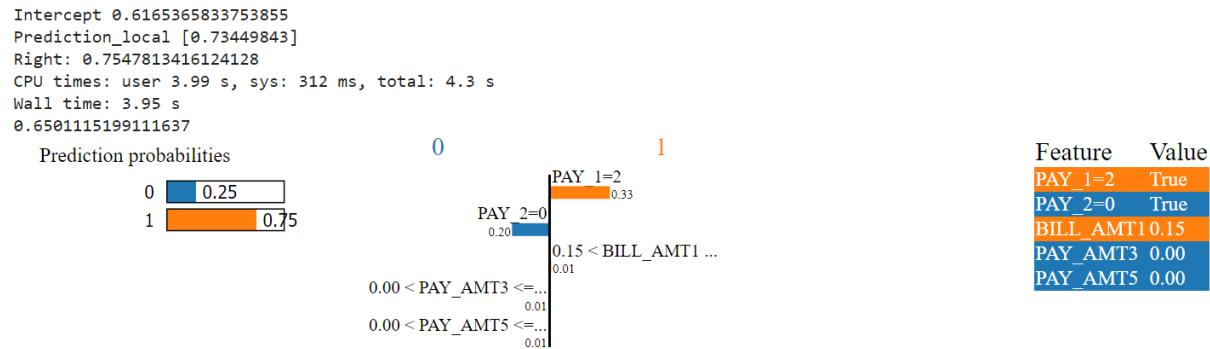


Abbildung 6.15: LIME-Erklärung zum Support Vector Machine Classifier für Instanz 2

Abbildung 6.16 zeigt, dass die wichtigsten Merkmale für die Vorhersage von Klasse 1  $PAY\_1 = 2$  und  $LIMIT\_BAL <= 0,04$  sind.

Auch bei der LIME-Erklärung des Multi-Layer Perceptron Classifier wird Merkmal  $PAY\_1 = 2$  als stärkster Einfluss für die Vorhersage von Klasse 1 und Merkmal  $PAY\_2 = 0$  für die Vorhersage von Klasse 0 identifiziert (vgl. Abbildung 6.17).

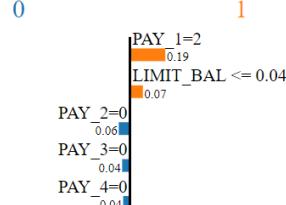
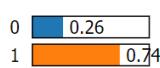
Die Kernel-SHAP-Erklärung in Abbildung 6.18 zeigt, wie die Merkmale  $PAY\_1 = 2$  und  $LIMIT\_BAL = 0,0202$  bzw.  $LIMIT\_BAL = 30.000$  zur Erhöhung der Vorhersage vom Basiswert zum Ausgabewert des Modells beitragen. Diese beiden Merkmale wurden auch von LIME-Erklärungen zum Random Forest Classifier und Multi-Layer Perceptron Classifier als wichtigste Einflussgrößen für die Entscheidung der Klasse 1 ausgegeben.

```

Intercept 0.5174806674038772
Prediction_local [0.6404675]
Right: 0.74
CPU times: user 1.13 s, sys: 300 ms, total: 1.43 s
Wall time: 1.09 s
0.39781062691381536

```

Prediction probabilities



Feature	Value
PAY_1=2	True
LIMIT_BAL	0.02
PAY_2=0	True
PAY_3=0	True
PAY_4=0	True

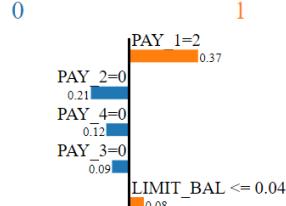
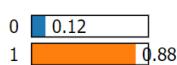
Abbildung 6.16: LIME-Erklärung zum Random Forest Classifier für Instanz 2

```

Intercept 0.7628486133153582
Prediction_local [0.79443109]
Right: 0.881724734200468
CPU times: user 966 ms, sys: 267 ms, total: 1.23 s
Wall time: 930 ms
0.5244831116770893

```

Prediction probabilities



Feature	Value
PAY_1=2	True
PAY_2=0	True
PAY_4=0	True
PAY_3=0	True
LIMIT_BAL	0.02

Abbildung 6.17: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2

```

CPU times: user 44.4 s, sys: 618 ms, total: 45 s
Wall time: 41.7 s

```



Abbildung 6.18: Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2

### Instanz 3

Als dritte Instanz wird der Datenpunkt 555 ausgewählt, wobei der Kreditnehmer zwar keinen Zahlungsausfall hatte, aber dennoch von allen Modellen als Klasse 1, also Zahlungsausfall, klassifiziert wurde. Somit fand eine falsch positive Klassifizierung statt, wobei der Datenpunkt die folgenden Werte hat:  $SEX=1$ ,  $EDUCATION=2$ ,  $MARRIAGE=1$ ,  $PAY\_1=2$ ,  $PAY\_2=2$ ,  $PAY\_3=0$ ,  $PAY\_4=0$ ,  $PAY\_5=0$ ,  $PAY\_6=0$ ,  $LIMIT\_BAL=220000$ ,  $AGE=40$ ,  $BILL\_AMT1=222.418$ ,  $BILL\_AMT2=206.473$ ,  $BILL\_AMT3=208.464$ ,  $BILL\_AMT4=175.407$ ,  $BILL\_AMT5=175.170$ ,  $BILL\_AMT6=174.726$ ,  $PAY\_AMT1=4$ ,  $PAY\_AMT2=8.027$ ,  $PAY\_AMT3=6.218$ ,  $PAY\_AMT4=6.220$ ,  $PAY\_AMT5=6.316$ ,  $PAY\_AMT6=6.504$ .

Die folgende Abbildung 6.19 zeigt die LIME-Erklärung, bei der richtigerweise die Merkmale  $PAY\_1 = 2$  und  $PAY\_2 = 2$  als Indikatoren für die Klassifizierung von Zahlungsausfall identifiziert wurden. Nichtsdestotrotz fand bei diesem Kreditkunden kein Zahlungsausfall statt.

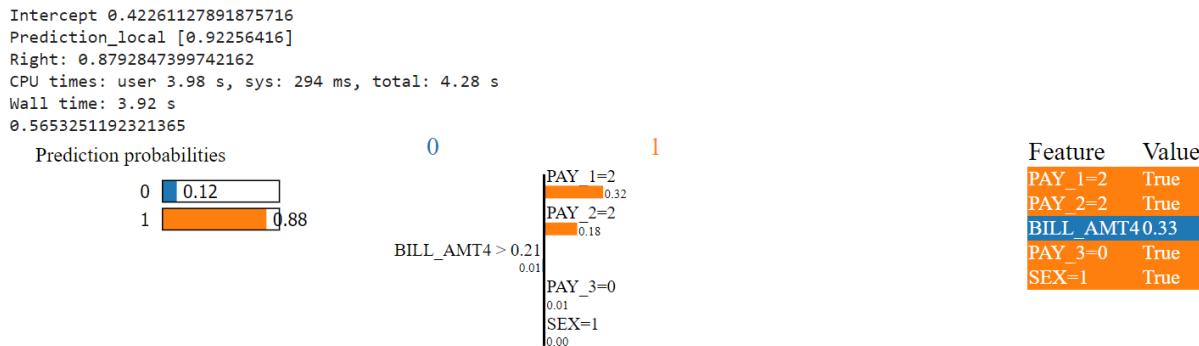


Abbildung 6.19: LIME-Erklärung zum Support Vector Machine Classifier für Instanz 3

In Abbildung 6.20 weisen die Werte von  $PAY\_1 = 2$  und  $PAY\_2 = 2$  darauf hin, dass die Vorhersage Klasse 1 sein sollte, während der Wert von  $PAY\_AMT2 > 0,00$  auf keinen Zahlungsausfall hindeutet.

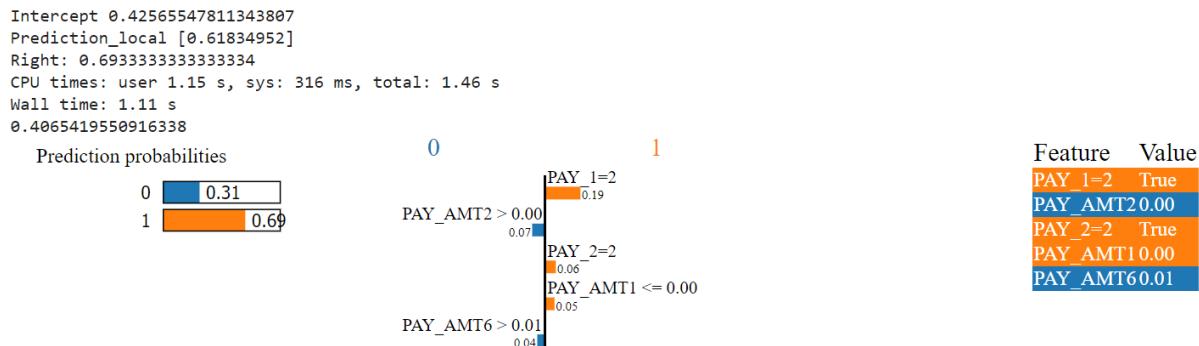


Abbildung 6.20: LIME-Erklärung zum Random Forest Classifier für Instanz 3

Wie beim Support Vector Machine Classifier werden auch hier beim Multi-Layer Perceptron Classifier v.a. die Merkmale  $PAY\_1 = 2$  und  $PAY\_2 = 2$  als größte Einflüsse auf die Vorhersage von Klasse 1 ausgegeben, wobei durch das Merkmal  $PAY\_4 = 0$  eigentlich ein gutes Indiz für die richtige Klasse 0 vorliegt (vgl. Abbildung 6.21).

Auch bei Kernel-SHAP in Abbildung 6.22 wurden die Merkmale  $PAY\_1 = 2$  und  $PAY\_2 = 2$  als die wichtigsten Einflussgrößen für die Entscheidung der Klassifizierung von 1 festgestellt. Zusätzlich wurden, im Gegensatz zu den LIME-Erklärungen, die Merkmale  $BILL\_AMT3=0,2008$  bzw.

```

Intercept 0.575174170239024
Prediction_local [0.88712598]
Right: 0.8822306520804934
CPU times: user 973 ms, sys: 245 ms, total: 1.22 s
Wall time: 922 ms
0.47404336021716476

```

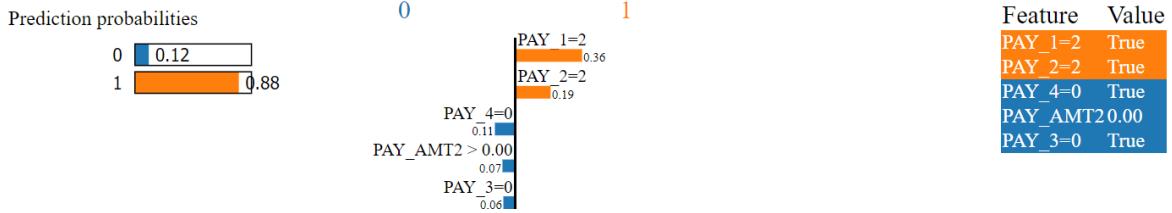


Abbildung 6.21: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 3

*BILL\_AMT3=208.464* als positive Beiträge für die Entscheidung des Multi-Layer Perceptron Classifier ermittelt.

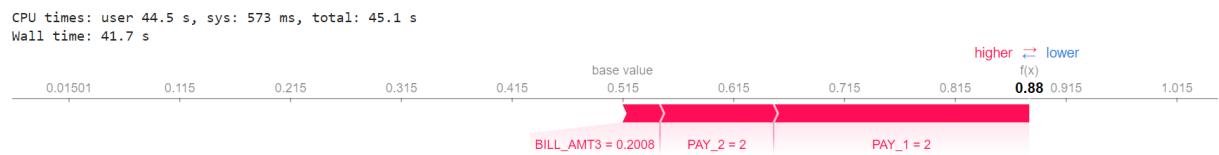


Abbildung 6.22: Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 3

#### Instanz 4

Als letztes wird eine vierte Instanz, der Datenpunkt 1880, untersucht. Hier wurde der Kreditnehmer fälschlicherweise als Klasse 0 klassifiziert, d.h. dieser Kreditkartenkunde hatte tatsächlich einen Zahlungsausfall, wurde aber von den Black-Box-Modellen nicht als solches identifiziert. Dieser Fall gehört zu den falsch negativen Klassifizierungen mit folgender Beschreibung der Instanz: *SEX=1*, *EDUCATION=3*, *MARRIAGE=2*, *PAY\_1=0*, *PAY\_2=0*, *PAY\_3=0*, *PAY\_4=0*, *PAY\_5=0*, *PAY\_6=0*, *LIMIT\_BAL=120.000*, *AGE=50*, *BILL\_AMT1=390*, *BILL\_AMT2=0*, *BILL\_AMT3=780*, *BILL\_AMT4=0*, *BILL\_AMT5=0*, *BILL\_AMT6=0*, *PAY\_AMT1=0*, *PAY\_AMT2=780*, *PAY\_AMT3=0*, *PAY\_AMT4=0*, *PAY\_AMT5=0*, *PAY\_AMT6=0*.

Die in Abbildung 6.23 - 6.25 dargestellten Ergebnisse der LIME-Erklärungen zeigen ein weiteres Mal die Merkmale, die primär für die Vorhersage des Modells verantwortlich waren. Hierbei fällt auf, dass die Vorhersage der Black-Box-Modelle v.a. auf den Merkmalen *PAY\_1 = 0* und *PAY\_2 = 0* basieren. Diese Merkmale sind, wie oben schon gesehen, gute Indizien für die Klassifizierung kein Zahlungsausfall. In diesem Fall hat aber tatsächlich ein Zahlungsausfall stattgefunden, was die Modelle nicht richtig prognostizieren konnten.

Im Vergleich zu vorherigen Erklärungen (siehe Abbildungen 6.13, 6.18 und 6.22) hat diesmal der Kernel-SHAP Merkmale festgestellt, die positive und negative Beitragswerte haben. Dabei wurde nicht nur das Merkmal *PAY\_1 = 0*, sondern auch Merkmale wie *EDUCATION = 3* und *MARRIAGE = 2* als Beiträge für die vorhergesagte Wahrscheinlichkeit identifiziert (vgl. Abbildung 6.26).

