BACHELORARBEIT

Implementierung und Evaluation ergänzender Korrekturmethoden für statistische Lernverfahren bei unbalancierten Klassifikationsproblemen

Tobias Kühn

Betreuung:
Prof. Dr. Bernd Bischl

Institut für Statistik
Ludwig-Maximilians-Universität München
15. Oktober 2014

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und alle verwendeten Quellen und Hilfsmittel benannt habe.

München, den 15. Oktober 2014

.....
(Tobias Kühn)

Danksagung

Diese Bachelorarbeit entstand am Institut für Statistik der Ludwig-Maximilians-Universität München. An dieser Stelle möchte ich mich bei meinem Betreuer, Prof. Dr. Bernd Bischl, der mir die Arbeit an diesem interessanten Thema ermöglicht hat, für die freundliche und engagierte Betreuung, die Unterstützung bei der Durchführung der Experimente sowie die vielen hilfreichen Anregungen und Ideen bedanken.

Abstract

Im Falle binärer Klassifikationsprobleme liefern statistische Lernverfahren bei Vorliegen stark unbalancierter Klassen oftmals keine zufriedenstellenden Ergebnisse insbesondere hinsichtlich der Vorhersage von Beobachtungen aus der kleinen Klasse. Diese sind jedoch in vielen Beispielen aus der Praxis wie z.B. Credit Scoring, Betrugserkennung oder Stornovorhersage gerade von großem Interesse. Ziel dieser Arbeit ist daher die Evaluation und der Vergleich verschiedener Methoden zur Korrektur des Klassenungleichgewichts, die in Kombination mit einem Lernverfahren als Klassifikator auf unbalancierte Daten angewendet werden können. Zu diesem Zweck wurden im Rahmen dieser Arbeit mehrere Korrekturmethoden in R implementiert sowie in das über CRAN verfügbare Paket mlr (Machine Learning in R) integriert. Die betrachteten Methoden lassen sich dabei allgemein in Sampling- und kostenbasierte Ansätze unterscheiden, wobei der Fokus dieser Arbeit auf den Sampling-basierten Korrekturmethoden liegt. Als Gütemaße für die Performance der Verfahren werden AUC (Fläche unter der ROC-Kurve) sowie der F1-Score betrachtet, welche für unbalancierte Klassifikationsprobleme besser geeignet sind als beispielsweise die Gesamtgenauigkeit der Vorhersagen über alle Klassen (Accuracy).
In einer Vielzahl von Experimenten wurden insgesamt 23 öffentlich zugängliche Datensätze evaluiert - dabei wurden jeweils die betrachteten Lernverfahren (Logistische Regression, Klassifikationsbaum, Random Forest, Gradient Tree Boosting, Support Vector Machine) mit und ohne Parameter-Tuning sowie in Kombination mit verschiedenen Korrekturmethoden auf die einzelnen Datensätze angewendet. Es zeigt sich, dass bereits durch geeignete Wahl des Lernverfahrens, ggf.mit ergänzendem Tuning der Hyperparameter, sehr gute Ergebnisse erzielt werden können. Auch die zusätzliche Verwendung der Korrekturmethoden führt häufig zu weiteren Verbesserungen des AUC und F1-Score. Hierbei liefern insbesondere das SMOTE-Verfahren sowie Class Weighting für verschiedene Datensätze und unter zusätzlicher Berücksichtigung der Laufzeit gute Ergebnisse.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Bei der Betrachtung von Klassifikationsproblemen kommt dem Verhältnis der vorliegenden Klassen eine entscheidende Bedeutung zu. Im Idealfall steht je Klasse etwa die gleiche Anzahl an Beobachtungen zur Verfügung. Typischerweise ist es jedoch der Fall, dass bestimmte Klassen in den Daten deutlich unterrepräsentiert sind. Auch bei der binären Klassifikation, d.h. bei Klassifikationsproblemen mit nur zwei Klassen (z.B. positive und negative Klasse) kommt es häufig vor, dass die Anzahl der Beobachtungen einer Klasse (i.d.R. der negativen Klasse) die der Anderen (i.d.R. der positiven Klasse) deutlich übertrifft. Bezüglich der Notation und den Bezeichnungen der beiden Klassen innerhalb der binären Klassifikation soll fortan gelten – kleine Klasse $\Leftrightarrow$ positive Klasse $\Leftrightarrow$ 1 sowie große Klasse $\Leftrightarrow$ negative Klasse $\Leftrightarrow$ 0 oder -1. Als einführendes Beispiel seien hierzu Kreditwürdigkeitsprüfungen im Bankenbereich (Credit Scoring) genannt – im Vorfeld einer Kreditvergabe soll dabei anhand der Daten des jeweiligen Kreditnehmers vorhergesagt werden, ob ein Kredit zurückgezahlt wird oder nicht. Zu beachten ist hierbei, dass Kreditausfälle zum einen deutlich seltener auftreten (kleine Fallzahl der positiven Klasse) und zum anderen die fälschliche Einstufung eines Kreditnehmers als kreditfähig (mit späterem Kreditausfall) i.d.R. mit höheren Kosten verbunden ist als die Ablehnung eines eigentlich kreditwürdigen Kunden. Somit sind die Fälle der kleinen bzw. positiven Klasse, die Kreditausfälle, von größerem Interesse und verursachen bei Fehlklassifikation höhere Kosten. Im Fokus dieser Arbeit stehen somit im Allgemeinen binäre Klassifikationsprobleme, welche eine stark unbalancierte Verteilung der Klassen vorweisen und bei denen darüber hinaus die jeweiligen Fehlklassifikationen bei der Prognose unterschiedlich bewertet werden, d.h. Fehlklassifikationen innerhalb der kleinen Klasse werden als teurer eingestuft als Fehlklassifikationen innerhalb der großen Klasse. Neben dem bereits genannten Bereich des Credit Scoring sind derartige Klassifikationsprobleme in einer Vielzahl weiterer Bereiche wie beispielsweise medizinische Diagnostik [21], Fehlerdiagnose bei Transformatoren [26], Betrugserkennung [10, 25], Stornovorhersage [7] oder Erkennung von Objekten auf Satellitenbildern [**?**] anzutreffen.

Die Prognosegüte von statistischen Verfahren und Algorithmen in Bezug auf binäre Klassifikationsprobleme wird im Allgemeinen anhand der Vorhersagegenauigkeit (Accuracy) oder Fehlklassifikationsrate (1 - Accuracy) über alle Klassen hinweg bewertet. Je höher

dabei die Genauigkeit bzw. je geringer die Fehlklassifikationsrate, desto besser das Verfahren. Im Falle unbalancierter Klassen sind diese Maße eher ungeeignet, da z.B. trotz sehr guter Vorhersagegenauigkeit über beide Klassen, speziell die Beobachtungen der kleinen Klasse nur sehr schlecht erkannt werden. Liegen beispielsweise Daten mit einem Klassenverhältnis von 1:99 vor, d.h. pro Beobachtung der kleinen Klassen existieren 99 Beobachtungen der großen Klasse, so wird bereits eine Vorhersagegenauigkeit von 99% allein durch die ausschließliche Prognose der großen Klasse erreicht. Beobachtungen der kleinen Klasse, die jedoch wie am Beispiel Credit Scoring verdeutlicht, häufig zusätzlich von größerem Interesse sind, werden in diesem Fall zu 100% fehlklassifiziert. Das Problem der beiden genannten Maße – Accuracy und Fehlklassifikationsrate – besteht somit darin, dass nicht zwischen den Fehlklassifikationen je Klasse unterschieden, sondern die Fehlklassifikationsrate über alle Beobachtungen (und somit Klassen) aggregiert betrachtet wird. Durch die Dominanz der großen Klasse neigen statistische Lernverfahren bei Optimierung bezüglich Accuracy oder Fehlklassifikationsrate dazu, vorhandene Beobachtungen der kleinen Klasse als Rauschen zu ignorieren und dementsprechend für neue Beobachtungen verstärkt die große Klasse vorherzusagen [13]. Um jedoch insbesondere die Beobachtungen der kleinen Klasse gut erkennen zu können, existieren zahlreiche Methoden, um dem Problem unbalancierter Klassen entgegen zu wirken. Grundsätzlich kann hierbei zwischen Sampling- und kostenbasierten Verfahren unterschieden werden. Als geeignetere Performance-Maße zur Schätzung der Prognosegüte wird i.d.R. die Receiver Operating Characteristic (ROC)-Kurve und die dazugehörige „Area under curve“ (AUC) sowie der F1-Score verwendet.

Der Hauptteil dieser Arbeit ist in insgesamt vier Kapitel aufgeteilt. In Kapitel 2 werden zunächst die angewendeten Lernverfahren sowie die Methoden zur Korrektur des Klassenungleichgewichts vorgestellt. Bezüglich den kostenbasierten Korrekturmethoden und der damit verbundenen Einführung von Fehlklassifikationskosten ist i.d.R. grundlegendes Fachwissen notwendig, um diese sinnvoll bzw. korrekt festlegen zu können. Da die tatsächlich anfallenden Fehlklassifikationskosten in vielen Fällen unbekannt sind, liegt der Schwerpunkt dieser Arbeit auf den alternativ genannten Sampling-basierten Methoden. Kapitel 2 enthält des Weiteren eine Beschreibung der verwendeten Performance-Maße – Area Under Curve (AUC) und F1-Score. In Kapitel 3 werden die in den durchgeführten Experimenten verwendeten Datensätze und Parametereinstellungen dargestellt. Des Weiteren wird eine kurze Übersicht zu den genutzten R-Paketen und Funktionen gegeben. In den im Rahmen dieser Arbeit betrachteten Experimenten werden statistische Lernverfahren sowohl mit als auch ohne vorheriges Parameter-Tuning (mittels Iteraed F-Racing [20]) und in Kombination mit Korrekturmethoden für insgesamt 23 unbalancierte Datensätze evaluiert und verglichen. Die maßgebliche Fragestellungen stellen dabei die folgenden Punkte dar:

- Gegenüberstellung und Beurteilung der Lernverfahren
- Analyse des Tuning-Effektes, d.h. Vergleich der Ergebnisse der Lernverfahren mit und ohne vorheriges Tuning der Hyperparameter
- Analyse des Effektes der Korrekturmethoden, d.h. Vergleich der Lernverfahren mit und ohne Einsatz verschiedener Korrekturmethoden

Hierbei soll insbesondere analysiert werden, in welchen Fällen die passende Wahl des Lernverfahrens (inklusive Tuning) ggf. bereits ausreichend ist, d.h. in welchen Fällen der Einsatz zusätzlicher Korrekturmethoden eher hilfreich oder eher vernachlässigbar ist. Eine entsprechende Vorstellung und Diskussion der Ergebnisse findet sich in Kapitel 4. Die Arbeit schließt mit einer kurzen Zusammenfassung sowie einem Ausblick mit weiterführenden Fragestellungen in Kapitel 5.

Diese Arbeit stellt eine Fortführung der Analysen und Experimente des Konferenzberichtes „On Class Imbalancy Correction for Classification Algorithms in Credit Scoring“ [1] dar. Erweiterungen und Unterschiede zum Bericht sind in Abschnitt 3.4 dargestellt.

2. Methoden und Verfahren

2.1. Statistische Lernverfahren

2.1.1. Logistische Regression

Die logistische Regression [12] stellt das wohl am weitesten verbreitete und bekannteste statistische Lernverfahren für binäre Klassifikationsprobleme dar. Im Unterscheid zur linearen Regression wird hierbei der Erwartungswert einer binären bzw. Bernoulli-verteilten Zielgröße Y (bedingt auf den Kovariablenvektor $\boldsymbol{x}$) durch ein Modell angepasst.

$$y \in \{0,1\} \qquad \begin{matrix} 0: \text{ negative Klasse} \\ 1: \text{ positive Klasse} \end{matrix} \tag{2.1}$$

$$Y \sim Ber(\pi), \quad E(Y|\boldsymbol{x}) = \mathbb{P}(Y=1|\boldsymbol{x}) = \pi \ \in [0,1]$$

Der Erwartungwert der Zielgröße Y entspricht dabei gerade der Wahrscheinlichkeit π (vgl. Formel 2.1), so dass letztendlich die Wahrscheinlichkeit für das Eintreten der positiven Klasse („Y = 1“) modelliert wird. Da der Wertebereich der Wahrscheinlichkeit π im Intervall [0,1] liegt, der lineare Prädiktor $\boldsymbol{x'\beta}$ jedoch prinzipiell auch Werte außerhalb dieses Intervalls annehmen kann, werden die beiden Größen π und $\boldsymbol{x'\beta}$ durch eine Responsefunktion h bzw. eine Linkfunktion $g = h^{-1}$ miteinander verknüpft:

$$h(\eta_i) = h(\boldsymbol{x_i'\beta}) = \pi_i, \quad g(\pi_i) = \eta_i = \boldsymbol{x_i'\beta}, \quad i = 1,..,n \tag{2.2}$$

Der lineare Prädiktor $\eta_i = \boldsymbol{x_i'\beta}$ besteht wie auch bei der linearen Regression aus mehreren unabhängigen Variablen bzw. Features, die sowohl metrisch als auch kategorial sein können. Bei der Responsefunktion h handelt es sich grundsätzlich um eine streng monoton wachsende Verteilungsfunktion. Im Rahmen dieser Arbeit wird dabei Logit-Modell

betrachtet, bei dem die logistische Funktion als Responsefunktion verwendet wird. Als Umkehrfunktion ergibt sich hierbei die logarithmierte Chance (log-odds oder „Logit“) als Linkfunktion:

$$\pi_i = \frac{1}{1+\exp(\eta_i)} \quad \Leftrightarrow \quad logit(\pi_i) = \ln\left(\frac{\pi_i}{1-\pi_i}\right) = \eta_i = \boldsymbol{x_i'\beta} \tag{2.3}$$

Wie in Formel 2.3 dargestellt, ergibt sich bei Verwendung des Logit-Links ein lineares Modell für die logarithmierten Chancen $\ln\left(\frac{\pi_i}{1-\pi_i}\right)$ sowie durch weitere Transformation mit der Exponentialfunktion ein (exponentiell-)multiplikatives Modell für die so genannten Chancen bzw. Odds $\frac{\pi_i}{1-\pi_i}$. Die Modellanpassung bzw. Schätzung der Parameter erfolgt üblicherweise über die Maximum-Likelihood-Schätzung. Da hierbei nach Nullsetzen der Score-Gleichungen ein mehrdimensionales, nichtlineares Gleichungssystem entsteht, kommt zur Lösung der ML-Gleichungen i.d.R. ein iteratives Verfahren wie z.B. Newton-Raphson- oder Fisher-Scoring-Algorithmus zum Einsatz. Nach der Schätzung der Koeffizienten $\hat{\boldsymbol{\beta}}$ lässt sich für neue Beobachtungen die geschätzte Wahrscheinlichkeit $\hat{\pi}_{n+1} = \mathbb{P}(y_{n+1} = 1|\boldsymbol{x_{n+1}})$, d.h. die Wahrscheinlichkeit, dass eine neue Beobachtung der positiven Klassen zuzuordnen ist, durch Einsetzen von $\boldsymbol{x'_{n+1}}\hat{\boldsymbol{\beta}}$ in die Responsefunktion berechnen. Die entsprechende Klasse, d.h. $\hat{y}_{n+1} = 1$ (positive Klasse) oder $\hat{y}_{n+1} = 0$ (negative Klasse), wird anhand der prognostizierten Wahrscheinlichkeit sowie eines festgelegten Schwellenwertes c (i.d.R. $c = 0.5$) abgeleitet. Im Allgemeinen wird somit $\hat{y} = 1$ vorhergesagt, falls $\hat{\pi} \geq c$ und alternativ $\hat{y} = 0$, falls $\hat{\pi} < c$.

Eine ausführliche Beschreibung zur logistischen Regression findet sich bei Fahrmeir et al., S. 270 - 293 [12]. Für die Umsetzung der Experimente im Rahmen dieser Arbeit wurde die Implementierung der logistischen Regression im R-Paket stats (Funktion glm() mit family=binomial(link=logit)) verwendet.

2.1.2. Entscheidungsbaum (CART)

Bei der Verwendung eines Entscheidungsbaums (bei binärer Zielgröße auch: Klassifikationsbaum) werden die vorliegenden Daten anhand der jeweiligen Variablen und deren Ausprägungen schrittweise in disjunkte und möglichst homogene Untergruppen (→ Partitionen) zerlegt. Das von Breiman et al. [6] eingeführte CART-Verfahren (Classification And Regression Tree) zählt hierbei zu einem der gängisten Implementierungen. Es handelt sich um ein nicht-parametrisches Verfahren, bei dem die Aufteilung der Daten in jedem Schritt in zwei disjunkte Gruppen („binäre Splits“) sowie anhand eines der Features erfolgt. Durch die wiederholte Anwendung der Vorgehensweise auf die entstehenden

Partitionen ergibt sich eine wie beispielhaft in Abbildung 2.1 dargestellte Baumstruktur.

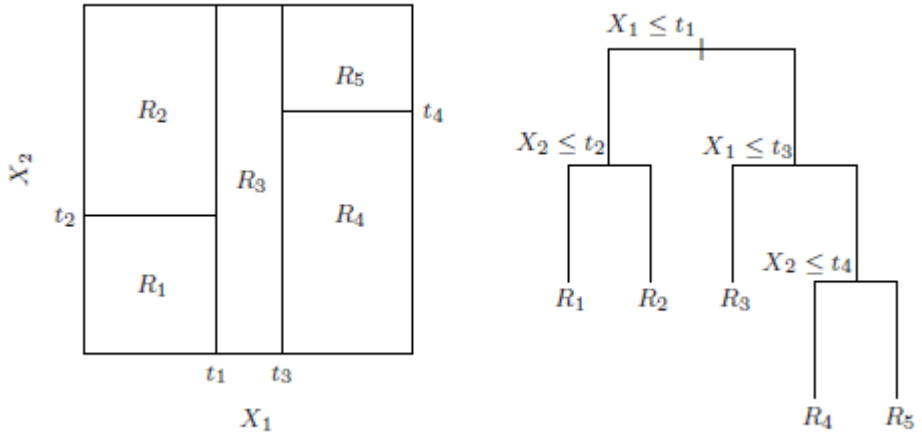

Abbildung 2.1.: Partitionierung und Klassifikationsbaum mittels CART (vgl. Hastie et al., S. 306 [16])

Die linke Grafik in Abbildung 2.1 veranschaulicht die Aufteilung eines zweidimensionalen Raumes, d.h. eines Modell mit zwei Features X_1, X_2 mittels CART-Verfahren. In der rechten Grafik ist der zugehörige Klassifikationsbaum dargestellt. Zur Aufteilung der Trainingsdaten in möglichst homogene Gruppen sowie zur Ermittlung der besten Variable x_j sowie des dazugehörigen besten Punktes/Merkmalsausprägung s wird bei kategorialen und binären Zielgrößen je Split ein so genanntes Unreinheitsmaß wie z.B. Fehlklassifikationsrate, Devianz oder Gini-Index berechnet. Der im Rahmen dieser Arbeit genutzte Gini-Index ergibt sich für einen Knoten m bei der binären Klassifikation (Anzahl Klassen $K = 2$) vereinfacht wie folgt:

$$\begin{aligned} G_m &= \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\ &= \hat{p}_{m0}(1 - \hat{p}_{m0}) + \hat{p}_{m1}(1 - \hat{p}_{m1}) \\ &= 2\hat{p}_{m1}(1 - \hat{p}_{m1}), \qquad \hat{p}_{m1} = 1 - \hat{p}_{m0} \end{aligned} \tag{2.4}$$

Im Zwei-Klassen-Fall bezeichnen $\hat{p}_{m0}$ und $\hat{p}_{m1}$ in Formel 2.4 die relativen Häufigkeiten der Klassen 0 und 1 im Knoten m. Das Minimum für G_m ($\rightarrow G_m = 0$) wird jeweils erreicht, wenn alle Beobachtungen auschließlich einer der beiden Klassen angehören. Für alle möglichen, resultierenden Partitionen m_L und m_R „unterhalb“ des Knoten m lassen sich nun die Gini-Indizes berechnen und anschließend gewichtet mit der Anzahl der Beobach-

tungen in den neuen Partitionen N_L und N_L summieren. Letztendlich wird dann derjenige Split, d.h. diejenige Kovariable x_j sowie der dazugehörige Punkt/Merkmalsausprägung s gewählt, welche die gewichtete Summe der Gini-Indizes in den resultierenden Partitionen minimiert (vgl. Formel 2.5).

$$min(j,s)[\frac{N_L}{N}G_L + \frac{N_R}{N}G_R] \tag{2.5}$$

Die sukzessive Zerlegung der Daten kann prinzipiell so lange fortgeführt werden bis in jedem End- bzw. Terminalknoten des Baumes jeweils nur Beobachtungen einer der beiden Klassen oder im Extremfall nur noch genau eine Beobachtung vorliegt. Da hierdurch jedoch die Komplexität und damit auch die Gefahr des Overfitting des Modells an die zugrunde liegenden Trainingsdaten erheblich ansteigt, existieren verschiedene Stopp-Kriterien, die vor Erzeugung des Baumes gesetzt werden und so die Anzahl der Knoten und Splits begrenzen. Alternativ kann der Klassifikationsbaum zunächst komplett konstruiert und anschließend gemäß bestimmter Kriterien wieder gestutzt werden (Pruning). Im Rahmen dieser Arbeit wurde auf Pruning verzichtet und stattdessen diverse Stopp-Kriterien wie die vorgegebene minimale Anzahl an Beobachtungen pro Knoten und die minimale Verbesserung der Anpassungsgüte, die je Split erreicht werden muss, vorgegeben.

Die Vorhersage der Klassenzugehörigkeit für neue Beobachtungen erfolgt anhand des angepassten Baumes, indem dieser für die neuen Daten sowie deren Merkmale und Ausprägungen durchlaufen wird. Im jeweiligen Terminalknoten wird die Klassifizierung anhand einer Mehrheitswahl (Majority Vote) durchgeführt, d.h. die Klasse mit der größten relativen Häufigkeit im Terminalknoten wird vorhergesagt.
Weiterführende Information zu Entscheidungs- bzw. Klassifikationsbäumen sind bei Hastie et al., S.305ff [16] zu finden. Für die durchgeführten Experimente wurde das R-Paket rpart mit der gleichnamigen Funktion rpart() als Implementierung des CART-Verfahrens genutzt.

2.1.3. Random Forest

Random Forests [5] zählen zu den so genannten (homogenen) Ensemble-Methoden. Die Klassifizierung erfolgt hierbei grundsätzlich durch wiederholte Anwendung eines Lernverfahrens auf Bootstrap-Samples der Trainingsdaten sowie der anschließenden Aggregation der einzelnen Ergebnisse ($\rightarrow$ Bagging = „Bootstrap-Aggregation“). Der Random Forest stellt dabei im Speziellen eine Erweiterung der in Abschnitt 2.1.2 vorgestellten Klassi-

fikationsbäume dar. Die Bagging-Methode wird hierbei auf Bäume angewendet, so dass mehrere Klassifikationsbäume an die Daten angepasst werden. Die Klassifizierung neuer Daten erfolgt anschließend durch Betrachtung aller angepassten Modelle/Bäume sowie entsprechender Mehrheitswahl (Majority Vote), d.h. die Klasse, welche in den meisten Modellen vorhergesagt wurde, wird auch insgesamt prognostiziert ($\rightarrow$ Modus der Vorhersagen der einzelnen Modelle). Jedes Modell erhält somit grundsätzlich das gleiche Gewicht.

