TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit

Machine-Learning-Methoden zur Einschätzung des Konkursrisikos von Unternehmen anhand von Finanzdaten

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Finanz- und Versicherungsmathematik

eingereicht von

Matthias Remta, 01526683

ausgeführt am

Institut für Stochastik und Wirtschaftsmathematik der Technischen Universität Wien

Betreuer:

Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Gerhold

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract (English)

This Thesis is about bankruptcy forecasting. The goal was to compare four models: Logistic Regression, Bayesian Logistic Regression, Neural Networks and Support Vector Machine. The models were trained on financial data from companies. Then their respective performance was evaluated via different measures. No model performed better than the other models in every aspect. Furthermore the theoretical background of the used models and data-handling was discussed.

Abstract (Deutsch)

Das Thema dieser Arbeit ist die Vorhersage von Unternehmenskonkursen. Das Ziel war, vier Modelle zu vergleichen: die Logistische Regression, die Bayessche Logistische Regression, die Neural Networks und die Support Vector Machine. Dazu wurden die Modelle mittels Finanzdaten von Unternehmen kalibriert. Anschließend wurde die Performance der Modelle anhand verschiedener Maße evaluiert. Dabei konnte kein Modell in allen Aspekten besser abschneiden als alle anderen. Darüber hinaus wurden auch die Theorie der verwendeten Modelle und der Umgang mit Daten behandelt.

Danksagung

Zunächst möchte ich Dr. Stefan Gerhold dafür danken, dass er meine Diplomarbeit betreut hat. Darüber hinaus möchte ich Dr. Richard Warnung danken, der mein Interesse für das Thema geweckt hat.

Ich möchte mich auch bei meinen Studienkolleginnen und -kollegen bedanken, welche zu guten Freunden wurden. Durch euch wurde das Studium zu einer unvergesslichen Zeit!

Ein besonderer Dank gilt meiner Familie, allen voran meinen Eltern, die mir das Studium ermöglicht und mich stets unterstützt haben.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

_____

Matthias Remta

Einverständniserklärung zur Plagiatsprüfung

Ich nehme zur Kenntnis, dass die vorgelegte Arbeit mit geeigneten und dem derzeitigen Stand der Technik entsprechenden Mitteln (Plagiat-Erkennungssoftware) elektronisch-technisch überprüft wird. Dies stellt einerseits sicher, dass bei der Erstellung der vorgelegten Arbeit die hohen Qualitätsvorgaben im Rahmen der ausgegebenen der an der TU Wien geltenden Regeln zur Sicherung guter wissenschaftlicher Praxis - „Code of Conduct“ (Mitteilungsblatt 2007, 26. Stück, Nr. 257 idgF.) an der TU Wien eingehalten wurden. Zum anderen werden durch einen Abgleich mit anderen studentischen Abschlussarbeiten Verletzungen meines persönlichen Urheberrechts vermieden.

_____

Matthias Remta

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit befasst sich mit der Vorhersage (oder auch Prognose) von Unternehmenskonkursen. In Österreich wird der Begriff Konkurs folgendermaßen definiert: „Konkurs ist ein gerichtliches Verfahren der kostensparenden Vermögensverwertung zahlungsunfähiger Schuldnerinnen/Schuldner. Ziel ist die gleichmäßige Aufteilung des Vermögens auf die Gläubigerinnen/die Gläubiger.“ [31] Wenn ein Unternehmen nicht mehr in der Lage ist, seinen Verpflichtungen nachzukommen beziehungsweise seine Verbindlichkeiten zurückzuzahlen [1], dann wird dies meist negative Folgen für die Geschäftspartner, Investoren und Kreditgeber des Unternehmens haben. Dementsprechend werden diese Personen oder Unternehmen Interesse daran haben, zu prognostizieren, ob ein Unternehmen in Konkurs gehen wird oder eben nicht beziehungsweise dieses Risiko zu quantifizieren. Informationen über den Zustand eines Unternehmens finden sich in dessen Finanzdaten, also beispielsweise im Jahresabschluss. A priori ist jedoch nicht immer klar, wie bilanzielle Werte und Finanzkennzahlen die Wahrscheinlichkeit eines Konkurses beeinflussen. Noch ungewisser ist darüber hinaus, welches Ausmaß der Einfluss dieser Werte auf die Konkurswahrscheinlichkeit hat. An dieser Stelle kommen Machine-Learning-Methoden ins Spiel. Simpel ausgedrückt stellen diese einen Zusammenhang zwischen den Finanzdaten und der Konkurswahrscheinlichkeit via statistischer Modelle her. Die Herausforderung dabei ist es, ein Modell zu finden, das möglichst zuverlässig die stabilen Unternehmen, welche nicht in Konkurs gehen werden, von jenen Unternehmen zu separieren, denen bald ein Konkurs bevorsteht. Der Ansatz dabei ist stets der gleiche: man beobachtet Unternehmen und notiert einerseits ihre Finanzdaten und andererseits, ob diese in Konkurs gegangen sind oder nicht. Dies muss man nicht notwendigerweise selbst tun, teilweise sind derartige Daten auch frei verfügbar. Dann lässt man die Modelle von diesen Daten lernen, oder formaler ausgedrückt: man kalibriert die Modelle. Bevor man die Modelle zum Einsatz bringt, möchte man sich üblicherweise vergewissern, wie gut diese ihre Aufgabe erfüllen. Dazu kann man erneut Finanzdaten von Unternehmen verwenden, von denen man weiß, ob sie in Konkurs gegangen sind. Man liefert den Modellen einfach nur die Finanzdaten und lässt diese Vorhersagen treffen. Dann gleicht man die Vorhersagen der Modellen mit den tatsächlichen Resultaten ab.

Ziel dieser Arbeit ist es nicht, ein möglichst gutes Modell zur Vorhersage von Konkurswahrscheinlichkeiten zu entwickeln. Vielmehr geht es darum, unterschiedliche Modelle und Herangehensweisen vorzustellen und zu analysieren. Ein weiteres Ziel dieser Arbeit ist es, den Entwicklungsprozess eines Modells zur Vorhersage von Konkurswahrscheinlichkeiten - von der Analyse der Daten bis zur Evaluierung der Performance - zu explorieren und anhand von echten Daten durchzuexerzieren. Aus der Vielzahl der möglichen

[1] Mukeri, Shaik und Gaikwad verwenden dies sinngemäß als Definition eines Konkurses. [29, S. 2]

Machine-Learning-Methoden wurden dabei vier vorab ausgewählt: die Logistische Regression, welche schon vergleichsweise lange bekannt ist und beispielsweise im Bereich des Kreditrisikos angewendet wird, die Bayessche Logistische Regression, welche eine Erweiterung der klassischen Logistischen Regression darstellt, Neural Networks, die sich durch hohe Anpassungsfähigkeit auszeichnen und die Support Vector Machine, welche einen geometrischen Ansatz verfolgt. Bis auf die Support Vector Machine liefern alle diese Methoden eine Schätzung der Konkurswahrscheinlichkeit, die dann interpretiert werden muss. Die Support Vector Machine liefert hingegen direkt eine Prognose, ob es zu einem Konkurs kommen wird. Im Zuge der Arbeit werden alle diese Modelle auf die Daten angewendet und miteinander verglichen. Dieser Vergleich stellt den Schlusspunkt und gleichzeitig das Hauptziel dieser Arbeit dar, in welchem die zuvor genannten Ziele münden.

In Kapitel 2 werden die vier Machine-Learning-Methoden vorgestellt, welche bereits erwähnt wurden. Dabei wird jeweils eine Definition des Modells gegeben und beschrieben, wie das Modell kalibriert werden kann. Darüber hinaus werden Besonderheiten sowie potenzielle Stärken und Schwächen der Modelle beleuchtet. In Kapitel 3 werden diverse Aspekte des Themas Daten erläutert. Dies reicht von initialen Analysen der Daten über verschiedene Aufbereitungsschritte bis hin zu den finalen Vorbereitungen, die getroffen werden müssen oder können, bevor die Daten in die Modelle eingespeist werden. Kapitel 4 beschreibt den Prozess, der schließlich tatsächlich durchlaufen wurde, und präsentiert sowie diskutiert die Ergebnisse der Modelle, die diese auf den realen Daten erzielen konnten. Dabei wurde Wert darauf gelegt, getroffene Entscheidungen transparent zu beschreiben und zu begründen. Abgeschlossen wird die Arbeit durch Kapitel 5, in welchem die wichtigsten Eckpfeiler der Arbeit nochmal zusammengefasst werden und ein finales Fazit gegeben wird.

2 Methoden

Ziel dieses Kapitels ist es, einen Überblick über Machine-Learning-Methoden zu geben, welche zur Vorhersage von Unternehmenskonkursen verwendet werden können. Aus der Fülle der zur Verfügung stehenden Methoden wurden folgende drei ausgewählt: Generalized Linear Model, Support Vector Machine und Artificial Neural Network. Dabei sollen beim GLM zwei verschiedene Ansätze untersucht werden - zunächst die klassische Logistische Regression und dann ein spezieller Bayesscher Ansatz, bei dem a priori Verteilungen der Regressionskoeffizienten angenommen werden. Im Speziellen sollen die theoretischen Hintergründe der einzelnen Methoden herausgearbeitet werden.
Es wird in diesem Kapitel generell vorausgesetzt, dass ein Datensatz (x,y) zur Verfügung steht. Dabei werden die Einträge des Vektors y als Zielvariablen bezeichnet und die i-te Zeile der Matrix x enthält die zu y_i gehörenden Einflussvariablen. Die Spalten von x werden hingegen als Merkmale bezeichnet.

2.1 Generalized Linear Model

Das Generalized Linear Model ist eine Erweiterung des klassischen linearen Modells.

Definition 2.1. *Es seien $Y_1,\dots,Y_k$ paarweise unabhängige Zielvariablen und $\tilde{x}_i = (x_{i1},\dots,x_{in})$, $i = 1,\dots,k$ die zugehörigen Einflussvariablen. Zudem seien $\varepsilon_1,\cdots,\varepsilon_k$ Zufallsvariablen (Fehlerterme) mit den folgenden Eigenschaften:*

- $\mathbb{E}[\varepsilon_i] = 0$
- $\mathbb{V}[\varepsilon_i] = \sigma$
- $Cov[\varepsilon_i,\varepsilon_j] = 0,\ i \neq j$

*Dann ist das **lineare Modell** gegeben durch*

$$Y_i = \beta_0 + \tilde{x}_i\tilde{\beta} + \varepsilon_i \tag{2.1}$$

*wobei β_0 ein Skalar und $\tilde{\beta}$ ein n-dimensionaler Vektor ist. Dabei soll $\beta_0 + \tilde{x}_i\tilde{\beta}$, $i = 1,\dots,k$ als systematische Komponente und ε_i, $i = 1,\dots,k$ als zufällige Komponente des linearen Modells bezeichnet werden. Ziel der **linearen Regression** ist es, nun aus konkreten Daten (x,y) Schätzer für β_0 und $\tilde{\beta}$ zu konstruieren.*

[24, S. 360ff]

Eine praktische Konvention ist, $x_i := (1,\tilde{x}_i)$ zu setzen, also einen konstanten Eintrag mit Wert 1 zu $\tilde{x}$ hinzuzufügen. Dann kann man auch $\beta := (\beta_0,\tilde{\beta})^\top$ setzen und Gleichung (2.1) kann umformuliert werden zu

$$Y_i = x_i\beta + \varepsilon_i.$$

Aus der Definition des linearen Modells ergibt sich die Frage, wie β aus gegebenen Daten (x,y) geschätzt werden kann. Dazu können verschiedene Ansätze gewählt werden. Sehr bekannt ist die Methode der kleinsten Quadrate:

$$\sum_{i=1}^{n}(y_i - x_i\beta)^2 \to min!$$

Verwendet man die Matrixschreibweise, so lautet das Problem

$$(y - x\beta)^\top (y - x\beta) \to min!$$

Bildet man hier die partiellen Ableitungen nach den Komponenten von β, so erhält man das Gleichungssystem

$$x^\top (y - x\beta) = 0.$$

Wenn die Matrix $x^\top x$ invertierbar ist, so erhält man die eindeutige Lösung

$$\hat{\beta} = (x^\top x)^{-1} x^\top y,$$

welche der gesuchte Schätzer für β ist. [19, S. 12f]

Obwohl die lineare Regression eine recht simple Methode ist, können mittels dieser oft gute Modelle gebildet werden. Es gibt jedoch auch einige Nachteile. Beispielsweise können die geschätzten Werte in $(-\infty,\infty)$ liegen, obwohl der Wertebereich der Zufallsvariable beschränkt oder sogar diskret ist. Daher wurden verschiedene Erweiterungen des linearen Modells entwickelt, welche besser auf spezifische Anwendungsbereiche spezalisiert sind. Nelder und Wedderburn konnten viele dieser Erweiterungen in einem theoretischen Modell zusammenfassen, dem Generalized Linear Model (kurz GLM). [30, S.]

Definition 2.2. *Seien $Y_1,\ldots,Y_k$ paarweise unabhängige Zielvariablen und $x_i = (x_{i0},\ldots,x_{in})^\top$, $i = 1,\ldots,k$ die zugehörigen Einflussvariablen. Das **Generalized Linear Model** besteht aus drei Komponenten:*

1. *Zufällige Komponente: Die Zielvariablen besitzen Verteilungen mit Dichtefunktionen beziehungsweise Zähldichten der Form:*

$$f(y_i;\theta_i,\phi) = exp\left((y_i\theta_i - b(\theta_i))/a(\phi) + c(y_i,\phi)\right) \tag{2.2}$$

Der Ausdruck (2.2) wird als „Exponential-Dispersion-Model“ bezeichnet. Wenn ϕ

bekannt ist, erhält man als Spezialfall eine exponentielle Familie:

$$f(y_i;\theta_i) = A(\theta_i)B(y_i)exp(y_iQ(\theta_i)) \tag{2.3}$$

$Q(\theta_i)$ wird als natürlicher Parameter bezeichnet.

2. *Systematische Komponente: Diese besteht aus der systematischen Komponente eines linearen Modells (siehe Definition 2.1.). Explizit sei der lineare Schätzer η definiert durch seine Komponenten:*

$$\eta_i := x_i\beta, \;\; i = 1,\ldots,k$$

Die Parameter β können bekannt sein, üblicherweise werden sie aber unbekannt sein und müssen geschätzt werden.

3. *Link-Funktion: Die dritte Komponente ist eine monotone und differenzierbare Funktion g, welche die zufällige und systematische Komponente miteinander in Verbindung bringt. Sei $\mu_i := \mathbb{E}[Y_i]$, so wird diese Verbindung vermöge $\eta_i = g(\mu_i)$ hergestellt. Liegt der Spezialfall einer exponentiellen Familie* (2.3) *vor, dann wird jene Funktion, welche den Erwartungswert auf den natürliche Parameter transformiert, kanonische Link-Funktion genannt. Konkret gilt in diesem Fall $g(\mu_i) = Q(\theta_i) = x_i\beta$.*

Natürlich stellt sich auch beim GLM wieder die Frage, wie man zu einem Schätzer für β gelangt. Der Standard-Ansatz dafür ist die Maximum-Likelihood-Methode. Sei (x,y) ein Datensatz und es gelte die Notation aus Definition 2.2. Die Likelihood-Funktion ist dann gegeben durch:

$$\tilde{L}(\beta) = \prod_{i=1}^{k} f(y_i;\theta_i,\phi) \tag{2.4}$$

Man möchte $\hat{\beta}$ nun so zu wählen, dass dieser Ausdruck maximiert wird. Oft ist es einfacher, stattdessen die Log-Likelihood-Funktion zu maximieren:

$$L(\beta) = log\tilde{L}(\beta) = \sum_{i=1}^{k} logf(y_i;\theta_i,\phi) \tag{2.5}$$

Da die Logarithmus-Funktion streng monoton wachsend ist, maximiert man gleichzeitig mit der Loglikelihood-Funktion auch die Likelihood-Funktion. Möchte man das Maximum von L bestimmen, so kann man beispielsweise die partiellen Ableitungen von L bezüglich den Komponenten von β bilden. Dieser Ansatz führt auf die Likelihood-Gleichungen. Sei $L_i := logf(y_i;\theta_i,\phi)$, dann sind die Likelihood-Gleichungen gegeben durch

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{k} \frac{\partial L_i}{\partial \beta_j} = 0, \;\; j = 0,\ldots n. \tag{2.6}$$

Um diese Ableitungen zu berechnen, verwendet man die Kettenregel

$$\frac{\partial L_i}{\partial \beta_j} = \frac{\partial L_i}{\partial \theta_i}\frac{\partial \theta_i}{\partial \mu_i}\frac{\partial \mu_i}{\partial \eta_i}\frac{\partial \eta_i}{\partial \beta_j}. \tag{2.7}$$

Zudem sind zwei allgemeine Resultate zur Likelihoodfunktion sehr nützlich:

$$\mathbb{E}\left[\frac{\partial L}{\partial \theta}\right] = 0 \tag{2.8}$$

$$-\mathbb{E}\left[\frac{\partial^2 L}{\partial \theta^2}\right] = \mathbb{E}\left[\frac{\partial L}{\partial \theta}\right]^2 \tag{2.9}$$

Betrachtet man nur eine einzelne Beobachtung und verwendet das erste Resultat (2.8), so erhält man $\mathbb{E}\left[Y_i - b'(\theta_i)\right]/a(\phi) = 0$, woraus folgt, dass

$$\mu_i = \mathbb{E}[Y_i] = b'(\theta_i). \tag{2.10}$$

Das zweite Resultat (2.9) liefert dann

$$\frac{b''(\theta_i)}{a(\phi)} = \mathbb{E}\left[\frac{(Y_i - b'(\theta_i))}{a(\phi)}\right]^2 = \frac{\mathbb{V}[Y_i]}{a(\phi)^2}. \tag{2.11}$$

Aus der Definition von L_i und (2.10) folgt unmittelbar

$$\frac{\partial L_i}{\partial \theta_i} = \frac{y_i - b'(\theta_i)}{a(\phi)} = \frac{y_i - \mu_i}{a(\phi)}. \tag{2.12}$$

Aufgrund von (2.11) gilt

$$\frac{\partial \mu_i}{\partial \theta_i} = b''(\theta_i) = \frac{\mathbb{V}[Y_i]}{a(\phi)}. \tag{2.13}$$

Schließlich berechnet man direkt

$$\frac{\partial \eta_i}{\partial \beta_j} = x_{ij}. \tag{2.14}$$

Setzt man nun diese Ergebnisse zusammen und summiert über i, so erhält man die Likelihood-Gleichungen

$$\sum_{i=1}^{k} \frac{(y_i - \mu_i)x_{ij}}{\mathbb{V}[Y_i]}\frac{\partial \mu_i}{\partial \eta_i} = 0, \quad j = 1, \ldots, n. \tag{2.15}$$

Zunächst scheint es, als ob die zu schätzenden Parameter β hier in den Likelihood-Gleichungen nicht vorkommen würden. Erinnert man sich jedoch an Definition 2.2., so gilt ja $\mu_i = g^{-1}(x_i\beta)$. Es treten also implizit nicht nur die Parameter β auf, sondern auch die Link-Funktion g. Im Allgemeinen müssen die Likelihood-Gleichungen daher auch nicht linear in β sein. Gelöst können sie beispielsweise mit der Newton-Raphson-Methode werden. [3, S. 113f, 130ff, 143ff]

2.1.1 Logistische Regression

Ein wichtiger Spezialfall des Generalized Linear Models ist die logistische Regression. Dabei werden binäre Zielvariablen $Y_1, \ldots, Y_k$ betrachtet. Die logistische Regression kann recht einfach verallgemeinert werden für diskrete Zielvariablen, die nur endlich viele Werte annehmen können, dies wird allerdings an dieser Stelle keine Rolle spielen. [19, S. 119] Im Kontext dieser Arbeit können die Zielvariablen nur zwei Werte annehmen. Entweder $Y_i = 1$, was bedeutet, dass das Unternehmen in Konkurs gegangen ist oder $Y_i = 0$, das Unternehmen besteht noch. Es scheint somit sinnvoll, die Y_i als Bernoulli-verteilte Zufallsvariablen zu modellieren:

$$Y_i \sim Bernoulli(p_i).$$

Die Paramter p_i werden in diesem Kontext als Konkurswahrscheinlichkeiten oder Ausfallswahrscheinlichkeiten bezeichnet. Die Zähldichten dieser Bernoulli-Verteilungen mit Parameter p_i sind gegeben durch

$$f(y_i; p_i) = p_i^{y_i}(1-p_i)^{1-y_i}, \;\; i = 1, \ldots, k.$$

Handelt es sich hierbei um eine exponentielle Familie? Ja, denn

$$f(y_i; p_i) = p_i^{y_i}(1-p_i)^{1-y_i} = (1-p_i)\left(\frac{p_i}{1-p_i}\right)^{y_i} = (1-p_i)\, exp\left(y_i\, log\left(\frac{p_i}{1-p_i}\right)\right).$$

Im Sinne der Notation von (2.3) gilt $A(p_i) = 1-p_i$, $B(y_i) = 1$ und $Q(p_i) = log(p_i/(1-p_i))$. Bekanntermaßen gilt $\mu_i = \mathbb{E}[Y_i] = p_i$. Daraus folgt für die kanonische Link-Funktion

$$g(p_i) = g(\mu_i) = Q(p_i) = log\left(\frac{p_i}{1-p_i}\right).$$

Diese Transformation wird als Logit-Transformation bezeichnet. [3, S. 114f] Fasst man die obigen Erkenntnisse zusammen, erhält man eine formale Definition der logistischen Regression:

Definition 2.3. *Seien $Y_1, \ldots, Y_k$ paarweise unabhängige Zielvariablen und $x_i = (x_{i0}, \ldots, x_{in})^\top$, $i = 1, \ldots, k$ die zugehörigen Einflussvariablen. Die **logistische Regression** ist ein Generalized Linear Model (siehe Definition 2.2), wobei die Zähldichten der Zielvariablen gegeben sind durch*

$$f(y_i; p_i) = (1-p_i)\, exp\left(y_i\, log\left(\frac{p_i}{1-p_i}\right)\right)$$

und die Link-Funktion g die Logit-Transformation ist

$$g(\alpha) = log\left(\frac{\alpha}{1-\alpha}\right).$$

Insbesondere gilt bei der logistischen Regression

$$log\left(\frac{p_i}{1-p_i}\right) = x_i\beta \quad bzw.$$

$$p_i = \frac{exp(x_i\beta)}{1+exp(x_i\beta)} = \frac{1}{1+1/exp(x_i\beta)}.$$

[3, S. 117ff]

Die in Kapitel 2.1 hergeleiteten Likelihood-Gleichungen (2.15) können nun expliziter dargestellt werden. Dazu berechnet man zunächst die Ableitung von μ_i bezüglich η_i:

$$\frac{\partial \mu_i}{\partial \eta_i} = \frac{\partial g^{-1}}{\partial \eta_i}(\eta_i) = \frac{exp(\eta_i)}{(1+exp(\eta_i))^2} \tag{2.16}$$

Für die Bernoulli-verteilten Zielgrößen Y_i gilt zudem

$$\mathbb{V}[Y_i] = p_i(1-p_i) = \mu_i(1-\mu_i) = \frac{exp(\eta_i)}{(1+exp(\eta_i))^2}. \tag{2.17}$$

Setzt man nun (2.16) und (2.17) in (2.15) ein, so erhält man

$$\sum_{i=1}^{k}\left(y_i - \frac{exp(x_i\beta)}{1+exp(x_i\beta)}\right)x_{ij} = 0, \quad j = 1,\ldots,n. \tag{2.18}$$

Offensichtlich handelt es sich hierbei um nichtlineare Gleichungen, die algorithmisch gelöst werden müssen. [3, S. 192f]

Bisher wurden die Matrix x und insbesondere die Merkmale als gegeben vorausgesetzt. In der Praxis handelt es sich hierbei um Daten, die gesammelt werden müssen. Oft stehen zahlreiche Merkmale zur Verfügung. Dies wirft die Frage auf, welche Merkmale aussagekräftig sind, also einen wesentlichen Einfluss auf die Zielvariablen haben und welche nicht. Eine Möglichkeit, relevante Merkmale zu identifizieren, die nicht von der verwendente Machine-Learning-Technik abhängt, wird in Kapitel 3 diskutiert werden. An dieser Stelle soll der Fokus hingegen auf Methoden liegen, wie diese Merkmalsauswahl bei der logistischen Regression durchgeführt werden kann. Dazu gibt es zwei schrittweise Verfahren, welche Vorwärts-Erweiterung beziehungsweise Rückwärts-Reduktion genannt werden sollen. Bei der Vorwärts-Erweiterung testet man zunächst, welches einzelne Merkmal die Zielvariablen am besten beschreibt und verwendet dann ein Modell mit nur diesem Merkmal als Ausgangspunkt. Anschließend fügt man in jedem Schritt jenes Merkmal hinzu, welches die größte Verbesserung des Modells erzielt. Diese Prozedur wird so lange durchgeführt, bis nur noch unwesentliche Verbesserungen des Modells erzielt werden könnten. An diesem Punkt wird dann abgebrochen und alle übrigen Merkmale werden verworfen. Die Rückwärts-Reduktion funktioniert ähnlich, allerdings startet man mit ei-

nem Modell, das sämtliche verfügbaren Merkmale enthält. Nun entfernt man schrittweise jenes Merkmal, welches die geringste Verschlechterung herbeiführt. In diesem Fall wird abgebrochen, sobald das Entfernen eines weiteren Merkmales zu einer signifikanten Verschlechterung des Modells führen würde. [3, S. 209f]

2.1.2 Bayessche Logistische Regression

Die Bayessche Logistische Regression ist eine Erweiterung der herkömmlichen logistischen Regression (siehe Definition 2.3). Die Idee dabei ist es, für das logistische Regressionsmodell eine a priori Verteilung (via der Dichte) der Regressionskoeffizienten β zu spezifizieren:

$$\beta \sim \pi(\beta|\theta). \tag{2.19}$$

Dabei darf $\pi(\beta|\theta)$ im Prinzip eine beliebige Wahrscheinlichkeitsdichte sein. θ kann ein Skalar oder auch auch vektorwertig sein. Der zusätzliche Parameter θ wird als Hyperparameter bezeichnet. Beispielsweise könnte man als priori Verteilung eine multivariate Normalverteilung annehmen:

$$\beta \sim \mathrm{N}(\mu_\beta, \Sigma_\beta)$$

Bei diesem Beispiel sind die Hyperparameter μ_β und Σ_β. Hyperparameter können fest gewählt werden oder man weist ihnen ebenfalls a priori Verteilungen zu. Im zweiten Fall erhält man ein hierarchisches Modell. [44, S. 7ff] In dieser Arbeit sollen hierarchische Modelle allerdings keine Rolle spielen. Die Verteilung von β wird also vorab vollständig spezifiziert und daher kann einfach $\beta \sim \pi(\beta)$ geschrieben werden.

