Machine Learning und Prognosen

Umsetzung mit R im SQLServer 2017 anhand von Taxifahrten in New York

Bachelorthesis

Studiengang *Angewandte Informatik*

Duale Hochschule Baden-Württemberg Mannheim

von

Leonhard Applis

Abgabedatum:	26.09.2018
Matrikelnummer, Kurs:	2086307, TINF15AIBI
Ausbildungsfirma:	Atos Information Technology GmbH
Betreuer der Ausbildungsfirma	Jonas Mauer
Gutachter der Dualen Hochschule:	Prof. Dr. Rainer Colgen

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Bachelorthesis mit dem Thema

> *Machine Learning und Prognosen Umsetzung mit R im SQLServer 2017 anhand von Taxifahrten in New York*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Fürth, den 17. September 2018

_____

Leonhard Applis

Abstract

This thesis gives an overview and examples about machine learning in SQL Server 2017 using neural networks.

Goals of this thesis are to explain main-components necessary for neural networks as well as their use and representation in the SQL-Server with the programming language R, to predict use-cases about NYC-Taxidata.

The use-cases are predicting the tip-amount of a ride, guessing the taxirate, estimate total rides for a certain location, guess the passengercount of a ride and estimate the revenue made per location and hour.

Contents of this work cover general regression and classification as basis for neural networks, explanation and examples for the implementation of neural networks in the SQL Server with R, a case study of several usecases using NYC Taxi-Data including performancetuning and summarized best practices working with neural networks in R.

title:	Machine-Learning and Prognosis
author:	Leonhard Applis
matriculation number:	2086307
class:	TINF15AIBI
supervisor Atos:	Jonas Mauer
reviewer DHBW:	Prof. Dr. Rainer Colgen

Kurzfassung

Diese Bachelorarbeit gibt einen Überblick und Beispiele für Machine-Learning im SQL Server 2017 unter Verwendung neuronaler Netze.

Ziel dieser Arbeit ist eine Vorstellung der Hauptkomponenten eines neuronalen Netzes sowie seiner Repräsentation, Erstellung und Benutzung innerhalb des SQL Servers mit der Programmiersprache R, um verschiedene Prognosemodelle für ein New-Yorker Taxiunternehmen zu erstellen.

Die Prognosen behandeln die Schätzung des Trinkgeldes einer Fahrt, eine Erkennung der Taxi-Rate, eine Schätzung des Fahrtenaufkommens einer Gegend zu gegebener Zeit, ein Erkennen der Passagieranzahl einer Fahrt sowie eine Schätzung des Umsatzes anhand eines Ortes zu gegebenen Zeit.

Inhalte dieser Arbeit sind Regressions- und Klassifikationsalgorithmen als Basis zur Erklärung neuronaler Netze, Erklärungen und Implementation von neuronalen Netzen im SQL-Server mit R, eine Fallstudie anhand mehrerer Use-Cases für New Yorker Taxidaten inklusive Performanzverbesserungen und Best-Practices bei der Arbeit mit neuronalen Netzen im SQL-Server.

Titel:	Machine Learning und Prognosen
Author:	Leonhard Applis
Matrikelnummer:	2086307
Kurs:	TINF15AIBI
Betreuer der Firma:	Jonas Mauer
Gutachter der Dualen Hochschule:	Prof. Dr. Rainer Colgen

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DBMS	Database Management System
DHBW	Duale Hochschule Baden-Württemberg
SQL	Structured Query Language
ETL	Extract-Transform-Load

1 Einleitung

> "Hasta la Vista, Baby!"
>
> *Arnold Schwarzenegger* in Terminator 2

Dieses Zitat zählt wohl zu den bekanntesten der Filmgeschichte, und markiert einen der ersten bühnenreifen Auftritte *künstlicher Intelligenz*. Neben österreichischen Bodybuildern beschäftigt dieses Thema seit bald einem Jahrhundert Wissenschaftler, Ethiker und Science-Fiction-Fans gleichermaßen. Was vor zwei Jahrzehnten noch genauso fantasievoll wie schwebende Autos klang, wird in den Softwareschmieden des 21. Jahrhunderts Wirklichkeit:

Künstliche Intelligenzen besiegen Schachprofis, organisieren unsere Kalender, analysieren Bilder und helfen Pandemien einzudämmen. Neben diesen bahnbrechenden Erfolgen gibt es auch weiterhin vielversprechende Forschung in diesem Themengebiet, zum Beispiel computergesteuerte Autos. Aber was ist künstliche Intelligenz eigentlich?

Der Begriff der künstlichen Intelligenz ist sehr weit gefächert - ein Kernelement davon stellt das *Machine Learning* dar. Dieser Bereich, der sich auf die Erstellung von Modellen anhand von Trainingsdaten stützt, hat in den letzten Jahren durch *neuronale Netze* stark an Bedeutung gewonnen. Die Gründe hierfür sind vielseitig, dennoch sind zwei im Besonderen zu nennen: Zum Einen sind Computer deutlich leistungsfähiger geworden, und Aufgaben die früher einen Supercomputer benötigten, sind heute durch ein Smartphone umsetzbar. Zum Anderen sind deutlich mehr Bereiche digitalisiert, und die gewonnenen Daten detaillierter.

Genau diesem Themengebiet widmet sich diese Bachelorarbeit: Machine-Learning und explizit neuronalen Netzen.

1.1 Ziel der Arbeit

Die Ziele dieser Arbeit sind es, ein Grundverständnis für Machine-Learning Algorithmen zu schaffen und dem Leser die Möglichkeit zu geben, diese mit dem SQL-Server 2017 selbst umzusetzen.

Hierfür werden die Theorie verschiedener Algorithmen detailliert vorgestellt und in der Programmiersprache R umgesetzt.

Ebenfalls wird ein detailliertes Fallbeispiel mit Versuchsaufbau und Ergebnissen erarbeitet, damit der Leser eine Einschätzung der Algorithmen vornehmen kann ohne selbst Experimente durchzuführen.

Es ist **nicht** Ziel dieser Arbeit, einen Vergleich zwischen unterschiedlichen Machine-Learning Ansätzen und Frameworks zu ziehen. Auch wird ausschließlich mit R und dem SQL-Server gearbeitet.

Zudem werden weder Grundlagen der Sprachen SQL und R, noch die Vorbereitung des Fallbeispiels geschildert.

1.2 Aufbau der Arbeit

Kapitel 2 dieser Arbeit bildet die Theorie zu modernen Ansätzen des Machine-Learnings. Es werden die Algorithmen für lineare Regression, logistische Regression sowie neuronale Netzwerke detailliert vorgestellt (In Reihenfolge der Nennung). Dieses Kapitel stellt einen rein theoretischen Teil der Arbeit dar, und beinhaltet keine Umsetzung der Algorithmen als Programme.

Darauf aufbauend werden in Kapitel 3 zunächst Grundlagen zu Microsofts SQL-Server 2017 und R geklärt, anschließend liegt der Schwerpunkt des Kapitels auf der Umsetzung von Machine-Learning Algorithmen in R. Innerhalb des Abschnittes 3.3 finden sich Codebeispiele zu Prognosemodellen in T-SQL und R.

Kapitel 4 widmet sich der Umsetzung eines Fallbeispiels eines Taxiunternehmens. Zunächst werden in Abschnitt 4.1 die Ausgangslage der Daten sowie die Ziele des Fallbeispiels exakt definiert.

In Abschnitt 4.2 werden die Stammdaten des Taxiunternehmens und die Wetterdaten in Eigenschaften, Umfang und Bedeutung für Machine Learning dargestellt.

Hauptteil des Kapitels 4 bilden die Unterabschnitte 4.3.1 bis 4.3.3, welche die Erstellung, Verwendung und Bewertung verschiedener neuronaler Netze behandeln. Anschließend werden in Abschnitt **??** die Best Practices im Umgang mit neuronalen Netzen im SQL-Server vorgestellt.

Abschluss der Arbeit bildet in Kapitel 5 ein Fazit über die Qualität der Prognosen unter Berücksichtigung der Komplexität einzelner Teilaufgaben.

1.3 Voraussetzungen an den Leser

Innerhalb dieses Punktes werden die Kenntnisse abgesteckt, die der Leser für das Verständnis der Arbeit benötigt, welche **nicht** im Rahmen dieser Arbeit vorgestellt werden.

- **Mehrdimensionale Algebra:** Im Rahmen dieser Arbeit werden komplexe Algorithmen und Konzepte der mehrdimensionalen Algebra benötigt.

  Schwerpunkte liegen hier v.A. auf dem Lösen von mehrdimensionalen Gleichungen und Matrixoperationen.

- **Stochastik:** Zur Bewertung der Algorithmen werden tiefere Kenntnisse der Stochastik und Statistik benötigt. Die benötigten Schwerpunktthemen sind Verteilungsfunktionen, Hypothesentests und Korrelation.

