Duale Hochschule Baden-Württemberg Mannheim

Studienarbeit

Debiasing in Machine Learning

Verzerrung in Daten und Modellen erkennen und korrigieren

Studiengang Informatik

Studienrichtung Informationstechnik

Verfasser:	Denis Dengler, David Schader
Matrikelnummern:	6799951, 5524854
Kurs:	TINF18IT1
Studiengangsleiter:	Prof. Dr. Holger Hofmann
Wissenschaftlicher Betreuer:	Dr. Frank Schulz
Bearbeitungszeitraum:	01.11.2020 - 04.05.2021

Eingereicht: _____

Unterschrift Betreuer _____

Erklärung

Gemäß §5 (3) der „Studien- und Prüfungsordnung DHBW Technik – StuPrO DHBW Technik“ vom 29. September 2017.

Wir versicheren hiermit, dass wir unsere Studienarbeit mit dem Thema: *Debiasing in Machine Learning* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Wir versicheren zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* *falls beide Fassungen gefordert sind*

DHBW Mannheim, Mai 2021

Denis Dengler

David Schader

Kurzfassung

Machine Learning (ML)-Algorithmen, die ein Teilgebiet der Künstlichen Intelligenz darstellen, finden immer mehr Einzug in Bereiche des alltäglichen Lebens. Bekannte Beispiele sind Sprachassistenten wie Alexa und Siri oder diverse Gesichtserkennungssoftwares. Der steigende Einsatz solcher Algorithmen findet sich zudem immer häufiger in sensiblen und kritischen Bereichen wieder. So können ML-Algorithmen Banken dabei unterstützen die Kreditwürdigkeit ihrer Kunden automatisiert einzuschätzen oder die Rückfälligkeit eines Angeklagten vor Gericht zu prognostizieren.

Da diese Algorithmen jedoch auf realen Datensätzen beruhen, welche häufig historisch bedingt oder aufgrund der menschlichen kognitiven Verzerrung vorurteilsbehaftet sind, ist es möglich, dass Diskriminierung bestimmter Gruppen in den Daten auch zu unfairen Algorithmen führt. Aus diesem Anlass werden im Rahmen dieser Arbeit zunächst Definitionen und Metriken für Verzerrungen (*Bias*) und Fairness aus einschlägiger und aktueller Literatur herausgearbeitet. Mit der Implementierung dieser Metriken soll als Teil eines Debiasing-Frameworks die Möglichkeit geboten werden, Verzerrungen und Diskriminierungen in Daten zu detektieren.

Zusätzlich werden verschiedene Möglichkeiten des Debiasing, also der Entzerrung von vorurteilsbehafteten Daten, beleuchtet. Der Fokus liegt dabei auf verschiedenen Resampling-Verfahren, mit denen das Framework nach der Detektion eines Bias in der Lage sein soll, die für den Algorithmus zugrundeliegenden Daten so zu transformieren, dass ein faires Modell trainiert werden kann.

Dabei zeigt sich im Test mit drei Beispielen, dass der implementierte Resampling-Ansatz zu einer allgemeinen Verbesserung der Fairness führt. Jedoch kann die Verwendung verschiedener Parameter wie zum Beispiel unterschiedlicher Fairness-Metriken oder Resampling-Strategien einen großen Einfluss auf den Erfolg des Debiasing haben. In der aktuellen Version kann nicht immer ein vollständiges Debiasing garantiert werden, jedoch kann in den meisten Fällen mit einer deutlichen Verbesserung der Fairness gerechnet werden.

David Schader

Abstract

Machine Learning (ML) algorithms, which represent a subfield of artificial intelligence, are increasingly finding their way into areas of everyday life. Well-known examples are voice assistants such as Alexa and Siri or various facial recognition software. The growing use of such algorithms is also increasingly found in sensitive and critical areas. For example, ML algorithms can help banks automatically assess the creditworthiness of their customers or predict the recidivism of a defendant in court.

However, since these algorithms are based on real-world datasets, which are often historically biased or biased due to human cognitive bias, it is possible that discrimination against certain groups in the data may also lead to unfair algorithms. For this reason, this thesis will first extract definitions and metrics for bias and fairness from relevant and current literature. With the implementation of these metrics, as part of a debiasing framework, the possibility to detect biases and discriminations in data will be provided.

Additionally, various ways of debiasing will be highlighted. The focus is on different resampling methods, with which the framework should be able to transform the underlying data for the algorithm after the detection of a bias in such a way that a fair model can be trained.

Thereby, the test with three examples shows that the implemented resampling approach leads to a general improvement of fairness. However, the use of different parameters such as different fairness metrics or resampling strategies can have a large impact on the success of the debiasing. In the current version, complete debiasing cannot always be guaranteed, but a significant improvement in fairness can be expected in most cases.

David Schader

Vorwort

In dieser Arbeit werden einige besondere Begriffe hervorgehoben. Zum einen sind fachspezifische Begriffe und Abkürzungen in blauer Farbe markiert und werden im Abkürzungsverzeichnis auf Seite X erläutert. In digitaler Form sind diese mit einem Hyperlink zur jeweiligen Seite versehen. Zusätzlich sind alle Formelzeichen in der Nomenklatur auf Seite XI aufgeführt und erläutert. Außerdem wird jeder nummerierte Abschnitt mit dem Namen des jeweiligen Autors versehen. Dieser Befindet sich am Ende jedes Abschnitts.

Aus Gründen der besseren Lesbarkeit wird in dieser Arbeit die Sprachform des generischen Maskulinums verwendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ADASYN	ADAptive SYNthetic	40
AUC	Area Under the Curve	11
AUROC	Area Under the Receiver Operating Characteristics	11
CC	Cluster Centroids	33
CNN	Condensed Nearest Neighbors	37
COMPAS	Correctional Offender Management Profiling for Alternative Sanctions	18
DL	Deep Learning	4
ENN	Edited Nearest Neigbours	36
FP	False positive	7
FPR	False positive rate	10
FN	False negative	7
FNR	False negative rate	9
KI	Künstliche Intelligenz	3
KNN	K Nearest Neighbors	37
ML	Machine Learning	II
NCR	Neighborhood Cleaning Rule	38
OSS	One Sided Selection	37
ReLU	Rectified Linear Units	15
RENN	Repeated Edited Nearest Neighbors	36
ROC	Receiver Operating Characteristics	11
SMOTE	Synthetic Minority Oversampling TEchnique	39
SVM	Support Vector Machine	40
TP	True positive	7
TPR	True positive rate	9
TN	True negative	7
TNR	True negative rate	10

Nomenklatur

S	Menge aller sensitiven Attribute
X	Menge aller Datenobjekte
X_i	Datenobjekt
Y	Menge aller Klassenzugehörigkeiten/Label
Y_i	Reale Klassenzugehörigkeit eines Datenobjekts X_i
$\hat{Y}_i$	Prognostizierte Klassenzugehörigkeit eines Datenobjekts X_i
Y_+	Majorität
Y_-	Minorität
G	Gruppe (Kombination von Attributwerten a verschiedener sensitiven Attribute S_i)
G_+	Privilegierte Gruppe
S_i	Sensitives Attribut
G_-	Unprivilegierte Gruppe
B	Bias eines Attributs
F	Algorithmische Fairness
Z	Scoring-Ergebnis
P	Wahrscheinlichkeit eines Ereignisses
$\perp\!\!\!\perp$	Statistische Unabhängigkeit zweier Variablen
s	Wert eines sensitiven Attributs
A	Menge aller Attribute: $A = S \cap \bar{S}$
A_i	Attribut
a	Attributwert

1 Einleitung

ML-Algorithmen gehören zu einer Form der Künstlichen Intelligenz und werden schon heute in einer Vielzahl von Anwendungen verwendet. Beispiele dafür sind unter anderem die Spracherkennung bei Sprachassistenten sowie die Bild- und Mustererkennung. Neben solchen allgemein nutzbaren Anwendungen ziehen ML-Algorithmen auch immer mehr in spezifischere Bereiche ein. So werden beispielsweise in den USA ML-Algorithmen bei Strafverfahren eingesetzt. Diese Algorithmen sollen eine Einschätzung darüber geben, wie wahrscheinlich es ist, dass Straftäter rückfällig werden. [1]

Neben der alltäglichen Verwendung solcher Algorithmen ist es daher wichtig, diese mit einem kritischen Auge zu betrachten. Fragen, die man sich bei bestimmten Anwendungen stellen sollte, sind z.B. „Wie sicher ist sich der Algorithmus bei seinen Lösungen?“, „Wie *fair* ist der Algorithmus bei seiner Entscheidungsfindung?“ und „Kann auch ein Algorithmus so etwas wie *Vorurteile* besitzen?“. Da diese Art von Algorithmen immer mehr in sensiblen Bereichen verwendet werden, ist es wichtig, dass diese fair sind und keine Gruppen diskriminieren.

Nach einer theoretischen Einführung in die Funktionsweise von ML-Algorithmen soll zunächst eine Begriffsunterscheidung zwischen dem „Bias“ und der „Fairness“ eines Algorithmus stattfinden. Anschließend werden verschiedene Methoden und Vorgehensweisen vorgestellt, mit denen unfaire Daten soweit vorbereitet und transformiert werden können, dass eine höhere Fairness erreicht wird. Daran anknüpfend wird ein Konzept inklusive prototypischer Umsetzung eines Frameworks vorgestellt, mit dem man unter anderem Fairness von Algorithmen messen und Datensätze *debiasen* kann.

Das Ziel dieser Studienarbeit ist es zu zeigen, wie ein Bias in Daten detektiert bzw. Fairness in Modellen gemessen werden kann und basierend auf dieser Messung eine Entzerrung bzw. ein *Debiasing* der zugrundeliegenden Daten durchgeführt werden kann.

David Schader

1.1 Hypothese

Im Rahmen dieser Arbeit soll der Frage nachgegangen werden, ob ein Algorithmus eine Verzerrung in Daten oder Modellen erkennen und diese beheben kann. Es wird davon ausgegangen, dass Daten, mit denen ML-Algorithmen lernen und Vorhersagen treffen sollen, durch verschiedene Arten von Verzerrung (*Bias*) vorurteilsbehaftet sind. Diese Verzerrung kann in vielerlei Formen und in verschiedenen Stadien des Maschinellen Lernens auftreten. Der Ursprung von Bias ist die menschliche kognitive Verzerrung, welche dem Menschen einerseits das Leben erleichtert und hilft Entscheidungen schneller zu treffen, andererseits zur verzerrten Wahrnehmung und vorurteilsbehaftetem Denken führt. Dieser Zustand wird in den gesammelten Daten abgebildet und führt zu einer ebensolchen Verzerrung in den Trainingsdaten. Weiter ist die Frage zu stellen, inwiefern ein ML-Algorithmus auf Fairness untersucht und ein vorurteilsbehaftetes Modell debiased werden kann.

Denis Dengler

1.2 Aufgabenstellung

In dieser Studienarbeit soll zunächst anhand einiger ML-Grundlagen darauf eingegangen werden, wie ein Bias beschrieben werden kann und welchen Einfluss dieser auf die Fairness eines Algorithmus hat. Da diese Algorithmen auf Daten basieren, wird der Fokus dieser Arbeit auf der Bewertung und dem Debiasing der Datensätze liegen. Dazu werden Ansätze vorgestellt, mit denen man Verzerrungen in den Daten erkennen und korrigieren kann. Die prototypische Implementierung findet in Form eine Frameworks statt. Dieses soll Funktionen zur Verfügung stellen, die das Messen von Bias und Fairness und das Debiasing der Daten erleichtern. Dazu soll zunächst eine umfangreiche Literaturrecherche zu den Themen Verzerrung und Fairness durchgeführt werden und mathematische bzw. statistische Ansätze zur Beschreibung eben dieser Begriffe formuliert werden, auf Basis derer das Framework implementiert werden kann. Die Ergebnisse sollen anschließend aufbereitet und bewertet werden. Zur Eingrenzung des breiten Themas soll im Folgenden speziell die binäre Klassifizierung betrachtet werden. Die vorgestellten Methoden und Konzepte sind durch geringfügige Modifikationen auch für nichtbinäre Klassifizierungen und Regressionsmodelle anzuwenden.

Denis Dengler

2 Grundlagen

Als Grundlage für die folgenden Kapitel sollen die Kernbegriffe dieser Arbeit *Machine Learning*, *Bias* und *Fairness* definiert und erläutert werden. Beim Machine Learning wird zusätzlich auf den Trainingsprozess und verschiedene Bewertungsmetriken eingegangen. Es werden zusätzlich mehrere Bias-Arten und Metriken zur Beschreibung der algorithmischen Fairness definiert.

David Schader

2.1 Machine Learning

Machine Learning (ML) bzw. Maschinelles Lernen erlebt zurzeit einen Boom, da die benötigte Hardware sich kontinuierlich verbessert und verschnellert. Der Lern- bzw. Trainings-Prozess eines solchen Algorithmus hat sich mit aktuellen Prozessoren und leistungsstarken Grafikkarten deutlich verschnellert, wodurch die Verwendung von ML-Algorithmen an Attraktivität gewonnen hat. Algorithmen auf der Grundlage von ML ermöglichen eine implementierte Anwendung bzw. Repräsentation von großen Datenmengen. Komplexe Aufgaben, die für den Menschen viel Zeit in Anspruch nehmen, sind für den Einsatz von ML-Algorithmen besonders geeignet. Grundvoraussetzung ist jedoch eine schon vorhandene Menge an Beispieldaten bezogen auf die jeweilige Aufgabe, mit denen der Algorithmus *lernen* bzw. trainiert werden kann. Dabei kann es sich beispielsweise um Sensor- oder simple Eigenschafts-Daten handeln. Einen ML-Algorithmus kann man sich wie eine Person vorstellen, die in einer bestimmten Aufgabe schon sehr viel Erfahrung gesammelt hat und diese schnell auf neue Daten anwenden kann. Das Anwendungsspektrum reicht von simplen Klassifikationen bis hin zu aufwendiger Bildverarbeitung. [2]

Die Begriffe *Künstliche Intelligenz*, *Machine Learning* und *Deep Learning* werden häufig synonym verwendet, was verwirrend sein kann, da sie eigentlich nicht das gleiche bedeuten. Exakte Definitionen für die einzelnen Begriffe sind schwer zu formulieren sein. Im Folgenden kann eine knappe Unterscheidung der Begriffe in dem Gebiet aber durchaus nützlich sein.

Künstliche Intelligenz (KI)

Die KI stellt ein Teilgebiet der Informatik dar. Dabei handelt es sich um Algorithmen oder Maschinen, die dazu in der Lage sind, die Entscheidungsfindung des Menschen zu imitieren. Dies müssen noch nicht zwingend sich anpassbare Algorithmen sein,

die dazu in der Lage sind, zu *lernen*. Es kann sich hierbei auch um Algorithmen mit einem klar definierten Regelsatz handeln wie z.B. Schach-Computer. [2]

Machine Learning (ML)

ML ist ein Teilgebiet der Künstlichen Intelligenz. Es handelt sich dabei um Algorithmen, die mit Trainingsdaten Modelle als Wissensrepräsentationen erstellen, welche anschließend Vorhersagen für neue Daten treffen können. Die Anwendung von ML-Algorithmen bietet sich vor allem dann an, wenn eine analytische Beschreibung eines Problems zu komplex ist, jedoch genügend Trainingsdaten für die Problematik vorhanden sind. [2]

Deep Learning (DL)

DL ist ein Teilgebiet des Machine Learnings. Hierbei kommen sogenannte Künstliche Neuronale Netze als Algorithmen zum Einsatz, die die Struktur der Nervensysteme bei Lebewesen imitieren. Durch die Anordnung einer Vielzahl an Neuronen in mehreren Schichten, die jeweils Eingangssignale verarbeiten und diese Verarbeitung an Neuronen der nächsten Schicht als neue Eingangssignale weiterleiten, können bildklassifizierende Algorithmen, die beispielsweise Bilder in Hunde und Katzen unterscheiden können, erstellt werden. Neben der Bildklassifikation können DL-Algorithmen auch zur Sprachverarbeitung oder Bildsegmentierung genutzt werden. [2]

David Schader

2.1.1 Trainieren von ML-Algorithmen

Wie können ML-Algorithmen *lernen*? Zum Erläutern dieser Thematik wird in diesem Unterkapitel das Beispiel eines ML-Algorithmus betrachtet, der Bilder von Katzen und Hunden unterscheidet.

Kinder lernen mit der Zeit Namen zu bestimmten Objekten und können diese auch unterscheiden. Dieser Lernprozess funktioniert so, dass das Kind, wenn es eine Katze sieht, gesagt bekommt „Das ist ein Katze“. Wenn es einen Hund sieht, bekommt es gesagt „Das ist ein Hund“. Mit der Zeit verknüpft das Kind das Aussehen von Katzen bzw. Hunden mit dem Wort *Katze* bzw. *Hund*. Dieser Prozess wird als *Lernen* bezeichnet. Das Äquivalent dazu bei KI-Algorithmen wird als *Trainieren* bezeichnet. Dieser Prozess funktioniert ähnlich zu dem eben beschriebenen Lernprozess des Kindes und wird in Abbildung 2.1 dargestellt.

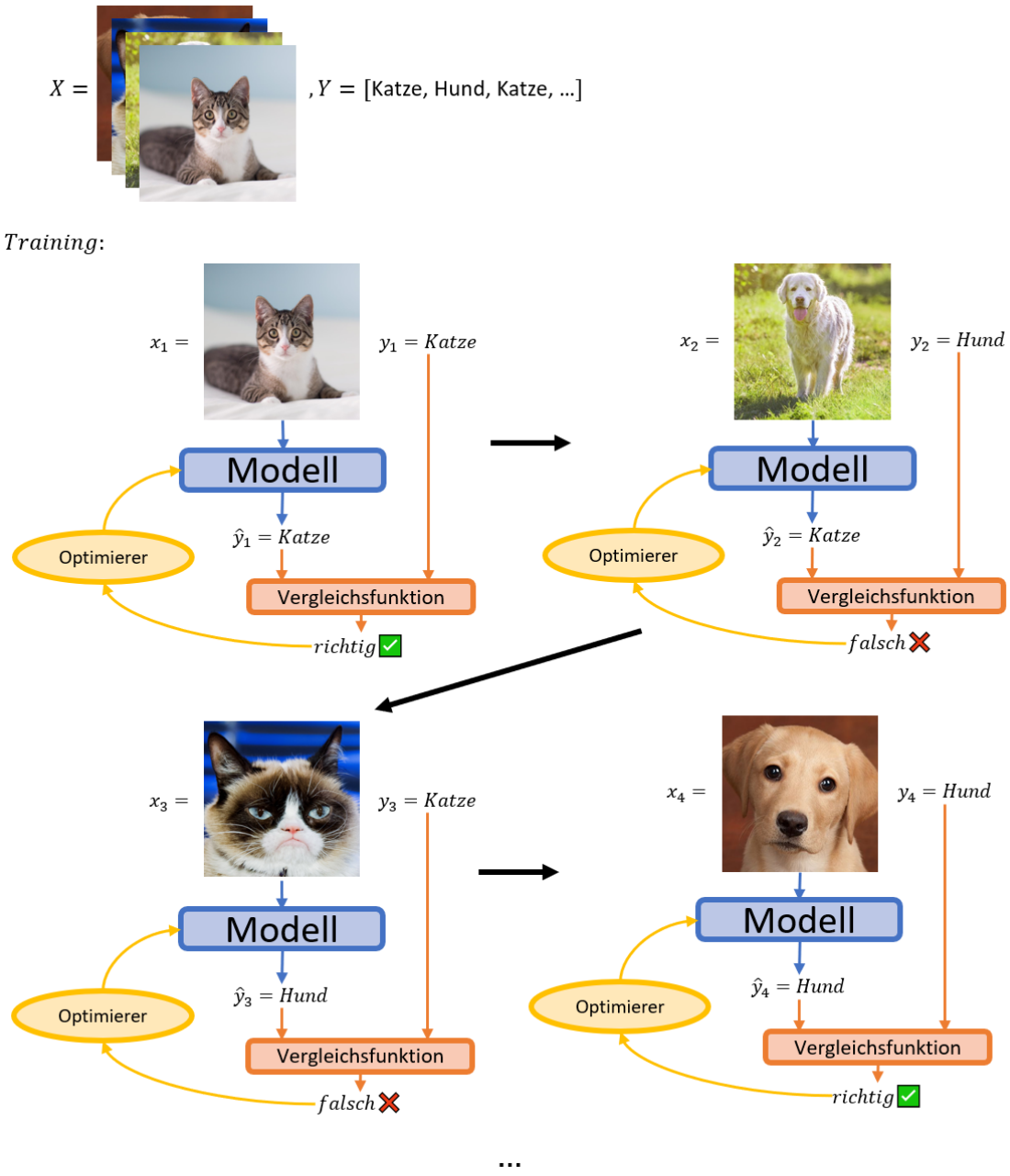


Abbildung 2.1: Training

Zunächst muss ein ML-Modell erstellt werden, welches in unserem Beispiel Bilder einer bestimmten Pixelgröße als Eingangsvariable x_n erhält und als Ausgangsvariable y_n z.B. eine `0` für eine Katze und eine `1` für einen Hund ausgibt. Zusätzlich haben viele ML-Modelle *Gewichte*. Das sind veränderliche Größen, die im *Trainingsprozess* angepasst und optimiert werden können, um die Aufgabenstellung richtig zu lösen.

Für das Trainieren des Modells werden viele Trainingsdaten benötigt. Die Trainingsdaten werden in den Listen X und Y gespeichert. X stellt dabei die Liste dar, in der viele Bilder von Katzen und Hunden gespeichert sind. In Y werden die dazugehörigen Label gespeichert. Beschreibt x_2 also z.B. ein Hundebild, beinhaltet y_2 die Information, dass es

sich bei dem Bild um einen Hund handelt.
Schritt für Schritt bekommt das Modell beim Trainieren zunächst ein einzelnes Bild als Eingangsvariable und berechnet automatisch eine Modell-Lösung $\hat{y}_n$. Anschließend wird in einer Vergleichsfunktion diese berechnete Lösung $\hat{y}_n$ mit dem tatsächlichen Label y verglichen. Im Katze-Hund-Beispiel gibt die Vergleichsfunktion entweder *richtig* aus, wenn sowohl das tatsächliche Label als auch das berechnete Ergebnis *Katze* lautet. Unterscheiden sich beide Werte, ergibt sich dementsprechend ein *falsch*. Das Ergebnis dieser Vergleichsfunktion wird danach an einen Optimierer weitergeleitet, welcher direkt Einfluss auf die Gewichte im Modell hat. Diese werden so angepasst, dass, wenn das Ergebnis richtig war, die Gewichte gestärkt werden, die zum richtigen Ergebnis beigetragen haben. War das Ergebnis *falsch*, werden die Gewichte so angepasst, dass bei dem eben berechneten Bild eher das richtige Ergebnis berechnet worden wäre.

Bild für Bild *lernt* das Modell also dazu. Wurde genug und mit ausreichend vielen verschiedenen Bildern trainiert, ist das Modell dazu in der Lage Bilder mit einer hohen Genauigkeit entweder einem Hund oder einer Katze zuzuordnen.

David Schader

2.1.2 Metriken

ML-Algorithmen sind oftmals keine unfehlbaren Algorithmen. Sie sind jedoch je nachdem wie gut sie trainiert wurden mit einer gewissen Wahrscheinlichkeit z.B. *genau*. Zur Bewertung von Modellen gibt es verschiedene Maße und Herangehensweisen. Daher werden im Folgenden einige vorgestellt:

Confusion Matrix

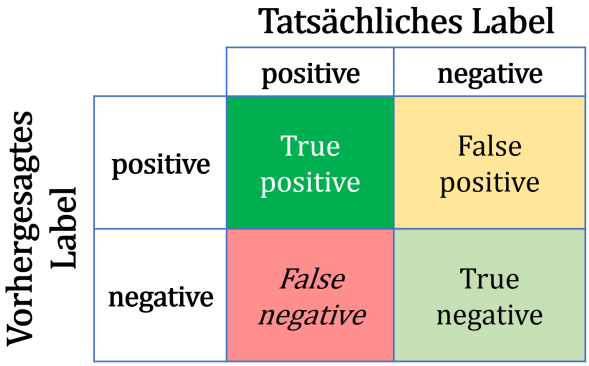


Abbildung 2.2: Confusion Matrix

Die *Confusion Matrix* (zu deutsch: Konfusionsmatrix) in Abbildung 2.2 zeigt die Einteilung von Ergebnissen in mehrere Teilgruppen. Dabei wird zunächst nur das Beispiel einer binären Klassifikation gezeigt. Es gibt daher nur die beiden Label *positive* und *negative*. Da es immer ein tatsächliches und ein vorhergesagtes Label gibt, kann man die Ergebnisse in vier Teilgruppen unterteilen:

- True positive (TP): das vorhergesagte und tatsächliche Label sind positiv
- True negative (TN): das vorhergesagte und tatsächliche Label sind negativ
- False positive (FP): das vorhergesagte Label ist positiv, tatsächlich aber negativ
- False negative (FN): das vorhergesagte Label ist negativ, tatsächlich aber positiv

Nichtbinäre Konfusionsmatrizen

Nicht immer geht es um Anwendungen, bei denen nur in die Klassen *positive* und *negative* unterschieden wird. Oftmals gibt es mehr als nur zwei Klassen. Eine solche Matrix ist in Abbildung 2.3 dargestellt.

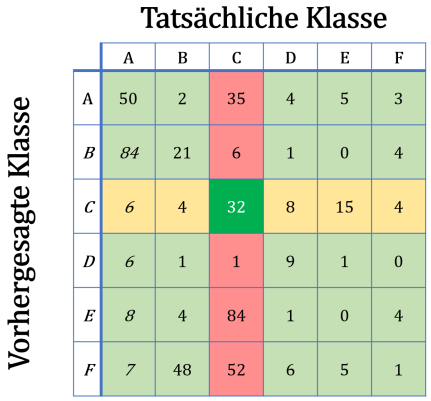


Abbildung 2.3: Nichtbinäre Konfusionsmatrix

Um nun Begriffe wie z.B. True positive oder False negative nutzen zu können, muss genau eine Klasse betrachtet werden, da die Begriffe sonst nicht mehr eindeutig sind. Betrachtet man wie in der Abbildung die Klasse C sind alle FN-Klassifikationen diejenigen, bei denen die tatsächliche Klasse C ist, die vorhergesagt Klasse aber A, B, D, E oder F sein kann. In der Abbildung ist FN rot dargestellt und bildet die Spalte über und unter der dunkelgrünen TP-Klassifikation. Die FP-Klassifikationen sind dementsprechend die gelb markierten Felder, also überall dort, wo zwar die Klasse C vom Modell vorhergesagt wurde, es sich tatsächlich aber um eine andere Klasse handelt. FP-Klassifikationen bilden also die Reihe links und rechts von der TP-Klassifikation und sind gelblich dargestellt. Alle anderen Klassifikationen in der Matrix gehören den TN-Klassifikationen an und sind hellgrün gefärbt. [3]

Selbsttest-Beispiel

Als aktuelles Beispiel für die Konfusionsmatrix wird im Folgenden auf Corona-Selbsttests eingegangen. Diese sind dazu gedacht, als Schnelltests im Eigengebrauch angewendet zu werden. Damit kann man herausfinden, ob man aktuell mit SARS-CoV-2 infiziert und infektiös ist. Verwendet werden in der folgenden Abbildung 2.4 Daten aus der beigelegten Anleitung des Selbsttests von *Hotgen*.