Zusammenfassend sind LIME- und Kernel-SHAP-Erklärungen sehr aufschlussreich und konnten sinnvolle Beschreibungen für die Entscheidung der Black-Box-Modelle liefern. Die generierten Erklärungen erscheinen plausibel, z.B. dass der Rückzahlungsstatus *PAY\_1* und *PAY\_2* als wichtigste Indikatoren für die Vorhersage der Klassifizierung dienen. Gilt für einen Kreditkar-

```

Intercept 0.882546408232073
Prediction_local [0.36422746]
Right: 0.3569330122544934
CPU times: user 4.04 s, sys: 317 ms, total: 4.36 s
Wall time: 4 s
0.8700084063477235

```

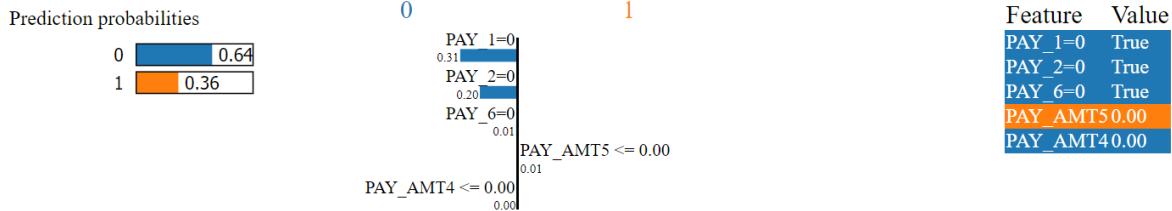


Abbildung 6.23: LIME-Erklärung zum Support Vector Machine Classifier für Instanz 4

```

Intercept 0.64399487276505
Prediction_local [0.41942654]
Right: 0.2133333333333335
CPU times: user 1.17 s, sys: 308 ms, total: 1.48 s
Wall time: 1.11 s
0.3326753833437992

```

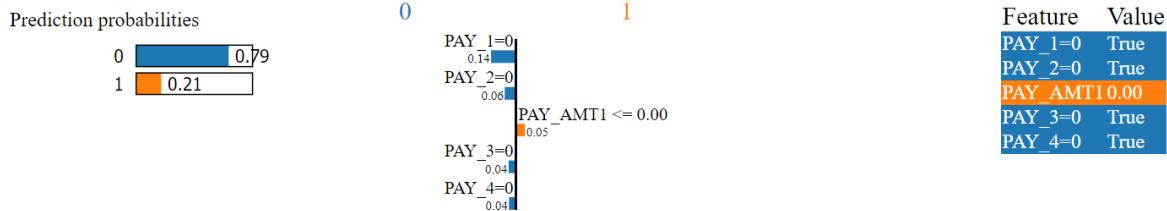


Abbildung 6.24: LIME-Erklärung zum Random Forest Classifier für Instanz 4

```

Intercept 1.0729312450809303
Prediction_local [0.32117]
Right: 0.3831789850241079
CPU times: user 965 ms, sys: 255 ms, total: 1.22 s
Wall time: 918 ms
0.47499633409421527

```

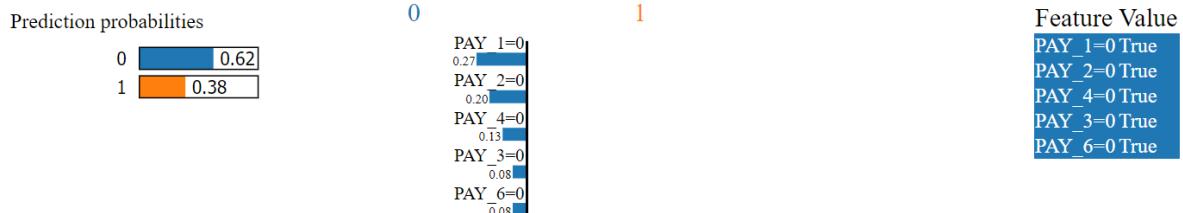


Abbildung 6.25: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 4

```

CPU times: user 44 s, sys: 632 ms, total: 44.6 s
Wall time: 41.1 s

```

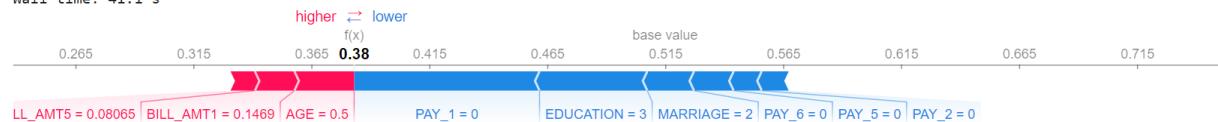


Abbildung 6.26: Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 4

tenkunde beispielsweise  $PAY\_1 = 0$  (pünktliche Zahlung im September 2005), so besteht eine hohe Wahrscheinlichkeit, dass dieser Kunde im nächsten Monat keinen Zahlungsausfall ( $Default\ Payment = 0$ ) haben wird. Umgekehrt gilt, fanden bei dem Kunden Verzögerungen bei früheren Zahlungen statt, z.B.  $PAY\_1 = 2$  (zwei Monate Zahlungsverzug im September 2005), so deutet dies auf einen Zahlungsausfall im nächsten Monat ( $Default\ Payment = 1$ ) hin. Die Erklärung sieht vernünftig aus, denn ein Kunde, der noch offene Zahlungen aus Vormonaten hat, wird auch im aktuellen Monat mit einer hohen Wahrscheinlichkeit die erforderliche Zahlung nicht ordnungsgemäß begleichen können.

### 6.3.2 Auswirkung von unterschiedlichen Kernel-Breiten auf den $R^2$ -Wert

Wie schon in Unterabschnitt 6.3.1 erwähnt, kann für die generierte LIME-Erklärung zusätzlich der dazugehörige  $R^2$ -Wert ausgegeben werden. Dabei handelt es sich um ein Gütemaß der linearen Regression und wird hier verwendet, um die lokale Treue des lokalen Ersatzmodells zu quantifizieren.  $R^2$ -Wert ist wie folgt definiert:

$$R^2 = \frac{\sum_{i=1}^N (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^N (y_i - \bar{y})^2} = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2},$$

wobei  $N$  die Anzahl der Stichproben,  $y_i$  die Werte der Black-Box-Modell-Vorhersage,  $\hat{y}_i$  die Werte des lokal angepassten linearen Modells und  $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$  der empirische Mittelwert sind.

Das bestmögliche Ergebnis ist  $R^2 = 1$ , d.h. die Vorhersagen des lokal interpretierbaren Modells stimmen perfekt mit den Vorhersagen des Black-Box-Modells überein. Ein Wert von  $R^2 = 0$  deutet darauf hin, dass das das lokal interpretierbare Modell genauso gut ist wie ein Modell, das immer den erwarteten Wert von  $y$  vorhersagt. Ist hingegen  $R^2 < 0$ , so schneidet das lokal interpretierbare Modell sogar schlechter ab als das Modell, das immer den erwarteten Wert von  $y$  vorhersagt [YA19].

Aus Tabelle 6.2 wird ersichtlich, dass die besten  $R^2$ -Werte vom Support Vector Machine Classifier stammen, gefolgt vom Multi-Layer Perceptron Classifier. Dabei hat der Random Forest Classifier im Schnitt die schlechtesten  $R^2$ -Werte. Der Grund dafür ist, dass die LIME-Erklärung auf lokalen linearen Modell basiert und lineare Annäherungen an komplexe nicht-lineare Black-Box-Modelle nicht immer zu geeigneten Erklärungen führen muss. Somit kann der Random Forest Classifier sogar in dieser lokalen Region eine nicht-lineare Entscheidungsgrenze haben. Es ist also nicht verwunderlich, dass der lineare Support Vector Machine Classifier die besten  $R^2$ -Werte erzielt. Um die Anpassungen lokaler zu machen, kann die Kernel-Breite verringert werden.

Black-Box Modell	Instanz 1	Instanz 2	Instanz 3	Instanz 4
Support Vector Machine Classifier	0,862	0,650	0,565	0,870
Random Forest Classifier	0,375	0,398	0,407	0,333
Multi-Layer Perceptron Classifier	0,440	0,524	0,474	0,475

Tabelle 6.2: Übersicht der  $R^2$ -Werte

Um das Vertrauen in die LIME-Erklärung der lokalen Region zu maximieren, wird nun der Einfluss von verschiedenen Kernel-Breiten  $\sigma$  auf die generierte Erklärung und somit auch auf  $R^2$ -Wert konkreter untersucht. Hierfür wird wiedereinmal die LIME-Erklärung von Multi-Layer Perceptron Classifier ausgewählt, wobei hier beispielhaft die zweite Instanz betrachtet wird. Die in Abbildung 6.27 - 6.31 dargestellten Ergebnisse heben die Merkmale hervor, die in erster Linie dafür verantwortlich waren, dass das Modell den Datenpunkt als Klasse 1 prognostiziert hat. Anscheinend waren es die Merkmale  $PAY\_1 = 2$  und  $LIMIT\_BAL <= 0,04$ , die den größten

Einfluss auf diese Entscheidung hatten. Im Gegensatz dazu deuten die Merkmale  $PAY\_2 = 0$ ,  $PAY\_4 = 0$  und  $PAY\_3 = 0$  auf eine Klassifikation als kein Zahlungsausfall. Bei allen Kernel-Breiten waren die Gewichte der Merkmale fast identisch. Nichtsdestotrotz gibt es eine Veränderung bei den  $R^2$ -Werten. Insbesondere ist erkennbar, dass sich die  $R^2$ -Werte bei jeder Verringerung der Kernel-Breite erhöht haben.

Bei den Kernel-Breiten  $\sigma = 5$  mit  $R^2 = 0,526$ ,  $\sigma = 4$  mit  $R^2 = 0,529$ ,  $\sigma = 3$  mit  $R^2 = 0,536$  und  $\sigma = 2$  mit  $R^2 = 0,558$  zeigt sich keine große Veränderung. Aber für  $\sigma = 1$  mit  $R^2 = 0,602$  ist eine Erhöhung des  $R^2$ -Wertes erkennbar. Somit kann das lineare Ersatzmodell das zugrunde liegende Verhalten des Multi-Layer Perceptron Classifier im Vergleich zu anderen Kernel-Breiten besser approximieren. In diesem Fall sieht es so aus, dass ein kleineres  $\sigma$  zu einem lokal besser passenden linearen Modell führt, was wiederum eine Verkleinerung der Konstante und der Koeffizienten mit sich bringt. Daher sollte bei der Wahl der Kernel-Breite aufgepasst werden, da auch durch andere Optimierungsmöglichkeiten, wie eine Veränderung der Distanzfunktion oder Regularisierung, eine bessere Anpassung erreicht werden kann.

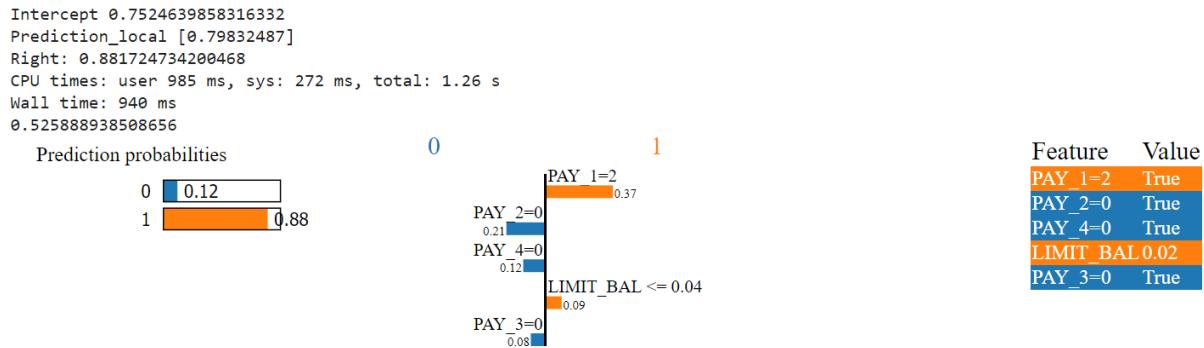


Abbildung 6.27: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit  $\sigma = 5$

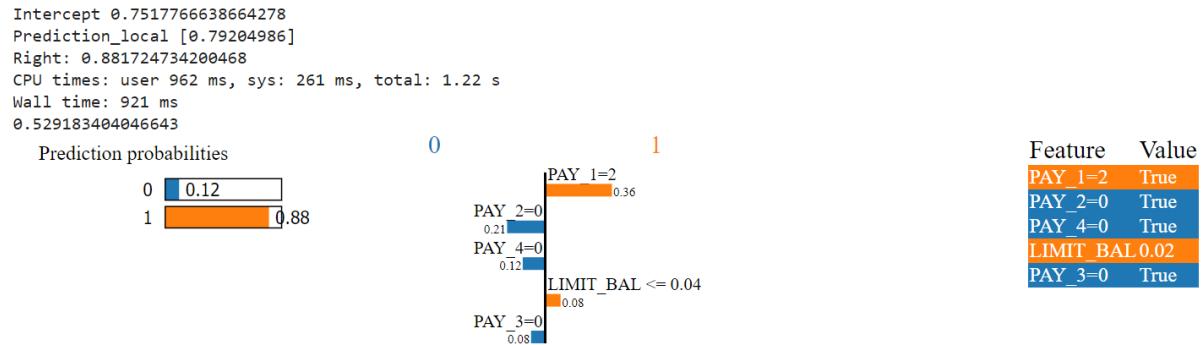


Abbildung 6.28: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit  $\sigma = 4$

### 6.3.3 Globale Erklärung mit LIME und Kernel SHAP

Abschließend werden die von LIME und Kernel SHAP generierten globale Erklärungen analysiert. Zunächst muss das SP-LIME-Modell initialisiert werden. Aufgrund der Rechenzeit werden statt dem ganzen Testdatensatz nur 20 Datenpunkte aus den Testdaten ausgewählt. Zudem soll am Ende fünf Merkmale anhand von zwei repräsentativen Stichproben die globale Erklärung der Black-Box-Modelle visualisiert werden.

```
%time sp_obj = submodular_pick.SubmodularPick(explainer_lime, X_test.values,
predict_svc, sample_size=20, num_features=5, num_exps_desired=2)
```

```

Intercept 0.744718161731869
Prediction_local [0.80177693]
Right: 0.881724734200468
CPU times: user 961 ms, sys: 258 ms, total: 1.22 s
Wall time: 915 ms
0.5362884852763443

```

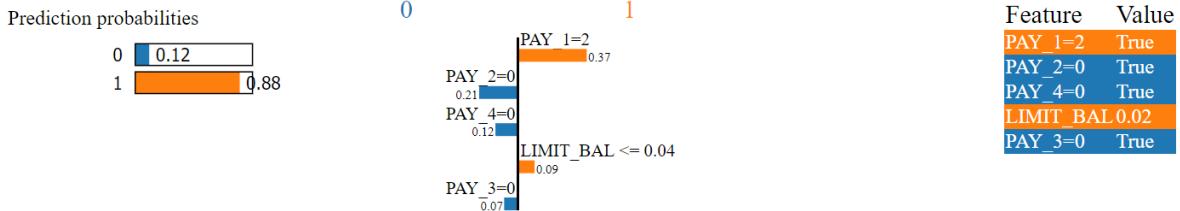


Abbildung 6.29: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit  $\sigma = 3$

```

Intercept 0.7452880577664628
Prediction_local [0.80482674]
Right: 0.881724734200468
CPU times: user 982 ms, sys: 252 ms, total: 1.23 s
Wall time: 944 ms
0.5581746440247689

```

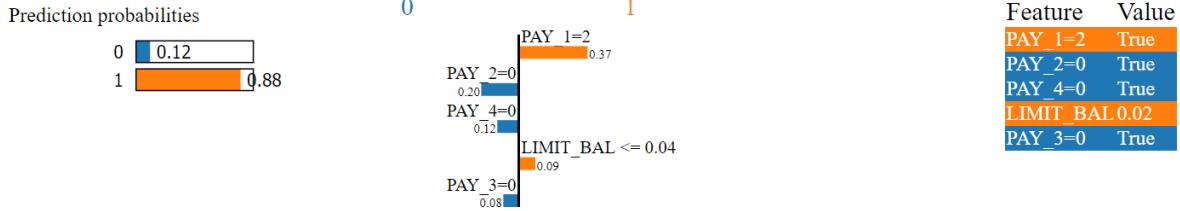


Abbildung 6.30: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit  $\sigma = 2$

```

Intercept 0.6136595605085354
Prediction_local [0.76491398]
Right: 0.881724734200468
CPU times: user 956 ms, sys: 264 ms, total: 1.22 s
Wall time: 921 ms
0.6020468732140188

```

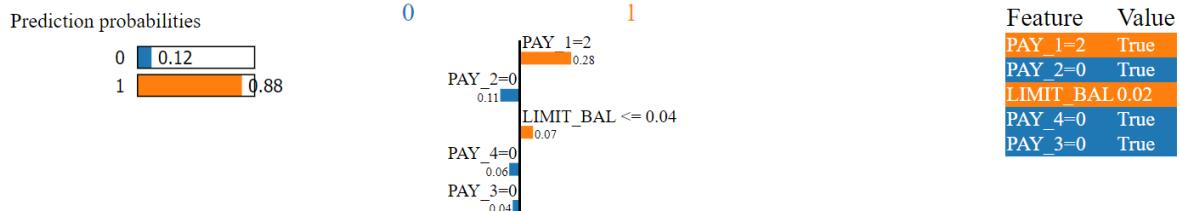


Abbildung 6.31: LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit  $\sigma = 1$

```
[ exp . as _ pyplot _ figure ( label=exp . available _ labels () [ 0 ] ) for exp
in sp_obj . sp_explanations ]
```

Wie bei lokalen muss auch bei globalen Kernel-SHAP-Erklärung auf die Verwendung eines großen Datensatzes verzichtet werden, da sich Kernel SHAP für größere Datensätze nicht berechnen lässt. Um hohe Rechenzeit des Algorithmus zu umgehen, wird nur eine Teilmenge des Datensatzes verwendet. Hierfür wird K-Means-Clustering als eine sinnvolle Möglichkeit zur Zusammenfassung und somit Reduzierung des Datensatzes eingesetzt [LSGH].

```
X_train_summary = shap.kmeans(X_train, 10)
explainer = shap.KernelExplainer(predict_mlp, X_train_summary)
%time shap_values = explainer.shap_values(X_test)
shap.initjs()
shap.summary_plot(shap_values, X_test)
```

In den Abbildungen 6.32 - 6.34 sind auf der linken Seite die globalen LIME-Erklärungen für Klasse 0 dargestellt, auf der rechten Seite für Klasse 1. Außerdem wird der positive Einfluss eines Merkmals mit grün, der negative mit rot visualisiert. Es wird ersichtlich, dass die zwei wichtigsten Merkmale  $PAY\_1 = 0$  und  $PAY\_2 = 0$  einen positiven Einfluss auf die Vorhersageergebnisse der Black-Box-Modelle für Klasse 0 haben. Für die Vorhersage von Klasse 1 sind hingegen  $PAY\_1 = 2$  und  $PAY\_2 = 2$  die einflussreichsten Merkmale, wobei der Random Forest Classifier zusätzlich die Merkmale  $PAY\_1 = 0$  und  $PAY\_2 = 0$  als gegen die Entscheidung für 1 wirkend identifiziert. Die vorgestellten Erklärungen von SP-LIME sehen sinnvoll aus und stimmen mit den Beobachtungen in 6.3.1 überein. Es sollte angemerkt werden, dass die Generierung von SP-LIME für den Support Vector Machine Classifier 1 Minute und 21 Sekunden dauerte, während bei Random Forest Classifier und Multi-Layer Perceptron Classifier nur 22 Sekunden Rechenzeit benötigt wurden.