- **R:** Die Programmiersprache R muss dem Leser im Umfang eines Basiskurses bekannt sein. Sie wird im Zuge der Arbeit verwendet, allerdings werden grundlegende Elemente nicht vorgestellt.

- **SQL:** Die Konzepte von SQL und der Dialekt von T-SQL sind in fortgeschrittenen Zügen benötigt. Die Verwendung des R-Codes innerhalb eines SQL-Servers wird im Zuge der Arbeit vorgestellt.

2 Grundlagen zu Machine-Learning

In diesem Kapitel werden die theoretischen Grundlagen ausgewählter Machine-Learning Algorithmen aus dem Bereich der Regression und Klassifikation vorgestellt.

Es werden lediglich Algorithmen behandelt, die zum Feld des *Supervised Learning* gezählt werden. Diese benötigen Trainingsdaten bestehend aus Ein- und Ausgabewerten, um das Modell daran auszurichten. Ein übliches Beispiel ist die Handschrifterkennung. Das Training wird im Abschnitt 2.5 genauer vorgestellt.

Motivation für alle Algorithmen stellt die Annahme dar, dass es innerhalb der vorliegenden Daten einen Zusammenhang der Werte gibt, eine Funktion, welche einen Satz Daten auf ein Ergebnis abbildet.

Falls eine Funktion existiert, ist diese allerdings häufig zu komplex, um von einem Menschen direkt formuliert zu werden.

Ziel jeder der vorgestellten Algorithmen ist es, ein Modell zu erzeugen, welches möglichst genau die oben vermutete Funktion abschätzt. Hierfür wird eine (große) Menge an Trainingsdaten, sowie eine Menge an Kontrolldaten benötigt.

Es werden zunächst die lineare und logistische Regression vorgestellt - diese stellen zwar keinen Schwerpunkt der Arbeit dar, allerdings liefern Sie viele Grundkonzepte und Basisbausteine, die in einem Neuronalen Netzwerk verwendet werden (z.B. die Aktivierungsfunktion der logistischen Regression).

Allgemeine Umsetzungen dieser Algorithmen finden sich im Abschnitt 3.2 zu R sowie konkret anhand des Fallbeispiels in Kapitel 4.

Zunächst werden allerdings einige gemeinsame Begriffe erläutert.

2.1 Definitionen und Notationen

In diesem Abschnitt werden kurz die benötigten Begriffe und Definitionen vorgestellt. Der Bias wird anschließend auf Grund seiner Bedeutung für Machine Learning gesondert und detailliert behandelt.

Feature Unter einer *Eigenschaft* versteht man eine konkrete Ausprägung eines Merkmals der Eingabewerte des Models.

Die Summe aller Ausprägungen einer Eigenschaft bezeichnet man als Eigenschaftsvektor.

Label & Class Als Label wird eine bestimmte Eigenschaft deklariert, welche v.A. dadurch definiert ist, das sie die Ausgabe des Models darstellt.

Label und Klassen sind synonyme Bezeichner.

Accuracy Die *Genauigkeit* stellt ein Maß dafür dar, wie genau das Modell die Funktion darstellt.

Je nach Art des Modells muss die Genauigkeit unterschiedlich evaluiert werden und wird jeweils im Abschnitt 2.3.4 für Regressionen und im Abschnitt 2.4.2 für Klassifikation vorgestellt.

2.2 Bias

Wenn man mit Prognosemodellen arbeitet, können die Fehler der Vorhersage in zwei Hauptursachen zerlegt werden: Fehler aufgrund von Verzerrung und Fehler aufgrund von [natürlicher] Varianz. Es gibt einen Zusammenhang zwischen der Fähigkeit eines Modells, die Abweichung und die Varianz zu minimieren. Diese beiden Fehler zu verstehen, hilft die Ergebnisse des Modells auszuwerten und die Fehler für *Under-* und *Overfitting* zu vermeiden. (vgl. [FR] Vorwort).

Die **Verzerrung** (engl. Bias) ist der Fehler ausgehend von falschen Annahmen im Lernalgorithmus. Eine hohe Verzerrung kann einen Algorithmus dazu veranlassen, nicht die entsprechenden Beziehungen zwischen Eingabe und Ausgabe zu modellieren. Dieses Problem bezeichnet man als **Unteranpassung**.

Die **Varianz** (engl. Variance) ist der Fehler ausgehend von der Empfindlichkeit auf kleinere Schwankungen in den Trainingsdaten. Eine hohe Varianz verursacht **Überanpassung**: es wird das Rauschen in den Trainingsdaten statt der vorgesehenen Ausgabe modelliert.

In diesem Abschnitt wird zuerst das Bias-Variance-Dilemma vorgestellt und anschließend kurz verschiedene Formen von *Bias*.

Diese Abweichungen spielen in allen Formen des Machine-Learnings und in der Auswahl der Trainingsdaten eine wichtige Rolle (vgl. [Pfe] Absatz 1) und werden in den entsprechenden Algorithmen berücksichtigt. Die nachfolgenden Arten von Bias stellen Überbegriffe dar - v.A. im Bereich der Psychologie wird deutlich genauer unterschieden.

Bias-Variance-Dilemma

Das sogenannte Verzerrungs-Varianz-Dilemma tritt auf, wenn man die Komplexität eines Modells festlegt. Mit steigender Komplexität eines Modells wird der

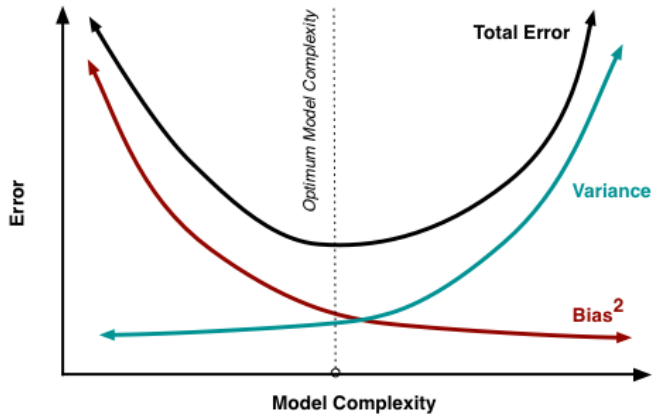

Abbildung 2.1: Bias-Variance-Dilemma

Fehler verringert und die Varianz steigt. Werden mehr Parameter zum Modell aufgenommen, steigt die Komplexität des Modells und die Varianz wird zu einer immer größeren Fehlerquelle, während die Verzerrung sinkt (vgl. [FR] Abschnitt 4.4 Absatz 1). Eine Darstellung für ein einfaches Modell stellt Abbildung 2.2 dar: Die

Kreuze markieren Trainingsdaten, die rote Linie stellt die Vorhersage des Modells dar.

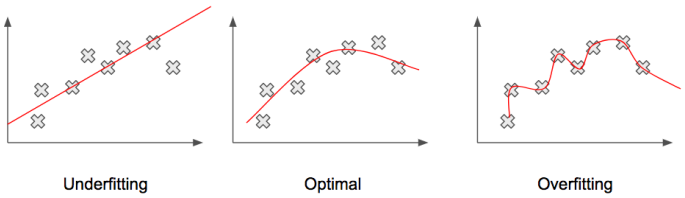

Abbildung 2.2: Über- und Unteranpassung

Wählt man die Komplexität *zu hoch*, so führt dies zu einem zu großen Fehler durch Verzerrung und wird als *Overfitting* bezeichnet.

Wählt man die Komplexität *zu niedrig*, so wird der Sachverhalt nicht vollständig erfasst, und man spricht von *Underfitting*.

Nachdem das Bias-Variance-Dilemma vorgestellt wurde, werden nun verschiedene Arten von Varianz und Bias vorgestellt, die für die Auswahl und Filterung der Trainingsdaten ebenfalls eine Rolle spielen:

Natürliche Varianz

Je nach Art und Gestalt der Erhebung können systematische Schwankungen der Werte auftreten. Diese stellen natürliche Verhältnisse dar, da kein perfektes Modell erfasst werden kann.

Als Beispiel sei die Messung der Zimmertemperatur genannt: Zwei Thermometer können im selben Raum unterschiedliche Ergebnisse liefern - etwa weil sie auf unterschiedlichen Höhen befestigt sind oder eines im Windzug liegt. Es ist im Allgemeinen nicht möglich, ein perfektes Modell zu erstellen, welches alle Faktoren berücksichtigt.

Die natürliche Varianz ist als Hauptgrund zu nennen, warum in jedem (modernen) Machine-Learning Algorithmus eine Abweichung berücksichtigt ist.

Insbesondere ist zu betonen, das die Genauigkeit eines Models, welches auf Machine-Learning beruht, nie höher sein kann als die Varianz der zugrunde liegenden Trai-

ningsdaten. Ein Modell kann lediglich erreichen, die selbe bzw. eine ähnliche Varianz zu simulieren.