Bezüglich der Vorgehensweise werden zunächst mehrere Bootstrap-Stichproben durch zufälliges Ziehen mit Zurücklegen aus den Trainingsdaten erzeugt und diese jeweils per Klassifikationsbaum angepasst. Durch die Vielzahl der angepassten Bäume und deren Aggregation wird die Varianz der resultierenden Vorhersagen i.d.R. grundsätzlich verringert. Sind die angepassten Bäume jedoch stark miteinander korreliert, kann die Varianz der Vorhersagen für neue Daten ggf. ansteigen. Um die Klassifikationsbäume möglichst zu dekorrelieren, wird daher (neben dem Bootstrapping) in den einzelnen Splits der Bäume eine zufällige Selektion der in Frage kommenden Merkmale bzw. Features durchgeführt. Anstatt der ursprünglichen p Variablen werden je Split somit nur $m < p$ zufällig ausgewählte, potentielle Split-Variablen betrachtet. Je kleiner der Parameter m, desto geringer i.d.R. die Korrelation zwischen den Bäumen bzw. desto unterschiedlicher auch die Struktur der Bäume. Der Parameter m kann grundsätzlich frei gewählt oder über ein Tuning-Verfahren bestimmt werden – als Faustregel wird bei p verfügbaren Features häufig $m = \sqrt{p}$ als Faustregel angegeben (vgl. Hastie et al., S. 592 [16]). Wie bereits beschreiben, erfolgt die finale Vorhersage der Klasse abschließend per Mehrheitswahl (Majority Vote) aller angepassten Bäume. Ist eine Vorhersage der Klassenwahrscheinlichkeiten erforderlich, kann die relative Häufigkeit der vorhergesagten Klassen der einzelnen Modelle herangezogen werden.

Die Anzahl der Modelle/Bäume ist frei wählbar oder kann über ein Tuning-Verfahren bestimmt werden. Alternativ lässt sich die Anzahl der Bäume auch aus dem so genannten OOB-Error (Out-Of-Bag Error) ableiten. Dabei handelt es sich um die mittlere Fehlklassifikationsrate je Beobachtung, wobei die Vorhersagen ausschließlich auf Basis der Bäume gebildet werden, bei deren Aufbau die jeweilige Beobachtung nicht verwendet wurde. Der OOB-Error wird parallel zur Durchführung des Verfahrens berechnet und stellt damit sozusagen ein im Algorithmus integriertes Kreuzvalidierungsverfahren dar. Prinzipiell sind bei Betrachtung des OOB-Errors nur so viele Modellanpassungen notwendig bis sich dieser stabilisiert hat (vgl. Hastie et al., S. 592/593 [16]).
Weiterführende Informationen zu Random Forests finden sich bei Hastie et al., S. 587ff [16]. Im Rahmen dieser Arbeit wurde zur Durchführung der Experimente die Funktion randomForest() basierend auf CARTs aus dem gleichnamigen R-Paket verwendet.

2.1.4. Gradient (Tree) Boosting

Gradient Boosting [16] stellt neben dem Random Forest (vgl. Abschnitt 2.1.3) eine weitere, so genannte Ensemble-Methode dar. Hierbei wird somit wiederum ein Lernverfahren bzw. eine Basis-Methode mehrfach auf die vorliegenden Trainingsdaten angewendet. Im Unterschied zum Random Forest werden die einzelnen Modelle jedoch nicht separat betrachtet und angepasst, sondern in einem iterativen Verfahren zu einem additiven Gesamtmodell mit mehreren Basisfunktionen $b(x_i; \gamma_m)$ zusammengefasst (vgl. Formel 2.6). Die Parameter γ_m kennzeichnen dabei die jeweiligen Modellparameter der m-ten Basis-Methode.

$$F(x) = \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m) \tag{2.6}$$

Die einzelnen Basisfunktionen $b(x_i; \gamma_m)$ werden über die Koeffizienten β_m gewichtet. Die Schätzung der Koeffizienten erfolgt dabei über die Minimierung des Verlustes für eine zu bestimmende Verlustfunktion L:

$$\underset{\beta,\gamma}{argmin} \sum_{i=1}^{N} L(y_i, F(x_i)) = \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)) \tag{2.7}$$

Als Basis-Methoden sind grundsätzlich beliebige Verfahren und Funktionen denkbar. Bei dem im Rahmen dieser Arbeit betrachteten Gradient Tree Boosting werden Regressionsbäume ($\rightarrow$ CART) bzw. Baumstümpfe, d.h. Bäume mit geringer Tiefe, verwendet. Bäume bieten dabei den grundsätzlichen Vorteil, dass sie an sich bereits eine additive Struktur besitzen (vgl. Hastie et al., S. 359ff [16]). Die Tiefe des Baumes stellt im Rahmen des Modells grundsätzlich einen frei wählbaren bzw. über ein Tuning-Verfahren zu bestimmenden Modellparameter dar.
Anstatt nun das komplette Modell, d.h. die komplette Funktion $F(x)$ anzupassen, wird der erwartete Verlust sukzessive in jeder Iteration nur bezüglich einer Basisfunktion minimiert. Bereits im Modell enthaltene Komponenten werden nicht mehr verändert. In Jeder Iteration wird eine weitere additive Komponente zum Modell hinzugefügt, so dass der erwartete Verlust dadurch weiter reduziert wird.
Zur Bestimmung der Modellparameter in Iteration m, wird zunächst ausgehend von der aktuellen Funktion F_{m-1} die Richtung des negativen Gradienten des Verlustes anhand der so genannten Pseudo-Residuen r_i berechnet:

$$r_{im} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i)}{\partial F_{m-1}(x_i)}\right], \quad i = 1, .., n \tag{2.8}$$

In Formel 2.8 bezeichnet r_{im} dabei das Pseudo-Residuum in Iteration m bezüglich der i-ten Beobachtung. Im Falle der binären Klassifikation wird zur Berechnung der Pseudo-Residuen i.d.R. der binomiale Verlust ($\rightarrow$ Devianz) als Verlustfunktion verwendet, welcher zugleich der negativen Log-Likelihood der logistischen Regression entspricht.

$$L(y, f(x)) = -log\,(1 + exp(-2yf(x))) \tag{2.9}$$

Für die berechneten Pseudo-Residuen r_{im} wird anschließend ein Regressionsbaum $\sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$ angepasst, wobei R_{jm} die j-te Partition des m-ten Baumes und γ_{jm} den in dieser Partition konstant vorhergesagten Wert – i.d.R. den Mittelwert der Daten in dieser Partition – bezeichnet. Die Schätzung der Parameter γ_{jm} erfolgt dabei i.d.R. über die Minmierung des quadratischem Verlustes.

$$\gamma_m = \underset{\gamma}{argmin} \sum_{i=1}^{N} (r_{im} - b(x_i, \gamma))^2 = \sum_{i=1}^{N} (r_{im} - \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{2.10}$$

Nach der m-ten Iteration wird die Funktion F_m somit gemäß Formel 2.11 durch eine neue, additive Komponente erweitert. Dieser neu hinzugefügt Regressionbaum $\sum_{j=1}^{J} \gamma_j I(x \in R_j)$ bewegt das Modell durch die Anpassung an die Pseudo-Residuen sozusagen einen Schritt in die Richtung der stärksten Verringerung des Verlust.

$$F_m(x) = F_{m-1}(x) + \beta_m b(x_i; \gamma_m) = f_{m-1}(x) + \beta_m \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{2.11}$$

Die optimale Anzahl an Iterationen stellt einen weiteren Parameter des Gesamtmodells dar und wird i.d.R. durch Kreuzvalidierung oder vorheriges Tuning bestimmt. Zur Vermeidung von Overfitting kann die Anzahl der Iterationen / die Anzahl der additiven Komponenten grundsätzlich beschränkt werden. Alternativ besteht die Möglichkeit durch Multiplikation der einzelnen Komponenten mit einem Shrinkage-Parameter $\nu \in (0, 1]$ der Überanpassung an die Daten entgegen zu wirken. Der Shrinkage-Parameter (oder auch Lernrate) ν bewirkt, dass das Modell in jeder Iteration nicht mit der optimalen Schrittweite, sondern nur ein entsprechend kürzeres Stück in Richtung des negativen Gradienten bewegt wird. Die optimalen Einstellungen bezüglich der Anzahl der Iterationen m sowie der Lernrate ν hängen dabei stark voneinander ab ($\rightarrow$ kleinere Lernrate führt zu größerer Anzahl an notwendiger Iterationen um ein bestimmtes Ergebnis zu erreichen und umgekehrt).

Eine einfache Erweiterung des Gradient Boosting stellt das Stochastic Gradient Boosting dar. Anstatt in den einzelnen Iterationen stets alle zur Verfügung stehenden Trainingsdaten zu verwenden, wird je Iteration nur eine per Subsampling zufällig generierte Teilmenge der Trainingsdaten angepasst. Hierdurch wird das Gradient (Tree) Boosting zusätzlich um die Vorteile (allerdings auch die Nachteile) der Bagging-Verfahren erweitert.

Eine ausführlichere Beschreibung zu Boosting-Verfahren allgemein sowie zum Gradient Tree Boosting findet sich bei Hastie et al., S. 337ff [16]. Für die in dieser Arbeit durchgeführten Experimente wurde die Funktion gbm() aus dem gleichnamigen R-Paket gbm verwendet.

2.1.5. Support Vector Machine (SVM)

Support Vector Machines [16] sind so genannte „Large Margin Classifier“, bei denen zwei Klassen derart durch eine Hyperebene getrennt werden, dass zusätzlich der Abstand zwischen den Klassen und der Hyperebene maximal wird. Mittels des maximal breiten Abstandes soll insbesondere gewährleistet werden, dass auch neue Beobachtungen möglichst gut klassifiziert werden können.

Der mit Vorzeichen versehene Abstand eines Punktes $x_i \in \mathbb{R}^p$ von der in Abbildung 2.2

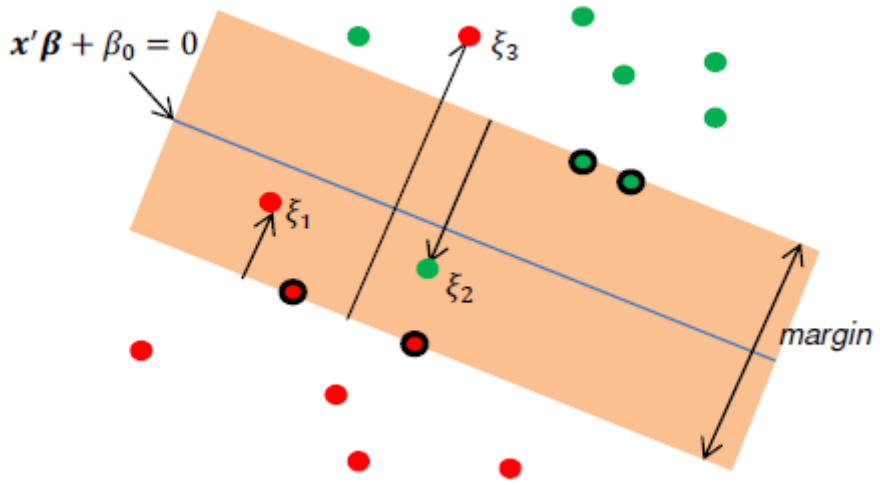

Abbildung 2.2.: Ansatz der Support Vector Machines inklusive Schlupfvariablen (vgl. Hastie et al., S. 418 [16]

dargestellten Hyperebene $\boldsymbol{x'\beta} + \beta_0 = 0$ mit Einheitsvektor β ($\rightarrow ||\beta|| = 1$) kann mittels $f(\boldsymbol{x_i}) = \boldsymbol{x_i'\beta} + \beta_0$ bestimmt werden (vgl. Hastie et al., S. 418 [16]). Wird weiterhin

eine binäre Zielgröße als $y_i \in \{-1, 1\}$ (1: positive Klasse, -1: negative Klasse) kodiert, entspricht in diesem Fall ein positiver Wert von $f(x_i)$, d.h. ein positiver Abstand von x_i zur Hyperebene, der Zuordnung der Beobachtung i zur positiven Klasse ($\rightarrow y_i = 1$). Als Klassifizierungsregel ergibt sich somit $\hat{y}_i = sign(f(x_i))$.
Sind die beiden Klassen linear trennbar, so lässt sich eine Hyperebene an die Daten anpassen, so dass für alle Beobachtungen gilt: $y_i \cdot f(x_i) > 0$, d.h. alle Beobachtungen können anhand der Hyperebene korrekt klassifiziert werden. Mittels des in Formel **??** dargestellten Optimierungsproblems ist des Weiteren eine Maximierung des absoluten Abstandes ($\rightarrow y_i \cdot f(x_i)$) zwischen der Hyperebene und den Klassen (und somit zwischen den Klassen) möglich.

$$\begin{aligned} &\max_{\beta,\beta_0,||\beta||=1} \gamma \\ \text{mit NB:} \quad &y_i \cdot f(x_i) = y_i \cdot (x_i'\beta + \beta_0) \geq \gamma, \quad i = 1, .., N \end{aligned} \tag{2.12}$$

Durch Umskalierung kann weiterhin der Wegfall der Annahme $||\beta|| = 1$ erreicht werden. Die Daten werden hierbei i.d.R. so skaliert, dass der Abstand der so genannten Supportvekoren zur Hyperebene genau 1 beträgt. Als Supportvektoren werden die Beobachtungen bzw. Vektoren bezeichnet, deren Abstand zur Hyperebene minimal ist ($\rightarrow \min_i(y_i \cdot f(x_i))$) und die dadurch den Verlauf der Hyperebene maßgeblich beeinflussen. In Abbildung 2.2 sind die Supportvektoren mit schwarzer Umrandung dargestellt. Nach Skalierung ergibt sich das zu Formel 2.12 äquivalente Optimierungsproblem:

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 \\ \text{mit NB:} \quad &y_i \cdot (x_i'\beta + \beta_0) \geq 1, \quad i = 1, .., N \end{aligned} \tag{2.13}$$

Für den Fall nicht linear trennbarer Klassen kann die in Formel 2.13 dargestellte Optimierung durch die Einführung so genannter Schlupfvariablen (slack variables) $\xi_i \geq 0$ erweitert werden. Diese erlauben eine Fehlklassifizierung von Beobachtungen, wobei die Anzahl der Fehlklassifizierungen bzw. die Abstände der fehlklassifizierten ($\xi_i > 1$) und innerhalb der Margin liegenden ($0 < \xi_i < 1$) Beobachtungen vom Rand der Margin so gering wie möglich gehalten werden. In Abbildung 2.2 sind beispielhaft drei entsprechende Beobachtungen dargestellt – ξ_1 liegt dabei zwischen 0 und 1, für die fehlklassifizierten Beobachtungen gehörenden Schlupfvariablen ξ_2, ξ_3 gilt: $\xi_2 > 1$ bzw. $\xi_3 > 1$. Zusammengefasst werden somit zwei Ziele verfolgt – zum einen die Anpassung einer Hyperebene mit größtmöglichem Abstand zu den beiden Klassen und zum anderen die Minimierung der über die Schlupfvariablen erlaubten Fehlklassifizierungen. Diese prinzipiell gegensätzlichen Ziele werden i.d.R. innerhalb des Optimierungsproblems in Form einer gewichteten Sum-

me ausgedrückt (vgl. Formel 2.14).

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C \cdot \sum_{i=1}^{N} \xi_i \\ &\text{mit NB:} \quad y_i \cdot (x_i'\beta + \beta_0) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{2.14}$$

Die in Formel **??** dargestellte, positive Konstante C steuert dabei den Ausgleich zwischen den beiden genannten Zielen. Die Wahl des Parameters C wird i.d.R. durch Anwendung eines Tuning-Verfahren bestimmt. Je größer C dabei gewählt wird, umso kleiner wird die Margin bzw. umso stärker wird der Fokus auf die korrekte Klassifizierung der Beobachtungen gelegt. Da es sich in Formel **??** um ein konvexes Optimierungsproblem handelt, kann über die Lagrange-Funktion und weitere Optimalitätsbedingungen eine Lösung für β, β_0 bestimmt werden: $\rightarrow \hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i$ (vgl. Hastie et al., S. 420f [16]).

Bei dem bis hierher beschriebenen Vorgehen handelt es sich prinzipiell um ein lineares Klassifikationsverfahren, d.h. die Klassifizierung ist nur anhand einer linearen Funktion möglich und bei nicht linear trennbaren Klassen kommen die Schlupfvariablen zum Einsatz. Dieser Ansatz kann jedoch erweitert werden, in dem die Trainingsdaten bzw. -vektoren in einen höherdimensionalen Raum transformiert werden. Mit steigender Anzahl der Dimension lassen sich dadurch selbst sehr verschachtelte bzw. sich stark überlappende Klassen linear trennen. Somit kann die gesuchte Hyperebene prinzipiell in einem höherdimensionalen Raum bestimmt und anschließend wieder in den Ursprungs-Vektorraum zurücktransformiert werden. In Abbildung 2.3 ist beispielhaft die Transformation zweier nicht linear trennbaren Klassen im $\mathbb{R}^2$ in den höherdimensionalen $\mathbb{R}^3$ dargestellt.
Da die Transformation des gesamten Ursprungs-Vektorraums in einen höherdimensionalen Raum i.d.R. viel zu aufwändig und rechenintensiv ist, kommt hierbei der so genannte Kernel-Trick zum Einsatz. Für eine existierende Transformation bzw. Basisfunktion ϕ der Ursprungsdaten in einen höherdimensionalen Raum ($\phi : X \rightarrow H$) wird hierbei über den so genannten Kernel bzw. eine Kernelfunktion k das Skalarprodukt je zweier Vektoren im transformierten Raum berechnet:

$$k: \; X \times X \rightarrow \mathbb{R}, \quad k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle \tag{2.15}$$

Wird der Kernel nun selbst auch als Basisfunktion verwendet, d.h. $\phi(x) = k(x, \cdot)$, so ist das Skalarprodukt zweier transformierter Vektoren gerade gleich der Kernelfunktion mit den jeweiligen Ursprungsvektoren als Funktionsargumente („reproducing property“, vgl. Formel 2.16). Die Berechnung der Skalarprodukte kann aufgrund dieser Eigenschaft

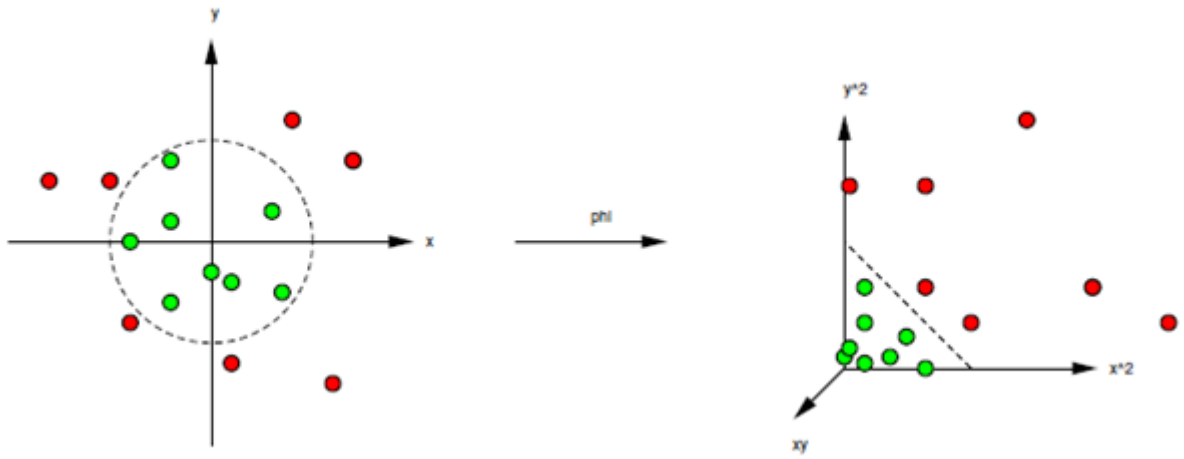

Abbildung 2.3.: Transformation nicht linear trennbarer Klassen ($\mathbb{R}^2 \rightarrow \mathbb{R}^3$)

implizit mittels Kernelfunktion und Ursprungsvektoren erfolgen ohne dass eine Hin- und Rücktransformation der Vektoren durchgeführt werden muss.

$$k(x_1, x_2) = \langle k(x_1, ..), k(x_2, ..) \rangle \tag{2.16}$$

Werden nun in Formel **??** alle Punkte x_i durch die entsprechenden Basisfunktionen ersetzt ($x_i \rightarrow k(x_i, \cdot)$) sowie $\beta = \sum_{i=1}^{N} \alpha_i xi$ ergibt sich als „neues“ Optimierungsproblem:

$$\begin{aligned} &\underset{\alpha,\xi_i}{min} \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) + C \cdot \sum_{i=1}^{N} \xi_i \\ &\text{mit NB:} \quad y_i \cdot (\sum_{j=1}^{N} \alpha_j k(x_i, x_j) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{2.17}$$

Die Klassifizierung einer Beobachtung kann des Weiteren erfolgen mittels:

$$f(x_i) = \sum_{i=1}^{N} \alpha_i k(x, x_i) + \beta_0 \tag{2.18}$$

Diese in den Formeln 2.17 und 2.18 dargestellte„kernelisierte“ Support Vector Machine stellt nun ein nicht-lineares Lernverfahren dar, welches auf beliebig verteilte Klassen angewendet werden kann. Es zeigt sich, dass die Koeffizienten α_i i.d.R. nur für genau die Beobachtungen ungleich 0 sind, welche die Nebenbedingung exakt erfüllen. Dabei handelt

es sich wiederum um die so genannten Supportvektoren, welche die resultierende Hyperebene auch in diesem Fall maßgeblich bestimmen.
Bei der verwendeten Kernelfunktion muss es sich prinzipiell „lediglich“ um eine symmetrische, positiv definite Funktion handeln, wodurch eine Vielzahl an verschiedenen Kernels existieren. Im Rahmen dieser Arbeit wurde der so genannte Radial Basis Kernel bzw. Gauss-Kernel verwendet ($k(x_i, x_j) = exp(-\gamma \cdot ||x_i x_j||^2, \gamma > 0)$, mit welchem die zugrunde liegenden Daten theoretisch in einen unendlich dimensionalen Raum abgebildet werden können. Da dadurch wiederum die perfekte Trennung aller Klassen möglich ist, dienen die ursprünglich zur Separierung nicht linear-trennbarer Klassen eingeführten Schlupfvariablen hauptsächlich zur Vermeidung der Überanpassung des Modells an die zugrunde liegenden Trainigsdaten.
Weitere ausführlichere Informationen zu Support Vector Machines finden sich bei Hastie et al., S. 417ff [16]. Im Rahmen dieser Arbeit wurde zur Durchführung der Experimente die Funktion ksvm() aus dem R Paket kernlab verwendet, welches automatisch auch kategoriale Features verarbeitet sowie die Ausgabe von Klassenwahrscheinlichkeiten unterstützt, die zur Berechnung der betrachteten Gütemaße benötigt werden (vgl. Abschnitt 2.3).