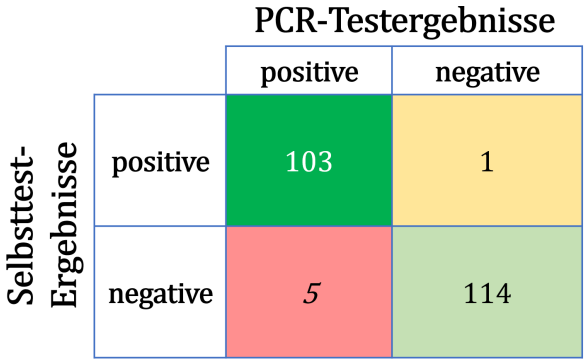


Abbildung 2.4: Selbsttest Konfusionsmatrix

Vor der Zulassung muss der Anbieter vorweisen, dass die Selbsttest zu einem gewissen Maß aussagekräftig sind. Welche Maße mithilfe einer Konfusionsmatrix berechnet werden können, wird im Folgenden gezeigt.

Accuracy (Genauigkeit)

Mit der *Accuracy* wird die Treffergenauigkeit berechnet. Sie stellt den Anteil der korrekt klassifizierten Tests in Bezug zu allen Tests dar und wird wie folgt berechnet:

$$accuracy = \frac{\text{korrekt klassifiziert}}{\text{alle Klassifikationen}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Bezogen auf das Selbsttest-Beispiel ergibt sich folgende Genauigkeit:

$$\text{Testgenauigkeit} = \frac{103 + 114}{103 + 114 + 1 + 5} = \frac{217}{223} \approx 97{,}31\% \tag{2.2}$$

Die *Accuracy* deckt auf der einen Seite alle Teile der Konfusionsmatrix ab, auf der anderen Seite kann sie sich aber auch als äußerst ineffizient erweisen, vor allem dann, wenn die Klassen nicht gleich verteilt sind. Für die Lösung dieser Problematik gibt es viele weitere Metriken.

Precision (Präzision)

Die *Precision* bezeichnet ein Maß, welches in Bezug zu allen positiv erkannten Klassen steht. Sie wird wie folgt berechnet:

$$precision = \frac{\text{richtig positiv}}{\text{positiv vorhergesagt}} = \frac{TP}{TP + FP} \tag{2.3}$$

Die Präzision der Selbsttests kann also wie folgt berechnet werden:

$$\text{Testpräzision} = \frac{103}{103 + 1} = \frac{103}{104} \approx 99{,}04\% \tag{2.4}$$

Recall / Sensitivität / True positive rate (TPR)

Der *Recall* gibt an, in welchem Bezug die richtig positiv erkannten Fälle zu allen tatsächlich positiven Fällen stehen:

$$recall = \frac{\text{richtig positiv}}{\text{tatsächlich positiv}} = \frac{TP}{TP + FN} \tag{2.5}$$

Damit ergibt sich folgende Test-Sensitivität:

$$\text{Test-Sensitivität} = \frac{103}{103 + 5} = \frac{103}{108} \approx 95{,}37\% \tag{2.6}$$

False negative rate (FNR)

Die FNR ist das Komplement zur TPR bzw. Sensitivität.

$$FNR = 1 - \text{Sensitivität} = \frac{FN}{TP + FN} \tag{2.7}$$

Damit ergibt sich folgende Test-FNR:

$$FNR_{Test} = 1 - 95{,}37\% = 4{,}63\% \tag{2.8}$$

Spezifität / True negative rate (TNR)

Entsprechend zur Sensitivität gibt die Spezifität an, in welchem Bezug die richtig negativ erkannten Fälle zu allen tatsächlichen negative Fällen stehen:

$$Spezifität = \frac{\text{richtig negativ}}{\text{tatsächlich negativ}} = \frac{TN}{TN + FP} \tag{2.9}$$

Damit ergibt sich folgende Test-Spezifität:

$$\text{Test-Spezifität} = \frac{114}{114 + 1} = \frac{103}{108} \approx 99{,}13\% \tag{2.10}$$

Bei Corona-Selbsttests ist z.B. die Spezifität das wichtigere Maß, welches sehr hoch sein soll, damit falsch negativ Getestete nicht in dem Glauben negativ zu sein weitere Personen infizieren.

False positive rate (FPR)

Die FPR ist das Komplement zur TNR bzw. Spezifität.

$$FPR = 1 - \text{Spezifität} = \frac{FP}{TN + FP} \tag{2.11}$$

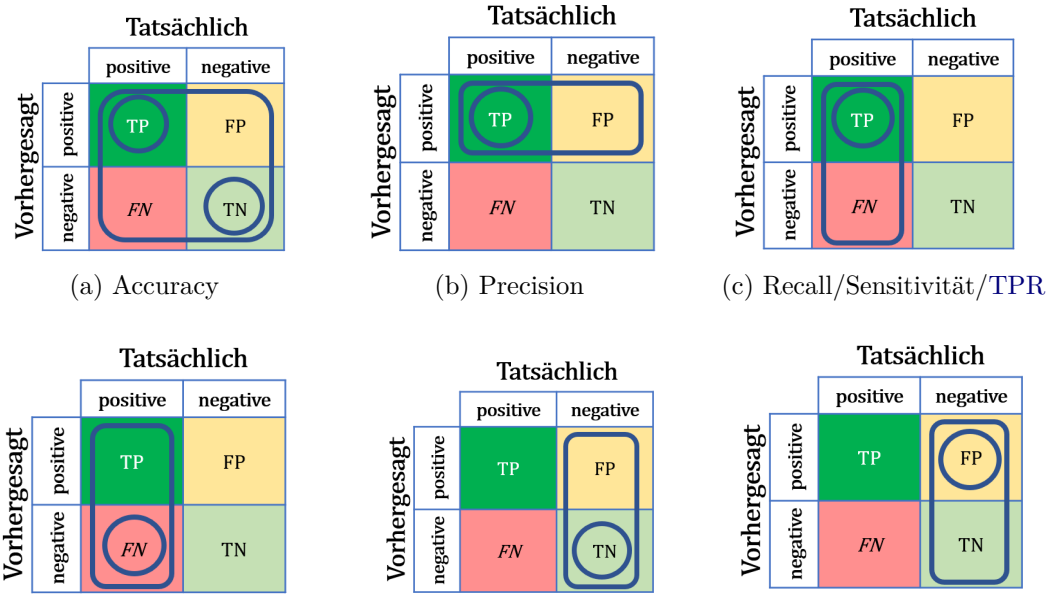
Damit ergibt sich folgende Test-FPR:

$$FPR_{Test} = 1 - 99{,}13\% = 0{,}87\% \tag{2.12}$$

Übersicht

In Abbildung 2.5 wird eine Übersicht der bisher vorgestellten Metriken mithilfe der Konfusionsmatrizen gezeigt. Dabei stellen die Punkte den Zähler und das Rechteck den Nenner der Gleichungen dar.

Die letzten präsentierten Metriken haben jedoch die Problematik, dass sie zwar spezifischer sind, jedoch nicht alle Bereiche der Konfusionsmatrix abbilden. Dafür wurde das F-Maß entwickelt.



(a) Accuracy (b) Precision (c) Recall/Sensitivität/TPR

(d) FNR (e) Spezifität/TNR (f) FPR

Abbildung 2.5: Metriken-Übersicht

F-Maß (F1-Score)

Das F-Maß fasst die *Precision* und den *Recall* zusammen. Dabei handelt es sich jedoch nicht um ein rein arithmetisches Mittel der beiden genannten Maße. Das F-Maß ist empfindlicher für das schwächere Maß. Die Gleichung für das F-Maß lautet wie folgt:

$$F1 = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision} \tag{2.13}$$

Bezogen auf das Test-Beispiel erhält man für das F-Maß also Folgendes:

$$F1_{Test} = 2 \cdot \frac{95{,}37\% \cdot 99{,}04\%}{95{,}37\% + 99{,}04\%} \approx 97{,}17\% \tag{2.14}$$

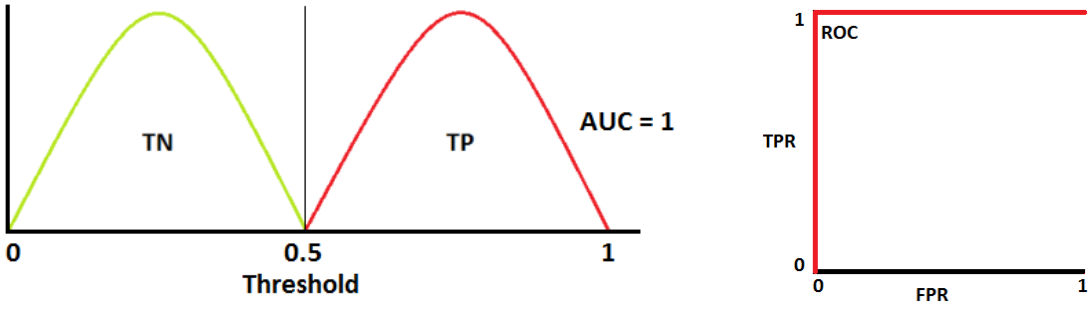
Der Vorteil des F-Maßes ist, dass es zwar nicht so grob ist wie die allgemeine *Accuracy*, jedoch zwei spezifische Metriken so zusammenfasst, dass das schwächere Maß mehr ins Gewicht fällt.

AUC - ROC Kurven

Eine weitere geeignete Metrik zum Bewerten von ML-Algorithmen sind die Receiver Operating Characteristics (ROC) - Kurven. Da diese unter anderem aus den Area Under the Curve (AUC) - Kurven hergeleitet werden, spricht man teilweise auch von Area Under the Receiver Operating Characteristics (AUROC). Mit einem ROC-Diagramm kann man

für Klassifikationsmodelle, die v.a. aus mehr als zwei Klassen bestehen, die Klassifikation für die einzelnen Klassen bewerten. Dabei wird für jede Klasse eine Kurve erstellt. Auf der x-Achse ist die FPR und auf der y-Achse die TPR dargestellt. Diese beiden Metriken korrelieren miteinander und die Kurve startet immer beim Punkt (0% | 0%) und endet beim Punkt (100% | 100%).

Ein Modell, welches nur in zwei Klassen unterteilen soll, hat im Optimalfall einen perfekten Threshold-Wert. Über und unter dem Wert kann das Modell die Eingangswerte dann immer zu 100% einer Klasse zuordnen; es gibt nur TP- und TN-Klassifikationen und keine FP- und FN-Detektionen. Dabei wäre AUC = 1 und die AUC-Kurve würde wie Abbildung 2.6 a) dargestellt aussehen.



(a) AUC = 1 (b) ROC für AUC = 1

Abbildung 2.6: AUROC = 1 [4]

Es gibt also keine False-Klassifikationen. Dementsprechend sieht die dazugehörige ROC-Kurve wie auf der rechten Seite der Abbildung dargestellt aus.

Weitere Beispiele für solche Kurven sind in Abbildung 2.7 zu sehen. Zunächst wird auf der linken Seite der Abbildung der schlechteste Fall betrachtet: Das Modell kann beide Klassen nicht unterscheiden, dementsprechend entsteht eine lineare ROC-Kurve. Die meisten Fälle sehen ähnlich zur rechten Seite der Abbildung aus. Dabei überlappen sich beide Klassen ein wenig, wodurch alle vier Klassifikationstypen der Konfusionsmatrix in unterschiedlichen Größenordnungen entstehen können. Die ROC-Kurve nimmt dabei eine logarithmische Gestalt an. [4, 5]

Bei einem Klassifikationsmodell, das in eine Vielzahl von Klassen unterscheiden soll, kann eine ROC-Kurve wie in Abbildung 2.8 dargestellt aussehen. In diesem Beispiel geht es um ein Modell, welches Bilder handgeschriebener Ziffern in die zehn Ziffern 0 bis 9 unterscheiden kann.

Hier ist gut erkennbar, dass es mehrere Klassen gibt, in die unterschieden wird. Zu jeder Klasse gibt es eine eigene Kurve. Die Kurven können sich durchaus unterscheiden. Mit der ROC-Kurve erhält man also eine Metrik, welche einem zeigt, wie unterschiedlich gut das

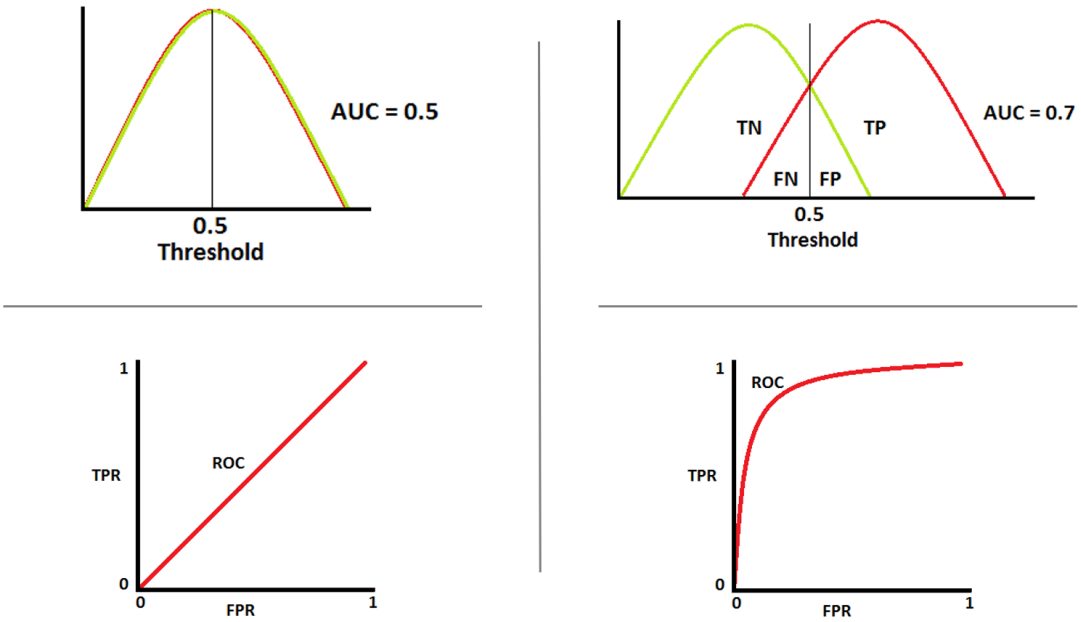


Abbildung 2.7: AUC- und ROC-Kurven [4]

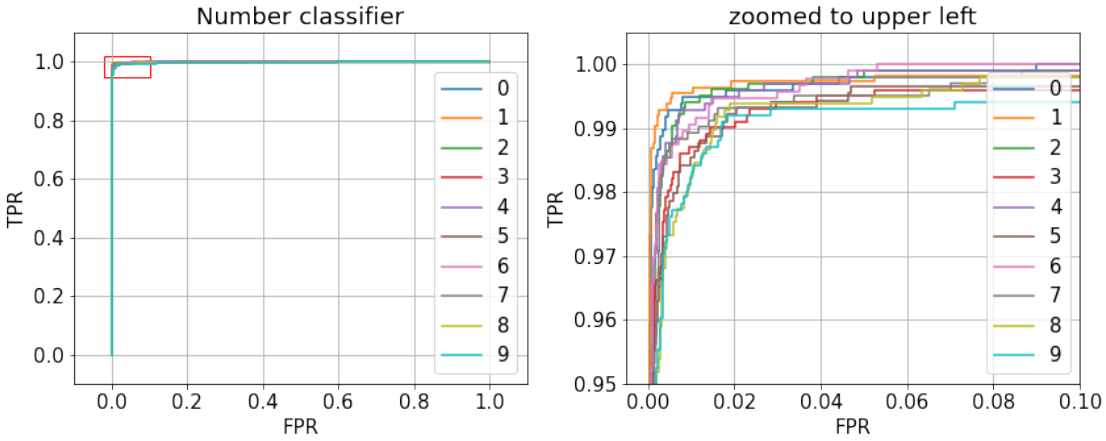


Abbildung 2.8: ROC-Numbers

Modell einzelne Klassen unterscheiden kann. Allgemein zeigt dieses Beispiel aber, dass es sich um ein sehr gutes Modell handelt, da man zum Sehen der Unterschiede zwischen den Klassen in die linke obere Hälft heranzoomen muss.
Auf das Zahlenbeispiel wird in dieser Studienarbeit nochmal in einem Unterabschnitt im Abschnitt 2.2 eingegangen.

David Schader

2.2 Bias

Unter dem Begriff *Bias* (zu deutsch: Verzerrung) versteht man ein Phänomen, bei dem vorurteilsbehaftete Daten zu fehlerhaften Ergebnissen führen. [6] Der Ursprung von Bias liegt in der evolutionär bedingten kognitiven Verzerrung des menschlichen unterbewussten Denkens und Handelns, welche dabei hilft, anhand von Erfahrung, Erziehung oder im Sinne des Überlebensinstinkts schnelle Entscheidungen zu treffen. Diese Effizienzsteigerung bei der Entscheidungsfindung lässt sich ebenso in der Informatik anwenden. ML-Algorithmen lernen so zum Beispiel aus vorgegebenen Daten, indem sie versuchen Muster zu erkennen, welche wiederum bei der Einordnung von neuen Daten helfen. Dieser Mechanismus führt zu effizienteren Algorithmen, weist jedoch gleichzeitig eine Verzerrung auf. Dies kann verschiedene Ursachen, wie etwa eine heterogene Datengrundlage oder die Abbildung unbewusster menschlicher Neigungen in den vorhandenen Daten, haben. [7] Um den Begriff und die Ausprägungen der Verzerrung besser zu verstehen, werden im Folgenden ausgewählte Typen von Verzerrung vorgestellt.

Denis Dengler

Historical / Real World Bias

Bei der Generierung von Daten wird ein Abbild der realen Welt erzeugt. Wie in der Einleitung dieses Abschnitts bereits angeklungen ist die reale Welt vorurteilsbehaftet, was dazu führt, dass es implizit zu Verzerrungen in den gesammelten Daten kommt. Einerseits können historische Daten Zustände und Verhältnisse abbilden, die heute nicht mehr aktuell sind und somit das Bild verfälschen, andererseits verstecken sich verschiedene Arten der menschlichen, kognitiven Verzerrung in der realen Welt und somit auch in den generierten Daten. [7]

Denis Dengler

Selection / Representation Bias

Der Selection-Bias bzw. Representation-Bias ist eine Verzerrung, die sehr schnell bei Nichtbeachtung auftauchen kann, indem im Trainings-Datensatz für einen Klassifizierungs-Algorithmus die jeweilige Klassenverteilung ungleichmäßig ist.

Im realen Leben könnte ein Algorithmus, der die Kreditwürdigkeit von Personen bestimmt und der mit deutlich mehr Trainings-Daten von männlichen Personen als mit weiblichen Personen trainiert wurde, im Nachhinein Frauen statistisch seltener eine hohe Kreditwürdigkeit zuordnen als Männern. Solche Beispiele sind im Allgemeinen zwar gut verständlich,

jedoch wurde für die Veranschaulichung des Selection-Bias im Folgenden zunächst ein abstrakteres Beispiel gewählt: Es geht um die Klassifizierung von handgeschriebenen Ziffern. Der dazugehörige Datensatz ist zu finden unter `http://yann.lecun.com/exdb/mnist/`. Dieser besteht aus 60.000 Trainings- und 10.000 Test-Bildern. Ein Bild einer handschriftlichen Zahl hat jeweils 28×28 Pixel, die einen 8-Bit-Grauwert beinhalten bzw. einen Wert im Bereich von 0 bis 255. Die ersten neun Bilder aus dem Trainings-Datensatz sind exemplarisch in Abbildung 2.9 dargestellt.

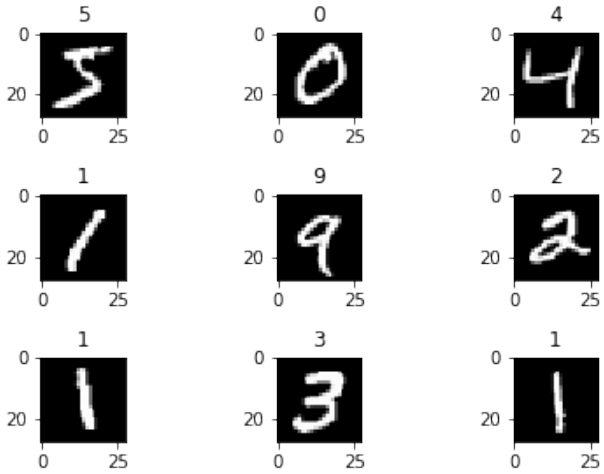


Abbildung 2.9: Beispiel Ziffern

Erstellt wurde ein Neuronales Netzwerk, welches zwischen der Zahl 2 und 7 unterscheiden soll. Dieser Netzwerktyp wurde gewählt, da der Selection Bias zunächst an einem Beispiel eines Klassifizierungs-Algorithmus, der nur in zwei Klassen unterscheidet, erläutert wird. Die beiden Ziffern wurden gewählt, da aus persönlicher Betrachtung eine Verwechslung der beiden Zahlen durchaus wahrscheinlicher ist als andere Kombinationen von Zahlen. Die Vermutung lautet: Je höher der Anteil von 7-Daten im Trainingsdatensatz ist, desto höher ist die Genauigkeit, eine 7 zu erkennen. Gleiches gilt umgekehrt für die 2. Es handelt sich bei dem Netzwerk um ein *Multiple Layer Perceptron*-Netzwerk, welches aus eindimensionalen Layern einzelner Neuronen besteht. Dieser Algorithmus-Typ kann also dem DL zugeordnet werden. Nach dem Input-Layer bestehend aus 784 Neuronen (ein Neuron für jedes Pixel eines Bildes) gibt es zwei Hidden-Layer mit jeweils 80 bzw. 40 Neuronen. Das Output-Layer besteht aus zehn Neuronen für entsprechend zehn Ziffern, da der Algorithmus mit beliebig vielen Ziffertypen trainiert werden kann. Die Aktivierungsfunktion der Neuronen aus dem letzten bzw. Output-Layer ist die Softmax-Funktion, welche eine Wahrscheinlichkeit zwischen null und eins ausgibt. Die Aktivierungsfunktion der Neuronen in den Hidden-Layern ist die Rectified Linear Units (ReLU)-Funktion. Wird wie in diesem Fall nur auf zwei Ziffern trainiert, geben am Ende auch nur zwei Neuronen für die entsprechenden Ziffern einen Wert größer als null aus.

Für eine Veranschaulichung des Selection-Bias wurde das Netzwerk zunächst mit unterschiedlichen Verhältnissen in der Anzahl der 2 und 7 trainiert. Anschließend kann mit dem Test-Datensatz der beiden Ziffern das trainierte Netzwerk auf die Genauigkeit getestet werden, also in wie viel Prozent aller Bilder für eine Ziffer diese richtig erkannt wurde. Als richtig zählt immer die Ziffer, deren Neuron die höchste Wahrscheinlichkeit ausgibt.

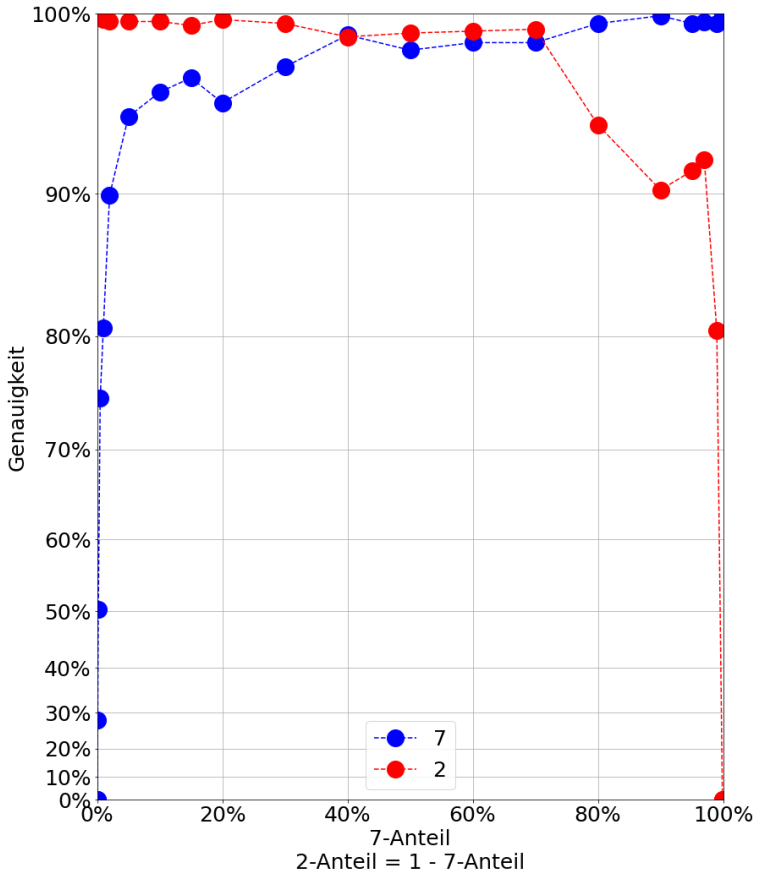


Abbildung 2.10: Selection Bias am Beispiel der Klassifikation von handschriftlichen Zahlen

Wichtig ist hierbei zunächst, dass die y-Achse bzw. die Genauigkeit logarithmisch skaliert wurde, da die Darstellung so anschaulicher ist. Diese Grafik zeigt das Phänomen des Selection-Bias sehr deutlich. Die Vermutung, dass ein höherer Anteil von 7-Daten im Trainingsdatensatz mit einer höheren 7-*Genauigkeit* korreliert, bewahrheitet sich. Entsprechendes gilt für die 2. Es wird deutlich, dass bei einem ausgewogenen Anteil zwischen 2 und 7 die Genauigkeit sehr hoch ist. Desto mehr der Anteil in eine Richtung zeigt, desto ungenauer wird die Genauigkeit für die entsprechend andere Ziffer. Zusätzlich ist aber auch zu sehen, dass die Architektur des Neuronalen Netzwerks an sich definitiv gut geeignet ist, da eine hohe Genauigkeit von über 90% für beide Ziffern auch bei einer unausgewogenen Verteilung im Trainings-Datensatz entsteht. Die Genauigkeit sinkt jedoch rapide, wenn im Trainingsdatensatz mit überwiegender Mehrheit eine der beiden Zahlen vertreten ist. [8]

Gender Bias

Der Gender Bias beschreibt eine Verzerrung, die durch Unterrepräsentation eines Geschlechts hervorgerufen wird. Diese ist oft historisch bedingt und kann sich zum Beispiel beim Thema Beruf äußern. Betrachtet man die Berufstätigkeit von Männern und Frauen vor 100 Jahren, wird man feststellen, dass zum einen die Anzahl der berufstätigen Frauen signifikant niedriger ist, als die der Männer und zum anderen, dass besonders höherqualifizierte Berufe von Männern dominiert werden. Diese Verteilung hat sich über die letzten 100 Jahre jedoch sehr stark verändert. Heutzutage sind Frauen häufig berufstätig und übernehmen zudem auch immer häufiger Führungspositionen. Stellt man sich nun einen Algorithmus vor, der mit den verzerrten Daten der letzten 100 Jahre lernt, so kommt es dazu, dass bestimmte Berufe eher mit dem männlichen Geschlecht assoziiert werden als andere und umgekehrt. Dieses Phänomen lässt sich natürlich nicht nur bei Berufsbezeichnungen beobachten, sondern auch bei anderen Bezeichnungen, sodass gewisse Substantive oder Adjektive eher als feminin und andere als maskulin klassifiziert werden. Eine solche Klassifizierung ist nicht immer falsch, da bestimmte Substantive wie etwa *Mutter* und *Tochter* mit einem eindeutigen Geschlecht einhergehen. In den meisten Fällen ist die Bezeichnung jedoch geschlechtsunabhängig zu verstehen und sollte deshalb gleichermaßen dem weiblichen als auch dem männlichen Geschlecht zugewiesen werden können. In [9] beschreibt eine Gruppe der *Boston University* in Kooperation mit der *Microsoft Research New England* dieses Phänomen in der englischen Sprache und führt es am Beispiel eines Algorithmus vor, der mit Daten aus Zeitungsartikeln Analogien zu den englischen Personalpronomina *she* und *he* findet. In Abbildung 2.11 werden ausgewählte Wörter entlang zweier Achsen projiziert: Auf der x-Achse kann der Grad der Assoziation zu einem bestimmten Geschlecht (links: *she*, rechts: *he*) und auf der y-Achse der Grad der Verzerrung abgelesen werden.