Neben diesen *harten* Kriterien gibt es noch zwei Arten von Verzerrung, die mit der Auswahl der Trainingsdaten einhergehen. Da diese nicht über technische Maßnahmen ausgeglichen werden können, werden diese gesondert vorgestellt:

Selection Bias

Unter der Selektionsverzerrung versteht man einen Fehler der Ergebnisse, welcher durch die Auswahl einer **nicht repräsentativen** Stichprobe entsteht (vgl. [Ins] Definition). Ein Beispiel einer Selektionsverzerrung tritt auf, wenn anhand der Umfragen auf einer Messe für vegane Ernährung die Ernährungsgewohnheiten aller Deutschen interpretiert wird.

Im Gegensatz dazu wäre diese Stichprobe sehr wohl geeignet, die Ernährung deutscher Veganer zu beurteilen.

Eine besondere Art des Selection Bias stellt der Confirmation-Bias dar:

Confirmation Bias

Unter dem *Confirmation Bias* (dt. Bestätigungsfehler) versteht man mehrere psychologische Aspekte, die zu einer Verzerrung der Ergebnisse durch den Prüfer führen (vgl. [Dar84] S. 21 Absatz 5 und S. 22 Absatz 1f).

Im Wesentlichen bezieht sich diese Abweichung darauf, das unbewusst Ergebnisse so interpretiert werden, um bestehende Meinungen zu bestätigen. Dies wird hauptsächlich über zwei Mechanismen erreicht:

Die Interpretation nicht-übereinstimmender Ergebnisse und Daten als fehlerhaft, sowie eine überproportionale Gewichtung übereinstimmender Ergebnisse. Hierzu gehört ebenfalls die explizite Suche nach Ergebnissen welche eine Hypothese bestätigen, ohne dieselbe Sorgfalt der Gegenhypothese zukommen zu lassen.

2.3 Lineare Regression

Als erster Machine-Learning-Algorithmus soll die lineare Regression vorgestellt werden.

Auch wenn die lineare Regression nicht mehr Bestandteil aktueller Forschung ist, sind sie für weitere Erklärungen hilfreich, da Neuronale Netze mit Regression sich ebenfalls auf dieses Verfahren stützen.

Zudem können mithilfe einfacher linearer Regression bereits sehr gute Ergebnisse erzielt werden.

2.3.1 Konzept und Ziele linearer Regression

Als Beispiel für die einfache lineare Regression dient das Abschätzen des Bremsweges von PKWs. Hierfür benötigen man eine Tabelle der Gestalt

Geschwindigkeit in km/h	Gewicht in kg	Bremsweg in m
50	1500	20
60	1400	30
90	1000	60
...	...	...

Es ist hierbei offensichtlich, das diese Messwerte zusammenhängen - lediglich die zugrunde liegende Formel ist unbekannt.

Mithilfe linearer Regression wollen wir eine modellhafte Formel finden, die das beste Ergebnis anhand unserer Daten liefert.

2.3.2 Einfache lineare Regression

Zunächst geht man zur Vereinfachung davon aus, dass der Bremsweg lediglich von der Geschwindigkeit abhängt. Bezeichnet man x_i als die Geschwindigkeit des i-ten Datensatzes der Tabelle 2.3.1 und y_i als den zugehörigen Bremsweg, kann man ein lineares Modell der Form

$$y_i := \vartheta_1 \cdot x_i + \vartheta_0 \tag{2.1}$$

herleiten.Zu berechnen ist ϑ_1 und ϑ_0 so, dass der **Mean-Squared-Error** minimal wird.

$$\texttt{MSE}(\vartheta_0, \vartheta_1) := \frac{1}{m-1} \cdot \sum_{i=1}^{m} \left(\vartheta_1 \cdot x_i + \vartheta_0 - y_i\right)^2 \tag{2.2}$$

Die optimalen Ergebnisse des MSE liefern die Variablen:

$$\vartheta_1 = r_{x,y} \cdot \frac{s_y}{s_x} \quad \text{und} \quad \vartheta_0 = \bar{\mathbf{y}} - \vartheta_1 \cdot \bar{\mathbf{x}}. \tag{2.3}$$

Wobei **x** und **y** das arithmetische Mittel der beiden Variablen darstellt, sowie s_x und s_y die Standardabweichungen. bei $r_{x,y}$ handelt es sich um den **Pearson-Korrelationskoeffizienten**.

Nach der Berechnung der *Gewichte* besitzt man ein Modell, welches für jeden beliebigen Geschwindigkeitswert den Bremsweg berechnet.

Dennoch kann man davon ausgehen, das ein lineares Modell für komplexere Sachverhalte keine zufriedenstellenden Ergebnisse liefert. Deswegen wird nun die lineare Regression unter Berücksichtigung mehrerer unabhängiger Variablen behandelt.

2.3.3 Allgemeine lineare Regression

Das Prinzip der allgemeinen linearen Regression ist das Gleiche: Wir suchen eine Funktion, welche uns den abhängigen Wert schätzt. Diese Funktion hat im Allgemeineren die Form $F : \mathbb{R}^m -> \mathbb{R}^1$, und bildet ein m-Eigenschaften umfassendes Tupel x_i auf einen Wert y_i ab.

Bezogen auf das Beispiel 2.3.1 hat man ein Tupel x_i der Form <Geschwindigkeit,Gewicht> und weiterhin einen dazugehörigen Bremsweg y_i. Wir suchen eine Funktion $F(x_i) \approx y_i$.

Diese Funktion können wir ebenfalls durch ein lineares Modell ausdrücken. Sie hat die Gestalt:

$$F(x_i) = \vartheta_2 \cdot x_i^2 + \vartheta_1 \cdot x_i^1 + \vartheta_0 \cdot x_i^0 \tag{2.4}$$

Wobei x_i^n die n-te Komponente des i-ten Elementes darstellt. x^0 ist eine Erweite-

rung um den Bias.

Für dieses Modell, bzw. allgemein für alle Modelle dieser Form kann ebenfalls der MSE berechnet und minimiert werden, um eine optimale Gewichtsmatrix zu erzeugen. Die genauen mathematischen Verfahren hierfür finden sich z.B. im Vorlesungsskript von Prof. Stroetmann [Str] Kapitel 5 Abschnitt 2 und Unterabschnitte.

2.3.4 Bewertung der linearen Regression

Um die statistische Aussagekraft des Modells zu bewerten, eignet sich ein **F-Test** ([Str] S. 86 Absatz 1), dieser ist definiert durch die Formel:

$$\mathrm{F} = \frac{\mathtt{TSS} - \mathtt{RSS}}{\mathtt{RSS}} \cdot \frac{m - p - 1}{p} \tag{2.5}$$

Die F-Statistik ist Fisher-Snedecor-verteilt mit $p - 1$ Freiheitsgraden im Nenner und $m - p$ Freiheitsgraden im Zähler.

$$\mathtt{TSS} = \sum_{i=1}^{m} \left(y_i - \bar{\mathbf{y}}\right)^2 \quad \mathtt{RSS} := \sum_{i=1}^{m} \left(\vartheta_1 \cdot x_i + \vartheta_0 - y_i\right)^2 \tag{2.6}$$

TSS wird hierbei als die *Total Sum of Squares* bezeichnet, RSS für die *Residual Sum of Squares*. (Weiterführend: [Str] S. 77 Absatz 4f).

Dieser Test ist dahingehend notwendig, da ein simples Vergleichen der Modellschätzung (immer) von den echten Werten abweicht. Innerhalb des F-Testes werden ebenfalls die Varianz der Grundgesamtheit in Relation zur Varianz der Modellwerte betrachtet, um ein aussagekräftiges Ergebnis zu erzielen.

Anmerkung: Für die Bewertung eines ML-Algorithmus, welcher lediglich eine einzelne Variable vorhersagt, ist ein Test auf den r^2-Wert ausreichend.

Da die Ergebnisse und Varianz der Test-Werte mit den Ergebnissen und der Varianz der Bild-Werte verglichen werden, liegt lediglich ein Freiheitsgrad vor.

2.4 Klassifizerung

Nach der linearen Regression soll nun die Klassifizierung vorgestellt werden. Hierfür werden zunächst allgemeine Ziele und ein Beispiel vorgestellt, anschließend wird als ausgewähltes Verfahren die logistische Regression erläutert.

2.4.1 Konzept und Ziele von Klassifizierung

Die Klassifizierung zählt zu den ältesten Anwendungen des Machine-Learning - Ein typisches Beispiel für (überwachte) Klassifizierung ist die Zuordnung einer E-Mail in *Spam* oder *Ham*.