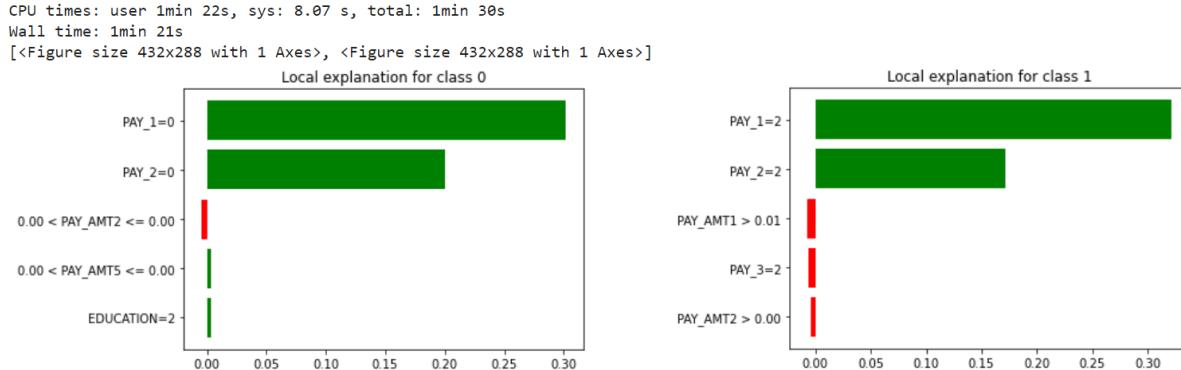


Abbildung 6.32: SP-LIME-Erklärung zum Support Vector Machine Classifier

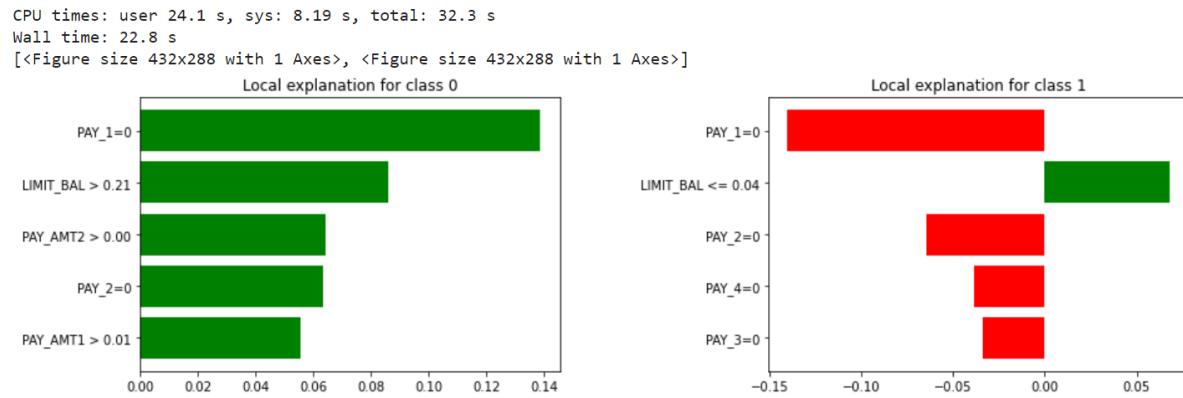


Abbildung 6.33: SP-LIME-Erklärung zum Random Forest Classifier

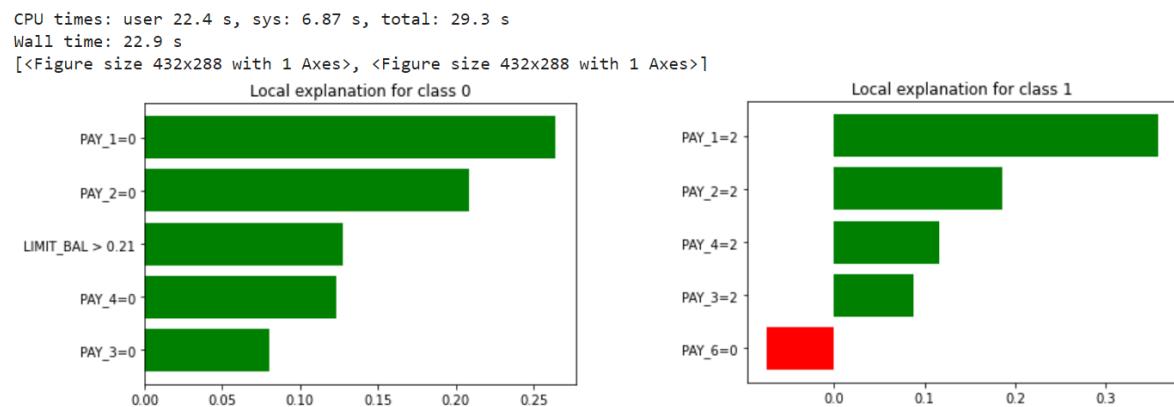


Abbildung 6.34: SP-LIME-Erklärung zum Multi-Layer Perceptron Classifier

Obwohl für die Erklärung von Kernel SHAP K-Means-Clustering verwendet wurde, dauerte die Generierung der globalen Erklärung zum Multi-Layer Perceptron Classifier über 49 Minuten (vgl. Abbildung 6.35). Für die Darstellung der Merkmalsrelevanz wird der Durchschnitt der SHAP-Werte jedes Merkmals über den Datensatz genommen und als ein Balkendiagramm dargestellt. Die Relevanz der Merkmale ist nach Klassen aufgeteilt, wobei rote Balken die Auswirkungen von Merkmalen auf Klasse 1 und die blauen Balken die Auswirkung auf Klasse 0 zeigen. Bei dieser Art der Visualisierung wird jedoch die Richtung des Einflusses nicht abgebildet. Wie zu erwarten, sind *PAY\_1*, *LIMIT\_BAL* und *PAY\_2* die wichtigsten Merkmale für die globale Erklärung des Multi-Layer Perceptron Classifier. Es fällt aber auf, dass das Merkmal *SEX* als eine der wichtigen Einflussgrößen aufzeichnet, was weder bei lokalen LIME- und Kernel-SHAP-Erklärungen noch bei globalen LIME-Erklärungen festgestellt wurde.

```
CPU times: user 30min 48s, sys: 18min 35s, total: 49min 23s
Wall time: 26min 35s
```

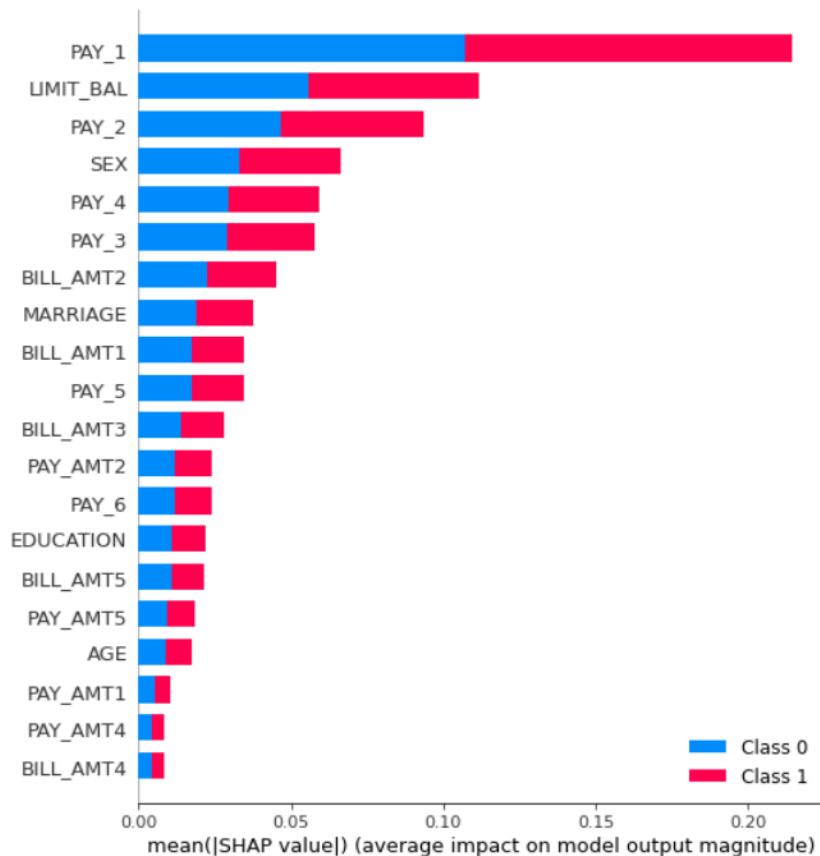


Abbildung 6.35: Globale Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier

## 7 Zusammenfassung und Ausblick

---

Das Ziel dieser wissenschaftlichen Arbeit war es, die Erklärungsmodelle LIME und Kernel SHAP detailliert vorzustellen und die generierten Erklärungen zu untersuchen und miteinander zu vergleichen. Um dies zu erreichen, wurden zunächst grundlegende Begriffe zum Machine Learning in Bezug auf Klassifikationsprobleme erläutert und ausgewählte Lernalgorithmen beschrieben. Anschließend wurde die Notwendigkeit von erklärbarer Künstlicher Intelligenz mit Beispielen wie Algorithmen für die Urteilsfindung in US-Gerichten oder der Frage der Verantwortlichkeit bei autonomem Fahren erörtert. Dadurch konnte ein besseres Verständnis der Anforderungen an transparente oder zumindest erklärbare KI-Systeme vermittelt werden. Nachfolgend wurde das Konzept der erklärbaren Künstlichen Intelligenz ausführlich ausgearbeitet. Mit dem Einsatz von Erklärungsmodellen können mögliche Diskriminierungen, Scheinkausalitäten oder algorithmische Voreingenommenheiten aufgedeckt werden, die durch intransparente Entscheidungen der KI-Systeme hervorgerufen werden. Es gibt zwar eine Vielzahl an Erklärungstechniken, aber hier liegt der Schwerpunkt auf modellagnostischen Post-hoc-Erklärungsmodellen. Zu diesem Zweck wurden, nach Betrachtung der erforderlichen Grundlagen, die Vorgehensweisen der zwei bekanntesten modellagnostischen Post-hoc-Erklärungsmodelle vorgestellt. Hierzu werden Black-Box-Modelle um Erklärungsmodelle erweitert, um die Prognose des Vorhersagemodells zu begründen und so die Nachvollziehbarkeit der Entscheidungsfindung zu gewährleisten. Dabei kann der Umfang der Interpretierbarkeit der Erklärungsmodelle von global bis lokal variieren, d.h. entweder das gesamte Modellverhalten oder nur einzelne Vorhersagen werden erklärt. Im Rahmen dieser Arbeit wurden die Erklärungsmodelle auf die Ergebnisse der drei eingesetzten Black-Box-Modelle für den Datensatz Default Of Credit Card Clients angewendet. Beide Methoden haben ähnliche Merkmale als entscheidende Einflussfaktoren für die Entscheidung der Vorhersagemodelle bestimmt.

Die Erklärungsmodelle haben Weiterentwicklungspotential, und demgemäß müssen für das Ziel der vollständigen Erklärbarkeit von KI-Entscheidungen weitere Forschung und Entwicklung in diesem Bereich gefördert werden. Als möglicher Ansatzpunkt für zukünftige Forschungen kann beispielsweise die Quantifizierung der Leistung von Erklärungsmodellen betrachtet werden. Die Motivation ist dabei, von der menschlichen Bewertung der generierten Erklärungen zur automatischen Bewertung mittels Metrik zu wechseln. Hierfür wurden beispielsweise bereits zwei Evaluationsmetriken in Bezug auf die Stabilität vorgestellt [[AJ18], [VB20]]. Nichtsdestotrotz werden allgemeine XAI-Metriken benötigt, mit denen die Evaluation der unterschiedlichen Erklärungsmodelle stattfinden kann. Erst nachdem die Leistungsfähigkeit der Erklärungsmodelle bewertet werden kann, ist eine Kontrolle von KI-Entscheidungen möglich und kann in der Folge Vertrauen in KI-Anwendungen aufgebaut werden.

## Literaturverzeichnis

---

- [AB18] Adadi, A. und Berrada, M.: *Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)*. In: IEEE Access, Volume 6, 2018.
- [AC19] Ancona, M., Ceolini, E., Öztireli, C. und Gross, M.: *Gradient-Based Attribution Methods*. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Springer Nature Switzerland, Cham, 2019.
- [AD19] Arrieta, A. B., Diaz-Rodriguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R. und Herrera, F.: *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI*. ArXiv preprint arXiv:1910.10045, 2019.
- [ADRM] Appen Disaster Response Messages. Abgerufen am 13.06.2020 von <https://appen.com/resources/datasets/>.
- [AE18] Ameisen, E. (31.01.2018): *How to solve 90% of NLP problems*. Abgerufen am 14.06.2020 von <https://www.oreilly.com/content/how-to-solve-90-of-nlp-problems-a-step-by-step-guide/>
- [AJ18] Alvarez-Melis, D. und Jaakkola, T. S.: *On the Robustness of Interpretability Methods*. In: ICML Workshop on Human Interpretability in Machine Learning, 2018.
- [AK20] Antonov, A. und Kerikmäe, T.: *Trustworthy AI as a Future Driver for Competitiveness and Social Change in the EU*. In: The EU in the 21st Century, Springer International Verlag, 2020.
- [AL16] Angwin, J., Larson, J., Mattu, S. und Kirchner, L. (23.05.2016): *Machine Bias - There's software used across the country to predict future criminals. And it's biased against blacks..* Abgerufen am 03.05.2020 von <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>.
- [AN19] Ahern, I., Noack, A., Guzman-Nateras, L., Dou, D., Li, B. und Huan, J.: *NormLime: A New Feature Importance Metric for Explaining Deep Neural Networks*. ArXiv preprint arXiv:1909.04200, 2019.
- [AS15] Aldeen, Y. A. A. S., Salleh, M und Razzaque, M. A.: *A comprehensive review on privacy preserving data mining*. In SpringerPlus, Volume 4:694, 2015.
- [BA19] Burkov, A.: *Machine Learning kompakt - Alles, was Sie wissen müssen*. Mitp Verlag, Frechen, 2019.
- [BB19] Bitkom (2019): *Blick in die Blackbox - Nachvollziehbarkeit von KI-Algorithmen in der Praxis*. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., abgerufen am 14.05.2020 von [https://www.bitkom.org/sites/default/files/2019-10/20191016\\_blick-in-die-blackbox.pdf](https://www.bitkom.org/sites/default/files/2019-10/20191016_blick-in-die-blackbox.pdf).