2.2. Methoden zur Korrektur des Klassenungleichgewichts

2.2.1. Überblick

Aufgrund der großen Anzahl binärer Klassifikationsprobleme mit unbalancierten Klassen existiert eine ebenso große Vielfalt an Verfahren, um diesem Problem zu begegnen. Bezüglich des methodischen Ansatzes lassen sich diese im Folgenden allgemein als Korrekturmethoden bezeichneten Verfahren in zwei Gruppen aufteilen – Sampling-basierte sowie kostenbasierte Methoden. Alternativ ist ebenso eine Einteilung in Algorithmus-interne und Algorithmus-externe Verfahren möglich, wobei bei Erstgenannten die Korrektur des Klassenungleichgewichts sozusagen in den jeweiligen Algorithmus integriert ist. Es handelt sich dabei i.d.R um Anpassungen bestehender Lernverfahren für das konkrete Problem unbalancierter, binärer Klassen (z.B. AdaCost oder RUSBoost als Anpassungen des AdaBoost-Algorithmus [13]). Da für einen Großteil dieser Algorithmus-internen Verfahren keine Implementierungen im Programmpaket R zur Verfügung stehen und die Anpassungen bestehender Lernverfahren unter Umständen sehr aufwändig sein können, beschränken sich die Implementierungen und Untersuchungen innerhalb dieser Arbeit aus-

schließlich auf externe Sampling- und kostenbasierte Korrekturmethoden (Wrapper-Based Approaches). Diese bieten insbesondere den Vorteil, dass sie unabhängig von eingesetzten Lernverfahren angewendet und dadurch auch mit beliebigen Lernverfahren kombiniert werden können.

Sampling-basierte Korrekturmethoden (vgl. Abschnitt 2.2.2) greifen unmittelbar in die vorliegenden Trainingsdaten ein und „verändern“ diese entsprechend, um der Unbalanciertheit der Klassen entgegen zu wirken. Grundsätzlich lässt sich hierbei zwischen Undersampling-, Oversampling- und Hybrid-Verfahren unterscheiden, wobei letztere eine Mischform zwischen Under- und Oversampling darstellen. Beim Undersampling wird versucht, die unterschiedlichen Häufigkeiten zwischen den Klassen durch Eliminierung von Beobachtungen der großen Klasse auszugleichen. Im Falle des Oversampling geschieht dies entsprechend durch Vervielfältigung von Beobachtungen der kleinen Klasse. Die vorliegenden Daten werden dabei grundsätzlich vor Anwendung des eigentlichen Lernverfahrens – sozusagen in der Vorverarbeitung – angepasst.
Im Gegensatz hierzu bleiben die zugrunde liegenden Daten bei den kostenbasierten Methoden (Cost-Sensitive Methods) unberührt. Durch Zuweisung unterschiedlicher Kosten für die beiden Fehlklassifikationen ϵ_{01} (wahre Klasse 0, Vorhersage 1) sowie ϵ_{10} (wahre Klasse 1, Vorhersage 0) wird i.d.R. die Fehlklassifikation von Beobachtungen der kleinen Klasse stärker gewichtet und deren korrekter Klassifikation somit eine höhe Priorität verschafft. Auch hierbei können sowohl Lernverfahren angewendet werden, bei denen der Einbezug vorgegebener Kosten unmittelbar in den Algorithmus integriert ist (z.B. Entscheidungsbäume mit kostenbasiertem Unreinheitsmaß [19]) als auch Methoden, die unabhängig vom jeweiligen Lernverfahren eingesetzt werden können (vgl. Abschnitt 2.2.3). Wie auch beim Sampling wird der Fokus auf den Fall der unabhängigen Methoden gelegt.

Sampling- und kostenbasierte Korrekturmethoden unterscheiden sich zwar bezüglich ihres methodischen Ansatzes, sind jedoch grundsätzlich eng miteinander verknüpft. Sofern Fehlklassifikationskosten C_{01} und C_{10} bekannt sind, werden diese i.d.R. in einer Kostenmatrix dargestellt:

	positive prediction	negative prediction
positive class	C_{11}	C_{10}
negative class	C_{01}	C_{00}

Tabelle 2.1.: Kostenmatrix

C_{ij} bezeichnet dabei die Kosten, die entstehen, wenn eine Beobachtung aus Klasse j fälschlicherweise der Klasse i zugeordnet wird. Mittels der beschriebenen Sampling-Verfahren

lassen sich diese unterschiedlichen Fehlklassifikationskosten nun insofern nachbilden, da sich das Kostenverhältnis der jeweilgenen Fehlklassifikationen in den Sampling-Raten wiederspiegelt. D.h. eine Anpassung der Klassenverteilung in den Daten via Sampling entspricht einer Anpassung des Verhältnisses der Fehlklassifikationskosten, so dass besipielsweise eine Verdopplung der Beobachtungen der kleinen Klasse mit der Verdopplung der Fehlklassifikationskosten für Beobachtungen der kleinen Klasse bzw. Halbierung der Fehlklassifikationskosten für Beobachtungen der großen Klasse korrespondiert [11, 18].
Zu beachten ist, dass sich durch das Sampling lediglich das Kostenverhältnisses „wiedergeben“ lässt. Die exakten, absoluten Kosten (sofern bekannt) bleiben bei Sampling-basierten Verfahren unberücksichtigt und fließen nicht direkt in das Verfahren mit ein. Sampling-Verfahren sind daher auch dann gut geeignet, wenn die wahren Fehlklassifikationskosten unbekannt sind. Ein weiterer Unterschied zwischen Sampling- und kostenbasierten Methoden besteht darin, dass die betrachteten Daten bei kostenbasierten Methoden unverändert bleiben, d.h. die Daten selbst werden nicht manipuliert, sondern lediglich mit weiteren Informationen (Kosten bzw. Gewichten) angereichert.

2.2.2. Sampling-Methoden

2.2.2.1. Random Undersampling

Sampling-Methoden werden grundsätzlich zur Anpassung der Klassenverhältnisse in den vorliegenden Daten verwendet, um dadurch den Anteil bzw. das Gewicht der kleinen Klasse zu erhöhen. Einer der grundlegenden Ansätze stellt das Random Undersampling (RUS) dar. Hierbei wird eine festgelegte Anzahl zufällig ausgewählter Beobachtungen der großen Klasse (majority class) aus den Daten entfernt. Die genaue Anzahl ergibt sich dabei anhand der Undersampling-Rate, welche als frei wählbarer Parameter vorab eingestellt oder durch Anwendung eines Tuning-Verfahrens bestimmt werden kann. Die Beobachtungen der kleinen Klasse (minority class) hingegen bleiben unverändert (vgl. Abbildung 2.4).

Da die Anzahl der Beobachtungen im Datensatz durch die Anwendung des Undersampling reduziert wird, verringert sich i.d.R. auch die Laufzeit des jeweils angewendeten Lernverfahrens. Der Nachteil des Random Undersampling besteht in der Eliminierung und damit dem Verlust von ggf. informativen bzw. charakteristischen Beobachtungen der großen Klasse. Des Weiteren können je nach Undersampling-Rate auch unverhältnismäßig viele Beobachtungen verworfen werden – entspricht die Rate beispielweise der so genannten

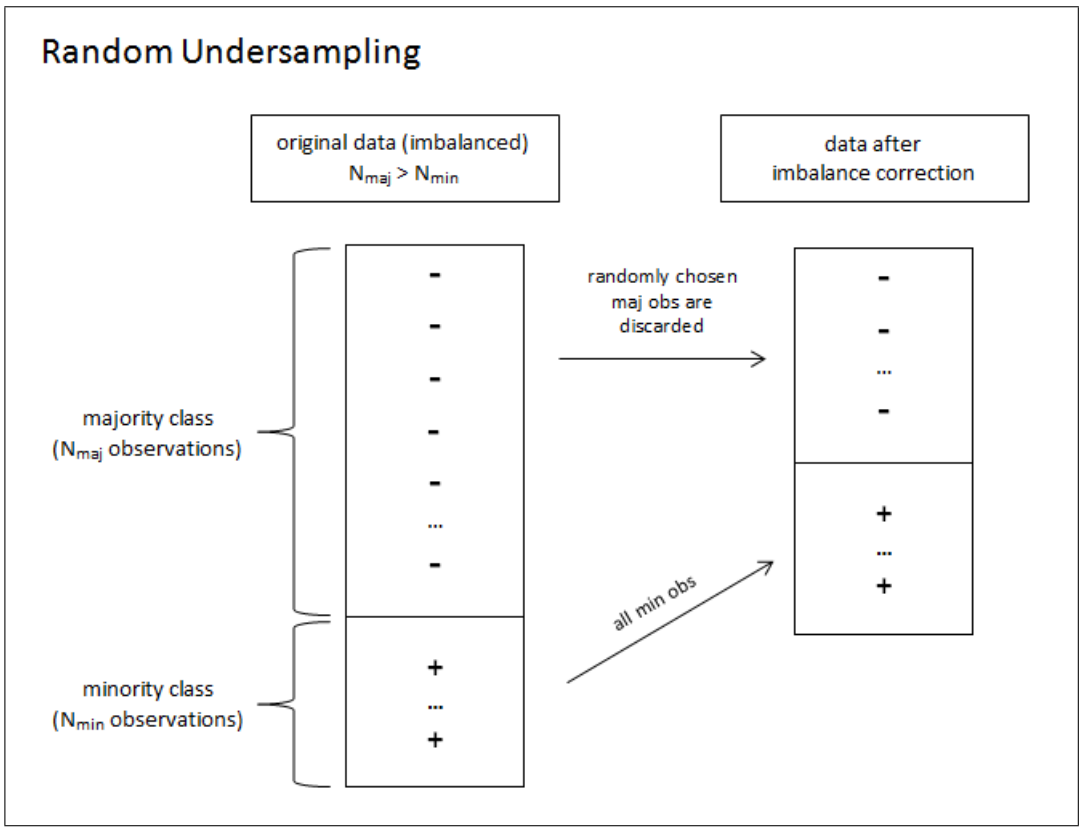

Abbildung 2.4.: Random Undersampling

inversen „imbalance ratio“ (IR) ($\rightarrow$ Anzahl Beobachtungen der kleinen Klasse N_{min} dividiert durch die Anzahl Beobachtungen der großen Klasse N_{maj}) ergibt sich die Größe des resultierenden Datensatzes nach Korrektur mit $N = N_{min} + N_{maj} * (N_{min}/N_{maj}) = 2 \cdot N_{min}$. Das Verhältnis der beiden Klassen ist in diesem Fall nach Durchführung des Undersampling ausgeglichen, je nach Grad der Unbalanciertheit verbleiben jedoch deutlich unter 50% der ursprünglichen Beobachtungen in den Daten.
Über erweiterte Ansätze des Undersampling wie z.B. Condensed Nearest Neighbors [9] wird versucht, Beobachtungen der großen Klasse nicht zufällig, sondern „intelligent“ zu selektieren, so dass vorzugsweise die Beobachtungen entfernt werden, die entweder redundant oder „verrauscht“ sind. Eine weitere Möglichkeit stellt das so genannte Underbagging dar. Gemäß dem Bagging-Ansatz werden hierbei das Undersampling der Daten sowie das anschließende Lernverfahren mehrfach ausgeführt. Da in jeder Iteration / jedem Modell zufällig andere Teilmengen der großen Klasse für die Modellanpassung verbleiben, wird mit steigender Anzahl der Modelle umso wahrscheinlicher, dass jede Beobachtung in zumindest eines der Modelle einfließt und somit nicht komplett verworfen wird.

2.2.2.2. Oversampling

Im Gegensatz zum Undersampling verändern Oversampling-Verfahren die Beobachtungen der kleinen Klasse (minority class). Beim klassischen Oversampling wird diese dabei künstlich vergrößert, in dem „neue“ Beobachtungen durch zufällige Vervielfältigung, d.h. Ziehen mit Zurücklegen aus den ursprünglichen Beobachtungen, zur kleinen Klasse hinzugefügt werden. Jede Beobachtung der kleinen Klasse ist somit mindestens einmal in den Daten enthalten. Die Anzahl der Ziehungen, d.h. der replizierten Beobachtungen richtet sich nach der Oversampling-Rate – diese kann frei gewählt oder mittels Tuning-Verfahren bestimmt werden. Die Beobachtungen der großen Klasse (majority class) bleiben unverändert.

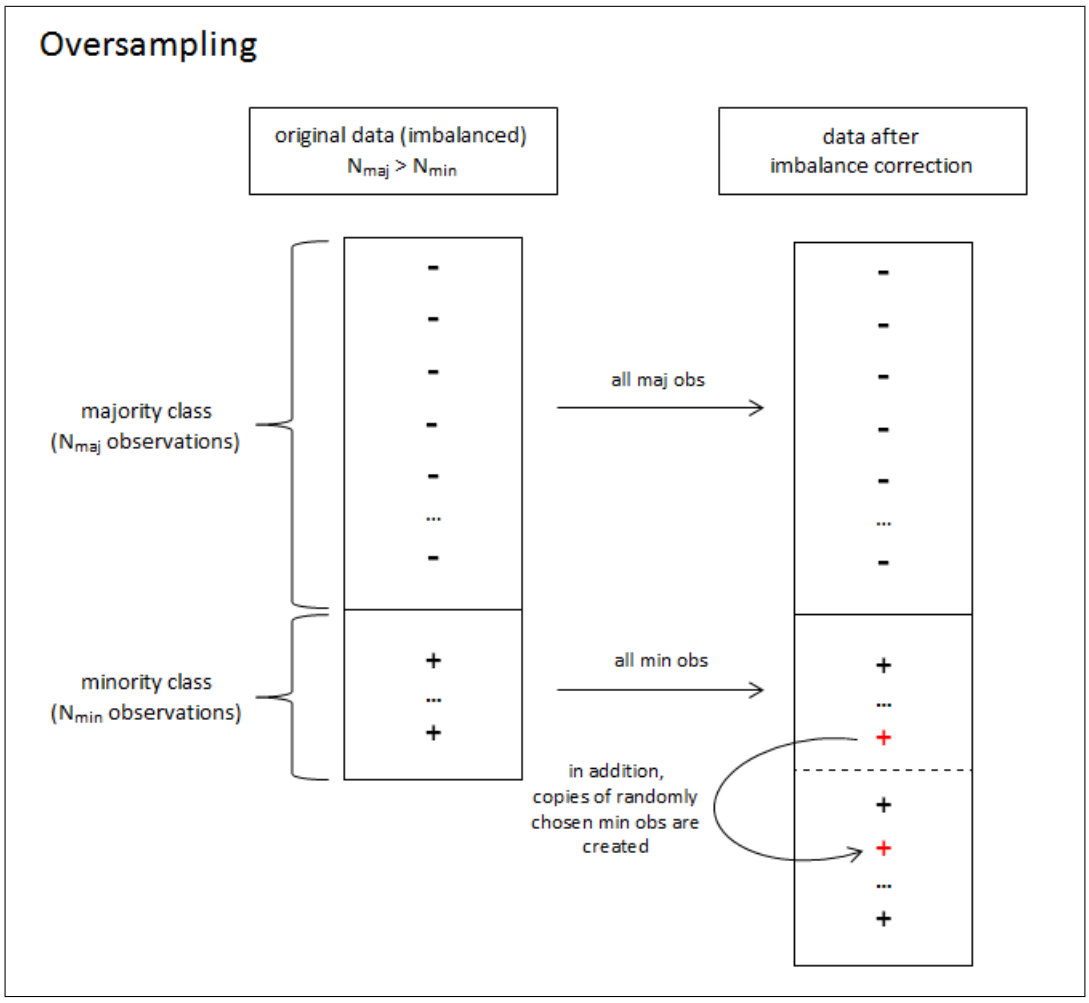

Abbildung 2.5.: Oversampling

Durch die zufällige Vervielfältigung von Beobachtungen und dem daraus folgenden Vorliegen exakter Kopien kann es bei Anwendung des Lernverfahrens leichter zu einer Überanpassung an die Trainingsdaten (Overfitting) kommen. Des Weiteren erhöht sich durch die Ver-

größerung des betrachteten Datensatzes die Laufzeit für die Durchführung des jeweiligen Lernverfahrens. Entspricht die Oversampling-Rate beispielsweise der „imbalance ratio“ (IR) ($\rightarrow$ Anzahl Beobachtungen der großen Klasse N_{maj} dividiert durch die Anzahl Beobachtungen der Klasse N_{min}) steigt die Anzahl der Beobachtungen des Datensatzes nach Korrektur auf $N = N_{min} * (N_{maj}/N_{min}) + N_{maj} = 2 \cdot N_{maj}$. Je nach Grad der Unbalanciertheit kann sich die Größe des Datensatzes im Extremfall dadurch knapp verdoppeln. Erweiterungen und Alternativen zum klassischen Oversampling stellen z.B. das Overbagging sowie SMOTE dar, welche in den beiden folgenden Abschnitten 2.2.2.3 und 2.2.2.4 beschrieben werden.

2.2.2.3. SMOTE

Das SMOTE-Verfahren (Synthetic Minority Oversampling Technique) [8] stellt eine Variante des Oversampling dar, bei dem insbesondere versucht wird, mögliches Overfitting durch die Erzeugung exakter Kopien der Beobachtungen der kleinen Klasse zu vermeiden. Anstelle der Replikation der Beobachtungen, werden neue Beobachtungen durch Interpolation zufällig ausgewählter Beobachtungen der kleinen Klasse (minority class) künstlich erzeugt. Die Anzahl der neuen Beobachtungen richtet sich wie auch beim Oversampling nach der frei wählbaren bzw. durch Tuning-Verfahren zu bestimmenden Oversampling-Rate. Die Beobachtungen der großen Klasse (majority class) bleiben wiederum unverändert.

Zur Konstruktion der neuen, künstlichen Beobachtungen werden zunächst eine Beobachtung der kleinen Klasse sowie jeweils einer der n nächsten Nachbarn dieser Beobachtung zufällig ausgewählt. Die Anzahl der betrachteten nächsten Nachbarn stellt dabei neben der Sampling-Rate einen weiteren frei wählbaren Parameter des Verfahrens dar. Liegen ausschließlich metrische Merkmale vor, werden die nächsten Nachbarn anhand der euklidischen Distanz $d(x_i, x_j) = \sqrt{\sum_{k=1}^{p}(x_{ik}x_{jk})^2}$ bestimmt. Je nach Kontext sollten die Variablen dabei ggf. vorher standardisiert werden. Sind sowohl kategoriale als auch metrische Variablen in den Daten enthalten, kann z.B. die Gower Distanz [14] zur Ermittlung der nächsten Nachbarn verwendet werden (vgl. Formel 2.19). Zur Bestimmung des „Abstandes“ bzw. der Ähnlichkeit zweier Beobachtungen wird dabei je Variable $k = 1, .., p$ zunächst in Abhängigkeit vom Skalenniveau ein Distanzmaß $d(x_{ik}, x_{jk})$ berechnet, welches im Intervall [0,1] liegt. Für kategoriale und binäre Variablen beträgt die Distanz dabei z.B. 0, wenn beide Werte identisch sind und 1, falls sich diese unterscheiden. Für metrische Variablen wird der L_1-Abstand zwischen den jeweiligen Ausprägungen berechnet und anschließend durch die Spannweite r_k des Merkmals dividiert $\rightarrow d(x_{ik}, x_{jk}) = \frac{|x_{ik}-x_{jk}|}{r_k}$. Die Gower Distanz berechnet sich anschließend gemäß Formel 2.19 als gewichtetes Mittel der

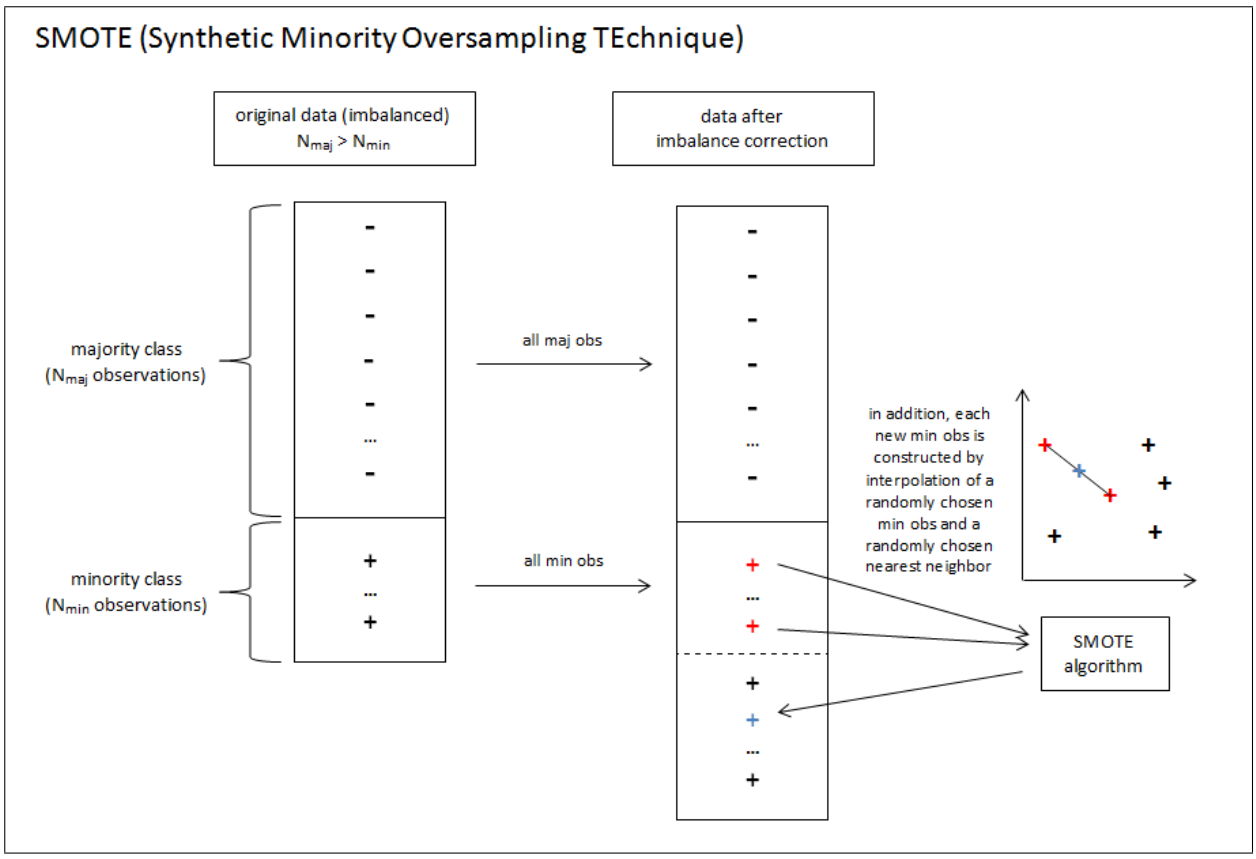

Abbildung 2.6.: Synthetic Minority Oversampling Technique (SMOTE)

Distanzen der einzelnen Variablen. Da diese alle im Intervall [0,1] liegen, ergibt sich auch für die Gower Distanz ein Wertebereich zwischen 0 und 1.

$$d(x_i, x_j) = \frac{\sum_{k=1}^{p} w_k d(x_{ik}, x_{jk})}{\sum_{k=1}^{p} w_k} \in [0,1], \quad \sum_{k=1}^{p} w_k = 1, \quad i,j = 1,..,n \tag{2.19}$$

Nach Berechnung der n nächsten Nachbarn je Beobachtung, wird anschließend mehrfach (je nach Oversampling-Rate) zufällig eine Beobachtung der kleinen Klasse sowie ein zufälliger, nächster Nachbar selektiert und daraus eine neue, „synthetische“ Beobachtung erzeugt. Dabei werden die Werte der metrischen Variablen der neuen Beobachtung durch zufällige Konvexkombination der Ausprägungen der Ausgangs-Beobachtungen berechnet ($x_{new} = \lambda \cdot x_{old,1} + (1-\lambda) x_{old,2}, \ \lambda \sim U(0,1)$), für kategoriale Variablen wird die neue Ausprägung per Zufallswahl (Bernoulli-Experiment mit p = 0.5) aus den beiden Ausgangs-Beobachtungen abgeleitet.