Im Rahmen der Klassifizierung soll ein Modell erzeugt werden, das anhand der Eigenschaften einer E-Mail (z.B. dem Auftreten des Wortes *Pharmacy* oder *Sex*) feststellt, ob es sich um typische Spam-E-Mails handelt.

Das Modell erzeugt einen Wahrscheinlichkeitswert, mit welchem die Klasse angenommen wird (bzw. im Umkehrschluss, wie wahrscheinlich das Gegenereignis ist) und *rät* die entsprechende Klasse.

Des Weiteren ist es möglich, eine sog. Multiklassen-Klassifizierung durchzuführen. Hierbei werden mehr als zwei Klassen betrachtet.

2.4.2 Logistische Regression

Innerhalb der logistischen Regression wird ein Modell erzeugt, welches einen Eigenschaftsvektor mit einem Gewichtsvektor multipliziert, und das Ergebnis über eine Aktivierungsfunktion in eine Wahrscheinlichkeit abbildet.

Dieses Verfahren ermöglicht uns, anstatt der numerischen Zählung der *Treffer*, die reale Wahrscheinlichkeit zu optimieren, und dahingehend unsere Gewichte *smooth* zu justieren.

Das Optimieren des Modells benötigt Trainingsdaten und eine **Optimierungsfunk-**

tion. Mit jedem Satz der Trainingsdaten wird der Gewichtsvektor dahingehend angepasst, das die resultierende Wahrscheinlichkeit in Richtung des korrekten Ergebnisses verschoben wird. Das Maß, in welchem die Gewichte angepasst werden, wird über die Optimierungsfunktion ermittelt.

Im Normalfall wird der Gewichtsvektor mit zufälligen Werten initialisiert. Es ist jedoch möglich, einen bereits bestehenden Vektor zu importieren.

Ebenso ist es üblich, sowohl Eingabewerte, als auch den Gewichtsvektor zu normieren. Einige Optimierungsfunktionen, wie z.B. *Stochastic Gradient Descent* terminieren schneller für normierte Daten (vgl. [Ham] Absatz 2). Grund hierfür sind die Eigenschaften der Loss-Function, die bei einer normierten Eingabe eine *glatte* Oberfläche besitzt, und somit eine bessere Anpassung der Gewichte ermöglicht (Weiterführend [Rei] Abschnitt *Gradient Descent* und Abschnitt *Learning Rate*).

2.4.3 Aktivierungsfunktion

Bei der Aktivierungsfunktion handelt es sich um eine statistische Verteilungsfunktion. Sie bildet einen Wert auf eine Wahrscheinlichkeit ab.

Als übliche Aktivierungsfunktionen werden *tanh*, die Sigmoid-Funktion oder die ReLu-Funktion gewählt. Diese besitzen besondere Eigenschaften innerhalb ihrer Ableitung, was eine Berechnung wesentlich erleichtert ([Str] S95ff *6.3.1 The Sigmoid Function*).

Sigmoid und Tanh

Die Sigmoidfunktion ist eine stochastische Verteilungsfunktion und stellt eine Abwandlung der Tangens-Hyperbolicus Funktion dar.

$$sig(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t} = \frac{1}{2} \cdot (1 + tanh\frac{t}{2}) \tag{2.7}$$

Die Range der Sigmoidfunktion ist $[0, 1]$ und eignet sich insofern explizit für die binäre Klassifizierung, um die Wahrscheinlichkeit der Klasse zu ermitteln.

Die Sigmoidfunktion und Tanh sind in Abbildung 2.3 zu sehen.

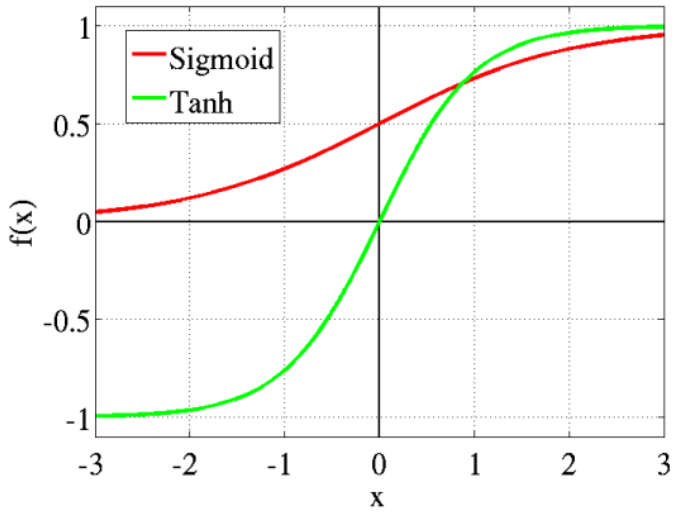

Abbildung 2.3: Sigmoid und Tanh

Die Tangens-Hyperbolicus Funktion unterscheidet sich dahingehend, das sie auf negative Eingaben stark negativ einschlägt, sowie auf Null-Eingaben ebenfalls Null liefert.

Die Ableitung der Sigmoidfunktion in Formel 2.8 ist dahingehend besonders, da Sie sich ebenfalls auf die Sigmoidfunktion beruft.

Nach dem einmaligen Berechnen der Sigmoidfunktion für einen Wert t, lassen sich in einfachen Operationen alle Ableitungen bestimmen.

$$\frac{d\, sig(t)}{d\, t} = sig(t) \cdot (1 - sig(t)) \tag{2.8}$$

Rectified Linear Unit

Die Rectified Linear Unit Funktion, kurz **ReLU** ist eine der am weitesten verwendeten Aktivierungsfunktionen in neuronalen Netzen und ist definiert als:

$$ReLU(t) := max(0, t) \tag{2.9}$$

Der größte Kritikpunkt an der ReLU-Funktion ist, dass negative Eingabewerte stets als Null gewertet werden, was die Möglichkeiten des Modells von den Daten

zu *lernen* stark einschränkt (vgl. [Sha] Abschnitt 3 Absatz 5). Es gibt verschiedene

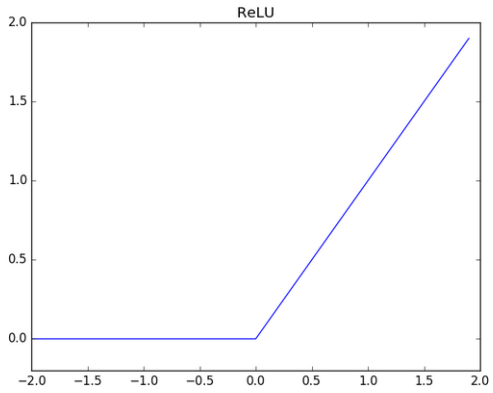

Abbildung 2.4: Plot der ReLU-Funktion

Ansätze, diesen Nachteil auszugleichen:

1. Bei **Leaky Rectified Linear Unit** werden negative Werte anstatt als 0 mit einem Bruchteil (ca. $\frac{1}{100}$-stel) ihres Wertes behandelt.
2. Bei **Randomized Rectified Linear Unit** wird dieser Bruchteil zufällig gewählt und ggfs. während der Laufzeit angepasst.
3. Bei **Euler Linear Unit** (*kurz: ELU*) werden negative Werte mit der Funktion $a(e^x - 1)$ abgebildet, wobei a ein gewählter Parameter zwischen 0 und 1 ist.

Softmax

Die (*echte*) Softmax-Funktion stellt eine logistische Verteilungsfunktion für mehrere Parameter dar. Sie ist definiert als:

$$softmax : \mathrm{R}^K \to \{z \in \mathrm{R}^K | z_i \geq 0, \sum_{i=0}^{K} z_i = 1\} \tag{2.10}$$

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, ..., K \tag{2.11}$$

Die Softmax-Funktion liefert eine Wahrscheinlichkeit für jede Klasse, der ein Trainingsbeispiel angehören kann.Die Wahrscheinlichkeit über alle Klassen ist 1.

Sie ist eine Annäherung an die *max*-Funktion.
Softmax als Aktivierungsfunktion (eines Neurons) mit k-Eingaben ist definiert als:

$$softmax(\vec{t}) = ln \sum_{i=1}^{K} e^{t_i} \tag{2.12}$$

Die Softmax-Aktivierungsfunktion stellt eine Annäherung an die *max* bzw. ReLU Funktion dar, wie dargestellt in Abbildung 2.5. *Softmax* gewinnt dahingehend

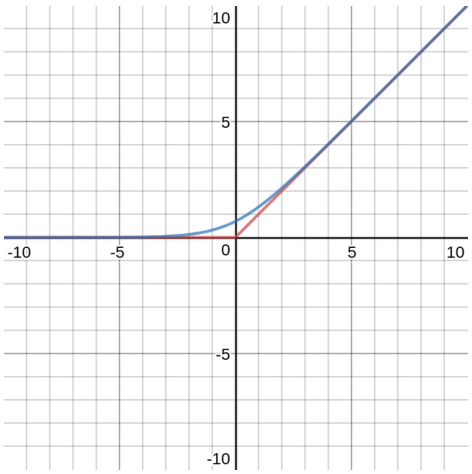

Abbildung 2.5: ReLU(Rot) und Softmax(Blau)

an Bedeutung, das sie ableitbar ist und somit in den versteckten Schickten des Neuronalen Netzes verwendet und trainiert werden kann (vgl. [Pat] Abschnitt *What is the Purpose[...]*).