Diese Vorgangsweise bietet die Möglichkeit, zusätzliche Informationen in das Modell zu inkludieren, welche nicht in den eigentlichen Daten enthalten sind. Solche Informationen können zum Beispiel Expertenwissen oder allgemeine Erfahrungswerte sein, aber auch regulatorische Vorschriften. Vor allem wenn nur spärlich Daten zur Verfügung stehen, kann dies sehr nützlich sein. [44, S. 4]

Um Rückschlüsse aus dem Modell ziehen zu können und Schätzer für die Regressionskoeffizienten zu erhalten, benötigt man die a posteriori Dichte der Regressionskoeffizienten $\pi(\beta|y)$. Diese ergibt sich folgendermaßen aufgrund des Satzes von Bayes (siehe zum Beispiel [14, S. 7f]):

$$\pi(\beta|y) = \frac{\pi(y|\beta)\pi(\beta)}{\int \pi(y|\beta)\pi(\beta)\mathrm{d}\beta}. \tag{2.20}$$

Dabei ist $\pi(y|\beta)$ die Likelihood (vgl. (2.4), aber Achtung: formal handelt es sich um unterschiedliche Ausdrücke, da ja die Regressionskoeffizienten β in diesem Kapitel als Zufallsvariablen behandelt werden).

Im Allgemeinen wird die a posteriori Dichte (2.20) keine geschlossene Form besitzen beziehungsweise sich nur äußerst schwierig analytisch berechnen lassen. Um trotzdem mit der a posteriori Verteilung arbeiten zu können, bedient man sich approximativer Methoden. Eine Möglichkeit sind die sogenannten *Markov Chain Monte Carlo* (MCMC) Methoden. [29, S. 7] Eine weitere Möglichkeit besteht in *Integrated Nested Laplace Approximations* (INLA). Die INLA-Methode erlaubt nur Normalverteilungen als priori Verteilungen der Regressionskoeffizienten. [44, S. 4ff] Im Folgenden werden die MCMC-Methoden etwas eingehender betrachtet.

Markov Chain Monte Carlo ist ein Überbegriff für verschiedene Algorithmen, die auf dem gleichen Konzept basieren. Die Grundidee ist recht einfach: wenn man eine hinreichend große Stichprobe aus einer Verteilung mit Dichte $\pi(\beta|y)$ hat, dann kann man diese gut approximieren. Der schwierige Teil ist es dabei, diese Stichprobe zu generieren. Bei den MCMC-Methoden werden Werte für β aus approximierenden Verteilungen gezogen und diese Werte werden dann aktualisiert, um sich der gewünschten Verteilung mit Dichte $\pi(\beta|y)$ zu nähern. Das Ziehen erfolgt dabei schrittweise, wobei die Verteilung, aus welcher gezogen wird, immer nur vom letzten gezogenen Wert abhängt. Dementsprechend erhalten wir Werte aus einer diskreten Markovkette. Die Verteilung der zuletzt gezogenen Stichprobe (also die Verteilung der N zuletzt gezogenen Werten) ist dann die aktuelle Approximation von $\pi(\beta|y)$. Die einzelnen Algorithmen sind so konzipiert, dass diese Approximationen gegen $\pi(\beta|y)$ konvergieren. Entscheidend dafür ist, dass eine Markovkette erzeugt wird, deren stationäre Verteilung die Dichte $\pi(\beta|y)$ hat. Dann muss man so lange ziehen, bis die Stichprobenverteilung eine hinreichend gute Approximation der stationären Verteilung darstellt. Diese Konvergenz festzustellen ist oft der kritischste Punkt bei der MCMC-Simulation. [14, S. 275ff]

Das allgemeine Konzept der MCMC-Methoden soll nun anhand eines konkreten Algorithmus verdeutlicht werden. Der nachfolgende Algorithmus wird als *Metropolis*-Algorithmus bezeichnet:

1. Ziehe einen Startwert β^0 mit $\pi(\beta^0|y) > 0$ von einer Startverteilung mit Dichte $\pi_0(\beta)$.

2. Ziehe im t-ten Schritt einen Kandidaten β^* für den nächsten Wert der Markovkette von einer Sprungverteilung $J_t(\beta^*|\beta^{t-1})$ (Achtung: Hier handelt es sich nicht um die Übergangswahrscheinlichkeiten).

3. Kalkuliere das Verhältnis zwischen den Dichten:

$$r = \frac{\pi(\beta^*|y)}{\pi(\beta^{t-1}|y)}. \tag{2.21}$$

4. Setze

$$\beta^t = \begin{cases} \beta^* & \text{mit Wahrscheinlichkeit min(r,1)} \\ \beta^{t-1} & \text{mit Wahrscheinlichkeit 1 - min(r,1).} \end{cases}$$

5. Springe zu Schritt 2 ($t = t+1$).

Beim Metropolis-Algorithmus muss die Sprungverteilung symmetrisch sein, das heißt $J_t(\beta_a|\beta_b) = J_t(\beta_b|\beta_a)$ für alle β_a, β_b und t. Diese Bedingung kann fallengelassen werden, wenn man r aus Gleichung (2.21) anders definiert:

$$r = \frac{\frac{\pi(\beta^*|y)}{J_t(\beta^*|\beta^{t-1})}}{\frac{\pi(\beta^{t-1}|y)}{J_t(\beta^{t-1}|\beta^*)}}. \tag{2.22}$$

Bei dieser Verallgemeinerung spricht man vom *Metropolis-Hastings*-Algorithmus. Tatsächlich besitzen die im Metropolis- beziehungsweise im Metropolis-Hastings-Algorithmus konstruierten Markovketten die gewünschte stationäre Verteilung mit Dichte $\pi(\beta|y)$ (siehe [14, S. 279f]). Es stellt sich also nur noch die Frage, wann (also nach wie vielen Iterationen) die simulierten Werte hinreichend repräsentativ für die Zielverteilung sind, also wann die stationäre Verteilung annähernd erreicht ist. Selbst wenn diese Konvergenz erreicht wurde, sind die ersten simulierten Werte der Markovkette oft nicht sehr repräsentativ. Als Daumenregel verwirft man daher meist die erste Hälfte der simulierten Werte. Dies wird als „Aufwärmen“ der Markovkette bezeichnet. Um die Konvergenz feststellen zu können, erzeugt man üblicherweise zumindest zwei Markovketten, eventuell auch mehr. Dann prüft man, ob die Markovketten individuell Stationärität erreicht haben und der gleichen Verteilung folgen. [14, S. 278ff]

2.2 Artificial Neural Networks

Bei Artificial Neural Networks (ANN) handelt es sich um einen Übergriff für mehrere statistische Modelle. Diese können als gerichteter und gewichteter Graph beschrieben und visualisiert werden. Dementsprechend werden in weiterer Folge die elementaren Komponenten der ANN als Knoten beziehungsweise Kanten bezeichnet. Konkret werden in diesem Abschnitt die sogenannten Multilayer Perceptrons (MLP) behandelt werden. Diese Modelle sind äußerst flexibel und können daher für viele unterschiedliche Verwendungszwecke herangezogen werden. Abbildung 1 zeigt ein allgemein gehaltenes Beispiel eines Multilayer Perceptrons.

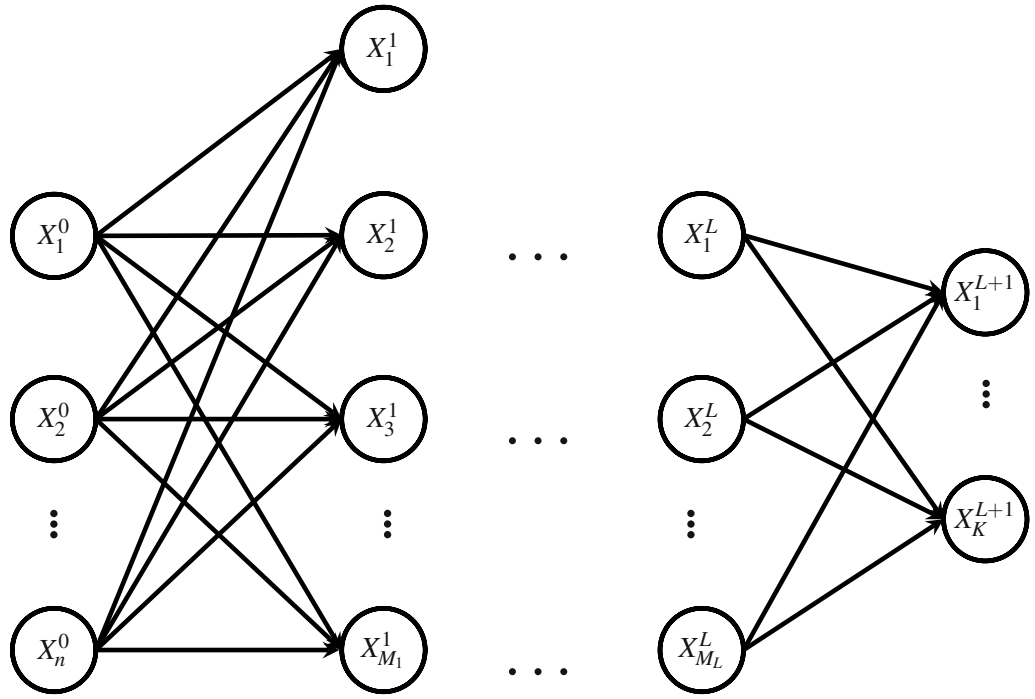

Abbildung 1: Ein Multilayer Perceptron mit L Hidden-Layern.
Quelle: Eigene Darstellung basierend auf [20, Abb. 4.1].

In der obigen Abbildung sind die wesentlichen Komponenten eines MLPs bereits gut ersichtlich. Dieser besteht aus mehreren Layern, von welchen zwei ausgezeichnet sind: der Input-Layer und der Output-Layer. Daten werden in Vorwärtsrichtung (in Kantenrichtung) durch das Modell geschleust, wobei die Knoten des Input-Layers die Eingabewerte und die Knoten des Output-Layers die Ausgabewerte des Modells darstellen. Es wurde bewusst entschieden, dieses Beispiel vor der formalen Definition zu liefern, da die graphische Darstellung wesentlich greifbarer ist. Die folgende Definition bezieht sich ausschließlich auf die Anwendung zur Regression beziehungsweise Klassifizierung.

Definition 2.4. *Ein **Multilayer Perceptron** ist durch folgende Eigenschaften definiert:*

1. *Das Modell besteht aus mindestens drei sogenannten Layern: am Anfang steht der Input-Layer, der für jede Einflussvariable der Daten einen Knoten enthält. Danach folgen L Hidden-Layer, $L > 0$, welche geordnet sind. Jeder Hidden-Layer l enthält wiederum eine beliebige Anzahl M_l von Knoten. Den Abschluss bildet stets der Output-Layer. Die Anzahl der Knoten des Output-Layers entspricht der Dimension der Zielvariablen.*[2]

[2]Bei Klassifizierungsproblemen codiert man die einzelnen Klassen als 0-1 Variablen. Entsprechend enthält der Output-Layer bei K Klassen auch K Knoten.

2. *Von den Knoten des Input-Layers oder eines Hidden-Layers führt eine gerichtete Kante zu allen Knoten des nachfolgenden Layers. Sonst existieren keine weiteren Kanten. Der Graph des Modells ist also insbesondere azyklisch.*

3. *Jedem Knoten wird ein Wert zugeordnet. Die Werte der Knoten X_i^0, $i = 1,\dots,n$ des Input-Layers entsprechen den Werten der Einflussvariablen. Die Werte der Knoten des l-ten Layers X_m^l werden mittels der folgenden Vorschrift gebildet:*

$$X_m^l = \sigma\left(w_{0m}^l + \sum_{i=1}^{M_{l-1}} w_{im}^l X_i^{l-1}\right) \tag{2.23}$$

Die Funktion σ wird als Aktivierungsfunktion bezeichnet und die Skalare w_{im}^l nennt man Gewichte. Es wird gefordert, dass σ sowohl nicht-linear als auch glatt (überall differenzierbar) ist. Die Werte der Knoten des Output-Layers X_k^{L+1}, $k = 1,\dots,K$, werden entweder auch vermöge (2.23) gebildet (setze $l = L+1$), oder es wird die Aktivierungsfunktion weggelassen:

$$X_k^{L+1} = w_{0k}^{L+1} + \sum_{i=1}^{M_L} w_{ik}^{L+1} X_i^L.$$

[19, S. 392f] [20, S. 178ff] [47, S. 72ff]

Eine typische Wahl für die Aktivierungsfunktion ist die Sigmoid-Funktion:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Offensichtlich ist die Sigmoid-Funktion nicht-linear und glatt, erfüllt also alle Voraussetzungen einer Aktivierungsfunktion. Manchmal transformiert man die Outputs des Modells auch noch mittels einer weiteren Funktion. Bei Regressionsproblemen ist dies eher unüblich, bei Klassifizierungsproblemen bietet sich hingegen die sogenannte Softmax-Funktion an:

$$g_k(X^{L+1}) = \frac{exp(X_k^{L+1})}{\sum_{i=1}^{K} exp(X_i^{L+1})}, \quad k = 1,\dots,K.$$

Aufgrund dieser Transformation sind die Outputs des Modells positiv und ergeben in Summe 1, was die Interpretation als Wahrscheinlichkeiten ermöglicht.
Artificial Neural Networks sind, wie der Name schon andeutet, ursprünglich als Modell für das menschliche Gehirn entwickelt worden. Jeder Knoten beschreibt dabei ein Neuron und die Kanten zwischen den Knoten stellen die Verbindungen zwischen den Neuronen, besthend aus Axon, Synapsyen und Dendriten, dar. Später hat man dann auch den Nutzen für Regressions- und Klassifizierungsprobleme erkannt. Die Relation zum menschlichen Gehirn und der zunächst komplex wirkende Modellaufbau erwecken den

Eindruck, dass es sich bei ANNs um sehr spezielle Modelle handelt. Im Grunde sind diese jedoch einfach nicht-lineare statistische Modelle. [19, S. 394f] Allerdings weisen sie einige günstige Eigenschaften auf: Erstens passen die Modelle sich selbst an die Daten an. Insbesondere müssen keine Annahmen über die Verteilung der zugrundeliegenden Daten oder über funktionale Abhängigkeiten zwischen den Daten gemacht werden. Zweitens sind sie universelle Funktionenschätzer, das heißt sie können stetige beziehungsweise Borel-messbare Funktionen mit beliebiger Genauigkeit approximieren. Drittens, wie bereits erwähnt, handelt es sich um nicht-lineare Modelle. Dadurch ist genügend Flexibilität gegeben, um auch hoch komplexe Beziehungen zwischen den Daten modellieren zu können. Dies ist für praktische Anwendung sehr wichtig. Der vierte und letzte Punkt bezieht sich speziell auf die Klassifizierung. ANNs können a posteriori Wahrscheinlichkeiten schätzen, wodurch die Einteilung in Klassen und statistische Analysen ermöglicht werden. [49, S. 451]

An dieser Stelle soll näher darauf eingegangen werden, in welchem Sinne ANNs universelle Schätzer sind. Dazu werden zunächst einige grundlegende Definitionen benötigt.

Definition 2.5. *Sei* $\psi : \mathbb{R} \to [0,1]$ *eine nicht-fallende Funktion mit* $\lim_{x\to\infty} \psi(x) = 1$ *und* $\lim_{x\to-\infty} \psi(x) = 0$. *Dann definiert man*

$$\Sigma^n(\psi) := \left\{ f : \mathbb{R}^n \to \mathbb{R}, f(x) = \sum_{i=1}^{M} \beta_i \psi \left(w_0 + \sum_{j=1}^{n} w_j x_j \right) \right\}, \tag{2.24}$$

wobei $M \in \mathbb{N}$, $\beta_i \in \mathbb{R}$, $i = 1,\ldots,M$, *und* $w_j \in \mathbb{R}$, $j = 0,\ldots,n$.

Wenn ψ eine Aktivierungsfunktion ist, die den Anforderungen von Definition 2.5 genügt (beispielsweise die Sigmoid-Funktion), dann besteht $\Sigma^n(\psi)$ genau aus den Multilayer Perceptrons mit einem einzelnen Hidden-Layer, welche n Input-Knoten und einen Output-Knoten enthalten. Die folgenden zwei Begriffsbildungen werden für das nachfolgende Theorem benötigt.

Definition 2.6. *Sei* $C(\mathbb{R}^n,\mathbb{R})$ *die Menge der stetigen Funktionen von* $\mathbb{R}^n$ *nach* $\mathbb{R}$. *Eine Teilmenge* S *von* $C(\mathbb{R}^n,\mathbb{R})$) *heißt* ***gleichmäßig dicht auf Kompakta*** *in* $C(\mathbb{R}^n,\mathbb{R})$, *wenn für jede kompakte Teilmenge* $K \subset \mathbb{R}^n$ S ρ_K*-dicht in* $C(\mathbb{R}^n,\mathbb{R})$ *liegt. Dabei ist* $\rho_K(f,g) := \sup_{x\in K} |f(x) - g(x)|$, $f,g \in C(\mathbb{R}^n,\mathbb{R})$.

Definition 2.7. *Sei* $B(\mathbb{R}^n)$ *die Borel-Sigmaalgebra auf* $\mathbb{R}^n$ *und* $\mathbb{P}$ *ein Wahrscheinlichkeitsmaß auf dem Messraum* $(\mathbb{R}^n, B(\mathbb{R}^n))$. *Zudem sei* B^n *die Menge der Borel-messbaren Funktionen von* $\mathbb{R}^n$ *nach* $\mathbb{R}$. *Die Metrik* $\rho^{\mathbb{P}} : B^r \times B^r \to \mathbb{R}^+$ *definiert man dann als*

$$\rho^{\mathbb{P}}(f,g) := \inf\{\varepsilon > 0 : \mathbb{P}\{x \in \mathbb{R}^n : |f(x) - g(x)| > \varepsilon\} < \varepsilon\}.$$

Theorem 2.8. *Für ein beliebiges Wahrscheinlichkeitsmaß $\mathbb{P}$ auf $(\mathbb{R}^n, B(\mathbb{R}^n))$ ist $\Sigma^n(\psi)$ gleichmäßig dicht auf Kompakta in $C(\mathbb{R}^n, \mathbb{R})$ und $\rho^{\mathbb{P}}$-dicht in B^n.*

Für den Beweis von Theorem 2.8 sei auf [21] verwiesen. Das Theorem kann so interpretiert werden, dass MLPs bereits mit einem Hidden-Layer jede stetige Funktion gleichmäßig auf beliebigen Kompakta approximieren können. Außerdem kann jede Borel-meßbare Funktion beliebig genau in der $\rho^{\mathbb{P}}$-Metrik approximiert werden. Dies ist gemeint, wenn man davon spricht, dass MLPs universelle Schätzer sind. Theorem 2.8 kann auch für Funktionen von $\mathbb{R}^n$ nach $\mathbb{R}^m$ verallgemeinert werden, worauf hier allerdings nicht näher eingegangen wird. [21, S. 359ff]

Hat man nun konkrete Daten (x, y) zur Hand, so steht man vor zwei Fragestellungen:

- Welches MLP-Modell bildet den funktionalen Zusammenhang zwischen den Daten gut ab? Beziehungsweise präziser formuliert: wie viele Hidden-Layer und wie viele Knoten pro Layer sind optimal?
- Wenn man die Spezifikationen des Modells einmal festgelegt hat, wie kalibriert man dieses dann?

Die zweite Frage ist einfacher zu beantworten als die erste. Um das gewählte Modell zu kalibrieren, müssen die Gewichte aus Gleichung (2.23) bestimmt werden. Die Schwierigkeit besteht darin, dass ein MLP typischerweise zahlreiche Knoten enthält und daher sehr viele Gewichte bestimmt werden müssen. Der bekannteste Algorithmus zur Lösung dieser Aufgabe ist der sogenannte Backpropagation-Algorithmus. Dieser soll nun hergeleitet werden. Ausgangspunkt dieser Herleitung ist ein Maß für die Qualität der Regression respektive der Klassifizierung. Für die Output-Knoten kennt man die erwarteten Ergebnisse, nämlich den Vektor y_i für den i-ten Datensatz aus (x, y). Entsprechend kann man den Fehler des Modells für den i-ten Datensatz $E(i)$ als Summe der Fehler der einzelnen Outputknoten $E_k(i)$ definieren:

$$E(i) := \sum_{k=1}^{K} E_k(i) \tag{2.25}$$

Summiert man über alle Datensätze, so erhält man den Gesamtfehler des Modells:

$$E := \sum_{i=1}^{n} E(i) \tag{2.26}$$

Die einzelnen Fehler $E_k(i)$ können wiederum unterschiedlich definiert werden. Eine häufig genutzte Möglichkeit ist die bereits bei der linearen Regression erwähnte quadratische Fehlerfunktion

$$E_k(i) = (y_{ik} - X_k^{L+1}(i))^2, \tag{2.27}$$

wobei $X_k^{L+1}(i)$ den vom Modell geschätzen Wert für y_{ik} bezeichnet. Die Werte $X_k^{L+1}(i)$ stehen a priori natürlich nicht zur Verfügung. Daher wird zunächst ein Vorwärtsdurchlauf des Modells durchgeführt, um die Schätzer $X_k^{L+1}(i)$ zu erhalten. Für den initialen Vorwärtsdurchlauf des Modells werden die Gewichte zumeist zufällig gewählt. Nach diesem initialen Durchlauf möchte man die Gewichte anpassen, sodass die Fehler kleiner werden. Dazu bedient man sich eines Gradientenverfahrens. Das Update der Gewichte nach einer Iteration wird dabei folgendermaßen berechnet:

$$\Delta W = -\eta \nabla E, \tag{2.28}$$

wobei W hier einen Vektor bezeichnet, der sämtliche Gewichte des Modells enthält. Bei η handelt es sich andererseits um einen Skalar, welcher als Lernfaktor bezeichnet wird. Dieser skaliert die Anpassungen pro Iteration. Entsprechend ergibt sich für ein einzelnes Gewicht

$$\Delta w_{jm}^l = -\eta \frac{\partial E}{\partial w_{jm}^l} = \sum_{i=1}^{n} -\eta \frac{\partial E(i)}{\partial w_{jm}^l}. \tag{2.29}$$

Die Ableitungen auf der rechten Seite von Gleichung (2.29) können mit Hilfe der Kettenregel bestimmt werden. Dafür definiert man zunächst folgende Hilfsvariable

$$z_m^l(i) := w_{0m}^l + \sum_{r=1}^{M_{l-1}} w_{rm}^l X_r^{l-1}(i). \tag{2.30}$$

Dies entspricht genau dem Input in die Aktivierungsfunktion für den Knoten X_m^l beim i-ten Datensatz im Sinne von Definition 2.4. Dann gilt

$$\frac{\partial E(i)}{\partial w_{jm}^l} = \frac{\partial E(i)}{\partial z_m^l(i)} \frac{\partial z_m^l(i)}{\partial w_{jm}^l} \tag{2.31}$$

Aus (2.30) folgt für den zweiten Faktor natürlich sofort

$$\frac{\partial z_m^l(i)}{\partial w_{jm}^l} = \frac{\partial}{\partial w_{jm}^l} \left(w_{0m}^l + \sum_{r=1}^{M_{l-1}} w_{rm}^l X_r^{l-1}(i) \right) = X_j^{l-1}(i). \tag{2.32}$$

Auf den ersten Faktor wendet man hingegen erneut die Kettenregel an

$$\frac{\partial E(i)}{\partial z_m^l(i)} = \frac{\partial E(i)}{\partial X_m^l(i)} \frac{\partial X_m^l(i)}{\partial z_m^l(i)}. \tag{2.33}$$

Für den zweiten Faktor von (2.33) ergibt sich

$$\frac{\partial X_m^l(i)}{\partial z_m^l(i)} = \frac{\partial}{\partial z_m^l(i)} \sigma(z_m^l(i)) = \sigma'(z_m^l(i)). \tag{2.34}$$

Die Differenzierbarkeit von σ ist dabei per Definition gegeben. Zur Bestimmung des ersten Faktors von (2.33) benötigt man eine Fallunterscheidung:

1. $X_m^l(i)$ ist ein Output-Knoten, also $l = L+1$.
2. $X_m^l(i)$ ist Knoten eines Hidden-Layers.

Im ersten Fall ist die Ableitung einfach

$$\frac{\partial E(i)}{\partial X_m^{L+1}(i)} = \frac{\partial}{\partial X_m^{L+1}(i)} \sum_{k=1}^{K} E_k(i) = \frac{\partial E_m(i)}{\partial X_m^{L+1}(i)}, \tag{2.35}$$

unter der Voraussetzung, dass der Fehler des k-ten Output-Knoten $E_k(i)$ nicht von anderen Output-Werten abhängig ist, außer $X_k^{L+1}(i)$, für alle $k = 1,\dots,K$. Für die quadratische Fehlerfunktion ist dies gegeben und es gilt konkret

$$\frac{\partial E(i)}{\partial X_m^{L+1}(i)} = -2\left(y_{ik} - X_m^{L+1}(i)\right). \tag{2.36}$$

Im zweiten Fall kann die Ableitung nur indirekt berechnet werden:

$$\frac{\partial E(i)}{\partial X_m^l(i)} = \sum_{s=1}^{M_{l+1}} \frac{\partial E(i)}{\partial z_s^{l+1}(i)} \frac{\partial z_s^{l+1}(i)}{\partial X_m^l(i)} = \sum_{s=1}^{M_{l+1}} \frac{\partial E(i)}{\partial z_s^{l+1}(i)} w_{ms}^{l+1}. \tag{2.37}$$

Aufgrund von (2.33) hat man somit die Ableitung $\frac{\partial E(i)}{\partial X_m^l(i)}$ auf die Ableitungen $\frac{\partial E(i)}{\partial X_s^{l+1}(i)}$, $s = 1,\dots,M_{l+1}$ zurückgeführt. Man erhält somit eine Rekursion, deren Startwerte sich aus dem ersten Fall ergeben. Die Grundidee des Backpropagation-Algorithmus ist also, dass man die Fehler an den Output-Knoten einfach bestimmten kann und die Fehler der Knoten in den Hidden-Layern als gewichtete Summe ihrer Nachfolgeknoten definiert. Betrachtet man Gleicheung (2.29) erneut, so erkennt man, dass eine Anpassung der Gewichte aufgrund der Fehler sämtlicher Datensätze vorgenommen wird. Diese Vorgehensweise wird auch als offline-Version oder batch-Version des Algorithmus bezeichnet. Im Gegensatz dazu werden bei der online-Version die Gewichte nach jedem Datensatz angepasst. Die Anpassungen erfolgen dabei vermöge

$$\Delta_i w_{jm}^l = -\eta \frac{\partial E(i)}{\partial z_m^l(i)} X_j^{l-1}(i). \tag{2.38}$$

Die Online-Version benötigt weniger Speicherplatz und hat meistens einen geringeren Trainingsaufwand, bis das Modell zufriedenstellend kalibriert ist. [47, S. 105ff]

Die Kalibrierung des Modells ist somit geklärt. Die interessantere Frage ist jedoch die Wahl des Modells selbst. Diese Fragestellung beinhaltet zwei Aspekte: einerseits müssen die zu verwendenden Einflussvariablen (Input-Knoten) ausgewählt werden, andererseits

muss die Anzahl der Hidden-Layer und die Anzahl der Knoten pro Layer festgelegt werden. Für eine modellunabhängige Methode zur Auswahl der Einflussvariablen sei erneut auf Kapitel 3 verwiesen.
Beschränkt man sich auf die Auswahl der Input-Knoten, so können die in Kapitel 2.1.1 vorgestellten Methoden, nämlich die Vorwärts-Erweiterung und die Rückwärts-Reduktion, angewandt werden. Es ist allerdings zu bedenken, dass die Modellkalibrierung jedes Mal wiederholt werden muss, wenn ein Knoten hinzugefügt beziehungsweise entfernt wurde. [15, S. 253f] Bei MLPs kann dies einen beträchtlichen Rechenaufwand verursachen.
Alternativ dazu wurden zahlreiche Maße entwickelt, die die Bedeutsamkeit der Einflussgrößen beziehungsweise den Beitrag zum Output messen und somit vergleichbar machen. Diese Maße werden als *Saliency*-Maße bezeichnet. [49, S. 457]
Ein bekanntes Saliency-Maß ist jenes von Garson. Dieses gibt den relativen Einfluss der Input-Variablen auf einen Output-Knoten an. Für die Berechnung werden die (absoluten) Gewichte der Verbindungen pfadweise multipliziert. Diese Produkte werden anschließend normiert und dann wird für jeden Input-Knoten die Summe der normierten Produkte über alle Pfade gebildet, die von diesem ausgehend zum Output-Knoten führen. Schließlich werden auch diese Summen noch normiert. [17, S. 150f] Allerdings hat Garson's Maß den Nachteil, dass es nur für MLPs mit einem Hidden-Layer verwendet werden kann. [45, S. 2f]
Eine Methode, die diesen Nachteil nicht hat, wurde von Olden und Jackson unter dem Namen *Randomization*-Test entwickelt. Dieser Test wird mittels der folgenden Prozedur durchgeführt:

1. Kalibriere mehrere MLPs auf den ursprünglichen Daten (x, y) mit verschiedenen (zufälligen) initialen Gewichten.
2. Wähle das Modell mit der besten Performance und notiere die Startgewichte. Berechne und notiere die folgenden Werte:
   (a) Das Produkt der Gewichte, die zum gleichen Pfad von einem Input-Knoten zu einem Output-Knoten gehören (für alle solche Pfade).
   (b) Die Summe der Produkte aus (a) über alle Pfade, die beim gleichen Input-Knoten beginnen (für alle Input-Knoten) und zum gleichen Output-Knoten führen.
3. Permutiere y, um neue Daten $(x, \tilde{y})$ zu erzeugen.
4. Kalibriere einen MLP auf $(x, \tilde{y})$ mit den gespeicherten Startgewichten.
5. Wiederhole die Schritte 3. und 4. hinreichend oft (z.B. 1000 Mal) und notiere jeweils die Werte aus 2.(a) und 2.(b).