Da die zugrunde liegenden Daten auch bei Anwendung des SMOTE-Verfahrens künstlich vergrößert werden, kommt es i.d.R. wie beim Oversampling zu einer Erhöhung der Lauf-

zeit bei Anwendung des Lernverfahrens. Im Extremfall (→ Oversampling-Rate entspricht IR und sehr starke Unbalanciertheit) kann die Anzahl der Beobachtungen nahezu verdoppelt werden (vgl. Abschnitt 2.2.2.2).
Als mögliche Erweiterung lässt sich SMOTE mit Random Undersampling zu einem hybriden Verfahren kombinieren [8]. Neben der Erzeugung neuer, künstlicher Beobachtungen, werden dabei zusätzlich zufällig Beobachtungen der großen Klasse verworfen. Modified SMOTE [13] sowie Borderline-SMOTE [15] stellen weitere Optimierungen des SMOTE-Verfahrens dar. Hierbei wird bei der Generierung der neuen Beobachtungen zusätzlich die Verteilung bzw. die Lage der Beobachtungen in der kleinen Klasse berücksichtigt. Entweder werden nur bestimmte Beobachtungen zur Generierung neuer Beobachtungen in Betracht gezogen oder die Selektion der nächsten Nachbarn ist abhängig von der Lage der jeweils ausgewählten Beobachtung. Eine kurze Beschreibung des Ensemble-basierten SMOTEBagging als weitere ALternative findet sich im folgenden Abschnitt 2.2.2.4.
Im Rahmen dieser Arbeit wird SMOTE als reines Oversampling-Verfahren betrachtet, um das Verfahren insbesondere mit dem Oversampling vergleichen zu können. Die genannten Erweiterungen – SMOTE inklusive Undersampling sowie Modified- und Borderline-SMOTE wurden in die durchgeführten Experimenten soit nicht berücksichtigt.

2.2.2.4. Overbagging

Oversampling kann durch Anwendung der Bagging-Methode zum so genannten Overbagging erweitert werden. Das Oversampling der kleinen Klasse sowie die anschließende Modellanpassung durch das Lernverfahren werden dabei mehrfach durchgeführt. Die Klassifizierung neuer Beobachtungen erfolgt per Mehrheitswahl (majority vote) der Ergebnisse der einzelnen Modelle, d.h. die Klasse, welche am häufigsten vorhergesagt wurde, wird auch final prognostiziert.

Für jedes Modell wird zunächst die kleine Klasse (minority class) per Oversampling vergrößert. Im Unterschied zum klassischen Oversampling 2.2.2.2 fließen jedoch nicht alle Beobachtungen der kleinen Klasse in jedes Modell ein. Stattdessen wird nach Oversampling der kleinen Klasse wiederum je Modell eine Bootstrap-Stichprobe erzeugt (vgl. Vorgehensweise Random Forest in Abschnitt 2.1.3). Um die Variabilität zwischen den Modellen zusätzlich zu erhöhen, werden auch die Beobachtungen der großen Klasse per Bootstrap gezogen. Die Anzahl der Beobachtungen der großen Klasse bleibt dabei unverändert. Bei Verwendung des Imbalance Ratio (IR) als Oversampling-Rate steuert jede Klasse somit beispielhaft N_{maj} Beobachtungen je Iteration bei.

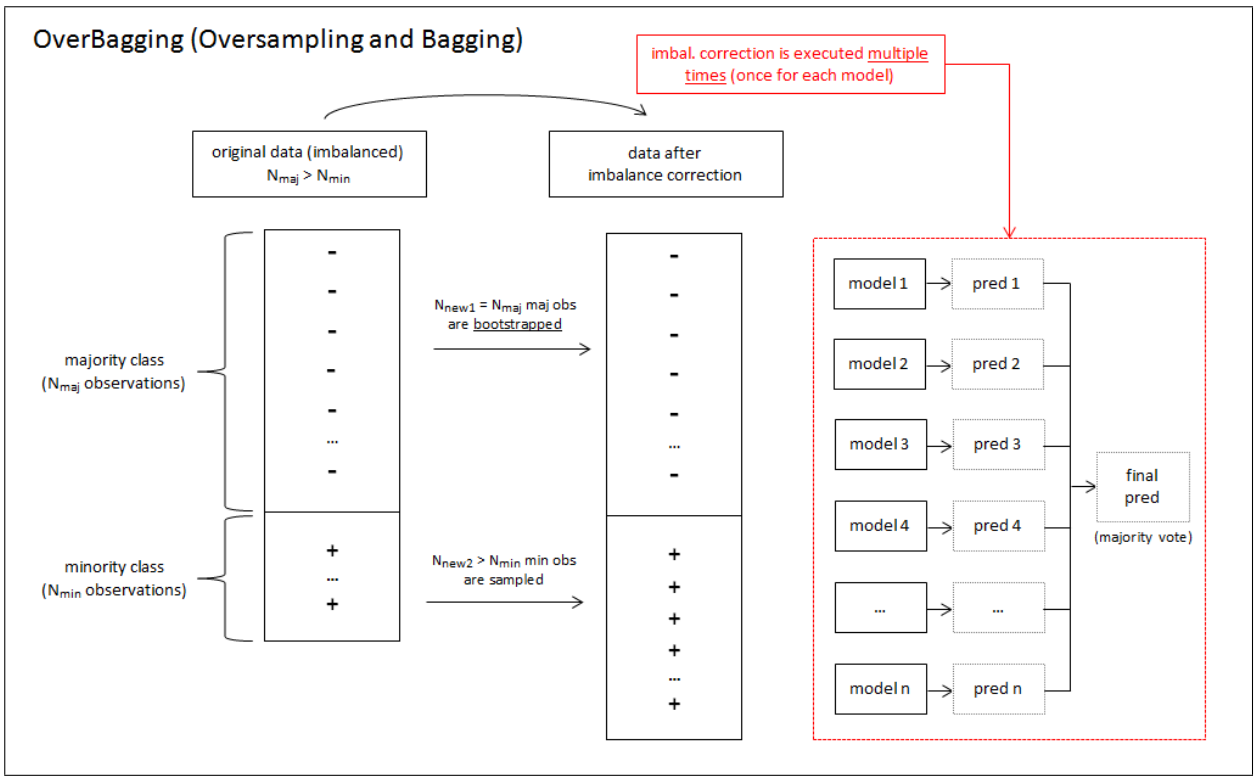

Abbildung 2.7.: Overbagging

Da auch beim Overbagging die Anzahl der Beobachtungen vergrößert wird, wirkt sich dies i.d.R. negativ auf die Laufzeit des jeweiligen Lernverfahrens aus. Des Weiteren hängt die Laufzeit insbesondere auch von der Anzahl der Overbagging-Iterationen ab, welche wie bei anderen Bagging-Verfahren einen frei wählbaren Parameter darstellt.
Entsprechend der Verknüpfung von Bagging und Oversampling existiert mit SMOTE-Bagging auch eine weitere Variante des in Abschnitt 2.2.2.3 beschriebenen SMOTE-Verfahrens. Dieses unterscheidet sich im Vergleich zu Overbagging zum einen durch die alternative Erzeugung der neuen Beobachtungen. Da auch beim SMOTEBagging die Beobachtungen der kleinen Klasse je Iteration durch Ziehen mit Zurücklegen bestimmt werden, kann des Weiteren der Anteil der (wahren) Ursprungs-Beobachtungen sowie der Anteil der per SMOTE (künstlich) erzeugten Beobachtungen variiert werden. Diese so genannte SMOTE Resampling-Rate $\alpha \in (0, 1]$ kann dabei z.B. von Modell zu Modell variiert werden, so dass sich die Daten für die einzelnen Modellen möglichst stark unterscheiden [13]. Die Beobachtungen der großen Klasse werden wie auch beim Overbagging per Bootstrap gezogen, um die Variabilität zwischen den einzelnen Modellen zusätzlich zu erhöhen.

2.2.3. Kostenbasierte Methoden

2.2.3.1. (Class) Weighting

Als einzige kostenbasierte Korrekturmethode wird im Rahmen dieser Arbeit das so genannte Weighting bzw. Class Weighting [23] betrachtet. Dieses Verfahren stellt eine der wenigen kostenbasierten und zugleich verfahrensunabhängigen Korrekturmethoden dar. Anstelle einer Kostenmatrix mit unterschiedlichen Fehlklassifikationskosten C_{10}, C_{01} werden diese allen Beobachtungen der jeweiligen Klasse direkt als Gewicht zugeordnet und anschließend eine gewichtete Klassifikation durchgeführt. Das vergebene Gewicht je Beobachtung (case weight) w_i richtet sich demnach ausschließlich nach der Ausprägung der Klasse sowie dem entsprechenden Klassengewicht (class weight) w_{min} bzw. w_{maj}. Das verwendete Lern- bzw. Klassifikationsverfahren muss somit lediglich Gewichte verarbeiten können, um zu einem kostensensitiven Verfahren zu werden. Da die meisten Klassifikationsverfahren in der Lage sind mit case weights umzugehen, lässt sich das Weighting genau wie die beschriebenen Sampling-Verfahren mit einer Vielzahl an Lernverfahren kombinieren.

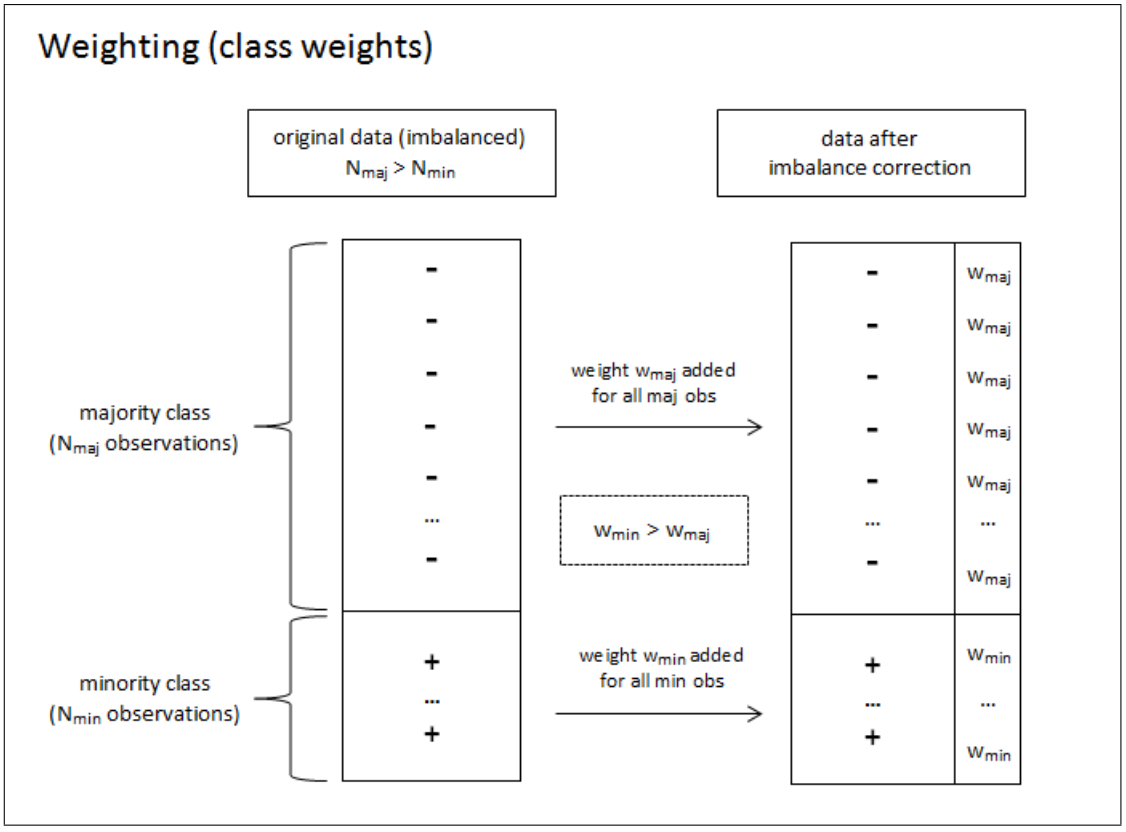

Abbildung 2.8.: Weighting

Die in Abbildung 2.8 dargestellten Gewichte w_{maj} und w_{min} entsprechen grundsätzlich den Fehlklassifikationskosten C_{10} sowie C_{01} (vgl. Abschnitt 2.2.1). Sind die wahren Kosten unbekannt, kann anstelle der tatsächlichen Kosten auch das Kostenverhältnis ($\rightarrow$ durch Division mit den i.d.R. geringeren Kosten für falsch klassifizierte negative Beobachtungen, d.h. $w_{maj} = \frac{C_{01}}{C_{01}} = 1$ und $w_{min} = \frac{C_{10}}{C_{01}}$) übergeben oder die Gewichte alternativ mittels Tuning-Verfahren bestimmt werden.
Im Unterschied zu den vorgestellten Sampling-Methoden werden die Beobachtungen der Klassen und dadurch die Größe des Datensatzes beim Weighting nicht verändert. Des Weiteren gehen alle verfügbaren Beobachtungen auch in das Modell ein, so dass kein Informationsverlust entsteht.

2.3. Gütemaße für die Performance

2.3.1. Kennzahlen der Konfusionsmatrix

Die Wahl des Gütemaßes zur Beurteilung eines Verfahrens ist von entscheidender Bedeutung, da die Bewertung je nach betrachteter Größe stark variieren kann. Im Zwei-Klassen-Fall eignet sich i.d.R. zunächst die Betrachtung der Konfusionsmatrix (vgl. Tabelle 2.2), welche sozusagen die Basis für die Analyse der Performance des jeweiligen Verfahrens darstellt. Zur Erstellung der Konfusionsmatrix wird zunächst für jede Beobachtung eines festgelegten Testdatensatzes die Klassenzugehörigkeit prognostiziert und daraufhin mit den wahren Klassen der jeweiligen Beobachtungen verglichen. Bei der Vorhersage kann dabei entweder die Klasse direkt geschätzt oder alternativ zunächst die Klassenwahrscheinlichkeiten bestimmt und die entsprechende Klasse anschließend über einen festgelegten Schwellenwertes (Threshold) c abgeleitet werden. Üblicherweise wird der dabei der Threshold $c = 0.5$ verwendet, d.h. ist die vorhergesagte Wahrscheinlichkeit einer Beobachtung für z.B. die positive Klasse größer als 0.5 ($\mathbb{P}(y_i = 1) > c = 0.5$), so wird für diese Beobachtung die positive Klasse bzw. Klasse 1 vorhergesagt. Prinzipiell ist der Threshold im Intervall [0,1] frei wählbar. Je kleiner c dabei gewählt wird ($c \rightarrow 0$), desto höher ist die Anzahl der anhand der Klassenwahrscheinlichkeiten als positiv klassifizierten Beobachtungen. Für den Fall $c = 0$ werden alle Beobachtungen der positiven Klasse zugeordnet, für $c = 1$ der negativen Klasse.

Die Konfusionsmatrix beschreibt grundsätzlich, wie gut ein Verfahren die einzelnen Klassen prognostiziert. Im Falle der binären Klassifikation ergibt sich somit die in Tabelle 2.2 darstgestellte 2×2-Matrix. Die Anzahlen der je Klasse korrekt klassifizierten Beobachtun-

	positive prediction	negative prediction
positive class	true positives (TP)	false negatives (FN)
negative class	false positives (FP)	true negatives (TN)

Tabelle 2.2.: Konfusionsmatrix

gen sind dabei auf der Hauptdiagonalen dargestellt (TP und TN). Die „True Positives“ ergeben sich dabei aus den Beobachtungen der positiven Klasse, die korrekt als positiv klassifiziert wurden. „True Negatives“ bezeichnen entsprechend die korrekt klassifizierte Beobachtungen der negativen Klasse. Bei den „False Positives“ handelt es sich hingegen um die fehlerhaft als positiv klassifizierten Beobachtungen, d.h. die Beobachtungen, die eigentlich zur negativen Klasse zählen, aber als positiv klassifiziert wurden. „False Negatives“ stellen die fehlerhaft als negativ klassifizierten Beobachtungen dar.
Aus der Konfusionsmatrix lässt sich zunächst die Gesamtgenauigkeit (Accuracy) bzw. Fehlklassifikationsrate ϵ (1-Accuracy) ableiten. Dabei handelt es sich um den Anteil der korrekt bzw. der fehlerhaft prognostizierten Beobachtungen an der Gesamtzahl der Beobachtungen in den zugrunde liegenden (Test-)Daten. Die Berechnung der Fehlklassifikationsrate ist besipielhaft in Formel 2.20 dargestellt.

$$\epsilon = 1 - \text{Acc} = 1 - \frac{TP + TN}{TP + FP + FN + TN} = \frac{FP + FN}{TP + FP + FN + TN} \tag{2.20}$$

Da bei der Fehlklassifikationsrate jedoch nicht mehr zwischen den Fehlklassifikationen der einzelnen Klassen (FN und FP) unterschieden wird, sind sowohl diese als auch die die Accuracy als Gütemaße insbesondere für stark unbalancierte Daten eher ungeeignet. So würde z.B. bei Daten mit einem 98%-igen Anteil der Beobachtungen der negativen Klasse bereits eine Genauigkeit von 98% durch ausschließliche Vorhersage der negativen Klasse erreicht. D.h. trotz sehr hoher Gesamtgenauigkeit, ist die Genauigkeit bezüglich der kleinen Klasse in diesem Beispiel gleich 0, d.h. keine positive Beobachtung wird korrekt erkannt. Neben Accuracy und Fehlklassifikationsrate lassen sich daher weitere Werte aus der Konfusionsmatrix ableiten, die als Grundlage für die im Rahmen dieser Arbeit betrachteten Maße - F1-Score (vgl. Abschnitt 2.3.2) und AUC (vgl. Abschnitt 2.3.3) dienen:

- True-Positive-Rate ($TPR = \frac{TP}{TP+FN}$)
  Anteil der korrekt als positiv klassifizierten Beobachtungen unter allen tatsächlich positiven Beobachtungen. Die True-Positive-Rate wird auch als Sensitivität, Trefferquote oder Recall bezeichnet.

- False-Negative-Rate ($FNR = 1 - TPR$)

Anteil der fälschlich als negativ klassifizierten Beobachtungen unter allen tatsächlich positiven Beobachtungen.

- True-Negative-Rate ($TNR = \frac{TN}{TN+FP}$)
  Anteil der korrekt als negativ klassifizierten Beobachtungen unter allen tatsächlich negativen Beobachtungen. Die True-Negative-Rate wird auch als Spezifität bezeichnet.

- False-Positive-Rate ($FPR = 1 - TNR$)
  Anteil der fälschlich als positiv klassifizierten Beobachtungen unter allen tatsächlich negativen Beobachtungen.

- Positive-Predictive-Value ($PPV = \frac{TP}{TP+FP}$)
  Anteil der korrekt als positiv klassifizierten Beobachtungen unter allen als positiv klassifizierten Beobachtungen. Der Positive-Predictive-Value wird auch als Precision oder Genauigkeit bezeichnet. Um eine Verwechselung mit der in Formel 2.20 enthaltenen (Gesamt-)Genauigkeit (Accuracy) zu vermeiden, wird der Positive-Predictive-Value fortan als Precision bezeichnet.

2.3.2. F1-Score

Für den so genannte F-Score (oder F-Maß) werden True-Positive-Rate ($\rightarrow$ Recall) und Precision in Form eines gewichteten Mittels verknüpft:

$$F_\beta = (1 + \beta) \cdot \frac{precision \cdot recall}{\beta \cdot precision + recall} \tag{2.21}$$

Der Wertebereich des in Formel 2.21 dargestellten F-Scores liegt dabei im Intervall [0,1], wobei 0 dem schlechtesten und 1 dem besten Wert ($\rightarrow$ Recall = 1 und Precision = 1) entspricht. Die Gewichtung der beiden Größen erfolgt über den Parameter β. Für $\beta = 1$ ergibt sich dabei das balancierte F-Maß oder auch F1-Maß / F1-Score als harmonisches Mittel, bei dem Recall und Precision gleich gewichtet sind. Für $\beta > 1$ lässt sich der Recall, für $\beta < 1$ die Precision entsprechend höher gewichten.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{TP}{\frac{(TP+FP)+(TP+FN)}{2}} \tag{2.22}$$

Die alternative Schreibweise in Formel 2.22 verdeutlicht, dass das F1-Score die True Po-

sitives sozusagen auf das arithmetische Mittel aus allen positiv vorhergesagten Beobachtungen und den wahren positiven Beobachtungen bezieht. Die True Negatives fließen hingegen nicht in den F1-Score ein bzw. werden nur indirekt über die False Positives berücksichtigt. Bei Betrachtung des F1-Score wird somit der Fokus maßgeblich auf die Prognosegüte bezüglich der positiven, d.h. i.d.R. der kleinen Klasse gelegt und die Performance bezüglich der großen Klasse eher vernachlässigt.

2.3.3. ROC-Kurve und AUC

Die Fläche unter der Receiver-Operater-Characteristic(ROC)-Kurve, die „Area under Curve“ (AUC), stellt eines der am häufigsten verwendeteten Gütemaße insbesondere im Rahmen der binären Klassifikation dar. Mittels der ROC-Kurve werden hierbei zunächst True-Positive-Rate (TPR) und False-Positive-Rate (FPR) (vgl. Abschnitt 2.3.1) gegenübergestellt. Die True-Positive-Rate wird dabei auf der Ordinate, die False-Positive-Rate auf der Abszisse abgetragen. Aus den Elementen der Konfusionsmatrix lässt sich somit exakt ein Punkt mit Koordinaten $(x, y) = (FPR, TPR)$ berechnen. Wie bereits in Abschnitt 2.3.1 beschrieben, ist bei Prognose der Klassenwahrscheinlichkeiten durch Anpassung des Thresholds $c \in [0, 1]$ prinzipiell die Berechnung mehrerer alternativer Konfusionsmatrizen und damit auch verschiedener Koordinaten für True-Positive- und False-Positive-Rates möglich. Der optimale Punkt für True-Positive- und False-Positive-Rate liegt bei (0,1), da hier alle Beobachtungen korrekt klassifiziert wurden. Die Punkte (0,0) bzw. (1,1) entsprechen den Thresholds $c = 1$ bzw. $c = 0$, bei denen jeweils ausschließlich eine der beiden Klassen für alle Beobachtungen vorhergesagt wird. Die ROC-Kurve lässt sich nun ableiten, indem die durch ein Verfahren bzw. Modell vorhergesagten Klassenwahrscheinlichkeiten für eine Folge von Thresholds betrachtet und die entsprechenden True-Positive- und False-Positive-Rates berechnet und in Form einer Kurve dargestellt werden (vgl. Abbildung 2.9).