Ein weiteres Alltagsbeispiel, welches das beschriebene Phänomen illustriert, ist der *Google Translator*, ein Übersetzungsalgorithmus des Technologieunternehmens Google. Dieses Beispiel wurde von Alexandra Geese, einer deutschen Politikerin und Mitglied des Europäischen Parlaments, während eines Workshops des *Panel for Future of Science and Technology (STOA)* des Europäischen Parlaments zum Thema *Policy options for the ethical governance of disruptive technologies* vorgestellt. [10] Dabei wurden Sätze aus dem Ungarischen ins Englische übersetzt. Die Besonderheit bei der ungarischen Sprache ist, dass es nur ein Personalpronomen gibt, welches geschlechtsunabhängig verwendet wird. Bei der Übersetzung musste der Algorithmus jedoch ein geschlechterspezifisches Pronomen im Englischen wählen, sodass folgende Übersetzungen zustande kamen: *She is beautiful. He is clever. He understands math. She is kind. He is a doctor. She is a cleaner. He is a politician. She is a teacher. He is strong. etc.* Da eine solche fehlerhafte Assoziation zur

Abbildung 2.11: Projektion ausgewählter Wörter nach Geschlecht und Grad der Verzerrung [9]

Diskriminierung eines Geschlechts führen kann, ist es wichtig, eine eindeutige Klassifizierung nur dann zuzulassen, wenn die Bezeichnung tatsächlich ein konkretes Geschlecht besitzt.

Denis Dengler

Racial Bias

Eine weitere wichtige Art der Verzerrung ist der Racial Bias. Bei dieser Art von Verzerrung kommt es zur ungleichen Behandlung verschiedener ethnischer Gruppen. Ein gutes Beispiel hierfür ist die Software Correctional Offender Management Profiling for Alternative Sanctions (COMPAS). COMPAS ist eine Unterstützungssoftware, die den amerikanischen Gerichten helfen soll, das Risiko, dass ein Angeklagter rückfällig wird, einzuschätzen. Anhand von 137 Fragen soll ein Algorithmus dieses Risiko auf einer Skala von 0 bis 10 vorhersagen. Die Korrektklassifizierungsrate liegt bei etwa 61% [1]. Es ist unklar, wie genau der vom U.S. Unternehmen *Equivant* entwickelte Algorithmus funktioniert, jedoch konnte über die Dauer der Verwendung festgestellt werden, dass der Algorithmus eher dazu tendiert, das Risiko für dunkelhäutige Angeklagte fälschlicherweise als hoch einzuschätzen, während hellhäutige Angeklagte als weniger risikoreich eingestuft werden. Laut der Analyse von *ProPublica* [1] wurden nahezu doppelt so viele Afroamerikaner mit einem hohen Risiko eingestuft, obwohl sie nicht rückfällig wurden, als Weiße. Umgekehrt wurden mehr Weiße mit einem niedrigen Risiko eingestuft, obwohl diese rückfällig wurden, als Afroamerikaner (vgl. Abbildung 2.12). Leider können die Ursachen für solche Unterschiede nicht vollständig nachvollzogen werden, da der zugrundeliegende Algorithmus nicht bekannt ist.

Denis Dengler

	WHITE	AFRICAN AMERICAN
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

Abbildung 2.12: Falschklassifizierung durch COMPAS-Software [1]

Measurement Bias

Der Measurement Bias bezieht sich auf Fehler, die bei der Daten-Messung oder -Auswertung auftauchen. Diese können z.B. beim reinen Transkribieren auftreten. Die gemessenen Daten werden nicht richtig digitalisiert oder es gibt systematische Fehlmessungen in bestimmten Bereichen. Es ist nicht unüblich, dass in großen Datensätzen, bei denen viele Merkmale erhoben werden, nicht alle Merkmale in jeder Zeile ausgefüllt sind oder nur mit einem Strich oder Fragezeichen versehen wurden. [7]

David Schader

Omitted Variable Bias / Exclusion Bias

Ausgelassene Variablen können zu Verzerrungen in Modellen führen, wenn diese einen Effekt auf die Ergebnisvariable haben. Es handelt sich bei diesen Variablen also um benötigte Variablen, die für ein korrektes Ergebnis notwendig sind und in Wahrheit Einfluss auf das tatsächliche Ergebnis haben. [7] Zum besseren Verständnis wird im Folgenden ein Beispiel erläutert: Es gibt ein Modell, welches die Lebenserwartung von Menschen schätzt. Dabei werden viele Daten gesammelt, z.B. ob die Menschen rauchen, wie gesund sie sich ernähren, wie sportlich sie sind etc. Vollkommen ausgelassen werden jedoch genetische Variablen, die bekanntermaßen ebenfalls einen Einfluss auf die Lebenserwartung haben können. Dies führt zu einer Verzerrung des Modells, da es genetische Komponenten außen vor lässt, obwohl diese einen entscheidenden Faktor auf die eigentliche Lebenserwartung haben.

David Schader

2.3 Algorithmische Fairness

Neben den bisher behandelten Thematiken Machine Learning und Bias, geht es nun um die *algorithmische Fairness F*, die im Folgenden nur noch als *Fairness* bezeichnet wird. Dabei stellt sich die Frage, was unter diesem Begriff zu verstehen ist.

Zunächst ist es wichtig zu betonen, dass die Fairness nicht mit einer Metrik zur klassischen Bewertung eines ML-Algorithmus wie der Genauigkeit verwechselt werden darf. Während die in Unterabschnitt 2.1.2 dargestellten Metriken zur allgemeinen Bewertung von ML-Algorithmen dienen, können mit der Fairness beispielsweise Diskriminierungseigenschaften eines ML-Algorithmus gezeigt werden. Eine genaue und eindeutige Definition der Fairness gibt es nicht, es gibt jedoch mehrere spezifische Ansätze zur Definition der Fairness, auf die im weiteren Verlauf weiter eingegangen wird. Neben diesen spezifischeren Ansätzen gibt es jedoch auch die Unterscheidung in das *disparate treatment*, zu Deutsch die ungleiche Behandlung, und *disparate impact*, zu Deutsch die ungleiche Auswirkung. Ersteres ist der Fall, wenn die Entscheidungen zumindest zum Teil auf sogenannten sensitiven Attributen beruhen, es handelt sich dabei um direkte Diskriminierung. Der *disparate impact* bezieht sich dabei mehr auf eine indirekte Diskriminierung; Objekte einer geschützten Klasse mit sensiblen Attributen sind überproportional negativ vom Modell beeinflusst. [11]

Unfaire Modelle setzen meist einen Bias in den Trainingsdaten voraus. Bestimmte Gruppen G werden also diskriminiert. Eine Gruppe wird dabei durch ein oder mehrere sensitive Attribute S definiert. Diese können von Fall zu Fall anders aussehen und werden vom Nutzer festgelegt oder zum Teil im Rahmen ethischer und politischer Diskussionen klar definiert. [7] Im englischen Sprachgebrauch werden diese Attribute je nach Betrachtungsweise entweder als *sensitive* oder als *protected* bezeichnet. Im Folgenden wird ausschließlich der Begriff *sensitive* verwendet. Zusätzlich kann man Gruppen in privilegierte, also jene, die eher positiv klassifiziert werden, und unprivilegierte Gruppen, also jene, die eher negativ klassifiziert werden, unterteilen.

Der Begriff *Fairness* sollte außerdem nicht direkt mit dem Begriff *Bias* verwechselt werden. Während der Bias eine Verzerrung in den zugrundeliegenden Daten beschreibt, bezieht sich die Fairness mehr auf die Modelle und Algorithmen, die bestimmte Gruppen benachteiligen bzw. unfair behandeln können. Jedoch folgt aus einem deutlichen Bias in den Daten meist auch ein unfairer Algorithmus.

Ein Großteil der Fairness-Definitionen gehören zu den gruppenbasierten Fairness-Metriken. Unterschieden werden kann dabei in die *Independenz*, *Separation* und *Suffizienz*. [12] Zur Vereinfachung wird im Folgenden nur eine binäre Klassifikation betrachtet.

David Schader

2.3.1 Independenz

Mit der Independenz wird allgemein die statistische Unabhängigkeit des Scoring-Ergebnisses Z von der zugehörigen Gruppe G ausgedrückt. Es gilt also:

$$Z \perp\!\!\!\perp G \tag{2.15}$$

In Bezug auf die Metriken gehört die Independenz zu den paritätsbasierten Metriken. Zu diesen gehören zum einen die demographische bzw. statistische Parität und zum anderen auch der *Disparate Impact*. [12]

Demographische/Statistische Parität

Bei dieser Metrik geht es darum, dass die Wahrscheinlichkeit, positiv klassifiziert zu werden, unabhängig von der Gruppenzugehörigkeit ist.

$$P(\hat{Y} = 1 \mid S = s_1) = P(\hat{Y} = 1 \mid S = s_2) \quad \forall s_1, s_2 \in S \tag{2.16}$$

Hierbei wird also nicht das tatsächliche Label betrachtet, sondern nur die positive Klassifikation. Diese muss unabhängig vom sensitiven Attribut sein und damit für alle Gruppen gleich.
Bei der Verwendung dieser Metrik sollte jedoch beachtet werden, dass Unterschiede zwischen verschiedenen Gruppen nicht bemerkt werden. [12]

Disparate Impact

Im Vergleich zur statistischen Parität wird hierbei das Verhältnis zwischen zwei Gruppen betrachtet.

$$\frac{P(\hat{Y} = 1 \mid S = s_1)}{P(\hat{Y} = 1 \mid S = s_2)} \quad \forall s_1, s_2 \in S \tag{2.17}$$

Mit dieser Metrik können also Unterschiede zwischen zwei Gruppen genauer beurteilt werden. Dabei gibt es z.B. eine 80%-Regel, bei der Metrik sollte also mindestens der Wert 0,8 vorhanden sein, damit der Algorithmus fair ist. [12]

David Schader

2.3.2 Separation

Im Gegensatz zur Independenz wird bei der Separation auch das tatsächliche Label betrachtet. Ein Algorithmus ist also fair, wenn das Scoring-Ergebnis Z unabhängig ist von der Gruppe G, welche von der tatsächlichen Klassenzugehörigkeit Y abhängig ist.

$$Z \perp\!\!\!\perp G \mid Y \tag{2.18}$$

Bei den Metriken der Separation kann man auch von Konfusionsmatrix-basierten Metriken sprechen. Allgemein werden dabei Gruppen verglichen, bei denen mögliche Unterschiede zugrunde liegen. [12]

Equal Opportunity

Bei der *Equal Opportunity* wird ein Algorithmus als fair betrachtet, wenn die Wahrscheinlichkeit für ein TP-Ergebnis bzw. die TPR bei unterschiedlichen Gruppen gleich ist. [12]

$$P(\hat{Y} = 1 \mid Y = 1 \mid S = s_1) = P(\hat{Y} = 1 \mid Y = 1 \mid S = s_2) \quad \forall s_1, s_2 \in S \tag{2.19}$$

Für beide Gruppen muss also die gleiche TPR vorliegen. Für zwei unterschiedliche Gruppen G_1 und G_2 mit sensitiven Attributen sollte also gelten:

$$TPR(G_1) = TPR(G_2) \tag{2.20}$$

Equalized Odds

Zusätzlich zur betrachteten TPR wie bei der Metrik *Equal Opportunity* wird bei *Equalized Odds* auch die FPR betrachtet. [12]

$$\begin{aligned} &P(\hat{Y} = 1 \mid Y = 1 \mid S = s_1) = P(\hat{Y} = 1 \mid Y = 1 \mid S = s_2) \wedge \\ &P(\hat{Y} = 1 \mid Y = 0 \mid S = s_1) = P(\hat{Y} = 1 \mid Y = 0 \mid S = s_2) \quad \forall s_1, s_2 \in S \end{aligned} \tag{2.21}$$

In der Gruppenschreibweise würde also folgendes gelten:

$$TPR(G_1) = TPR(G_2) \ \wedge \ FPR(G_1) = FPR(G_2) \tag{2.22}$$

Overall accuracy equality

Bei dieser Metrik wird allgemein auf die gesamte *Accuracy* geschaut. Diese ergibt sich jeweils aus der Summe der TPR und TNR. [12]

$$\begin{aligned} &P(\hat{Y} = 0 \mid Y = 0 \mid S = s_1) + P(\hat{Y} = 1 \mid Y = 1 \mid S = s_2) \\ &= P(\hat{Y} = 0 \mid Y = 0 \mid S = s_1) + P(\hat{Y} = 1 \mid Y = 1 \mid S = s_2) \quad \forall s_1, s_2 \in S \end{aligned} \tag{2.23}$$

In der Gruppenschreibweise könnte man folgende beiden Schreibweisen verwenden:

$$TNR(G_1) + TPR(G_1) = TNR(G_2) + TPR(G_1) \tag{2.24}$$

$$Accuracy(G_1) = Accuracy(G_2) \tag{2.25}$$

Conditional use accuracy equality

Im Gegensatz zur *Overall accuracy equality* wird bei der *Conditional use accuracy equality* nicht die ganze Genauigkeit betrachtet. Jedoch soll sowohl die TPR als auch die TNR bei unterschiedlichen sensiblen Gruppen gleichbleiben. Diese werden aber nicht zusammengefasst, sondern in zwei Bedingungen, die *und*-verknüpft sind, eingeteilt. [12]

$$\begin{aligned} &P(\hat{Y} = 1 \mid Y = 1 \mid S = s_1) = P(\hat{Y} = 1 \mid Y = 1 \mid S = s_2) \wedge \\ &P(\hat{Y} = 0 \mid Y = 0 \mid S = s_1) = P(\hat{Y} = 0 \mid Y = 0 \mid S = s_2) \quad \forall s_1, s_2 \in S \end{aligned} \tag{2.26}$$

In Gruppenschreibweise wäre die *Conditional use accuracy equality* daher wie folgt:

$$TPR(G_1) = TPR(G_2) \ \wedge \ TNR(G_1) = TNR(G_2) \tag{2.27}$$

Treatment equality

Mit der *Treatment equality* bzw. Gleichbehandlung gilt ein Algorithmus als fair, wenn bei zwei Gruppen das Verhältnis von der FPR zur FNR gleichbleibt. [12]

$$\frac{P(\hat{Y} = 1 \mid Y = 0 \mid S = s_1)}{P(\hat{Y} = 0 \mid Y = 1 \mid S = s_1)} = \frac{P(\hat{Y} = 1 \mid Y = 0 \mid S = s_2)}{P(\hat{Y} = 0 \mid Y = 1 \mid S = s_2)} \quad \forall s_1, s_2 \in S \tag{2.28}$$

In Gruppenschreibweise gilt also:

$$\frac{FPR(G_1)}{FNR(G_1)} = \frac{FPR(G_2)}{FNR(G_2)} \tag{2.29}$$

Equalizing disincentives

Auch bei dieser Metrik werden zwei ML-Maße, die TPR und FPR, mit einbezogen. Die Differenz dieser beiden Maße soll, damit der Algorithmus fair ist, bei verschiedenen Gruppen gleich sein. [12]

$$\begin{aligned} & P(\hat{Y} = 1 \mid Y = 1 \mid S = s_1) - P(\hat{Y} = 1 \mid Y = 0 \mid S = s_2) \\ = & P(\hat{Y} = 1 \mid Y = 1 \mid S = s_1) - P(\hat{Y} = 1 \mid Y = 0 \mid S = s_2) \quad \forall s_1, s_2 \in S \end{aligned} \tag{2.30}$$

Die Gruppenschreibweise sieht daher wie folgt aus:

$$TPR(G_1) - FPR(G_1) = TPR(G_2) - FPR(G_2) \tag{2.31}$$

David Schader

2.3.3 Suffizienz

Die Suffizienz beschreibt die statistische Unabhängigkeit der Klassenzugehörigkeit Y von der Gruppenzugehörigkeit G abhängig vom Scoring-Ergebnis Z.

$$Y \perp\!\!\!\perp G \mid Z \tag{2.32}$$

Die Suffizienz beschreibt kalibrierungsbasierte Metriken. Es wird also zusätzlich die Wahrscheinlichkeit der Vorhersage berücksichtigt. [12]

Test fairness

Mit der *Test fairness*, auch bekannt unter den Bezeichnungen *matching conditional frequencies* oder *calibration*, ist ein Algorithmus fair, wenn für zwei unterschiedliche Gruppen mit sensiblen Attributen die Wahrscheinlichkeit für ein positives Ergebnis gleich ist, wenn auch das berechnete Scoring-Ergebnis Z gleich ist. [12]

$$P(Y = 1 \mid Z = z \mid S = s_1) = P(Y = 1 \mid Z = z \mid S = s_2) \quad \forall s_1, s_2 \in S; \forall z \in Z \tag{2.33}$$

Zum Beispiel könnte es einen Algorithmus geben, der ausgeben soll, ob Menschen nach einer begangenen Straftat ein weiteres Mal eine Straftat begehen, und der bei seinen Eingangsdaten auch das sensitive Feature Geschlecht besitzt. Nun ergibt das Modell bei einer Frau und bei einem Mann als Ergebnis bzw. als Scoring-Ergebnis eine $0,8$. Dabei sollten beide gleichbetrachtet in die Kategorie eingestuft werden, dass sie eine

weitere Straftat begehen können, da der Schwellenwert z.B. bei 0,7 liegt. Unabhängig vom Geschlecht werden die Personen mit dem jeweiligen Scoring-Ergebnis fest einer Gruppe zugeordnet.

Well calibration

Die *Well calibration* ist eine Erweiterung der eben erläuterten *calibration*. Die Wahrscheinlichkeit, der positiven Klasse zugeordnet zu werden, soll dabei auch dem Scoring-Ergebnis Z entsprechen. [12]

$$P(Y = 1 \mid Z = z \mid S = s_1) = P(Y = 1 \mid Z = z \mid S = s_2) = z \quad \forall s_1, s_2 \in S; \forall z \in Z \tag{2.34}$$

David Schader

3 Methodik und Vorgehensweise

In diesem Kapitel sollen verschiedene Methoden vorgestellt werden, mit denen Verzerrung in Daten erkannt und korrigiert werden kann. Aus diesen lässt sich anschließend eine Vorgehensweise beschreiben, die als Basis für die konzeptuelle Entwicklung eines Debiasing-Frameworks dienen soll. Dazu wird zunächst die allgemeine Herangehensweise skizziert und anschließend werden die einzelnen Schritte auf dem Weg zu einem fairen ML-Algorithmus vorgestellt und erläutert.

Denis Dengler

3.1 Allgemeine Vorgehensweise

Der erste Schritt, um Verzerrung in Daten und Modellen erkennen zu können, ist die Festlegung sensitiver Attribute. Unter Berücksichtigung dieser können diverse Metriken hinsichtlich Bias und Fairness angewandt werden und so Daten und Modelle debiased werden. Prinzipiell lässt sich der ML-Prozess in drei Phasen unterteilen, die eine unterschiedliche Herangehensweise erfordern. Das **Preprocessing** beschreibt den Prozess der Datenvorbereitung. Dies beinhaltet unter anderem die Untersuchung auf fehlende oder invalide Daten innerhalb der Trainingsdaten. Weiter ist es notwendig, im Rahmen der Datenvorbereitung nominale bzw. kategorielle Werte in numerische Werte zu kodieren, um ML-Algorithmen anwenden zu können. Ein weiterer Teil des Preprocessings ist die Untersuchung auf Verzerrungen (*Bias*). Dazu müssen Methoden und Metriken definiert werden, mit denen eine Verzerrung in den Daten detektiert und bewertet werden kann. Zusätzlich müssen Verfahren entwickelt werden, mit denen eine solche Verzerrung korrigiert werden kann (*Debiasing*). Die zweite Phase wird **Inprocessing** genannt und beinhaltet das Trainieren des Algorithmus mit den idealerweise bereits korrigierten Trainingsdaten, die anschließende Bewertung hinsichtlich Fairness für einzelne aus den sensitiven Attributen hervorgehende Gruppen und, falls nötig, die Korrektur des unfairen Verhaltens. Die dritte und letzte Phase ist das **Postprocessing**, welches die Nachbearbeitung und Aufbereitung der Ergebnisse umfasst. Dabei wird ein bereits trainiertes Modell, welches womöglich biased gegenüber einem oder mehreren sensitiven Attributen ist, bewertet und korrigiert. [13, 14] Im Rahmen dieser Arbeit wird der Fokus auf das Preprocessing und die anschließende Bewertung der trainierten Modelle gelegt, sodass auf das Postprocessing nicht näher eingegangen wird.

Denis Dengler

3.2 Datenvorbereitung

In diesem Abschnitt werden die einzelnen Schritte der Datenvorbereitung beschrieben und erläutert. Dabei soll vor allem auf die Gründe für fehlende Werte und den Umgang mit diesen eingegangen werden. Zusätzlich soll kurz der Kodierungsvorgang nominaler Attributwerte vorgestellt werden.

Denis Dengler

3.2.1 Umgang mit fehlenden Werten

Ein Datensatz ist selten perfekt. Es muss immer davon ausgegangen werden, dass ein Datensatz unvollständig oder inkorrekt ist, das heißt, es fehlen Einträge für bestimmte Attribute oder die entsprechenden Werte sind ungültig. Um diesen Fehler zu beheben, soll der Datensatz zunächst auf fehlende oder potentiell ungültige Daten untersucht werden. Anschließend muss definiert werden, wie mit diesen Daten umgegangen werden soll. Dafür stehen mehrere Möglichkeiten zur Verfügung, von denen eine Auswahl im Folgenden vorgestellt werden soll. Zunächst ist es jedoch wichtig zu betrachten, aus welchen Gründen Daten fehlen.

Denis Dengler

Gründe für fehlende Daten

Es kann viele verschiedene Gründe geben, warum bestimmte Attributwerte fehlen. Allgemein lassen sich diese in drei Kategorien einteilen.

1. **Missing Completely at Random (MCAR)** bedeutet, dass das Fehlen eines Attributwerts in keiner Weise mit den fehlenden oder anderen Werten zusammenhängt.
2. **Missing at Random (MAR)** bedeutet, dass das Fehlen eines Attributwerts zwar nicht mit dem fehlenden Wert zusammenhängt, jedoch ein Zusammenhang mit anderen Werten besteht.
3. **Missing not at Random (MNAR)** bedeutet, dass das Fehlen eines Attributwerts mit dem fehlenden Wert zusammenhängt.

Diese Unterscheidung ist wichtig, da fehlende Werte der dritten Kategorie nicht ignoriert werden können und damit keine Vervollständigung der Daten (*Imputation*) möglich ist. [15, 16]

Mögliche Lösungen

Nach Betrachtung der möglichen Gründe für fehlende Daten, sollen nun mögliche Lösungsansätze diskutiert werden.

1. **Zeilen löschen.** Abhängig von der Anzahl fehlender Attributwerte kann abgewägt werden, ob ein Datenobjekt gelöscht werden soll. Dies hat den Vorteil, dass man nur mit (fast) vollständigen Datenobjekten arbeitet, was die Robustheit des ML-Algorithmus steigert. Auf der anderen Seite kann dieses Vorgehen bei übermäßiger Häufigkeit zu einer starken Datenreduktion und damit einem hohen Informationsverlust führen. Der Schwellenwert, ab dem ein Datenobjekt gelöscht wird, sollte deshalb wohlüberlegt gewählt werden. [17] Sollten Datenobjekte gelöscht werden, welche der zweiten Kategorie angehören und speziell im Zusammenhang mit einem sensitiven Attribut stehen, könnte dies zu einem Representation Bias und damit zur Diskriminierung einer Gruppe führen.

2. **Spalten löschen.** Fehlen vermehrt Werte eines Attributs, kann überlegt werden, ob die zugehörige Spalte des Datensatzes gelöscht werden soll. Dies sollte jedoch nur in Betracht gezogen werden, wenn ein signifikanter Anteil ($> 60\%$) fehlt oder ungültig ist und dieses Attribut für die Vorhersage von geringer Relevanz bzw. Aussagekraft ist. [16] Zudem kann das Löschen einer Spalte einen Omitted Variable Bias nach sich führen.

3. **Imputation stetiger Variablen mit Median oder Mittelwert.** Ein Ansatz fehlende Werte stetiger Variablen zu vervollständigen, ist die Verwendung des Median oder des Mittelwerts. Der Median und Mittelwert (in dieser Reihenfolge) sind definiert als

$$\bar{x} = \begin{cases} x_{m+1} & \text{für } n = 2m+1 \\ \frac{1}{2}(x_m + x_{m+1}) & \text{für } n = 2m \end{cases} \tag{3.1}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.2}$$

   Diese Vorgehensweise verhindert Daten- bzw. Informationsverlust und ist zudem einfach zu implementieren. Ein großer Nachteil ist die Missachtung einer eventuellen Kovarianz zwischen den Attributen. Zudem funktioniert dieser Ansatz nur mit numerischen und stetigen Werten. [16, 17] Außerdem kann die Überrepräsentation des Mittelwerts/Medians zu einem Representation Bias führen.

4. **Imputation kategorieller Variablen mit dem Modus.** Bei kategoriellen Variablen wird der Modus (auch Modalwert genannt), also der am häufigsten vorkommende Wert, zur Vervollständigung verwendet. Sollte ein großer Teil der Werte fehlen, kann

ein neuer Wert definiert werden. Diese Methode verhindert zwar den Daten- bzw. Informationsverlust und ist einfach umzusetzen, kann jedoch aufgrund eines neuen Werts zu einer sinkenden Performanz führen. [17] Auch hier kann die Überrepräsentation des Modalwert zu einem nicht intendierten Representation Bias führen.