Aus den Werten die in 2.(a) und 2.(b) berechnet wurden, kann die statistische Signifikanz der Pfade und der Input-Knoten abgeleitet werden. Dazu vergleicht man die Werte die „zufällig“, also mittels permutierten Daten, erzeugt wurden mit dem beobachten Wert, welcher mit den ursprünglichen Daten bestimmt wurde. Ausschlaggebend ist der Anteil der „zufälligen“ Werte inklusive des beobachteten Wertes, die gleich dem beobachteten Wert oder extremer als der beobachtete Wert sind. Ist dieser Anteil klein, so deutet dies auf einen signifikanten Pfad beziehungsweise einen signifikanten Input-Knoten hin. Als Grenze könnte beispielsweise kleiner 0.05 oder kleiner 0.10 festgelegt werden. [32, S. 143ff]

Zur Veranschaulichung von Garson's Maß und des Ranodmization-Tests betrachtet man am besten ein Beispiel. Es sei folgender MLP gegeben (2):

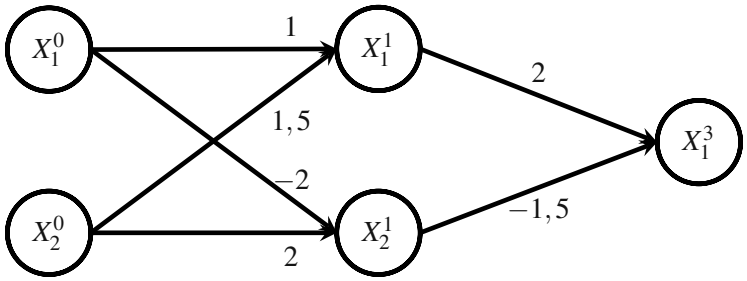

Abbildung 2: Ein MLP mit einem Hidden-Layer. Die Gewichte der Knoten sind bei den Kanten notiert.
Quelle: Eigene Darstellung.

Die nachfolgende Tabelle 1 zeigt die Berechnung von Garson's Maß (relativer Einfluss).

	Hidden 1	Hidden 2	Summe	Relativer Einfluss
Input 1	0,4	0,5	0,9	0,45
Input 2	0,6	0,5	1,1	0,55

Tabelle 1: Garson's Maß: In den Spalten Hidden 1 und Hidden 2 findet man die pfadweisen Produkte der Gewichte, spaltenweise normiert. Die Spalte Relativer Einfluss entsteht durch Normierung der Spalte Summe. Quelle: Eigene Darstellung.

Man beachte, dass bei Garson's Maß die Beträge der Gewichte in die Formel eingehen, zwischen negativen und positiven Gewichten wird also nicht unterschieden. Im Unterschied dazu werden beim Randomization-Test die tatsächlichen Werte der Gewichte verwendet. Tabelle 2 zeigt die Berechnung der Werte aus Schritt 2.(a) und 2.(b) des Algorithmus:

	Hidden 1	Hidden 2	Summe
Input 1	2	3	5
Input 2	3	-3	0

Tabelle 2: Randomization-Test: In den Spalten Hidden 1 und Hidden 2 findet man die in 2.(a) berechneten Werte. In der Spalte Summe findet man die in 2.(b) berechneten Werte. Quelle: Eigene Darstellung.

Betrachetet man nun den relativen Einfluss in Tabelle 2, so erkennt man, dass Garson's Maß Input 2 als wichtiger einschätzt. In Tabelle 2 sieht man jedoch, dass sich die Pfade, die beim zweiten Input-Knoten beginnen, sogar gänzlich aufheben. Dies ist zumindest ein erster Hinweis darauf, dass Input 2 beim Randomization-Test als unwesentlich eingestuft werden könnte. Offensichtlich ergibt sich dieser Unterschied daraus, dass bei Garson's Maß die absoluten Gewichte verwendet werden und beim Randomization-Test nicht.

Wie bereits erwähnt, kann der Randomization-Test auch für MLPs mit mehreren Hidden-Layern eingesetzt werden. Ein weiterer Vorteil dieser Methode ist, dass nicht nur ganze Input-Knoten bewerten kann, sondern auch einzelne Pfade. Dies kann helfen, ein möglichst einfaches und kleines Modell zu entwickeln. Zudem deuten Experimente darauf hin, dass der Randomization-Test bessere Ergebnisse als Garson's Maß liefert. [33, S. 392f] Es sei allerdings auch hier darauf hingewiesen, dass die Kalibrierung zahlreicher MLPs einen beachtlichen Aufwand verursachen kann.

Man hat also einige Methoden zur Verfügung, die Einflussvariablen zu selektieren. Wie wählt man allerdings die Anzahl der Knoten in den Hidden-Layern und die Anzahl der Hidden-Layer (auch *Architektur* des Modells genannt)? Zumindest theoretisch wird der Modellfehler auf den Trainingsdaten nicht größer, wenn man die Anzahl der Knoten erhöht. Es können ja einfach alle Gewichte der neuen Knoten auf 0 gestetzt werden, falls das kleinere Modell doch besser ist.

Reduziert man hingegen die Anzahl der Knoten, so kann dies nicht nur die Interpretation vereinfachen, sondern oft auch die Performance auf ungesehenen Daten verbessern. Die Ursache dafür ist, dass MLPs dazu neigen, die Trainingsdaten sehr gut zu modellieren. Dies ist nicht verwunderlich, da es sich ja um universelle Schätzer handelt (siehe Theorem 2.8). Dadurch kann es allerdings auch zu overfitting kommen und die Performance auf ungesehenen Daten leidet. [49, S. 455]

Tatsächlich wird bei der Wahl der Architektur oft auf *brute-force* zurückgegriffen, wobei

hier verschiedene Architekturen ausprobiert und anhand ihrer Fehler auf (ungesehenen) Daten verglichen werden. Es gibt allerdings auch methodische Vorgangsweisen. Eine davon besteht darin, zunächst mit einem zu großen Modell zu starten. [3] Danach entfernt man Gewichte oder sogar ganze Knoten, die unwesentlich sind. Dieser Schritt wird als *Pruning* bezeichnet. Inzwischen stehen zahlreiche Pruning-Algorithmen zur Verfügung. Viele davon basieren auf dem älteren Algorithmus *Optimal Brain Damage* (OBD), welcher im Folgenden als Beispiel für einen Pruning-Algorithmus skizziert wird. [5, S. 106ff]

Der OBD-Algorithmus basiert auf der Taylorreihenentwicklung der Fehlerfunktion E. Man trifft dabei drei vereinfachende Annahmen:

1. Die Hesse-Matrix der Fehlerfunktion E wird durch eine Diagonalmatrix approximiert.
2. Das Verfahren wird nur eingesetzt, nachdem des Modell erfolgreich kalibriert wurde. Insbesondere hat man also ein (lokales) Minimum des Fehlers erreicht.
3. Die Fehlerfunktion ist in der Nähe des Minimus lokal quadratisch.

Diese Annahmen haben zur Folge, dass nur die Diagonaleinträge der Hesse-Matrix von E berechnet werden müssen. Heuristisch besteht der OBD-Algorithmus aus den folgenden Schritten:

1. Wähle eine angemessene, eher zu große, Netzwerkstruktur.
2. Kalibriere das Netzwerk.
3. Berechne die Diagonaleinträge der Hesse-Matrix: $\frac{\partial^2 E}{\partial (w^l_{jm})^2}$.
4. Berechne die *Saliency* s^l_{jm} für jedes der Gewichte:

$$s^l_{jm} := \frac{\partial^2 E}{\partial (w^l_{jm})^2} \frac{(w^l_{jm})^2}{2}. \tag{2.39}$$

5. Sortiere die Gewichte nach ihrer Saliency. Setze Gewichte mit dem geringsten Wert auf 0.
6. Springe zu Schritt 2.

[47, S. 320ff]

Eine Verallgemeinerung des OBD-Algorithmus ist Optimal Brain Surgeon (OBS). Dieser Algorithmus basiert ebenfalls auf der Taylorreihenentwicklung der Fehlerfunktion E.

[3] Die initiale Größe wird oft mit Augenmaß festgelegt. Es gibt aber auch Arbeiten dazu, wie groß ein Modell mindestens sein muss (siehe z.B. [22]).

Im Gegensatz zum OBD-Algorithmus lässt man allerdings die erste Annahme fallen, die Hesse-Matrix von E muss also nicht annähernd diagonal sein. Die zweite und dritte Annahme werden hingegen weiterhin benötigt. Der OBS-Algorithmus besteht aus den folgenden Schritten:

1. Wähle eine angemessene, eher zu große, Netzwerkstruktur.
2. Kalibriere das Netzwerk.
3. Berechne die Inverse der Hesse-Matrix H von E.
4. Berechne die *Saliency* s^l_{jm} für jedes der Gewichte w^l_{jm}. Sei dafür q der Index von w^l_{jm}, wenn man die Gewichte als einen (langen) Vektor W auffasst. Dann gilt:

$$s^l_{jm} := \frac{1}{2}\frac{W_q^2}{[H^{-1}]_{qq}}. \tag{2.40}$$

   Hat ein Gewicht eine Saliency, die viel kleiner als der Gesamtfehler E ist, so gehe zu Schritt 5. Sonst gehe zu Schritt 6.
5. Passe die Gewichte mittels der folgenden Formel an:

$$\delta W = -\frac{W_q}{[H^{-1}]_{qq}} H^{-1} e_q. \tag{2.41}$$

   Dabei ist q der Index des in Schritt 4 gefundenen Gewichts. Dieses wird somit gelöscht und gleichzeitig werden die anderen Gewichte angepasst. Gehe anschließend zu Schritt 3.
6. Es können keine weiteren Gewichte entfernt werden, ohne den Fehler stark zu erhöhen. Eventuell kann das Netzwerk nun erneut kalibriert werden.

Unter der Annahme, dass sich die Fehlerfunktion lokal quadratisch verhält, liefert der OBS-Algorithmus mindestens so gute Ergebnisse wie der OBD-Algorithmus. In gewissen Konstellationen löscht der OBD-Algorithmus das falsche Gewicht und der OBS-Algorithmus nicht, wodurch dieser dann echt bessere Ergebnisse liefert. Es ist jedoch zu bedenken, dass beim OBS-Algorithmus die Hesse-Matrix invertiert werden muss. Diese Matrix kann, je nach Modellzweck, recht groß sein, was wiederum viel Aufwand verursacht. [47, S. 322ff]

2.3 Support Vector Machine

Die bisher in Kapitel 2 diskutierten Machine-Learning-Methoden basieren darauf, den funktionalen Zusammenhang zwischen den Einflussvariablen und der/den Zielvariable(n)

möglichst gut zu approximieren. Bei der *Support Vector Machine* (SVM) verfolgt man einen anderen Ansatz. Diese Methode ist maßgeschneidert für Klassifizierungsprobleme mit zwei möglichen Klassen entwickelt worden. [8] Gegeben seien also Daten (x,y), wobei y die Zugehörigkeit zu einer von zwei Klassen angibt. Dann fasst man die zu einem Datensatz (x_i,y_i) gehörenden Einflussvariablen x_i als Punkt in einem affinem Raum auf. Dieser Raum wird auch als *Merkmalraum* bezeichnet. Üblicherweise handelt es sich dabei um den $\mathbb{R}^n$ mit Ursprung 0, wobei die Dimension n der Anzahl der Einflussvariablen entspricht. Man ordnet diese Punkte x_i nun einer von zwei Mengen zu, nämlich entsprechend der Klasse y_i. Simpel ausgedrückt ist es nun Ziel der SVM, eine Hyperebene zu finden, welche die Punktmengen linear separiert. Der Vollständigkeit halber folgen die formalen Definitionen dieser Begriffe aus der linearen Algebra:

Definition 2.9. *Eine **Hyperebene** in $\mathbb{R}^n$ ist eine $(n-1)$-dimensionale affine Menge $H \subset \mathbb{R}^n$.*

[37, S. 4f]

Definition 2.10. *Zwei Punktmengen A und B in einem n-dimensionalen Raum heißen **linear separabel**, wenn $n+1$ reelle Zahlen $w_1,\dots,w_{n+1}$ existieren, sodass für jeden Punkt $(x_1,\dots,x_n) \in A$ und für jeden Punkt $(y_1,\dots,y_n) \in B$ gilt, dass*

$$\sum_{i=1}^{n} w_i x_i \geq w_{n+1} \quad \text{und}$$
$$\sum_{i=1}^{n} w_i y_i < w_{n+1}.$$

[38, S. 63]

Wenn die beiden Punktmengen linear separabel sind, wird meist nicht nur eine Hyperebene existieren, welche diese separiert. Es stellt sich sich also die Frage, welche Hyperebene in so einem Fall gewählt werden soll. Andererseits kann die Situation eintreten, dass die Punktmengen nicht linear separabel sind. Diese beiden Probleme können recht einfach durch geschickte Erweiterungen der Definition der SVM abgefangen werden, wie im Folgenden gezeigt wird.

Definition 2.11. *Es seien (x,y) k Datensätze, wobei $y_i = 1$ oder $y_i = -1$ und $x_i = (x_{i1},\dots,x_{in}) \in \mathbb{R}^n$ für alle $i \in \{1,\dots,k\}$. Man definiert eine Hyperebene durch*

$$\{z \in \mathbb{R}^n : z\beta + \beta_0 = 0\}, \tag{2.42}$$

mit $\|\beta\| = 1$. Daraus kann eine Klassifizierungsregel abgeleitet werden:

$$G(x) = sign(z\beta + \beta_0). \tag{2.43}$$

Die Definition der ***Support Vector Machine*** *ergibt sich nun aus der Wahl von* β *und* β_0*. Dies geschieht implizit mittels eines Optimierungsproblems. Im Fall, dass die beiden Klassen linear separabel sind, definiert man das folgende Optimierungsproblem:*

$$\min_{\beta,\beta_0} \|\beta\| \tag{2.44}$$

unter den Nebenbedingungen:

$$y_i(x_i\beta + \beta_0) \geq 1, \quad i = 1,\ldots,k.$$

Sind die beiden Klassen hingegen nicht linear separabel, benötigt man zusätzlich Fehlergrößen ξ_i*,* $i = 1,\ldots,k$*. Sei außerdem* $C > 0$ *eine Konstante. Das Optimierungsproblem lautet dann:*

$$\min_{\beta,\beta_0} \|\beta\| \tag{2.45}$$

unter den Nebenbedingungen:

$$y_i(x_i\beta + \beta_0) \geq 1 - \xi_i, \quad i = 1,\ldots,k.$$
$$\xi_i \geq 0 : \sum_{i=1}^{k} \xi_i \leq C.$$

[19, S. 417ff]

Die nachfolgende Abbildung illustriert den Unterschied zwischen dem separablen und dem nicht-separablen Fall:

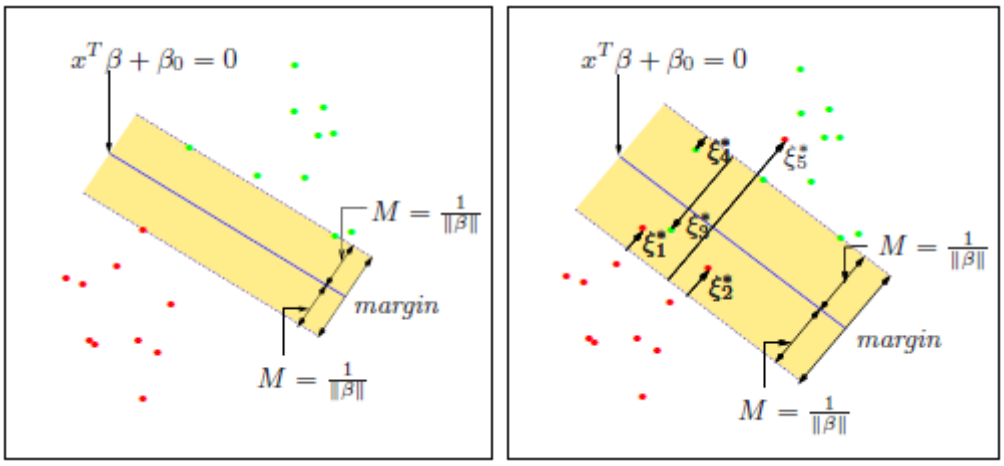

Abbildung 3: Die Punkte sind je nach Klassenzugehörigkeit rot oder grün eingefärbt. Im linken Panel sind die Punktmengen linear separabel, im rechten nicht. Die separierende Hyperebne ist im zweidimensionalen Fall eine Gerade, gegeben durch $x\beta + \beta_0 = 0$. Quelle: [19, S. 418: Fig. 12.1].

In Abbildung 3 ist auch die sogenannte *Margin* eingezeichnet (das gelbe Band parallel zur separierenden Hyperebene). Es handelt sich dabei um jene Teilmenge des Vektorraums, deren Punkte weniger als $\frac{1}{\|\beta\|}$ Abstand zur separierenden Hyperebene haben. Im separablen Fall können die Einflussvariablen nur am Rand der Margin liegen, aber nicht innerhalb. Das Optimierungsproblem (2.44) kann nämlich auch äquivalent so formuliert werden:

$$\max_{\beta,\beta_0,\|\beta\|=1} M \tag{2.46}$$

unter den Nebenbedingungen:

$$y_i(x_i\beta + \beta_0) \geq M, \quad i = 1,\ldots,k.$$

Es soll also jene Hyperebene gefunden werden, die eine möglichst große Margin zwischen den beiden Punktmengen erzeugt. Dementsprechend sind für die Wahl der Hyperebene jene Punkte einer Klasse maßgeblich, die nahe an der anderen Klasse liegen. [19, S. 418f] Punkte, deren Abstand zur Hyperebene genau M beträgt, werden *Support Vectors* genannt. Die Support Vectors spielen eine wichtige Rolle für die Performance der SVM, wie die folgende Abschätzung zeigt:

$$\mathbb{E}[\mathbb{P}(\text{Fehler})] \leq \frac{\mathbb{E}[\#\{\text{Support Vectors}\}]}{k} \tag{2.47}$$

Die erwartete Wahrscheinlichkeit eines Fehlers bei ungesehenen Daten ist also beschränkt durch das Verhältnis der erwarteten Anzahl der Support Vectors zu der Anzahl der Trainingsdatensätze k. [8, S. 275 ff] Im nicht-separablen Fall adaptiert man die Nebenbedingungen für die äquivalente Formulierung mit der Margin folgendermaßen:

$$y_i(x_i\beta + \beta_0) \geq M(1-\xi_i), \quad i = 1,\ldots,k.$$
$$\xi_i \geq 0 : \sum_{i=1}^{k} \xi_i \leq C.$$

Man erkennt an der ersten Nebenbedingung, dass ein Punkt falsch klassifiziert wird, wenn das entsprechende ξ_i größer als 1 ist. Die zweite Nebenbedingung garantiert, dass maximal $\lfloor C \rfloor$ Punkte falsch klassifiziert werden. [19, S. 418f]

Im Folgenden soll die Frage geklärt werden, wie das Optimierungsproblem (2.45) gelöst werden kann. Es handelt sich dabei um ein quadratisches Problem mit linearen Ungleichungen als Nebenbedingungen, also um ein konvexes Optimierungsproblem. Demnach sind die Karush-Kuhn-Tucker-Bedingungen nicht nur notwendig, sondern auch hinreichend. [6, S. 243f] Das Optimierungsproblem kann folgendermaßen besonders kompakt formuliert werden:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + K\sum_{i=1}^{k}\xi_i \tag{2.48}$$

unter den Nebenbedingungen:

$$y_i(x_i\beta + \beta_0) \geq 1 - \xi_i, \quad i = 1,\ldots,k.$$

Dabei wird die obere Schranke C für die Summe der Fehlergrößen durch den Kostenparameter K ersetzt. Die Lagrange-Funktion für dieses Optimierungsproblem lautet:

$$L(\beta,\beta_0,\xi,\alpha,\mu) = \frac{1}{2}\|\beta\|^2 + K\sum_{i=1}^{k}\xi_i - \sum_{i=1}^{k}\alpha_i\left[y_i(x_i\beta+\beta_0) - (1-\xi_i)\right] - \sum_{i=1}^{k}\mu_i\xi_i. \tag{2.49}$$

Die Lagrange-Funktion wird minimiert bezüglich β, β_0 und $\xi = (\xi_1,\ldots,\xi_k)$. Die Vektoren $\alpha = (\alpha_1,\ldots,\alpha_k)$ und $\mu = (\mu_1,\ldots,\mu_k)$ sind die Lagrange-Multiplikatoren. Differenziert man nun bezüglich β respektive β_0 respektive $\xi_1,\ldots,\xi_k$ und setzt die entsprechenden Ableitungen 0, so erhält man

$$\beta = \sum_{i=1}^{k}\alpha_i y_i x_i^\top \tag{2.50}$$

$$0 = \sum_{i=1}^{k}\alpha_i y_i \tag{2.51}$$

$$\alpha = K - \mu. \tag{2.52}$$

Dabei muss $\xi_i \geq 0$ für alle i gelten. Die Lagrangemultiplikatoren müssen ebenfalls nichtnegativ sein. Substituiert man (2.50) - (2.52) in (2.49), so erhält man die duale (Wolfe) Lagrange-Funktion:

$$L_D = \sum_{i=1}^{k}\alpha_i - \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k}\alpha_i\alpha_j y_i y_j x_i x_j^\top \tag{2.53}$$

Man maximiert L_D unter den Bedingungen $0 \leq \alpha_i \leq K$ und $\sum_{i=1}^{k}\alpha_i y_i = 0$. Das duale Optimierungsproblem liefert eine untere Schranke für (2.48). Zusätzlich zu den Bedingungen an die Ableitungen (2.50) - (2.52) erhält man durch die Karush-Kuhn-Tucker Bedingungen die folgenden (Un-)gleichungen:

$$\alpha_i\left[y_i(x_i\beta+\beta_0) - (1-\xi_i)\right] = 0 \tag{2.54}$$

$$\mu_i\xi_i = 0 \tag{2.55}$$

$$y_i(x_i\beta+\beta_0) - (1-\xi_i) \geq 0 \tag{2.56}$$

Insgesamt charakterisieren (2.50) - (2.56) die Lösung des eigentlichen Optimierungspro-

blem und des dualen Problems eindeutig. Das duale Problem kann einfacher algorithmisch gelöst werden als das eigentliche Problem. [19, S. 420f] Eine mögliche Herangehensweise wären *Active Set* Methoden, bei denen eine Teilmenge der Ungleichungen als Gleichungen angesetzt wird, um somit eine Rückführung auf Optimierungsprobleme mit linearen Gleichungen als Nebenbedingungen zu erreichen. [16, S. 167ff]

An der Gleichung (2.50) sieht man, dass die Lösung $\hat{\beta}$ die Form

$$\hat{\beta} = \sum_{i=1}^{k} \hat{\alpha}_i y_i x_i^\top \tag{2.57}$$

haben muss. Aufgrund von (2.54) ist ein Koeffizient $\hat{\alpha}_i$ nur dann von 0 verschieden, wenn in (2.56) Gleichheit gilt. Im nicht-separablen Fall werden die diesen Koeffizienten entsprechenden Vektoren x_i als Support Vectors bezeichnet. Manche dieser Support Vectors werden genau am Rand der Margin liegen, was gleichbedeutend mit $\hat{\xi}_i = 0$ ist. Aufgrund von (2.55) gilt $\hat{\mu}_i = 0$ für alle anderen Vektoren x_i mit $\hat{\xi}_i \geq 0$. In Kombination mit (2.52) bedeutet dies, dass ein Punkt x_i jedenfalls am Rand der Margin liegt, wenn $0 < \hat{\alpha}_i < C$ gilt. Die Gleichung (2.54) kann dann mit einem beliebigen dieser Punkte verwendet werden, um eine Lösung für $\hat{\beta}_0$ zu erhalten. Es ist üblich, den Durchschnitt über alle diese Lösungen zu bilden, um eine höhere Stabilität zu erzielen.

Die Punktmengen im rechten Panel von Abbildung 3 sind zwar nicht linear separabel, allerdings erhält man dennoch eine sinnvolle Klassifizierungsregel durch die Lösung des modifizierten Optimierungsproblems (2.45). Es kann jedoch auch vorkommen, dass eine lineare Klassifizierungsregel schlecht geeignet ist. In diesem Fall greift man auf den sogenannten Kernel-Trick zurück. Die Idee dabei ist es, den Merkmalraum mittels Basiserweiterungen in eine höhere Dimension zu heben. In günstigen Fällen liefert die Hyperebene im erweiterten Merkmalraum eine gute Trennung der beiden Klassen. Im ursprünglichen Merkmalraum erhält man dann nicht-lineare Grenzen zwischen den Klassen.