Verläuft die ROC-Kurve entlang oder in der Nähe der Diagonalen, so ist das zu evaluierende Verfahren in etwa so „gut“ wie ein Zufallsprozess und damit gewissermaßen wertlos. Eine nahezu optimale ROC-Kurve steigt zunächst vom Punkt (0,0) sehr stark senkrecht an, d.h. die True-Positive-Rate liegt möglichst nahe bei 1 bei verhältnismäßig kleiner False-Positive-Rate. Erst danach erhöht sich langsam die False-Positive-Rate bis die Kurve den Punkt(1,1) erreicht.
Die Fläche unterhalb der ROC-Kurve, die „Area under Curve“ (AUC), fasst die gesamte Kurve sowie deren Verlauf in einer einzigen Kennzahl zusammen. Der AUC-Wert beschreibt damit allgemein, wie gut ein Modell im Mittel, d.h. unabhängig vom Threshold

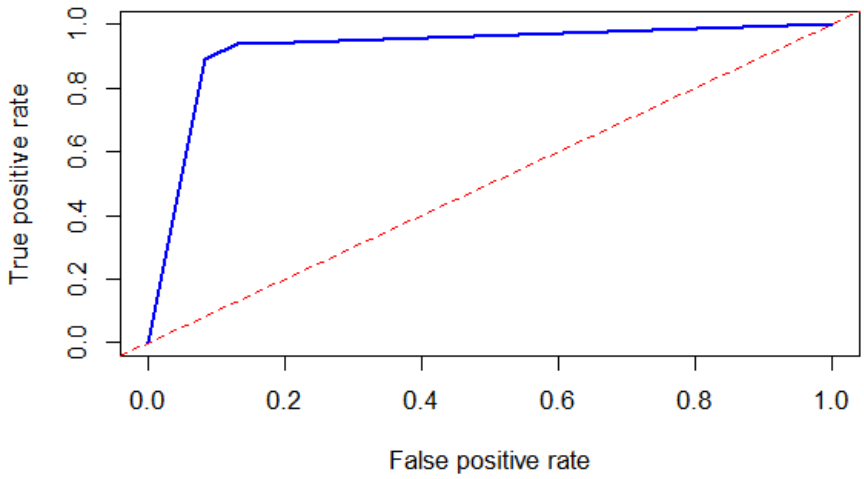

Abbildung 2.9.: ROC Kurve

c, sowohl Beobachtungen der positiven als auch der negativne Klasse korrekt klassifiziert. Der AUC berechnet sich als Fläche zwischen der ROC-Kurve und der Abszisse.

$$AUC = \frac{1 + TPR - FPR}{2} \quad \in [0, 1] \tag{2.23}$$

Obwohl der Wertebereich des AUC-Wertes prinzipiell im Intervall $[0, 1]$ liegt, stellt ein Wert von 0.5 das schlechteste Ergebnis dar, da die entsprechende ROC-Kurve in diesem Fall wiederum entlang bzw. in der Nähe der Diagonalen verläuft. Für AUC-Werte kleiner 0.5 kann die Vorhersage des entsprechenden Modells im Zwei-Klassen-Fall durch Negation der Vorhersagen in einen Wert größer 0.5 umgewandelt werden. Bei optimalem Verlauf der ROC-Kurve erreicht der AUC den Wert 1.
Anschaulich bedeutet ein hoher AUC, dass für eine geeignete Wahl des Thresholds ein gutes Ergebnis mit dem jeweiligen Modell erreicht werden kann. Der Vorteil des AUC-Wertes gegenüber der Fehlklassifikationsrate besteht somit darin, dass zum einen durch Betrachtung der True-Positive-Rate und der False-Positive-Rate die Prognosegüte bezüglich beider Klassen betrachtet wird. Aufgrund der Unabhängigkeit vom Threshold c ist des Weiteren eine allgemeinere Aussage zur Performance des Modells möglich. Die Fehlklassifikationsrate wie auch der F1-Score (vgl. Abschnitt 2.3.2) werden jeweils nur anhand eines konkreten Thresholds (i.d.R. $c = 0.5$) berechnet. Aufgrund der Thrshold-übergreifenden Bewertung der Perfomance eignet sich der AUC-Wert dadurch insbesondere für einen generelle Vergleiche zwischen zwei oder mehreren, ggf. sehr unterschiedlichen Modellen ($\rightarrow$ verschiedene Modellparameter, verschiedene Lernverfahren etc.). Hierbei gilt jedoch zu

beachten, dass wiederum für einen konkreten Threshold ein Modell mit eigentlich kleinerem AUC-Wert besser geeignet ist als ein Modell mit höherem AUC, da keine der beiden entsprechenden ROC-Kurven gleichmäßig besser als die andere. Da Die ROC-Kurve durch den AUC in einem Wert zusammengefasst wird und damit Informationen verloren gehen, bei Interesse eines bestimmten Thresholds bzw. eines bestimmten Bereichs die Betrachtung des AUC ggf. eher ungeeignet und es empfiehlt sich stattdessen z.B. die Analyse des Partial AUC [22].

3. Experimente

3.1. Daten

Zur Analyse und zum Vergleich der Performance der verschiedenen Lern- und Korrekturverfahren werden im Rahmen dieser Arbeit 23 Datensätze (vgl. Tabelle 3.1) betrachtet, welche über die Machine Learning Plattform Open ML (`http://www.openml.org/`) [24] zugänglich und abrufbar sind. Da es sich zum Großteil ursprünglich um Multi-Klassen-Probleme handelt, wurde jeweils eine der vorhandenen Klassen als „kleine Klasse“ ausgewählt und die anderen zu einer großen Klasse zusammengefasst. Die Reduktion auf die resultierenden Zwei-Klassen-Probleme lässt sich als One-vs-All bzw. One-vs-Rest Fragestellung ansehen, d.h. es soll prognostiziert wird, ob eine Beobachtung zu einer entsprechenden Klasse gehört oder nicht. Eine Übersicht der festgelegten kleinen Klassen je Datensatz findet sich in ANHANG A.

In Tabelle 3.1 sind die maßgeblichen Eigenschaften der untersuchten Datensätze dargestellt. Diese sind dabei aufsteigend nach dem Imbalance Ratio (IR) (Spalte *ir*), d.h. dem Quotient aus der Anzahl der Beobachtungen der großen Klasse und der Anzahl der Beobachtungen der kleinen Klasse sortiert. Die Anzahl der Beobachtungen insgesamt (Spalte *n*) schwankt zwischen 336 (*ecoli4*) und 28.056 (*kropt16*), der Anteil der kleinen Klasse (Spalte *pct.min*) an allen Beobachtungen zwischen 0.8% (*abalone19*) und 15% (*scenesunset*), was einem IR von 5.6 bis 129.5 entspricht.
In den Datensätzen sind sowohl metrische als auch kategoriale Features enthalten (Spalten *num*, *cat*). Konstante Features, d.h. Variablen mit nur einer einzigen Ausprägung wurden vorab aus dem jeweiligen Datensatz entfernt (Spalte *rem*). Die resultierende Anzahl der betrachten Features (Spalte *feat*) reicht schließlich von 4 (*balance2*) bis 294 (*scenesunset*).
Bei der Auswahl der Datensätze und Zusammenfassung der Klassen wurde vornehmlich auf den Grad der Unbalanciertheit ($\rightarrow$ IR > 5) geachtet. Andere Aspekte wie die Trennbarkeit der Klassen oder die Verteilung der Beobachtungen innerhalb der Klassen wurden bei der Selektion nicht betrachtet.

dsname	n	n.min	n.max	pct.min	ir	num	cat	rem	feat
scenesunset	2407	364	2043	0.15	5.61	294	0	0	294
ecoli4	336	35	301	0.10	8.60	7	0	0	7
optdigits0	5620	554	5066	0.10	9.14	64	0	2	62
satelliteimage4	6435	626	5809	0.10	9.28	36	0	0	36
pendigits9	10992	1055	9937	0.10	9.42	16	0	0	16
vowel1	990	90	900	0.09	10.00	10	2	0	12
spectrometer42	531	45	486	0.08	10.80	100	1	1	100
balance2	625	49	576	0.08	11.76	4	0	0	4
anneal5	898	67	831	0.07	12.40	6	32	7	31
coil2000	9822	586	9236	0.06	15.76	85	0	0	85
arrhythmia6	452	25	427	0.06	17.08	206	73	17	262
oilspill	937	41	896	0.04	21.85	49	0	1	48
solarflare5	1066	43	1023	0.04	23.79	0	12	1	11
car4	1728	65	1663	0.04	25.58	0	6	0	6
letter26	20000	734	19266	0.04	26.25	16	0	0	16
yeast5	1484	51	1433	0.03	28.10	8	0	0	8
winequality4	6497	216	6281	0.03	29.08	11	0	0	11
ozonelevel	2536	73	2463	0.03	33.74	72	0	0	72
nursery3	12960	328	12632	0.03	38.51	0	8	0	8
mammography	11183	260	10923	0.02	42.01	6	0	0	6
pageblocks5	5473	115	5358	0.02	46.59	10	0	0	10
kropt16	28056	390	27666	0.01	70.94	0	6	0	6
abalone19	4177	32	4145	0.01	129.53	7	1	0	8

Tabelle 3.1.: Übersicht Datensätze

3.2. Verfahren und Parameter

In den durchgeführten Experimenten kommen die in Abschnitt 2.1 beschriebenen Klassifikationsverfahren – Logistische Regression (logreg), Klassifikationsbaum (cart), Random Forest (rf), Gradient Tree Boosting (gbm) und Support Vector Machine (svm) – zum Einsatz. Als Methoden zur Korrektur des Klassenungleichgewichts werden des Weiteren Random Undersampling (us), Oversampling (os), Oversampling anhand SMOTE (sm), Overbagging (ob) und Class Weighting (cw) untersucht.
Die Klassifikationsverfahren wurden dabei sowohl mit den jeweiligen Standardeinstellungen als auch mit vorherigem Tuning bestimmter Parameter betrachtet. Je Lernverfahren wurden dabei die in Tabelle 3.2 aufgelisteten Parameter mit jeweiligem Optimierungsbereich (Spalte *tuning*) berücksichtigt. In der Spalte *pkg* ist zusätzlich das entsprechend verwendete R-Paket zum jeweiligen Lernverfahren angegeben.

Für die betrachteten Sampling-Methoden Undersampling (us), Oversampling (os), SMOTE (sm) und Overbagging (ob) wird jeweils zusätzlich die Sampling-Rate per Tuning-

lrn	pkg	tuning
cart	rpart	cp (0.0001, 0.1) / minsplit (1, 50)
gbm	gbm	n.trees (100, 5000) / interaction.depth (1, 3) / shrinkage (1e-05, 0.1) / bag.fraction (0.7, 1)
logreg	stats	-
rf	randomForest	ntree (10, 500) / mtry (1, 10)
svm	kernlab	C (2^{-12}, 2^{12}) / sigma (2^{-12}, 2^{12})

Tabelle 3.2.: Übersicht der Lernverfahren sowie der Tuning-Parameter und -Bereiche

Verfahren bestimmt. Für das Weighting wird das Klassengewicht der kleinen Klasse per Tuning bestimmt, das Klassengewicht der großen Klasse wird jeweils fest auf 1 gesetzt. Als Grenzen für die Oversampling-Raten (os, sm, ob) sowie für das Klassengewicht der kleinen Klasse (cw) wurde datensatzabhängig das Intervall $[1, \mathrm{IR}]$ (vgl. Tabelle 3.1) festgelegt. Bei Auswahl von IR als obere Grenze wird die Anzahl der Beobachtungen der kleinen Klasse auf die Anzahl der Beobachtungen der großen Klasse erhöht, so dass genau gleich viele Beobachtungen je Klasse im Datensatz enthalten sind. Entsprechendes gilt für die Undersampling-Rate, deren Optimierungsbereich im Intervall $[\mathrm{IR}^{-1}, 1]$ liegt. Hierbei wird die Fallzahl der großen Klasse bei Auswahl von IR^{-1} auf die Fallzahl der kleinen Klasse reduziert.
Als ein weiterer, konstanter Parameter wurden 10 Iterationen für das Overbagging festgelegt. Für das SMOTE-Verfahren wird neben der Oversampling-Rate zusätzlich ein logischer Parameter (TRUE, FALSE) zur Ablaufsteuerung des Algorithmus über das Tuning bestimmt. Dieser Parameter steuert, ob für jede neu zu erzeugende Beobachtung sowohl die ursprüngliche Beobachtung als auch der entsprechende nächste Nachbar per Zufall bestimmt werden (FALSE) oder ob alternativ jede Beobachtung der kleinen Klasse in Abhängigkeit von der Sampling-Rate mehrfach für die Erzeugung einer neuen Beobachtung herangezogen und nur der nächste Nachbar per Zufall bestimmt wird. Die Anzahl der betrachteten nächsten Nachbarn ist mit $k = 5$ fest angegeben und wird nicht innerhalb des Tuning berücksichtigt.

method	additional tuning parameters
undersampling	usw.rate (IR^{-1},1)
oversampling	osw.rate (1,IR)
smote	sw.rate (1,IR) / sw.alt.logic (TRUE,FALSE)
overbagging	obw.rate (1,IR)
weighting	wcw.weight (1,IR)

Tabelle 3.3.: Übersicht der Korrekturverfahren sowie der Tuning-Parameter im Paket mlr [1] mit Optimierungsbereich

3.3. Parameter-Tuning

Das Tuning der in Abschnitt 3.2 aufgelisteten Parameter wird mittels Iterated F-Racing [**?**] (R-Paket irace) durchgeführt. Bei diesem Tuning-Verfahren werden iterativ mehrere, zufällig gezogene Konfigurationen, d.h. Kombinationen von Parametereinstellungen, miteinander verglichen. Je Durchgang wird je betrachteter Konfigurationen ein Modell angepasst und eine Evaluierung auf einem Testdatensatz vorgenommen. Konfigurationen, die hinsichtlich ihrer Performance deutlich unterlegen sind ($\rightarrow$ Bestimmung z.B. anhand des nichtparametrischen Friedman-Tests) werden aus dem Pool an Konfiguration entfernt und an deren Stelle neue, wiederum zufällig gezogene Konfigurationen hinzugefügt. Die nach jedem Durchgang im Pool verbleibenden Konfigurationen werden als so genannte Elite-Konfigurationen bezeichnet.
Die Auswahl der betrachteten Konfigurationen erfolgt in der ersten Iteration je Parameter über dessen gesamten Tuning-Bereich, d.h. die Konfigurationen werden initial aus einer Gleichverteilung über den kompletten Parameterraum gezogen. In den folgenden Iterationen wird diese initiale Verteilung sukzessive angepasst, so dass insbesondere die Umgebungen der Elite-Konfigurationen intensiver untersucht werden. Für numerische Parameter werden hierbei trunkierte Normalverteilungen genutzt, welche um die entsprechenden Werte der Elite-Konfigurationen zentriert sind und deren Varianz mit der Anzahl der Iterationen abnimmt. Im Falle kategorialer Parameter wird die Wahrscheinlichkeit je Ausprägung gemäß der Häufigkeiten der Ausprägungen in den Elite-Konfigurationen erhöht bzw. reduziert. Durch dieses Vorgehen wird prinzipiell gewährleistet, dass anfangs zunächst der komplette Parameterraum möglichst gut abgedeckt und in den folgenden Iterationen insbesondere die Umgebung guter Konfigurationen im Detail untersucht wird ($\rightarrow$ exploration and exploitation).
I.d.R wird das Verfahren so lange durchgeführt bis eine vorab festgelegte, maximale Anzahl an (über alle Iterationen übergreifend durchzuführenden) Modellanpassungen und Evaluationen erreicht wird. Sobald dieses Budget ausgeschöpft ist, wird die Konfiguration aus dem aktuellen Pool an Elite-Konfigurationen zurückgegeben, welche den größten Mittelwert aller bisherigen Evaluationen, d.h. der Ergebnisse aus allen Iterationen, in denen die Konfiguration enthalten war, vorweist.
In den durchgeführten Experimenten bestehen die einzelnen Konfigurationen aus den Kombinationen der entsprechenden Parameter der Lernverfahren sowie den ggf. zusätzlichen Parametern der Korrekturmethoden (vgl. Abschnitt 3.2). Für die durchzuführenden Evaluation wird der jeweilige Trainingsdatensatz einmalig in 80% Tuning-Trainingsdaten sowie 20% Tuning-Testdaten aufgeteilt (Holdout). Als Budget werden maximal 300 Evaluationen festgelegt, d.h. über alle Iterationen werden maximal 300 Modelle angepasst und evaluiert. Das Gütemaß innerhalb des Tuning entpricht dem jeweiligen Gütemaß für

die spätere Evaluation des Modells, d.h. es wird entweder der F1-Score oder der AUC verwendet.

3.4. Durchführung und Evaluation

Im Rahmen dieser Arbeit wurde eine Vielzahl an Experimenten durchgeführt, um die betrachteten Klassifikationsverfahren sowohl alleinstehend und mit und ohne vorheriges Parameter-Tuning als auch in Kombination mit den einzelnen Korrekturverfahren zu evaluieren. Um alle Kombinationen aus Lern-, Tuning- und Korrekturverfahren abzudecken wurden daher je Datensatz mehrere Durchläufe absolviert (vgl. Tabelle 3.4). Im ersten Durchlauf wurden zunächst für die in Abschnitt 3.1 aufgelisteten Datensätze nur die Lernverfahren mit ihren jeweiligen Standardeinstellungen verwendet. Für alle weiteren Durchläufe wurden sowohl die relevanten Parameter der Lernverfahren als die auch der Korrekturmethoden (vgl. Abschnitt 3.2) per Tuning bestimmt.

Durchlauf	Daten	Lernverfahren	Tuning	Korrekturmethode
1	alle	alle	nein	-
2	alle*	alle	ja	-
3	alle*	alle	ja	Undersampling
4	alle*	alle	ja	Oversampling
5	alle*	alle	ja	SMOTE
6	alle*	alle	ja	Overbagging
7	alle*	alle	ja	Weighting

*alle Datensätze, deren AUC im 1.Durchlauf immer < 0.99

Tabelle 3.4.: Übersicht der Experimente

Als Gütemaße werden sowohl der in Abschnitt 2.3.2 vorgestellte F1-Score als auch der in Abschnitt 2.3.3 beschriebene AUC-Wert verwendet. Die Datensätze, für welche bereits im ersten Durchlauf mit mindestens einem der Lernverfahren ein AUC-Wert größer 0.99 erreicht werden konnte, wurden in den folgenden Durchläufen nicht mehr betrachtet.
Als Resampling-Strategie wird eine geschachtelte Kreuzvalidierung verwendet, um eine möglichst unverzerrte Schätzung der Gütemaße zu gewährleisten (vgl Abbildung 3.1). In einer äußeren Schleife wird dabei zunächst eine wiederholte, stratifizierte 5-fache Kreuzvalidierung (repeated stratified 5-fold Cross-Validation) durchgeführt. Die Daten werden dabei zufällig in 5 disjunkte, ungefähr gleich große Blöcke (folds) aufgeteilt, von denen jeder der Blöcke einmal als Testdatensatz fungiert, während das zugehörige Modell auf Basis der jeweils vier anderen Blöcke angepasst wird. Dadurch wird gewährleistet, dass

alle Beobachtungen sowohl für die Modellanpassung (Training) als auch für die Vorhersage (Test) berücksichtigt und somit effizient genutzt werden. Aufgrund der Stratifizierung bei der Generierung der Blöcke wird zusätzlich sichergestellt, dass das Klassenverhältnis in den Blöcken ungefähr dem Klassenverhältnis in den gesamten Daten entspricht. Die Anzahl der Blöcke / Folds bei der Kreuzvalidierung kann grundsätzlich frei gewählt werden. Für die durchgeführten Untersuchungen wurde die Anzahl dabei auf fünf Blöcke festgelegt, so dass sich die jeweiligen Trainingsdaten nicht zu ähnlich () sind und somit möglichst unabhängige Modelle entstehen.
Der Tuning-Prozess (vgl. Abschnitt 3.3) ist als "innere Schleife" in die stratifizierte 5-fache Kreuzvalidierung eingebettet. Die jeweiligen Trainigsdaten werden dabei zur Evaluation innerhalb des Tuning nochmals in Tuning-Trainingsdaten (80%) sowie Tuning-Testdaten (20%) aufgeteilt (Holdout). Die Aufteilung wird dabei ebenfalls mittels Stratifizierung vorgenommen, so dass das Klassenverhältnis auch hier ungefähr dem Klassenverhältnis im gesamten Datensatz entspricht.

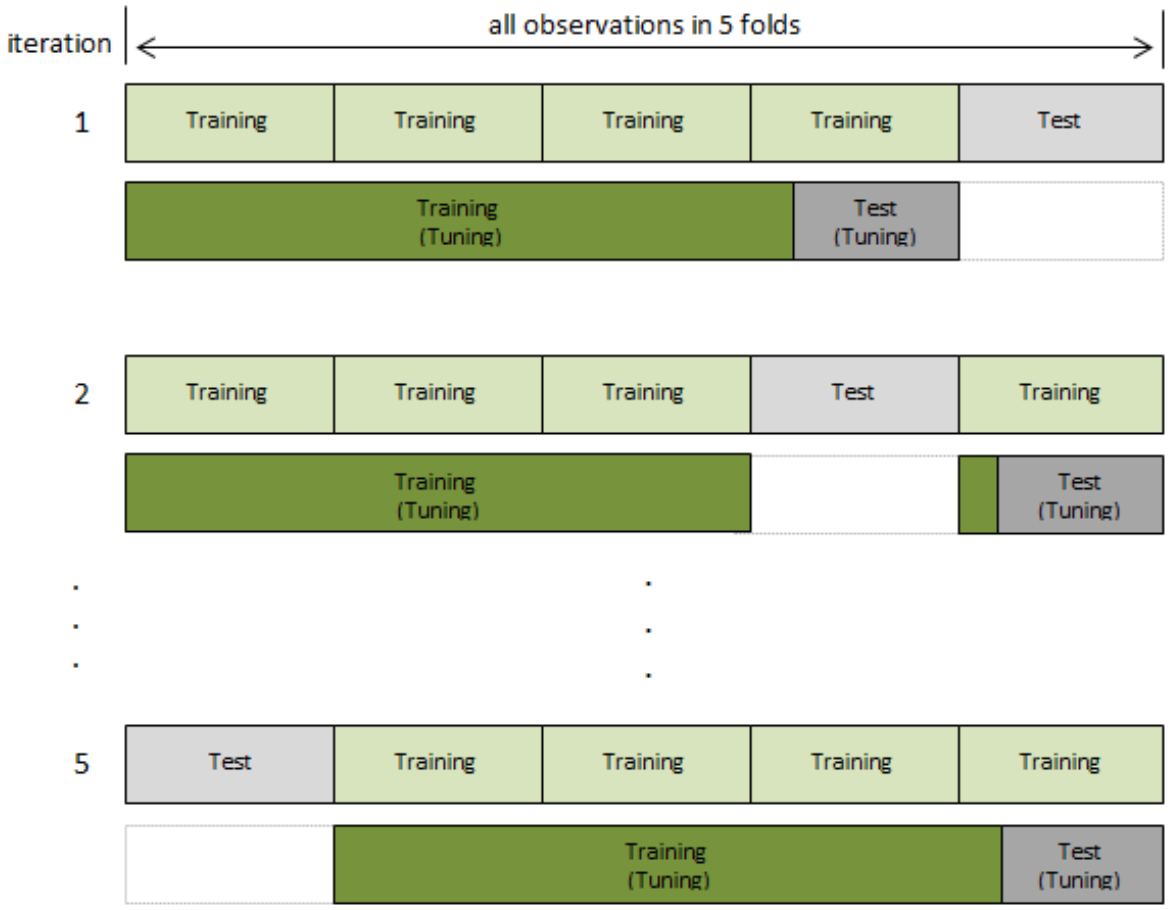

Abbildung 3.1.: Resampling mit stratifizierter 5-facher Kreuzvalidierung und Holdout

Für jeden Block wird im Rahmen der Kreuzvalidierung anhand der Vorhersagen für die einzelnen Beobachtungen der F1-Score bzw. der AUC-Wert berechnet. Als zusammengefasstes Gütemaß für den kompletten Datensatz werden diese fünf berechneten Werte anschließend gemittelt. Um des Weiteren den Einfluss zufälliger Effekte bei der Bildung der Blöcke zu verringern, wird die stratifizierte 5-fache Kreuzvalidierung insgesamt fünfmal wiederholt (repeated cross-validation). Für jede der fünf Wiederholungen wird der Datensatz dabei zufällig in jeweils andere Blöcke unterteilt. Die fünf Ergebnisse aus den Wiederholungen werden abschließend wiederum zu einem Gesamtwert für die (möglichst unverzerrte) Prognosegüte gemittelt.

$$AUC_{mean} = \sum_{i=1}^{5} AUC_{mean,i} = \sum_{i=1}^{5} \sum_{j=1}^{5} AUC_{ij} \tag{3.1}$$

Für das Beispiel des AUC steht in Formel 3.1 der Laufindex i für die Wiederholungen und der Index j für die einzelnen Blöcke innerhalb der Kreuzvalidierung. AUC_{ij} stellt somit den AUC im j-ten Block in der i-ten Wiederholung dar. Um zusätzlich die Unsicherheit der Schätzer zu quantifizieren, werden neben dem Mittelwert entweder der minimale und maximale AUC aus den fünf Wiederholungen der Kreuzvalidierung oder alternativ die 25 einzelnen AUC-Werte aus allen Wiederholungen und Iterationen als Spannweite betrachtet.