- [BC20] Bhargava, V., Couceiro, M. und Napoli, A.: *LimeOut: An Ensemble Approach To Improve Process Fairness*. In: ECML PKDD International Workshop on eXplainable Knowledge Discovery in Data Mining, 2020.
- [BF18] Bundesanstalt für Finanzdienstleistungsaufsicht (15.06.2018): *Big Data trifft auf künstliche Intelligenz*. Abgerufen am 25.05.2020 von [https://www.bafin.de/SharedDocs/Downloads/DE/dl\\_b dai\\_studie.html;jsessionid=161E5ADB56AF4D36B0F085B2C867822F.2\\_cid390?nn=9021442](https://www.bafin.de/SharedDocs/Downloads/DE/dl_b dai_studie.html;jsessionid=161E5ADB56AF4D36B0F085B2C867822F.2_cid390?nn=9021442).
- [BP16] Beuth, P. (24.03.2016): *Twitter-Nutzer machen Chatbot zur Rassistin*. Abgerufen am 18.04.2020 von <https://www.zeit.de/digital/internet/2016-03/microsoft-tay-chatbot-twitter-rassistisch>.
- [BR18] Die Bundesregierung (November 2018): *Strategie Künstliche Intelligenz der Bundesregierung*. Abgerufen am 14.05.2020 von [https://www.bmbf.de/files/Nationale\\_KI-Strategie.pdf](https://www.bmbf.de/files/Nationale_KI-Strategie.pdf).
- [BS10] Baehrens, D., Schroeter, T., Harmeling, S., Kawabe, M., Hansen, K., und Müller, K.-R.: *How to Explain Individual Classification Decisions*. In: Journal of Machine Learning Research, Volume 11, 2010.
- [BX20] Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J. M. F. und Eckersley, P.: *Explainable Machine Learning in Deployment*. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 2020.
- [CB02] Chawla, N., Bowyer, K., Hall, L. und Kegelmeyer, P.: *SMOTE: Synthetic Minority Over-sampling Technique*. In: Journal of Artificial Intelligence Research, Volume 16, AI Access Foundation and Morgan Kaufmann Verlag, 2002.
- [CT88] Chandrasekaran, B., Tanner, M. C. und Josephson, J. R.: *Explanation: the Role of Control Strategies and Deep Models*. In: Expert Systems: The User Interface, Ablex Verlag, 1988.
- [CT89] Chandrasekaran, B., Tanner, M. C. und Josephson, J. R.: *Explaining Control Strategies in Problem Solving*. In: IEEE Expert, Volume 4, 1989.
- [CP18] Choudhary, P. (22.03.2018): *Interpreting predictive models with Skater: Unboxing model opacity*. Abgerufen am 01.06.2020 von <https://www.oreilly.com/content/interpreting-predictive-models-with-skater-unboxing-model-opacity/>.
- [CP19] Carvalho, D. V., Pereira, E. M. und Cardoso, J. S.: *Machine Learning Interpretability: A Survey on Methods and Metrics*. In: Electronics, Volume 8, 2019.
- [DB18] Deutscher Bundestag (22.05.2018): *Autonomes und automatisiertes Fahren auf der Straße - rechtlicher Rahmen*. Abgerufen am 01.05.2020 von <https://www.bundestag.de/resource/blob/562790/c12af1873384bcd1f8604334f97ee4b9/wd-7-111-18-pdf-data.pdf>.
- [DE19] Datenethikkommission der Bundesregierung (23.10.2019): *Gutachten der Datenethikkommission*. Abgerufen am 13.05.2020 von [https://www.bmjjv.de/SharedDocs/Downloads/DE/Themen/Fokusthemen/Gutachten\\_DEK\\_DE.pdf?\\_\\_blob=publicationFile&v=5](https://www.bmjjv.de/SharedDocs/Downloads/DE/Themen/Fokusthemen/Gutachten_DEK_DE.pdf?__blob=publicationFile&v=5).
- [DK17] Doshi-Velez, F. und Kim, B.: *Towards A Rigorous Science of Interpretable Machine Learning*. ArXiv preprint arXiv:1702.08608, 2017.
- [DL18] Du, M., Liu, N. und Hu, X.: *Techniques for Interpretable Machine Learning*. ArXiv preprint arXiv:1808.00033, 2018.

- [DS18] Dreyer, S. und Schulz, W. (April 2018): *Was bringt die Datenschutz-Grundverordnung für automatisierte Entscheidungssysteme?*. Abgerufen am 26.05.2020 von [https://www.bertelsmann-stiftung.de/fileadmin/files/BSt/Publikationen/GrauePublikationen/BSt\\_DSGVOundADM\\_dt.pdf](https://www.bertelsmann-stiftung.de/fileadmin/files/BSt/Publikationen/GrauePublikationen/BSt_DSGVOundADM_dt.pdf).
- [DV19] Dignum, V.: *Responsible Artificial Intelligence - How to Develop and Use AI in a Responsible Way*. Springer Nature Switzerland, Cham, 2019.
- [EK17] Ethik-Kommission (Juni 2017): *Automatisiertes und Vernetztes Fahren*. Abgerufen am 03.05.2020 von [https://www.bmvi.de/SharedDocs/DE/Publikationen/DG/bericht-der-ethik-kommission.pdf?\\_\\_blob=publicationFile](https://www.bmvi.de/SharedDocs/DE/Publikationen/DG/bericht-der-ethik-kommission.pdf?__blob=publicationFile).
- [EK19a] Europäische Kommission (08.11.2019): *Ethik-leitlinien für eine vertrauenswürdige KI*. Abgerufen am 28.05.2020 von <https://op.europa.eu/de/publication-detail/-/publication/d3988569-0434-11ea-8c1f-01aa75ed71a1/language-de>.
- [EK19b] Europäische Kommission (26.06.2019): *Neue ethische Leitlinien für Künstliche Intelligenz vorgelegt*. Abgerufen am 01.05.2020 von [https://ec.europa.eu/germany/news/20190626-ethische-leitlinien-fuer-kuenstliche-intelligenz-vorgelegt\\_de](https://ec.europa.eu/germany/news/20190626-ethische-leitlinien-fuer-kuenstliche-intelligenz-vorgelegt_de).
- [ES19] El Shawi, R., Sherif, Y., Al-Mallah, M., Sakr, S.: *ILIME: Local and Global Interpretable Model-Agnostic Explainer of Black-Box Decision*. In: Advances in Databases and Information Systems, Springer Nature Switzerland, 2019.
- [EU16] Europäischen Union (04.05.2016): *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)*. Abgerufen am 28.05.2020 von <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679>.
- [EW16] Ertel, W.: *Grundkurs Künstliche Intelligenz - Eine praxisorientierte Einführung*. Springer Vieweg, 4. Auflage, Wiesbaden, 2016.
- [FG18] Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (2018): *Maschinelles Lernen - Eine Analyse zu Kompetenzen, Forschung und Anwendung*. Abgerufen am 16.05.2020 von [https://www.bigdata.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/Fraunhofer\\_Studie\\_ML\\_201809.pdf](https://www.bigdata.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/Fraunhofer_Studie_ML_201809.pdf).
- [GB19] Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa A., Specter, M. und Kagal, L.: *Explaining Explanations: An Overview of Interpretability of Machine Learning*. ArXiv preprint arXiv:1806.00069, 2019.
- [GD16] Gunning, D.: *Broad Agency Announcement - Explainable Artificial Intelligence (XAI)*. Defense Advanced Research Projects Agency (DARPA), abgerufen am 28.04.2020 von <https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf>.
- [GL20] Garreau, D. und Von Luxburg, U.: *Explaining the Explainer: A First Theoretical Analysis of LIME*. ArXiv preprint arXiv:2001.03447, 2020.
- [GM18] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F. und Pedreschi, D.: *A Survey of Methods for Explaining Black Box Models*. In: ACM Computing Surveys, Volume 51, 2018.

- [HA18] Holzinger, A.: *Interpretierbare KI - Neue Methoden zeigen Entscheidungswege künstlicher Intelligenz auf*. Heise Medien, Heft 22, 2018.
- [HG19] Hall, P., Gill, N.: *An Introduction to Machine Learning Interpretability - An Applied Perspective on Fairness, Accountability, Transparency, and Explainable AI*. O'Reilly Media, 2. Auflage, 2019.
- [HK70] Hoerl, A. E. und Kennard, R. W.: *Ridge regression: Biased estimation for nonorthogonal problems*. In: *Technometrics*, Volume 12, 1970.
- [HL19] Holzinger, A., Langs, G., Denk, H., Zatloukal, K. und Müller, H.: *Causability and explainability of artificial intelligence in medicine*. In: Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Volume 9(4), 2019.
- [HP18] Hall, P.: *On the Art and Science of Machine Learning Explanations*. ArXiv preprint arXiv:1810.02909, 2018.
- [HR19] Hansen, L. K und Rieger, L.: *Interpretability in Intelligent Systems – A New Concept?*. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Springer Nature Switzerland, Cham, 2019.
- [HS18] Hedderich, J. und Sachs, L.: *Angewandte Statistik - Methodensammlung mit R*. Springer Spektrum, 16. Auflage 2018.
- [HT09] Hastie, T., Tibshirani, R. und Friedman, J.: *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer Science+Business Media, 2. Auflage, New York, 2009.
- [HU92] Hoppe, U.: *Einsatz von Hypertext/Hypermedia zur Verbesserung der Erklärungsfähigkeit Wissensbasierter Systeme*. In Wissensbasierte Systeme in der Wirtschaft 1992 - Anwendungen und Integration mit Hypermedia, Springer Gabler, 1992.
- [HY20] Huang, Q., Yamada, M., Tian, Y., Singh, D., Yin, D. und Chang, Y.: *GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks*. ArXiv preprint arXiv:2001.06216, 2020.
- [IE20] Islam, S. R., Eberle, W., Ghafoor, S. K., Siraj, A. und Rogers, M.: *Domain Knowledge Aided Explainable Artificial Intelligence for Intrusion Detection and Response*. ArXiv preprint arXiv:1911.09853, 2020.
- [JS17] Jaume-Palasi, L. und Spielkamp, M. (2017): *Ethik und algorithmische Prozesse zur Entscheidungsfindung oder -vorbereitung*. Abgerufen am 21.05.2020 von [https://algorithmwatch.org/wp-content/uploads/2017/06/AlgorithmWatch\\_Arbeitspapier\\_4\\_Ethik\\_und\\_Algorithmen.pdf](https://algorithmwatch.org/wp-content/uploads/2017/06/AlgorithmWatch_Arbeitspapier_4_Ethik_und_Algorithmen.pdf).
- [KB96] Kohavi, R. und Becker, B.: *Adult Data Set*. Abgerufen am 28.07.2020 von <http://archive.ics.uci.edu/ml/datasets/Adult>.
- [KC16] Köllner, C. (20.10.2016): *Wie Autos das Denken lernen*. Abgerufen am 23.04.2020 von <https://www.springerprofessional.de/automatisiertes-fahren/car-to-x/wie-autos-das-denken-lernen/10915232>.
- [KDDT] Kaggle: Real or Not? NLP with Disaster Tweets. Abgerufen am 13.06.2020 von <https://www.kaggle.com/c/nlp-getting-started>.

- [KS19] Kreutzer, R. T. und Sirrenberg, M.: *Künstliche Intelligenz verstehen. Grundlagen - Use-Cases - unternehmenseigene KI-Journey*. Springer Gabler, 2019.
- [KU20] Kovalev, M. S., Utkin, L. V. und Kasimov, E. M.: *SurvLIME: A method for explaining machine learning survival models*. ArXiv preprint arXiv:2003.08371, 2020.
- [LE18] Lundberg, S. M., Erion, G. G. und Lee, S.-I.: *Consistent Individualized Feature Attribution for Tree Ensembles*. ArXiv preprint arXiv:1802.03888, 2018.
- [LF19] Li, H., Fan, W., Shi, S. und Chou, Q.: *A Modified LIME and Its Application to Explain Service Supply Chain Forecasting*. In: Natural Language Processing and Chinese Computing, 2019.
- [LL16] Lundberg, S. M. und Lee, S.-I.: *An unexpected unity among methods for interpreting model predictions*. ArXiv preprint arXiv: 1611.07478, 2016.
- [LL17] Lundberg, S. M. und Lee, S.-I.: *A unified approach to interpreting model predictions*. In: Advances in Neural Information Processing Systems, 2017.
- [LR18] Laugel, T., Renard, X., Lesot, M. J., Marsala, C. und Detyniecki, M.: *Defining Locality for Surrogates in Post-hoc Interpretability*. In: ICML Workshop on Human Interpretability in Machine Learning, 2018.
- [LSGH] Lundberg, S. M.: GitHub-Repository. Abgerufen am 25.08.2020 von <https://github.com/slundberg/shap>.
- [LW19] Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W. und Müller, K.-R.: *Unmasking Clever Hans predictors and assessing what machines really learn*. In: Nature communications, Nature Group Verlag, 2019.
- [LZ16] Lipton, Z. C.: *The Mythos of Model Interpretability*. In: ICML Workshop on Human Interpretability in Machine Learning, 2016.
- [MC20] Molnar, C.: *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. Abgerufen am 20.08.2020 von <https://christophm.github.io/interpretable-ml-book/>.
- [MK19] Mueller, S. T., Hoffman, R. R., Clancey, W., Emrey, A. und Klein, G.: *Explanation in Human-AI Systems: A Literature Meta-Review Synopsis of Key Ideas and Publications and Bibliography for Explainable AI*. ArXiv preprint arXiv:1902.01876, 2019.
- [MM19] Martini, M.: *Blackbox Algorithmus – Grundfragen einer Regulierung Künstlicher Intelligenz*. Springer Verlag, 2019.
- [MS17] Mishra, S., Sturm, B. L. und Dixon, S.: *Local Interpretable Model-Agnostic Explanations for Music Content Analysis*. In: International Society for Music Information Retrieval Conference, 2017.
- [MV11] Müller-Benedict, V.: *Grundkurs Statistik in den Sozialwissenschaften - Eine leicht verständliche, anwendungsorientierte Einführung in das sozialwissenschaftlich notwendige statistische Wissen*. VS Verlag, 5. Auflage, 2011
- [NI11] Northpointe Inc. (2011): *Sample COMPAS Risk Assessment*. Abgerufen am 18.05.2020 von <https://www.documentcloud.org/documents/2702103-Sample-Risk-Assessment-COMPAS-CORE.html>.