5. **Vorhersage fehlender Werte mit ML-Algorithmen.** Eine weitere Methode, um fehlende Werte zu vervollständigen, ist die Anwendung von ML-Algorithmen zur Vorhersage der fehlenden Werte. Dabei wird das Attribut, welches vervollständigt werden soll, als Label verwendet und aus dem Datensatz entfernt und die Datenobjekte mit vorhandenem Wert als Trainingsdaten genutzt. Mit dem trainierten Modell können anschließend die fehlenden Werte vorhergesagt werden. Ein passender Algorithmus für kategorielle Variablen wäre unter anderem der *K Nearest Neighbors (KNN)*, welcher basierend auf der Distanz zu anderen Datenpunkten den wahrscheinlichsten Wert vorhersagt. Für numerische Variablen käme unter anderem die *Lineare Regression* in Frage. Der Vorteil zur einfachen Imputation liegt in der Mitbeachtung der Kovarianz, während der größte Nachteil darin liegt, dass verzerrte Daten durch die Verwendung von solchen Algorithmen auch zu verzerrten Vorhersagen führen. [16, 17]

Denis Dengler

3.2.2 Kodieren nominaler Attributwerte

Da die Mehrzahl von Klassifizierungsalgorithmen nicht mit nominalen bzw. kategoriellen Werten umgehen kann, ist es notwendig, diese in numerische Werte umzuwandeln. Dafür stehen verschiedene Möglichkeiten bereit. Allgemein können zwei Typen nominaler Werte unterschieden werden. Außerdem werden jene Werte, die in sinnvoller Reihenfolge sortiert werden können, wie etwa die Wörter *erstens, zweitens, drittens*, als *ordinal* bezeichnet und können der Reihe nach durchnummeriert werden. Dieses Verfahren nennt sich *Integer-* oder *Ordinal Encoding*. Für das vorangegangene Beispiel würde es wie folgt aussehen:
erstens $\rightarrow 1$, *zweitens* $\rightarrow 2$, *drittens* $\rightarrow 3$. Alle anderen nominalen Werte können nicht so einfach durch numerische Werte ersetzt werden, da sonst der Schein einer zugrundeliegenden Reihenfolge entstehen würde. Deshalb wird für diese ein anderes Verfahren verwendet, das *One Hot Encoding*. Bei diesem Verfahren wird jedem nominalen Attributwert ein binärer Vektor zugewiesen. Ein Beispiel dafür sind etwa Farbbezeichnungen wie *rot, grün* und *blau*. Diese würden nach dem One Hot Encoding wie folgt kodiert werden:
rot $\rightarrow \begin{pmatrix}1\\0\\0\end{pmatrix}$, *grün* $\rightarrow \begin{pmatrix}0\\1\\0\end{pmatrix}$, *blau* $\rightarrow \begin{pmatrix}0\\0\\1\end{pmatrix}$. [18]

Denis Dengler

3.3 Verzerrung in Daten erkennen und messen

Die einfachste Art und Weise Verzerrungen zu messen ist die Gegenüberstellung der absoluten Anzahl x_i einzelner Attributwerte a_i. Dafür werden für alle sensitiven Attribute $A_{S,i}$ die jeweilige Anzahl der vorkommenden Werte berechnet und beispielsweise in einem Balkendiagramm visualisiert. Als Maß zur Bewertung von Verzerrung kann der empirische Variationskoeffizient

$$v = \frac{s}{\bar{x}} \tag{3.3}$$

mit der empirischen Standardabweichung

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{3.4}$$

und dem empirischen Mittelwert

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.5}$$

herangezogen werden. Dank diesem lässt sich die Varianz als relatives Maß innerhalb eines Attributs berechnen und damit zwischen verschiedenen Attributen vergleichen. Dabei deutet ein hoher empirischer Variationskoeffizient auf einen starken *Representation Bias* hin.

Denis Dengler

3.4 Verzerrung in Daten korrigieren

Nachdem in Abschnitt 3.3 vorgestellt wurde, wie ein Bias erkannt und gemessen werden kann, sollen nun Methoden betrachtet werden, mit denen dieser korrigiert werden kann. Diese sollen jeweils kurz vorgestellt und im Hinblick auf ihre Anwendbarkeit und Wirksamkeit bewertet werden.

Denis Dengler

3.4.1 Sensitive Feature Dropping

Die einfachste Methode Verzerrung zu korrigieren ist die Auslassung der sensitiven Attribute im Lernprozess, also konkret das Entfernen der Attribute aus dem Datensatz. Dieser Ansatz wirkt zwar auf den ersten Blick logisch und vielversprechend, denn wo keine sensitiven Attribute sind, dort gibt es auch keinen Bias. Leider ist es nicht ganz so einfach. Um zu verstehen, wieso das Entfernen der sensitiven Attribute alleine nicht ausreicht, ist es notwendig diese Attribute im Kontext des gesamten Datensatzes zu betrachten. Es

ist davon auszugehen, dass sensitive Attribute mit anderen nicht-sensitiven Attributen korrelieren. Ist dies der Fall, würde das Entfernen der sensitiven Daten nicht nur den Bias nicht entfernen, sondern auch das Auffinden und Bewerten von Verzerrung in den reduzierten Daten um ein vielfaches erschweren. Ist eine Korrelation jedoch auszuschließen, das heißt, alle anderen nicht-sensitiven Attribute sind statistisch unabhängig von den sensitiven Attributen, so ist das Löschen dieser Attribute die effizienteste Lösung. [19]

Denis Dengler

3.4.2 Relabeling / Messaging

Beim Relabeling wird versucht, durch Anpassung der Label innerhalb der Trainingsdaten Fairness und damit eine Unabhängigkeit zwischen Gruppenzugehörigkeit und Klassenzuweisung zu erreichen. Eine in der Literatur häufig angewandte Methode ist das **Messaging**. Dabei werden Elemente des Trainingsdatensatzes mit einem sogenannten *Ranker* R hinsichtlich ihrer Wahrscheinlichkeit einen positiven Ausgang y vorhergesagt zu bekommen sortiert. Anschließend werden jene Elemente, welche der privilegierten Gruppe G_+ angehören, absteigend nach ihrem Ranking-Wert sortiert und jene, welche nicht der privilegierten Gruppe angehören G_- aufsteigend sortiert. Die obersten Kandidaten werden jeweils zum Relabeling ausgewählt, da deren Modifikation den geringsten Einfluss auf die Vorhersagegenauigkeit hat. Diese Anpassung wird solange durchgeführt, bis eine statistische Unabhängigkeit erreicht wurde. Nach [20] kann die Anzahl der notwendigen Modifikationen M mit der folgenden Formel berechnet werden:

$$M = \frac{(g_+ \times g_- \wedge y) - (g_- \times g_+ \wedge y)}{g_+ + g_-} \tag{3.6}$$

wobei g_+ die Anzahl der zu einer privilegierten Gruppe G_+ zugehörigen Objekte beschreibt und g_- die Anzahl derjenigen, die nicht zu dieser Gruppe gehören. $g_+ \wedge y$ bedeutet, dass das Objekt der Gruppe G_+ angehört **und** das Label y besitzt ($g_- \wedge y$ analog). [13, 19, 20] Dieser Ansatz hat leider den großen Nachteil, dass die zugrundeliegenden Trainingsdaten verändert werden und es sich somit um einen intrusiven Ansatz handelt.

Denis Dengler

3.4.3 Reweighting und Resampling

Reweighting und Resampling verfolgen einen anderen Ansatz. Statt die Label in den Trainingsdaten anzupassen, wird Elementen, welche einer bestimmten Gruppe $g_i \in G$ angehören, ein Gewicht w_i zugewiesen, welches sich, abhängig von der gewählten Fairness-Metrik (in diesem Fall *Demographische Parität*), wie folgt berechnet [21]:

$$w_i = \frac{P(\hat{Y} = y_i)P(G = g_i)}{P(\hat{Y} = y_i, G = g_i)} = \frac{P(\hat{Y} = y_i)}{P(\hat{Y} = y_i \mid G = g_i)} \tag{3.7}$$

Das Gewicht entspricht also dem Verhältnis zwischen dem Anteil der der Klasse y_i zugewiesenen Elemente und dem Ergebnis der gewählten Fairness-Metrik für eine Gruppe g_i. Diese Gewichte erlauben es, unterrepräsentierte Gruppen im Lernprozess stärker zu gewichten oder, falls der verwendete Lernalgorithmus keine Gewichte unterstützt, die Trainingsdaten zu resamplen.[13, 19–21]

Denis Dengler

3.5 Resampling-Verfahren

Es gibt verschiedene Verfahren, mit denen man Daten resamplen kann. Dabei unterscheidet man zunächst zwischen Under- und Oversampling. Diese beiden Methoden werden im weiteren Verlauf näher erläutert. Allgemein versteht man unter Resampling ein Verfahren, mit dem die Menge von Datenobjekten der überrepräsentierten Klasse Y_+, im Folgenden *Majorität* genannt, verringert, bzw. die der unterrepräsentierten Klasse Y_-, im Folgenden *Minorität* genannt, vergrößert wird.

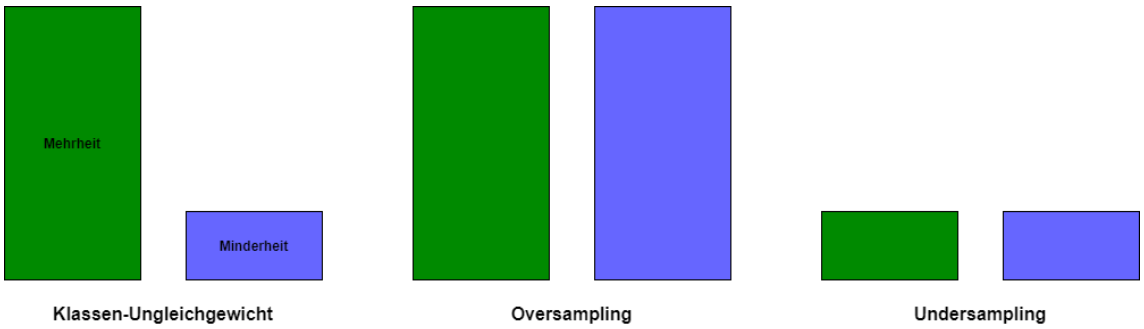


Abbildung 3.1: Vergleich Over- und Undersampling

Denis Dengler

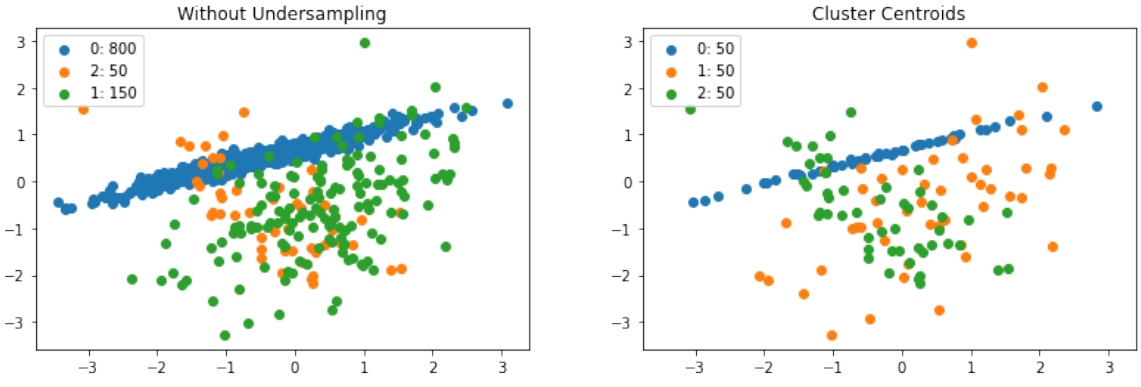
3.5.1 Undersampling

Undersampling beschreibt einen Datenreduktionsansatz, bei dem die Anzahl der in einer Trainingsdatenmenge vorkommenden Datenobjekte, welche einer Majorität angehören, reduziert wird. Je nach Algorithmus kann auch ein gewünschtes Verhältnis zwischen einzelnen Klassen festgelegt werden. Für diesen Ansatz gibt es verschiedene Methoden, mit

denen die zu löschenden Datenobjekte ausgewählt werden können. Man unterscheidet dabei zunächst zwischen generierenden und selektierenden Algorithmen. Bei den generierenden Algorithmen wird eine neue Datenmenge X' anhand der vorliegenden Trainingsdaten X erzeugt, sodass $|X'| < |X| \wedge X' \not\subset X$ gilt. Ein Beispiel für diesen Typ ist das *Cluster Centroids*-Verfahren, auf welches im weiteren Verlauf eingegangen wird. Im Gegensatz dazu bilden selektierende Algorithmen eine neue Datenmenge X', indem Datenobjekte aus der Originaldatenmenge X selektiert werden, sodass $|X'| < |X| \wedge X' \subset X$ gilt. Dieser Typ lässt sich weiter in zwei Kategorien unterteilen: *kontrolliertes Undersampling* und *Cleaning Undersampling*. Der größte Unterschied zwischen diesen beiden Kategorien besteht darin, dass beim kontrollierten Undersampling die gewünschte Anzahl an Datenobjekten vorgegeben werden kann, während die Cleaning-Algorithmen eine eigene Heuristik verwenden, die eine solche Spezifikation nicht erlaubt. Eine Auswahl verschiedener Algorithmen soll im Folgenden vorgestellt werden. [22]

Cluster Centroids (CC)

Cluster Centroids ist ein Beispiel für generierende Undersampling-Algorithmen. Dieses basiert auf dem K-Means Algorithmus, bei dem N zufällig positionierte Punkte, sogenannte Zentroiden, erzeugt werden, die jeweils einem Cluster von Datenobjekten zugeordnet werden, zu dessen Zentrum der Punkt am nächsten liegt. Ausgehend von dieser Zuordnung werden die Zentroiden neu berechnet. Dieser Vorgang wird solange wiederholt, bis sich die Position der Zentroiden nicht mehr ändert. Die erzeugten Zentroiden stellen die neuen reduzierten Daten dar. Bei diesem Verfahren lässt sich die gewünschte Anzahl an Datenpunkten N konkret festlegen. [23–25]

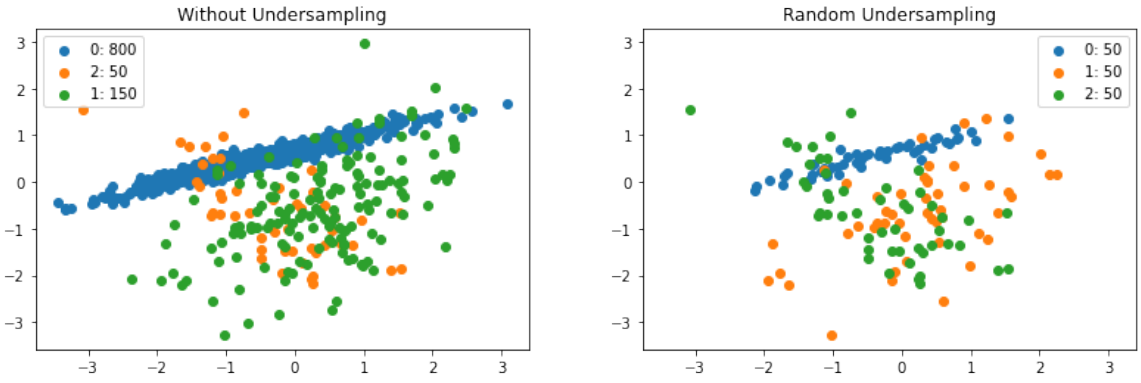


(a) Original (b) ClusterCentroids

Abbildung 3.2: Cluster Centroids

Random Undersampling

Unter Random Undersampling versteht man ein Vorgehen, bei dem zufällig Datenobjekte einer Majorität Y_+ gelöscht werden, bis das gewünschte Verhältnis erreicht wurde. Damit gehört es zu den kontrollierten Undersampling-Verfahren. Dieser Ansatz ist sehr simpel und schnell, kann jedoch dazu führen, dass Datenobjekte gelöscht werden, die für eine präzise Vorhersage von großer Relevanz sind. [22]



(a) Original (b) RandomUnderSampler

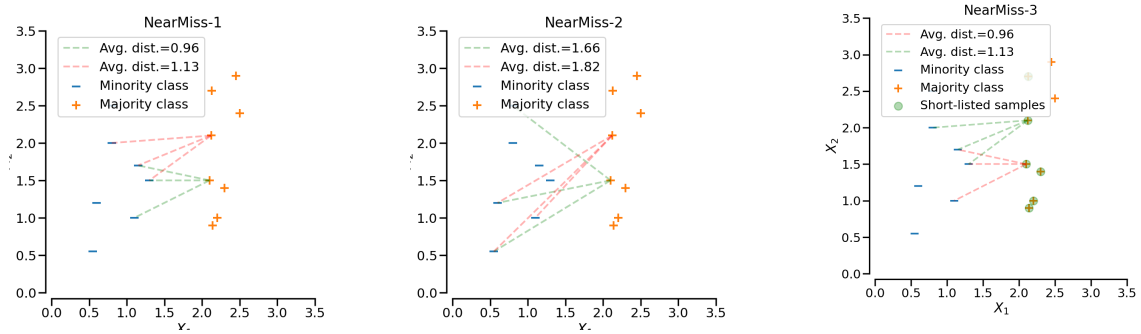
Abbildung 3.3: Random Undersampling

Near Miss

NearMiss ist ein Nearest Neighbors Algorithmus, das heißt, er berechnet die Distanz zu den umliegenden Nachbarn. Diese Methode gehört ebenfalls zur ersten Kategorie. Bei der Distanzberechnung gibt es drei Versionen (vgl. Abbildung 3.4). Bei Version 1 werden diejenigen privilegierten Datenobjekte gelöscht, deren durchschnittlicher Abstand zu den N nächstgelegenen unprivilegierten Datenobjekten am geringsten ist. Analog wird bei Version 2 die Distanz zu den N am weitesten gelegenen unprivilegierten Datenobjekten berechnet. NearMiss Version 3 verwendet einen zweistufigen Algorithmus, bei dem sich zunächst für jedes Datenobjekt der Minorität Y_- die M nächstgelegen Datenobjekte der Majorität Y_+ gemerkt werden und anschließend diejenigen ausgewählt werden, deren durchschnittlicher Abstand zu den N nächstgelegenen Nachbarn am größten ist. [22, 26, 27]

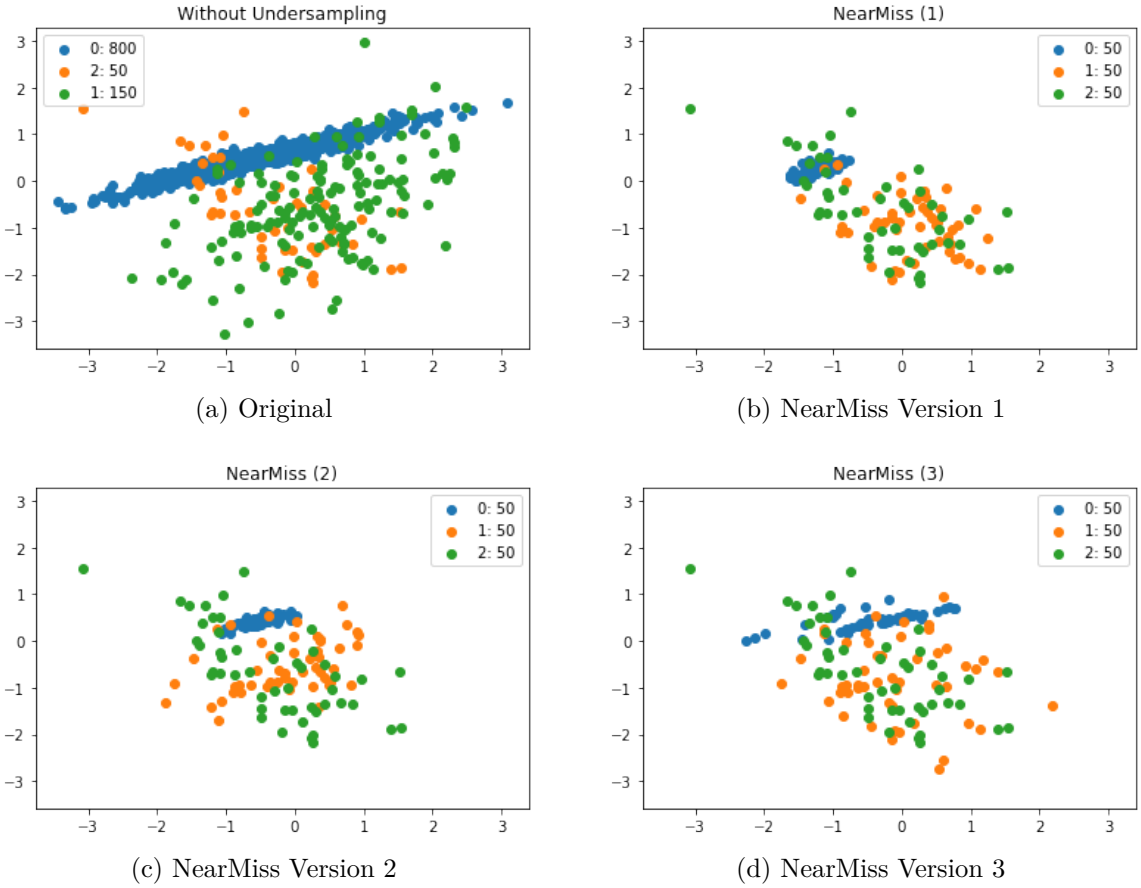
TomekLinks

TomekLinks ist das erste Cleaning Undersampling Verfahren. Um zu verstehen, wie diese Methode funktioniert, ist es zunächst notwendig zu klären, was ein *Tomek Link* ist. Ein Tomek Link ist definiert als ein Paar von Datenobjekten unterschiedlicher Klassen, welche



(a) Version 1 (b) Version 2 (c) Version 3

Abbildung 3.4: Near Miss Distanzberechnung Version 1-3 [22]



(a) Original (b) NearMiss Version 1

(c) NearMiss Version 2 (d) NearMiss Version 3

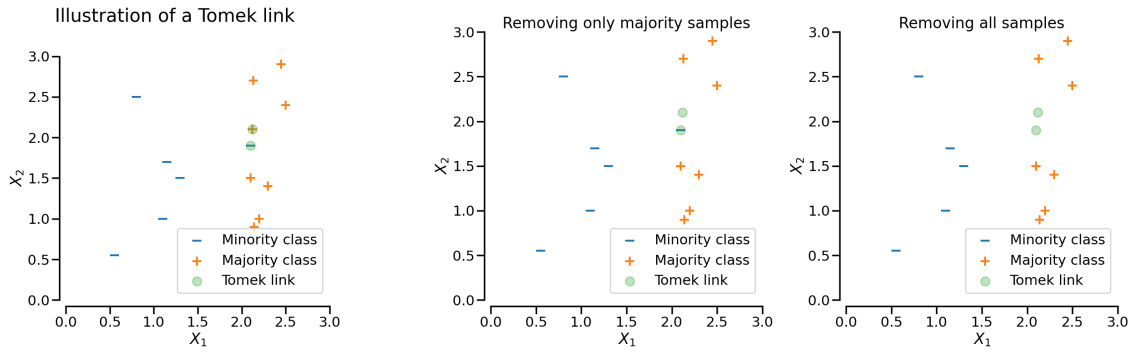
Abbildung 3.5: NearMiss Vergleich Version 1-3

zueinander die kürzeste Distanz haben, das heißt, die Datenpunkte sind sich gegenseitig die nächsten Nachbarn. Diese Beziehung lässt sich mit dem mathematischen Ausdruck

$$d(X_1, X_2) < d(X_1, X_3) \wedge d(X_1, X_2) < d(X_2, X_3) \tag{3.8}$$

beschreiben, wobei d die Distanz zwischen zwei Datenobjekten beschreibt und X_1, X_2 Datenobjekte verschiedener Klassen der Datenmenge sind und X_3 ein beliebiges drittes Datenobjekt derselben Datenmenge ist.