Für die Umsetzung der Experimente im Programmpaket R wurden hauptsächlich die Pakete mlr [1], BatchJobs [3] und BatchExperiments [2,3] verwendet. Das R Paket mlr bietet dabei eine Sammlung an Klassifikations- und Regressionsverfahren sowie deren Evaluation und Optimierung mittels verschiedener Resampling-Strategien und Tuning-Verfahren und bildet somit eine zentrale Schnittstelle für eine Vielzahl an gängigen und in R verfügbaren Lernverfahren. Darüber hinaus wurden die im Rahmen dieser Arbeit vorgestellten und betrachteten Korrekturmethoden in R implementiert und in das Paket mlr integriert. Über die Funktionen makeUndersampleWrapper, makeOversampleWrapper, makeSMOTEWrapper, makeOverbaggingWrapper sowie makeUndersampleWrapper können dabei beliebige Lernverfahren mit den jeweiligen Korrekturmethoden kombiniert bzw. erweitert werden. Die aktuelle Version von mlr ist über CRAN (`http://cran.r-project.org/web/packages/mlr/index.html`) verfügbar.
Die Pakete BatchExperiments und BatchJobs bieten in Kombination eine Infrastruktur für die Verwaltung und Durchführung umfassender Experimente. Die betrachteten Probleme (→ Datensätze) und Algorithmen werden dabei zunächst definiert und in einer Registry erfasst. Im Anschluss lassen sich aus Kombination von Problemen und Algorithmen die jeweiligen Experimente bzw. einzelne Jobs generieren. Die Ausführung der

generierten Jobs lässt sich dabei über Funktionen des Pakets BatchJobs parallelisieren. Zur Sicherstellung der Reproduzierbarkeit der Ergebnisse sind innerhalb der Registry verschiedene Seeds gesetzt. Für die im Rahmen dieser Arbeit durchgeführten Experimente wurde zunächst ein übergreifender Seed für die gesamte Registry festgelegt. Des Weiteren wurde für jeden Datensatz ein weiterer „problem seed“ festgelegt, wodurch gewährleistet wird, dass für die verschiedenen Algorithmen (Kombinationen aus Lern-, Tuning- und Korrekturverfahren) die (stochastischen) Rahmenbedingungen prinzipiell gleich sind. Lediglich die Aufteilung der Datensätze in die einzelnen Blöcke bzw. Folds im Rahmen der Kreuzvalidierung ist für jedes einzelne Experiment unterschiedlich.

Diese Arbeit stellt grundsätzlich eine Fortführung des Konferenzberichtes „On Class Imbalancy Correction for Classification Algorithms in Credit Scoring“ [1] dar. Im Vergleich zu den dort durchgeführten Untersuchungen kam es bei den Experimenten im Rahmen dieser Arbeit zu folgenden Änderungen:

- Verwendung neuer Datensätze (vgl. Abschnitt 3.1) über die Machine Learning Plattform Open ML (`http://www.openml.org/`) [24]
- zusätzliche Betrachtung des F1-Score (neben AUC)
- Modifikationen der Korrekturmethoden Oversampling und SMOTE
- Evaluation der Gütemasse anhand wiederholter 5x 5-fach Kreuzvalidierung anstelle „einfacher“ 5-fach Kreuzvalidierung
- Anpassung der Optimierungsbereiche der Sampling-Paramter innerhalb des Tuning ($\rightarrow$ minimale Undersampling-Rate = IR^{-1}, maximale Oversampling-Rate = IR)

4. Diskussion der Ergebnisse

4.1. Datenbasis

Im Rahmen der durchgeführten Experimente wurden in einem ersten Durchlauf für alle in Abschnitt 3.1 dargestellten Datensätze nur die Lernverfahren mit ihren jeweiligen Standardeinstellungen für Modellanpassung und Prognose verwendet. Bei insgesamt 9 der 23 Datensätze konnte hierbei bereits durch mindestens eines der Lernverfahren ein nahezu perfekter AUC-Wert größer 0.99 erreicht werden (vgl. Tabelle 4.1). Diese wurden daher für die folgenden Durchläufe (vgl. Tabelle 3.4) nicht mehr berücksichtigt, wodurch 14 Datensätze in den weiteren untersucht wurden.

Data	Learner	AUC
anneal5	cart	1.000
car4	rf	1.000
kropt16	rf	0.996
letter26	rf	1.000
nursery3	rf	1.000
optdigits0	svm	1.000
pendigits9	rf	0.999
scenesunset	rf	0.995
vowel1	svm	1.000

Tabelle 4.1.: Datensätze und Lernverfahren mit AUC > 0.99

In Summe wurden insgesamt 26750 Experimente / Jobs durchgeführt. Dieser Wert ergibt aus dem Produkt der Anzahl der Datensätze je Durchlauf, den fünf Lernverfahren, zwei Evalutionsmaßen sowie 5 Wiederholungen mit 5 Blöcken innerhalb der Kreuzvalidierungen, welche aufgrund ihrer Unabhängigkeit und zum Zwecke der Paralellisierung als separate Jobs angelegt wurden. Zu insgesamt 702 Experimenten konnten dabei keine Ergebnisse generiert werden:

- in 65 Fällen (ca. 0.24%) traten Fehler innerhalb des Tuning auf
- in 566 Fällen (ca. 2.1%) konnte keine Modellanpassung oder Prognose erfolgen
- weitere 71 Experimente (ca. 0.27%) wurden aufgrund Überschreitung der maximalen Laufzeit von 48 Stunden abgebrochen. Hierbei handelt sich ausschließlich um Experimente, bei den Overbagging als Korrekturverfahren angewendet wurde. Des Weiteren handelt es sich auschließelich um den Datensatz *coil2000* sowie die Lernverfahren Gradient Tree Boosting (25 Experimente) sowie SVM (46 Experimente).

Eine Übersicht der betroffen Datensätze sowie der jeweiligen Verfahren ist in ANHANG B aufgelistet. Die Resultate der verbleibenden Jobs bilden die Grundlage für die in den folgenden Kapiteln dargestellten Ergebnisse und Analysen.

4.2. Ergebnisse je Datensatz

4.2.1. Bestes Verfahren

In Tabelle 4.2 sind zunächst die Verfahren mit den besten Ergebnissen bezüglich des AUC je Datensatz zusammengefasst.

Data	IR	n	Feat	Base	Tuning	Imbal
ecoli4	8.60	336	7	0.939 (rf)	0.944 (svm)	**0.947** (svm, cw)
satelliteimage4	9.28	6435	36	0.962 (rf)	0.965 (svm)	**0.967** (svm, sm)
spectrometer42	10.80	531	100	0.955 (rf)	0.955 (svm)	**0.968** (rf, sm)
balance2	11.76	625	4	0.896 (svm)	0.926 (svm)	**0.946** (svm, sm)
coil2000	15.76	9822	85	0.735 (logreg)	0.755 (gbm)	**0.760** (gbm, cw)
arrhythmia6	17.08	452	262	0.967 (gbm)	0.976 (rf)	**0.979** (rf, sm)
oilspill	21.85	937	48	0.933 (rf)	0.927 (rf)	**0.939** (rf, os)
solarflare5	23.79	1066	11	0.909 (logreg)	0.909 (logreg)	**0.925** (logreg, us)
yeast5	28.10	1484	8	0.926 (rf)	0.924 (rf)	**0.927** (rf, us)
winequality4	29.08	6497	11	0.873 (rf)	0.874 (rf)	**0.878** (rf, sm)
ozonelevel	33.74	2536	72	0.902 (rf)	0.897 (rf)	**0.906** (gbm, cw)
mammography	42.01	11183	6	0.948 (rf)	0.949 (gbm)	**0.956** (gbm, cw)
pageblocks5	46.59	5473	10	0.986 (rf)	0.985 (rf)	**0.989** (rf, cw)
abalone19	129.53	4177	8	0.810 (logreg)	0.810 (logreg)	**0.845** (logreg, os)

Tabelle 4.2.: Übersicht der besten Verfahren je Datensatz (AUC)

Die Datensätze (Spalte *Data*) sind dabei aufsteigend gemäß des Imbalance Ratio (Spalte

IR) sortiert und über die horizontalen Linien in drei Gruppen unterteilt ($\rightarrow$ IR $< 15, \text{IR} <$ 30 und IR > 30). Je größer der IR, desto kleiner entsprechend der Anteil der kleinen Klasse an der Gesamtzahl der Beobachtungen – dabei entspricht IR > 30 ungefähr einem Anteil kleiner 3% und ein IR > 15 einem Anteil kleiner 6% (vgl. Tabelle 3.1). Neben dem IR als Kennzahl für das Ungleichgewicht der Klassen sind des Weiteren die Anzahl Beobachtungen (Spalte *n*) sowie die Anzahl der Merkmale bzw. Features (Spalte *Feat*) je Datensatz aufgelistet. Die drei folgenden Spalten enthalten die AUC-Werte des jeweils besten Verfahrens ($\rightarrow$ in Klammern) für:

- die Lernverfahren mit Standardeinstellungen sowie ohne Tuning und Korrekturmethoden (Spalte *Base*)
- die Lernverfahren inklusive Tuning (Spalte *Tuning*)
- die Lernverfahren inklusive Tuning und in Kombination mit allen Korrekturmethoden (Spalte *Imbal*)

Für die erste Zeile (Datensatz *ecoli4*) bedeutet dies beispielhaft, dass bei Anwendung der Lernverfahren mit Standardeinstellungen der Random Forest mit einem AUC von 0.939 das beste Ergebnis erzielen konnte. Wird zusätzlich bei allen Lernverfahren ein Parameter-Tuning durchgeführt, liefert hingegen die SVM mit einem AUC von 0.944 das beste Ergebnis. Eine weitere, (geringe) Optimierung des AUC-Wertes kann durch Kombination von SVM und Class Weighting erreicht werden (AUC = 0.947). Der beste Wert je Datensatz ist zusätzlich fett markiert.
Die Betrachtung der Spalte *Base* zeigt, dass bereits durch eine geeignete Auswahl des Lernverfahrens hohe AUC-Werte erreicht werden konnten. In den dargestellten Fällen wurden insbesondere durch Anwendung des Random Forest gute Ergebnisse bezüglich des AUC erzielt. Durch zusätzliches Parameter-Tuning konnte der AUC nur für einen Teil der Datensätze verbessert werden. Da sich in bestimmten Fällen (Datensätze *oilspill*, *yeast5*, *ozonelevel*, *pageblocks5*) nach Durchführung des Tuning leicht schlechtere Ergebnisse ergeben als ohne Tuning, ist es ggf. sinnvoll, das Tuning-Budget, d.h. die Anzahl der Experimente innerhalb des des Iterated F-Racing (vgl. Abschnitt 3.3), zu erhöhen. Bezüglich der Ergebnisse in Spalte *Tuning* konnten mit Random Forest und Support Vector Machine, letzteres insbesondere in der Gruppe mit IR < 15), die besten Ergebnisse erzielt werden. Durch die zusätzliche Anwendung der Korrekturmethoden wurde in allen betrachteten Fällen durch mindestens eine der Methoden eine weitere Verbesserung des AUC erreicht. Die besten Ergebnisse konnten dabei insbesondere durch Class Weighting in Kombination mit Gradient Boosting sowie durch SMOTE mit Random Forest oder SVM erreicht werden. Die größte Steigerung des AUC konnte für den stark unbalancier-

ten Datensatz *abalone19* mit einer Erhöhung von 0.035 durch Anwendung der logistischen Regression in Verbindung mit Oversampling gegenüber der logistischen Regression ohne Korrekturmethode erzielt werden.
Die dargestellten Ergebnisse entsprechen grundsätzlich den Resultaten der vorhergehenden Experimente und Analysen [1].
In Tabelle 4.3 sind die entsprechenden Ergebnisse für den F1-Score (und einen festen Threshold von 0.5) dargestellt. Für die Spalte *Base* ergeben sich hierbei zum Teil sehr geringe Werte bzw. in zwei Fällen (Datensätze *balance2* und *abalone19*) ein F1-Score von 0. Dies bedeutet, dass für den Threshold von 0.5 keine True Positives existieren, d.h. hierbei konnte keine Beobachtung der kleinen Klasse korrekt vorhergesagt werden. Durch Anwendung des Tuning konnte im Unterschied zum AUC-Wert in fast allen Fällen

Data	IR	n	Feat	Base	Tuning	Imbal
ecoli4	8.60	336	7	0.616 (svm)	0.640 (gbm)	**0.674** (logreg, ob)
satelliteimage4	9.28	6435	36	0.642 (rf)	0.708 (svm)	**0.727** (svm, ob)
spectrometer42	10.80	531	100	0.614 (svm)	0.590 (svm)	**0.734** (svm, sm)
balance2	11.76	625	4	0.000 (svm)	0.166 (gbm)	**0.661** (svm, sm)
coil2000	15.76	9822	85	0.073 (rf)	0.108 (gbm)	**0.251** (gbm, ob)
arrhythmia6	17.08	452	262	0.685 (cart)	0.764 (rf)	**0.783** (gbm, sm)
oilspill	21.85	937	48	0.539 (svm)	0.525 (svm)	**0.561** (svm, os)
solarflare5	23.79	1066	11	0.189 (logreg)	0.216 (cart)	**0.391** (rf, os)
yeast5	28.10	1484	8	0.339 (cart)	**0.417** (rf)	0.397 (logreg, sm)
winequality4	29.08	6497	11	0.187 (rf)	0.239 (gbm)	**0.305** (rf, sm)
ozonelevel	33.74	2536	72	0.230 (logreg)	0.230 (logreg)	**0.363** (gbm, cw)
mammography	42.01	11183	6	0.674 (rf)	0.698 (gbm)	**0.709** (rf, sm)
pageblocks5	46.59	5473	10	0.723 (rf)	0.723 (rf)	**0.727** (rf, ob)
abalone19	129.53	4177	8	0.000 (logreg)	0.051 (gbm)	**0.070** (rf, sm)

Tabelle 4.3.: Übersicht Gesamtergebnisse F1

eine Verbesserung erzielt werden. In ebenfalls nahezu allen Fällen (Ausnahme: Datensatz *yeast5*) konnte durch Anwendung mindestens eines Korrekturverfahrens eine weitere Verbesserung des F1-Score erreicht werden. Für den Datensatz *balance2* ergibt sich nach Anwendung der Support Vector Machine inklusive Parameter-Tuning und SMOTE-Verfahren eine Verbesserung des F1-Score von ca. 0.5 gegenüber der alleinigen Anwendung der SVM inklusive Tuning. Im Gegensatz dazu konnte für den Datensatz *abalone19* auch durch das beste Verfahren (→ Random Forest und SMOTE)nur maximaler Wert von 0.07 erreicht werden. Der Anteil der True Positives ist diesem Fall somit nach wie vor deutlich geringer als der Anteil der falsch klassifizierten Beobachtungen (False Negatives und False Positives) (vgl. Formel 2.22). Auch für einige weitere Datensätze (*coil2000*, *solarflar5*, *yeast5* und *ozonelevel*) konnten mit den besten Verfahren eher geringe Werte für den F1-Score erzielt werden. In den genannten Fällen empfiehlt sich ggf. eine Herabsetzung des Thresholds.

4.2.2. Alle Verfahren

Für den Vergleich aller Verfahren wurden zur besseren Veranschaulichung und aus Gründen der Übersichtlichkeit beispielhaft drei Datensätze (*abalone19*, *coil2000*, *mammography*) ausgewählt, zu denen insbesondere bereits vergleichbare Ergebnisse aus anderen Arbeiten vorliegen.

Für den Datensatz *abalone19* sind alle zugehörigen Ergebnisse der Experimente in Abbildung 4.1 aufgeführt. Hierbei sind für jedes Lernverfahren in einer separaten Grafik die AUC-Werte bei Verwendung des Lernverfahrens mit Standardeinstellungen (bl), inklusive Tuning (tune) sowie in Verbindung mit den einzelnen Korrekturmethoden (us, os, sm, ob, cw) in Form eines Boxplots dargestellt. Dieser ergibt sich aus den jeweils 25 Werten der 5x 5-fachen Kreuzvalidierung (vgl. Abschnitt 3.4).
Wie in Abschnitt 4.2.1 beschrieben konnte für diesen Datensatz über die Kombination aus logistischer Regression und Oversampling der beste mittlere AUC erreicht werden. Wie in der oberen linken Grafik zu sehen, konnten auch durch Verknüpfung der logistischen Regression mit dem SMOTE-Verfahren ähnlich gute Ergebnisse erzielt werden. Insgesamt zeigt sich, dass die logistische Regression (ggf. mit Ausnahme des Overbagging) für alle Varianten ein gutes Lernverfahren für den Datensatz darstellt. Weiterhin ist auffällig, dass vor allem Random Forest und Gradient Boosting in diesem Fall nicht gut mit Overbagging „harmonieren“. An der oberen,rechten Grafik für den Entscheidungsbaum zeigt sich die Steigerung der Performance durch Tuning und Korrekturmethoden am deutlichsten.
In vergleichbaren Studien konnte für den Datensatz *abalone19* durch Verwendung des C4.5-Algorithmus ($\rightarrow$ Implementierung für Entscheidungsbäume, d.h. Alternative zu CART) mit komplexeren Korrekturverfahren ein maximaler AUC von 0.7206 erreicht werden [13]. Dieser Wert wird bereits durch Anwendung der logistischen Regression (0.810), Gradient Tree Boosting (0.748), Support Vector Machines (0.738) oder des Random Forest (0.729) jeweils mit den Standardeinstellungen übertroffen.

Der Datensatz *coil2000* wird u.a. bei Kuhn und Johnson, S.419ff [17] diskutiert. Dabei wird der Fokus insbesondere auf den Random Forest als Lernverfahren in Verbindung mit Under- und Oversampling Methoden gelegt. Die im Rahmen dieser Arbeit auf Basis des Random Forest erzielten Ergebnisse bezüglich des AUC liegen unter den bei Kuhn und Johnson erreichten Werten. Der maximale AUC von 0.764 wurde dort mit dem dem Random Forest in Kombination mit einer Variante des Undersampling erreicht. In Abbil-

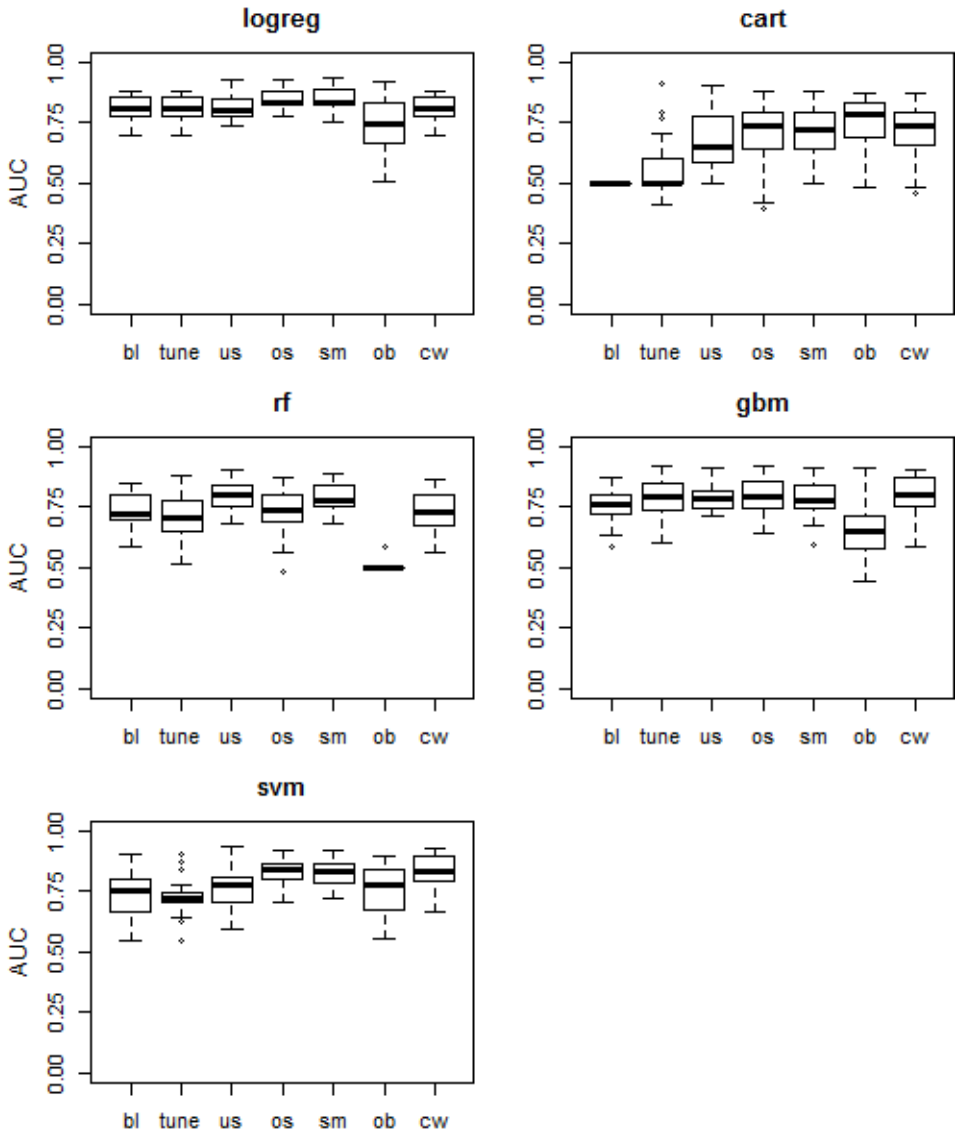

Abbildung 4.1.: Übersicht aller Verfahren für Datensatz *abalone19*

dung 4.2 sind die Ergebnisse aus den Experimenten dieser Arbeit wiederum je Lernverfahren dargestellt. Für den Random Forest stellt auch hierbei das Undersampling mit einem mittleren AUC von 0.713 die beste Korrekturmethode dar. Insgesamt liefert für diesen Datensatz jedoch das Gradient Boosting leicht bessere Ergebnisse ab. Der beste AUC wird mit einem Wert von 0.760 durch Gradient Boosting in Kombination mit Class Weighting erreicht. Grundsätzlich ist die Streuung der einzelnen AUC-Werte je Verfahren sehr gering. Die größten Steigerung des AUC lassen sich wiederum für Entscheidungsbäume in der oberen rechten Grafik erzielen, während die Verbesserungen für die logistische Regression, Random Forest und Gradient Boosting eher gering sind. Bei der SVM in der untersten Grafik lässt sich weiterhin eine Verbesserung des AUC durch die Anwendung der Korrekturmethoden erkennen. Für das Overbagging konnten hierbei keine Ergebnisse

dargestellt werden, da die entsprechenden Experimente aufgrund der Überschreitung der maximalen Laufzeit nicht erfolgreich beendet werden konnten. In Untersuchungen zum