Formal funktioniert der Kernel-Trick folgendermaßen: zunächst wählt man Basisfunktionen $h_m : \mathbb{R}^n \to \mathbb{R}$, $m = 1 \ldots, M$. Dann wendet man die SVM auf die neuen Einflussvariablen $h(x_i) = (h_1(x_i), \ldots, h_M(x_i))$, $i = 1, \ldots, k$ an. Dadurch erhält man die (nicht-lineare) Klassifizierungsregel

$$\hat{G}(x) = \operatorname{sign}\left(h(x)\hat{\beta} + \hat{\beta}_0\right). \tag{2.58}$$

Die Dimension des erweiterten Merkmalraumes darf bei dieser Vorgangsweise sehr hoch werden. Dies legt die Befürchtung nahe, dass die Berechnungen sehr aufwändig werden. Das ist allerdings nicht notwendigerweise der Fall. Ersetzt man nämlich im dualen Pro-

blem (2.53) x_i durch $h(x_i)$ so erhält man

$$L_D = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} \alpha_i \alpha_j y_i y_j \langle h(x_i), h(x_j) \rangle. \tag{2.59}$$

Die Klassifizierungsregel hat dann die Form

$$h(z)\beta + \beta_0 = \sum_{i=1}^{k} \alpha_i y_i \langle h(z), h(x_i) \rangle + \beta_0. \tag{2.60}$$

Man sieht also, dass die transformierten Einflussvariablen $h(x_i)$ nur innerhalb von Skalarprodukten enthalten sind. Für geeignete Funktionen h kann die Berechnung dieser Skalarprodukte sehr günstig durchgeführt werden. Tatsächlich ist es nicht einmal notwendig, $h(x_i)$ explizit anzugeben. Stattdessen reicht es, die Kernel-Funktion zu spezifizieren:

$$K(x, x') = \langle h(x), h(x') \rangle. \tag{2.61}$$

Die Kernel-Funktion liefert also ein Skalarprodukt im erweiterten Merkmalraum. Bekannte Beispiele für Kernel-Funktionen sind:

$$\text{Polynom vom Grad p:} K(x, x') = (1 + \langle x, x' \rangle)^p \tag{2.62}$$

$$\text{Radiale Basis:} K(x, x') = \exp(-\gamma \|x - x'\|^2) \tag{2.63}$$

$$\text{Neural Network:} K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2) \tag{2.64}$$

Im erweiterten Merkmalraum kann es durchaus vorkommen, dass Punktmengen, die im eigentlichen Merkmalraum nicht linear separabel sind, perfekt separiert werden können. Dabei kommt dem Kostenparamter K eine wichtige Rolle zu. Ist dieser sehr hoch angesetzt, fördert dies Lösungen, die die Punktmengen möglichst gut separieren. Dadurch können die Grenzen im eigentlichen Merkmalraum allerdings sehr kurvig werden, was wiederum ungünstig bei der Verallgemeinerung auf ungesehene Daten sein kann. [19, S. 423f]

3 Daten

Die statistischen Modelle aus Kapitel 2 sind so konzipiert, dass sie anhand von Daten kalibriert werden. Die zugrundeliegenden Daten sind dabei wesentlich für die Qualität der Modelle. Deswegen lohnt es sich beim Modellieren, wenn man sich zunächst eingehend mit den Daten auseinandersetzt. Dieses Kapitel wird folgende Aspekte des Themas „Daten“ beleuchten: Datenquellen, Datenanalyse, Datenaufbereitung, modellunabhängige Merkmalauswahl und der SMOTE-Algorithmus.

Allgemein werden in dieser Arbeit die Daten immer in der Form (x,y) betrachtet. Dabei ist x eine $n \times k$-Matrix und y ein n-dimensionaler Vektor. Insgesamt ist (x,y) somit eine $(n+1) \times k$-Matrix. Dabei bildet eine einzelne Zeile i immer einen zusammengehörigen Datensatz bestehend aus Einflussvariablen x_i, welche Informationen über das fragliche Unternehmen enthalten, und dem Indikator y_i, welcher angibt, ob das Unternehmen in Konkurs gegangen ist oder nicht.

3.1 Datenquellen

In dieser Arbeit werden Finanzdaten, wie beispielsweise Jahresumsatz oder Gewinn/Verlust, die Grundlage für die Modelle bilden. Eine gute Quelle für diese Daten ist der Jahresabschluss der jeweiligen Unternehmen, sofern dieser vorhanden und zugänglich ist. Für Unternehmen ab einer gewissen Mindestgröße ist die Erstellung verbindlich (für Österreich siehe §189 ff UGB). Für Aktiengesellschaften gibt es sogar Veröffentlichungspflichten (siehe §277 UGB). Ist man also an Konkurswahrscheinlichkeiten von börsennotierten Unternehmen interessiert (zum Beispiel als Aktionär oder als Käufer von Anleihen), so hat man einen zuverlässigen Zugang zu den nötigen Daten.

Wenn man hingegen an kleinen, nicht an der Börse notierten Unternehmen interessiert ist, dann ist der Zugang zu den Daten nicht mehr so einfach. Üblicherweise werden aber vor allem Banken (als Kreditgeber) mit kleineren Unternehmen zu tun haben. Diese können die erforderlichen Daten auf Antragsformularen abfragen und dann den eigenen Bestand und die Historie verwenden, um die Modelle zu kalibrieren. Diese Vorgangsweise wird als *Credit Scoring* bezeichnet. [1, S. 59f]

Ein weiterer Aspekt, der bedacht werden muss, ist, für welchen Zeitraum die Konkurswahrscheinlichkeiten geschätzt werden sollen. Man muss nämlich mindestens ebenso lange jene Unternehmen beobachten, deren Daten zur Kalibrierung des Modells dienen sollen.

3.2 Datenanalyse

Der Begriff „Datenanalyse“ ist sehr weit gefasst und könnte viele Aspekte enthalten, bis hin zur Modellierung mittels einer der Methoden aus Kapitel 2. Im Kontext dieses Kapitels ist eine erste Analyse der Daten (x, y) nach der Akquirierung der selbigen gemeint. Dabei geht es nicht notwendigerweise darum, bereits Erkenntnisse über die eigentliche Problemstellung zu gewinnen. Vielmehr dient dieser Schritt dazu, sich einen ersten Überblick über die Daten zu verschaffen und Erkenntnisse über die Daten an sich zu erlangen. Wenn man die Daten nicht selbst gesammelt hat, ist dieser Schritt umso wichtiger, da man sich erst mit diesen vertraut machen muss. Folgende Fragen können beispielsweise interessant sein:

- Welche Merkmale sind vorhanden?
- Wie viele Datensätze stehen zur Verfügung (Stichprobenumfang)?
- Wie viele Datensätze sind pro Kategorie (nicht in Konkurs gegangen/in Konkurs gegangen) vorhanden?
- Gibt es Datensätze, bei denen Merkmale fehlen (siehe auch Kapitel 3.3)?
- Gibt es Ausreißer oder Werte, die fehlerhaft wirken?
- Wie sieht die Datenstruktur der einzelnen Merkmale aus (z.B. stetig oder diskret, empirische Verteilung)?

[7, S. 6ff]

Manche dieser Punkte mögen trivial wirken, trotzdem ist diese erste Analysephase unerlässlich. Die ersten vier Fragestellungen können bereits durch eine rein visuelle Untersuchung geklärt werden. Sollte sich dabei herausstellen, dass der Stichprobenumfang sehr klein ist oder eine der beiden Kategorien (nicht in Konkurs gegangen/in Konkurs gegangen) nur wenige Datensätze umfasst, dann sind dies bereits wichtige Informationen. Derartige Daten können sich nämlich nachteilig auf die Performance der Modelle auswirken. Auch das Verhältnis von Merkmalen zum Stichprobenumfang (pro Kategorie) kann interessant sein. Eine Studie zur logistischen Regression suggeriert beispielsweise, dass pro Merkmal mindestens 10 Datensätze von jeder Kategorie vorhanden sein sollten. Dies ist natürlich nur eine Daumenregel, aber wenn dieses Verhältnis stark verletzt ist, dann sollten wahrscheinlich Merkmale ausgeschlossen werden. [3, S. 208] Stellt man darüber hinaus fest, dass manche Datensätze nicht vollständig sind, dann muss man dies berücksichtigen. Die statistischen Modelle können nämlich im Allgemeinen nicht mit fehlenden Werten umgehen. Details zu diesem Thema finden sich in Kapitel 3.3.

Die letzten zwei Fragestellungen hingegen werden sich nicht durch eine reine Untersuchung der Rohdaten restlos klären lassen. Mittels einfacher Visualisierungsmethoden kann man sich allerdings ein gutes Bild verschaffen. Man könnte zum Beispiel einen Boxplot für jedes Merkmal erstellen. Dabei können etwa Ausreißer sichtbar werden. [43, S. 39ff] Auch die empirische Verteilung, beispielsweise als Histogramm dargestellt, könnte interessant sein.

3.3 Datenaufbereitung

Der nächste Schritt nach der ersten Analysephase besteht darin, die Daten für das Modellieren vorzubereiten. Hat man festgestellt, dass es fehlende Werte gibt, dann muss man sich nun darum kümmern. Bevor man sich entscheidet, wie man mit den fehlenden Werten umgehen möchte, kann es sich lohnen zu analysieren, wie die fehlenden Werte zustande gekommen sind. Dafür gibt es einen formalen Ansatz, der drei Fälle unterscheidet:

Definition 3.1. *Seien $(x,y)_{com}$ die vollständigen Daten. Diese setzen sich zusammen aus den beobachteten Daten $(x,y)_{obs}$ und den fehlenden Daten $(x,y)_{mis}$. Es gilt also*

$$(x,y)_{com} = (x,y)_{obs} \cup (x,y)_{mis},\ (x,y)_{obs} \cap (x,y)_{mis} = \emptyset. \tag{3.1}$$

Sei I eine Matrix von Zufallsvariablen, welche binär sind, also nur die Werte 0 oder 1 annehmen können. Je nach Ausprägung dieser Zufallsvariablen fehlt ein Wert (1) oder nicht (0). Die multivariate Verteilung dieser Zufallsvariablen wird mit $\mathscr{L}(I)$ bezeichnet. Man unterscheidet dann drei Gründe für fehlende Werte:

- ***Missing completely at random** (MCAR):*

$$\mathscr{L}(I|(x,y)_{com}) = \mathscr{L}(I). \tag{3.2}$$

- ***Missing at random** (MAR):*

$$\mathscr{L}(I|(x,y)_{com}) = \mathscr{L}(I|(x,y)_{obs}). \tag{3.3}$$

- ***Missing not at random** (MNAR):*

$$\mathscr{L}(I|(x,y)_{com}) = \mathscr{L}(I|(x,y)_{obs},(x,y)_{mis}). \tag{3.4}$$

[13, S. 31ff] [39, S. 150ff]

Es wird also unterschieden, ob das Fehlen von Werten abhängig von den beobachteten Werten $(x,y)_{\text{obs}}$ beziehungsweise den tatsächlichen Werten der nicht beobachteten (=fehlenden) Merkmalen $(x,y)_{\text{mis}}$ ist oder nicht. In der Praxis wird der erste Fall (MCAR)

eher selten auftreten. Üblicherweise wird sich die Ursache für fehlende Werte nämlich in den beobachteten Werten finden lassen (MAR). Dazu folgendes Beispiel: in Österreich sind Bilanzierungs- und Veröffentlichungspflichten oft abhängig von der Firmengröße (beispielsweise am Umsatzerlös gemessen). Die Wahrscheinlichkeit, ob ein Merkmal fehlt, könnte also zum Beispiel vom beobachteten Merkmal „Umsatzerlös“ abhängig sein. [13, S. 36f]

Müssen hingegen gewisse Finanzkennzahlen beispielsweise dann nicht gemeldet werden, wenn diese unter eine Geringfügigkeitsgrenze fallen, dann ist die Wahrscheinlichkeit abhängig vom nicht beobachteten, tatsächlichen Wert (MNAR). MNAR beschreibt also eine Situation, in welcher Werte systematisch für einen bestimmten Wertebereich fehlen. Auch wenn die fehlenden Werte MAR sind, kann es einen Zusammenhang zwischen der Wahrscheinlichkeit, dass ein Wert fehlt, und dem nicht beobachteten, tatsächlichen Wert geben. Der feine Unterschied zu MNAR besteht darin, dass dieser Zusammenhang bei MAR keine Rolle mehr spielt, sobald die Informationen aus $(x,y)_{\text{obs}}$ einbezogen wurden. [39, S. 151]

Um mit fehlenden Werten umgehen zu können, gibt es verschiedene Methoden, wobei manche davon im Folgenden beschrieben werden. Viele dieser Methoden setzen MAR voraus. Allerdings gibt es keine Möglichkeit auf MAR zu testen, da dafür genau die fehlenden Werte benötigt werden würden. Diese Voraussetzung muss also durch theoretische Überlegungen plausibilisiert werden. [13, S. 35f]

Die einfachste Methode, um mit fehlenden Werten umzugehen, ist es, nicht vollständige Datensätze zu löschen. Damit erhält man eine Teilmenge der Daten, welche keine fehlenden Werte aufweist. Im Fall von MCAR kann dies attraktiv sein, da diese Methode dann günstige statistische Eigenschaften aufweist. Beispielsweise sind Schätzer, die auf den vollständigen Daten unverzerrt wären, auch auf der Teilmenge unverzerrt. Darüber hinaus sind statistische Tests genauso sinnvoll auf der kleineren Datenmenge, wie sie das auf den eigentlichen Daten gewesen wären. Liegt allerdings nur MAR vor, dann können Schätzer durch diese Methode verzerrt werden. [4, S. 6] Darüber hinaus gehen durch diese Methode natürlich auch Datensätze verloren. Dies ist vor allem bei kleinem Stichprobenumfang ungünstig. Eine Alternative bieten sogenannte *Imputation*-Methoden. Dabei werden die Lücken in den Daten mit sinnvollen Werten aufgefüllt. Beispielsweise könnten Mittelwert oder Median der vorhandenen Werte eines Merkmals verwendet werden. Den Mittelwert zu verwenden hat den Vorteil, dass der Mittelwert der Daten unverändert bleibt. Allerdings wird durch diese Methode die Varianz systematisch unterschätzt, nämlich proportional zum Anteil der fehlenden Werte. Sind die Daten asymmetrisch verteilt, dann kann der Median eine sinnvollere Wahl sein. Die Nachteile betreffend der Varianz können da-

durch aber auch nicht ausgemerzt werden. Ein anderer Ansatz besteht darin, nicht einen Wert pro fehlendem Wert einzusetzen, sondern m verschiedene Werte. Dadurch generiert man m vollständige Datenmengen, welche dann alle als Input für die statistischen Modelle verwendet werden. Dadurch erhält man natürlich auch m Schätzer, welche dann zu einem Schätzer kombiniert werden müssen. Die einzusetzenden Werte können beispielsweise mit MCMC-Methoden von einer passenden Verteilung gezogen werden. Durch diesen Ansatz trägt man der Unsicherheit bezüglich der fehlenden Werten Rechnung. Allerdings verursacht diese Methode auch den größten Aufwand. [13, S. 42ff]

Eine besonders elegante Technik, die ebenfalls das Problem der fehlenden Werte löst, ist Binning. Die Idee dahinter ist es, stetige Merkmale in diskrete Merkmale (mit endlich vielen möglichen Ausprägungen) umzuwandeln. Die einzelnen Ausprägungen oder Kategorien werden als *Bins* bezeichnet. Auch Merkmale, die bereits diskret sind, können so gehandhabt werden. In diesem Falle kann es zu einer Reduktion der möglichen Ausprägungen kommen. [34, S. 1ff] Am besten kann dies durch ein Beispiel veranschaulicht werden. Der Umsatzerlös, welcher bereits in diesem Kapitel erwähnt wurde, kann Werte aus $\mathbb{R}$ annehmen. [4] Nun kann man $\mathbb{R}$ beispielsweise in die folgenden Intervalle zerlegen:

$$\{(-\infty, 0), [0, 700000), [700000, 100000000), [100000000, \infty)\}.$$

Damit hat man nun ein kategorisches Merkmal mit 4 Ausprägungen. Falls dieses Merkmal bei manchen Datensätzen fehlt, fügt man einfach eine weitere Ausprägung hinzu:

$$\{(-\infty, 0), [0, 700000), [700000, 100000000), [100000000, \infty), \text{Missing}\}.$$

Kategorische Merkmale kommen in der Praxis öfter vor und stellen daher meistens kein Problem für statistische Modelle dar. Somit ist dies eine praktikable Möglichkeit, um mit fehlenden Werten umzugehen.

Ein weiterer Vorteil von Binning besteht darin, dass für kategorische Merkmale das sogenannte *Information Value* (IV) berechnet werden kann. Dieses kann verwendet werden, um Merkmale (modellunabhängig) zu selektieren. [18, S. 528f] Details dazu findet man in Kapitel 3.4. Es sollte allerdings stets beachtet werden, dass Binning natürlich auch einen Einfluss auf die Modelle hat, die auf Basis der Daten kalibriert werden. Daher muss im Einzelfall entschieden werden, ob Binning sinnvoll ist oder sich möglicherweise negativ auf die Performance auswirkt.

[4] Negative Werte werden in der Praxis eher nicht vorkommen, a priori muss man diese aber nicht ausschließen.

Wenn man sich dazu entscheidet, Binning durchzuführen, stellt sich die Frage, wie man die Einteilung in Bins vornimmt. Die zwei einfachsten Möglichkeiten sind *equal length* und *equal weight*. Im ersten Fall wird der Wertebereich in gleich lange Intervalle aufgeteilt. Dabei erhält man endlich viele Intervalle, da ja der maximale und der minimale Wert jedes Merkmals der Daten bekannt ist. Im zweiten Fall werden die Bins so gewählt, dass jeder Bin gleich viele Datensätze enthält. Das Problem bei diesen Methoden ist, dass die daraus resultierenden Transformationen der Daten nicht notwendigerweise zu guten Modellen führen. [48, S. 3230]

Alternative Binning-Methoden nutzen häufig das sogenannte *Weight of Evidence* (WOE):

Definition 3.2. *Es sei ein diskretes Merkmal mit endlich vielen Ausprägungen gegeben. Das* ***Weight of Evidence*** w_i *für die i-te Ausprägung ist gegeben durch:*

$$w_i := ln\left(\frac{p_i}{q_i}\right). \tag{3.5}$$

Dabei ist p_i die Anzahl der nicht in Konkurs gegangenen Unternehmen, welche die i-te Ausprägung besitzen, dividiert durch die Gesamtzahl der nicht in Konkurs gegangenen Unternehmen. Dementsprechend ist q_i die Anzahl der in Konkurs gegangenen Unternehmen, welche die i-te Ausprägung besitzen, dividiert durch die Gesamtzahl der in Konkurs gegangenen Unternehmen.

[18, S. 527f]

Ersetzt man die Ausprägungen eines Merkmals durch ihre WOE-Werte, so erhält man wieder ein numerisches Merkmal, welches theoretisch stetig ist (allerdings nur so viele Werte annimmt, wie das Merkmal Ausprägungen besitzt).

Die folgenden Richtlinien garantieren zwar auch nicht, dass das Binning zu sinnvollen Modellen führt, können aber hilfreich sein:

- Fehlende Werte sind in einem eigenen Bin enthalten.
- Jeder Bin enthält zumindest 5% der Datensätze.
- Jeder Bin enthält Datensätze beider möglichen Ausgänge (in Konkurs gegangen/-nicht in Konkurs gegangen).
- Der Anteil der in Konkurs gegangenen Unternehmen, und das WOE sind hinreichend unterschiedlich zwischen den Bins.
- Das WOE für die nicht-fehlenden Werte folgt einer logischen Verteilung, von negativ zu positiv.

[41, S. 80]

Es ist durchaus üblich, dass das Binning zumindest teilweise manuell durchgeführt wird. Dies hat allerdings auch Nachteile: der Prozess kann zeitintensiv sein, es muss nicht zwingend eine Lösung existieren, die sämtlichen Anforderungen gerecht wird (dies ist allerdings nicht a priori ersichtlich), und es ist nicht garantiert, dass eine gefundene Lösung auch optimal ist. Alternativ gibt es auch Algorithmen, die das Binning durchführen können. Beispielsweise kann Binning als Optimierungsproblem aufgefasst werden. Dabei können die bereits erwähnten Richtlinien als Nebenbedingungen realisiert werden. [34, S. 2ff] Dies soll an dieser Stelle jedoch nicht weiter vertieft werden.

Entscheidet man sich gegen Binning und arbeitet daher direkt mit den stetigen Merkmalen, dann kann es sinnvoll sein, diese zu standardisieren. Dabei wird der Stichprobenmittelwert subtrahiert und anschließend durch die Stichprobenstandardabweichung dividiert. [2, S. 95] Beispielsweise suggeriert eine Studie zu Neural Networks, dass die Standardisierung einen positiven Einfluss auf die Modelle hat. [40, S. 395f]

Zum Abschluss des Kapitels über Datenaufbereitung wird nun noch eine Technik vorgestellt, welche Verbindungen zur Merkmalauswahl aufweist, die im nächsten Kapitel behandelt werden wird. Bei dieser Technik handelt es sich um die *Principal Component Analysis* (PCA).

Definition 3.3. *Sei X ein n-dimensionaler Zufallsvektor. Darüber hinaus sei α_1 ein Vektor, sodass*

$$\mathbb{V}\left[\alpha_1^\top X\right] = \mathbb{V}\left[\sum_{i=1}^{n} \alpha_{1i} X_i\right] \tag{3.6}$$

*maximal ist, wobei man $\alpha_1^\top \alpha_1 = 1$ voraussetzt. Dann heißt $\alpha_1^\top X$ die erste **Principal Component** von X. Die k-te Principal Component wird dann ähnlich definiert. Sei nämlich α_k ein Vektor mit $\alpha_k^\top \alpha_k = 1$, welcher*

$$\mathbb{V}\left[\alpha_k^\top X\right] \tag{3.7}$$

unter der zusätzlichen Bedingung maximiert, dass $\alpha_k^\top X$ unkorreliert mit $\alpha_1^\top X, \alpha_2^\top X, \ldots, \alpha_{k-1}^\top X$ ist, also

$$Cov\left[\alpha_k^\top X, \alpha_i^\top X\right] = 0, \quad i = 1, \ldots, k-1. \tag{3.8}$$

[25, S. 2ff]

Insgesamt können n Principal Components für einen n-dimensionalen Zufallsvektor gefunden werden. Die Idee der PCA ist es, die Dimension der Daten zu reduzieren, indem man zu anderen Variablen transformiert (den Principal Components) und dabei möglichst viel der Varianz innerhalb der Daten zu erhalten. Idealerweise erklären bereits $p << n$

Principal Components einen Großteil der Varianz der Daten. Die restlichen $n - p$ Principal Components werden in diesem Fall verworfen (aber Achtung, dies ist nicht immer die korrekte Vorgehensweise, siehe weiter unten). [25, S. 1ff] Ein weiterer Vorteil der Principal Components ist, dass diese per definitionem unkorreliert sind.

Aus Definition 3.3 wird ersichtlich, dass man für jede Principal Component ein Optimierungsproblem lösen muss, um diese zu bestimmen. Angenommen man kennt die Kovarianz-Matrix Σ von X. Dann stellt sich heraus, dass der maximale Wert, den die Varianz $\mathbb{V}\left[\alpha_k^\top X\right]$ annehmen kann, dem k-t größten Eigenwert λ_k von Σ entspricht. Der Lösung für α_k entspricht wiederum dem zu λ_k gehörenden Eigenvektor. [25, S. 5f]

Im Kontext dieser Arbeit ist die Kovarianzmatrix der Merkmale natürlich nicht bekannt. In diesem Fall ersetzt man diese einfach durch die Stichproben-Kovarianzmatrix, welche als Schätzer für die tatsächliche Kovarianzmatrix dient. [25, S. 30f]

Die PCA besitzt also einige wünschenswerte Eigenschaften, allerdings gibt es auch Nachteile: da es sich um eine lineare Technik handelt, ist sie nicht für komplexere Probleme mit nicht-linearen Korrelationen geeignet. [49, S. 456] Darüber hinaus wird die Korrelation zwischen Merkmalen und der Zielvariable (in Konkurs gegangen/nicht in Konkurs gegangen) nicht berücksichtigt. Es gibt durchaus Beispiele, bei denen Principal Components, die für einen signifikanten Anteil der Varianz in den Daten verantwortlich sind, nur wenig Einfluss auf die Zielvariable haben. Umgekehrt sind bei den gleichen Beispielen manche Principal Components, welche man normalerweise verwerfen würde, stark korreliert mit der Zielvariable. [25, S. 173ff] Es ist also nicht notwendigerweise eine gute Strategie, einfach die letzten Principal Components zu verwerfen, welche nur für einen geringen Anteil der Varianz in den Daten verantwortlich sind.

Es existieren auch nicht-lineare Verallgemeinerungen der PCA. Dies kann beispielsweise durch den Einsatz von Neural Networks erreicht werden. Dadurch kann zumindest ein Nachteil der Technik ausgeglichen werden. [27, S. 553ff] Dies soll in dieser Arbeit allerdings nicht näher beleuchtet werden.

3.4 Modellunabhängige Merkmalauswahl

In Kapitel 2 wurden bereits verschiedene Möglichkeiten erwähnt, wie man für ein konkretes Modell aussagekräftige Merkmale selektiert. Im Gegensatz dazu sollen nun in diesem Kapitel Techniken vorgestellt werden, die modellunabhängig anwendbar sind. Die erste dieser Techniken basiert auf dem sogenannten *Information Value*.

Definition 3.4. *Es sei ein diskretes Merkmal mit k unterschiedlichen Ausprägungen gegeben. Das* ***Information Value*** *wird folgendermaßen berechnet:*

$$\sum_{i=1}^{k}(p_i - q_i)w_i$$

Dabei ist p_i die Anzahl der nicht in Konkurs gegangenen Unternehmen, welche die i-te Ausprägung besitzen, dividiert durch die Gesamtzahl der nicht in Konkurs gegangenen Unternehmen. Dementsprechend ist q_i die Anzahl der in Konkurs gegangenen Unternehmen, welche die i-te Ausprägung besitzen, dividiert durch die Gesamtzahl der in Konkurs gegangenen Unternehmen. Schließlich ist w_i das WOE der i-ten Ausprägung des Merkmals (siehe Definition (3.2)*).*

[18, S. 528f]

Das Information Value ist also ein Maß dafür, wie sich die Verteilungen der in Konkurs gegangen Unternehmen beziehungsweise der nicht in Konkurs gegangenen Unternehmen bei einem Merkmal unterscheiden. Die Merkmale können dann anhand ihres Information Values gereiht werden. Üblicherweise zieht man es in Betracht, Merkmale in das Modell zu inkludieren, wenn deren Information Value über 0,1 liegt. [18, S. 528f]

Abbildung 4 zeigt das Information Value für 64 Finanzkennzahlen (Merkmale) von polnischen Unternehmen. Tatsächlich haben nur zwei Merkmale ein Information Value unter 0,1. In Tabelle 3 ist eine feinere Kategorisierung des Information Values dargestellt.

Information Value	Vorhersagekraft
< 0,02	kaum
0,02 bis 0,1	schwach
0,1 bis 0,3	mittel
> 0,3	stark

Tabelle 3: Kategorisierung des Information Value. Quelle: [46, S. 6].

Entscheidet man sich beispielsweise, nur Merkmale mit einer starken Vorhersagekraft zu verwenden, dann kann man im Beispiel aus Abbildung 4 immerhin 20 Merkmale ausschließen. Es führt allerdings nicht notwendigerweise zu den besten Modellen, wenn man einfach die Merkmale mit dem höchsten Information Value kombiniert.