Bei diesem Verfahren werden alle Datenobjekte der Majorität gelöscht, die eine solche Beziehung zu einem Element der Minorität haben, es werden also alle Tomek Links aufgelöst. Optional können die unprivilegierten Datenobjekte ebenfalls gelöscht werden. Dieser Vorgang ist in Abbildung 3.6 illustriert. [28]



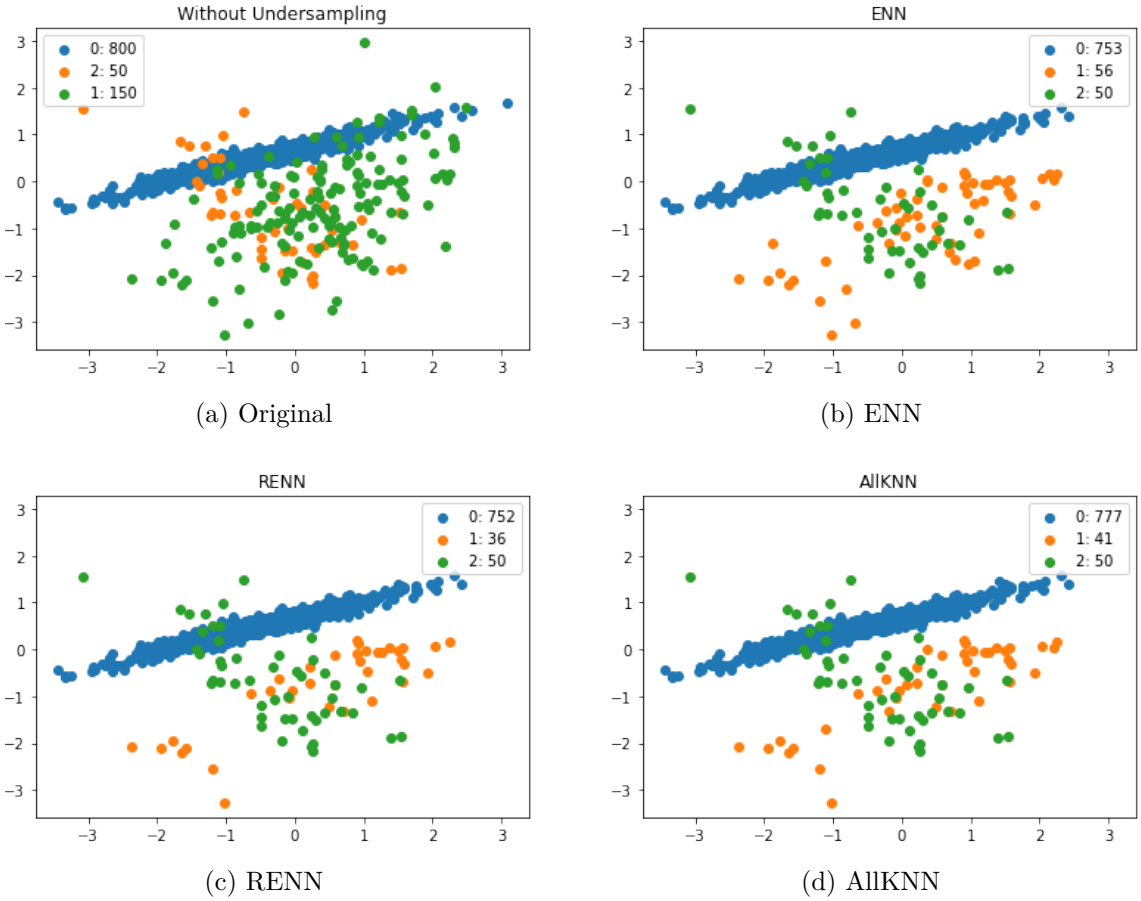
(a) Tomek's link

(b) Auflösen von Tomek's links

Abbildung 3.6: TomekLinks Verfahren [22]

Edited Nearest Neigbours (ENN)

Der Edited Nearest Neigbours Algorithmus basiert auf dem Nearest Neighbors Algorithmus, erweitert diesen jedoch um eine Anpassung, bei der Datenobjekte, welche nicht ausreichend mit der Nachbarschaft übereinstimmen, gelöscht werden. Dabei werden für jedes Datenobjekt aus der Majorität die Nearest Neighbors berechnet und überprüft, ob das betrachtete Datenobjekt derselben Klasse wie die Nachbarn angehört. Ist dies nicht der Fall, wird das Datenobjekt entfernt. [29] Eine Erweiterung dieses Algorithmus ist das *Repeated Edited Nearest Neighbors* (RENN)-Verfahren, bei dem der beschriebene Prozess mehrfach wiederholt wird. Eine weitere Adaption dessen ist das *AllKNN*-Verfahren, welches zusätzlich pro Iteration die Anzahl der zu betrachtenden Nachbarn erhöht. [30]



(a) Original (b) ENN

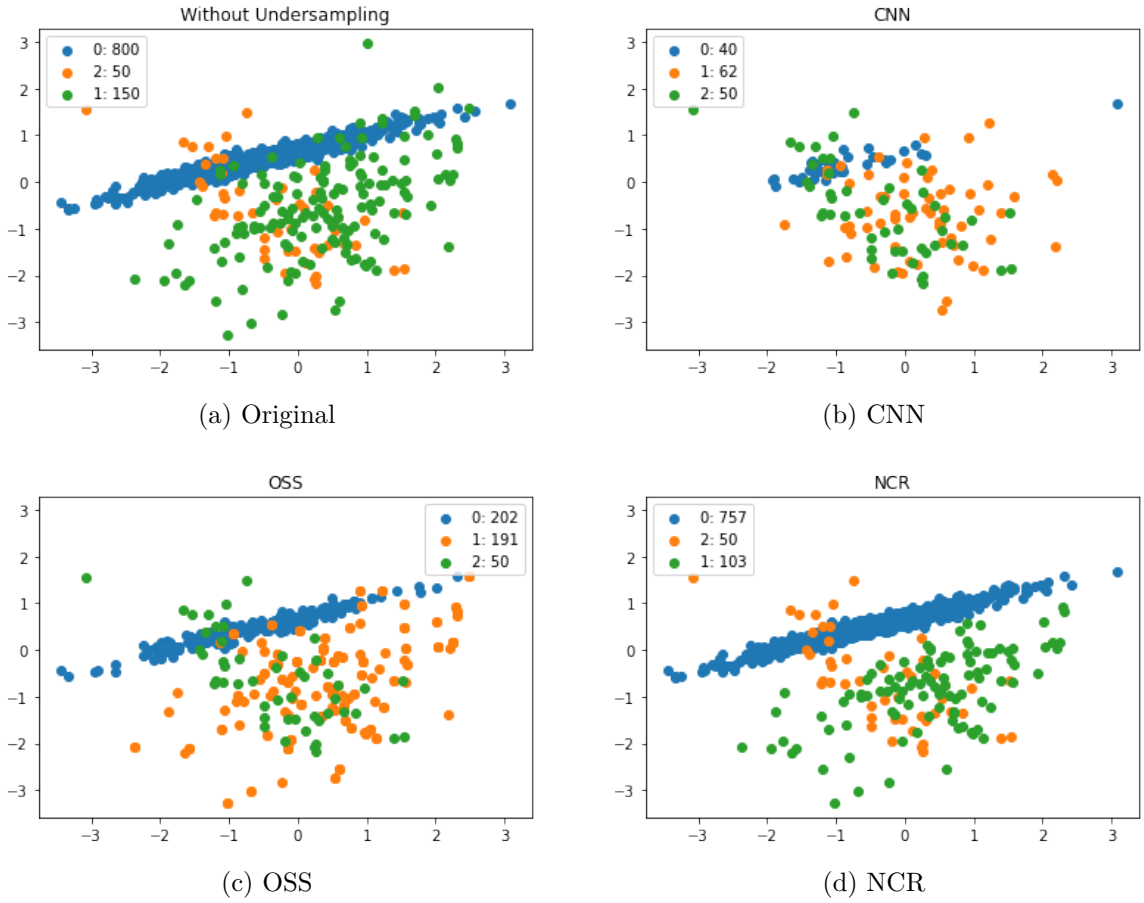
(c) RENN (d) AllKNN

Abbildung 3.7: Vergleich ENN, RENN und AllKNN

Condensed Nearest Neighbors (CNN)

CNN wurde ursprünglich 1968 von Peter Hart zur Datenreduktion für den K Nearest Neighbors (KNN)-Algorithmus entwickelt. Dabei wird nur ein Nearest Neighbor verwendet, mit dem iterativ entschieden wird, ob ein Datenobjekt übernommen wird. Bei diesem Verfahren wird zunächst die Minorität aus der Trainingsdatenmenge X in eine Ergebnismenge R kopiert. Über die anderen Datenobjekte der Menge X wird iteriert und für jedes Element der Nearest Neighbor berechnet. Dabei wird die Klasse des Nachbars mit der eigenen verglichen. Wenn sich die Klassen unterscheiden, wird das aktuelle Datenobjekt der Ergebnismenge hinzugefügt, anderenfalls geschieht nichts. Dieses Prozedere wird solange wiederholt, bis kein Datenobjekt aus X zu R hinzugefügt werden kann. Es werden genau jene Datenobjekte übernommen, die nicht korrekt klassifiziert werden konnten. Die Ergebnisdatenmenge bildet die neuen Trainingsdaten. [27, 31, 32] Der Nachteil dieses Verfahrens ist, dass sehr nah beieinander liegende Datenobjekte unterschiedlicher Klassen schwer zu unterscheiden sind und zu Problemen führen können. Eine Weiterentwicklung des CNN-Algorithmus, die dieses Problem zu beheben versucht, ist die One Sided Selecti-

on (OSS). Um dies zu erreichen, wird das TomekLinks-Verfahren verwendet, um eben diese Datenobjekte zu entfernen. Ein weiterer Unterschied zum CNN liegt darin, dass alle Datenobjekte gleichzeitig betrachtet werden und deshalb nicht mehrfach über die Datenmenge X iteriert wird. [33] Eine zweite Abwandlung des CNN-Algorithmus ist die Neighborhood Cleaning Rule (NCR). Diese besteht aus dem CNN-Verfahren zum Entfernen redundanter Datenobjekte und dem ENN-Ansatz zum Entfernen von verrauschten Datenobjekten in der Ergebnismenge. [33, 34]



(a) Original (b) CNN

(c) OSS (d) NCR

Abbildung 3.8: Vergleich CNN, OSS und NCR

Denis Dengler

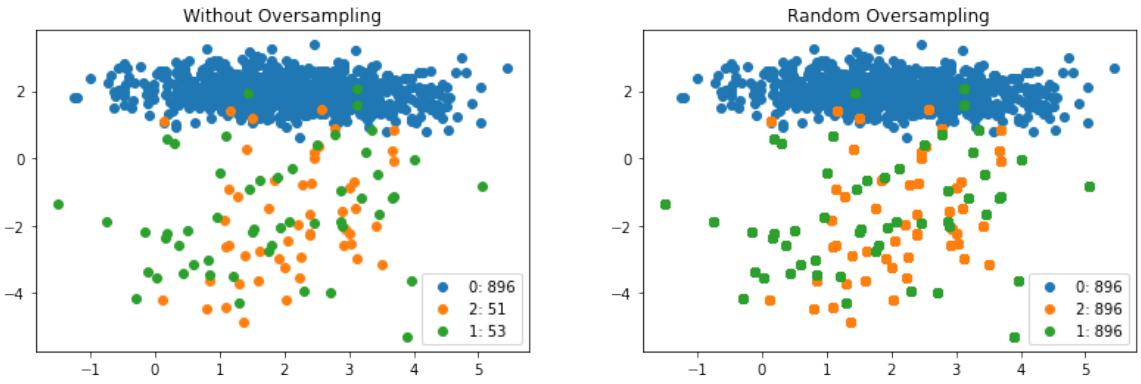
3.5.2 Oversampling

Oversampling beschreibt einen Ansatz, bei dem die Anzahl der in einer Trainingsdatenmenge vorkommenden Datenobjekte, welche einer Minorität angehören, erhöht wird. Dabei kann in der Regel ein gewünschtes Verhältnis zwischen einzelnen Klassen festgelegt werden. Man unterscheidet hierbei allgemein zwischen einer Vervielfachung und einer Interpolation.

Vervielfachung lässt sich so definieren, dass eine neue Datenmenge X' anhand der vorliegenden Trainingsdaten X erzeugt wird, sodass $|X'| > |X| \wedge X \subset X' \wedge \forall X'_i \in X' : X'_i \in X$ gilt. Bei der Interpolation wird eine neue Datenmenge X' erzeugt, indem neue Datenobjekte X'_i zwischen den in der Originaldatenmenge X vorhandenen Datenobjekten erzeugt werden. Dies lässt sich wie folgt mathematisch beschreiben: $|X'| > |X| \wedge X \subset X' \wedge \exists X'_i \in X' : X'_i \notin X$. Die bekanntesten Oversampling-Verfahren sollen im Folgenden vorgestellt werden.

RandomOverSampler

Beim *Random Oversampling* werden solange zufällige Datenobjekte der Minorität dupliziert, bis die gewünschte Menge erreicht worden ist. Das bedeutet, es werden keine neuen Datenobjekte erzeugt. Die Varietät der Daten bleibt gleich und es resultiert kein Informationsgewinn. [35]



(a) Original (b) RandomOverSampler

Abbildung 3.9: Random Oversampling

Synthetic Minority Oversampling TEchnique (SMOTE)

SMOTE ist eine Oversampling-Methode, die neue synthetische Datenobjekte hinzufügt, um so die Menge der Datenobjekte der Minorität zu erhöhen. Dieses Verfahren basiert auf einer Interpolation entlang einer Linie, welche zwischen einem betrachteten Datenobjekt und seinen k Nearest Neighbors gleicher Klasse gespannt wird. Abhängig von dem gewünschten Verhältnis werden m der k Nachbarn zufällig ausgewählt, auf deren Verbindungslinie ein neues Datenobjekt erzeugt wird (vgl. Abbildung 3.10). Die Position des Datenobjekts wird zufällig durch Multiplikation der Länge der Verbindungslinie mit einem zufälligen Faktor zwischen 0 und 1 bestimmt. Ein Nachteil dieses Verfahrens ist, dass Ausreißer (engl. *Outlier*), welche normalerweise missachtet werden sollten, vom Algorithmus erfasst werden

können und das Cluster der Datenobjekte in eine ungewollte Richtung ausgedehnt wird. [36, 37]

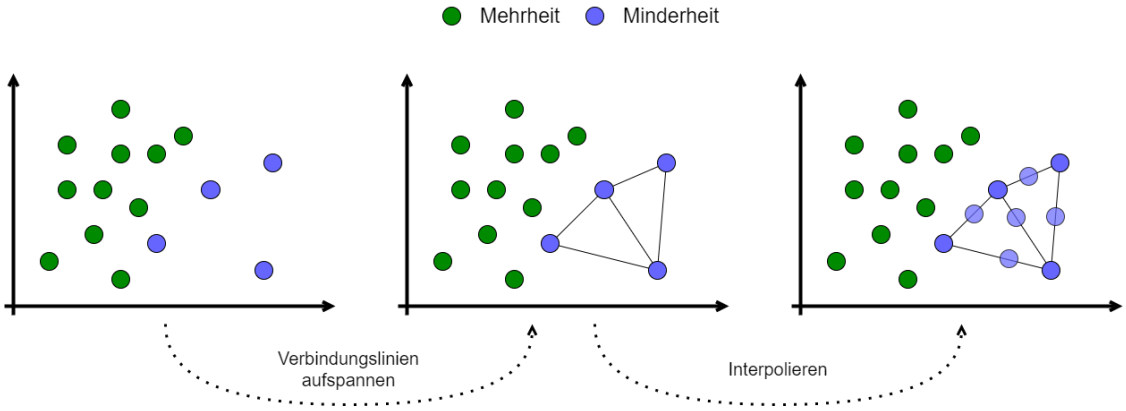


Abbildung 3.10: Generierung neuer Daten bei SMOTE und ADASYN

Varianten von SMOTE

Um SMOTEs Problem mit den Ausreißern zu beheben, verfolgen die Varianten Borderline-SMOTE, SVM-SMOTE und KMeans-SMOTE einen etwas abgewandelten Ansatz. Alle drei Varianten bestimmen zunächst die optimale Grenze zwischen den einzelnen Klassen mit einer Entscheidungsfunktion und erzeugen neue Datenobjekte in der entgegengesetzten Richtung. Zur Berechnung dieser Grenze werden je nach Variante die Methoden *Borderline*, *Support Vector Machine* (SVM) oder *KMeans* angewandt. [36, 38–40]

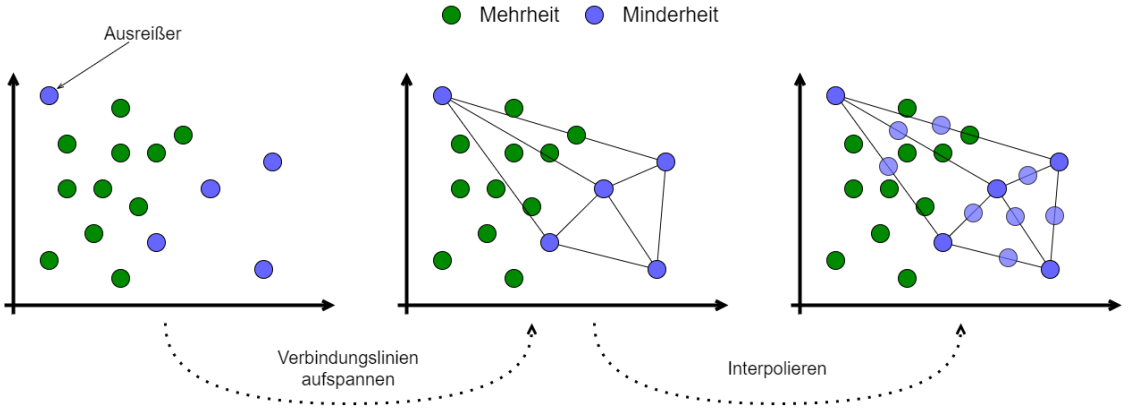
ADAptive SYNthetic (ADASYN) Oversampling

ADASYN ist eine Methode, die ähnlich wie SMOTE neue Datenobjekte durch Interpolation erzeugt. Im Unterschied zum SMOTE-Ansatz werden neue Datenobjekte nicht vollkommen zufällig generiert, sondern orientieren sich an der Dichte umliegender Datenobjekte aus anderen Klassen. Hierfür wird ein Klassenungleichgewicht mit

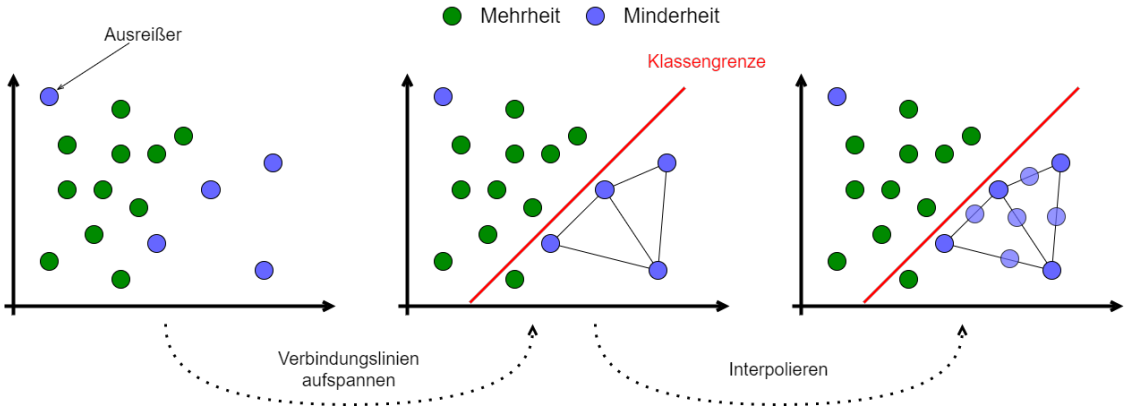
$$d = \frac{|Y_-|}{|Y_+|} \tag{3.9}$$

berechnet, wobei $|Y_-|$ der Anzahl von Datenobjekten aus der Minorität und $|Y_+|$ der Anzahl von Datenobjekten der Majorität entspricht. Zusätzlich wird ein Schwellenwert d_{th} festgelegt. Ist nun $d < d_{th}$, so werden an dieser Stelle neue Datenobjekte erzeugt. Anderenfalls wird ein anderer Nachbar gesucht. [41]

Denis Dengler

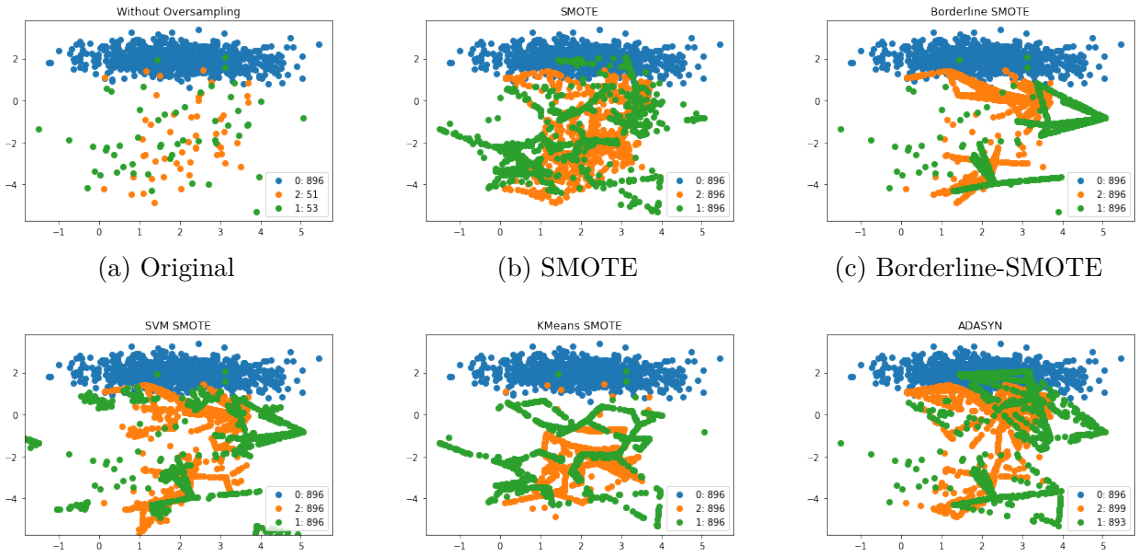


(a) SMOTE mit Ausreißer



(b) SMOTE mit Klassengrenze

Abbildung 3.11: Vergleich SMOTE und Varianten



(a) Original (b) SMOTE (c) Borderline-SMOTE

(d) SVM-SMOTE (e) KMeans-SMOTE (f) ADASYN

Abbildung 3.12: Vergleich SMOTE Varianten und ADASYN

4 Konzept und Implementierung

In diesem Kapitel wird das Konzept sowie die Implementierung bzw. Vorgehensweise für das Debiasing von ML-Algorithmen anhand der Anpassung von Trainingsdaten dargestellt. Am Ende soll ein Framework bzw. eine Python-Bibliothek namens `fairml`, kurz für Fair Machine Learning, stehen, mit der es unter anderem möglich sein soll, die Fairness zu berechnen, die Trainingsdaten von einem Bias zu entzerren und mit der danach berechneten Fairness die beiden Modelle zu vergleichen. Im optimalen Fall zeigt sich, dass der Algorithmus durch die Anpassung der Daten eine deutlich höhere Fairness besitzt als zuvor.

Für die Umsetzung mit der Programmiersprache Python [42] wurden dazugehörige Bibliotheken verwendet. Zu den wichtigsten verwendeten Bibliotheken gehören *scikit-learn* (`sklearn`) [43] zur Erstellung und zum Trainieren von ML-Algorithmen, *imbalanced-learn* (`imblearn`) [44] zum Resampling und *pandas* (`pd`) [45] zur Verwaltung von großen Datensätzen wie den Trainingsdaten.

In den folgenden Unterkapiteln wird zunächst auf die Architektur des Frameworks und die Initialisierung mit dem Framework eingegangen. Anschließend wird die Evaluierung bzw. Fairness-/Bias-Messung und das dazugehörige Resampling zur Entzerrung dargestellt. Am Ende kann gezeigt werden, welchen Unterschied das Resampling auf die Fairness des ML-Algorithmus hat.

Die Implementierung des Debiasing-Frameworks ist zu finden unter `https://github.com/denglerdevelopment/debiasing-in-machine-learning.git`.

David Schader

4.1 Systemarchitektur

Um zunächst einmal einen Überblick über das gesamte Framework zu erhalten, soll in diesem Abschnitt die Systemarchitektur vorgestellt werden. Das Framework besteht aus einer Hauptklasse namens `FairML`, welche die implementierten Klassen `DataSet, Encoder, Scaler, Evaluator, Resampler` und `FairnessMetric` verwendet. Das Zusammenspiel dieser Klassen ist im Klassendiagramm in Abbildung 4.1 dargestellt. Auf die genaue Funktion und Verwendung der einzelnen Klassen soll in den folgenden Abschnitten eingegangen werden.

Denis Dengler

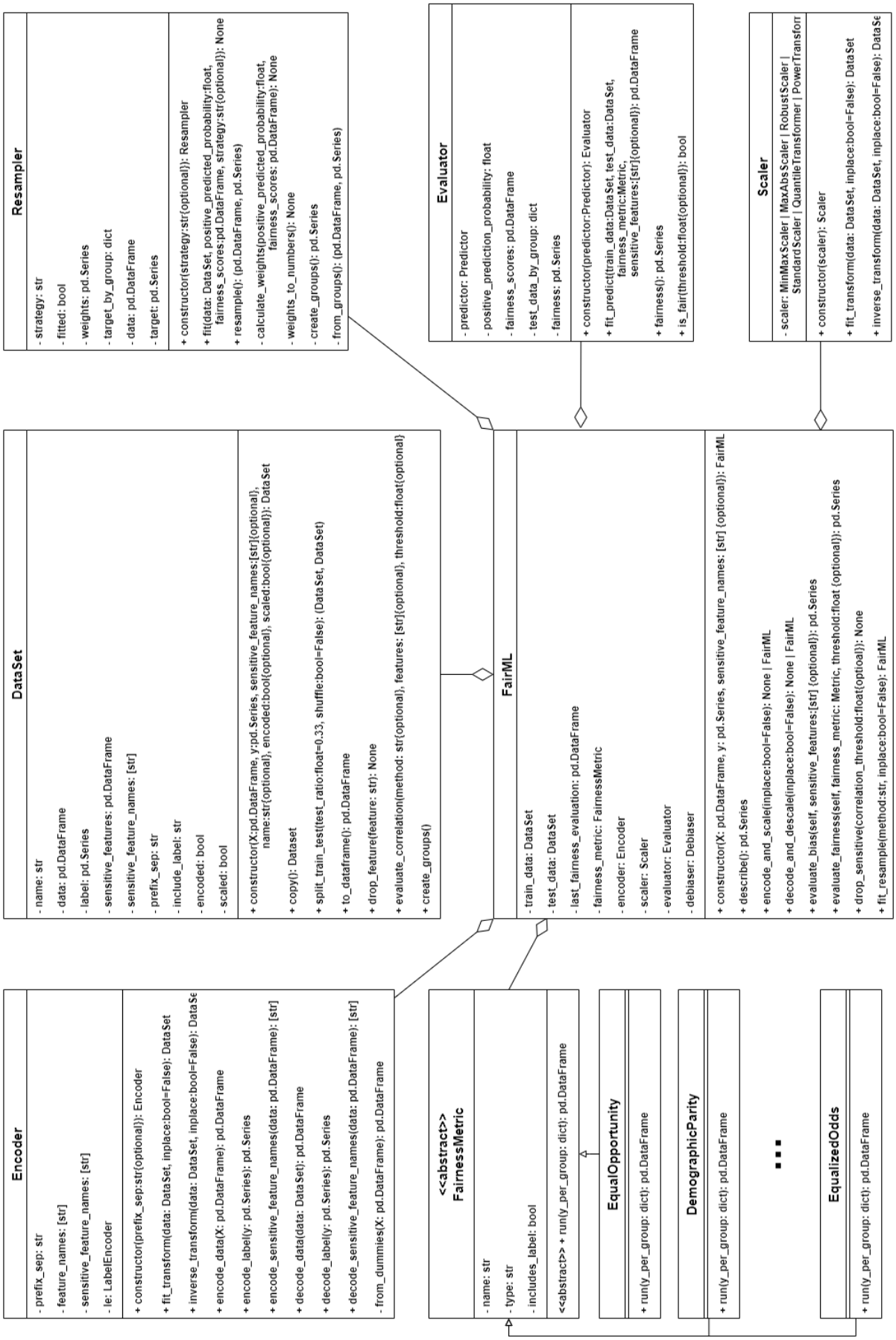


Abbildung 4.1: Klassendiagramm *FairML*

4.2 Initialisierung und Datenvorbereitung

Das Framework `fairml` selbst benötigt zur Initialisierung einen Datensatz. Dabei ist nicht vorgesehen, dass das Framework alles von der Überprüfung und vom Korrigieren bis hin zur Entzerrung selbstständig durchführt, es soll jedoch dabei unterstützen. Es ist grundlegend vorgesehen, dass bereits ein Trainings-Datensatz `pd.DataFrame` vorhanden ist. Anwender müssen also selbst entscheiden, wie Sie z.B. mit fehlenden Werten also leeren Zellen in einem Datensatz umgehen wollen. Mögliche Lösungen dafür wurden bereits im Abschnitt 3.2.1 behandelt.