Die Softmax-Funktion wird in der Ausgabeschicht verwendet für Multiklassen-Klassifizierung. Die Softmax-Aktivierungsfunktion kann innerhalb der versteckten Schichten verwendet werden.

2.4.4 Optimierungsfunktion

Die Optimierungsfunktion hilft, für eine (unbekannte) Funktion ein Extremum zu finden und wird konkret benötigt, um das Minimum der Fehlerfunktion zu erreichen.

Im Rahmen des Machine-Learning verwaltet die Optimierungsfunktion ebenfalls Trainingsparameter, z.B. die Lernrate, eine Beschleunigungs- und Verfallslogik. Ebenso oft Bestandteil sind Funktionen, welche eine zufällige Verschiebung bewirken. Grund hierfür ist eine Schwäche der meisten Optimierungsfunktionen, sich auf ein lokales Extremum einzupendeln.

Im Folgenden wird der Gradientenanstieg und dessen Optimierungen vorgestellt. Dieser Unterabschnitt stellt im Wesentlichen eine Auswahl und Aufbereitung von Sebastian Ruders Blogeintrag *An overview of gradient descent optimization algorithms* [Rud] dar.

Der Gradientenanstieg Kern des Gradientenanstiegs bildet die iterative Suche nach dem (globalen) Minimum der Loss-Funktion. Einen beispielhaften Plot einer

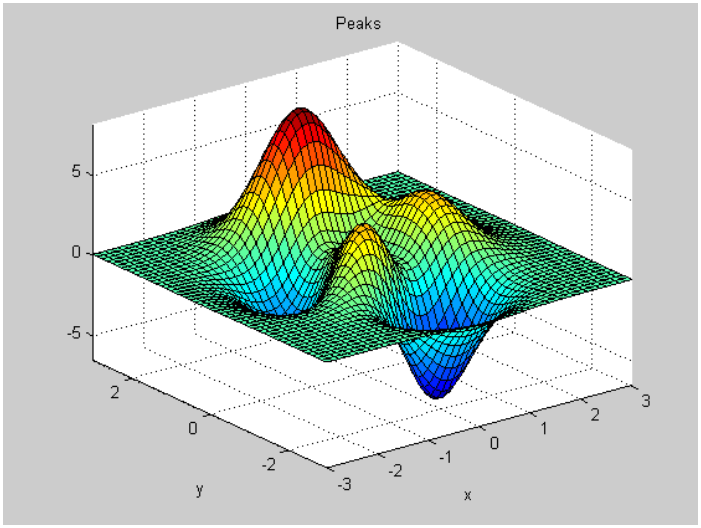

Abbildung 2.6: Beispiel einer visualisierten Verlust-Funktion

Verlustfunktion liefert Abbildung 2.6: Der Verlust ist hierbei auf der Z-Achse eingetragen, die Y- und X-Achse bilden die Gewichte ab. Die vorgestellten Verfahren sind ebenfalls in Räumen höherer Dimension möglich.

Stellt man sich diese Funktion als (hügelige) Landschaft vor (Siehe Abbildung 2.6), so hilft als Veranschaulichung *the blindfolded hiker*: Ein Wanderer mit Augen-

binde steht an einem Berghang und möchte sicher das nächste Tal erreichen. Hierfür findet er zunächst heraus, wo es gerade bergab geht, und tätigt einen vorsichtigen Schritt. Dies wiederholt er so lange, bis er an einer Stelle steht, wo es ausschließlich bergauf geht.

Um mathematisch *die Hanglage* zu bestimmen, benötigt man den Gradienten der Verlustfunktion. Prinzipiell ist es möglich, anhand des Gradienten bereits alle Extrema der Verlustfunktion zu bestimmen - allerdings stellt dies vor den Anwendungen, welche mehrere tausend Dimensionen aufweisen können, keine technisch umsetzbare Variante dar.

Vor diesem Hintergrund wird nicht direkt versucht, die **besten** Gewichte zu bestimmen, sondern lediglich ausgehend von den bisherigen **bessere**. Hierfür werden die bestehenden Gewichte um einen Bruchteil des Gradienten verändert. Diesen Bruchteil bezeichnet man als **Lernrate** (engl. learning rate, oft auch Schrittgröße).

Der Gradient kann hierbei numerisch approximiert werden, oder über die Ableitung gebildet werden. Im Rahmen des Machine Learnings ist die Ableitung üblich - die oben vorgestellten Aktivierungsfunktionen besitzen einfach zu berechnende, bzw. bekannte Ableitungen und erzeugen keine Rechenlast, wie dies bei einer Approximation der Fall ist.

Die Wahl der Lernrate ist für die Anwendung des Algorithmus entscheidend: Wird bis zu einem Minima iteriert [1], so kann die Terminierung des Algorithmus sehr hohe Zeit in Anspruch nehmen. Wird eine gewisse Anzahl an Iterationen durchgeführt, so liegen die Ergebnisse eventuell sehr nah an den zufällig initialisierten Gewichten.

Auf der anderen Seite kann eine hohe Lernrate zu einem *herumspringen* der Gewichte führen, und eine tatsächliche Optimierung verhindern.

Ausgehend von diesem Grundverfahren, gibt es mehrere Ansätze den Gradienten-

[1] Das Verfahren des Gradientenanstiegs kann ggfs. nicht terminieren - in den tatsächlichen Implementierungen wird deswegen immer eine gewisse Anzahl an Iterationen durchgeführt oder andere Kontrollparameter aufgestellt.

anstieg zu verbessern:

Stochastischer Gradientenanstieg Im Rahmen des stochastischen Gradientenanstieg wird das Set der Trainingsdaten in jeder Epoche zufällig Sortiert.

Der durchgeführte Schritt ist also in einer zufälligen Reihenfolge, und kann dazu führen, dass sich der Algorithmus nicht dem nächsten Minimum nähert, sondern sich zunächst hiervon entfernt.

Dies stellt allerdings einen Vorteil dar: Durch dieses Verhalten wird verhindert, dass lediglich ein lokales Minimum gefunden wird. Allerdings terminiert der stochastische Gradientenanstieg nicht so schnell, bzw. nicht so nah am reellen Minima, wie der normale Gradientenanstieg, zumindest ohne Einbindung weiterer Verbesserungen.

Mini-Batch-Gradientdescent In dieser Variante wird die Veränderung der Gewichte nicht anhand des Gradientens eines einzelnen Datensatzes, sondern aufgrund einer Teilmenge (*Batch*) geupdatet.

Diese Variation reduziert die auftretende Varianz innerhalb der Schritte und sorgt somit für eine besser absehbare Konvergenz. Zusätzlich wird Batch-Verarbeitung von Vektoren innerhalb vieler Prozessoren und Grafikkarten unterstützt. Die **Batch-Größe** stellt den zweiten wichtigen Hyperparameter dar und wird im Normalfall (aus technischen Gründen) als Zweierpotenz gewählt.

Es ist weiterhin üblich, diese Variante mit dem stochastischen Gradientenanstieg zu kombinieren. In jeder Epoche werden hierbei die Batches zufällig sortiert. Diese Kombination adressiert die Probleme des stochastischen Gradientenverfahrens bezüglich der Terminierung und besitzt trotzdem die Möglichkeit lokale Minima zu verlassen.

Während nun bereits ein gutes Grundverfahren vorliegt, gibt es noch weitere Möglichkeiten und Hyperparameter, um die Optimierung zu beschleunigen:

Momentum Eine einfache Verbesserung stellt die **Beschleunigung** dar: Sie verwaltet für jedes Gewicht einen Parameter, welcher bei einer Änderung angepasst wird. Werte, die häufige bzw. große Änderungen erfahren, erhalten eine größere Lernrate. Hiermit soll erreicht werden, dass ausschlaggebende Werte besser berücksichtigt sind und der Algorithmus im Gesamten schneller terminiert.

Ergänzend zur Beschleunigung gibt es ebenfalls den **Verfall**. Dieser sorgt für eine regelmäßige Abnahme der Lernrate (d.h. bei jedem Durchlauf), und sollte mit der Beschleunigung abgestimmt werden. Der Verfall sollte nicht zu hoch gewählt werden, damit die Beschleunigungen bzw. die effektiven Lernraten stets bemerkbar bleiben.