- [NS19] Nassar, M., Salah, K., Habib ur Rehman, M. und Svetinovic, D.: *Blockchain for explainable and trustworthy artificial intelligence*. In: Wiley Interdisciplinary Reviews Data Mining and Knowledge Discovery, Volume 10, 2019.
- [PT18] Peltola, T.: *Local Interpretable Model-agnostic Explanations of Bayesian Predictive Models via Kullback-Leibler Projections*. ArXiv preprint arXiv:1810.02678, 2018.
- [PW19] Papp, S., Weidinger, W., Meir-Huber, M., Ortner, B., Georg Langs, G. und Wazir, R.: *Handbuch Data Science - Mit Datenanalyse und Machine Learning Wert aus Daten generieren*. Carl Hanser Verlag, München, 2019.
- [RB20] Roscher, R., Bohn, B., Duarte, M. F. und Garske, J.: *Explainable Machine Learning for Scientific Insights and Discoveries*. In: IEEE Access, Volume 8, 2020.
- [RD20] Rabold, J., Deininger, H., Siebers, M. und Schmid, U.: *Enriching Visual with Verbal Explanations for Relational Concepts – Combining LIME with Aleph*. Machine Learning and Knowledge Discovery in Databases, 2020.
- [RM03] Ren, X. und Malik, J.: *Learning a classification model for segmentation*. In: IEEE International Conference on Computer Vision, 2003.
- [RMBG] Ribeiro, M. T. (20.07.2015): *The greedy algorithm for monotone submodular maximization*. Abgerufen am 24.05.2020 von <https://homes.cs.washington.edu/~marcotcr/blog/greedy-submodular/>.
- [RMGH] Ribeiro, M. T.: GitHub-Repository. Abgerufen am 20.08.2020 von <https://github.com/marcotcr/lime>.
- [RR14] Romei, A. und Ruggieri, S.: *A multidisciplinary survey on discrimination analysis*. In: The Knowledge Engineering Review, Cambridge University Press, 2014.
- [RS16a] Ribeiro, M. T., Singh, S. und Guestrin, C.: „*Why Should I Trust You?*“ - *Explaining the Predictions of Any Classifier*. In: Knowledge Discovery and Data Mining, 2016.
- [RS16b] Ribeiro, M. T., Singh, S. und Guestrin, C. (12.08.2016): *Local Interpretable Model-Agnostic Explanations (LIME): An Introduction*. Abgerufen am 09.06.2020 von <https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/>.
- [RS18] Ribeiro, M. T., Singh, S. und Guestrin, C.: *Anchors: High-Precision Model-Agnostic Explanations*. In: Association for the Advancement of Artificial Intelligence, 2018.
- [RY19] Rasouli, P. und Yu, I. C.: *Meaningful Data Sampling for a Faithful Local Explanation Method*. In: Intelligent Data Engineering and Automated Learning, Springer International Verlag, 2019.
- [SD20] Shi, S., Du, Y und Fan, W.: *An Extension of LIME with Improvement of Interpretability and Fidelity*. ArXiv preprint arXiv:2004.12277, 2020.
- [SF19] Shakerin, F.: *Induction of Non-monotonic Logic Programs To Explain Statistical Learning Models*. In: International Conference on Logic Programming, 2019.
- [SF20a] Sokol, K. und Flach, P.: *Explainability Fact Sheets: A Framework for Systematic Assessment of Explainable Approaches*. ArXiv preprint arXiv:1912.05100, 2020.
- [SF20b] Sokol, K. und Flach, P.: *LIMEtree: Interactively Customisable Explanations Based on Local Surrogate Multi-output Regression Trees*. ArXiv preprint arXiv:2005.01427, 2020.

- [SG18] Shakerin, F. und Gupta, G.: *Induction of Non-Monotonic Logic Programs to Explain Boosted Tree Models Using LIME*. ArXiv preprint arXiv:1808.00629, 2018.
- [SH18] Stiffler, M., Hudler, A., Lee, E., Braines, D., Mott, D. und Harborne, D.: *An Analysis of Reliability Using LIME with Deep Learning Models*. In: Annual Fall Meeting of the Distributed Analytics and Information Science International Technology Alliance, AFM DAIS ITA, 2018.
- [SH19] Sokol, K., Hepburn, A., Santos-Rodriguez, R. und Flach, P.: *bLIMEy: Surrogate Prediction Explanations Beyond LIME*. ArXiv preprint arXiv:1910.13016, 2019.
- [SK10] Štrumbelj, E. und Kononenko, I.: *An Efficient Explanation of Individual Classifications using Game Theory*. In: Journal of Machine Learning Research, Volume 11, 2010.
- [SK14] Štrumbelj, E. und Kononenko, I.: *Explaining prediction models and individual predictions with feature contributions*. In: Knowledge and Information Systems, 2014.
- [SL53] Shapley, L. S.: *A Value for n-person Games*. In: Contributions to the Theory of Games, Volume 2, Princeton University Press, 1953.
- [SL15] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. und Rabinovich, A.: *Going Deeper with Convolutions*. In: IEEE Conference on Computer Vision and Pattern Recognition, 2015.
- [SM19] Samek, W. und Müller, K.-R.: *Towards Explainable Artificial Intelligence*. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Springer Nature Switzerland, Cham, 2019.
- [SM93] Swartout, W. R. und Moore, J. D.: *Explanation in Second Generation Expert Systems*. In: Second Generation Expert Systems, Springer Verlag, 1993.
- [SV13] Simonyan, K., Vedaldi, A. und Zisserman, A.: *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. ArXiv preprint arXiv:1312.6034, 2013.
- [SW19] Stubbe, J., Wessels, J. und, Zinke, G.: *Neue Intelligenz, neue Ethik?*. In: iit-Themenband: Künstliche Intelligenz, Springer Vieweg, 2019.
- [SW83] Swartout, W. R.: *XPLAIN: a System for Creating and Explaining Expert Consulting Programs*. In: Artificial Intelligence, Volume 21, 1983.
- [SW85] Swartout, W. R.: *Knowledge Needed for Expert System Explanation*. In: Proceedings of the National Computer Conference, 1985.
- [SZ19] Shi, S., Zhang, X. und Fan, W.: *Explaining the Predictions of Any Image Classifier via Decision Trees*. ArXiv preprint arXiv:1911.01058, 2019.
- [SZ20] Shi, S., Zhang, X. und Fan, W.: *A Modified Perturbed Sampling Method for Local Interpretable Model-agnostic Explanation*. ArXiv preprint arXiv:2002.07434, 2020.
- [TR96] Tibshirani, R.: *Regression Shrinkage and Selection Via the Lasso*. In: Journal of the Royal Statistical Society, Series B, 1996.
- [VB20] Visani, G., Bagli, E., Chesani, F., Poluzzi, A. und Capuzzo, D.: *Statistical stability indices for LIME: obtaining reliable explanations for Machine Learning models*. ArXiv preprint arXiv:2001.11757, 2020.

- [VF04] Van Lent, M., Fisher, W. C. und Mancuso, M.: *An Explainable Artificial Intelligence System for Small-unit Tactical Behavior*. In: Proceedings of the 16th conference on Innovative Applications of Artificial Intelligence, 2004.
- [VH19] Van der Linden, I., Haned, H. und Kanoulas, E.: *Global Aggregations of Local Explanations for Black Box models*. ArXiv preprint arXiv:1907.03039, 2019.
- [VM15] Vezzetti, E. und Marcolin, F.: *Similarity Measures for Face Recognition*. Bentham Science Verlag, 2015.
- [VS08] Vedaldi, A. und Soatto, S.: *Quick Shift and Kernel Methods for Mode Seeking*. In: European Conference on Computer Vision, 2008.
- [WA19] Weller, A.: *Transparency - Motivations and Challenges*. In: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Springer Nature Switzerland, Cham, 2019.
- [WH05] Wiese, H.: *Kooperative Spieltheorie*. De Gruyter Oldenbourg Verlag, 2005.
- [YA19] Yoon, J., Arik, S. Ö. und Pfister, T.: *RL-LIM: Reinforcement Learning-based Locally Interpretable Modeling*. ArXiv preprint arXiv:1909.12367, 2019.
- [ZK19] Zafar, M. R. und Khan, N. M.: *DLIME: A Deterministic Local Interpretable Model-Agnostic Explanations Approach for Computer-Aided Diagnosis Systems*. ArXiv preprint arXiv:1906.10263, 2019.
- [ZS19] Zhang, Y., Song, K., Sun, Y., Tan, S. und Udell, M.: „*Why Should You Trust My Explanation?“ Understanding Uncertainty in LIME Explanations*. In: International Conference on Machine Learning, 2019.

# Abbildungsverzeichnis

1.1	Konzept der erklärbaren Künstlichen Intelligenz [HA18] . . . . .	4
3.1	Der Kompromiss zwischen Genauigkeit und Erklärbarkeit [GD16] . . . . .	12
3.2	Globale und lokale Modellinterpretierbarkeit [CP18] . . . . .	20
4.1	Generierte LIME-Erklärung einer einzelnen Modellvorhersage [HA18] . . . . .	24
4.2	Originaldarstellung und interpretierbare Repräsentation der Textdaten . . . . .	25
4.3	Transformation eines Bildes in eine interpretierbare Datendarstellung [RS16b] . .	26
4.4	Zuordnung der Tabellendaten von originaler zu interpretierbarer Datendarstellung	28
4.5	Stichprobenerhebung der Textdaten . . . . .	32
4.6	LIME-Erklärung eines Textklassifizierungsmodells [AE18] . . . . .	32
4.7	LIME-Erklärung zur Bildklassifizierung [RS16b] . . . . .	33
4.8	LIME-Erklärungen für die drei vorhergesagten Klassen zur Bildklassifizierung [RS16b]	34
4.9	Stichprobenerhebung in diskretisierter Darstellung . . . . .	35
4.10	Ermittlung der Klassenzugehörigkeit mittels originaler Darstellung der Stichproben	35
4.11	Berechnung der Distanz anhand der interpretierbaren Stichproben . . . . .	36
4.12	LIME-Algorithmus für tabellarische Daten [MC20] . . . . .	37
4.13	LIME-Erklärung für Tabellendaten [RMGH] . . . . .	37
4.14	Musterbeispiel zum Submodular-Pick-Algorithmus . . . . .	40
4.15	Kernel-Breiten im Vergleich [BC20] . . . . .	41
5.1	Schematische Darstellung der SHAP-Werte [LL17] . . . . .	46
6.1	Anzahl Zahlungsausfall nach Geschlecht . . . . .	50
6.2	Anzahl Zahlungsausfall nach Familienstand . . . . .	51
6.3	Anzahl Zahlungsausfall nach Bildungsniveau . . . . .	51
6.4	Anzahl Zahlungsausfall nach der monatlichen Rückzahlungsstatus . . . . .	52
6.5	Korrelationsmatrix . . . . .	53
6.6	Wahrheitsmatrix und Klassifizierungsmetriken für Support Vector Machine Classifier	54
6.7	Wahrheitsmatrix und Klassifizierungsmetriken für Random Forest Classifier . . .	55
6.8	Wahrheitsmatrix und Klassifizierungsmetriken für Multi-Layer Perceptron Classifier	55
6.9	ROC-Kurven . . . . .	56
6.10	LIME-Erklärung zum Support Vector Machine Classifier für Instanz 1 . . . . .	58
6.11	LIME-Erklärung zum Random Forest Classifier für Instanz 1 . . . . .	59
6.12	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1 . . . . .	59
6.13	Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1 . .	59
6.14	Ausgabe der Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 1 . . . . .	60
6.15	LIME-Erklärung zum Support Vector Machine Classifier für Instanz 2 . . . . .	60
6.16	LIME-Erklärung zum Random Forest Classifier für Instanz 2 . . . . .	61
6.17	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 . . . . .	61
6.18	Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 . . .	61
6.19	LIME-Erklärung zum Support Vector Machine Classifier für Instanz 3 . . . . .	62

6.20	LIME-Erklärung zum Random Forest Classifier für Instanz 3 . . . . .	62
6.21	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 3 . . . . .	63
6.22	Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 3 . . .	63
6.23	LIME-Erklärung zum Support Vector Machine Classifier für Instanz 4 . . . . .	64
6.24	LIME-Erklärung zum Random Forest Classifier für Instanz 4 . . . . .	64
6.25	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 4 . . . . .	64
6.26	Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 4 . . .	64
6.27	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit $\sigma = 5$ .	66
6.28	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit $\sigma = 4$ .	66
6.29	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit $\sigma = 3$ .	67
6.30	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit $\sigma = 2$ .	67
6.31	LIME-Erklärung zum Multi-Layer Perceptron Classifier für Instanz 2 mit $\sigma = 1$ .	67
6.32	SP-LIME-Erklärung zum Support Vector Machine Classifier . . . . .	68
6.33	SP-LIME-Erklärung zum Random Forest Classifier . . . . .	69
6.34	SP-LIME-Erklärung zum Multi-Layer Perceptron Classifier . . . . .	69
6.35	Globale Kernel-SHAP-Erklärung zum Multi-Layer Perceptron Classifier . . . . .	70

# Tabellenverzeichnis

2.1	Wahrheitsmatrix . . . . .	9
5.1	Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 1 . . .	44
5.2	Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 2 . . .	44
5.3	Ermittlung der Gewichtungsfaktoren und marginalen Beiträge von Spieler 3 . . .	44
6.1	Übersicht der Klassifizierungsmetriken . . . . .	56
6.2	Übersicht der $R^2$ -Werte . . . . .	65

## Import Libraries

```
In [1]: # LIME and SHAP packages have to be installed via pip
# The %%capture command hide code cell output in Google Colab
%%capture
!pip install lime
!pip install shap

In [2]: from google.colab import files
import io

import pandas as pd
import numpy as np
import tensorflow as tf
import time

# Plot results
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Data preparation and pre-processing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Model classifiers
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

# Classifier metrics
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_score, recall_score, roc_auc_score

# Resample dataset
import collections
from imblearn.over_sampling import SMOTE

# Explainability
import lime
from lime.lime_tabular import LimeTabularExplainer
from lime import submodular_pick
import shap

import warnings
warnings.filterwarnings("ignore")

/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
    import pandas.util.testing as tm
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).
    ("https://pypi.org/project/six/"), FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.base module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the private API.
    warnings.warn(message, FutureWarning)
```

## Data Upload and Cleaning

```
In [3]: # Upload the Credit Card Default dataset with google.colab.files
# Wait till the upload is 100%
uploaded = files.upload()

Dateien auswählen Keine ausgewählt
Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving DefaultOfCreditCardClients.csv to DefaultOfCreditCardClients.csv

In [4]: # Read the dataset from the excel file
data = pd.read_csv(io.BytesIO(uploaded['DefaultOfCreditCardClients.csv']), sep=";", header=1)

In [5]: data.head(3)

Out[5]:
   ID LIMIT_BAL SEX EDUCATION MARRIAGE AGE PAY_0 PAY_2 PAY_3 PAY_4 PAY_5 PAY_6 BILL_AMT1 BILL_AMT2 BILL_AMT3 BILL_AMT4 BILL_AMT5 BILL_AMT6 PAY_AMT1
0   1     20000   2      2     1    24     2     2    -1    -1    -2    -2     3913     3102      689       0       0       0       0
1   2     120000   2      2     2    26    -1     2     0     0     0     2     2682     1725     2682     3272     3455     3261       0
2   3      90000   2      2     2    34     0     0     0     0     0     0     29239    14027    13559    14331    14948    15549    1518
```

```
In [6]: # Dataset without the ID column
data.drop('ID', axis=1, inplace=True)

In [7]: # Checking missing values - there aren't any non-null values
#data.isnull().sum()
```

```
In [8]: # Statistical description
data.describe()
# There are unusual values for PAY_0-PAY_6 the -2, for MARRIAGE the 0 or EDUCATION the 6
# Therefore clean the data
```

Out[8]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000
mean	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	-0.220667	-0.266200	-0.291100	51223.330900	49179
std	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	1.169139	1.133187	1.149988	73635.860576	71173
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-165580.000000	-69777
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	3558.750000	2984
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	22381.500000	21200
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	67091.000000	64006
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000	964511.000000	983931

```
In [9]: # Rename PAY_0 and Target column
data = data.rename(columns={'PAY_0': 'PAY_1', 'default payment next month': 'Default Payment'})
```

```
In [10]: data.loc[data['SEX'] == 2, 'SEX'] = 0
data.loc[data['MARRIAGE'] == 0, 'MARRIAGE'] = 3
clean_education = (data['EDUCATION'] == 0) | (data['EDUCATION'] == 5) | (data['EDUCATION'] == 6)
data.loc[clean_education, 'EDUCATION'] = 4
```

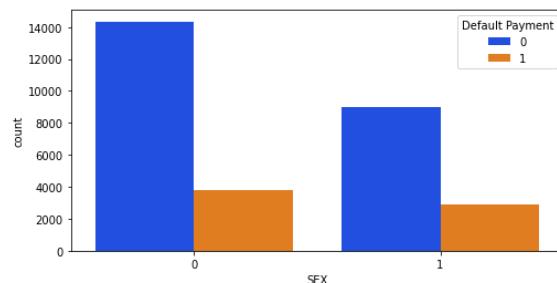
```
In [11]: clean_pay1 = (data['PAY_1'] == -2) | (data['PAY_1'] == -1)
data.loc[clean_pay1, 'PAY_1'] = 0
clean_pay2 = (data['PAY_2'] == -2) | (data['PAY_2'] == -1)
data.loc[clean_pay2, 'PAY_2'] = 0
clean_pay3 = (data['PAY_3'] == -2) | (data['PAY_3'] == -1)
data.loc[clean_pay3, 'PAY_3'] = 0
clean_pay4 = (data['PAY_4'] == -2) | (data['PAY_4'] == -1)
data.loc[clean_pay4, 'PAY_4'] = 0
clean_pay5 = (data['PAY_5'] == -2) | (data['PAY_5'] == -1)
data.loc[clean_pay5, 'PAY_5'] = 0
clean_pay6 = (data['PAY_6'] == -2) | (data['PAY_6'] == -1)
data.loc[clean_pay6, 'PAY_6'] = 0
#data[['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']].describe()
```