Wichtig bei dieser Bereinigung fehlerhafter Daten ist, dass alle Anwender zunächst *gute* Datensätze benötigen. Mit diesen Methoden können zwar einige fehlerbehaftete Daten korrigiert werden, jedoch ist es die Aufgabe der Anwender, größtenteils vollständige Datensätze zu organisieren. Mit diesen Methoden können die restlichen unvermeidbaren Datenlücken gefüllt werden. Beim Korrigieren eines Großteils des Datensatzes entsteht jedoch ein völlig anderer Datensatz, was immer bedacht werden sollte.

Liegt der Datensatz vor, kann ein `fairml`-Objekt erstellt werden. Der Datensatz muss vorher in die Eingangsdaten (`X`) und Label (`y`) eingeteilt werden. `X` ist ein `pd.DataFrame`, welches in den Spalten die Eingangsfeatures des ML-Algorithmus beinhaltet und die Zeilen die einzelnen Objekte darstellen. In `y` soll eine `pd.Series` gespeichert sein, was sozusagen einer Spalte eines Dataframes entspricht. Darin sind die gewünschten Ausgangsdaten für die einzelnen Objekte dargestellt.
Bei der Erstellung des `fairml`-Objektes werden zusätzlich die gewünschte Fairness-Metrik und die sensitiven Attribute festgelegt. Hier können Anwender also auch die Features übergeben, welche für einen fairen Algorithmus berücksichtig werden sollen. Die Erstellung des Objektes könnte wie folgt aussehen:

```
1 from fairml import FairML
2 from fairml.metrics import EqualOpportunity
3 fairml_object = FairML(X, y, fairness_metric=EqualOpportunity(),
      sensitive_feature_names = ['sex', 'race'])
```

Quellcode 4.1: `FairML`-Initialisierung

In diesem Beispiel wurde die sensitiven Attribut *sex* und *race* gesetzt.
Das `FairML`-Objekt wird also initialisiert. Dieses beinhaltet nach der Initialisierung neben einem Objekt der Klasse *Datasets*, welches `X` und `y` zusammenfasst, auch drei weitere Objekte, auf die im Folgenden eingegangen wird.

Encoder

Der Encoder ist ein Objekt, welches zum kodieren des Datensatzes dient. Damit können also nominale Attributspalten in numerische Daten transformiert werden, da ML-Algorithmen nur mit solchen arbeiten können.

Scaler

Mit dem Scaler können die schon vom Encoder transformierten Daten noch sklaliert werden. Die Werte werden also so angepasst, dass sie zwar keine Information verlieren, jedoch einer gewissen Verteilung entsprechen. Damit können anschließend ML-Algorithmen beim Trainieren effizienter arbeiten.

Resampler

Der Resampler kann im weitern Verlauf die Trainingsdaten resamplen. Dabei kann sowohl Oversampling als auch Undersampling stattfinden, damit die Trainingsdaten ausgewogener und der damit trainierte ML-Algorithmus fairer wird. Zusätzlich können beide Verfahren kombiniert werden.

Um nicht selbst die Schritte des Encodings und Scaling zu implementieren, besitzt das `FairML`-Objekt die Funktion `encode_and_scale`, die in Quellcode 4.2 zu sehen ist.

```
1 def encode_and_scale(self):
2     if not hasattr(self, '_scaler'):
3         print("You have to set a scaler first.")
4         return
5     if not self._train_data.get_encoded():
6         self._encoder.fit_transform(self._train_data, inplace=True)
7         self._scaler.fit_transform(self._train_data, inplace=True)
8     else:
9         print("Data is already encoded.")
```

Quellcode 4.2: `FairML.encode_and_scale`-Methode

Zu sehen ist hier, dass die Funktion selbst zum einen den Encoder und zum anderen auch den Scaler aufruft, um den Datensatz vollständig zu transformieren und für das Training vorzubereiten. Die Funktion `fit_transform` ist jeweils für den Encoder in Quellcode 4.3 und für den Scaler in Quellcode 4.4 dargestellt.

```
def fit_transform(self, data: DataSet, inplace:bool=False):
    if not inplace: data = data.copy()
    self._feature_names = data.get_x().columns
    self._sensitive_feature_names = data.get_sensitive_feature_names()
    data.set_x(self.encode_data(data.get_x()))
    data.set_y(self.encode_label(data.get_y()))
    data.set_sensitive_feature_names(self.encode_sensitive_feature_names
        (data))
    data.set_encoded()
    data.set_prefix_sep(self._prefix_sep)
    return data

def encode_data(self, X:pd.DataFrame):
    return pd.get_dummies(X, drop_first=False, prefix_sep=self.
        _prefix_sep)

def encode_label(self, y:pd.Series):
    return pd.Series(self._le.fit_transform(y), name=y.name)
```

Quellcode 4.3: `Encoder.fit_transform`-Methode

```
def fit_transform(self, data: DataSet, inplace:bool=False):
    if not inplace: data = data.copy()
    data.set_x(pd.DataFrame(self._scaler.fit_transform(data.get_x()),
        columns=data.get_x().columns))
    data.set_scaled()
    return data
```

Quellcode 4.4: `Scaler.fit_transform`-Methode

Für den Encoder wird `get_dummies` von *pandas* verwendet. Dabei werden kategorielle Spalten durch binäre Vektoren ersetzt. Aus der Spalte *Geschlecht* entstehen in einem Datensatz bestehend aus Männern und Frauen die zwei Spalten *weiblich* und *männlich*. Für die Label wird standardmäßig der *LabelEncoder* von *sklearn* verwendet. Bestehen die Label also beispielsweise nur aus den zwei Klassifikationen *Katze* und *Hund*, wird der Katze die `0` und dem Hund die `1` zugewiesen.

Der Scaler skaliert den vom Encoder hervorgegangenen Datensatz. Standardmäßig wird hierbei der *MaxAbsScaler* von *sklearn* verwendet, Anwender können jedoch auch andere gewünschte Scaler wie z.B. den *StandardScaler* verwenden.

Neben der Funktion `encode_and_scale`, welche die Transformation der Daten zusammenfasst, besitzt das `FairML`-Objekt auch eine Funktion `decode_and_descale`. Mit dieser lassen sich die Transformationsschritte wieder rückgängig machen.

David Schader

4.3 Trainieren und Bewerten eines ML-Modells

Der Kern dieses Frameworks ist die Bewertung von Bias und Fairness mit anschließender Korrektur. In diesem Abschnitt soll konzeptuell beschrieben werden, wie die Fairness abhängig vom ML-Algorithmus und gewünschter Metrik gemessen und bewertet werden kann. Der prinzipielle Ablauf einer solchen Fairness-Messung besteht aus den folgenden Schritten.

Denis Dengler

4.3.1 Festlegen und Trainieren eines Klassifizierungsalgorithmus

Fairness ist abhängig von der Wahl eines Klassifizierungsalgorithmus. Jeder Algorithmus, ob SVM, Random Forest Classifier oder Neuronale Netze, sagt Klassen auf eine andere Art und Weise voraus und hat damit auch einen großen Einfluss auf die Fairness. Es wird zunächst ein Klassifizierungsalgorithmus mit `FairML.set_evaluator(predictor)` festgelegt und anschließend mit den zugrundeliegenden Trainingsdaten trainiert. Anschließend kann zum Berechnen der Fairness die Methode `FairML.evaluate_fairness()` (vgl. Quellcode 4.5) mit den optionalen bool'schen Parametern `plot, show_description` und `compare` aufgerufen werden.

Sind die Trainingsdaten zu diesem Zeitpunkt noch nicht in Test- und Trainingsdaten aufgeteilt, wird dieser Schritt vor der Fairness-Berechnung ausgeführt. Dafür wird die `train_test_split` Funktion aus `sklearn.model_selection` verwendet (vgl. Quellcode 4.6). Aus den neu entstandenen `DataFrames` werden zwei `DataSet`-Instanzen für Trainings- und Testdaten erzeugt. Hier wurde festgelegt, dass der Datensatz zu $1/3$ in die Testdaten und zu $2/3$ in die Trainingsdaten aufgeteilt wird. Während sich die Trainingsdaten im weiteren Verlauf, etwa beim Resampling (siehe Unterabschnitt 4.4.2), verändern können, muss sichergestellt werden, dass die Testdaten für jede Wiederholung der Vorhersage gleich bleiben. Dies wird dadurch erreicht, dass die Daten vor dem Training separat abgespeichert werden. Zu diesem Zeitpunkt sind die Daten bereits kodiert und skaliert, sodass sichergestellt wird, dass die Trainings- und Testdaten die selben Dimensionen haben.

```
def evaluate_fairness(self, threshold:float=0.8, plot:bool=True, compare
    :bool=False, show_description:bool=False):
        if not hasattr(self, '_evaluator'):
            print("You have to set an evaluator first.")
            return
        if not hasattr(self, '_test_data'):
            self._train_data, self._test_data = self._train_data.
                split_train_test(shuffle=True)
        self._evaluator.fit_predict(self._train_data, self._test_data,
            self._fairness_metric)
        if plot:
            if compare:
                if hasattr(self, '_last_fairness_evaluation'):
                    compare_fairness = pd.concat([self.
                        _last_fairness_evaluation, self._evaluator.
                        _fairness_scores], axis = 1)
                    compare_fairness.columns = ['old', 'new']
                    visualize_fairness(compare_fairness.copy(), title=
                        self._fairness_metric.get_name(),
                        show_description=False)
                else:
                    print("Nothing to compare yet.")
                    visualize_fairness(self._evaluator._fairness_scores.
                        copy(), title=self._fairness_metric.get_name(),
                        show_description=show_description)
            else:
                visualize_fairness(self._evaluator._fairness_scores.copy
                    (), title=self._fairness_metric.get_name(),
                    show_description=show_description)
        self._last_fairness_evaluation = self._evaluator.
            _fairness_scores
        return self._evaluator.evaluate_fairness()
```

Quellcode 4.5: `FairML.evaluate_fairness`-Methode

```
from sklearn.model_selection import train_test_split
def split_train_test(self, test_ratio:float=0.33, shuffle:bool=True):
    X_train, X_test, y_train, y_test = train_test_split(self.get_x(),
        self.get_y(), test_size=test_ratio, shuffle=shuffle)
    train_data = DataSet(X_train, y_train, sensitive_feature_names=self.
        get_sensitive_feature_names(), encoded=self.get_encoded(),
        scaled=self.get_scaled(), prefix_sep=self.get_prefix_sep(),
        include_label=self._include_label)
    test_data = DataSet(X_test, y_test, sensitive_feature_names=self.
        get_sensitive_feature_names(), encoded=self.get_encoded(),
        scaled=self.get_scaled(),prefix_sep=self.get_prefix_sep(),
        include_label=self._include_label)
    return train_data, test_data
```

Quellcode 4.6: `DataSet.split_train_test`-Methode

```
def fit_predict(self, train_data:DataSet, test_data: DataSet,
    fairness_metric:FairnessMetric, sensitive_features:[str]=[]):
        self._predictor.fit(train_data.get_x(), train_data.get_y())
        sensitive_features = train_data.get_sensitive_feature_names() if
             len(sensitive_features) == 0 else sensitive_features
        self._test_data_by_group = dict()
        df = test_data.to_dataframe()

        df = test_data.to_dataframe(include_groups=True)
        reduced = df.groupby(test_data.get_groups().name)
        groups = test_data.get_groups().unique()
        for group in groups:
            X = reduced.get_group(group)
            g = X.pop(X.columns[-1])
            y = X.pop(X.columns[-1])
            y_dict = dict()
            y_dict['y_true'] = y
            y_dict['y_pred'] = self._predictor.predict(X)
            self._test_data_by_group[group] = y_dict

        self._positive_predicted_probability = metrics.
            positive_predicted_probability(self._test_data_by_group)
        self._fairness_scores = fairness_metric.run(self.
            _test_data_by_group)
        self._fairness_scores.name = fairness_metric.get_name()
        return self._fairness_scores
```

Quellcode 4.7: `Evaluator.fit_predict`-Methode

4.3.2 Bewertung der algorithmischen Fairness

Um die Fairness messbar zu machen, können die in Abschnitt 2.3 definierten Fairness-Metriken angewandt werden. Dazu werden die Testdaten bereits beim Initialisieren um eine `pd.Series` erweitert, in dem die Gruppenzugehörigkeit jedes Datenobjekts als String abgespeichert wird. Dabei wird die Klassenzugehörigkeit ebenfalls als sensitives Attribut gewertet und fließt damit in die Gruppenbezeichnungen mit ein. Die Trainingsdaten werden nun nach Gruppenzugehörigkeit gruppiert und auf jede dieser gruppierten Testdaten der trainierte Klassifizierungsalgorithmus angewandt, um die Klassenzugehörigkeit vorherzusagen. Diese und die tatsächlichen Label werden pro Gruppe in einem *Python Dictionary* gespeichert und anschließend an die gewählte Fairness-Metrik übergeben, welche pro Gruppe die Fairness in einem `pd.DataFrame` zurückgibt.

Die Ergebnisse Z der Fairness-Berechnung werden nun verglichen. Da das Modell dann als fair gilt, wenn die zugrundeliegende Metrik für alle Gruppen das gleiche Ergebnis liefert, ist es notwendig, eine Funktion zu definieren, welche genau diese Bedingung überprüft. Da es in der Realität nicht vorkommen wird, dass das Ergebnis Z_i zweier Gruppen identisch ist, wird eine sogenannte p%-Regel angewandt. Dabei kann eine Akzeptanzgrenze p festgelegt werden, die prüft, ob zwei Ergebnisse zu p% gleich sind. [46] Damit ein solcher Vergleich über mehrere Gruppen hinweg und unabhängig von der Betrachtungsrichtung möglich ist, werden die Extremwerte $min(Z)$ und $max(Z)$ betrachtet und p wie folgt definiert:

$$p = \frac{min(Z)}{max(Z)} \tag{4.1}$$

In der Implementierung wird deshalb zunächst die Methode `fairness(self)` definiert, welche p gemäß Gleichung 4.1 berechnet und eine Methode `is_fair(self, threshold:float=0.8)`, welche überprüft, ob der Wert p die Anforderungen erfüllt, also größer gleich dem hier festgelegte Grenzwert (`threshold`) von 80% ist.

```
1 def fairness(self):
2     self._fairness = pd.Series(name=self._fairness_scores.name)
3     for metric, values in self._fairness_scores.items():
4         self._fairness[metric] = np.min(values)/np.max(values)
5     return self._fairness
6
7 def is_fair(self, threshold:float=0.8) -> bool:
8     return self._fairness.apply(lambda x: x>=threshold).all()
```

Quellcode 4.8: `Evaluator.fairness`- und `.is_fair`-Methode

4.3.3 Fairness-Metriken

Wie in Unterabschnitt 2.3.1 und den darauf folgenden Unterabschnitten bereits dargestellt, gibt es viele verschiedene Metriken zum Messen der Fairness. Hier wurden Metriken der Independenz und der Separation implementiert. Die Funktionen selbst bekommen jeweils ein *Python Dictionary* mit der Bezeichnung `y_per_group`, welches pro Gruppe ein Array namens `y_true`, in dem die Label gespeichert sind, und ein Array namens `y_pred`, in dem die Vorhersagen durch das ML-Modell gespeichert sind, beinhaltet.
Als Rückgabe geben die Funktionen ein `pd.DataFrame`, in dem pro Gruppe die berechnete Fairness-Metrik als Wert beinhaltet ist. Dieses wird nach der Evaluierung zusätzlich in `Evaluator._test_data_by_group` gespeichert. Bei der Metrik *Equal Opportunity* kann das *DataFrame* z.B. wie in Tabelle 4.1 dargestellt aussehen.

	TPR
group	
male	0,592
female	0,503

Tabelle 4.1: *Equal Opportunity* Output

Da bei der Metrik *Equal Opportunity* die TPR zwischen verschiedenen Gruppen verglichen wird, wird hier pro Gruppe auch nur die TPR berechnet. Zur Visualisierung wurde auch die Funktion `visualize_fairness` im Modul `fairml.metrics` implementiert. Mit dieser kann das *DataFrame* wie in Abbildung 4.2 dargestellt abgebildet wird.

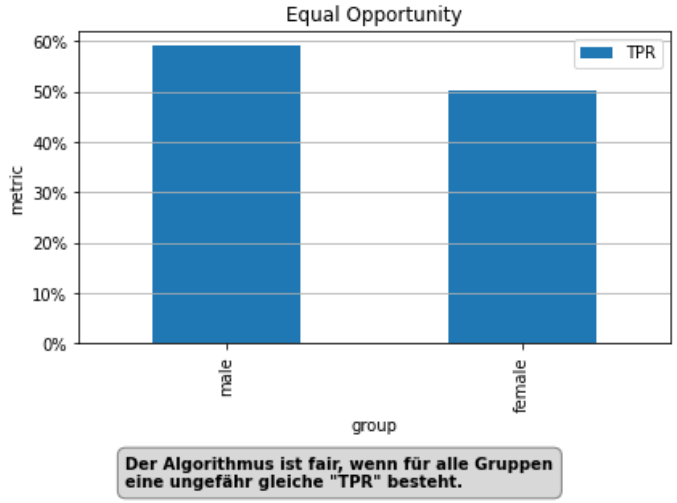


Abbildung 4.2: *Equal Opportunity* Visualisierung

Nach einem durchgeführten Resampling, kann sich auch graphisch darstellen lassen, wie sich die Fairness-Metrik nach dem Resampling im Vergleich zum Datensatz vorher verändert hat.

Dazu muss in der `evaluate_fairness`-Methode (Quellcode 4.5) `compare = True` gesetzt werden. Für die *Equal Opportunity* könnte solch ein Vergleich z.B. wie in Abbildung 4.3 dargestellt aussehen.

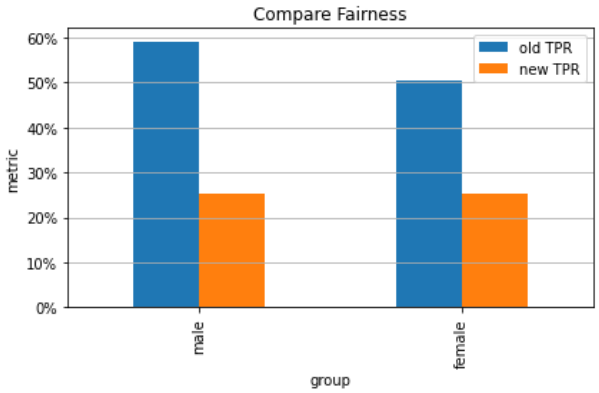


Abbildung 4.3: Fairness-Vergleich mit *Equal Opportunity*

In den folgenden Quellcode-Abschnitten wird gezeigt, wie die Fairness-Metriken in `fairml.metrics` implementiert wurden. Für die Implementierung wurden die Independenz-Metrik *Demographic Parity* und einige Separations-Fairness-Metriken verwendet, dabei jedoch auch nur diejenigen, bei denen nur eine ML-Metrik, wie z.B. nur die TPR bei der *Equal Opportunity*, verwendet wird. Die implementierten Resampling-Verfahren sind nur dazu geeignet, mit einer ML-Metrik zu arbeiten, da die Gewichtsberechnung pro Gruppe bei der Berücksichtigung mehrerer Faktoren wie z.B. bei der Fairness-Metrik *Equalized Odds* nicht funktioniert. Daher sind bei der Verwendung nur die Fairness-Metriken *Demographic Parity*, *Equal Opportunity*, *Equal Accuracy Equality*, *Treatment Equality* und *Equalizing Disincentives* verfügbar.
Grundlegend gibt es eine Klasse `FairnessMetric`, auf der die einzelnen Fairness-Metriken aufbauen, diese ist in Quellcode 4.9 dargestellt. In Quellcode 4.10 ist die Implementierung der *Demographic Parity* und in Quellcode 4.11 die Implementierung der *Equal Opportunity* dargestellt, die anderen Separations-Metriken wurden analog zu dieser implementiert.

```
1 class FairnessMetric(ABC):
2     def get_name(self): return self._name
3     def get_type(self): return self._type
4     def get_includes_label(self): return self._includes_label
5 
6     @abstractmethod
7     def run(self, y_per_group:dict): pass
```

Quellcode 4.9: Abstrakte Klasse `fairml.metrics.FairnessMetric`

```
class DemographicParity(FairnessMetric):
    def __init__(self):
        self._name = "Demographic Parity"
        self._type = "Independence"
        self._includes_label = True

    def run(self, y_per_group):
        result = pd.DataFrame(data = {'group': y_per_group.keys()},
            columns = ['group', 'DP']).set_index('group')
        for group, results in y_per_group.items():
            if any(results['y_true']):
                result.loc[group]['DP'] = np.count_nonzero(results['
                    y_pred'])/len(results['y_pred'])
        result.name = "demographic_parity"
        return result.dropna(how='all')
```

Quellcode 4.10: `fairml.metrics.DemographicParity`-Klasse

```
class EqualOpportunity(FairnessMetric):
    def __init__(self):
        self._name = "Equal Opportunity"
        self._type = "Separation"
        self._includes_label = True

    def run(self, y_per_group):
        result = pd.DataFrame(data = {'group': y_per_group.keys()},
            columns = ['group', 'TPR']).set_index('group')
        for group, results in y_per_group.items():
            if any(results['y_true']):
                result.loc[group]['TPR'] = MLMetrics(results['y_pred'],
                    results['y_true']).tpr
        result.name = 'equal_opportunity'
        return result.dropna(how='all')
```

Quellcode 4.11: `fairml.metrics.EqualOpportunity`-Klasse

Zu sehen ist hier, dass sich die Implementierung der Separations-Metriken zusätzlich der implementierten Klasse `MLMetrics` bedienen. Diese ist in Quellcode 4.12 dargestellt. Die Klasse `MLMetrics` unterstützt bei den Separations-Fairness-Metriken, da diese auf ML-Metriken wie z.B. der TPR oder FNR aufbauen. Zur Initialisierung bekommt die Klasse jeweils eine Liste mit vorhergesagten und tatsächlichen Labeln und erstellt daraus eine Konfusionsmatrix. Daraus können anschließend die Anzahl an TP-, TN-, FP- und FN-Klassifikationen extrahiert werden, mit denen anschließend die Standard-Metriken eines ML-Modells berechnet werden. Diese können anschließend in den Separations-Fairness-Metriken einfach pro Gruppe aufgerufen werden.

```
class MLMetrics():
    def __init__(self, y_pred, y_true):
        self.cm = confusion_matrix(y_true, y_pred)
        if self.cm.shape != (2,2):
            cm = np.array([[0, 0], [0, 0]])
            for kombi in zip(y_true, y_pred): cm[kombi]+=1
            self.cm = cm
        tn, fp, fn, tp = self.cm.ravel()
        #accuracy
        if self.cm.sum() == 0: self.accuracy = 0
        else:                  self.accuracy = (tp+tn)/self.cm.sum()
        #precision
        if tp+fp == 0: self.precision = 0
        else:          self.precision = tp/(tp+fp)
        #tpr & fnr
        if tp+fn == 0: self.tpr = self.fnr = 0
        else:
            self.tpr = tp/(tp+fn)
            self.fnr = fn/(tp+fn)
        #tnr & fpr
        if tn+fp == 0: self.tnr = self.fpr = 0
        else:
            self.tnr = tn/(tn+fp)
            self.fpr = fp/(tn+fp)
        #F1
        if self.tpr + self.precision == 0: self.F1 = 0
        else:
            self.F1 = 2*self.tpr*self.precision/(self.tpr + self.
                precision)
```

Quellcode 4.12: `fairml.metrics.MLMetrics`-Klasse

David Schader

4.4 Debiasing

Nach erfolgreicher Bewertung von Bias und Fairness ist der nächste Schritt die Korrektur einer vorhandenen Verzerrung. Dazu sollen zwei Ansätze konzeptuell erarbeitet und implementiert werden: Das Löschen von sensitiven Attributen (*Sensitive Feature Dropping*) und Reweighting/Resampling. Das in Unterabschnitt 3.4.2 vorgestellte Relabeling wird aufgrund des intrusiven Ansatzes nicht weiter verfolgt und der Fokus damit mehr auf die anderen beiden Verfahren gelegt. Das Sensitive Feature Dropping und das Reweighting/Resampling sollen sich nicht gegenseitig ausschließen, sondern es soll ein Verfahren erarbeitet werden, welches diese beiden Methoden sinnvoll kombiniert.

Denis Dengler

4.4.1 Sensitive Feature Dropping

Wie in Unterabschnitt 3.4.1 theoretisch erläutert, ist es notwendig zu überprüfen, inwiefern ein sensitives Attribut mit den anderen Attributen korreliert. Dafür wird mithilfe der `pd.DataFrame.corr`-Methode zunächst die Korrelation zwischen allen Attributen berechnet und in einem `pd.DataFrame` gespeichert. Um nun die Korrelation der sensitiven Attribute zu betrachten, werden die entsprechenden Spalten des DataFrames extrahiert. Da diese auch die Korrelation mit sich selbst beinhalten, werden die entsprechenden Zeilen gelöscht. Anschließend wird für jeden Wert in der Korrelationsmatrix überprüft, ob dessen Betrag einen Schwellenwert, welcher standardmäßig auf 0,2 gesetzt ist, überschreitet. Ist dies nicht der Fall, wird der Wert verworfen, sodass als Ergebnis eine Korrelationsmatrix mit absoluten Korrelationen größer dem Schwellwert entsteht. Der Schwellenwert kann prinzipiell frei zwischen 0 und 0,5 gewählt werden, da dieser Wertebereich eine eindeutige Korrelation ausschließt, jedoch empfiehlt sich die Wahl eines möglichst niedrigen Werts, um so eine Korrelation mit hoher Sicherheit ausschließen zu können.

```
1  def evaluate_correlation(self, features:[str]=[], threshold:float=0.2):
2      if not self.get_encoded():
3          raise Exception('Data has to be encoded before correlation
               evaluation.')
4      features = features if len(features) > 0 else self.
           get_sensitive_feature_names()
5      self._correlation = self.get_x().corr()
6      if self.get_y().name in features:
7          features.remove(self.get_y().name)
8      features = [col for col in self.get_x().columns if col.startswith(
           tuple(features))]
9      result = self._correlation.filter(regex='^('+'|'.join(features)+')')
           .copy()
10     result.drop(index=features, inplace=True)
11     result = result[abs(result)>threshold]
12     result_by_feature = result.groupby(axis=1, by=lambda x: str(x).split
           (self._prefix_sep)[0])
13     return result_by_feature
```

Quellcode 4.13: `DataSet.evaluate_correlation`-Methode

Diese Vorgehensweise wird für jedes sensitives Attribut durchgeführt und anschließend überprüft, ob die Ergebnismatrix gültige Werte beinhaltet. Ist dies nicht der Fall, korreliert dieses Attribut zu weniger als dem Schwellenwert mit allen anderen Attributen und kann somit gelöscht werden. Dieser Schritt wird immer vor dem Resampling durchgeführt. Sollten alle sensitiven Attribute entfernt worden sein, ist ein Resampling nicht mehr notwendig.

```
def drop_feature(self, feature:str):
    columns = self.get_x().filter(regex='^'+ feature).columns
    self.get_x().drop(columns=columns, inplace=True)
    sensitive_features = self.get_sensitive_feature_names()
    for c in columns:
        sensitive_features.remove(c)
    self.set_sensitive_feature_names(sensitive_features)
```

Quellcode 4.14: `DataSet.drop_feature`-Methode

Denis Dengler

4.4.2 Debiasing durch Resampling

Sollte mindestens ein sensitives Attribut mit anderen Attributen korrelieren, kann dieses nicht einfach entfernt werden. In diesem Fall müssen die Trainingsdaten entsprechend der berechneten Fairness-Scores resampled werden. Dafür wird die Python-Bibliothek *imbalanced-learn* (`imblearn`) [44] verwendet, welche bereits eine Vielzahl an Resampling-Verfahren zur Verfügung stellt. Dennoch müssen einige Schritte durchgeführt werden, bevor ein Resampling-Algorithmus angewandt werden kann.