Attr15_woe	1.11093011525452	Attr51_woe	0.435913871167118
Attr34_woe	1.08197855527697	Attr31_woe	0.431195690688295
Attr26_woe	1.07277591956598	Attr24_woe	0.42078575046041
Attr25_woe	1.06029534078988	Attr30_woe	0.418563725133119
Attr16_woe	0.92939181831241	Attr54_woe	0.410497064459454
Attr41_woe	0.926144294149296	Attr50_woe	0.409253064998285
Attr6_woe	0.923527660840244	Attr28_woe	0.405058944242441
Attr9_woe	0.842023081529542	Attr53_woe	0.404906108901391
Attr13_woe	0.764067899522117	Attr32_woe	0.391561897725319
Attr39_woe	0.755972274106835	Attr3_woe	0.357736409600557
Attr12_woe	0.75335091993937	Attr11_woe	0.356972209363195
Attr2_woe	0.752343113913953	Attr5_woe	0.355829227698815
Attr17_woe	0.744006305025117	Attr40_woe	0.347768743040695
Attr23_woe	0.714135414940433	Attr52_woe	0.329567708484176
Attr8_woe	0.666504654587341	Attr33_woe	0.32950467097278
Attr1_woe	0.63001685652704	Attr55_woe	0.284099510838971
Attr10_woe	0.62436811948054	Attr48_woe	0.242810529698204
Attr19_woe	0.623628293519079	Attr56_woe	0.241536022941092
Attr35_woe	0.603282607870454	Attr29_woe	0.239946137993203
Attr46_woe	0.577288600025841	Attr58_woe	0.236930875396948
Attr45_woe	0.574575165484483	Attr59_woe	0.226431784624906
Attr27_woe	0.571684091506124	Attr49_woe	0.2211550935085
Attr42_woe	0.536153817339355	Attr21_woe	0.210579481205652
Attr7_woe	0.53615212147665	Attr43_woe	0.163676527797842
Attr14_woe	0.53615212147665	Attr37_woe	0.159566559040414
Attr18_woe	0.53615212147665	Attr20_woe	0.159101401753999
Attr22_woe	0.52053568665261	Attr61_woe	0.15333244410451
Attr62_woe	0.519492047970587	Attr64_woe	0.152224334781324
Attr38_woe	0.501205399537984	Attr44_woe	0.131971457786087
Attr57_woe	0.496515637462703	Attr60_woe	0.122172429347522
Attr63_woe	0.49180293203314	Attr36_woe	0.121387497588717
Attr4_woe	0.452252384888798	Attr47_woe	0.0975302002672442

Abbildung 4: Information Value für die 64 Merkmale der Daten von polnischen Unternehmen. Quelle: Eigene Darstellung.

Eine weniger rigorose Alternative besteht darin, Expertenwissen und Erfahrungswerte in die Merkmalauswahl einfließen zu lassen. [18, S. 528] Dieses Wissen kann den oft mühsamen Modellierungsprozess beschleunigen und die resultierenden Modelle können als Augsangspunkt dienen. Modelle, die aus anderen Methoden der Merkmalselektion entstanden sind, müssen dann das „Expertenmodell“ übertreffen, um in Betracht gezogen zu werden.

3.5 Trainingsdaten, Validierungsdaten und Testdaten

Man unterscheidet bei den Daten zwischen drei Begriffen:

- Trainingsdaten: Jene Datensätze, die zur Kalibrierung der Modelle verwendet werden.
- Validierungsdaten: Jene Datensätze, die verwendet werden, um Parameter festzulegen. Beispielsweise kann die Anzahl der Hidden Layer und Knoten bei Neural Networks mittels der Performance auf den Validierungsdaten festgelegt werden.
- Testdaten: Jene Datensätze, die ausschließlich zur Beurteilung der Performance des fertigen Modells verwendet werden.

[36, S. 354]

Um diese drei Mengen zu erhalten, kann man beispielsweise die gesamten verfügbaren Daten aufspalten. Dieser Schritt sollte früh in der Aufbereitungs- und Analysephase erfolgen, da man ja nicht möchte, dass die Testdaten die Modelle beeinflussen. Typischerweise wählt man dazu eine zufällige Teilmenge der Daten aus, welche als Testdaten genutzt werden sollen. Die restlichen Daten können dann in Trainingsdaten und Validierungsdaten aufgeteilt werden. Es gibt keine feste Regel, wie viele Datensätze als Testdaten verwendet werden. Dies kann beispielsweise davon abhängig gemacht werden, wie viele Datensätze benötigt werden, um eine gute Kalibrierung der Modelle zu erreichen. In dieser Arbeit wurden 20% der insgesamt vorhandenen Datensätze als Testdaten verwendet und 10% als Validierungsdaten, also bleiben 70% der Daten zur Kalibierung übrig.

Diese Vorgangsweise wird gewählt, weil ja die Modellierung der Relationen in den vorhandenen Daten nicht das primäre Ziel, sondern nur Mittel zum Zweck ist. Vielmehr möchte man ein Modell erstellen, welches bei neuen Daten, bei welchen die Zielvariable noch unbekannt ist, zuverlässige Ergebnisse liefert. Um dies zu simulieren, verwendet man die Testdaten, welche nicht in die Modellbildung eingeflossen sind. Diese werden auch als „ungesehene“ Daten bezeichnet. Mittels der Testdaten kann man untersuchen, ob *overfitting* vorliegt. Overfitting liegt nämlich dann vor, wenn die Performance auf den

Trainingsdaten gut, aber auf den Testdaten schlecht ist. Dies ist möglich, da man, im Gegensatz zu einer realen Situation, die Zielvariablen auf den Testdaten kennt. [23, S. 29ff]

3.6 SMOTE

Bei Klassifizierungsproblemen tritt immer wieder das Problem auf, dass zu einer Klasse nur sehr wenige Datensätze vorhanden sind. Zum Beispiel bei den für diese Arbeit verwendeten Daten machen die in Konkurs gegangenen Unternehmen nur etwa 2 % der Datensätze aus. Dies kann zu schlechter Performance der Modelle auf der wenig vertretenen Klasse führen. Genauer gesagt, können die Modelle dazu tendieren, Datensätze eher der stark vertretenen Klasse zuzuordnen. Dies ist besonders unerfreulich, da oft die Datensätze der kleinen Klasse erkannt werden sollen.

Um diesem Problem entgegenzuwirken, wurden verschiedene Techniken entwickelt. Eine Möglichkeit besteht darin, eine Teilmenge der stark vertretenen Daten zu entfernen, um somit eine ausgeglichenere Datenmenge zu erhalten. Dies wird als *Undersampling* bezeichnet. Liegt allerdings ein sehr deutliches Ungleichgewicht in den ursprünglichen Daten vor, so müssen viele Datensätze verworfen werden. Dabei können auch wertvolle Informationen verloren gehen und die gewünschte Performancesteigerung fällt schwächer aus als erhofft, falls sie überhaupt eintritt. Alternativ dazu könnte man auch *Oversampling* anwenden. Dazu werden Datensätze der kleinen Klasse - meist zufällig ausgewählt - mehrfach den Trainingsdaten hinzugefügt. Dadurch verursachen Fehler bei der Vorhersage der kleinen Klasse zwar mehr Kosten in den Trainingsalgorithmen, allerdings führt dies nicht notwendigerweise zu einer präziseren Abgrenzung der Klassen. Darüber hinaus können viele gleichartige Datensätze zum Overfitting der kleinen Klasse führen, wodurch das Modell schlechter auf neue Daten verallgemeinert. Um die Nachteile des Oversamplings auszugleichen, wurde die *Synthetic Minority Oversampling Technique* (SMOTE) entwickelt. [10, S. 863ff]

Wie beim herkömmlichen Oversampling sollen bei SMOTE zusätzliche Datensätze der kleinen Klasse hinzugefügt werden. Beispielsweise könnte eine Anzahl N an zusätzlichen Datensätzen festgelegt werden, die pro Datensatz aus der kleinen Klasse generiert werden sollen. Dann wird für jeden Datensatz aus der kleinen Klasse die folgende Prozedur durchgeführt: Zunächst werden die *K nearest neighbors* [5] aus der gleichen Klasse bestimmt. Dann werden N von diesen K Datensätzen (Ziehen mit Zurücklegen) ausgewählt, um neue Datensätze zu generieren. Dies geschieht, indem die Differenz (i.A. vektorwertig) zwischen dem jeweiligen nearest neighbor und dem ursprünglichen Datensatz gebildet

[5] K nearest neighbors: Damit sind die K Datensätze gemeint, die den geringsten Abstand zu dem ursprünglichen Datensatz bezüglich einer Metrik haben. [10, S. 866]

wird. Schließlich wird diese Differenz mit einem zufälligen Skalar aus dem Intervall $(0, 1)$ multipliziert und das Ergebnis zu dem ursprünglichen Datensatz addiert. Abbildung 5 visualisiert dieses Verfahren. Mit einfachen Anpassungen können auch nicht-ganzzahlige Vielfache der Datensätze der kleinen Klasse generiert werden. [10, S. 866ff]

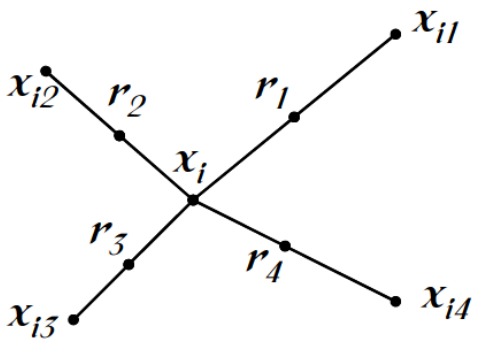

Abbildung 5: Visualisierung des SMOTE-Algorithmus. Der Punkt x_i ist der ursprüngliche Datensatz und x_{i1} bis x_{i4} sind die vier nearest neighbors. Die synthetischen Datensätze sind r_1 bis r_4. Quelle: [10, S. 867, Abb. 1] .

Die beschriebene Vorgangsweise beim SMOTE-Algorithmus soll die angesprochenen Probleme des Oversamplings reduzieren und gleichzeitig die Performance der Modelle verbessern, unter anderem indem das Verhältnis zwischen den Klassen in Daten ausgeglichen wird. [10, S. 864ff] Für R ist SMOTE beispielsweise im Paket „performanceEstimation“ verfügbar. [42]

4 Praktische Anwendung

Für dieses Kapitel wurden Techniken aus Kapitel 3 angewendet und Modelle aus Kapitel 2 kalibriert. Dies geschah mittels der Statistiksoftware R [6] und mittels einiger für R verfügbare Softwarepakete, welche Implementationen der benötigten Algorithmen bereitstellen. Teile des Codes für diese Arbeit wurden von Mukeri, Shaikh und Gaikwad übernommen, die ihren Code unter https://github.com/amir1m/companies-bankruptcy-forecast (zuletzt abgerufen am 15.09.2021) zur Verfügung stellen (siehe auch [29]).

4.1 Beispieldaten und vorbereitende Schritte

Um die Modelle zu kalibrieren und zu testen, wurden Daten von polnischen Unternehmen verwendet. Eine Beschreibung der insgesamt 64 Finanzkennzahlen ist in Appendix A zu finden. Ein weiteres Merkmal der Daten gibt an, ob das entsprechende Unternehmen in Konkurs gegangen ist oder nicht. Tabelle 4 zeigt eine Zusammenfassung der Daten. Daraus ist ersichtlich, dass insgesamt 10000 Datensätze zur Verfügung stehen, von denen jedoch nur 203 in Konkurs gegangen sind.

Nicht in Konkurs gegangen	In Konkurs gegangen	Gesamt
9797	203	10000

Tabelle 4: Zusammenfassung der Daten von polnischen Unternehmen. Quelle: Eigene Darstellung.

In weiterer Folge werden die Daten von den polnischen Unternehmen kurz als „Beispieldaten“ bezeichnet. Auf den ersten Blick ist aus Tabelle 4 ersichtlich, dass der Anteil der in Konkurs gegangenen Unternehmen mit circa 2% sehr gering ist. Nachdem sich mit den Beispieldaten vertraut gemacht wurde, war der erste Schritt, auf fehlende Werte zu prüfen. Bei den Beispieldaten sind sämtliche Werte vorhanden. Dies liegt daran, dass es sich um eigens ausgewählte Daten handelt (siehe auch [26]).

Danach erfolgte auch schon die Abspaltung der Testdaten, wobei dazu 20% der Datensätze zufällig ausgewählt wurden. Die verbleibenden 80% der Datensätze wurden in weiterer Folge noch gemeinsam untersucht. Später müssen diese Datensätze dann noch in Trainings- und Validierungsdaten aufgespaltet werden.

Der nächste Schritt bestand darin, die Struktur der Beispieldaten (exlusive der Testdaten) zu untersuchen. Dazu wurden Boxplots für alle 64 Merkmale erstellt. Sämtliche Grafiken

[6] https://www.r-project.org/.

an dieser Stelle zu zeigen wäre zu umfangreich. Stellvertretend zeigt Abbildung 6 den Boxplot für das erste Merkmal (net profit / total assets).

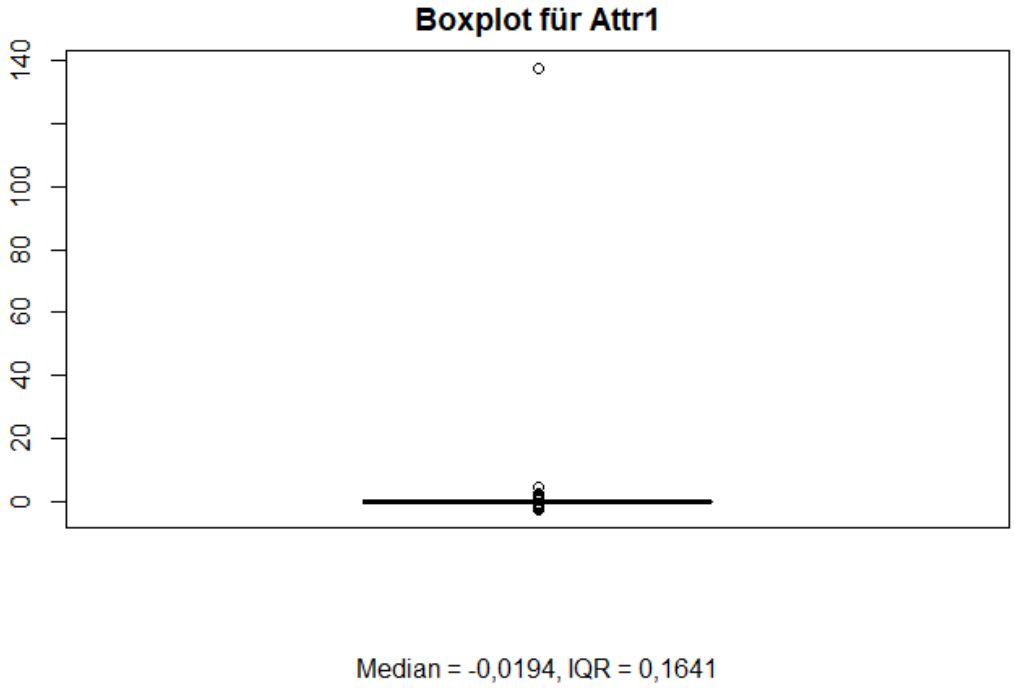

Abbildung 6: Der Boxplot für das erste Merkmal der Beispieldaten (net profit / total assets). Quelle: Eigene Darstellung.

Abbildung 6 ist in der Tat typisch für die meisten Merkmale der Beispieldaten. Diese weisen nämlich häufig einen Median in der Nähe von 0, einen geringen Quartilsabstand (IQR) und wenige, aber teils starke, Ausreißer auf. Man könnte auf den ersten Blick denken, dass das 29. Merkmal eine seltene Ausnahmen ist. Tatsächlich hat dieses Merkmal, wenn man die Beispieldaten standardisiert, die geringste Spannweite. Allerdings handelt es sich hierbei um logarithmierte Werte, wie man der Beschreibung in Appendix A entnehmen kann. Dadurch wurden die größten Ausreißer gedämpft. Dementsprechend ist Abbildung 7, die den Boxplot für das 29. Merkmal zeigt, mit Vorsicht zu genießen.

Teilt man die Trainingsdaten in jene Datensätze, die in Konkurs gegangen sind, und jene, die nicht in Konkurs gegangen sind, und untersucht diese Teilmengen getrennt voneinander, so erkennt man, dass die markantesten Ausreißer nicht von den wenigen in Konkurs gegangenen Unternehmen stammen. Daher könnte es gerechtfertigt sein, die Ausreißer bei den Trainingsdaten zu entfernen, um dadurch eventuell stabilere Modelle zu enthalten. Darauf wurde allerdings zunächst verzichtet.

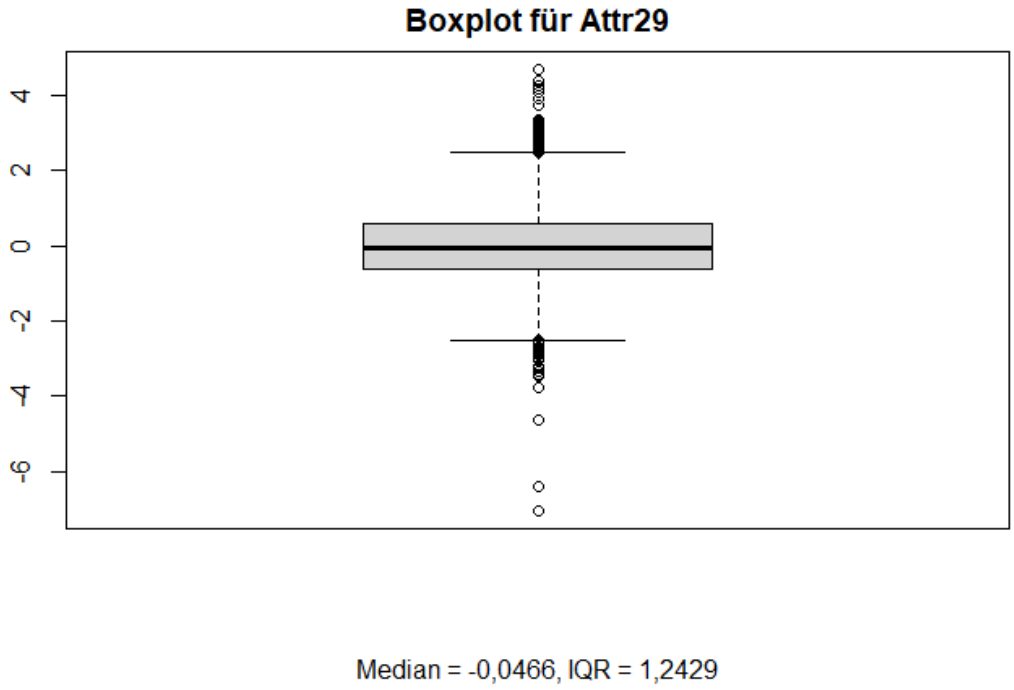

Abbildung 7: Der Boxplot für das 29. Merkmal der Beispieldaten (logarithm of total assets). Quelle: Eigene Darstellung.

Nun erfolgte die finale Aufspaltung von Trainings- und Validierungsdaten. Dabei wurden nochmal 10% der Datensätze (gemessen an den insgesamt vorhandenen Datensäten) für die Validierungsdaten abgespalten.

Im nächsten Schritt wurden die Daten standardisiert (Trainings-, Validierungs- und Testdaten). Dabei wurden Stichprobenmittelwert und Stichprobenvarianz der Trainingsdaten ermittelt und sowohl Trainings- als auch Validierungs- und Testdaten mit diesen Werten skaliert. Danach wurde ein Binning-Algorithmus auf die Trainingsdaten angewendet (ein solcher ist - unter anderem - im R-Package „scorecard“ implementiert, siehe [46, S. 25ff]). Dabei wurden nur zwei Restriktionen vorgegeben: Jeder Bin enthält mindestens 5% der Werte und für jedes Merkmal werden maximal 8 Bins angelegt. Die zweite Restriktion wurde allerdings nur bei einem einzigen Merkmal ausgereizt. Nachdem die Merkmale in diskrete Merkmale transformiert wurden (Werte innerhalb eines Bins wurden durch das WOE ersetzt), konnte das Information Value berechnet werden. Die Ergebnisse sind in Abbildung 4 zu sehen. Diese Vorgehensweise bietet eine erste Möglichkeit, Modelle zu formulieren: betrachtet man die Korrelation der 6 Merkmale mit dem höchsten Information Value, so sind diese nur schwach bis mäßig korreliert (siehe Abbildung 8). Damit hat man eine erste Teilmenge an Merkmalen gefunden, auf Basis derer man Modelle kalibrieren kann.

	Attr15	Attr34	Attr26	Attr25	Attr16	Attr41
Attr15	1	−0.00374791652590927	−0.00505664337404008	−0.000426612561769378	−0.00519632563081575	0.00889608249633171
Attr34	−0.00374791652590927	1	0.349241211162545	0.00172595431937465	0.342378800343439	0.00210402320642447
Attr26	−0.00505664337404008	0.349241211162545	1	0.0643105806751596	0.994200813869383	−0.0153475547774786
Attr25	−0.000426612561769378	0.00172595431937465	0.0643105806751596	1	0.0895664367989413	−0.00652894651598289
Attr16	−0.00519632563081575	0.342378800343439	0.994200813869383	0.0895664367989413	1	−0.0152959628580622
Attr41	0.00889608249633171	0.00210402320642447	−0.0153475547774786	−0.00652894651598289	−0.0152959628580622	1

Abbildung 8: Korrelation der 6 Merkmale mit dem höchsten Information Value. Quelle: Eigene Darstellung.

Üblicherweise ist die Auswahl der Merkmale ein komplexerer Prozess, bei welchem unterschiedliche Methoden zum Einsatz kommen können (siehe Kapitel 2 und Kapitel 3). Allerdings ist es nicht das Ziel dieses Kapitels, ein bestmögliches Modell zu konstruieren, sondern die unterschiedlichen Modelle zu vergleichen. Dazu werden jedem der vier Modelle aus Kapitel 2 die gleichen Merkmale zur Verfügung gestellt. Daher ist es nicht erforderlich, eine ideale Merkmalsauswahl vorzunehmen.

Im folgenden Kapitel 4.2 werden Möglichkeiten vorgestellt, wie die Performance der Modelle evaluiert werden kann. Die Performance der Modelle auf Basis der standardisierten Daten ist in Kapitel 4.3.1 zu sehen. Dabei wird sich zeigen, dass die Logistische Regression, die Bayessche Logistische Regression und das Neural Network eine Unterscheidung zwischen den Klassen vornehmen können, jedoch nur bei Grenzwahrscheinlichkeiten in der Nähe von 0. Die SVM kann hingegen keinen einzigen in Konkurs gegangenen Datensatz erkennen. Deswegen wurden weitere Möglichkeiten ausgelotet, diese Resultate zu verbessern. Es ist denkbar, dass die Modelle - heuristisch gesprochen - durch die Ausreißer in der Kalibrierung fehlgeleitet wurden. Der erste Ansatz könnte also darin bestehen, die gröbsten Ausreißer zu entfernen. Dazu sind allerdings auch einige Fragen zu beantworten: wann wird ein Datenpunkt als Ausreißer angesehen? Standardisiert man die Daten trotzdem? Wenn man die Daten standardisiert, dann bevor oder nachdem man die Ausreißer entfernt hat? Auf die erste Frage gibt es mehrere mögliche Antworten. Tukey bezeichnet beispielsweise Werte, die mehr als den eineinhalbfachen Interquartilabstand kleiner als das erste Quartil beziehungsweise größer als das dritte Quartil sind, als „outside“. Werte, die mehr als den dreifachen Interquartilabstand kleiner als das erste Quartil beziehungsweise größer als das dritte Quartil sind, bezeichnet er hingegen als „far out“. [43, S. 43f] Damit hätte man bereits zwei Möglichkeiten, wie man Ausreißer definieren könnte. Eine weitere Möglichkeit besteht darin, jene Werte als Ausreißer zu definieren, die mindestens ein bestimmtes Vielfaches der Standardabweichung vom Mittelwert entfernt sind. [35, S. 44f] Diese Vorgangsweise erscheint beispielsweise bei normalverteilten Daten sinnvoll. Angenommen man wüsste hingegen, dass die Daten stetig gleichverteilt sind, so würde man mit dieser Methode Werte als Ausreißer deklarieren, die völlig legitim sind.

Bei dem Beispiel der stetigen Gleichverteilung könnte man vermutlich überhaupt nur jene Werte als Ausreißer klassifzieren, die außerhalb des Intervalls der möglichen Werte liegen. Bei jenen Werten muss es sich nämlich folgerichtig um fehlerhafte Datensätze handeln.

Messfehler sind also eine mögliche Quelle für Ausreißer. Puristisch könnte man also auch nur Werte, die durch Messfehler entstanden sind, als Ausreißer definieren und entfernen. [35, S. 44ff] Jedoch wird die Identifizierung dadurch komplizierter. Pollet und van der Meij raten als Resultat ihrer Studie explizit zur Vorsicht, wenn man Ausreißer entfernen möchte. Sie haben nämlich herausgefunden, dass dies einerseits die Schlussfolgerungen von statistischen Tests verändern kann und andererseits eine Möglichkeit bietet, die Resultate aktiv zu beeinflussen, indem man die passende Methode zur Entfernung der Ausreißer wählt. [35, S. 44ff]

In dieser Arbeit soll überprüft werden, ob eine simple Methode zur Entfernung der Ausreißer zu besseren Modellen führt. Dafür werden jene Werte entfernt, die Tukey als „far out“ bezeichnet. Betrachtet werden jedoch nur jene Merkmale, die tatsächlich in den Modellen verwendet werden. Die Standardisierung der Daten wird nach der Entfernung der Ausreißer vorgenommen. Zudem sei angemerkt, dass Ausreißer stets nur bei den Trainingsdaten entfernt werden, da die Testdaten ja möglichst eine reale Situation widerspiegeln sollen.

Das Entfernen der Ausreißer hat teilweise zu Verbesserungen geführt, welche allerdings bei der SVM nur sehr gering waren und bei dem Neural Network wurde die Performance sogar schlechter (siehe Kapitel 4.3.2). Möglicherweise wurde die Performance nicht primär von den Ausreißern beeinträchtigt, sondern von den unausgewogenen Daten (nur etwa 2 % in Konkurs gegangen). Um dem entgegenzuwirken kann der SMOTE-Algorithmus verwendet werden (siehe auch Kapitel 3.6). Die Implementierung des Algorithmuses im Paket „performanceEstimation“ ermöglicht gleichzeitig Undersampling und das Generieren von synthetischen Datensätzen. Es wurde entschieden, vier synthetische Datensätze pro in Konkurs gegangenen Datensatz zu generieren. Das Undersampling funktioniert bei dieser Implementierung folgendermaßen: es wird festgelegt, wie viele Datensätze aus der großen Klasse pro synthetischen Datensatz ausgewählt werden. [42, S.67] In diesem Fall wurde zwei gewählt. Dadurch ergaben sich neue Trainingsdaten, welche in Tabelle 5 zusammengefasst werden. Offensichtlich ist das Verhältnis zwischen den Klassen wesentlich ausgeglichener in den neuen Trainingsdaten.

Nicht in Konkurs gegangen	In Konkurs gegangen	Gesamt
1136	710	1846

Tabelle 5: Neue Trainingsdaten, generiert durch SMOTE. Quelle: Eigene Darstellung.

Die Ergebnisse der Modelle mit den neuen Trainingsdaten sind in Kapitel 4.3.3 zu sehen.