## Simple Exploratory Data Analysis

```
In [12]: print(data['SEX'].value_counts())
plt.figure(figsize=(8,4))
sns.countplot(x='SEX', data=data, hue='Default Payment', palette='bright')
```

```
0    18112
1    11888
Name: SEX, dtype: int64
```

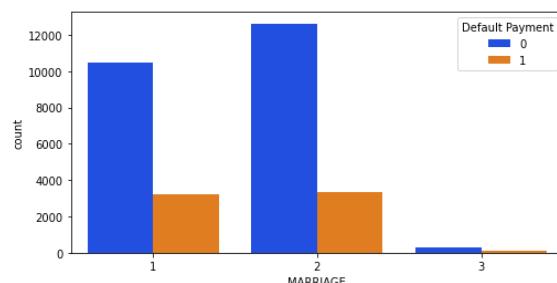
Out[12]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb0d6c25c0>



```
In [13]: print(data['MARRIAGE'].value_counts())
plt.figure(figsize=(8,4))
sns.countplot(x='MARRIAGE', data=data, hue='Default Payment', palette='bright')
```

```
2    15964
1    13659
3     377
Name: MARRIAGE, dtype: int64
```

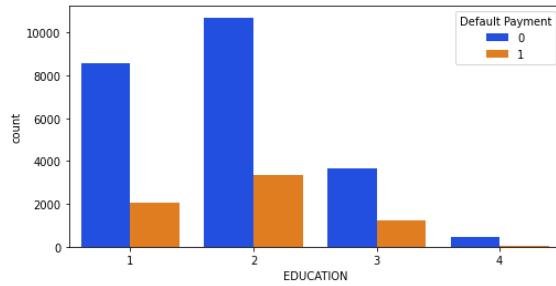
Out[13]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb0d6cf438>



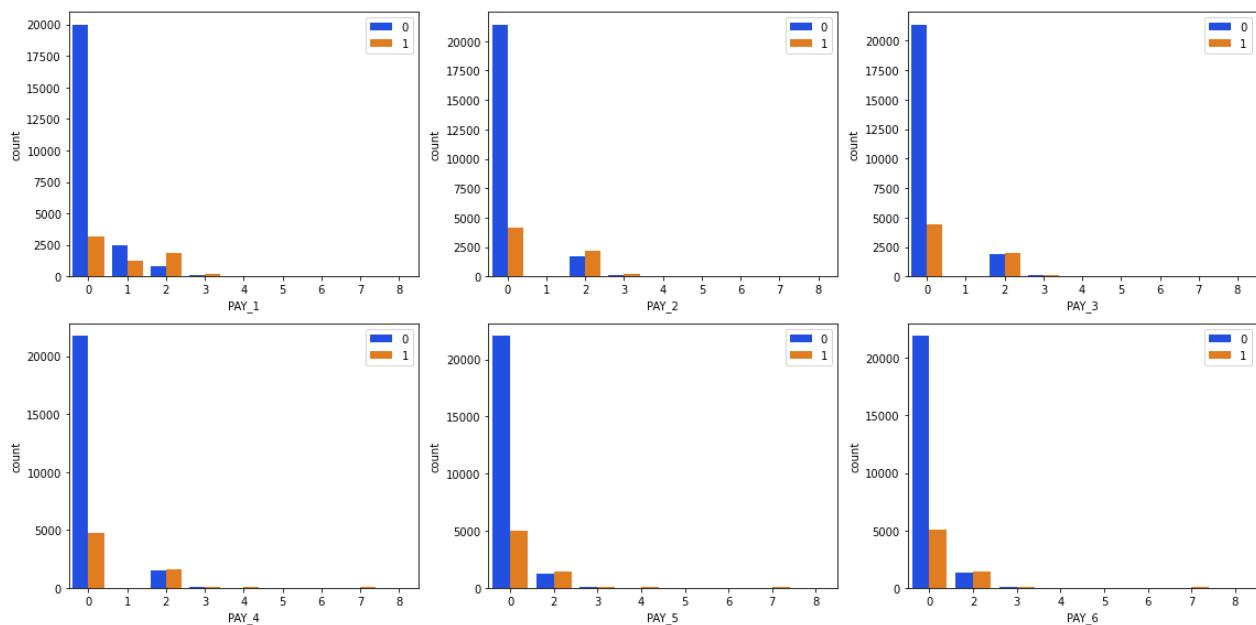
```
In [14]: print(data['EDUCATION'].value_counts())
plt.figure(figsize=(8,4))
sns.countplot(x='EDUCATION', data=data, hue='Default Payment', palette='bright')
```

```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb0e2d35f8>
```



```
In [15]: fig, ax = plt.subplots(2,3)
fig.set_size_inches(16,8)
sns.countplot(x='PAY_1', data=data, hue='Default Payment', palette='bright', ax=ax[0,0]).legend(loc="upper right")
sns.countplot(x='PAY_2', data=data, hue='Default Payment', palette='bright', ax=ax[0,1]).legend(loc="upper right")
sns.countplot(x='PAY_3', data=data, hue='Default Payment', palette='bright', ax=ax[0,2]).legend(loc="upper right")
sns.countplot(x='PAY_4', data=data, hue='Default Payment', palette='bright', ax=ax[1,0]).legend(loc="upper right")
sns.countplot(x='PAY_5', data=data, hue='Default Payment', palette='bright', ax=ax[1,1]).legend(loc="upper right")
sns.countplot(x='PAY_6', data=data, hue='Default Payment', palette='bright', ax=ax[1,2]).legend(loc="upper right")
fig.tight_layout()
fig.show()
```



```
In [16]: #data[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].describe()
```

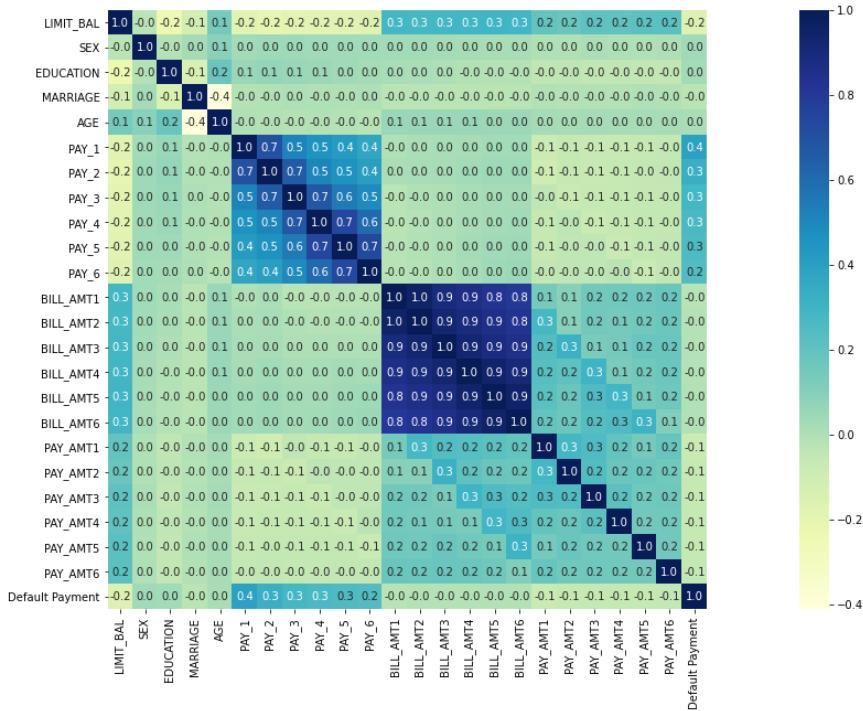
```
In [17]: #data[['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].describe()
```

```
In [18]: #data[['LIMIT_BAL']].describe()
```

```
In [19]: #plt.figure(figsize=(16,8))
#sns.countplot(x='AGE', data=data, hue='Default Payment', palette='bright')
```

```
In [20]: # Correlation analysis
plt.subplots(figsize=(30,10))
sns.heatmap(data.corr(), square=True, annot=True, fmt=".1f", cmap="YlGnBu")
# It seems that PAY_1 has the highest correlation to the target Default Payment
```

Out[20]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fbdb0d0beb70>



## Simple Feature Engineering and Preprocessing

```
In [21]: Y = data['Default Payment']
#Y.head()
```

```
In [22]: # Order features first categorical and second continuous
```

```
feature_order = ['SEX',
                 'EDUCATION',
                 'MARRIAGE',
                 'PAY_1',
                 'PAY_2',
                 'PAY_3',
                 'PAY_4',
                 'PAY_5',
                 'PAY_6',
                 'LIMIT_BAL',
                 'AGE',
                 'BILL_AMT1',
                 'BILL_AMT2',
                 'BILL_AMT3',
                 'BILL_AMT4',
                 'BILL_AMT5',
                 'BILL_AMT6',
                 'PAY_AMT1',
                 'PAY_AMT2',
                 'PAY_AMT3',
                 'PAY_AMT4',
                 'PAY_AMT5',
                 'PAY_AMT6']
```

```
X = data[feature_order]
#X.head()
```

```
In [23]: # List of categorical features - preparation for LIME input
categorical_names = {}
categorical_names[0] = ['female', 'male']
categorical_names[1] = [1, 2, 3, 4]
categorical_names[2] = [1, 2, 3]
categorical_names[3] = [0, 1, 2, 3, 4, 5, 6, 7, 8]
categorical_names[4] = [0, 1, 2, 3, 4, 5, 6, 7, 8]
categorical_names[5] = [0, 1, 2, 3, 4, 5, 6, 7, 8]
categorical_names[6] = [0, 1, 2, 3, 4, 5, 6, 7, 8]
categorical_names[7] = [0, 1, 2, 3, 4, 5, 6, 7, 8]
categorical_names[8] = [0, 1, 2, 3, 4, 5, 6, 7, 8]

# List of continuous features
continuous_features = list(X.columns[9:])
```

```
In [24]: # Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))

def scaleColumns(X, cols_to_scale):
    for col in cols_to_scale:
        X[col] = pd.DataFrame(scaler.fit_transform(pd.DataFrame(X[col])),
                               columns=[col])
    return X

X_scaled = scaleColumns(X,[continuous_features])
#X_scaled.head()
```

```
In [25]: # Split the data into train/test datasets
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.2, random_state=12345)
#print(len(X_train)) 24000
#print(len(X_test)) 6000
```

## Resampling

```
In [26]: # Set random state and make the outputs stable
np.random.seed(12345)
```

```
In [27]: # Number of default payment and the ratio of it
# clearly imbalanced data, therefore resample with SMOTE
print(data["Default Payment"].value_counts())
print("Default Payment Percentage 0: {:.2f} %".format(data[data["Default Payment"]==0].shape[0] / data.shape[0] * 100) )
print("Default Payment Percentage 1: {:.2f} %".format(data[data["Default Payment"]==1].shape[0] / data.shape[0] * 100) )

0    23364
1     6636
Name: Default Payment, dtype: int64
Default Payment Percentage 0: 77.88 %
Default Payment Percentage 1: 22.12 %
```

```
In [28]: print(collections.Counter(Y_train))
#print(len(Y_train)) 24000
# The minority class 1 have just 5331 instances while the majority class 0 have 18669
```

```
Counter({0: 18669, 1: 5331})
```

```
In [29]: # Resample the train data
X_resampled, Y_resampled = SMOTE().fit_sample(X_train, Y_train)
# Convert the data to the same type as before SMOTE
X_train = pd.DataFrame(X_resampled, columns=feature_order)
Y_train = pd.Series(Y_resampled)
```

```
In [30]: print(collections.Counter(Y_train))
#print(len(Y_train)) 37338
# After applying SMOTE Method the classes are balanced
```

```
Counter({0: 18669, 1: 18669})
```

```
In [31]: # Already tested the Black-Box models for unbalanced and balanced data
# --> the balanced data have better recall and f1 score
# Metrics without SMOTE:
# Model Precision Recall F1 Score Accuracy ROC
# SVC 0.651106 0.406130 0.500236 0.823500 0.672820
# RFC 0.637191 0.375479 0.472517 0.817667 0.658027
# MLP 0.660274 0.369349 0.473710 0.821500 0.658263

# Metrics with SMOTE:
#Model Precision Recall F1 Score Accuracy ROC
# SVC 0.545605 0.504215 0.524094 0.800833 0.693747
# RFC 0.521073 0.521073 0.521073 0.791667 0.693976
# MLP 0.403059 0.686590 0.507937 0.710667 0.701974
```

## Black-Box Model Training

```
In [32]: # Fitting a Support Vector Machine Classifier
# Actually the SVC model don't need so much computing time
# but the Lime package requires probabilities and
# therefore the default probability=False has to be changed to True
svc = SVC(kernel='linear', probability=True, random_state=12345)
%time svc.fit(X_train, Y_train)
svc_pred = svc.predict(X_test)
```

```
CPU times: user 5min 45s, sys: 733 ms, total: 5min 45s
Wall time: 5min 46s
```

```
In [33]: # Confusion Matrix
pd.crosstab(Y_test, svc_pred, rownames=['Actual'], colnames=['Predicted'])
```

```
Out[33]:
Predicted      0      1
Actual
_____
0    4073   622
1     628   677
```

```
In [34]: # Model performs
print(classification_report(Y_test, svc_pred))
```

	precision	recall	f1-score	support
0	0.87	0.87	0.87	4695
1	0.52	0.52	0.52	1305
accuracy			0.79	6000
macro avg	0.69	0.69	0.69	6000
weighted avg	0.79	0.79	0.79	6000

```
In [35]: # Fitting a Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=150, criterion='entropy', random_state=12345)
%time rfc.fit(X_train, Y_train)
rfc_pred = rfc.predict(X_test)
```

CPU times: user 29.4 s, sys: 16.9 ms, total: 29.4 s  
Wall time: 29.5 s

```
In [36]: # Confusion Matrix
pd.crosstab(Y_test, rfc_pred, rownames=['Actual'], colnames=['Predicted'])
```

```
Out[36]:
Predicted      0      1
Actual
_____
0    4073   622
1     635   670
```

```
In [37]: # Model performs
print(classification_report(Y_test, rfc_pred))
```

	precision	recall	f1-score	support
0	0.87	0.87	0.87	4695
1	0.52	0.51	0.52	1305
accuracy			0.79	6000
macro avg	0.69	0.69	0.69	6000
weighted avg	0.79	0.79	0.79	6000

```
In [38]: # Fitting a Multi-Layer Perceptron Classifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000, random_state=12345)
%time mlp.fit(X_train, Y_train.values.ravel())
mlp_pred = mlp.predict(X_test)
```