Berechnung der Gruppengewichte

Um einen fairen Klassifizierungsalgorithmus zu erhalten, wird pro Gruppe ein Gewicht gemäß Gleichung 3.7 berechnet, welches dann unprivilegierte Gruppen im Vergleich zu den privilegierten Gruppen stärker gewichtet. Zur Gruppierung der Daten werden die im Datensatz definierten Gruppenzugehörigkeiten verwendet. Die Art der Gewichtsberechnung entscheidet bereits, welche Resampling-Strategie angewandt wird. Sollen die Daten oversampled werden, so wird das Gewicht jeder Gruppe durch das minimale Gruppengewicht geteilt, sodass die privilegierteste Gruppe mit 1 und alle anderen Gruppen mit einem Wert größer 1 gewichtet werden. Im Falle des Undersamplings wird analog das Gewicht jeder Gruppe mit dem maximalen Gruppengewicht geteilt, sodass die am wenigsten privilegierte Gruppe mit 1 und alle anderen mit einem Wert kleiner 1 gewichtet werden. Soll ein kombiniertes Resampling stattfinden, wird das Gewicht jeder Gruppe mit dem Median der Gruppengewichte geteilt, sodass gleichzeitig unprivilegierte Gruppen oversampled und privilegierte undersampled werden können.

```
def calculate_weights(self, positive_predicted:float, fairness_scores:pd
    .Series):
    self._weights = pd.Series()
    for group in fairness_scores.index:
        rate = fairness_scores.loc[str(group)].values
        if len(rate)>1:
            tpr, tnr = rate
            if not np.isnan(tpr) and not float(tpr) == 0:
                self._weights.loc[str(group)] = float(positive_predicted
                    / tpr)
            elif not np.isnan(tnr) and not float(tnr) == 0:
                self._weights.loc[str(group)] = float((1-
                    positive_predicted) / tnr)
        elif not float(rate) == 0:
            self._weights.loc[str(group)] = float(positive_predicted /
                rate)
    if self._strategy == 'combined': comp = np.median(self._weights)
    elif self._strategy == 'over':   comp = np.min(self._weights)
    elif self._strategy == 'under':  comp = np.max(self._weights)
    self._weights = self._weights.apply(lambda x: x/comp)
```

Quellcode 4.15: `Resampler.calculate_weights`-Methode

Berechnung der gruppenbezogenen Zieldatenmengen

Da die Resampling-Algorithmen von `imblearn` nicht mit individuellen Gewichten arbeiten können, müssen diese zunächst in ganzzahlige Zieldatenmengen umgerechnet werden. Dafür wird je nach gewählter Resampling-Strategie die aktuell vorhandene Menge an Datenobjekten einer Gruppe mit dem entsprechenden Gruppengewicht multipliziert und dieser Wert in einem *Python Dictionary* gespeichert. Ist das Gewicht größer 1 und damit die Zieldatenmenge größer als die aktuelle Menge, wird der Wert im Dictionary `target_over`, anderenfalls im Dictionary `target_under` gespeichert. Bei der Verwendung eines Resampling-Algorithmus wird das jeweils entsprechende Dictionary an diesen übergeben.

```
def weights_to_number(self):
    self._group_target_amount = dict()
    value_counts = self._target.value_counts()
    for group in self._weights.index:
        self._group_target_amount[str(group)] = int(np.ceil(value_counts
            [str(group)] * self._weights[str(group)]))
```

Quellcode 4.16: `Resampler.weights_to_numbers`-Methode

Resampling

Wie bereits erwähnt, bietet die Python Bibliothek `imblearn` eine Vielzahl an Resampling-Algorithmen. Diese wurden bereits in Abschnitt 3.5 vorgestellt und näher erläutert. Bei der Wahl des Oversampling-Algorithmus ist die Entscheidung auf eine Kombination des RandomOverSamplers und dem Borderline SMOTE-Algorithmus gefallen. Dies ist damit zu begründen, dass SMOTE allgemein sehr gute Ergebnise liefert, da nicht nur Datenobjekte vervielfacht werden, sondern eine bewusste Interpolation der Daten stattfindet. Die Erweiterung um eine Borderline verbessert diesen Ansatz weiter, indem Ausreißer isoliert werden, um das Ergebnis nicht zu verfälschen.

```
1  def fit(self, data: DataSet, positive_predicted:float,
       fairness_evaluation:pd.Series):
2      if not hasattr(self, '_strategy'):
3          raise Exception('No resampling strategy set yet. Please set
               using `set_strategy(strategy)` method.')
4      self._data = data.to_dataframe()
5      self._sensitive_features = data.get_sensitive_features()
6      self._target = data.get_groups().copy()
7      self.calculate_weights(positive_predicted, fairness_evaluation)
8      self.weights_to_number()
9      self._target_init, self._target_under, self._target_over = dict(),
           dict(), dict()
10     value_counts = self._target.value_counts()
11     for group in self._group_target_amount.keys():
12         if self._group_target_amount[str(group)] < value_counts[str(
               group)]:
13             self._target_under[str(group)] = self._group_target_amount[
                   str(group)]
14         elif self._group_target_amount[str(group)] > value_counts[str(
               group)]:
15             self._target_over[str(group)] = self._group_target_amount[
                   str(group)]
16         elif value_counts[str(group)] < self.MIMIMUM_SAMPLES:
17             self._target_init[str(group)] = self.MIMIMUM_SAMPLES
18     self._categorical_features = [self._data.columns.get_loc(col) for
           col in list(self._data.select_dtypes(include=['category', object
           ]).columns)]
19     self._fitted = True
```

Quellcode 4.17: `Resampler.fit`-Methode

Ein Problem bei der Verwendung von SMOTE-basierten Algorithmen ist, dass Minoritäten mit einer sehr starken Unterrepräsentation (< 6) dazu führen, dass der SMOTE-

Algorithmus, der mit den k Nearest Neighbors arbeitet, um synthetische Daten zu erzeugen, nicht ausgeführt werden kann. Deshalb wird als Vorbereitung auf den Borderline SMOTE ein RandomOverSampler verwendet, um die Anzahl aller Gruppen auf mindestens sechs zu heben. Diese Zahl ist durch die Anzahl der Nearest Neighbors `k_neighbors = 5` bedingt, welche standardmäßig benötigt werden. Aufgrund der vorherigen Gewichts- und Zielmengenberechnung ist ein kontrolliertes Undersampling notwendig. Diese Anforderung wird nur von NearMiss, RandomUnderSampler und ClusterCentroids erfüllt. In diesem Fall wird der NearMiss-Algorithmus verwendet. Dieser löscht Elemente der Majorität durch Berechnung des geringsten Abstands zu anderen Gruppen, sodass dadurch eine bessere Trennung der Gruppen erreicht werden kann. Prinzipiell stehen nun drei Möglichkeiten des Resamplings zur Verfügung: *over*, *under* oder *combined*. Der Borderline-SMOTE-Algorithmus erhält in jedem Fall das vorher definierte *Python Dictionary* `target_over` und der NearMiss-Algorithmus analog das `target_under` Dictionary. Um die Daten auf das Oversampling vorzubereiten, wird zusätzlich ein `target_init` Dictionary erzeugt, welches dem RandomOverSampler übergeben wird und somit sicherstellt, dass die mindestens benötigte Menge einzelner Gruppendatenobjekte vorhanden ist.

```
1  def resample(self):
2      if not self._strategy in ['combined', 'over', 'under']:
3          raise Exception("invalid strategy " + self._strategy)
4      if self._strategy in ['combined', 'over']:
5          if len(self._target_init) > 0:
6              init = RandomOverSampler(sampling_strategy=self._target_init
                   )
7              print('Initial RandomOversampling started')
8              self._data, self._target = init.fit_resample(self._data,
                   self._target)
9              print('Initial RandomOversampling finished')
10         over = BorderlineSMOTE(sampling_strategy=self._target_over)
11         print("Oversampling started")
12         self._data, self._target = over.fit_resample(self._data, self.
               _target)
13         print("Oversampling finished")
14     if self._strategy in ['combined', 'under']:
15         print("Undersampling started")
16         resampler = NearMiss(sampling_strategy=self._target_under)
17         self._data, self._target = resampler.fit_resample(self._data,
               self._target)
18         print("Undersampling finished")
19     return self._data.iloc[:, :-1], self._data.iloc[:, -1]
```

Quellcode 4.18: `Resampler.resample`-Methode

5 Evaluation

In diesem Kapitel soll anhand von beispielhaften Datensätzen gezeigt werden, inwiefern das implementiere Framework unfaire Algorithmen erkennen kann und wie mithilfe von Resampling diese Algorithmen fairer gemacht werden können. Es sollen im Folgenden mehrere Datensätze betrachtet werden. Anhand dieser können die Stärken und Schwächen des implementierten Frameworks gezeigt werden.

David Schader

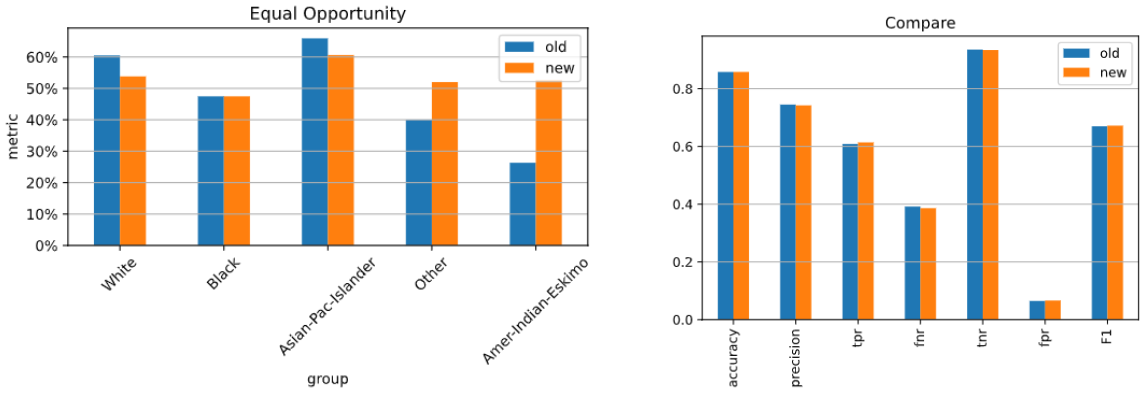
5.1 Gehaltseinschätzung

Im ersten Beispiel wird auf den Datensatz *adults*, zu finden unter `https://archive.ics.uci.edu/ml/datasets/Adult`, eingegangen. Die X-Daten des Datensatzes stellen jeweils 14 Eigenschaften von 48.842 Personen dar, dazu gehören z.B. Alter, Bildungsgrad, Beziehungsstatus, Heimatland etc. Zusätzlich gibt es die zwei Eigenschaften *sex*, also Geschlecht, dazu gehören weiblich und männlich, und *race*, dazu gehören z.B. hellhäutig, dunkelhäutig, asiatisch etc. Diese beiden Eigenschaften lassen schon vermuten, dass sie Bias-anfällig sind. Als Y-Daten beinhaltet der Datensatz das jährliche Einkommen. Dabei wird unterschieden, ob eine Person jährlich über oder unter 50.000 $ verdient.
Mit diesen Daten kann man also einen ML-Algorithmus trainieren, der für bestimmte Personen schätzt, ob diese *gut* oder *schlechter* verdienen.

David Schader

5.1.1 Debiasing nach *race* mit *Equal Opportunity*

Im ersten Beispiel wurde mithilfe des Frameworks erkannt, dass eine trainierte SVM, ein ML-Algorithmus, unfair ist bezüglich des sensitiven Attributes *race*. Dabei wurde die *Equal Opportunity*-Methode verwendet, da diese nur auf die TPR schaut und man möchte, dass Menschen unabhängig von der *race* gut verdienen können, also positiv (über 50.000 $ pro Jahr) klassifiziert werden. Zum Resampling wurde die *combined*-Methode verwendet. Die Veränderung der TPR, der Metrik, die die *Equal Opportunity* verwendet, vor und nach dem Resampling ist in Abbildung 5.1 a) dargestellt.



(a) Fairness-Metrik (b) ML-Metrik

Abbildung 5.1: *adults*-Debiasing nach *race* mit *Equal Opportunity*

Vor dem Resampling zeigt sich ganz klar, dass das Modell eine deutliche Verzerrung hatte. Z.B. hatten hellhäutige mit ca. 60% eine deutlich höhere TPR als dunkelhäutige Menschen mit einer TPR von nur ca. 50%.
Nach dem Resampling bzw. Debiasing zeigt sich, dass nun alle Objekte des Attributes *race* eine ungefähr gleiche TPR besitzen. Diese liegen nun deutlich näher beieinander als zuvor.

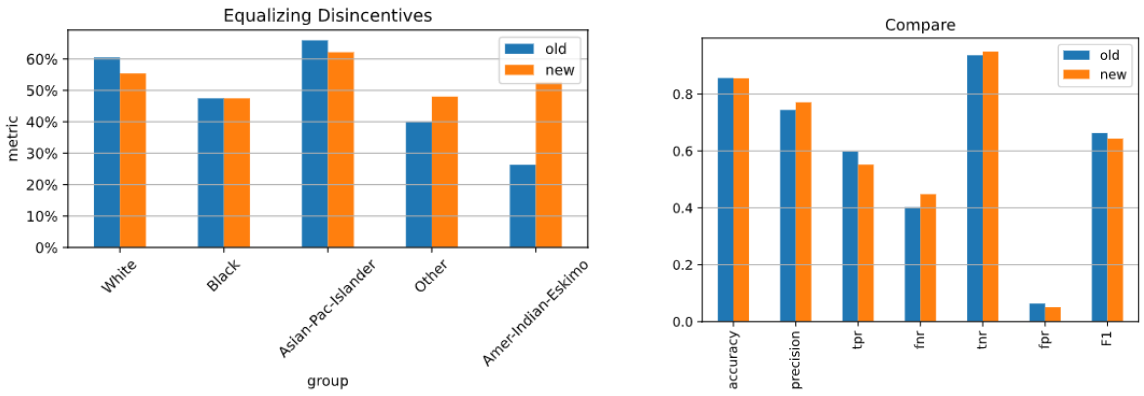
Mit der Betrachtung des p-Wertes (Gleichung 4.1) lässt sich zeigen, wie stark die Fairness gestiegen ist. Vor dem Resampling wurde $p_{old} = 0{,}40$ ermittelt. Nach dem Resampling wurde in diesem Beispiel $p_{new} = 0{,}77$ ermittelt. Dieser p_{new}-Wert liegt deutlich näher an der allgemein festgelegten $p \geq 80\%$-Regel, welche einen fairen Algorithmus kennzeichnet. Mit dem Resampling wurde also ein diskriminierender Algorithmus bezüglich der *race* deutlich entzerrt und das Debiasing wurde erfolgreich durchgeführt.

Oft führt eine Entzerrung bzw. ein Debiasing dazu, dass die Algorithmen zwar fairer werden, jedoch an ihrer Genauigkeit und anderen ML-Genauigkeitsmaßen einbüßen. Bei diesem Beispiel zeigt sich jedoch, dass das Debiasing einen äußerst geringen Effekt auf die Genauigkeit hat. Dies verdeutlicht Abbildung 5.1 b), dort werden die ML-Metriken des Modells vor und nach dem Resampling verglichen.

David Schader

5.1.2 Debiasing nach *race* mit *Equalizing Disincentives*

Um zu zeigen, welchen Einfluss die Wahl der Fairness-Metrik haben kann, wird im nächsten Beispiel in Abbildung 5.2 das selbe Verfahren wie zuvor visualisiert, jedoch mit der *Equalizing Disincentives* als Fairness-Metrik.



(a) Fairness-Metrik (b) ML-Metrik

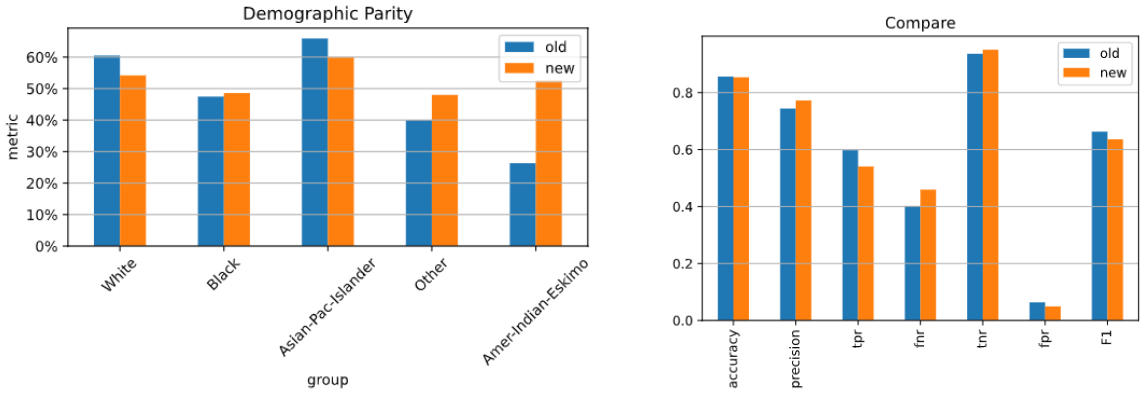
Abbildung 5.2: *adults*-Debiasing nach *race* mit *Equalizing Disincentives*

Beim Verwenden der Fairness-Metrik *Equalizing Disincentives* steigt der p-Wert auch ähnlich zur *Equal Opportunity*. Der neue p-Wert liegt nun bei $p_{new} = 0{,}76$, er ist damit nur geringfügig schwächer als bei der *Equal Opportunity*. Es zeigt sich also, dass die Metrik nicht zwingend Einfluss auf das Verbesserungs-Ergebnis haben muss, jedoch sollte darauf geachtet werden, eine geeignete und logisch passende Metrik für das Beispiel zu wählen.

David Schader

5.1.3 Debiasing nach *race* mit *Demographic Parity*

Im Vergleich zu den beiden Separations-Metriken soll nun einmal die Demographische Parität als Beispiel für eine Independenz-Metrik evaluiert und mit den vorangehenden Metriken verglichen werden. Die Ergebnisse unter Verwendung dieser Metrik sind in Abbildung 5.3 zu sehen.



(a) Fairness-Metrik (b) ML-Metrik

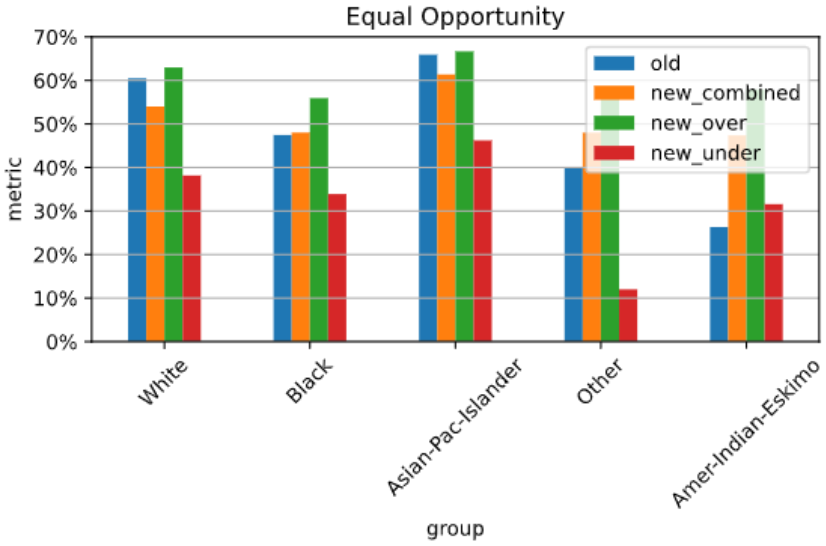
Abbildung 5.3: *adults*-Debiasing nach *race* mit *Demographic Parity*

Bei Betrachtung der genauen Werte können ist erkennbar, dass die Fairness, welche vor dem Resampling bei $p_{old} = 0{,}40$ lag, nun einen Wert von $p_{new} = 0{,}80$ erreicht hat und damit als fair einzustufen ist.

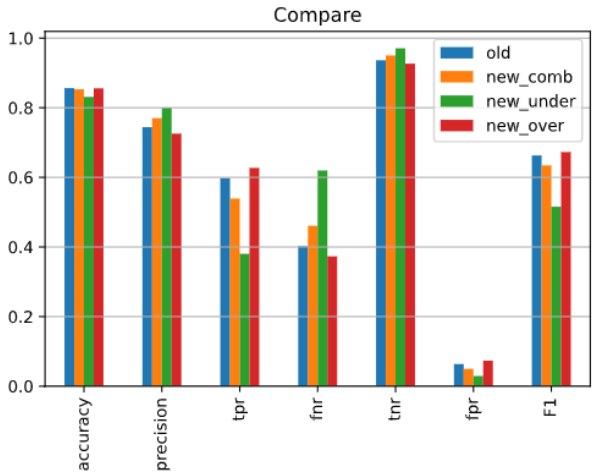
Denis Dengler

5.1.4 Unterschiedliche Resampling-Methoden

In diesem Beispiel soll gezeigt werden, welchen Einfluss die unterschiedlichen Resampling-Methoden *combined*, *over* und *under* haben können. Dabei soll wie in Unterabschnitt 5.1.1 schon mit einer SVM gearbeitet werden. Zusätzlich wird nur ein Debiasing nach dem sensitiven Attribut *race* durchgeführt. Die unterschiedlichen Ergebnisse sind in Abbildung 5.4 zusammengetragen.



(a) Vergleich-Fairness



(b) Vergelich-ML-Metriken

Abbildung 5.4: Vergleich Resampling (*combined*, *over* und *under*)

Neben der tatsächlichen Fairness-Verbesserung sollte bei der Wahl der unterschiedlichen Resampling-Methoden auch die Rechenzeit mitberücksichtig werden. Während man bei der *combined*-Methode wieder einen ungefähr gleich großen Datensatz zum Trainieren erhält, sinkt die Anzahl der Objekte bei der *under*-Methode und steigt bei der *over*-Methode.

In Tabelle 5.1 ist zusammengetragen, wie sich der p-Wert je nach Resampling-Verfahren verändert. Zusätzlich wird zur Information auch die Größe an Objekten mit angegeben, die vor bzw. nach dem Resampling im Trainingsdatensatz vorhanden war.

resampling	p	Datensatz-Größe
ohne Resampling (old)	0,40	32.724
combined	0,77	31.141
over	0,84	33.590
under	0,26	28.338

Tabelle 5.1: Effizienz bei unterschiedlichen Resampling-Verfahren *adults*

Hier zeigt sich deutlich, dass die *combined*-Strategie in diesem Beispiel bereits ein gutes Ergebnis hervorbringt, der p-Wert ist schon nah an der 80%-Grenze. Das beste Ergebnis ist in diesem Beispiel mit dem reinen Oversampling möglich, hierbei steigt der p-Wert über die 80%-Grenze. Ein reines Undersampling ist für dieses Beispiel nicht geeignet, hierbei sinkt sogar der p-Wert.
Ebenso ist zu erkennen, dass sich bei der *combined*-Strategie die Größe des Datensatzes nicht deutlich verändert. Beim Oversampling erhöht sich die Größe jedoch deutlich und beim Undersampling sinkt sie. Diese Eigenschaften sollten wegen der Rechenzeit bedacht werden.

David Schader

5.1.5 Mehrere sensitive Attribute

In diesem Beispiel wird darauf eingegangen, ob das Resampling auch zu einer Fairness-Steigerung führen kann, wenn mehrere sensitive Attribute berücksichtig werden müssen. Dabei wird neben dem bisher gewählten sensitiven Attribut *race* auch *sex* berücksichtigt. Die Schwierigkeit liegt bei diesem Beispiel dabei, dass als Gruppen Tupel erstellt werden und es so viel Gruppen wie mögliche Tupel bestehend aus einem *sex*- und einem *race*-Element gibt.

In Abbildung 5.5 ist das Beispiel dargestellt. Dabei wurde als Fairness-Metrik wieder die *Equal Opportunity* und als ML-Modell eine SVM-Modell gewählt. Außerdem wurden hier wieder wie im vorigen Beispiel alle Resampling-Methoden gleichzeitig getestet, um zu sehen, welche Version das maximale Potential für einen fairen Algorithmus aufweist.

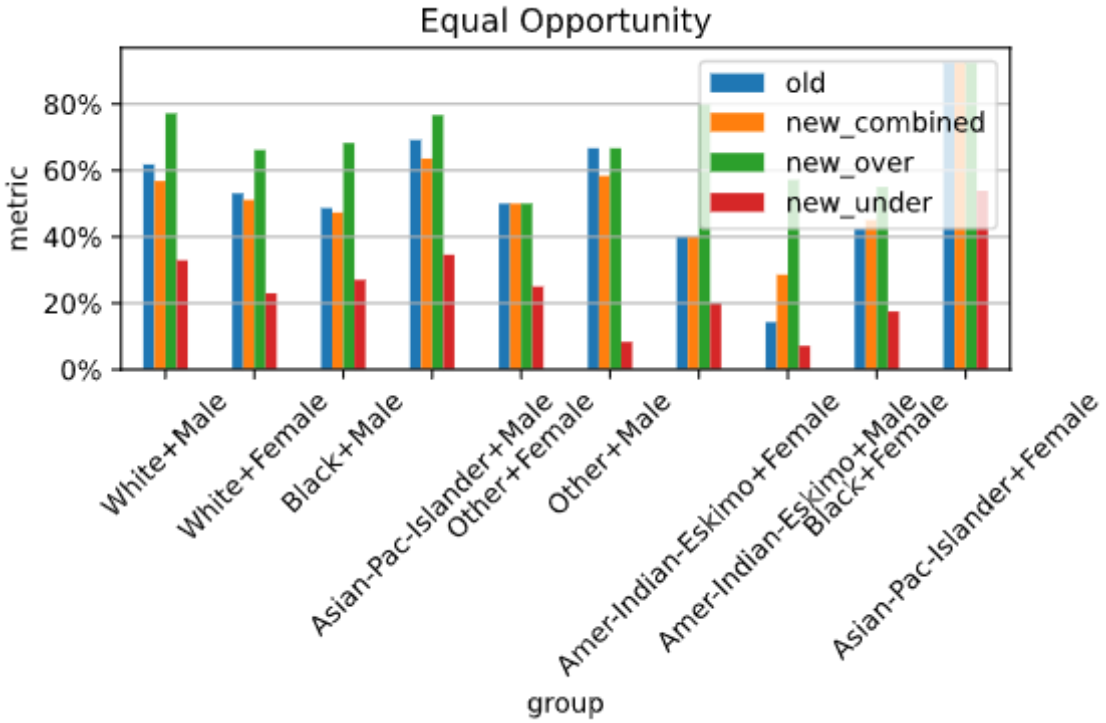


Abbildung 5.5: *adults*-Debiasing nach *race* und *sex* mit *Equal Opportunity*

Die unterschiedlichen p-Werte für die unterschiedlichen Resampling-Methoden sind in Tabelle 5.2 dargestellt.

resampling	p
ohne Resampling (old)	0,15
combined	0,31
over	0,54
under	0,13

Tabelle 5.2: p-Werte für unterschiedliche Resampling-Methoden mit zwei sensitiven Attributen

Hier zeigt sich wieder, dass reines Oversampling die besten Ergebnisse bringt. Es zeigt sich aber auch, dass eine Verbesserung der Fairness bei zwei sensitiven Attributen deutlich schwieriger ist als nur bei einem sensitiven Attribut. In diesem Beispiel kommt man nur

zu einem p-Wert von 0,54. Dieser ist jedoch noch nicht über der 80%-Akzeptanzgrenze. Allgemein verbessert sich die Fairness sowohl beim Oversampling als auch bei der *combined*-Option. Wie auch im vorigen Beispiel zeigt sich, dass sich der vorliegende Datensatz nicht dazu eignet, das Undersampling anzuwenden, da hierbei die Fairness noch deutlich verschlechtert wird.