4.2 Performance-Maße

Um die Performance der Modelle vergleichen zu können, muss man diese messen können. Zu diesem Zweck gibt es in der Literatur verschiedene Performance-Maße. Einige dieser Performance-Maße sollen in diesem Unterkapitel vorgestellt werden. Ausgangspunkt dafür bildet stets die sogenannte Konfusionsmatrix (siehe Tabelle 6).

		Echte Klasse	
		0	1
Vorhergesagte Klasse	0	# true negatives	# false negatives
	1	# false positives	# true positives

Tabelle 6: Schema einer Konfusionsmatrix. 0 steht für nicht in Konkurs gegangen, 1 für in Konkurs gegangen. Quelle: Eigene Darstellung basierend auf [9, S. 862, Abb. 1].

Eine Konfusionsmatrix ist (im Falle von binären Klassifizierungsproblemen) einfach eine 2x2-Matrix, die die Häufigkeiten der vier möglichen Ergebnisse bei der Vorhersage enthält. Wie in Tabelle 6 dargestellt, kann das Modell entweder richtig vorhersagen, dass ein Unternehmen in Konkurs geht (*true positive*) beziehungsweise nicht in Konkurs geht (*true negative*) oder es liegt einer von zwei möglichen Fehlern vor. Wenn kein Konkurs vorhergesagt wurde, das Unternehmen allerdings tatsächlich in Konkurs gegangen ist, dann handelt es sich um einen *false negative*. Umgekehrt, wenn das Unternehmen nicht in Konkurs gegangen ist, das Modell allerdings dies vorhergesagt hat, dann handelt es sich um einen *false positive*. [9, S. 862] Standardmäßig unterscheiden die Modelle nicht zwischen diesen beiden Fehlertypen (siehe Kapitel 2), da die Fehlerfunktionen symmetrisch sind. In der Praxis kann es allerdings durchaus sinnvoll sein, diese Unterscheidung vorzunehmen. Man stelle sich beispielsweise eine Anlegerin vor, die anhand des Modells entscheidet, welche Anleihen sie kauft. Sagt das Modell fälschlicherweise einen Konkurs vorher, so lässt sie sich ein Geschäft entgehen. Schlägt das Modell allerdings nicht Alarm

und das Unternehmen geht in Konkurs, so erleidet sie einen Verlust. In diesem Beispiel haben false negatives also gravierendere Folgen.

Von der Konfusionsmatrix werden nun einige Performance-Maße abgeleitet. Dazu seien # total positves := # true positives + # false negatives und # total negatives := # true negatives + # false positives.

$$\text{false positive rate} := \frac{\#\text{ false positives}}{\#\text{ total negatives}} \tag{4.1}$$

$$\text{recall} = \text{true positive rate} := \frac{\#\text{ true positives}}{\#\text{ total positves}} \tag{4.2}$$

$$\text{precision} := \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}} \tag{4.3}$$

$$\text{accuracy} := \frac{\#\text{ true positives} + \#\text{ true negatives}}{\#\text{ total positves} + \#\text{ total negatives}} \tag{4.4}$$

$$\text{F-Maß} := \frac{2}{1/\text{precision} + 1/\text{recall}} \tag{4.5}$$

[9, S. 862]

Eine Möglichkeit, die Performance der Modelle zu visualisieren, bieten die sogenannten *Receiver-Operating-Characteristics*-Graphen (kurz: ROC-Graphen). Dabei wird die true positive rate auf der Y-Achse aufgetragen und die false positive rate auf der X-Achse. Abbildung 9 zeigt einen ROC-Graphen mit fünf Schätzern (Modellen). Dabei ist ein Schätzer echt besser als ein anderer, wenn er sich weiter links und oben im Graphen befindet, da er dann eine bessere true positive rate und eine bessere false positive rate aufweist. In der Abbildung ist der Schätzer D perfekt. Liegt ein Schätzer hingegen auf der Diagonalen, so entspricht seine Performance einem Schätzer, welcher die Einflussvariablen ignoriert und einfach bei einem festen Anteil der Daten einen Konkurs vorhersagt. Sagt ein Schätzer beispielsweise für jeden zweiten Datensatz einen Konkurs vorher, dann wird er im Mittel 50% der in Konkurs gegangen und 50% der nicht in Konkurs gegangen Unternehmen korrekt vorhersagen. Dies entspricht dem Punkt (0,5; 0,5) im ROC-Graphen. Variiert man den Anteil, mit dem dieser Schätzer einen Konkurs vorhersagt, so verschiebt sich der Punkt im ROC-Graphen entlang der Diagonale. Unterhalb der Diagonale (zum Beispiel Schätzer E) sollten üblicherweise keine Schätzer liegen. Vertauscht man nämlich für solche Schätzer die Vorhersagen (nicht in Konkurs gegangen wird als in Konkurs gegangen vorhergesagt und vice versa) so würden diese über der Diagonale liegen. [9, S. 862f]

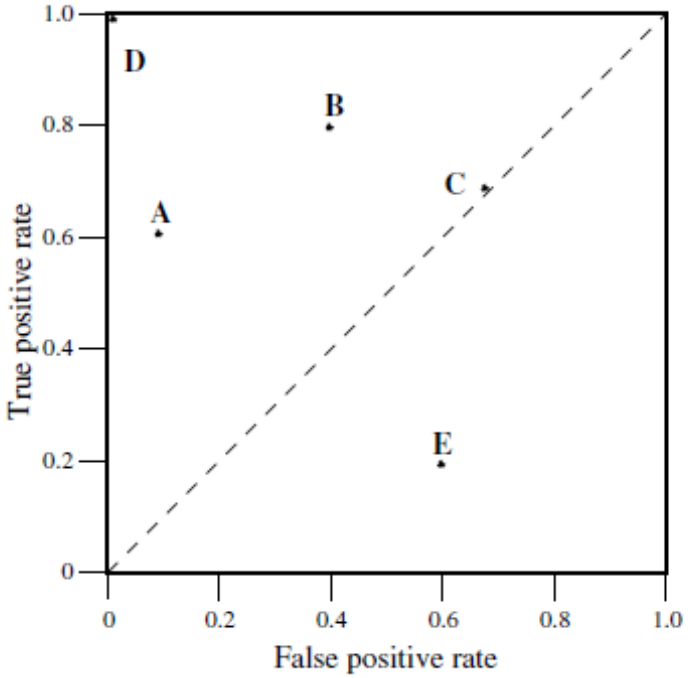

Abbildung 9: Beispiel für einen ROC-Graphen. Quelle: [9, S. 862, Abb. 2].

Einige Modelle liefern nur zurück, welcher Klasse ein Datensatz zuzuordnen ist (Support Vector Machine). Andere Modelle sind hingegen in der Lage, die Wahrscheinlichkeit zu schätzen, mit welcher ein Datensatz zu einer Klasse gehört (Logistische Regression, Bayessche Logistische Regression und Neural Networks). Die Entscheidung, welcher Klasse ein Datensatz zugeordnet wird, fällt dann anhand gewisser Grenzen (beispielsweise könnten Datensätze mit Konkurswahrscheinlichkeit größer 0,5 der Klasse „in Konkurs gegangen“ zugeordnet werden). Indem man diese Grenzen variiert erhält man, statt einem einzelnen Punkt, eine Kurve, nämlich die ROC-Kurve. [9, S. 863] Dadurch kann man untersuchen, wie sich die Performance des Modells mit der Grenze verändert. Zusätzlich kann man die Performance verschiedener Modelle anhand ihrer ROC-Kurven vergleichen.

Aus der ROC-Kurve kann auch ein weiteres skalares Performance-Maß gewonnen werden. Dabei handelt es sich um die *Area under an ROC curve* (AUC). Seinem Namen entsprechend handelt es sich bei diesem Performance-Maß einfach um die Fläche unter der ROC-Kurve. Dieses Performance-Maß kann maximal den Wert 1 annehmen und sollte stets über 0,5 betragen, da ja bereits der Schätzer, der einen festen Anteil der Daten als Konkurs vorhersagt, eine AUC von 0,5 erreicht. [9, S. 868]

4.3 Resultate

Nachdem die Daten vorbereitet und die Merkmale, die verwendet werden sollen, ausgewählt wurden, können die Modelle kalibriert werden. Die nötigen Funktionalitäten sind bereits direkt in R implementiert (Logistische Regression) oder stehen in verschiedenen Paketen zur Verfügung. Für diese Arbeit wurden die Pakete „rstanarm“ (Bayessche Logistische Regression), „neuralnet“ (Neural Networks) und „e1071“ (Support Vector Machine) verwendet. [12] [11] [28] Nach der Kalibrierung wurden die Trainingsdaten und die Validierungsdaten anhand der jeweiligen Modelle vorhergesagt. Im Folgenden werden die kalibrierten Modelle und ihre jeweilige Performance gezeigt. Die natürliche Vorgehensweise die Performance zu analysieren, wäre von der Konfusionsmatrix auszugehen, allerdings ist diese abhängig von der Wahrscheinlichkeit, die für die Abgrenzung der Klassen gewählt wurde. Da a priori nicht klar ist, welche Wahrscheinlichkeit gewählt werden sollte, muss dies zunächst analysiert werden. Dazu können die ROC-Graphen verwendet werden. Diese werden bei den ersten zwei Durchläufen zeigen, dass die Modelle besonders gut performen, wenn die Wahrscheinlichkeit sehr klein gewählt wird. Daher werden bevorzugt die ROC-Graphen gezeigt, wenn dies möglich ist.

4.3.1 Standardisierte Daten

Für den ersten Durchlauf wurden die Daten nur standardisiert (siehe Kapitel 4.1). Die ersten gezeigten Resultate stammen von der Logistischen Regression. Die Kalibrierung lieferte die folgenden Koeffizienten:

Merkmal	Koeffizient
Konstante	-4,26031
Attr34	0,37431
Attr26	0,24855
Attr25	-1,11153
Attr41	-0,03256
Attr15	0,01462
Attr16	-1,35762

Tabelle 7: Koeffizienten der Logistischen Regression. Quelle: Eigene Darstellung.

Die beiden folgenden Abbildungen zeigen die ROC-Graphen der Logistischen Regression, einmal auf den Trainingsdaten (Abbildung 10) und dann auf den Validierungsdaten (Abbildung 11). Auffällig ist dabei, dass das Modell nur dann eine sinnvolle Unterteilung in die Klassen liefert, wenn man die Wahrscheinlichkeit zur Abgrenzung sehr klein wählt. Dies ist anhand der Farbe der Kurve ersichtlich, die einen groben Überblick gibt, bei welcher Wahrscheinlichkeit die entsprechenden true positive bzw. false positive rates erzielt wurden (siehe Farbskala am rechten Rand der Abbildungen).

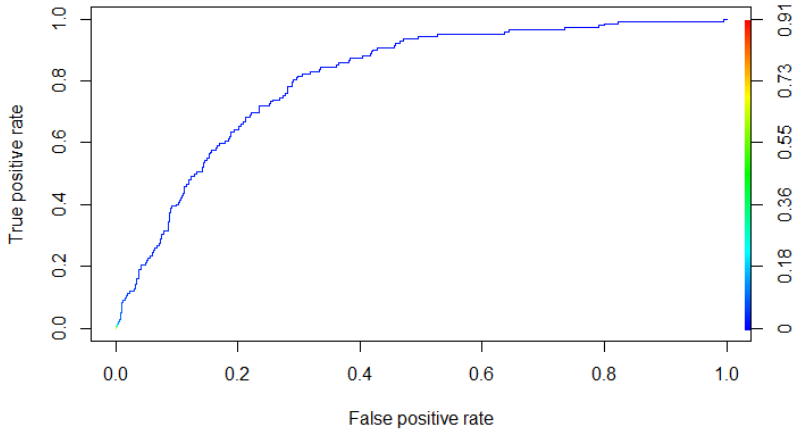

Abbildung 10: ROC-Graph der Logistischen Regression auf den Trainingsdaten. AUC: 0,8099. Quelle: Eigene Darstellung.

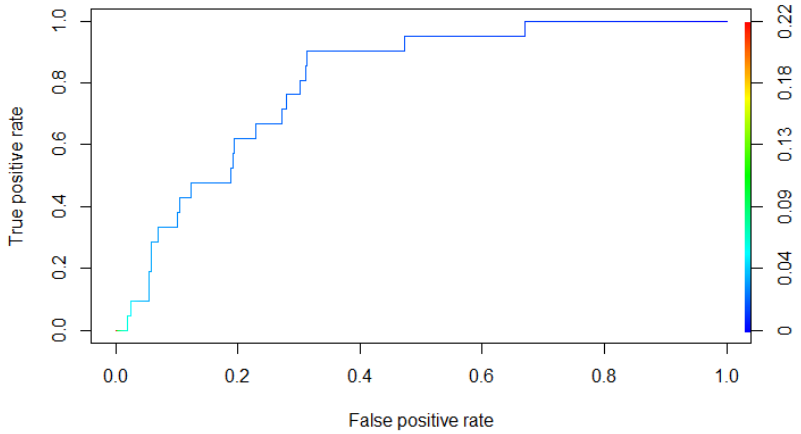

Abbildung 11: ROC-Graph der Logistischen Regression auf den Validierungsdaten. AUC: 0,8055. Quelle: Eigene Darstellung.

Das nächste Modell ist die Bayessche Logistische Regression. Bei diesem Modell muss zunächst eine a priori Verteilung für die Koeffizienten festgelegt werden. Diese wurde wie

folgt gewählt:

$$\beta_i \sim \text{Student-t}(\text{df} = 7, \mu = 0, \sigma = 2{,}5) \quad i = 0, \ldots, 6. \tag{4.6}$$

Die t-Verteilung hat den Vorteil, dass sie eher extremere Werte für die Koeffizienten zulässt als beispielsweise die Normalverteilung. Dies kann dazu beitragen, ein gutes Modell zu generieren. [14, S. 437] Auch Mukeri, Shaikh und Gaikwad verwenden die t-Verteilung für die a priori Verteilung. [29, S. 8]

Die Bayessche Logistische Regression liefert kein deterministisches Modell, sondern generiert Stichproben der a posteriori Verteilung der Koeffizienten (siehe Kapitel 2.1.2). Eine Zusammenfassung des Modells wie in Tabelle 7 für die Logistische Regression ist also nicht möglich. Allerdings können der Stichprobenerwartungswert und die Stichprobenstandardabweichung der Koeffizienten berechnet werden, welche einen Eindruck vom Modell liefern. Diese sind in Tabelle 8 zu sehen.

Merkmal	Erwartungswert	Standardabweichung
Konstante	-4,20008	0,10589
Attr34	0,35141	0,05963
Attr26	-0,44923	1,19134
Attr25	-0,86805	0,22480
Attr41	-0,00613	0,05174
Attr15	-0,00392	0,04951
Attr16	-0,57021	1,23862

Tabelle 8: Koeffizienten der Bayesschen Logistischen Regression. Quelle: Eigene Darstellung.

Vergleicht man die Erwartungswerte der Koeffizienten aus Tabelle 8 mit den Koeffizienten aus 7, so gibt es einige Unterschiede. Besonders auffällig sind die unterschiedlichen Vorzeichen (zum Beispiel Attr26), da sich mit dem Vorzeichen auch die Richtung ändert, in welche die Merkmale die Konkurswahrscheinlichkeit (im Mittel) beeinflussen. Auch bei der Bayesschen Logistischen Regression findet nur bei sehr geringen Wahrscheinlichkeiten eine Abgrenzung in die Klassen statt (siehe Abbildung 12 und Abbildung 13). Auch die Performance ist ähnlich wie bei der herkömmlichen Logistischen Regression, allerdings ist die AUC etwas schlechter.

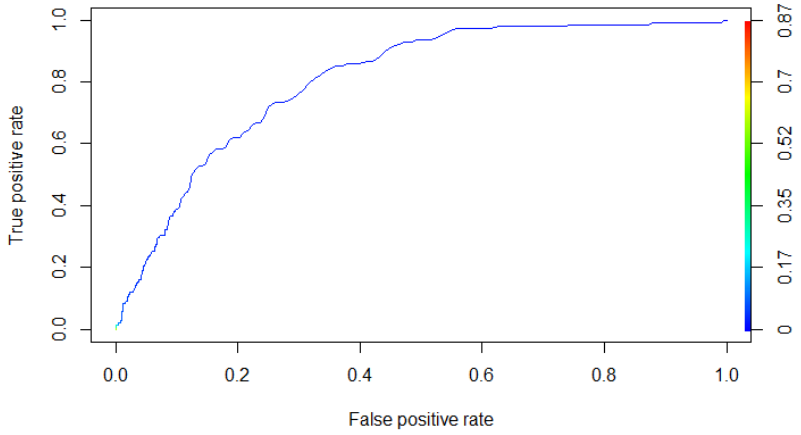

Abbildung 12: ROC-Graph der Bayesschen Logistischen Regression auf den Trainingsdaten. AUC: 0,8077. Quelle: Eigene Darstellung.

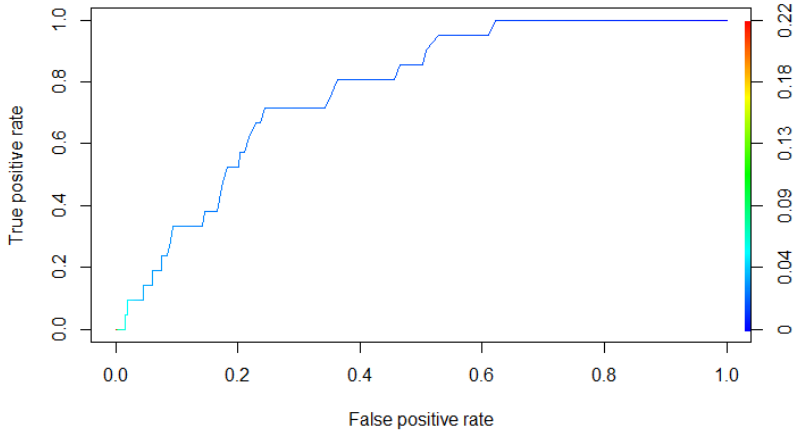

Abbildung 13: ROC-Graph der Bayesschen Logistischen Regression auf den Validierungsdaten. AUC: 0,7741. Quelle: Eigene Darstellung.

Das dritte Modell ist das Neural Network. Es wurden Neural Networks mit verschieden vielen Hidden Layern und verschieden vielen Knoten in den Hidden Layern getestet und auf den Validierungsdaten verglichen. Die besten Ergebnisse hat ein Neural Network mit einem Hidden Layer und 4 Knoten geliefert. (Kurzschreibweise (4)). Dieses Modell ist in Abbildung 14 zu sehen.

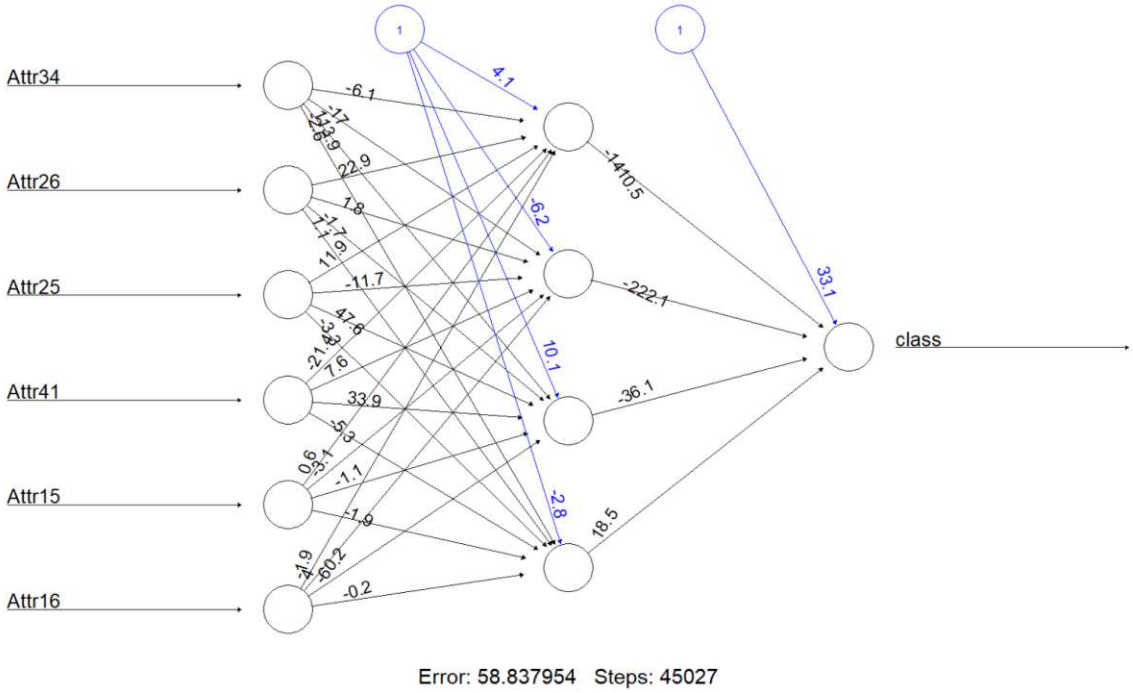

Abbildung 14: Graph des Neural Networks. Quelle: Eigene Darstellung.

Abbildung 15 und Abbildung 16 zeigen die ROC-Graphen des Neural Networks auf den Trainings- beziehungsweise den Validierungsdaten. Das Neural Network performt besser auf den Trainingsdaten als die Logistische Regression und die Bayessche Logistische Regression. Dies ist wohl mit der hohen Anpassungsfähigkeit der Neural Networks zu erklären (siehe Kapitel 2.2). Allerdings ist die Performance auf den Validierungsdaten sogar noch besser. Ein weiterer Unterschied besteht darin, dass auch bereits eine Unterteilung in die Klassen stattfindet, wenn die Grenzwahrscheinlichkeit nicht sehr nahe bei 0 ist. Allerdings werden dann auch beim Neural Network die meisten Datensätze als „nicht in Konkurs gegangen“ eingestuft.

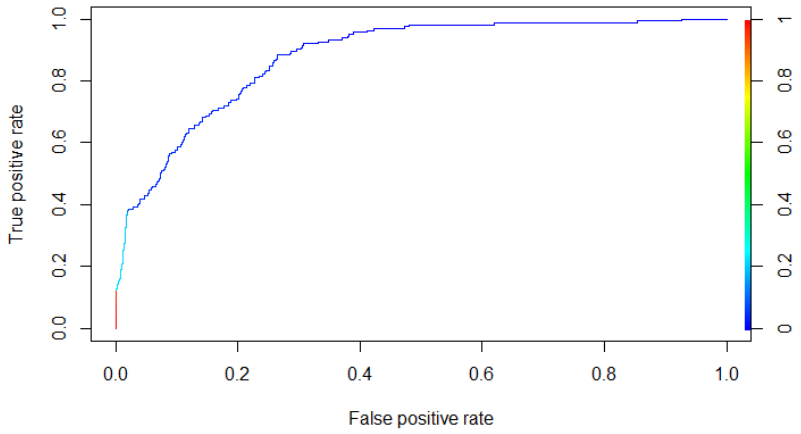

Abbildung 15: ROC-Graph des Neural Networks auf den Trainingsdaten. AUC: 0,8528. Quelle: Eigene Darstellung.

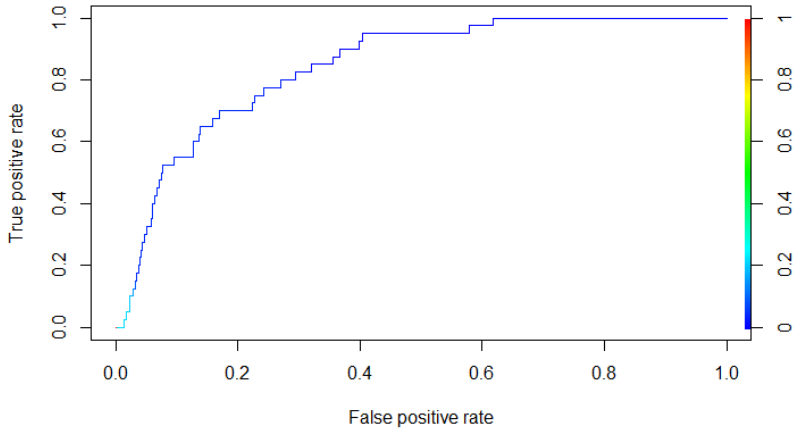

Abbildung 16: ROC-Graph des Neural Networks auf den Validierungsdaten. AUC: 0,8774. Quelle: Eigene Darstellung.

Das letzte Modell ist die SVM. Dabei wurde die radiale Basisfunktion als Kernel gewählt (siehe (2.63)). Eine kompakte Darstellung des kalibrierten Modells wie bei den drei vorangegangenen Modellen ist in diesem Fall nicht möglich. Das liegt daran, dass bei der Lösung des Optimierungsproblems die Support Vectors und ihre entsprechenden Koeffizienten bestimmt werden, woraus sich die Klassifizierungsregel ergibt (siehe (2.60)). Bei den vorliegenden Daten wurden insgesamt 491 Support Vectors bestimmt. Eine Auflistung jener Support Vectors und ihrer Koeffizienten erscheint nicht sinnvoll. Darüber hinaus liefert die SVM eine binäre Entscheidung, die nicht von einem zusätzlichen Metaparameter (der Ausfallswahrscheinlichkeit) abhängt. Daher können an dieser Stelle auch nicht die ROC-Graphen gezeigt werden. Stattdessen wird in Tabelle 9 die Konfusionsmatrix für die Trainingsdaten beziehungsweise in Tabelle 10 die Konfusionsmatrix für die

Testdaten gezeigt. Daraus geht hervor, dass die SVM keinen einzigen in Konkurs gegangegen Datensatz richtig identifiziert.

		Echte Klasse: 0	Echte Klasse: 1
Vorhergesagte Klasse	0	6858	142
	1	0	0

Tabelle 9: Konfusionsmatrix der SVM auf den Trainingsdaten. Quelle: Eigene Darstellung.

		Echte Klasse: 0	Echte Klasse: 1
Vorhergesagte Klasse	0	979	21
	1	0	0

Tabelle 10: Konfusionsmatrix der SVM auf den Validierungsdaten. Quelle: Eigene Darstellung.

4.3.2 Daten ohne Ausreißer

Für den zweiten Durchlauf wurden zuerst Ausreißer entfernt und dann die Daten standardisiert (siehe auch Kapitel 4.1). Den Anfang macht erneut die Logistische Regression. Tabelle 11 zeigt die Koeffizienten des kalibrierten Modells. Besonders interessant sind nun die Unterschiede, die die veränderten Trainingsdaten hervorgerufen haben. Vergleicht man die Koeffizienten mit jenen aus Tabelle 7, so erkennt man, dass sich diese teilweise recht deutlich unterscheiden. Besonders bemerkenswert ist, dass die Koeffizienten von Attr26 und Attr16 das Vorzeichen gewechselt haben. Damit ändert sich nämlich auch die Richtung, in welche die Merkmale die Ausfallswahrscheinlichkeit beeinflussen.

Merkmal	Koeffizient
Konstante	-4,71214
Attr34	0,45995
Attr26	-2,30491
Attr25	-0,49016
Attr41	-0,13653
Attr15	1,88534
Attr16	1,53874

Tabelle 11: Koeffizienten der Logistischen Regression (Ausreißer entfernt). Quelle: Eigene Darstellung.