Adadelta Eine weitere Verbesserung stellt der *Adaptive Delta Gradientdescent* dar. Ziel dieser Variante ist es, die Lernrate immer passend zu wählen - in *steilen* Gebieten *große* Schritte zu vollziehen und in solchen nahe des Minimas *kleinere*.

Adadelta addiert auf die bisherige Lernrate einen Wert abhängig von der aktuellen Änderung und den durchschnittlichen Gradienten-Verläufen über alle bisherigen Änderungen. Die konkrete Mathematische Ausarbeitung, sowie Beweise über die Terminierung und Verbesserungen der Trainingszeiten finden sich weiterführend im Whitepaper von Matthew D. Zeiler et. Al [Zei].

Während die Beschleunigung eine *First-Order*-Änderung darstellt, da nur der aktuelle Wert berücksichtigt wird, nimmt Adadelta auf den gesamten Verlauf Rücksicht und wird dahingehend als *Second-Order* bezeichnet.

Die Hauptstärke von Adadelta liegt darin, dass bewiesenermaßen eine gute Lernrate erzeugt wird ohne Input des Entwicklers. Dies spart viele Parameter im Gesamtprozess des Machine Learnings und reduziert somit ebenfalls Fehlerquellen.

2.4.5 Bewertung der Klassifizierung

Die Bewertung einer Klassifizierung erfolgt anhand dessen, wie viele Testdaten korrekt klassifiziert wurden. Dieses Verfahren eignet sich sowohl für die Binäre-, als auch für eine Multiklassen-Klassifikation

Die (relative) Genauigkeit ergibt sich als:

$$acc = \frac{\#\{t \in TestSample | guess(t) == class(t)\}}{\#TestSample} \tag{2.13}$$

2.5 Neuronale Netzwerke

Dieser Abschnitt widmet sich den Konzepten künstlicher neuronaler Netze. Als Hauptquelle dient der Artikel *Building a neural network from scratch* von David Selby [Sel] sowie die Vorlesung *Artificial Intelligence* von Dr. Stroetmann [Str].

Neuronale Netze erhielten ihren Namen, da man zu Beginn der Forschung dachte, dass das menschliche Gehirn wie ein *computational graph* funktionierte. Diese These wurde biologisch weitgehend widerlegt und deswegen werden die neuronalen Netze der Informatik mit dem Zusatz *künstlich* markiert.

Im Folgenden wird zunächst der Aufbau des Modells und anschießend das Training vorgestellt.

2.5.1 Modell künstlicher neuronaler Netze

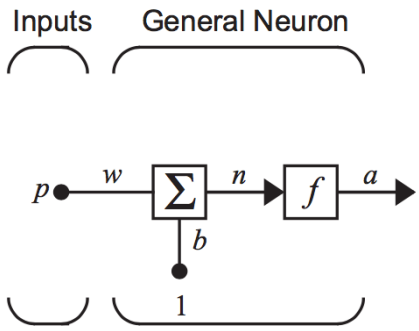

Abbildung 2.7: Einzelnes Neuron

Grundidee des Modells bildet das Konzept eines **Neurons**. Dieses erhält Eingabewerte, und sobald ein gewisser Schwellwert erreicht wurde, *feuert* es sein Signal ab, um andere Neuronen zu kontaktieren oder Handlungen hervorzurufen.

Im Rahmen der Informatik äußert sich diese Neuronen-Logik durch eine Aktivierungsfunktion, die üblicherweise ohne Bedingung eine Ausgabe erzeugt. Dies zeigt Abbildung 2.7.
Diese Neuronen werden zu einen Graphen, welche sich als *Schichten* anordnen.

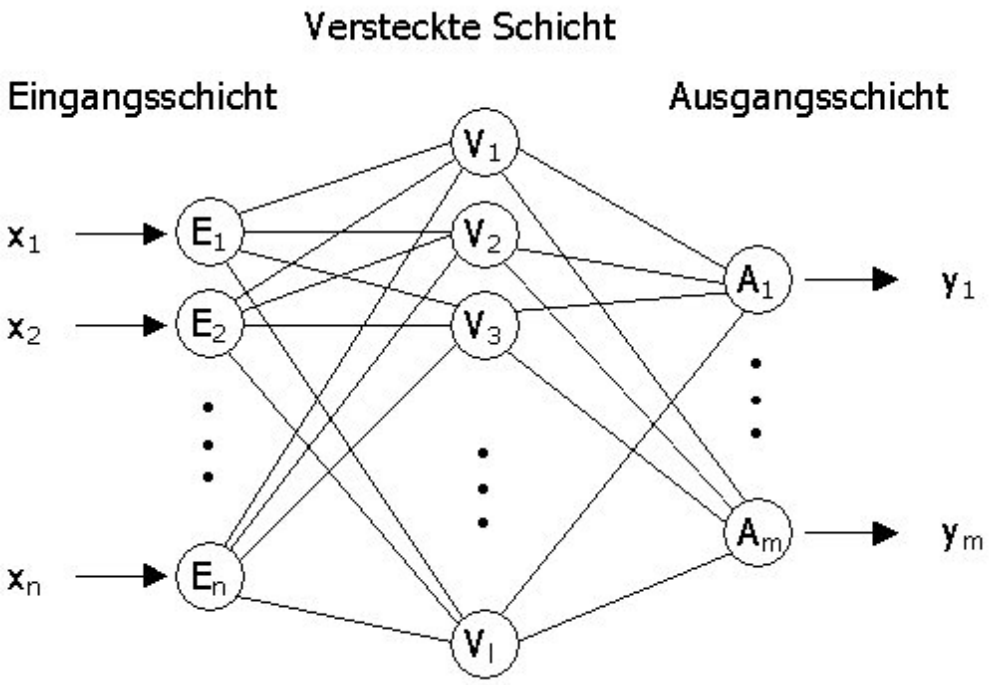

Abbildung 2.8: Modell eines neuronalen Netzwerkes

Jedes Neuron einer Schicht erhält Eingaben von jedem Neuron der vorhergehenden. Diese Eingaben werden zusätzlich gewichtet.

Die Eingabe in das neuronale Netzwerk erfolgt über den Inputlayer, welcher keine Aktivierungsfunktion hat. Die Ausgabe des neuronalen Netzwerkes erfolgt in der sog. Ausgabeschicht, welche je nach Art des Netzes einen (für Regressionen), zwei (für binäre Klassifikationen) oder n (für n-Klassen Multiklassifikation) Knoten besitzt.

Die Neuronen zur Berechnung befinden sich in den *Hidden Layers*. Gibt es mehr als einen Hidden-Layer spricht man von einem *deep neuronal network*[2]. Dieses vollständige Modell zeigt Abbildung 2.8.

Um das Modell zu trainieren, muss ebenfalls der Fehler der Schätzung minimiert

[2]Grund hierfür ist die Funktion der tieferen Schichten - Anstatt nur die Eingabe zu gewichten, werden hier weitere Features erkannt bzw. erzeugt, welche nur für den Algorithmus erkennbar sind.

werden. Hierbei werden die Konzepte der linearen und logistischen Regression verwendet, jedoch mit dem Zusatz, das anstatt eines einzelnen Gewichtsvektors eine Gewichtsmatrix angepasst werden muss.

2.5.2 Forward Propagation

Unter der *Vorwärtsausbreitung* versteht man den Algorithmus, welcher einen Eingabevektor durch alle Gewichtsvektoren und Schichten transfomiert.

Dieser *Feed Forward*-Prozess kann sowohl iterativ über alle Vektoren erfolgen, oder zusammengefasst als Matrizenoperation.

2.5.3 Backward Propagation

Die *Rückwärtsausrichtung* bezeichnet den Algorithmus, welcher die Gewichte anhand des gemessenen Fehlers nachjustiert.

Hierbei wird in gleichem Maße wie in der logistischen Regression vorgegangen, mit dem Unterschied, dass die Gewichte in einer Matrix vorliegen.

Die Eigenschaften der Aktivierungsfunktionen bezüglich ihrer Ableitung finden hier im besonderen Maße Anwendung, denn um von der Ausgabeschicht auf den letzten Hidden-Layer nachzujustieren, benötigt man die erste Ableitung. Um auf den nächsten Hidden Layer Einfluss zu nehmen, muss die zweite Ableitung gebildet werden (usw.). Eine mathematische Ausarbeitung findet sich unter [Ola] *Computational Victories*.

2.5.4 Training

Das Training bezeichnet den (iterativen) Prozess, mit den vorliegenden Trainingsdaten zunächst Forward-Propagation durchzuführen, um anschließend mittels Backward Propagation das Netz auszurichten.

Der Umfang dieses Trainings bleibt dem Anwender überlassen. Es ist möglich, bessere Ergebnisse zu erzielen, indem man mit denselben Daten häufiger trainiert. Einen solchen wiederholten Trainingsdurchlauf nennt man eine **Epoche**.