Insgesamt konnte hiermit aber gezeigt werden, dass es prinzipiell möglich ist, mit dem implementierten Ansatz ein Debiasing bei unfairen Algorithmen durchzuführen. Um optimale Ergebnisse zu erhalten, muss jedoch individuell der vorliegenden Datensatz betrachtet werden. Möchte man optimale Ergebnisse erhalten, ist es wichtig, mit der Auswahl des ML-Algorithmus, der Anzahl sensitiver Attribute, der Fairness-Metrik und der Resampling-Methode die Möglichkeiten so anzupassen, dass man für den vorliegenden Datensatz ein optimales Debiasing durchführen kann.

David Schader

5.2 Kreditwürdigkeit

Als zweites Beispiel wird ein Datensatz betrachtet, der die Kreditwürdigkeit von Personen mit Attributen wie etwa *Beruf, Alter, Kredithöhe* aber auch *Familienstand/Geschlecht* beinhaltet. Dieser Datensatz ist unter `https://www.openml.org/d/31` zu finden und besitzt 21 Attribute, darunter eine binäre Klasseninformation von 1.000 Personen. Diese Daten sollen dazu verwendet werden, für Personen anhand der gegebene Attribute eine Kreditwürdigkeit vorherzusagen, die lediglich zwischen *good* und *bad* unterscheidet.

Um die Ergebnisse des Debiasing bei diesem Datensatz mit dem vorherigen vergleichen zu können, werden die gleichen Einstellungen, also SVM als Klassifizierungsalgorithmus, *Equal Opportunity* als Fairness-Metrik und ein Trainings-Test-Verhältnis von $2:1$, verwendet. Als sensitives Attribut kommt in diesem Fall der Familienstand (*personal_status*) in Frage. Außerdem werden alle drei Resampling-Verfahren angewendet. Die Ergebnisse dieser Evaluation sind in Abbildung 5.6 zu sehen.

Betrachtet man die Abbildung und die Ergebnisse in Tabelle 5.3, kann erschlossen werden, dass die Fairness bei allen Resampling-Verfahren um etwa 20% gestiegen ist und damit sogar den kritischen Wert von 80% erreicht hat. Hierbei ist hervorzuheben, dass sogar das Undersampling sehr gute Ergebnisse liefert, obwohl nach den ersten Untersuchungen mit dem *adult*-Datensatz eher eine Verschlechterung zu erwarten war. Dies zeigt, dass die Effektivität des Resampling-Verfahrens von Datensatz zu Datensatz unterschiedlich aussehen kann und deshalb nach Möglichkeit alle Verfahren in Betracht gezogen werden sollten.

Denis Dengler

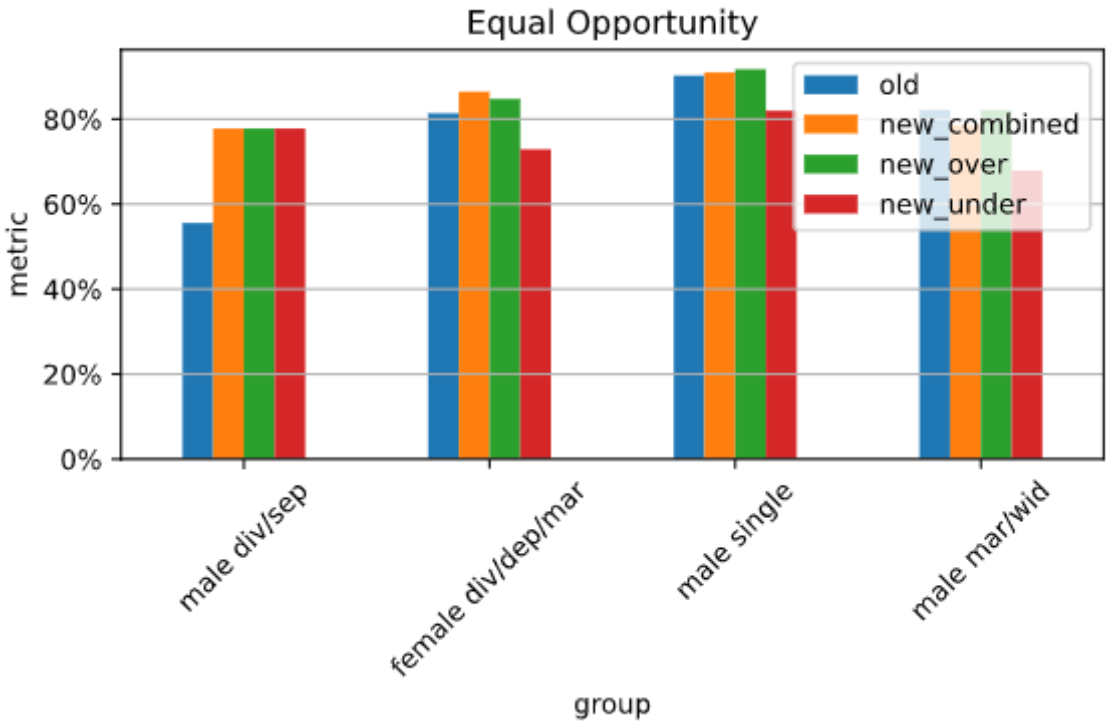


Abbildung 5.6: *credit-g*-Debiasing nach *personal-status* mit *Equal Opportunity*

resampling	p	Datensatz-Größe
ohne Resampling (old)	0,621	670
combined	0,854	656
over	0,847	704
under	0,828	510

Tabelle 5.3: Effizienz bei unterschiedlichen Resampling-Verfahren *credit-g*

5.3 Rückfälligkeit von Straftätern

Im letzten Beispiel soll der bekannte COMPAS-Datensatz untersucht werden. Dieser wird unter `https://github.com/propublica/compas-analysis/` von *ProPublica* bereitgestellt und enthält Informationen zu der Rückfälligkeit von Straftätern mit Attributen wie *Geschlecht, Alter, Anzahl an Vorstraftaten* und auch *Ethnie*. Als Klassenbezeichnungen wird die Rückfälligkeit nach zwei Jahren verwendet. Für diesen Test wurde der Datensatz auf die wesentlichen Merkmale reduziert, da dieser ursprünglich Informationen wie *Vor- und Nachname*, *Geburtsdatum* oder *Datum der Inhaftierung* beinhaltet, die für diese Klassifizierung nicht von Relevanz sind und zudem häufig unvollständig sind. Der reduzierte Datensatz besteht aus zehn Attributen und der Klasse, wobei das Attribut *race* als sensitives Attribut verwendet wird. Dieser Datensatz ist deshalb interessant, weil der COMPAS-Algorithmus, welcher diese Daten zur Einschätzung der Rückfälligkeit vor

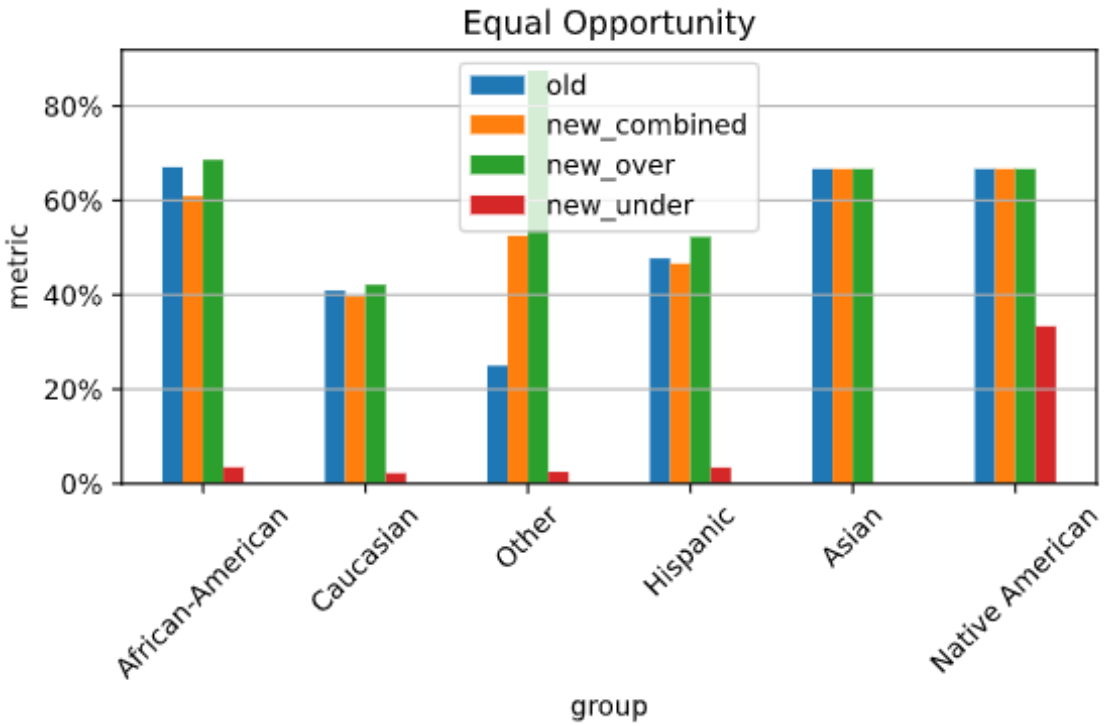


Abbildung 5.7: *compas-scores-two-years*-Debiasing nach *race* mit *Equal Opportunity*

resampling	p	Datensatz-Größe
ohne Resampling (old)	0,373	4.833
combined	0,594	4.755
over	0,482	5.051
under	0,0	3.715

Tabelle 5.4: Effizienz bei unterschiedlichen Resampling-Verfahren COMPAS

Gericht verwendet, in der Vergangenheit zu großen Diskussionen geführt hat. Häufig wurden Rassismusvorwürfe geäußert, da die Vorhersagen scheinbar dunkelhäutige Personen benachteiligen und häufiger falsch positiv klassifizieren.
Auch in diesem Fall sollen die gleichen Bedingungen erfüllt sein. Die Ergebnisse der Evaluation für alle drei Resampling-Verfahren sind in Abbildung 5.7 und in Tabelle 5.4 zu sehen. Es ist zu beobachten, dass die Fairness beim kombinierten Resampling um etwa 20% steigt, jedoch bei weitem nicht den kritischen Wert erreicht. Zudem fällt auf, dass die Fairness nach dem Undersampling bei 0 liegt. Dies ist damit zu erklären, dass beim Löschen von Datenobjekten womöglich jene Objekte entfernt wurden, von denen nur sehr wenige vorhanden waren, sodass die betroffene Gruppe nach dem Resampling nicht mehr vertreten ist. Hier sollte darauf geachtet werden, dass beim Undersampling nur dann Daten gelöscht werden, wenn ein solcher Fall auszuschließen ist.

6 Ergebnisse und Ausblick

Im Rahmen dieser Arbeit sollte der Frage nachgegangen werden, inwiefern Verzerrungen (*Bias*) in Daten, welche zum Trainieren von Machine Learning (ML)-Algorithmen verwendet werden, erkannt, gemessen und korrigiert werden können. Zusätzlich sollten Methoden gefunden werden, trainierte ML-Algorithmen bezüglich ihrer Fairness gegenüber bestimmten sensitiven Attributen wie etwa dem Geschlecht oder ethnischer Herkunft zu bewerten und auf Basis dessen die zugrundeliegenden Trainingsdaten zu entzerren, bzw. zu *debiasen*. Dafür wurden zunächst die Grundlagen des Maschinellen Lernens, der Datenverzerrung und der algorithmischen Fairness auf Basis aktueller und einschlägiger Fachliteratur erarbeitet und darauf aufbauend Methoden und Vorgehensweisen zum Messen und Korrigieren von Bias und Diskriminierung in Daten und Modellen vorgestellt. Es hat sich herausgestellt, dass das Resampling die vielversprechendste Möglichkeit ist, um Daten zu debiasen.

Deshalb wurde ein Konzept für ein Debiasing-Framework entwickelt, welches Daten auf Verzerrungen untersucht, Modelle auf Fairness bewertet und zudem verschiedene Resampling-Strategien zum Korrigieren eines zugrundeliegenden Bias zur Verfügung stellt. Um dies zu erreichen, werden die Daten basierend auf den sensitiven Attributen gruppiert und auf Fairness bewertet. Anschließend werden sie nach ihrem Fairness-Wert neu gewichtet. Anhand dieser Gewichtungen kann eine neue Mengenverteilung innerhalb des Trainingsdatensatzes errechnet und durch Verwendung von Over- und Undersampling-Verfahren erreicht werden. Die transformierten Daten können anschließend erneut auf Fairness untersucht werden.

Zur Evaluation wurden drei verschiedene Datensätze verwendet und mit unterschiedlichen Metriken und Resampling-Strategien unter Verwendung der Support Vector Machine (SVM) zur Klassifizierung transformiert und bewertet. Bei Betrachtung der Fairness-Messungen vor und nach dem Resampling fällt auf, dass sich das Ergebnis allgemein deutlich verbessert hat. Das Ergebnis ist jedoch abhängig von vielen Faktoren, wie Klassifizierungsalgorithmus, Fairness-Metrik und Resampling-Strategie. Da die Wahl des Klassifizierungsalgorithmus und der Metrik meist durch Anforderungen des Zielsystems vorgegeben werden, hängt das Ergebnis primär von der Resampling-Strategie ab. Das Framework bietet dazu drei Strategien: Oversampling mit Borderline-SMOTE-Algorithmus, Undersampling mit NearMiss-Algorithmus oder eine Kombination der beiden Verfahren. In den meisten Fällen ist Oversampling oder die kombinierte Variante die beste Wahl. Beim Undersampling kann es je nach Datensatz dazu führen, dass relevante Datenobjekte gelöscht werden und damit die Fairness sinkt.

Dies wäre der erste Punkt, welcher in der nächsten Version des Frameworks korrigiert werden könnte. Es wäre in diesem Kontext interessant, andere Undersampling-Verfahren wie CNN oder RENN zu testen. Diese wurden in dieser Version nicht betrachtet, da sie kein kontrolliertes Undersampling ermöglichen. Weiter könnte untersucht werden, inwiefern ein Verstärkungsfaktor bei der Gewichtung oder mehrfaches Resampling Einfluss auf die Fairness haben und ob eine solche Anpassung der Implementierung sinnvoll wäre. Des Weiteren wäre es sinnvoll das Resampling auf Basis mehrerer sensitiven Attribute erneut zu untersuchen und zu optimieren, da die Ergebnisse im Vergleich zur Verwendung eines Attributs etwas schlechter ausfielen.

Insgesamt konnte jedoch gezeigt werden, dass mit dem implementierten Ansatz Datensätze, die einen Bias enthalten, erfolgreich entzerrt werden können und damit ein Debiasing von ML-Algorithmen möglich ist.

Denis Dengler

Literatur

[1] Julia Angwin, Jeff Larson, Lauren Kirchner und Surya Mattu. „Machine Bias“. In: *ProPublica* (23.05.2016). URL: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[2] Inga Döbel, Miriam Leis, Manuel Molina Vogelsang, Dmitry Neustroev, Henning Petzka, Stefan Rüping, Angelika Voss, Martin Wegele und Juliane Welz. *Maschinelles Lernen – Kompetenzen, Anwendungen und Forschungsbedarf*. URL: https://www.bigdata-ai.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/BMBF_Fraunhofer_ML-Ergebnisbericht_Gesamt.pdf.

[3] Sofia Visa, Brian Ramsay, Anca L. Ralescu und Esther van der Knaap. „Confusion Matrix-based Feature Selection“. In: 2011, S. 120–127. URL: https://www.researchgate.net/publication/220833270_Confusion_Matrix-based_Feature_Selection.

[4] Sarang Narkhede. „Understanding AUC - ROC Curve - Towards Data Science“. In: *Towards Data Science* (26.06.2018). URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[5] Nathan Kallus und Angela Zhou. *The Fairness of Risk Scores Beyond Classification: Bipartite Ranking and the xAUC Metric*. URL: https://arxiv.org/pdf/1902.05826.

[6] Jaspreet. „Understanding and Reducing Bias in Machine Learning“. In: *Towards Data Science* (5.04.2019). URL: https://towardsdatascience.com/understanding-and-reducing-bias-in-machine-learning-6565e23900ac.

[7] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman und Aram Galstyan. *A Survey on Bias and Fairness in Machine Learning*. URL: https://arxiv.org/pdf/1908.09635.

[8] „Bias in Machine Learning“. In: *ForeSee Medical* (21.04.2020). URL: https://www.foreseemed.com/blog/bias-in-machine-learning.

[9] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama und Adam Kalai. *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*. URL: https://arxiv.org/pdf/1607.06520.

[10] Alexandra Geese, Aurélie Pols, Vincent C. Müller und Matthias Spielkamp. *Interactive Panel II: Mitigation options - What can be done to identify and address current and future challenges of emerging technologies*. 23.03.2021. URL: https://www.europarl.europa.eu/stoa/en/events/details/policy-options-for-the-ethical-governanc/20210303WKS03282.

[11] Dana Pessach und Erez Shmueli. *Algorithmic Fairness*. URL: http://arxiv.org/pdf/2001.09784v1.

[12] Simon Caton und Christian Haas. *Fairness in Machine Learning: A Survey*. URL: http://arxiv.org/pdf/2010.04053v1.

[13] Jannik Dunkelau und Michael Leuschel. „Fairness-Aware Machine Learning: An Extensive Overview“. Diss. Düsseldorf: Heinrich-Heine-Universität Düsseldorf. URL: https://www.phil-fak.uni-duesseldorf.de/fileadmin/Redaktion/Institute/Sozialwissenschaften/Kommunikations-_und_Medienwissenschaft/KMW_I/Working_Paper/Dunkelau___Leuschel__2019__Fairness-Aware_Machine_Learning.pdf.

[14] Harini Suresh und John V. Guttag. *A Framework for Understanding Unintended Consequences of Machine Learning*. URL: https://arxiv.org/pdf/1901.10002.

[15] Karen Grace-martin. „How to Diagnose the Missing Data Mechanism“. In: *The Analysis Factor* (20.05.2013). URL: https://www.theanalysisfactor.com/missing-data-mechanism/.

[16] Alvira Swalin. „How to Handle Missing Data - Towards Data Science“. In: *Towards Data Science* (31.01.2018). URL: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.

[17] Satyam Kumar. „7 Ways to Handle Missing Values in Machine Learning“. In: *Towards Data Science* (24.07.2020). URL: https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e.

[18] Jason Brownlee. „3 Ways to Encode Categorical Variables for Deep Learning“. In: *Machine Learning Mastery* (2019-11-21). URL: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/ (besucht am 21. 04. 2021).

[19] Faisal Kamiran und Toon Calders. „Data preprocessing techniques for classification without discrimination“. In: *Knowledge and Information Systems* 33.1 (2012), S. 1–33. ISSN: 0219-1377. DOI: 10.1007/s10115-011-0463-8. URL: https://www.researchgate.net/publication/228975972_Data_Pre-Processing_Techniques_for_Classification_without_Discrimination.

[20] T. Calders, F. Kamiran und M. Pechenizkiy. „Building Classifiers with Independency Constraints“. In: *2009 IEEE International Conference on Data Mining Workshops*. 2009, S. 13–18. ISBN: 2375-9259. DOI: 10.1109/ICDMW.2009.83.

[21] Ashrya Agrawal, Florian Pfisterer, Bernd Bischl, Jiahao Chen, Srijan Sood, Sameena Shah, Francois Buet-Golfouse, Bilal A. Mateen und Sebastian Vollmer. *Debiasing classifiers: is reality at variance with expectation?* URL: http://arxiv.org/pdf/2011.02407v1.

[22] *3. Under-sampling — Version 0.8.0*. 2.03.2021. URL: https://imbalanced-learn.org/stable/under_sampling.html.

[23] Show-Jane Yen und Yue-Shi Lee. „Cluster-based under-sampling approaches for imbalanced data distributions“. In: *Expert Systems with Applications* 36.3 (2009), S. 5718–5727. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2008.06.108. URL: https://www.sciencedirect.com/science/article/pii/S0957417408003527.

[24] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu und Jing-Shang Jhang. „Clustering-based undersampling in class-imbalanced data“. In: *Information Sciences* 409-410.1 (2017), S. 17–26. ISSN: 00200255. DOI: 10.1016/j.ins.2017.05.008.

[25] Navoneel Chakrabarty. „Implementation of Cluster Centroid based Majority Undersampling Technique (CCMUT) in Python“. In: *Towards Data Science* (2018-10-26). URL: https://towardsdatascience.com/implementation-of-cluster-centroid-based-majority-under-sampling-technique-ccmut-in-python-f006a96ed41c (besucht am 18. 04. 2021).

[26] Jianping Zhang und Inderjeet Mani. *kNN approach to unbalanced data distributions: a case study involving information extraction*. 2003. URL: https://www.site.uottawa.ca/~nat/workshop2003/jzhang.pdf.

[27] Jason Brownlee. „Undersampling Algorithms for Imbalanced Classification“. In: *Machine Learning Mastery* (19.01.2020). URL: https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/.

[28] Ivan Tomek. „Two Modifications of CNN“. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.11 (1976), S. 769–772. ISSN: 2168-2909. DOI: 10.1109/TSMC.1976.4309452.

[29] Dennis L. Wilson. „Asymptotic Properties of Nearest Neighbor Rules Using Edited Data“. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2.3 (1972), S. 408–421. ISSN: 0018-9472. DOI: 10.1109/TSMC.1972.4309137.

[30] Ivan Tomek. „An Experiment with the Edited Nearest-Neighbor Rule“. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.6 (1976), S. 448–452. ISSN: 2168-2909. DOI: 10.1109/TSMC.1976.4309523.

[31] P. Hart. „The condensed nearest neighbor rule (Corresp.)“ In: *IEEE Transactions on Information Theory* 14.3 (1968), S. 515–516. ISSN: 0018-9448. DOI: 10.1109/TIT.1968.1054155.

[32] Ethem Alpaydin. „Voting over Multiple Condensed Nearest Neighbors“. In: *Artificial Intelligence Review* 11.1/5 (1997), S. 115–132. ISSN: 0269-2821. DOI: 10.1023/A:1006563312922. URL: https://www.researchgate.net/publication/2644628_Voting_over_Multiple_Condensed_Nearest_Neighbors.

[33] Miroslav Kubat und Stan Matwin. *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection*. 2000. URL: https://www.researchgate.net/publication/2624358_Addressing_the_Curse_of_Imbalanced_Training_Sets_One-Sided_Selection.

[34] Jorma Laurikkala. „Improving Identification of Difficult Small Classes by Balancing Class Distribution“. In: *Artificial Intelligence in Medicine*. Hrsg. von Silvana Quaglini, Steen Andreassen und Pedro Barahona. SpringerLink Bücher. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2001, S. 63–66. ISBN: 978-3-540-48229-1.

[35] Ramin Ghorbani und Rouzbeh Ghousi. „Comparing Different Resampling Methods in Predicting Students' Performance Using Machine Learning Techniques“. In: *IEEE Access* 8.1 (2020), S. 67899–67911. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2986809. URL: https://www.researchgate.net/publication/340567248_Comparing_Different_Resampling_Methods_in_Predicting_Students%27_Performance_Using_Machine_Learning_Techniques.

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall und W. P. Kegelmeyer. „SMOTE: Synthetic Minority Over-sampling Technique“. In: *Journal of Artificial Intelligence Research* 16 (2002), S. 321–357. ISSN: 1076-9757. DOI: 10.1613/jair.953. URL: https://www.jair.org/index.php/jair/article/view/10302.

[37] B. C. Wallace, K. Small, C. E. Brodley und T. A. Trikalinos. „Class Imbalance, Redux“. In: *2011 IEEE 11th International Conference on Data Mining*. 2011, S. 754–763. ISBN: 2374-8486. DOI: 10.1109/ICDM.2011.33.

[38] Hui Han, Wen-Yuan Wang und Bing-Huan Mao. „Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning“. In: *Advances in Intelligent Computing*. Hrsg. von De-Shuang Huang, Xiao-Ping Zhang und Guang-Bin Huang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 878–887. ISBN: 978-3-540-31902-3.

[39] Hien M. Nguyen, Eric W. Cooper und Katsuari Kamei. „Borderline over-sampling for imbalanced data classification“. In: *International Journal of Knowledge Engineering and Soft Data Paradigms* 3.1 (2011), S. 4. ISSN: 1755-3210. DOI: 10.1504/IJKESDP.2011.039875.

[40] Felix Last, Georgios Douzas und Fernando Bacao. „Oversampling for Imbalanced Learning Based on K-Means and SMOTE“. In: *Information Sciences* 465.1 (2018), S. 1–20. ISSN: 00200255. DOI: 10.1016/j.ins.2018.06.056. URL: https://arxiv.org/pdf/1711.00837.

[41] Haibo He, Yang Bai, E. A. Garcia und Shutao Li. „ADASYN: Adaptive synthetic sampling approach for imbalanced learning“. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, S. 1322–1328. ISBN: 2161-4407. DOI: 10.1109/IJCNN.2008.4633969.

[42] Guido van Rossum und Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697. URL: https://www.python.org/.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und E. Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830. ISSN: 1533-7928. URL: https://scikit-learn.org/.

[44] Guillaume Lemaître, Fernando Nogueira und Christos K. Aridas. „Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning“. In: *Journal of Machine Learning Research* 18.17 (2017), S. 1–5. ISSN: 1533-7928. URL: http://jmlr.org/papers/v18/16-365.html.

[45] The pandas development team. *pandas-dev/pandas: Pandas*. 2020. DOI: 10.5281/zenodo.3509134. URL: https://pandas.pydata.org/.

[46] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez und Krishna P. Gummadi. *Fairness Constraints: Mechanisms for Fair Classification*. URL: http://arxiv.org/pdf/1507.05259v5.