Die ROC-Graphen in Abbildung 17 und Abbildung 18 zeigen erneut, dass die Unterteilung in Klassen nur bei sehr geringen Grenzwahrscheinlichkeiten funktioniert. Allerdings konnte die Performance (gemessen an der AUC) durch das Entfernen der Ausreißer gesteigert werden. Auf den Trainingsdaten ist das nicht so verwunderlich, da das Modell sich ja an Daten ohne Ausreißer leichter anpassen können sollte. Auf den Validierungsdaten wurden die Ausreißer jedoch nicht entfernt und auch dort wurde die Performance gesteigert.

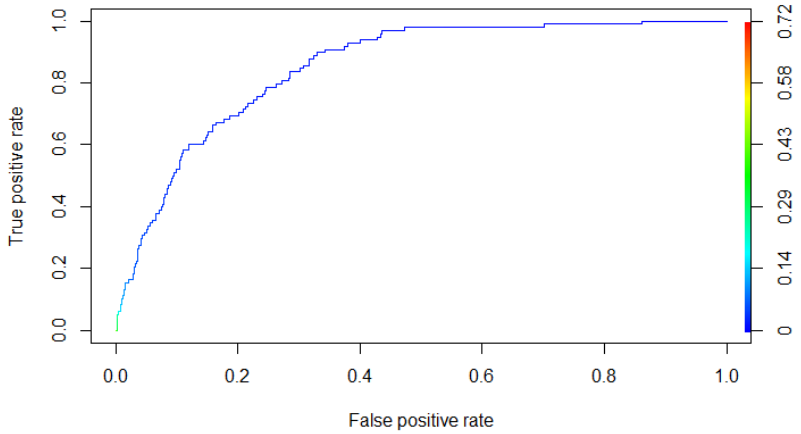

Abbildung 17: ROC-Graph der Logistischen Regression auf den Trainingsdaten (Ausreißer entfernt). AUC: 0,8503. Quelle: Eigene Darstellung.

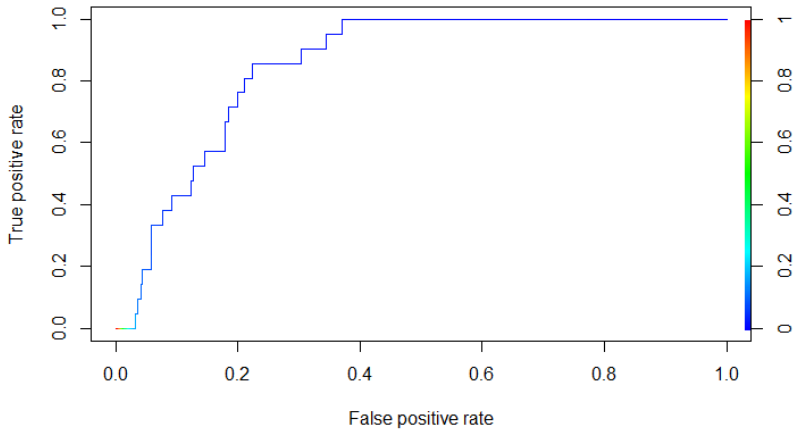

Abbildung 18: ROC-Graph der Logistischen Regression auf den Validierungsdaten (Ausreißer entfernt). AUC: 0,8528. Quelle: Eigene Darstellung.

Auch bei der Bayesschen Logistischen Regression haben sich Vorzeichen geändert (siehe Tabelle 12), nämlich jene von Attr15 und Attr16. Bei den Daten ohne Ausreißer stimmen die Vorzeichen der Logistischen und der Bayesschen Logistischen Regression überein.

Merkmal	Erwartungswert	Standardabweichung
Konstante	-4,60635	0,14071
Attr34	0.41240	0.09290
Attr26	-2,00874	0,66129
Attr25	-0,45405	0,08823
Attr41	-0,07799	0,08037
Attr15	0,47074	0,38174
Attr16	1,29134	0,67367

Tabelle 12: Koeffizienten der Bayesschen Logistischen Regression (Ausreißer entfernt). Quelle: Eigene Darstellung.

Die Performance der Bayesschen Logistischen Regression liegt erneut etwas unter jener der Logistischen Regression (siehe Abbildungen 19 und 20), konnte allerdings im Vergleich zum ersten Durchlauf gesteigert werden.

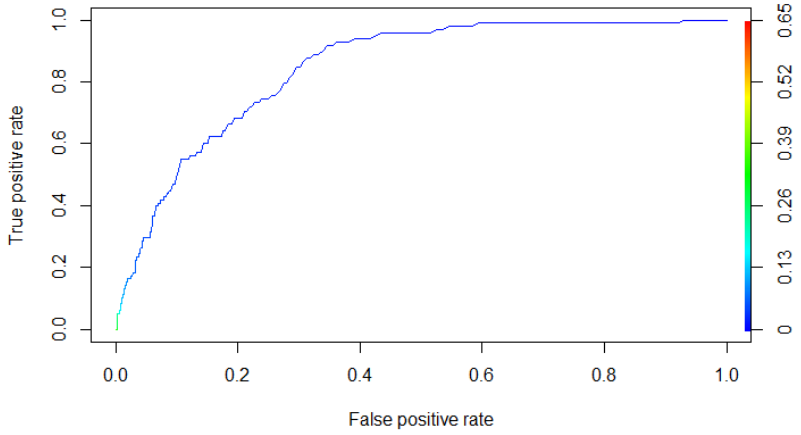

Abbildung 19: ROC-Graph der Bayesschen Logistischen Regression auf den Trainingsdaten (Ausreißer entfernt). AUC: 0,8450. Quelle: Eigene Darstellung.

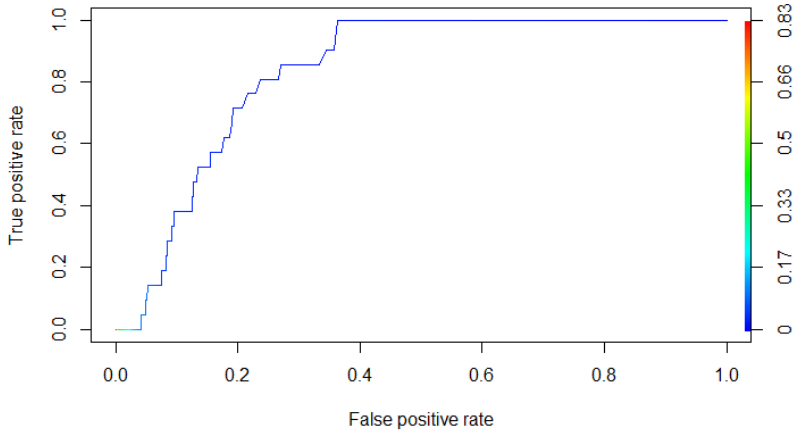

Abbildung 20: ROC-Graph der Bayesschen Logistischen Regression auf den Validierungsdaten (Ausreißer entfernt). AUC: 0,8365. Quelle: Eigene Darstellung.

Erneut lieferte ein Neural Network mit nur einem Hidden-Layer die besten Ergebnisse, diesmal jedoch mit 5 Knoten. Dieses ist in Abbildung 21 zu sehen.

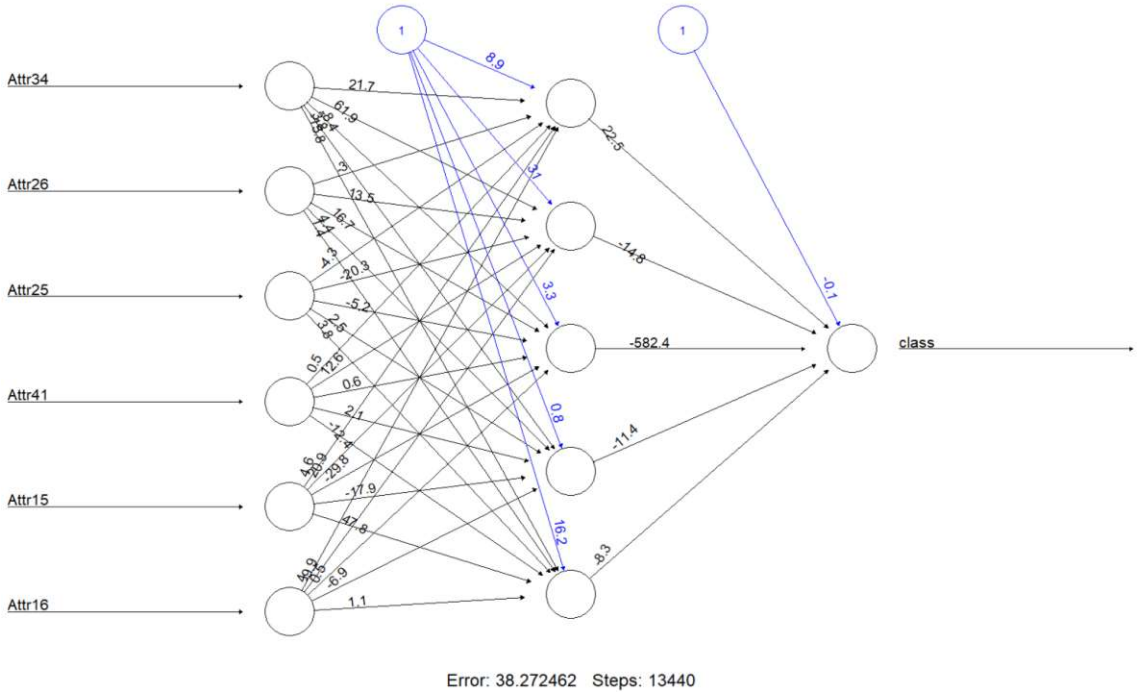

Abbildung 21: Graph des Neural Networks (Ausreißer entfernt). Quelle: Eigene Darstellung.

Im Gegensatz zur Bayesschen Logistischen Regression und zur Logistischen Regression hat die Performance des Neural Networks leicht abgenommen im Vergleich zum ersten Durchlauf (siehe Abbildungen 22 und 23).

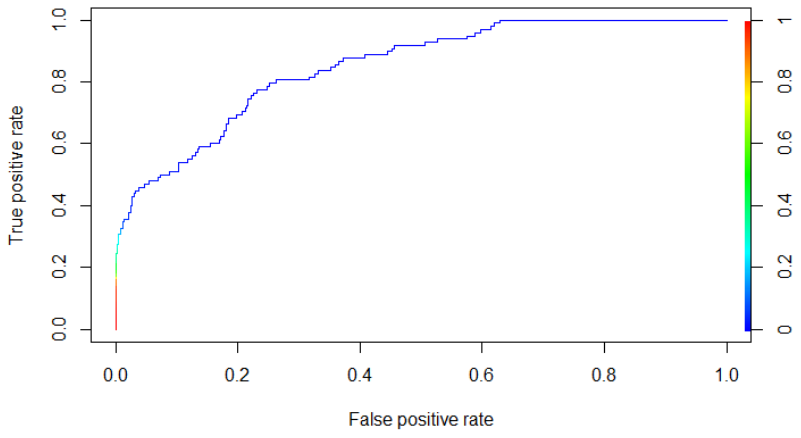

Abbildung 22: ROC-Graph des Neural Networks auf den Trainingsdaten (Ausreißer entfernt). AUC: 0,8495. Quelle: Eigene Darstellung.

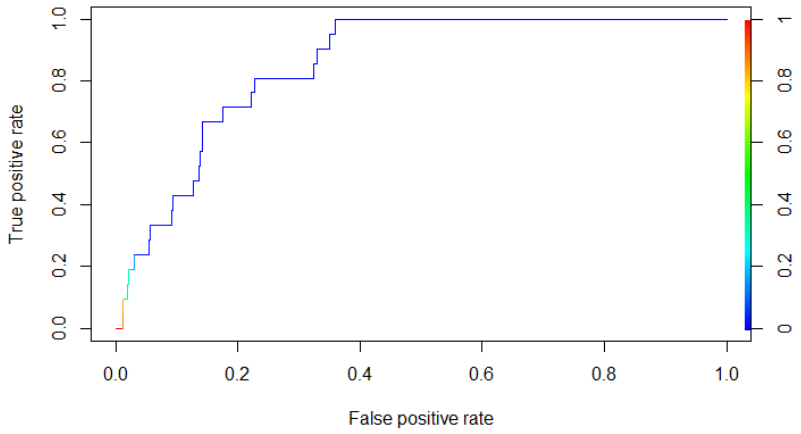

Abbildung 23: ROC-Graph des Neural Networks auf den Validierungsdaten (Ausreißer entfernt). AUC: 0,8544. Quelle: Eigene Darstellung.

Die SVM besteht bei diesem Durchlauf aus 365 Support Vectors. Im Vergleich zum ersten Durchlauf wurde die Performance auf den Trainingsdaten deutlich gesteigert (siehe Tabelle 13). Allerdings wurde trotzdem die Mehrheit der Konkurse nicht erkannt. Auf den Validierungsdaten wurde auch diesmal kein einziger Konkurs erkannt (siehe Abbildung 14).

		Echte Klasse: 0	Echte Klasse: 1
Vorhergesagte Klasse	0	5524	76
	1	3	22

Tabelle 13: Konfusionsmatrix der SVM auf den Trainingsdaten (Ausreißer entfernt). Quelle: Eigene Darstellung.

		Echte Klasse: 0	Echte Klasse: 1
Vorhergesagte Klasse	0	966	21
	1	13	0

Tabelle 14: Konfusionsmatrix der SVM auf den Validierungsdaten (Ausreißer entfernt). Quelle: Eigene Darstellung.

4.3.3 SMOTE

Für den dritten Durchlauf wurden ausgeglichenere Trainingsdaten mit Hilfe von SMOTE synthetisiert (siehe Kapitel 3.6 und 4.1). Diese wurden dann, wie auch in den vorangeganenen Durchläufen, standardisiert. Wie üblich werden die Resultate der Logistischen Regression als erstes präsentiert. Tabelle 15 zeigt die Koeffizienten des Modells. Besonders auffällig ist, dass die Konstante deutlich näher bei 0 liegt. Dies wird sich auch bei den ROC-Graphen bemerkbar machen, da nun die Unterteilung nicht nur bei Grenzwahrscheinlichkeiten um 0 funktioniert.

Merkmal	Koeffizient
Konstante	-0,75988
Attr34	0,71625
Attr26	-2,48509
Attr25	-0,35890
Attr41	0,02107
Attr15	0,09872
Attr16	0,67116

Tabelle 15: Koeffizienten der Logistischen Regression (SMOTE). Quelle: Eigene Darstellung.

Die Performance auf den Trainingsdaten ist nicht so gut wie im zweiten Durchlauf (siehe Abbildung 24). Abbildung 25 zeigt jedoch, dass die Performance auf den Validierungsdaten durchaus mithalten kann.

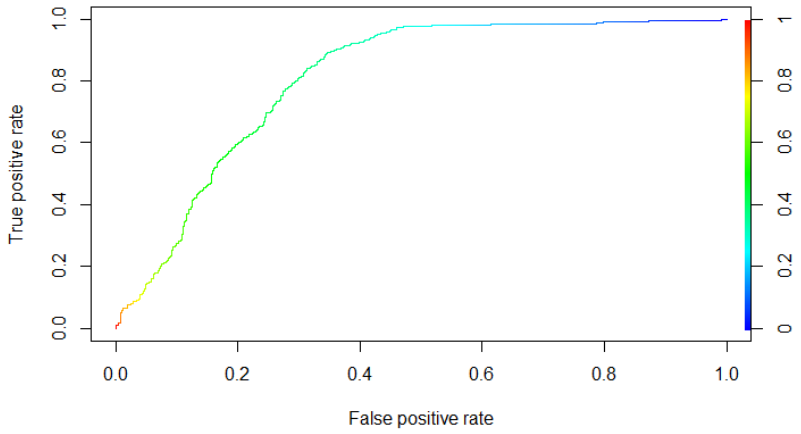

Abbildung 24: ROC-Graph der Logistischen Regression auf den Trainingsdaten (SMOTE). AUC: 0,8074. Quelle: Eigene Darstellung.

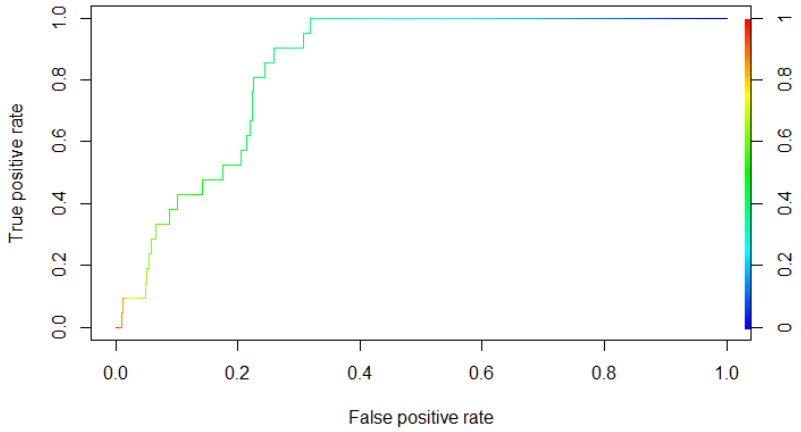

Abbildung 25: ROC-Graph der Logistischen Regression auf den Validierungsdaten (SMOTE). AUC: 0,8458. Quelle: Eigene Darstellung.

Die Bayessche Logistische Regression verhält sich (im Mittel) im dritten Durchlauf erneut ähnlich wie die herkömmliche Logistische Regression. Erwartungswert und Standardabweichung der Koeffizienten sind in Abbildung 16 aufgelistet.

Merkmal	Erwartungswert	Standardabweichung
Konstante	-0,74645	0,06173
Attr34	0,69496	0,06999
Attr26	-2,30870	0,78953
Attr25	-0,34856	0,06746
Attr41	0,02683	0,05331
Attr15	0,10338	0,06604
Attr16	0,55822	0,77505

Tabelle 16: Koeffizienten der Bayesschen Logistischen Regression (SMOTE). Quelle: Eigene Darstellung.

Auch bei der Performance sind die Logistische Regression und die Bayessche Logistische Regression sehr ähnlich. Vergleicht man Abbildung 26 und 19, so zeigt sich auch hier, dass die Performance auf den Trainingsdaten schlechter als im zweiten Durchlauf ist. Bei den Validierungsdaten war die Performance hingegen sogar besser (siehe Abbildung 27).

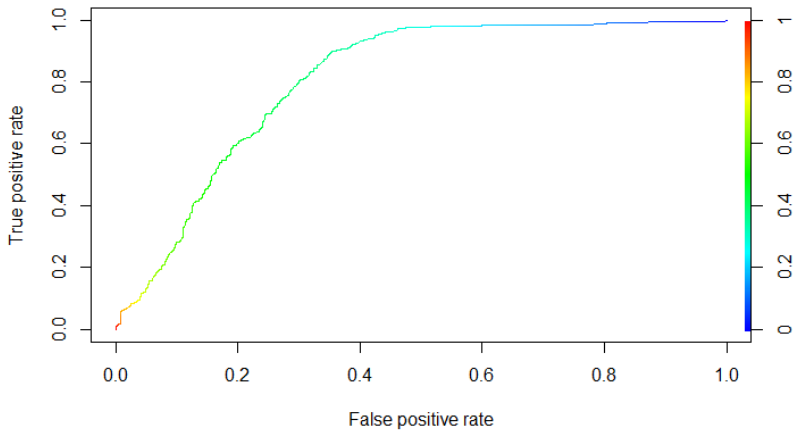

Abbildung 26: ROC-Graph der Bayesschen Logistischen Regression auf den Trainingsdaten (SMOTE). AUC: 0,8061. Quelle: Eigene Darstellung.

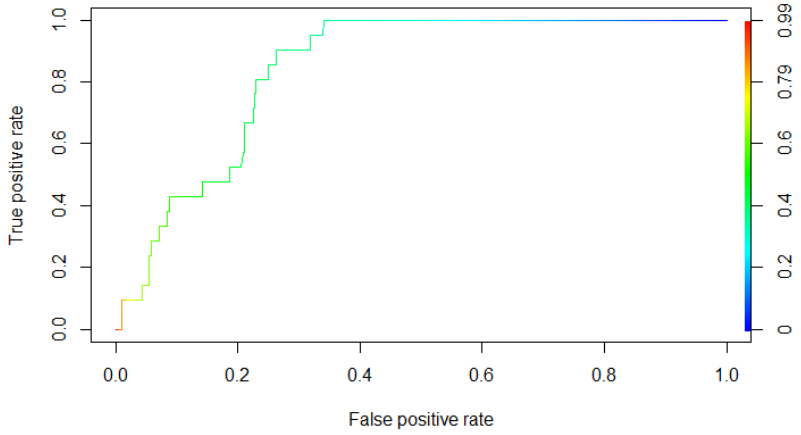

Abbildung 27: ROC-Graph der Bayesschen Logistischen Regression auf den Validierungsdaten (SMOTE). AUC: 0,8439. Quelle: Eigene Darstellung.

Wie beim ersten Durchlauf war erneut ein Neural Network mit einem Hidden Layer und 4 Knoten am besten (siehe Abbildung 28).

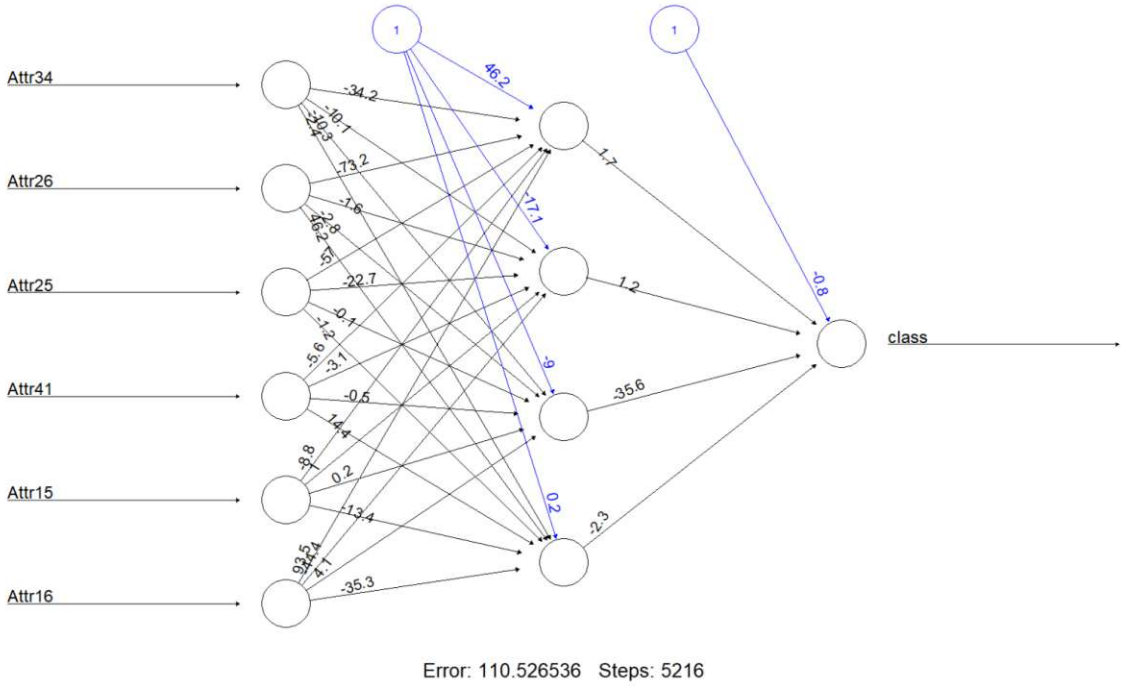

Abbildung 28: Graph des Neural Networks (SMOTE). Quelle: Eigene Darstellung.

Sowohl bei den Trainings- als auch bei den Validierungsdaten lieferte das Neural Network neue Bestleistungen, wie in Abbildungen 29 und 30 zu sehen ist.

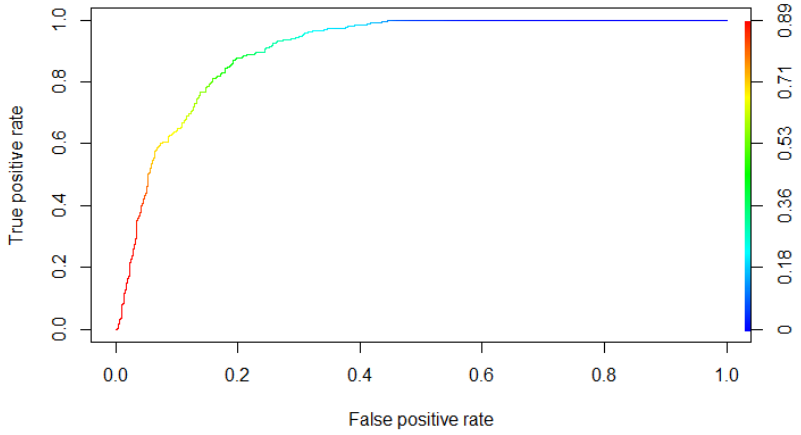

Abbildung 29: ROC-Graph des Neural Networks auf den Trainingsdaten (SMOTE). AUC: 0,9051. Quelle: Eigene Darstellung.

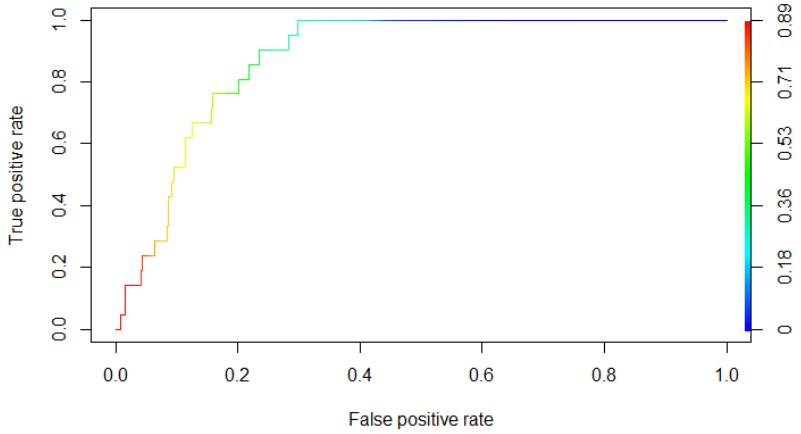

Abbildung 30: ROC-Graph des Neural Networks auf den Validierungsdaten (SMOTE). AUC: 0,8792. Quelle: Eigene Darstellung.

Beim dritten Durchlauf wurden 993 Support Vectors durch die SVM identifiziert, also deutlich mehr als bei den ersten beiden Durchläufen. Gleichzeitig hat sich die Performance deutlich verbessert, denn es konnten die meisten der Konkurse richtig identifiziert werden und zwar auf den Trainings- und den Validierungsdaten. Darüber hinaus zeigen die Tabellen 17 und 18 aber auch, dass deutlich mehr nicht in Konkurs gegangene Unternehmen fälschlicherweise als Konkurs vorhergesagt wurden.

		Echte Klasse	
		0	1
Vorhergesagte Klasse	0	931	149
	1	205	561

Tabelle 17: Konfusionsmatrix der SVM auf den Trainingsdaten (SMOTE). Quelle: Eigene Darstellung.

		Echte Klasse	
		0	1
Vorhergesagte Klasse	0	788	3
	1	191	18

Tabelle 18: Konfusionsmatrix der SVM auf den Validierungsdaten (SMOTE). Quelle: Eigene Darstellung.