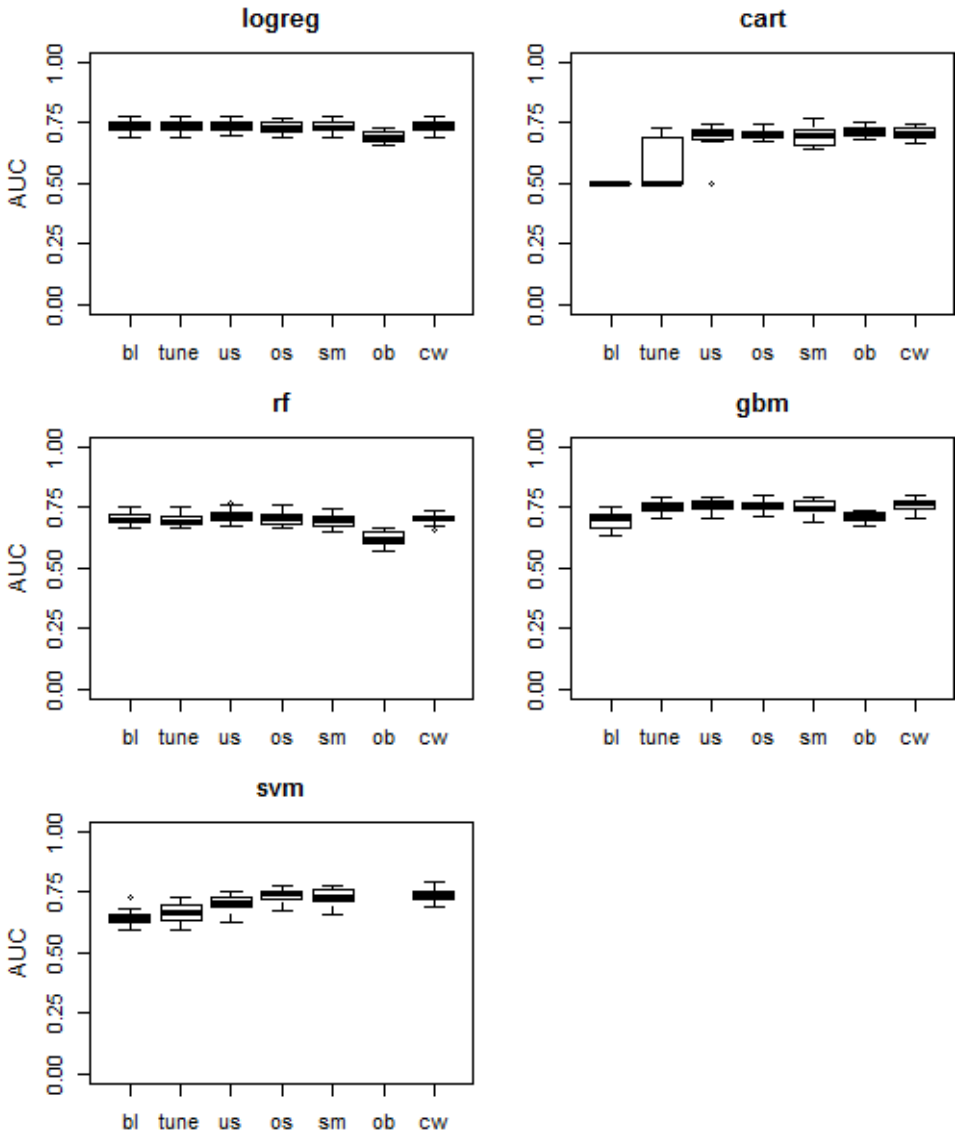

Abbildung 4.2.: Übersicht aller Verfahren für Datensatz *coil2000*

Datensatz *mammography* durch Chawla et. al [8] konnte auf Basis des C4.5-Algorithmus (→ Entscheidungsbaum) und SMOTE ein AUC von 0.933 erreicht werden. Die Ergebnisse dieser Arbeit sind in Abbildung 4.3 dargestellt. Die Streuung der einzelnen AUC-Werte ist dabei wiederum sehr gering und für eine Vielzahl an Verfahren bzw. Kombinationen konnte ein hoher AUC erreicht werden. Das Maximum der mittleren AUC-Werte weist dabei wiederum das Gradient Boosting in Kombination mit Class Weighting vor (Ø AUC = 0.956). Aufgrund der leicht geringeren Streuung der einzelnen AUC-Werte aus der Kreuzvalidierung eignet sich hierbei insbesondere auch der Random Forest in Verbindung mit SMOTE (Ø AUC = 0.952) oder Oversampling (Ø AUC = 0.951) sehr gut. Bei Be-

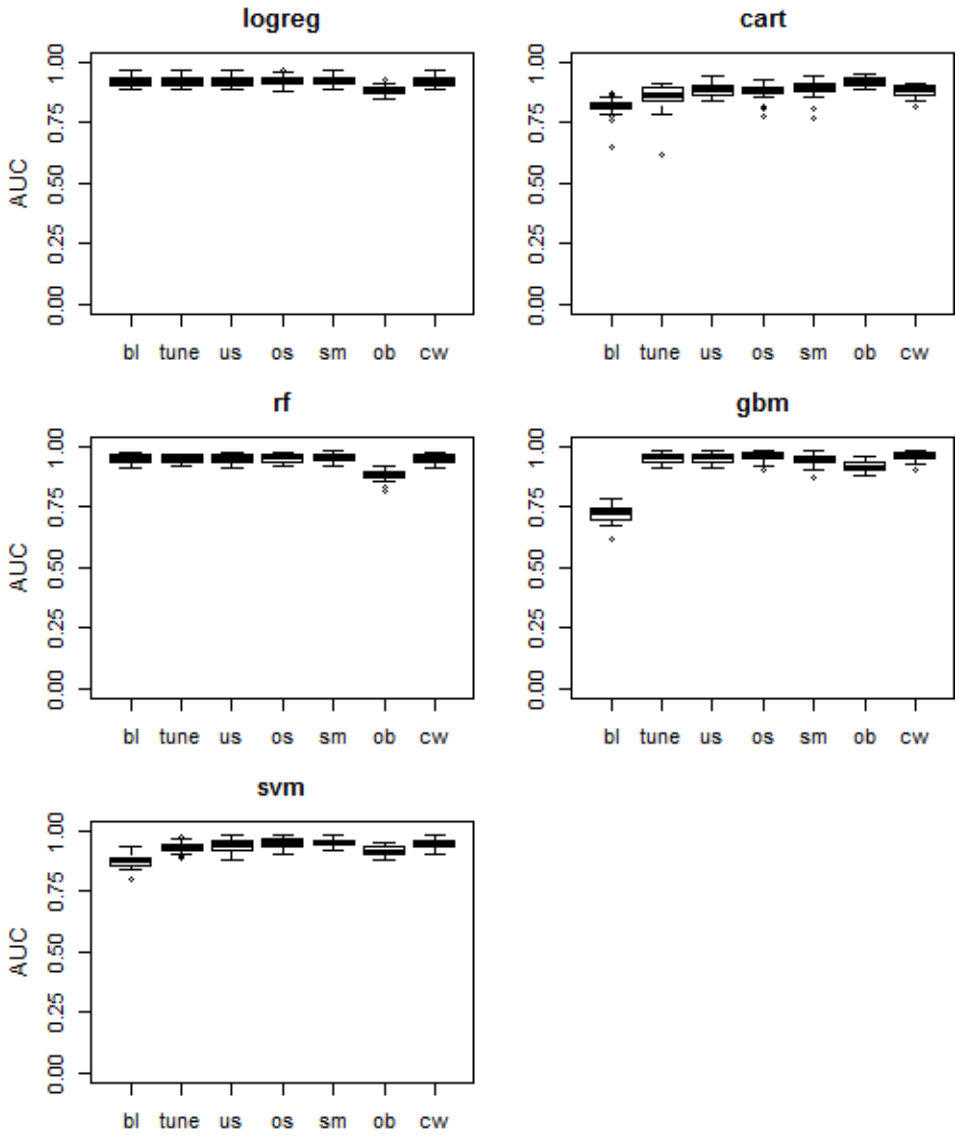

Abbildung 4.3.: Übersicht aller Verfahren für Datensatz *mammography*

trachtung des Random Forest in der mittleren linken Grafik fällt wiederum auf, dass in Verbindung mit Overbagging schlechtere Ergebnisse erzielt werden als mit den anderen Verfahren. Beim Gradient Boosting sowie der SVM liefert vor allem die zusätzliche Anwendung des Tuning eine Steigerung des AUC. Durch die Korrekturmethoden wurden bezüglich des Datensatzes *mammography* eher nur geringe Verbesserungen erreicht.

4.3. Ergebnisse je Verfahren

4.3.1. Tuning- und Korrektur-Effekte

In Tabelle 4.4 sind die größten, erreichten Verbesserungen zum einen durch Parameter-Tuning gegenüber der Anwendung des entsprechenden Verfahrens ohne Tuning (link Übersicht) sowie zum anderen durch zusätzliche Verwendung einer Korrekturmethode (Spalte *Method*) im Vergleich zur Anwendung des Lernverfahrens ohne Korrekturmethode (jedoch inklusive Parameter-Tuning) (rechte Übersicht) dargestellt.
Es zeigt sich, dass in Abhängigkeit von Datensatz und Lernverfahren sowohl durch das

Data	Learner	Base	Tuning
balance2	gbm	0.30	0.81
mammography	gbm	0.72	0.95
satelliteimage4	gbm	0.78	0.96
solarflare5	cart	0.67	0.81
pageblocks5	gbm	0.87	0.98
spectrometer42	gbm	0.83	0.94
winequality4	svm	0.73	0.83
winequality4	gbm	0.71	0.80
coil2000	cart	0.50	0.58
ozonelevel	gbm	0.82	0.90

Data	Learner	Tuning	Imbal	Method
abalone19	cart	0.56	0.75	ob
winequality4	cart	0.62	0.79	ob
balance2	cart	0.50	0.67	ob
coil2000	cart	0.58	0.71	ob
oilspill	cart	0.71	0.84	ob
spectrometer42	cart	0.79	0.92	ob
ozonelevel	cart	0.72	0.85	ob
solarflare5	svm	0.79	0.91	os
abalone19	svm	0.73	0.83	os
balance2	logreg	0.40	0.50	ob

Tabelle 4.4.: Top 10 Verbesserungen des AUC-Wertes durch Tuning und Korrekturverfahren

Tuning als auch durch die Korrekturmethoden deutliche Verbesserungen hinsichtlich des AUC erreicht werden können. So ist in den betrachteten Fällen insbesondere bei Verwendung des Gradient Boosting (gbm) ein zusätzliches Tuning in einigen Fällen sehr wirksam. Bei ergänzender Verwendung der Korrekturmethoden läßt sich in bestimmten Fällen bei Support Vector Machines (svm) durch zusätzliches Oversampling (os) sowie insbesondere bei Entscheidungsbäumen (cart) mit Overbagging (ob) eine deutliche Steigerung des AUC-Wertes erreichen.

In Tabelle 4.5 sind die durchschnittlichen, absoluten Verbesserungen des AUC durch Tuning (Spalte *Mean Tuning Opt.*) sowie durch die jeweils besten Korrekturmethode je Lernverfahren (Spalte *Mean Imbal Opt.*) datensatzübergreifend dargestellt. Für die logistische Regression (logreg) wird hierbei durch das Tuning keine Verbesserung erreicht, da sich die Tuning-Ergebnisse aufgrund nicht vorhandener Tuning-Parameter (vgl. Tabelle 3.2) nicht von den Ergebnissen des Verfahrens ohne Tuning unterscheiden.

Insbesondere beim Gradient Tree Boosting (gbm) konnte durch das Tuning im Mittel eine Verbessung des AUC erreicht werden. Für Klassifikationsbäume (cart), Support Vector

Learner	Mean Tuning Opt.	Mean Imbal Opt.
cart	0.026	0.104
gbm	0.107	0.011
logreg	0.000	0.026
rf	0.001	0.018
svm	0.015	0.038

Tabelle 4.5.: Durchschnittliche und maximale Verbesserungen des AUC durch Anwendung der Korrekturmethoden

Machines und Random Forest sind die Steigerungen des AUC durch zusätzliches Tuning im Mittel eher gering. Für alle Lernverfahren konnte darüber hinaus im Mittel eine weitere Steigerung des AUC durch Anwendung der Korrekturmethoden erreicht werden. Diese zeigt sich insbesondere bei Verwendung der Entscheidungsbäume als sehr wirksam, was jedoch wiederum vor allem auf die Kombinationen von Entscheidungsbäumen mit Overbagging zurückzuführen ist. Diese Verbindung entspricht grundsätzlich der Kombination von Random Forest ($\rightarrow$ Bagging mit Entscheidungsbäumen, vgl. Abschnitt 2.1.3) und Oversampling, was bedeutet, dass die Steigerungen des AUC hierbei hauptsächlich von der Anwendung der Bagging-Methode und weniger vom wiederholten Oversampling ausgehen.
Für die anderen Lernverfahren konnten durch die Korrekturverfahren im Mittel Steigerungen des AUC zwischen 0.011 bis 0.038 gegenüber der Anwendung der Lernverfahren inklusive Tuning „herausgeholt“ werden.

4.3.2. Vergleich Oversampling-Methoden

Bei einem datensatzübergreifenden Vergleich der Oversampling-Methoden, zeigt sich bei der Gegenüberstellung von Oversampling und SMOTE je Lernverfahren mit Ausnahme der SVM ein leichter Vorteil zugunsten des SMOTE-Verfahrens. In Tabelle 4.6 sind die Anzahlen der Fälle dargestellt, in denen das jeweilige Korrekturverfahren bessere Ergebnisse bezüglich des AUC liefern konnte als das andere. Die entsprechenden prozentualen Anteile sind in Klammern dargestellt.

In Abbildung 4.4 wurden die einzelnen Differenzen zwischen den beiden AUC-Werten ($AUC_{os} - AUC_{sm}$) in Form eines Boxplot dargestellt. Hierbei sind die resultierenden Werte (ggf. mit Ausnahme der Entscheidungsbäume zugunsten von SMOTE) ungefähr um den Wert 0 zentriert, was dafür spricht, dass sich die beiden Methoden nur sehr gering unterscheiden. Eine Auswahl ist daher eher in Abhängigkeit des betrachteten Datensatzes zu treffen.

Learner	os	sm
cart	5 (0.36)	9 (0.64)
gbm	4 (0.29)	10 (0.71)
logreg	5 (0.38)	8 (0.62)
rf	5 (0.36)	9 (0.64)
svm	9 (0.64)	5 (0.36)

Tabelle 4.6.: Anzahlen und Anteile gemäß der besseren Performance (AUC) je Lernverfahren (Oversampling/SMOTE)

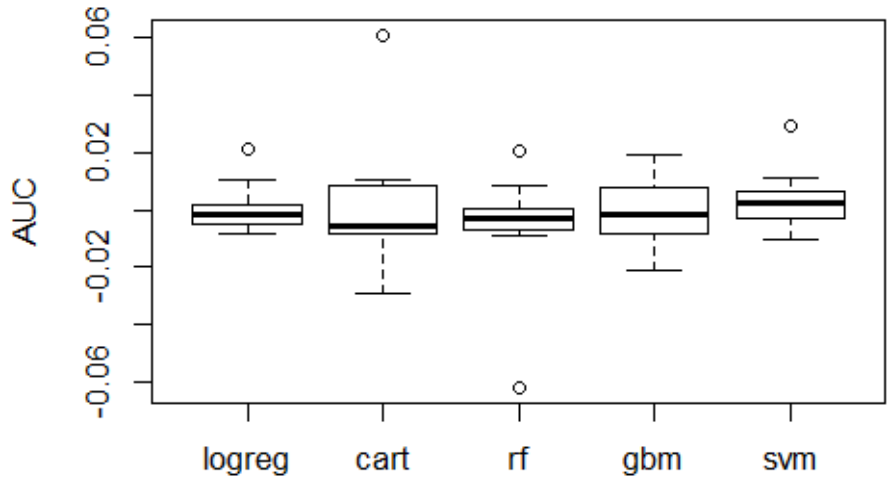

Abbildung 4.4.: AUC-Differenzen (Oversampling - SMOTE)

Bei einem Vergleich von Oversampling und Overbagging ergeben sich aus den Ergebnissen hingegen eher Präferenzen bezüglich der zu kombinierenden Lern- und Korrektuverfahren (vgl. Tabelle 4.7 sowie Boxplot der Differenzen $AUC_{os} - AUC_{ob}$ in Abbildung 4.5).

Während Overbagging als ergänzende Methode insbesondere bei Entscheidungsbäumen sehr gut funktioniert, konnten mittels Oversampling bei Random Forest, Gradient Boosting und SVM in allen Fällen bessere Ergebnisse erzielt werden. Da es sich bei Overbagging Namen gemäß um ein Bagging-Verfahren handelt, ist dieses ggf. nicht immer sinnvoll mit beliebigen Lernverfahren kombinierbar, sondern insbesondere eher für schwache Lernverfahren geeignet. Eine Kombination aus Overbagging und einem stabilen Verfahren wie dem Random Forest kann dessen Prognosegüte ggf. nicht weiter verbessern (vgl. Breiman [4]) oder wie im Fall der durchgeführten Ergebnisse zum Teil sogar verschlechtern. Grundsätzlich stellen Overbagging mit Entscheidungsbäumen sowie Oversampling mit

Learner	os	ob
cart	14 (1.00)	0 (0.00)
gbm	0 (0.00)	14 (1.00)
logreg	4 (0.31)	9 (0.69)
rf	0 (0.00)	14 (1.00)
svm	0 (0.00)	13 (1.00)

Tabelle 4.7.: Anzahlen und Anteile gemäß der besseren Performance je Lernverfahren (Oversampling/Overbagging)

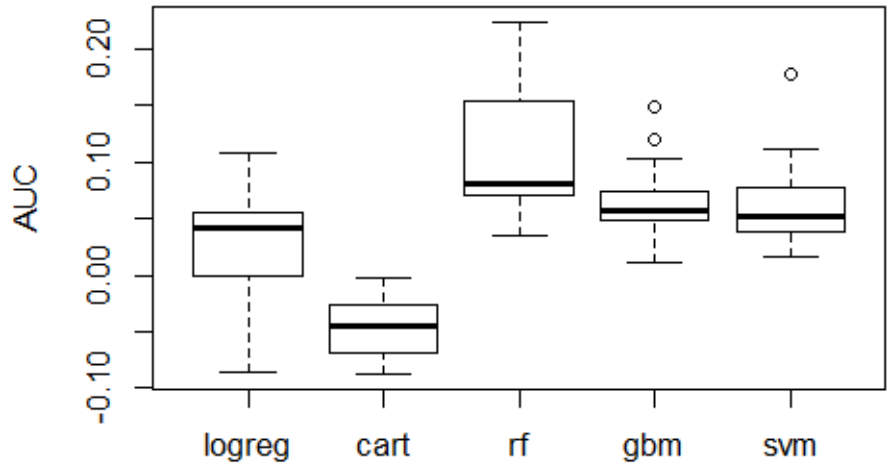

Abbildung 4.5.: AUC-Differenzen (Oversampling - Overbagging)

Random Forest die gleiche Methode dar (→ Oversampling sowie Bagging mit Bäumen). Bei Vergleich der beiden Kombinationen hinsichtlich der mittleren AUC-Werte sowie dem minimalen der maximalen AUC aus den 5-fachen Kreuzvalidierungen je Datensatz (vgl. Abbildung 4.6 erzielen Oversampling und Random Forest in den meisten Fällen die besseren Ergebnisse. Es sei jedoch darauf hingewiesen, dass für dass Overbagging aufgrund der festen Voreinstellung jeweils nur 10 Modelle erstellt wurden, während der Optimierungsbereich im Rahmen des Tuning für die Anzahl der Modelle beim Random Forest zwischen 10 bis 500 schwankt.

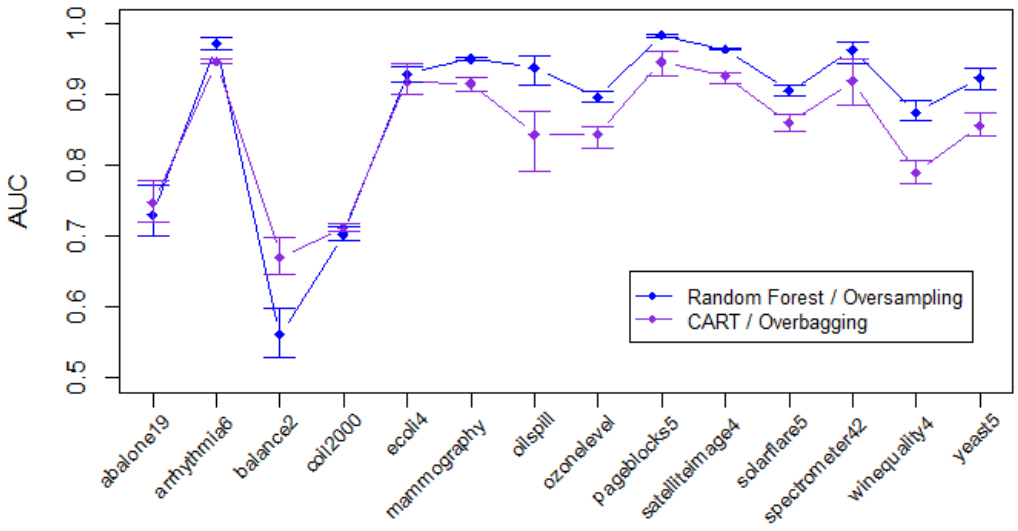

Abbildung 4.6.: Vergleich von RandomForest/Oversampling und CART/Overbagging je Datensatz

4.3.3. Analyse der Laufzeit

Wie in Abschnitt 4.1 beschrieben, konnte ein Teil der Experimente aufgrund von Fehlern und Zeitüberschreitungen nicht erfolgreich durchgeführt werden. Da des Weiteren zum Zweck der Parallelisierung der Experimente die Kreuzvalidierungsblöcke und Wiederholungen (vgl. Abschnitt 3.4) in separate Jobs aufgeteilt wurden, wurden zur Betrachtung der Laufzeit nur die Ergebnisse berücksichtigt, bei denen alle Iterationen innerhalb der Kreuzvalidierung vollständig durchlaufen konnten. Die Laufzeiten wurden anschließend über die Iterationen der betrachteten Kombination aus Datensatz und Gesamt-Verfahren (→ Lern-/Tuning-/Korrekturverfahren) summiert. In ANHANG B sind die entsprechenden Experimente, die nicht berücksichtigt wurden, inklusive der Anzahl der nicht erfolgreichen Iterationen aufgelistet.
Zur allgemeinen Analyse der Laufzeit wurde zunächst ein kompletter Regressionsbaum (ohne Pruning) mit den Einflussgrößen Lernverfahren, Methode (Tuning- und Korrekturverfahren), Anzahl Beobachtungen in der kleinen Klasse sowie der Features und IR der jeweiligen Daten angepasst (vgl. Abbildung 4.7). Die Laufzeit ist dabei in Stunden dargestellt.