2.5.5 Bewertung des neuronalen Netzes

Die Bewertung des neuronalen Netzes erfolgt, je nach Art des Ergebnisses, analog wie die der linearen oder logistischen Regression.

2.5.6 Einflüsse auf den Trainingserfolg

Zum Abschluss dieses Abschnittes werden noch einmal die *Stellschrauben* vorgestellt, anhand derer Änderungen des Trainingserfolges erzielt werden können.

- **Netzaufbau und Struktur:** Die Anzahl der Knoten, Schichten, und Einstellungen zur Verknüpfung können variiert werden.
- **Optimierungsfunktion:** Hierbei kann der grundlegende Algorithmus (Gradient Descent oder Stochastic Gradient Descent), sowie Trainingsparameter (Lerngeschwindigkeit, Beschleunigung, Verfall) gewählt werden.
- **Aktivierungsfunktion der Neuronen**
- **Menge der Trainingsdaten:** Eine größere Menge an Trainingsdaten hilft maßgeblich, den Sachverhalt besser erfassen zu können. Auch sind große Datenmengen notwendig, um bei komplexeren Netzen *overfitting* zu vermeiden.
- **Anzahl der Features**
- **Anzahl der Epochen und Iterationen**

Konkrete Anwendungen dieser Parameter und die damit zusammenhängende Ergebnisse finden sich im Kapitel 4 dieser Arbeit.

3 SQLServer 2017 und R

In diesem Kapitel werden zunächst die Umgebung des SQL-Servers 2017 sowie die Programmierprache R kurz vorgestellt, bevor in Abschnitt 3.3 eine konkrete Umsetzung der unter Kapitel 2 gezeigten Algorithmen mit R erfolgt.

3.1 SQL-Server 2017

In diesem Abschnitt werden zunächst Grundlagen des SQL-Server 2017 vorgestellt, und anschließend vor dem Hintergrund des Machine Learnings zwei besondere Dienste, der ML-Server und die R-Services gesondert erläutert.

Microsoft's SQL Server ist ein relationales Datenbankmanagementsystem (kurz RDBMS), hat sich allerdings zu einer größeren Plattform für Business Intelligence, Data Mining, Reporting und ETL-Prozesse weiterentwickelt. Es wird ein Dialekt von SQL verwendet, das sog. Transact-SQL (Siehe [Woo16] Seite 4 Absatz 7). Seit der Version 2016 werden ebenfalls die beiden Dienste R-Services und der Machine-Learning Server unterstützt, seit 2017 sind sie nativ integriert.

Neben der reinen Datenhaltung bietet der SQL-Server folgende Schlüsselelemente (vgl. [Mica] S9-12):

1. (Web-) Schnittstellen um Remote auf Daten zuzugreifen
2. Möglichkeiten zur prozeduralen Programmierung
3. Unterstützende Tools für ETL-Prozesse und Reporting
4. Schnittstellen zu anderen Programmiersprachen, wie JavaScript, C# und Python

5. Security-Dienste zur Verschlüsselung der Daten on Rest sowie Nutzer und Rechteverwaltung
6. Performance-Optimierung, z.B. durch In-Memory-Column-Stores

Diese Features führen dazu, das der SQL-Server zu den Marktführern der RDBMS´ gezählt wird. Ein weiteres, besonders wichtiges Feature für diese Arbeit stellen die R-Services dar, welche innerhalb des ML-Servers bereitgestellt werden:

3.1.1 Machine-Learning-Server

Der Microsoft Machine-Learning-Server stellt eine Erweiterung des 2015 eingeführten R-Servers dar (vgl. [Micc] Absatz 1f) und erweitert diesen um eine Python-Engine. Die unter R-Services vorgestellten Funktionen werden ebenfalls über den Machine-Learning Server bereitgestellt.

Prinzipiell ist der ML-Server eine eigene Anwendung und kann ohne einen SQL-Server in Betrieb genommen werden. Die Stärken des ML-Servers liegen allerdings in der engen Integration des SQL-Servers: Der ML-Server kann auf Datenbanken, Sichten, Funktionen und Nutzerverwaltung des SQL-Servers zugreifen, und bildet somit eine umfangreiche Entwicklungsumgebung.

Insbesondere kann der Machine-Learning Server aber auch als einfache Webschnittstelle benutzt werden, um bereits erzeugte Modelle zu nutzen.

Eine weitere Besonderheit ist die nahtlose Verwendung von Skripts in der Azure-Cloud.

Möglichkeiten in R Die Sprache R besitzt verschiedene Optionen, um Machine-Learning Modelle zu erzeugen. Neben der Implementation *von Grund auf* gibt es eine Vielzahl von Paketen und Bibliotheken.

Für die lineare und logistische Regression werden die Bibliotheken *RevoscaleR* und *MicrosoftML* von Microsoft benutzt (Die Dokumentation findet sich unter [Ste]). Sie wird bereits mit dem SQL-Server geliefert. Hauptargument für diese Umsetzung waren die gründliche Dokumentation von Microsoft, die eine Benutzung in-

nerhalb des SQL-Servers bereits behandelt, sowie die gemeinsame Produktfamilie, welche einen einheitlichen Technik-Stack ergibt.

Möglichkeiten in Python Neben den R-Services wird ebenfalls eine Python-ML-Umgebung (welche [1] ebenfalls ML-Server heißt) unterstützt. Dieses Open Source Projekt ist auf Github [Micb] zu finden und stellt eine Alternative zu den in R vorgestellten Methoden dar.

Der ML-Server selbst ist in Python implementiert.

3.1.2 R-Services

Die R-Services stellen eine Erweiterung des SQL-Servers dar, und bilden eine integrierte Plattform für R-Programme. Die wichtigsten Erweiterung zu einer *normalen* R-Plattform bietet sich durch die **ScaleR**-Bibliothek, welche es ermöglicht die Datenobjekte von R persistent als Datenbank oder Datei zu speichern (vgl. [Woo16] Seite 7 Absatz 9).

Des Weiteren werden hier auch Parallelisierung und ggfs. Verteilung (bei einem verteilten SQL-Server-Cluster) behandelt.

Zugriff auf die R-Services bieten sich über zwei Möglichkeiten: Der Einbindung von R-Skripten direkt in T-SQL, oder (remote) über einen R-Workspace.

Die R-Services unterstützen ebenfalls das modulare Verwenden von R-Paketen, diese werden optional bei Anfragen mitgesendet oder sind auf dem SQL-Server installiert.

Es gibt zwei Wesentliche Vorteile, die für eine Verwendung der integrierten R-Services sprechen:

1. **Bewegung des Codes zu den Daten:** Die Operationen werden dort ausgeführt, wo die Daten liegen. Somit entfallen Probleme bezüglich großen Datentransfers, Latenz sowie der Last auf dem Client, die Berechnungen durchzuführen (vgl. [Woo16] Seite 7 Absatz 6). Vor Allem im Bereich des Machine-

[1] gnädigerweise

Learning werden sowohl rechenintensive Operationen benötigt, als auch große Datenmengen, zwei Probleme die hiermit adressiert werden.

2. **Integration in bestehende Systeme:** Viele Unternehmen benutzen bereits eine Instanz des SQL-Servers für ihre Datenhaltung, und benutzen periphere Ansätze zur Analyse und Forecasting. Durch die Verwendung der R-Services können hier Lösungen auf bestehende Systeme erarbeitet werden.

Dennoch entstehen (derzeit) auch Probleme bei der Verwendung der integrierten R-Services, konkret nach Wahl der Entwicklungsplattform:

Bei Verwendung der R-GUI oder R-Plattform wird keine direkte Einsicht in die Datenstrukturen des SQL-Servers gewährt. Insofern müssen benötigte Tabellen sowie ihre Definitionen bekannt sein, bzw. ein zweites Tool zur Einsicht der Datenbank bereitstehen. Vor allem bei komplexeren Datenbankabfragen, die Joins und Aggregationen beinhalten, stellt diese Einschränkung ein starkes Handicap dar.

Bei Verwendung der R-Services innerhalb des SQL-Server-Managementstudios (Oder anderen SQL-fokussierten Umgebungen) liegt das Problem darin, dass das R-Skript als Text übergeben wird, und somit nicht in der Vorschau *kompiliert* wird. Liegt ein Fehler im Code vor, so zeigt sich dieser erst zur Laufzeit - teilweise nachdem bereits längere Operationen durchgeführt wurden.

Aufgrund dieser beiden Probleme sollte die Entwicklungsumgebung dahingehend gewählt werden, wo die komplexeren Anforderungen der Lösung liegen: Im Falle komplexer Datenaufbereitung sollte mit integriertem Skript gearbeitet werden, im Falle komplexer R-Anfragen mit der R-GUI [2]. Optional bietet sich eine Auswahl nach bestehendem Vorwissen der Sprachen SQL und R an.