4.4 Performance auf den Testdaten

Um die Performance der Modelle fair vergleichen zu können, werden im finalen Schritt die bisher nicht verwendeten Testdaten in die Modelle eingespeist. Zuvor muss anhand der Performance auf den Trainings- und Validierungsdaten entschieden werden, welche Variante des jeweiligen Modells verwendet werden soll. Also ob entweder das Modell, welches auf den standardisierten Daten basiert, oder das Modell, welches auf den Daten ohne Ausreißer basiert, oder das Modell, welches auf den SMOTE-Daten basiert, verwendet werden soll. Darüber hinaus muss entscheiden werden, welche Grenzwahrscheinlichkeit benutzt werden soll. Am einfachsten fällt diese Entscheidung bei der SVM. Diese hat den zusätzlichen Metaparameter in Form einer Grenzwahrscheinlichkeit nicht und hatte eindeutig die beste Performance auf den mit SMOTE generierten Daten. Beim Neural Network war die Performance auf den Validierungsdaten beim ersten Durchlauf (standardisierte Daten) ähnlich wie beim dritten Durchlauf (SMOTE). Insgesamt war die Performance beim dritten Durchlauf jedoch etwas besser. Daher fällt die Wahl auf dieses Modell. Bei der Logistischen Regression und der Bayesschen Logistischen Regression ist die Performance auf den Validierungsdaten beim zweiten Durchlauf (Ausreißer entfernt) und beim dritten Durchlauf (SMOTE) sehr ähnlich. Auf den Trainingsdaten ist die Performance im zweiten Durchlauf besser, allerdings sind die Validierungsdaten wichtiger für

diese Entscheidung. Da das Entfernen von Ausreißern nicht unumstritten ist (siehe Kapitel 4.1) und die Performance im dritten Durchlauf in etwa gleich war, werden auch bei der Logistischen Regression und bei der Bayesschen Logistischen Regression jene Modelle gewählt, die auf den SMOTE-Daten kalibriert wurden.

Wie bereits erwähnt, müssen für die Logistische Regression, die Bayessche Logistische Regression und das Neural Network noch die Grenzwahrscheinlichkeiten festgelegt werden. Dabei sind durchaus unterschiedliche Vorgehensweisen möglich. Man könnte beispielsweise die Grenzwahrscheinlichkeiten so wählen, dass ein gewisses Performance-Maß (auf den Validierungsdaten) maximiert wird. In der Praxis sollte man auch die eigene Risikobereitschaft mit einbeziehen. Wenn man die Grenzwahrscheinlichkeit höher ansetzt, prognostiziert man für weniger Unternehmen einen Konkurs und umgekehrt. Hilfreich kann auch ein Blick auf die ROC-Graphen der Modelle sein. Dabei kann man beispielsweise auf die Steigung des Graphen achten. Ist diese gering, so verbessert sich die true positive rate nur wenig, während die false positive rate stark ansteigt. Nach Betrachtung der jeweiligen ROC-Graphen (Abbildungen 25, 27 und 30) wurde folgende Vorgehensweise gewählt: die Grenzwahrscheinlichkeit wird als maximale Wahrscheinlichkeit angesetzt, sodass alle in Konkurs gegangenen Unternehmen (in den Validierungsdaten) als solche erkannt werden. Oder anders formuliert: die Grenzwahrscheinlichkeit ist die maximale Wahrscheinlichkeit, sodass die true positive rate 1 ist. Dies wurde entschieden, da in den ROC-Graphen keine signifikante Abnahme der Steigung bis zum Erreichen einer true positive rate von 1 zu sehen war.

Die Grenzwahrscheinlichkeiten (Cutoff) und die Performance-Maße (auf den Testdaten) aus Kapitel 4.2 sind in Tabelle 19 zu sehen. Diese Tabelle fasst die finale Evaluierung der Modelle zusammen.

	Cutoff	Accuracy	Precision	Recall	F-Maß	AUC
Log. Reg.	0,3584	0,6585	0,0503	0,9000	0,0954	0,8318
Bayes Log. Reg.	0,3520	0,6460	0,0486	0,9000	0,0923	0,8336
Neural Network	0,2825	0,7200	0,0578	0,8500	0,1083	0,8405
SVM	-	0,7980	0,0625	0,6500	0,1140	-

Tabelle 19: Performance-Maße auf den Testdaten. Quelle: Eigene Darstellung.

Es stellt sich natürlich nun die Frage, welches Modell am besten abgeschnitten hat. Je nachdem welches Performance-Maß in Tabelle 19 man betrachtet, so fällt die Antwort

unterschiedlich aus. Bei der Accuracy schneidet die SVM eindeutig am besten ab, gefolgt vom Neural Network. Logistische Regression und Bayessche Logistische Regression performen, wie so oft, sehr ähnlich, allerdings deutlich schlechter als die anderen Modelle. In diesem Fall wird die Accuracy hauptsächlich davon beeinflusst, wie viele true negatives die Modelle erreicht haben. Die SVM hatte die geringste Tendenz, stabile Unternehmen als Konkurse vorherzusagen, und erreichte somit den höchsten Wert bei diesem Maß. Da die Anzahl der in Konkurs gegangenen Unternehmen in den Testdaten gering ist, spielen die false positives eine wichtige Rolle bei der Precision. Alle Modelle haben weit mehr false positives als true positives vorhergesagt, wodurch dieses Performance-Maß gering ausfällt. Die Reihung der Modelle ist analog zur Accuracy - erneut war die SVM am besten, da sie mehr true negatives und damit auch weniger false positives als die anderen Modelle vorhergesagt hatte. Der Recall ist einfach als true positive rate definiert. Bei diesem Performance-Maß zeigt sich nun ein gänzlich anderes Bild. Die meisten true positives (90%) konnten die Logistische Regression und die Bayessche Logistische Regression erreichen. Das Neural Network folgt dicht dahinter, während die SVM klar abgeschlagen an letzter Stelle liegt. Das F-Maß setzt sich aus Precision und Recall zusammen. Alle Modelle schneiden dabei eher schwach ab, wobei die Reihung jener von der Precision folgt, also die SVM am besten abschneidet. Bekanntermaßen lässt sich die AUC für die SVM nicht berechnen. Bei jenen drei Modellen, bei denen die AUC berechnet werden kann, ist diese sehr ähnlich.

Lässt man die AUC außen vor, so schneidet die SVM bei drei von vier Performance-Maßen am besten ab. Man könnte daher dieses Modell als am geeignetsten für die Aufgabe betrachten. Jedoch ist der Recall sehr wichtig, denn dieser misst, wie viele Konkurse richtig erkannt wurden. Bei diesem Maß ist die SVM deutlich schlechter als die anderen Modelle. Die Wahl des geeignetsten Modells hängt also auch davon ab, worauf man Wert legt. Oft wird dies die Vermeidung von verlustbringenden Geschäften sein. Wenn man darüber hinaus bereit ist, einige gewinnbringende Geschäfte aufzugeben, so ist die SVM sicher nicht die beste Wahl. Logistische Regression und Bayessche Logistische Regression bieten nämlich einen wesentlich besseren Recall, erkennen also deutlich mehr Konkurse. Einen soliden Mittelweg bietet das Neural Network, es erkennt etwas weniger Konkurse als die beiden zuvor genannten Modelle, produziert dafür allerdings auch deutlich weniger Fehlalarme.

5 Zusammenfassung und Fazit

In dieser Arbeit dreht sich alles um die Einteilung von Unternehmen in zwei Gruppen: in jene, die in Konkurs gehen werden, und jene, die nicht in Konkurs gehen werden. Auf der formalen Ebene handelt es sich hier um ein Klassifizierungsproblem mit zwei Klassen. Wie in der Einleitung erwähnt, ist das Hauptziel dieser Arbeit, zu vergleichen, wie gut verschiedene Machine-Learning-Methoden diese konkrete Aufgabe erfüllen.

In Kapitel 2 wurden vier Machine-Learning-Methoden vorgestellt, nämlich die Logistische Regression, die Bayessche Logistische Regression, Neural Networks und die Support Vector Machine. Die Logistische Regression ist eine bekannte und etablierte Methode, die im Kern aus einem linearen Modell besteht. Dieses wird jedoch, im Gegensatz zur linearen Regression, nicht direkt verwendet, um die Erwartungswerte der Zielvariablen zu schätzen. Stattdessen werden die Erwartungswerte zunächst mit einer sogenannten Link-Funktion transformiert. Eine Erweiterung dieses Konzepts ist die Bayessche Logistische Regression, welche es ermöglicht, den Parametern des Logistischen Modells eine a priori Verteilung zuzuweisen. Mittels Markov-Chain-Monte-Carlo zieht man dann Stichproben aus der a posteriori Verteilung der Parameter, welche sich ergibt, wenn man die Informationen aus den Daten zur a priori Verteilung hinzuzieht. Damit erhält man nicht ein deterministisches Modell, sondern jeweils ein unterschiedliches Modell, je nachdem welche zufällige Ziehung aus der Stichprobe man betrachtet. Diese können dann beispielsweise durch Durchschnittsbildung zusammengefasst werden. Als nächstes wurden Neural Networks betrachtet. Genauer gesagt ein Spezialfall dieser Modelle, nämlich die Multilayer Percepetrons. Diese können als azyklischer Graph visualisiert werden, welcher aus mehreren Ebenen, den sogenannten Layern besteht. Jeder Knoten bildet eine gewichtete Summe über die Outputs der Knoten des vorangegangenen Layers und erzeugt dadurch selbst wieder einen Output. Den Anfang dieses schrittweisen Prozesses bildet der Input-Layer, in welchem die Daten eingespeist werden. Der Output des/der letzten Knoten ist gleichzeitig der Output des gesamten Modells. Die letzte Methode war die Support Vector Machine. Diese ist von geometrischen Überlegungen motiviert. Man stelle sich die Datensätze als Punkte in einem affinen Raum vor. Dann ist es das Ziel der Support Vector Machine, eine Hyperebene zu finden, die die beiden Klassen separiert. Können die Klassen nicht separiert werden, so soll zumindest eine Hyperebene gefunden werden, die diesem Ziel möglichst nahe kommt.

In Kapitel 3 lag der Fokus auf den Daten. Möchte man eine möglichst gute Performance erreichen, so muss man sich unweigerlich mit diesem Thema auseinandersetzen. Dies beginnt bereits damit, wie man möglichst aussagekräftige Daten erhält. Schließlich kann man durch die Modelle nicht mehr Informationen extrahieren, als in den Daten vorhan-

den sind. In dieser Arbeit wurden der Jahresabschluss oder vergleichbare Finanzberichte als angemessene Datenquellen ausgemacht. Hat man eine zufriedenstellende Datenmenge gesammelt, dann ist es sinnvoll, sich zunächst einen Eindruck von den Daten zu verschaffen. Dazu gibt es verschiedenste Ansätze, in Kapitel 3.2 wurden jedoch einige Fragen ausgearbeitet, die als Anhaltspunkte dienen können. Darüber hinaus können Visualisierungen sinnvoll sein, beispielsweise mittels Boxplots. Stellt man dabei fest, dass bei manchen Datensätzen nicht alle Merkmale vorhanden sind, dann sollte man dieses Problem im nächsten Schritt angehen. Es gibt drei verschiedene Möglichkeiten, warum Werte fehlen. Im Wesentlichen unterscheidet man dabei, ob die Werte zufällig fehlen oder die Ursache in den Daten begründet ist. Je nach Situation gibt es unterschiedliche Lösungsansätze, welche in Kapitel 3.3 vorgestellt wurden. Darüber hinaus wurden noch zwei Techniken zur Datenaufbereitung diskutiert, nämlich Binning und die Principal Component Analysis. Darüber hinaus wurde eine modellunabhängige Methode zur Merkmalauswahl vorgestellt, welche auf dem Information Value basiert. Schließlich beschäftigte sich dieses Kapitel noch mit dem SMOTE-Algorithmus, welcher dazu dient, Datensätze zu synthetisieren. Dies kann beispielsweise verwendet werden, wenn Untergruppen in den Daten sehr unausgewogen sind. Im konkreten Kontext betrifft dies die in Konkurs gegangenen Unternehmen, welche nur etwa 2% aller Unternehmen in den Beispieldaten ausmachen. Außerdem wurde in diesem Kapitel die Aufspaltung der Daten besprochen. Man teilt die Daten nämlich in drei disjunkte Untermengen: die Trainingsdaten, die Validierungsdaten und die Testdaten. Die Trainingsdaten dienen zur Kalibrierung der Modelle. Die Validierungsdaten werden verwendet, um gewisse Parameter festzulegen und unterschiedliche Varianten eines Modells zu vergleichen. Die Testdaten werden nur mit den fertigen Modellen verwendet. Es dürfen also keine Anpassungen vorgenommenw werden, um die Performance der Modelle auf diesen Daten zu verbessern (bei den Validierungsdaten ist dies hingegen sehr wohl erlaubt). Dadurch soll eine möglichst faire Bewertung der Modelle ermöglicht werden.

Kapitel 4 beschrieb schließlich, wie die Theorie aus Kapiteln 2 und 3 in die Praxis umgesetzt werden kann. Dies geschah mittels Beispieldaten. Dabei wurde der Prozess von der Analyse und Aufbereitung der Daten über die Kalibrierung der Modelle bis hin zur Evaluierung der Performance vollständig durchlaufen. Darüber hinaus wurden Performance-Maße vorgestellt. Besonders wichtig waren dabei der ROC-Graph und die damit assoziierte AUC, welche einfach die Fläche unter diesem Graph ist. Es wurden in diesem Kapitel drei Durchläufe durchgeführt. Für den ersten Durchlauf wurden die Daten nur standardisiert. Beim zweiten Durchlauf wurden zunächst die Ausreißer entfernt und dann die Daten standardisiert. Die Trainingsdaten des finalen Durchlaufs wurden hingegen mit dem SMOTE-Algorithmus synthetisiert. Wie bei den anderen Durchläufen wurden auch diese Daten standardisiert. Bei allen drei Durchläufen wurden die vier in Kapitel 2 vorgestellten

Modelle auf den jeweiligen Trainingsdaten kalibriert und dann auf den Validierungsdaten getestet. Die Performance wurde dabei meist mittels der ROC-Graphen und der AUC gemessen. Nur bei der SVM war dies nicht möglich. Deswegen wurde für dieses Modell die Konfusionsmatrix betrachtet. Auf Basis der Validierungsdaten wurde für jedes Modell der beste Durchlauf ausgewählt. In diese Modelle wurden dann die Testdaten eingespeist und es wurden nochmals Performance-Maße berechnet, um einen finalen Vergleich treffen zu können.

Zum Abschluss dieser Arbeit soll nun noch ein Fazit gegeben werden. Das Hauptziel war ja, die vier Machine-Learning-Methoden zu vergleichen. Die konzeptionellen Unterschiede sind in Kapitel 2 ersichtlich. Interessanter sind jedoch die Unterschiede in der Performance bei der Vorhersage von Konkursen. Eine kompakte Zusammenfassung der Performance der einzelnen Modelle liefert Tabelle 19. Berechtigterweise könnte man nun die Frage stellen, welches Modell denn am besten ist. Leider kann aus den Experimenten, die im Zuge dieser Arbeit durchgeführt wurden, keine klare Antwort gegeben werden. Die Antwort hängt nämlich davon ab, worauf man Wert legt. In manchen Anwendungsbereichen wird es beispielsweise wichtig sein, möglichst viele Konkurse korrekt vorherzusagen, und gleichzeitg nicht so nachteilhaft sein, wenn dadurch auch einige Fehlalarme entstehen. [7] In diesem Fall haben die Logistische Regression und die Bayessche Logistische Regression die beste Performance geliefert. Das Neural Network war in dieser Hinsicht nur geringfügig schlechter. Die SVM hingegen landete klar abgeschlagen auf dem letzten Platz. Kann man also zumindest sagen, dass die SVM am schlechtesten für die Konkursvorhersage geeignet ist? Nicht unbedingt, denn in drei von vier Performance-Maßen schnitt die SVM am besten ab. Die Ursache dafür war, dass die SVM weniger Fehlalarme als die anderen Modelle produziert hat. Somit bleibt festzuhalten, dass man im konkreten Anwendungsfall entscheiden muss, welches Modell am geeignetsten ist. Darüber hinaus konnte beobachtet werden, dass die logistische Regression und die Bayessche Logistische Regression, welche als Erweiterung der logistischen Regression gesehen werden kann, beinahe identisch performt haben. Die zusätzliche Komplexität der Bayesschen Logistischen Regression konnte ihr also zumindest in diesen Experimenten keine Vorteile verschaffen.

[7] Dieser vorsichtige Ansatz wurde auch bei der Wahl der Grenzwahrscheinlichkeiten verfolgt.

A Verwendete Daten

Für Visualisierungen und für die Performance-Tests in Kapitel 4 wurden Daten von polnischen Unternehmen verwendet. Die Unternehmen, welche in Konkurs gegangen sind, wurden von 2000 bis 2012 analysiert, jene die nicht in Konkurs gegangen sind von 2007 bis 2013. [26] Die Daten können unter https://www.kaggle.com/c/companies-bankruptcy-forecast/data abgerufen werden. Es folgt eine Auflistung der 64 zur Verfügung stehenden Attribute inklusive einer kurzen Beschreibung. Die 65. Variable der Daten enthält die Klasse.

- attr1 - net profit / total assets
- attr2 - total liabilities / total assets
- attr3 - working capital / total assets
- attr4 - current assets / short-term liabilities
- attr5 - [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
- attr6 - retained earnings / total assets
- attr7 - EBIT / total assets
- attr8 - book value of equity / total liabilities
- attr9 - sales / total assets
- attr10 - equity / total assets
- attr11 - (gross profit + extraordinary items + financial expenses) / total assets
- attr12 - gross profit / short-term liabilities
- attr13 - (gross profit + depreciation) / sales
- attr14 - (gross profit + interest) / total assets
- attr15 - (total liabilities * 365) / (gross profit + depreciation)
- attr16 - (gross profit + depreciation) / total liabilities
- attr17 - total assets / total liabilities
- attr18 - gross profit / total assets
- attr19 - gross profit / sales

- attr20 - (inventory * 365) / sales
- attr21 - sales (n) / sales (n-1)
- attr22 - profit on operating activities / total assets
- attr23 - net profit / sales
- attr24 - gross profit (in 3 years) / total assets
- attr25 - (equity - share capital) / total assets
- attr26 - (net profit + depreciation) / total liabilities
- attr27 - profit on operating activities / financial expenses
- attr28 - working capital / fixed assets
- attr29 - logarithm of total assets
- attr30 - (total liabilities - cash) / sales
- attr31 - (gross profit + interest) / sales
- attr32 - (current liabilities * 365) / cost of products sold
- attr33 - operating expenses / short-term liabilities
- attr34 - operating expenses / total liabilities
- attr35 - profit on sales / total assets
- attr36 - total sales / total assets
- attr37 - (current assets - inventories) / long-term liabilities
- attr38 - constant capital / total assets
- attr39 - profit on sales / sales
- attr40 - (current assets - inventory - receivables) / short-term liabilities
- attr41 - total liabilities / ((profit on operating activities + depreciation) * (12/365))
- attr42 - profit on operating activities / sales
- attr43 - rotation receivables + inventory turnover in days
- attr44 - (receivables * 365) / sales

- attr45 - net profit / inventory
- attr46 - (current assets - inventory) / short-term liabilities
- attr47 - (inventory * 365) / cost of products sold
- attr48 - EBITDA (profit on operating activities - depreciation) / total assets
- attr49 - EBITDA (profit on operating activities - depreciation) / sales
- attr50 - current assets / total liabilities
- attr51 - short-term liabilities / total assets
- attr52 - (short-term liabilities * 365) / cost of products sold)
- attr53 - equity / fixed assets
- attr54 - constant capital / fixed assets
- attr55 - working capital
- attr56 - (sales - cost of products sold) / sales
- attr57 - (current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
- attr58 - total costs /total sales
- attr59 - long-term liabilities / equity
- attr60 - sales / inventory
- attr61 - sales / receivables
- attr62 - (short-term liabilities *365) / sales
- attr63 - sales / short-term liabilities
- attr64 - sales / fixed assets
- class - the response variable Y: 0 = did not bankrupt; 1 = bankrupt

Literatur

[1] H. Abdou und J. Pointon. “Credit Scoring, Statistical Techniques and Evaluation Criteria: A Review of Literature”. In: *Intelligent Systems in Accounting, Finance and Management* 18 (2011), S. 59–88.

[2] A. Afifi, S. May und V. Clark. *Practical Multivariate Analysis*. 5th Edition. Chapman und Hall/CRC, 2012.

[3] A. Agresti. *Categorical Data Analysis*. 3rd Edition. John Wiley & Sons, 2013.

[4] P. Allison. *Missing Data*. Sage University Paper Series on Quantitative Applications in the Social Sciences. 07-136. Sage, 2001.

[5] M. Augasta und T. Kathirvalavakumar. “Pruning algorithms of neural networks - a comparative study”. In: *Central European Journal of Computer Science* 3 (2013), S. 105–115.

[6] S. Boyd und L. Vandenberghe. *Convex Optimization*. 7th Printing. Cambridge University Press, 2004.

[7] C. Chatfield. *Problem Solving: A Statistician’s Guide*. Chapman and Hall, 1988.

[8] C. Cortes und V. Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20 (1995), S. 273–297.

[9] T. Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* 27 (2006), S. 861–874.

[10] A. Fernandez u. a. “SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary”. In: *Journal of Artificial Intelligence Research* 61 (2018), S. 863–905.

[11] S. Fritsch, F. Guenther und M. Wright. *Package ’neuralnet’*. Zuletzt aufgerufen am 12.08.2021 auf https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf.

[12] J. Gabry und B. Goodrich. *Package ’rstanarm’*. Zuletzt aufgerufen am 12.08.2021 auf https://cran.rstudio.com/web/packages/rstanarm/rstanarm.pdf.

[13] B. Galler. *Scarce Data based Credit Risk Assessment: Evaluating Missing Data Methods in PD-estimation by Logistic Regression*. Tectum Verlag, 2014.

[14] A. Gelman u. a. *Bayesian Data Analysis*. 3rd Edition. Chapman and Hall/CRC, 2013.

[15] M. Gevrey, I. Dimopoulos und S. Lek. “Review and comparison of methods to study the contribution of variables in artificial neural network models”. In: *Ecological Modelling* 160 (2003), S. 249–264.

[16] P. Gill, W. Murray und M. Wright. *Practical Optimization*. 11th Printing. Academic Press, 1981.

[17] A. Goh. “Back-propagation neural networks for modeling complex systems”. In: *Artificial Intelligence in Engineering* 9 (1995), S. 143–151.

[18] D. J. Hand und W. E. Henley. “Statistical Classification Methods in Consumer Credit Scoring: a Review”. In: *Statistics in Society* 160 (1997), S. 523–541.

[19] T. Hastie, R. Tibshirani und J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd Edition. Springer New York, 2009.

[20] S. Haykin. *Neural Networks: A Comprehensive Foundation*. 2nd Edition. Prentice Hall, 1999.

[21] K. Hornik, M. Stinchcombe und H. White. “Multilayer Feedforward Networks are Universal Approximators”. In: *Neural Networks* 2 (1989), S. 359–366.

[22] S. Huang und Y. Huang. “Bounds on the Number of Hidden Neurons in Multilayer Perceptrons”. In: *IEEE: Transactions on Neural Networks* 2 (1991), S. 47–54.

[23] G. James u. a. *An Introduction to Statistical Learning*. 2nd Edition. Springer, 2021.

[24] R. Johnson und D. Wichern. *Applied Multivariate Statistical Analysis*. 6th Edition. Prentice Hall, 2007.

[25] I. Jolliffe. *Principal Component Analysis*. 2nd Edition. Springer, 2002.

[26] Kaggle.com. *Companies Bankruptcy Forecast*. Zuletzt aufgerufen am 19.07.2021 auf https://www.kaggle.com/c/companies-bankruptcy-forecast/data.

[27] J. Karhunen und J. Joutsensalo. “Generalizations of Principal Component Analysis, Optimization Problems, and Neural Networks”. In: *Neural Networks* 8 (1995), S. 549–562.

[28] D. Meyer u. a. *Package ’e1071’*. Zuletzt aufgerufen am 12.08.2021 auf https://cran.r-project.org/web/packages/e1071/e1071.pdf.

[29] A. Mukeri, H. Shaikh und D. Gaikwad. “Financial Data Analysis Using Expert Bayesian Framework For Bankruptcy Prediction”. Preprint, siehe: https://arxiv.org/abs/2010.13892. 2020.

[30] J. Nelder und R. Wedderburn. “Generalized Linear Models”. In: *Journal of the Royal Statistical Society*. Series A (General) 135 (1972), S. 370–384.

[31] oesterreich.gv.at-Redaktion. *Begriffsdefinition Konkurs*. Zuletzt aufgerufen am 10.10.2021 auf https://www.oesterreich.gv.at/lexicon/K/Seite.990070.html.

[32] J. Olden und D. Jackson. “Illuminating the “black box“: a randomization approach for understanding variable contributions in artificial neural networks”. In: *Ecological Modelling* 154 (2002), S. 135–150.

[33] J. Olden, M. Joy und R. Death. “An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data”. In: *Ecological Modelling* 178 (2004), S. 389–397.

[34] I. Oliveira, M. Chari und S. Haller. *Rigorous Constrained Optimized Binning for Credit Scoring*. Techn. Ber. SAS Institute Inc., 2008.

[35] T. Pollet und L. van der Meij. “To Remove or not to Remove: the Impact of Outlier Handling on Significance Testing in Testosterone Data”. In: *Adaptive Human Behaviour and Physiology* 3 (2017), S. 43–60.

[36] B. Ripley. *Pattern Recognition and Neural Networks*. 8th Printing. Cambridge University Press, 1996.

[37] R. Rockafellar. *Convex Analysis*. 2nd Printing. Princeton University Press, 1972.

[38] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer Verlag, 1996.

[39] J. Schafer und J. Graham. “Missing Data: Our View of the State of the Art”. In: *Psychological Methods* 7 (2002), S. 147–177.

[40] M. Shanker, M. Hu und M. Hung. “Effect of Data Standardization on Neural Network Training”. In: *Omega* 24 (1996), S. 385–397.

[41] N. Siddiqi. *Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring*. John Wiley & Sons, 2006.

[42] L. Torgo. *Package ’performanceEstimation’*. Zuletzt aufgerufen am 12.09.2021 auf https://cran.r-project.org/web/packages/performanceEstimation/performanceEstimation.pdf.

[43] J. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[44] M. Wilhelmsen u. a. *Bayesian Modelling of Credit Risk Using Integrated Nested Laplace Approximations*. Zuletzt aufgerufen am 28.05.2021 auf https://projects.nr.no/en/nrpublication?query=/file/4949/Wilhelmsen_-_Bayesian_modelling_of_credit_risk_using_integrated.pdf. 2009.

[45] S. Xie, A. Lawniczak und J. Hao. “Modelling Autonomous Agents’ Decisions in Learning to Cross a Cellular Automaton-Based Highway via Artificial Neural Networks”. In: *Computation* (2020), S. 143–151.

[46] S. Xie. *Package ’scorecard’*. Zuletzt aufgerufen am 21.07.2021 auf https://cran.r-project.org/web/packages/scorecard/scorecard.pdf.

[47] A. Zell. *Simulation neuronaler Netze*. 1. Auflage. Addison-Wesley, 1994.

[48] G. Zeng. “A Necessary Condition for a Good Binning Algorithm in Credit Scoring”. In: *Applied Mathematical Science* (2014), S. 3229–3242.

[49] G. P. Zhang. “Neural Networks for Classification: A Survey”. In: *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews* 30 (2000), S. 451–461.