CPU times: user 54 s, sys: 38.9 ms, total: 54 s  
Wall time: 54.1 s

```
In [39]: # Confusion Matrix
pd.crosstab(Y_test, mlp_pred, rownames=['Actual'], colnames=['Predicted'])
```

```
Out[39]:
Predicted      0      1
Actual
_____
0    3566   1129
1     479   826
```

```
In [40]: # Model performs
print(classification_report(Y_test, mlp_pred))
```

	precision	recall	f1-score	support
0	0.88	0.76	0.82	4695
1	0.42	0.63	0.51	1305
accuracy			0.73	6000
macro avg	0.65	0.70	0.66	6000
weighted avg	0.78	0.73	0.75	6000

```
In [41]: # Get a table for the metrics of the model performs
svc_prec = precision_score(Y_test, svc_pred)
svc_rec = recall_score(Y_test, svc_pred)
svc_f1 = f1_score(Y_test, svc_pred)
svc_acc = accuracy_score(Y_test, svc_pred)
svc_roc = roc_auc_score(Y_test, svc_pred)

rfc_prec = precision_score(Y_test, rfc_pred)
rfc_rec = recall_score(Y_test, rfc_pred)
rfc_f1 = f1_score(Y_test, rfc_pred)
rfc_acc = accuracy_score(Y_test, rfc_pred)
rfc_roc = roc_auc_score(Y_test, rfc_pred)

mlp_prec = precision_score(Y_test, mlp_pred)
mlp_rec = recall_score(Y_test, mlp_pred)
mlp_f1 = f1_score(Y_test, mlp_pred)
mlp_acc = accuracy_score(Y_test, mlp_pred)
mlp_roc = roc_auc_score(Y_test, mlp_pred)

metric = pd.DataFrame([['SVC', svc_prec, svc_rec, svc_f1, svc_acc, svc_roc]],
                      columns = ['Model', 'Precision', 'Recall', 'F1 Score', 'Accuracy', 'ROC'])

metric2 = pd.DataFrame([['RFC', rfc_prec, rfc_rec, rfc_f1, rfc_acc, rfc_roc]],
                      columns = ['Model', 'Precision', 'Recall', 'F1 Score', 'Accuracy', 'ROC'])

metric3 = pd.DataFrame([['MLP', mlp_prec, mlp_rec, mlp_f1, mlp_acc, mlp_roc]],
                      columns = ['Model', 'Precision', 'Recall', 'F1 Score', 'Accuracy', 'ROC'])

metric = metric.append([metric2, metric3], sort=False)
metric
```

Out[41]:

	Model	Precision	Recall	F1 Score	Accuracy	ROC
0	SVC	0.521170	0.518774	0.519969	0.791667	0.693146
0	RFC	0.518576	0.513410	0.515980	0.790500	0.690464
0	MLP	0.422506	0.632950	0.506748	0.732000	0.696241

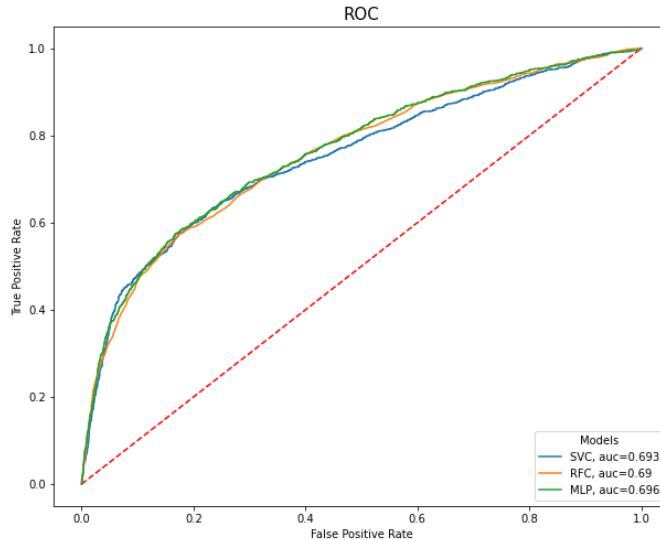
```
In [42]: # ROC Curve
probs_svc = svc.predict_proba(X_test)[:,1]
FPR1, TPR1, _ = metrics.roc_curve(Y_test, probs_svc)

probs_rfc = rfc.predict_proba(X_test)[:,1]
FPR2, TPR2, _ = metrics.roc_curve(Y_test, probs_rfc)

probs_mlp = mlp.predict_proba(X_test)[:,1]
FPR3, TPR3, _ = metrics.roc_curve(Y_test, probs_mlp)

plt.figure(figsize=(10,8))
plt.plot([0, 1], [0, 1], 'r--')
plt.plot(FPR1,TPR1,label="SVC, auc="+str(round(svc_roc,3)))
plt.plot(FPR2,TPR2,label="RFC, auc="+str(round(rfc_roc,3)))
plt.plot(FPR3,TPR3,label="MLP, auc="+str(round(mlp_roc,3)))
plt.legend(loc=4, title='Models', facecolor='white')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC', size=15)
```

Out[42]: Text(0.5, 1.0, 'ROC')



## Explainability

**Compare the LIME explanations of the Black-Box models at 4 different instances, additionally the Kernel SHAP explanation for MLP**

```
In [43]: predict_svc = lambda x: svc.predict_proba(x).astype(float)
predict_rfc = lambda x: rfc.predict_proba(x).astype(float)
predict_mlp = lambda x: mlp.predict_proba(x).astype(float)
#rfc.predict(X_test) array([1, 0, 0, ..., 0, 0, 0])
#rfc.predict_proba(X_test) array([[0.3133333, 0.68666667], [0.86, 0.14], [0.8, 0.2], ...])
```

```
In [44]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                               class_names=Y_train.unique(),
                                                               feature_names=X_train.columns,
                                                               categorical_features=categorical_names,
                                                               verbose=True)
# verbose = True, so the intercept the LIME and Black-Box Model prediction will be shown
```

Instance 1 - Actual Label: 0, Predicted Label: 0

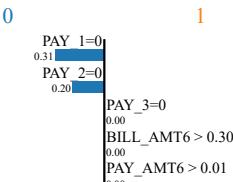
```
In [45]: #i = 35
#print(X_test.iloc[i]) shows the instance as the actual data point
#print(Y_test.iloc[i])    0
#print(svc_pred[i])      0
#print(rfc_pred[i])      0
#print(mlp_pred[i])      0
```

```
In [46]: i = 35
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_svc, num_features=5)
print(exp.score)
exp.show_in_notebook()
#exp.local_exp
exp.as_list()
```

Intercept 0.8694245959934388  
Prediction\_local [0.36850833]  
Right: 0.3569111830679485  
CPU times: user 3.98 s, sys: 324 ms, total: 4.3 s  
Wall time: 3.97 s  
0.862279738698042

Prediction probabilities

0	0.64
1	0.36



Feature	Value
PAY_1=0	True
PAY_2=0	True
PAY_3=0	True
BILL_AMT6	0.30
PAY_AMT6	0.01

```
Out[46]: [('PAY_1=0', -0.31115890778276223),
           ('PAY_2=0', -0.20141194001493934),
           ('PAY_3=0', 0.003953150195173049),
           ('BILL_AMT6 > 0.30', 0.00391378520958269),
           ('PAY_AMT6 > 0.01', 0.00378764933835298)]
```

```
In [47]: # The prediction of LIME model is the sum of the intercept and coefficients
0.8694245959934388 + (-0.31115890778276223) + (-0.20141194001493934) + 0.003953150195173049 + 0.00391378520958269 + 0.00378764933835298
```

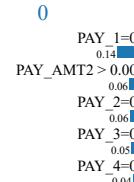
Out[47]: 0.36850833293884594

```
In [48]: i = 35
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_rfc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.6814565699659403  
Prediction\_local [0.32728557]  
Right: 0.1866666666666668  
CPU times: user 1.17 s, sys: 322 ms, total: 1.49 s  
Wall time: 1.12 s  
0.3749560960082812

Prediction probabilities

0	0.81
1	0.19



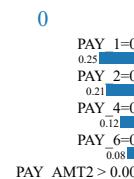
Feature	Value
PAY_1=0	True
PAY_AMT2	0.00
PAY_2=0	True
PAY_3=0	True
PAY_4=0	True

```
In [49]: i = 35
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 1.011229539754286  
Prediction\_local [0.2923108]  
Right: 0.2066158066500271  
CPU times: user 979 ms, sys: 250 ms, total: 1.23 s  
Wall time: 930 ms  
0.44401059872943155

Prediction probabilities

0	0.79
1	0.21



Feature	Value
PAY_1=0	True
PAY_2=0	True
PAY_4=0	True
PAY_6=0	True
PAY_AMT2	0.00

```
In [50]: explainer = shap.KernelExplainer(predict_mlp, X_train)
shap.initjs()

i = 35
%time shap_values = explainer.shap_values(X_test.iloc[i], nsamples=100)
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc[i], link="identity")
```

Using 37338 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K samples.



CPU times: user 43.7 s, sys: 1.74 s, total: 45.4 s  
Wall time: 41.6 s



```
In [51]: shap_values[1]
```

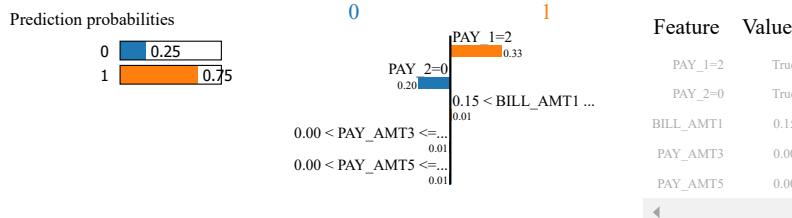
```
Out[51]: array([-0.00403602, 0.          , -0.04542741, -0.05777129, -0.04465046,
       -0.00629676, -0.02115615, -0.00933667, 0.          , 0.          ,
       0.          , 0.          , 0.          , 0.          , 0.          ,
       0.          , 0.          , 0.          , -0.01378795, 0.          ,
       0.          , -0.10593468, 0.          ])
```

Instance 2 - Actual Label: 1, Predicted Label: 1

```
In [52]: #i = 2675
#print(X_test.iloc[i])
#print(Y_test.iloc[i])    1
#print(svc_pred[i])      1
#print(rfc_pred[i])      1
#print(mlp_pred[i])      1
```

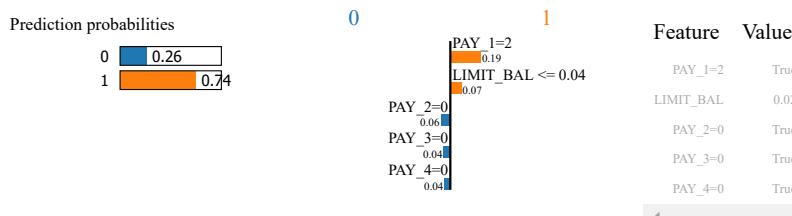
```
In [53]: i = 2675
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_svc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.6165365833753855  
Prediction\_local [0.73449843]  
Right: 0.7547813416124128  
CPU times: user 3.99 s, sys: 312 ms, total: 4.3 s  
Wall time: 3.95 s  
0.6501115199111637



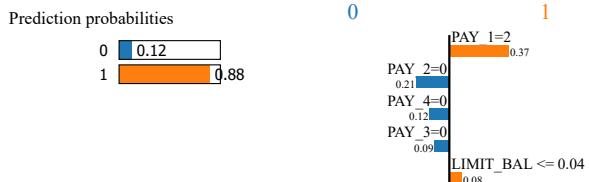
```
In [54]: i = 2675
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_rfc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.5174806674038772  
Prediction\_local [0.6404675]  
Right: 0.74  
CPU times: user 1.13 s, sys: 300 ms, total: 1.43 s  
Wall time: 1.09 s  
0.39781062691381536



```
In [55]: i = 2675
        %time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
        print(exp.score)
        exp.show_in_notebook()
```

```
Intercept 0.762848613315382  
Prediction_local [0.79443109]  
Right: 0.881724734200468  
CPU times: user 966 ms, sys: 267 ms, total: 1.23 s  
Wall time: 930 ms  
0.5244831116770893
```



Feature	Value
PAY_1=2	True
PAY_2=0	True
PAY_4=0	True
PAY_3=0	True
LIMIT_BAL	0.0

```
In [56]: explainer = shap.KernelExplainer(predict_mlp, X_train)
         shap.initjs()
```

```
i = 2675
%time shap_values = explainer.shap_values(X_test.iloc[i], nsamples=100)
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc
```

Using 37338 background data samples could cause slower run times. Consider using `shap.sample(data, K)` or `shap.kmeans(data, K)` to summarize the background as K samples.



```
In [57]: shap_values[1]
```

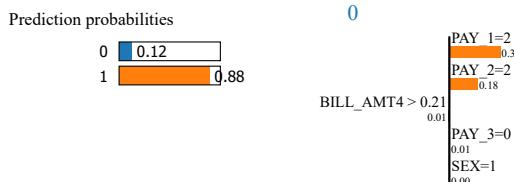
```
Out[57]: array([0.          , 0.          , 0.          , 0.32525366, 0.          ,  
   0.          , 0.          , 0.          , 0.          , 0.04145788,  
   0.          , 0.          , 0.          , 0.          , 0.          , 0.          ,  
   0.          , 0.          , 0.          , 0.          , 0.          , 0.          ,  
   0.          , 0.          , 0.          , 1])
```

Instance 3 - Actual Label: 0, Predicted Label: 1

```
In [58]: #i = 555  
#print(X_test.iloc[i])  
#print(Y_test.iloc[i])  
#print(svc_pred[i])  
#print(rfc_pred[i])  
#print(mlp_pred[i])
```

```
In [59]: i = 555
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_svc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

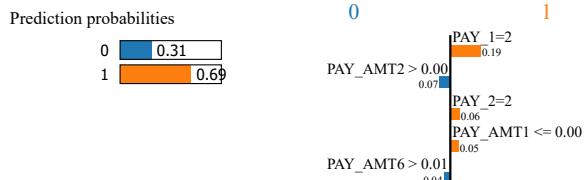
```
Intercept 0.42261127891875716
Prediction_local [0.92256416]
Right: 0.8792847399742162
CPU times: user 3.98 s, sys: 294 ms, total: 4.28 s
Wall time: 3.92 s
0.565325193231365
```



Feature	Value
PAY_1=2	True
PAY_2=2	True
BILL_AMT4	0.3
PAY_3=0	True
SEX=1	True

```
In [60]: i = 555
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_rfc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.42565547811343807  
Prediction local [0.61834952]  
Right: 0.693333333333334  
CPU times: user 1.15 s, sys: 316 ms, total: 1.46 s  
Wall time: 1.11 s  
0.4065419550916338

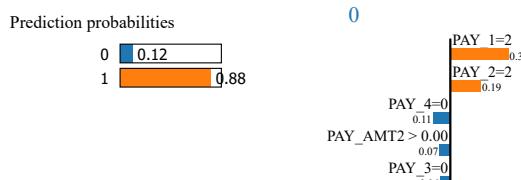


Feature Value

PAY_1=2	True
PAY_AMT2	0.00
PAY_2=2	True
PAY_AMT1	0.00
PAY_AMT6	0.01

```
In [61]: i = 555
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.575174170239024  
Prediction\_local [0.88712598]  
Right: 0.8822306520804934  
CPU times: user 973 ms, sys: 245 ms, total: 1.22 s  
Wall time: 922 ms  
0.47404336021716476



Feature Value

PAY_1=2	True
PAY_2=2	True
PAY_4=0	True
PAY_AMT2	0.00

```
In [62]: explainer = shap.KernelExplainer(predict_mlp, X_train)
shap.initjs()
```

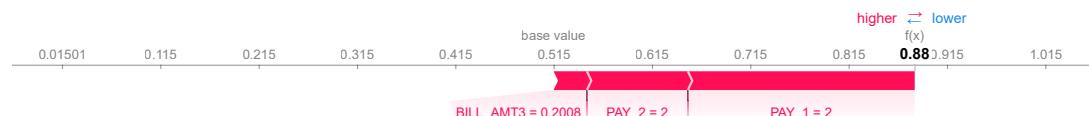
```
i = 555
%time shap_values = explainer.shap_values(X_test.iloc[i], nsamples=100)
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc[i], link="identity")
```

Using 37338 background data samples could cause slower run times. Consider using shap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K samples.

js

CPU times: user 44.5 s, sys: 573 ms, total: 45.1 s  
Wall time: 41.7 s

Out[62]:



```
In [63]: shap_values[1]
```

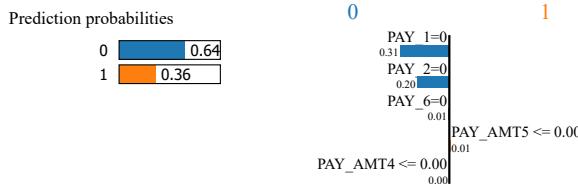
```
Out[63]: array([0.          , 0.          , 0.          , 0.23069281, 0.10266177,
       0.          , 0.          , 0.          , 0.          , 0.          ,
       0.          , 0.          , 0.          , 0.03386288, 0.          ,
       0.          , 0.          , 0.          , 0.          , 0.          ,
       0.          , 0.          , 0.          , 0.          ])
```

Instance 4 - Actual Label: 1, Predicted Label: 0