Im ersten Split erfolgt zunächst eine Aufteilung nach der Methode - der komplette rechte Ast des Baumes bezieht sich somit nur auf Overbagging, bei dessen Anwendung ein Job im Mittel ungefähr 11.4 Stunden läuft. In Kombination mit Gradient Boosting erhöht

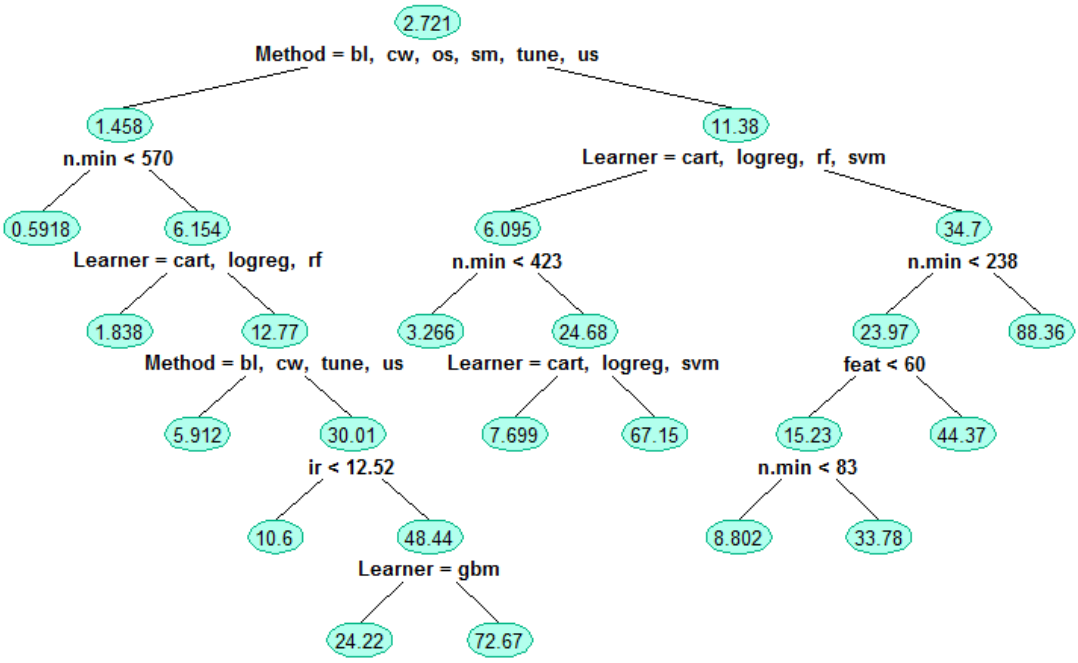

Abbildung 4.7.: Regressionsbaum zur Laufzeit in Stunden

sich die mittlere Laufzeit weiter auf 34.7 Stunden. Für die restlichen Methoden im linken Ast beträgt die mittlere Laufzeit ca. 1.5 Stunden.

In Abbildung 4.8 sind abschließend AUC und Laufzeit für die einzelnen Verfahren gegenüberstellt. Unter Berücksichtigung beider Größen lässen sich die besten Ergebnisse per SMOTE und Class Weighting erzielen.

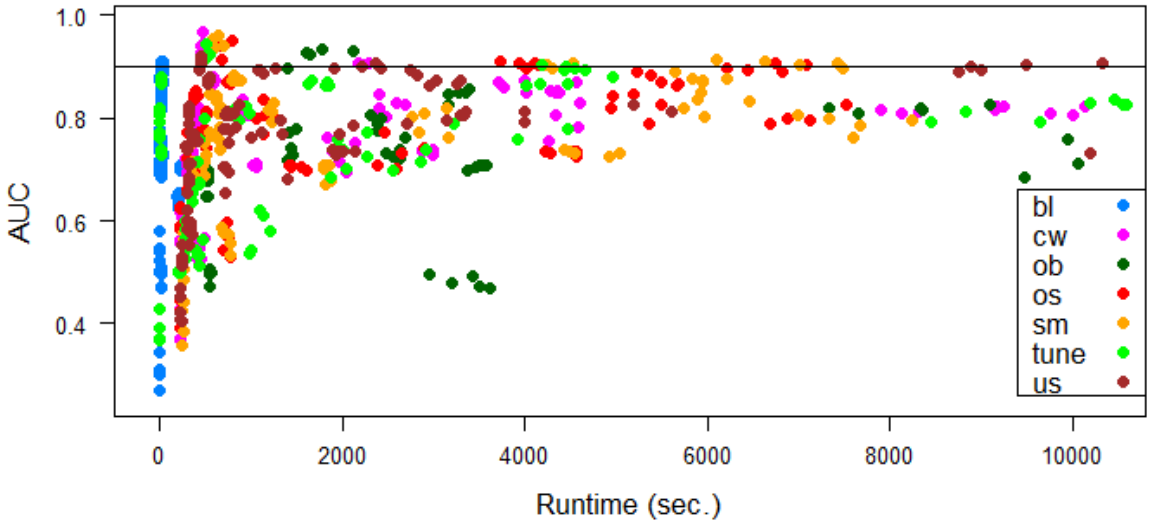

Abbildung 4.8.: AUC vs. Runtime

5. Zusammenfassung und Ausblick

Im Rahmen der binären Klassifikation stellt im Falle unbalancierter Klassen insbesondere die korrekte Vorhersage der Beobachtungen der kleinen Klasse eine Herausforderung dar. Wie anhand der in dieser Arbeit durchgeführten Experimente gezeigt wurde, können jedoch bereits zum Teil sehr gute Ergebnisse durch die Auswahl eines geeigneten Lernverfahrens erzielt werden. Bei übergreifender Betrachtung zeigte sich hierbei z.B. der Random Forest für verschiedene Datensätze als wirksam. Nach zusätzlichem Parameter-Tuning führten zudem Gradient Boosting und Support Vector Machines in vielen Fällen bereits ohne ergänzende Korrekturmethoden zu guten Ergebnissen. Sowohl durch das Tuning als auch durch die zusätzliche Anwendung der implementierten Korrekturmethoden konnten jedoch in nahezu allen Fällen weitere, zumindest geringe Verbesserungen der Prognosegüte erreicht werden. Somit eignen sich selbst „einfachere“ Korrekturmethoden wie Undersampling oder Class Weighting als Ergänzung bzw. als Verfeinerung für gute und stabile Lernverfahren. Die Kombination aus gutem Lernverfahren und ergänzender Korrekturmethode erscheint insbesondere auch aus Sicht der Laufzeit geeigneter als komplexere Korrekturmethoden wie z.B. Overbagging in Verbindung mit schwachen Lernverfahren. In den durchgeführten Experimenten konnten insbesondere durch das SMOTE-Verfahren sowie durch Class Weighting gute Ergebnisse mit akzeptabler Laufzeit erzielt werden.
Zur weiterführenden Analyse empfiehlt sich die Betrachtung weiterer, einfacher Korrekturverfahren oder Erweiterungen der bis hierhin untersuchten Methoden. Interessant wäre dabei, ob z.B. durch „intelligentes Undersampling“ der Kombination von SMOTE mit Undersampling weitere Verbesserungen entstehen.

A. Übersicht der kleinen Klassen

Da es sich bei den betrachteten Datensätzen größtenteils um Multi-Klassen-Probleme handelt, wurde im Rahmen der Vorverarbeitung eine der vorhandenen Klassen als kleine Klasse selektiert und alle anderen Klassen zu einer großen Klasse zusammengefasst. Die Auswahl der kleinen Klasse je Datensatz ist in folgender Übersicht dargestellt:

Data	y	minClass
abalone	Classnumberofrings	19
arrhythmia	class	6
anneal	class	5
balancescale	class	B
car	class	vgood
coil2000	CARAVAN	1
ecoli	class	imU
kropt	game	sixteen
letter	class	Z
mammography	class	1
nursery	class	veryrecom
oilspill	class	1
optdigits	class	0
ozonelevel	Class	1
pageblocks	class	5
pendigits	class	9
scene	sunset	1
satelliteimage	class	4
solarflare	class	F
spectrometer	LRS.class	42
vowel	Class	hid
winequality	quality	4
yeast	classproteinlocalization	ME2

Tabelle A.1.: Festlegung der kleinen Klasse bei Multi-Klassen-Problemen

B. Übersicht der fehlerhaften Jobs

Data	Learner	Method	NrExcluded
abalone19	cw	gbm	1
abalone19	ob	gbm	2
abalone19	os	svm	1
abalone19	tune	gbm	2
abalone19	us	gbm	2
anneal5	bl	logreg	5
arrhythmia6	tune	gbm	1
balance2	sm	gbm	1
balance2	tune	gbm	1
coil2000	cw	gbm	1
coil2000	ob	gbm	5
mammography	cw	gbm	1
mammography	os	gbm	1
mammography	us	logreg	1
ozonelevel	os	gbm	1
ozonelevel	tune	gbm	1
ozonelevel	us	cart	1
ozonelevel	us	svm	1
pageblocks5	os	gbm	1
pageblocks5	us	gbm	3

Data	Learner	Method	NrExcluded
satelliteimage4	cw	gbm	1
satelliteimage4	os	gbm	1
solarflare5	bl	logreg	5
solarflare5	cw	logreg	5
solarflare5	ob	gbm	1
solarflare5	ob	logreg	5
solarflare5	os	logreg	5
solarflare5	sm	gbm	1
solarflare5	sm	logreg	5
solarflare5	tune	logreg	5
solarflare5	us	logreg	5
spectrometer42	cw	gbm	1
spectrometer42	tune	gbm	1
winequality4	ob	gbm	1
winequality4	us	gbm	2
yeast5	cw	gbm	1
yeast5	ob	gbm	1
yeast5	os	gbm	1
yeast5	tune	gbm	1
yeast5	us	gbm	1

Tabelle B.1.: Ausgeschlossene Experimente bei der Analyse der Laufzeit

C. Ergebnisse je Durchlauf

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
scenesunset	2407	294	5.61	rf	0.99	svm	0.92	0.03
ecoli4	336	7	8.60	rf	0.94	svm	0.62	0.07
optdigits0	5620	62	9.14	svm	1.00	svm	1.00	0.00
satelliteimage4	6435	36	9.28	rf	0.96	rf	0.64	0.06
pendigits9	10992	16	9.42	rf	1.00	svm	0.99	0.00
vowel1	990	12	10.00	svm	1.00	svm	0.99	0.00
spectrometer42	531	100	10.80	rf	0.96	svm	0.61	0.06
balance2	625	4	11.76	svm	0.90	svm	0.00	0.08
anneal5	898	31	12.40	cart	1.00	cart	1.00	0.00
coil2000	9822	85	15.76	logreg	0.74	rf	0.07	0.07
arrhythmia6	452	262	17.08	gbm	0.97	cart	0.68	0.03
oilspill	937	48	21.85	rf	0.93	svm	0.54	0.03
solarflare5	1066	11	23.79	logreg	0.91	logreg	0.19	0.05
car4	1728	6	25.58	rf	1.00	svm	0.93	0.01
letter26	20000	16	26.25	rf	1.00	rf	0.95	0.00
yeast5	1484	8	28.10	rf	0.93	cart	0.34	0.03
winequality4	6497	11	29.08	rf	0.87	rf	0.19	0.03
ozonelevel	2536	72	33.74	rf	0.90	logreg	0.23	0.04
nursery3	12960	8	38.51	rf	1.00	svm	0.95	0.00
mammography	11183	6	42.01	rf	0.95	rf	0.67	0.01
pageblocks5	5473	10	46.59	rf	0.99	rf	0.72	0.01
kropt16	28056	6	70.94	rf	1.00	svm	0.70	0.01
abalone19	4177	8	129.53	logreg	0.81	logreg	0.00	0.01

Tabelle C.1.: Ergebnisse Baseline

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	svm	0.94	gbm	0.64	0.06
satelliteimage4	6435	36	9.28	svm	0.97	svm	0.71	0.05
spectrometer42	531	100	10.80	svm	0.96	svm	0.59	0.06
balance2	625	4	11.76	svm	0.93	gbm	0.17	0.10
coil2000	9822	85	15.76	gbm	0.76	gbm	0.11	0.07
arrhythmia6	452	262	17.08	rf	0.98	rf	0.76	0.03
oilspill	937	48	21.85	rf	0.93	svm	0.52	0.03
solarflare5	1066	11	23.79	logreg	0.91	cart	0.22	0.05
yeast5	1484	8	28.10	rf	0.92	rf	0.42	0.03
winequality4	6497	11	29.08	rf	0.87	gbm	0.24	0.03
ozonelevel	2536	72	33.74	rf	0.90	logreg	0.23	0.04
mammography	11183	6	42.01	gbm	0.95	gbm	0.70	0.01
pageblocks5	5473	10	46.59	rf	0.98	rf	0.72	0.01
abalone19	4177	8	129.53	logreg	0.81	gbm	0.05	0.01

Tabelle C.2.: Ergebnisse Tuning

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	svm	0.94	logreg	0.64	0.09
satelliteimage4	6435	36	9.28	gbm	0.96	svm	0.72	0.05
spectrometer42	531	100	10.80	svm	0.96	svm	0.66	0.06
balance2	625	4	11.76	svm	0.91	svm	0.22	0.19
coil2000	9822	85	15.76	gbm	0.76	gbm	0.23	0.13
arrhythmia6	452	262	17.08	rf	0.98	gbm	0.78	0.03
oilspill	937	48	21.85	rf	0.92	rf	0.51	0.04
solarflare5	1066	11	23.79	logreg	0.92	rf	0.38	0.08
yeast5	1484	8	28.10	rf	0.93	rf	0.39	0.04
winequality4	6497	11	29.08	rf	0.87	rf	0.30	0.06
ozonelevel	2536	72	33.74	rf	0.90	rf	0.35	0.04
mammography	11183	6	42.01	gbm	0.95	rf	0.69	0.01
pageblocks5	5473	10	46.59	rf	0.99	rf	0.72	0.01
abalone19	4177	8	129.53	logreg	0.82	logreg	0.06	0.08

Tabelle C.3.: Ergebnisse Undersampling

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	svm	0.94	svm	0.66	0.09
satelliteimage4	6435	36	9.28	svm	0.97	gbm	0.72	0.05
spectrometer42	531	100	10.80	svm	0.97	svm	0.70	0.05
balance2	625	4	11.76	svm	0.94	svm	0.60	0.09
coil2000	9822	85	15.76	gbm	0.76	gbm	0.25	0.15
arrhythmia6	452	262	17.08	rf	0.97	gbm	0.78	0.03
oilspill	937	48	21.85	rf	0.94	svm	0.56	0.04
solarflare5	1066	11	23.79	rf	0.91	rf	0.39	0.08
yeast5	1484	8	28.10	rf	0.92	logreg	0.39	0.06
winequality4	6497	11	29.08	rf	0.88	gbm	0.29	0.06
ozonelevel	2536	72	33.74	svm	0.90	gbm	0.34	0.05
mammography	11183	6	42.01	gbm	0.96	rf	0.70	0.01
pageblocks5	5473	10	46.59	rf	0.98	rf	0.71	0.01
abalone19	4177	8	129.53	logreg	0.84	gbm	0.06	0.08

Tabelle C.4.: Ergebnisse Oversampling

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	gbm	0.94	logreg	0.67	0.08
satelliteimage4	6435	36	9.28	svm	0.97	gbm	0.71	0.05
spectrometer42	531	100	10.80	rf	0.97	svm	0.73	0.05
balance2	625	4	11.76	svm	0.95	svm	0.66	0.07
coil2000	9822	85	15.76	gbm	0.75	svm	0.23	0.16
arrhythmia6	452	262	17.08	rf	0.98	gbm	0.78	0.03
oilspill	937	48	21.85	rf	0.94	gbm	0.55	0.04
solarflare5	1066	11	23.79	logreg	0.91	svm	0.32	0.09
yeast5	1484	8	28.10	rf	0.92	logreg	0.40	0.05
winequality4	6497	11	29.08	rf	0.88	rf	0.30	0.04
ozonelevel	2536	72	33.74	rf	0.91	svm	0.35	0.05
mammography	11183	6	42.01	rf	0.95	rf	0.71	0.01
pageblocks5	5473	10	46.59	rf	0.99	rf	0.72	0.01
abalone19	4177	8	129.53	logreg	0.84	rf	0.07	0.02

Tabelle C.5.: Ergebnisse SMOTE

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	cart	0.92	logreg	0.67	0.08
satelliteimage4	6435	36	9.28	svm	0.93	svm	0.73	0.05
spectrometer42	531	100	10.80	svm	0.93	svm	0.72	0.05
balance2	625	4	11.76	svm	0.92	svm	0.63	0.09
coil2000	9822	85	15.76	cart	0.71	gbm	0.25	0.13
arrhythmia6	452	262	17.08	gbm	0.95	gbm	0.77	0.03
oilspill	937	48	21.85	logreg	0.89	svm	0.55	0.04
solarflare5	1066	11	23.79	cart	0.86	logreg	0.39	0.07
yeast5	1484	8	28.10	cart	0.86	rf	0.38	0.05
winequality4	6497	11	29.08	cart	0.79	gbm	0.29	0.06
ozonelevel	2536	72	33.74	cart	0.86	gbm	0.34	0.05
mammography	11183	6	42.01	gbm	0.92	rf	0.71	0.01
pageblocks5	5473	10	46.59	logreg	0.96	rf	0.73	0.01
abalone19	4177	8	129.53	svm	0.75	gbm	0.06	0.07

Tabelle C.6.: Ergebnisse Overbagging

prob	n	feat	ir	lrn.auc	mean.auc	lrn.f1	mean.f1	mean.mmce
ecoli4	336	7	8.60	svm	0.95	svm	0.67	0.08
satelliteimage4	6435	36	9.28	svm	0.97	gbm	0.72	0.05
spectrometer42	531	100	10.80	svm	0.96	gbm	0.69	0.05
balance2	625	4	11.76	svm	0.94	gbm	0.24	0.14
coil2000	9822	85	15.76	gbm	0.76	gbm	0.25	0.14
arrhythmia6	452	262	17.08	gbm	0.97	gbm	0.78	0.03
oilspill	937	48	21.85	svm	0.93	svm	0.56	0.03
solarflare5	1066	11	23.79	logreg	0.91	svm	0.34	0.08
yeast5	1484	8	28.10	rf	0.92	gbm	0.38	0.05
winequality4	6497	11	29.08	rf	0.85	gbm	0.29	0.06
ozonelevel	2536	72	33.74	gbm	0.91	gbm	0.36	0.05
mammography	11183	6	42.01	gbm	0.96	gbm	0.70	0.01
pageblocks5	5473	10	46.59	rf	0.99	gbm	0.70	0.01
abalone19	4177	8	129.53	svm	0.82	gbm	0.06	0.07

Tabelle C.7.: Ergebnisse Class Weighting

Literaturverzeichnis

[1] B. Bischl, T. Kühn, and G. Szepannek. On class imbalancy correction for classification algorithms in credit scoring. 2014.

[2] B. Bischl, M. Lang, and O. Mersmann. *BatchExperiments: Statistical experiments on batch computing clusters.*, 2014.

[3] B. Bischl, M. Lang, O. Mersmann, J. Rahnenfuehrer, and C. Weihs. Computing on high performance clusters with r: Packages batchjobs and batchexperiments. Technical Report 1, TU Dortmund, 2011.

[4] L. Breiman. Bagging predictors. *Mach. Learn., vol. 24, no. 2, pp. 123-140*, 1996.

[5] L. Breiman. Random forests. *Machine Learning, vol. 45, no. 1, pp. 5-32*, 2001.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[7] J. Burez and D. Van den Poel. Handling class imbalance in customer churn prediction. *Expert Syst. Appl., vol. 36, no. 3, pp. 4626-4636*, 2009.

[8] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res., vol. 16, no. 1, pp. 321-357*, 2002.

[9] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory, vol. 13, no. 1, pp. 21-27*, 1967.

[10] M. Di Martino, F. Decia, J. Molinelli, and A. Fernández. Improving electric fraud detection using class imbalance strategies. In *ICPRAM 2012 - Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods, Volume 2, Vilamoura, Algarve, Portugal, 6-8 February, 2012*, pages 135–141, 2012.

[11] C. Elkan. The foundations of cost-sensitive learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[12] L. Fahrmeir, T. Kneib, S. Lang, and B. Marx. *Regression. Models, Methods and Applications*. Springer Verlag, 2013.

[13] M. Galar, A. Fernández, E. Barrenechea Tartas, H. Bustince Sola, and F. Herrera. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C, vol. 44, no. 4, pp. 463–484*, 2012.

[14] J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics 27: pp. 857-874*, 1971.

[15] H. Han, W.-Y. Wang, and B.-H. Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I*, pages 878–887. Springer-Verlag, 2005.

[16] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction - Second Edition*. Springer, 2009.

[17] M. Kuhn and K. Johnson. *Applied Predictive Modeling*. Springer, 2013.

[18] C.X. Ling and V.S. Sheng. Class imbalance problem. In *Encyclopedia of Machine Learning*, page 171. 2010.

[19] C.X. Ling, Q. Yang, J. Wang, and S. Zhang. Decision trees with minimal costs. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[20] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[21] M.A. Mazurowski, P.A. Habas, J.M. Zurada, J.Y. Lo, J.A. Baker, and G.D. Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks, vol. 21, no. 2-3, pp. 427-436*, 2008.

[22] D.K. McClish. Analyzing a portion of the roc curve. *Medical Decision Making., vol. 9, no. 3, pp. 190-195*, 2002.

[23] K.M. Ting. Inducing cost-sensitive trees via instance weighting. In *Principles of Data Mining and Knowledge Discovery, Second European Symposium, PKDD '98, Nantes, France, September 23-26, 1998, Proceedings*, pages 139–147, 1998.

[24] J. Vanschoren, J.N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explor. Newsl., vol. 15, no. 2, pp. 49-60*, 2014.

[25] W. Wei, J. Li, L. Cao, Y. Ou, and J. Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web, vol. 16, no. 4, pp. 440-475*, 2013.

[26] Z.B. Zhu and Z.H. Song. Fault diagnosis based on imbalance modified kernel fisher discriminant analysis. *Chemical Engineering Research and Design, vol. 88, no. 8, pp. 936951*, 2010.