```
In [64]: #i = 1880
#print(X_test.iloc[i])
#print(Y_test.iloc[i])    1
#print(svc_pred[i])      0
#print(rfc_pred[i])      0
#print(mlp_pred[i])      0
```

```
In [65]: i = 1880
        %time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_svc, num_features=5)
        print(exp.score)
        exp.show_in_notebook()
```

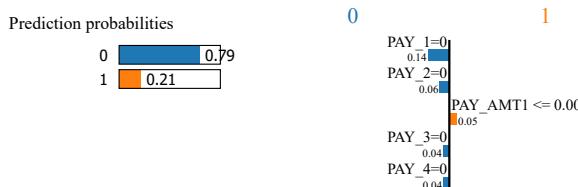
```
Intercept 0.882546408232073
Prediction_local [0.36422746]
Right: 0.3569330122544934
CPU times: user 4.04 s, sys: 317 ms, total: 4.36 s
Wall time: 4 s
0.8700084063477235
```



Feature	Value
PAY_1=0	True
PAY_2=0	True
PAY_6=0	True
PAY_AMT5	0.00
PAY_AMT4	0.00

```
In [66]: i = 1880
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_rfc, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

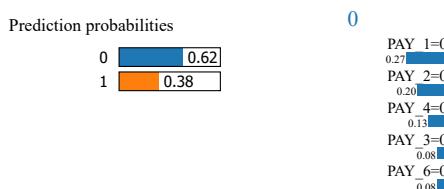
```
Intercept 0.64399487276505
Prediction_local [0.41942645]
Right: 0.21333333333333335
CPU times: user 1.17 s, sys: 308 ms, total: 1.48 s
Wall time: 1.11 s
0.3326753833437992
```



Feature	Value
PAY_1=0	True
PAY_2=0	True
PAY_AMT1	0.00
PAY_3=0	True
PAY_4=0	True

```
In [67]: i = 1880
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

```
Intercept 1.0729312450809303  
Prediction_local [0.32117]  
Right: 0.3831789850241079  
CPU times: user 965 ms, sys: 255 ms, total: 1.22 s  
Wall time: 918 ms  
0.4749963340942152
```



Feature	Value
PAY_1=0	True
PAY_2=0	True
PAY_4=0	True
PAY_3=0	True
PAY_6=0	True

```
In [68]: explainer = shap.KernelExplainer(predict_mlp, X_train)
shap.initjs()

i = 1880
%time shap_values = explainer.shap_values(X_test.iloc[i], nsamples=100)
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc[i], link="identity")
```

Using 37338 background data samples could cause slower run times. Consider using `shap.sample(data, K)` or `shap.kmeans(data, K)` to summarize the background as K samples.

js

```
CPU times: user 44 s, sys: 632 ms, total: 44.6 s  
Wall time: 41.1 s
```



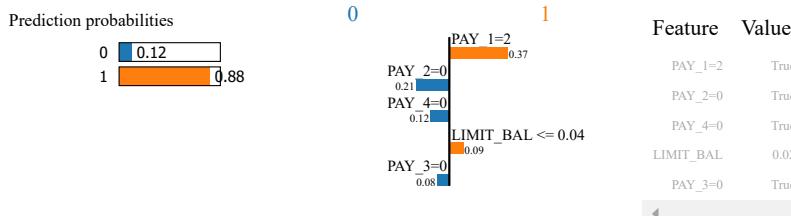
```
In [69]: shap_values[1]
```

```
Out[69]: array([ 0.        , -0.04568959, -0.01980776, -0.07857919, -0.0114412 ,  
   0.        , 0.        , -0.01191077, -0.01679955, 0.        ,  
   0.02530845, 0.01675877, 0.        , 0.        , 0.        , 0.        ,  
   0.01032664, 0.        , 0.        , 0.        , 0.        , 0.        ,  
   0.        , 0.        , 0.        ])
```

### Choose the MLP model at the Instance 2 and compare LIME results for different kernels

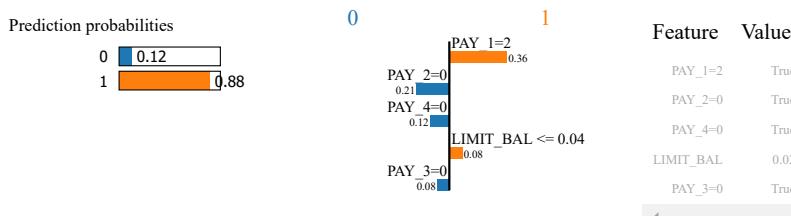
```
In [81]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,  
                                                               class_names=Y_train.unique(),  
                                                               feature_names = X_train.columns,  
                                                               categorical_features = categorical_names,  
                                                               kernel_width=5,  
                                                               verbose=True)  
  
i = 2675  
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)  
print(exp.score)  
exp.show_in_notebook()
```

```
Intercept 0.7524639858316332  
Prediction_local [0.79832487]  
Right: 0.881724734200468  
CPU times: user 985 ms, sys: 272 ms, total: 1.26 s  
Wall time: 940 ms  
0.525888938508656
```



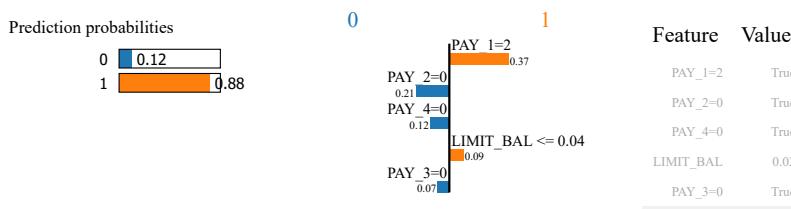
```
In [82]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,  
                                                               class_names=Y_train.unique(),  
                                                               feature_names = X_train.columns,  
                                                               categorical_features = categorical_names,  
                                                               kernel_width=4,  
                                                               verbose=True)  
  
i = 2675  
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)  
print(exp.score)  
exp.show_in_notebook()
```

```
Intercept 0.7517766638664278  
Prediction_local [0.79204986]  
Right: 0.881724734200468  
CPU times: user 962 ms, sys: 261 ms, total: 1.22 s  
Wall time: 921 ms  
0.529183404046643
```



```
In [83]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,  
                                                               class_names=Y_train.unique(),  
                                                               feature_names = X_train.columns,  
                                                               categorical_features = categorical_names,  
                                                               kernel_width=3,  
                                                               verbose=True)  
  
i = 2675  
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)  
print(exp.score)  
exp.show_in_notebook()
```

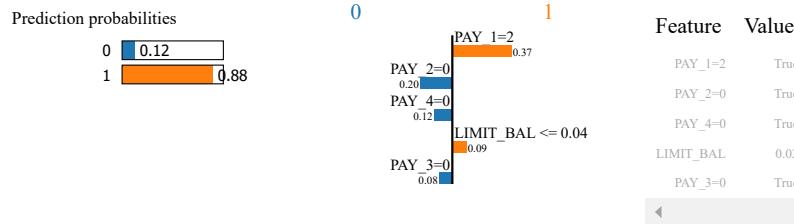
```
Intercept 0.744718161731869  
Prediction_local [0.80177693]  
Right: 0.881724734200468  
CPU times: user 961 ms, sys: 258 ms, total: 1.22 s  
Wall time: 915 ms  
0.5362884852763443
```



```
In [84]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                               class_names=Y_train.unique(),
                                                               feature_names=X_train.columns,
                                                               categorical_features=categorical_names,
                                                               kernel_width=2,
                                                               verbose=True)

i = 2675
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

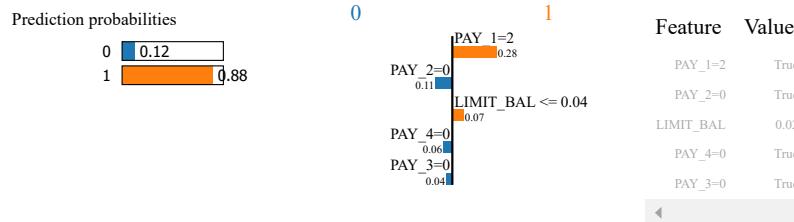
Intercept 0.7452880577664628  
Prediction\_local [0.80482674]  
Right: 0.881724734200468  
CPU times: user 982 ms, sys: 252 ms, total: 1.23 s  
Wall time: 944 ms  
0.5581746440247689



```
In [85]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                               class_names=Y_train.unique(),
                                                               feature_names=X_train.columns,
                                                               categorical_features=categorical_names,
                                                               kernel_width=1,
                                                               verbose=True)

i = 2675
%time exp = explainer_lime.explain_instance(X_test.iloc[i], predict_fn = predict_mlp, num_features=5)
print(exp.score)
exp.show_in_notebook()
```

Intercept 0.6136595605085354  
Prediction\_local [0.76491398]  
Right: 0.881724734200468  
CPU times: user 956 ms, sys: 264 ms, total: 1.22 s  
Wall time: 921 ms  
0.6020468732140188



**Compare the SP-LIME across the three Black-Box models, additionally the global Kernel SHAP explanation for MLP**

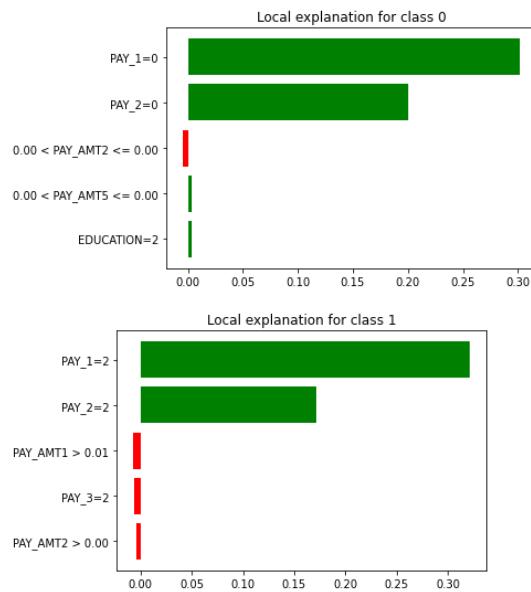
```
In [90]: explainer_lime = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                               class_names=Y_train.unique(),
                                                               feature_names=X_train.columns,
                                                               categorical_features=categorical_names,
                                                               verbose=False)

#verbose=False, so the information about local explainer don't show up
```

```
In [94]: %time sp_obj = submodular_pick.SubmodularPick(explainer_lime, X_train.values, predict_svc, sample_size=20, num_features=5, num_exps_desired=2)
[exp.as_pyplot_figure(label=exp.available_labels()[0]) for exp in sp_obj.sp_explanations]
```

CPU times: user 1min 22s, sys: 8.07 s, total: 1min 30s  
Wall time: 1min 21s

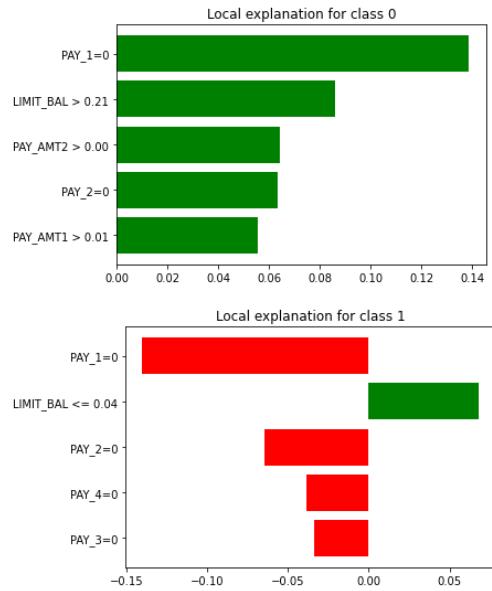
```
Out[94]: [<Figure size 432x288 with 1 Axes>, <Figure size 432x288 with 1 Axes>]
```



```
In [98]: %time sp_obj = submodular_pick.SubmodularPick(explainer_lime, X_train.values, predict_rfc, sample_size=20, num_features=5, num_exps_desired=2)
[exp.as_pyplot_figure(label=exp.available_labels()[0]) for exp in sp_obj.sp_explanations]
```

CPU times: user 24.1 s, sys: 8.19 s, total: 32.3 s  
Wall time: 22.8 s

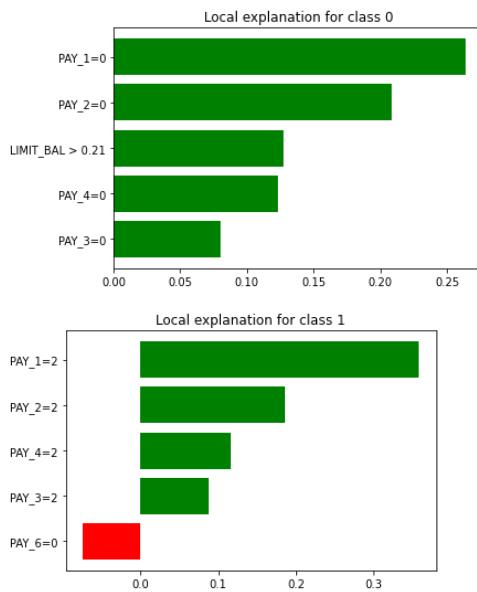
```
Out[98]: [<Figure size 432x288 with 1 Axes>, <Figure size 432x288 with 1 Axes>]
```



```
In [99]: %time sp_obj = submodular_pick.SubmodularPick(explainer_lime, X_train.values, predict_mlp, sample_size=20, num_features=5, num_exps_desired=2)
[exp.as_pyplot_figure(label=exp.available_labels()[0]) for exp in sp_obj.sp_explanations]
```

CPU times: user 22.4 s, sys: 6.87 s, total: 29.3 s  
Wall time: 22.9 s

```
Out[99]: [<Figure size 432x288 with 1 Axes>, <Figure size 432x288 with 1 Axes>]
```

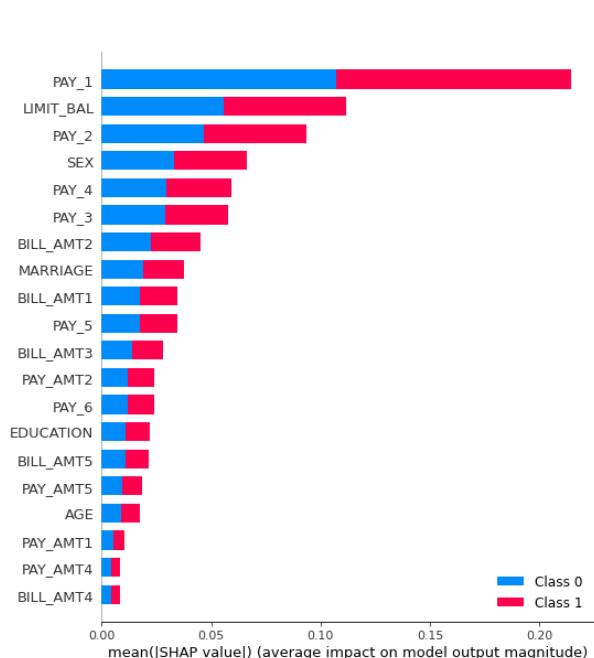


```
In [100]: # Sampling data from the training set to reduce time
# Running without the kmeans end up with ram crash
X_train_summary = shap.kmeans(X_train, 10)
```

```
In [101]: explainer = shap.KernelExplainer(predict_mlp, X_train_summary)
%time shap_values = explainer.shap_values(X_test)
shap.initjs()
shap.summary_plot(shap_values, X_test)
```

Error rendering Jupyter widget: missing widget manager

CPU times: user 30min 48s, sys: 18min 35s, total: 49min 23s  
Wall time: 26min 35s



## **Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Soweit ich auf fremde Materialien, Texte oder Gedankengänge zurückgegriffen habe, enthalten meine Ausführungen vollständige und eindeutige Verweise auf die Urheber und Quellen. Alle weiteren Inhalte der vorgelegten Arbeit stammen von mir im urheberrechtlichen Sinn, sowie keine Verweise und Zitate erfolgen. Mir ist bekannt, dass ein Täuschungsversuch vorliegt, wenn die vorstehende Erklärung sich als unrichtig erweist.

Friedberg, den 16. September 2020

B.Karaoglan

Unterschrift