[2] Prinzipiell lassen sich ebenfalls alle Aggregationen und Aufbereitungen in R vornehmen. Die speziellen Operationen und Strukturen von R machen dies allerdings v.a. für Neulinge teilweise deutlich komplexer als SQL.

3.2 Programmiersprache R

Die Sprache R stellt vor allem für Statistiker und Psychologen ein Standard-Tool dar, dennoch ist sie auch bei Data-Scientisten beliebt für ihre vielseitigen Plot- und Modellierungs-Möglichkeiten. Im Rahmen dieser Arbeit wurden alle Modelle mithilfe von R erstellt, was innerhalb dieses Abschnittes vorgestellt wird:

R ist eine Sprache, sowie eine Entwicklungsumgebung für statistische Berechnungen und Grafiken. R ist ein GNU-Projekt und beruht auf der Sprache *S*, welche von John Chambers et al. entwickelt wurde. R stellt eine Implementation von S dar (vgl. [Fou] Absatz 1) und liegt als Open Source Projekt vor.

R bietet eine große Bandbreite an statistischen Funktionen (z.B. lineare und nichtlineare Regression, Klassifikation und Signifikanztests) sowie grafische Aufbereitungen dieser und ist hochgradig modular (vgl. [Fou] Absatz 2).

Die größten Stärken von R liegen neben der einfachen Anwendung statistischer Funktionen in der Aufbereitung als Plots. R erzeugt schnell verständliche Grafiken der Daten, bieten erfahrenen Nutzern allerdings viele Optionen, exakt benötigte Darstellungen zu erzeugen.

R Umfeld R umfasst folgende integrierte Dienste (Siehe [Fou] Absatz 5f):

1. Eine Speichereinheit und Daten-Engine
2. Eine Umgebung für Berechnungen auf Listen, Vektoren und insbesondere Matrizen
3. Eine Sammlung an Werkzeugen zur Datenanalyse, statistischen Auswertung und Erzeugung von Grafiken
4. Eine Programmiersprache, welche auf Bedingungen, Schleifen und nutzerdefinierte Funktionen eingeht

Für rechen- und zeitintensive Operationen kann zusätzlich C und C++ Code zur Laufzeit eingebunden werden. Ebenso kann man mit C direkt Objekte manipulieren.

Die Funktionalitäten von R können über ein Paket-System erweitert werden. Das wichtigste Paket innerhalb dieser Arbeit stellt *RevoScaleR* dar, welches eine persistente Speicherung von Datenobjekten in Datenbanken und Files ermöglicht.

Besonderheiten in der Programmierung

Die wichtigste Besonderheit in R ist, das jedes Objekt als Vektor aufgefasst wird. Ein einzelner Wert wird ebenfalls als Vektor der Größe 1 betrachtet.

Vektoren können für arithmetische Ausdrücke verwendet werden, in diesem Fall werden die Operationen Element für Element ausgeführt. Zwei Vektoren, welche in einer Anweisung vorkommen, müssen nicht die selbe Länge besitzen. Falls dies nicht der Fall ist, ist die Ausgabe der Anweisung ein Vektor der Länge des längsten Vektors. Die kürzen Vektoren werden solange wiederholt, bis sie die Länge des längsten Vektors erreicht haben.

Insbesondere Konstanten werden auf jedes Element angewendet (vgl. [WNV18] Seite 13 Abschnitt 2.2 *Vektorarithmetik* Absatz 1).

Diese Eigenschaft der Vektoren ist vor dem Hintergrund mit Datenbanken zu arbeiten, ein zweischneidiges Schwert: Zum Einen werden die Operationen und Anweisungen sehr *einfach* und übersichtlich (Hilfsstrukturen für Schleifen entfallen), allerdings bringt v.a. die Wiederholung der kleineren Vektoren erhebliche Fehlerquellen mit sich.

Ein Faktor ist ein Vektor mit einem fest definierten Wertebereich (z.B. ein Charakter-Vektor, Siehe auch [WNV18] Kapitel 4 *Ordered and Unordered Factors* Absatz 1).

Ein *Array* stellt in R eine Kombination aus einem Wert-Vektor und einem Dimensions-Vektor dar. Die Dimension gibt hierbei eine Form für den Wert-Vektor vor, und bestimmt in welcher Reihenfolge und ggfs. mit welchen Eigenschaften Operationen ausgeführt werden. Für arithmetische Operationen zweier Arrays wird ebenfalls die o.G. *Recycling Rule* angewendet. Im Falle einer Anweisung eines Arrays und eines Vektors, wird zunächst versucht, aus dem Vektor ein Array derselben Dimension zu erzeugen. Eine Matrix stellt ein zweidimensionales Array dar.

Ein *Data-Frame* stellt eine besondere Form einer Liste dar, die folgende Eigenschaften erfüllen muss (Siehe [WNV18] Abschnitt 6.3 *Data-Frames* Absatz 1f):

1. Ein Data-Frame darf lediglich Vektoren, Matrizen, Faktoren und Data-Frames enthalten.
2. Alle Vektoren und Faktoren des Data-Frames müssen die selbe Länge besitzen, Matrizen zusätzlich eine einheitliche Breite.
3. Charakter- und String-Vektoren werden zu Faktoren vereinfacht.

Data-Frames sind dahingehend wichtig, da eine Tabelle aus dem SQL-Server als Data-Frame interpretiert wird.

3.3 Machine Learning im SQL-Server 2017

Innerhalb dieses Abschnittes befinden sich Code-Beispiele zur Umsetzung der in Kapitel 2 vorgestellten Algorithmen.

Es werden im Folgenden kurz die Einbindung der R-Skripte in TSQL behandelt, anschließend werden nur die R-Skripte für die einzelnen Punkte erläutert.

Verwendung von R im SQL-Server Um R im SQL-Server zu benutzen, wird die Stored Procedure *sp_execute_external_script* benötigt. Im Folgenden ein einfaches Beispiel:

```
1 EXECUTE sp_execute_external_script
2 @language = N'R',
3 @script = N'
4   mytextvariable <- c("hello", " ", input_data);
5   OutputDataSet <- as.data.frame(mytextvariable);',
6 @input_data = N' SELECT name FROM readers'
7 WITH RESULT SETS (([Greetings] char(20) NOT NULL));
```

Hierbei wird in Zeile 2 zunächst die Sprache als Parameter übergeben, in Zeile 4 wird innerhalb des R Skriptes ein Begrüßungs-String erstellt, welcher in Zeile 5 als Ausgabe wiedergeben wird.

In Zeile 6 wird die Inputvariable definiert, an dieser Stelle sind SQL Befehle und gültige T-SQL Variablen möglich. Es können beliebig viele Inputvariablen definiert werden.

In Zeile 7 wird die Ausgabe in Tabellenform überführt. Diese Zeile ist nicht zwingend notwendig.

Dieses Schema bleibt allen Skript-Aufrufen gleich. Im Folgenden werden nur die R-Skripte vorgestellt.

3.3.1 Lineare Regression

Für diese Form der Regression gelten innerhalb des Paketes MicrosoftML folgende Bedingungen:

1. Strings und kalendarische Daten müssen über einen Faktor realisiert. werden
2. Der Ausgabewerte ist eine reelle Zahl.

Um ein Modell für die lineare Regression zu erstellen, sind in R nur wenige Zeilen notwendig:

```
1 formel <- C ~ A+B;
2 model <- rxLinMod(formula=formel, data=TrainingsData);
3 serializedModel <- data.frame(payload = as.raw(serialize(model,
      connection=null)));
```

In der ersten Zeile wird zunächst eine allgemeine Formel definiert. Diese Formel ist zu interpretieren als $f : (A\ x\ B) \rightarrow C$, das '+' ist hierbei nicht als Addition zu verstehen.

In Zeile 2 wird das Modell mithilfe der Bibliothek RevoscaleR und dem Methodenaufruf rxLinMod erstellt **und** trainiert. Als Parameter werden die Formel und die Trainingsdaten benötigt.

In der dritten Zeile findet eine Serialisierung des Modells statt - dies ist nicht notwendig für eine direkte Verwendung, ermöglicht allerdings das Speichern des Modells innerhalb des SQL-Servers als Blob.

Um das Modell anzuwenden, reichen ebenfalls wenige Zeilen R-Skript:

```
1 model <- unserialize(as.raw(serializedModel));
2 C <- rxPredict(model,data.frame(TestData));
```

Hierbei wird zunächst in Zeile 1 das serialisierte Modell wieder nutzbar gemacht.

In Zeile 2 wird die Methode *rxPredict* der RevoScaleR-Bibliothek aufgerufen, welche aus den zu testenden Daten und dem Model eine Prognose erstellt.

3.3.2 Klassifikation

Für die binäre Klassifikation mit RevoscaleR gelten folgende Bedingungen:

1. Die Klasse stellt einen Faktor mit Level 2 dar.
2. Der Ausgabewert ist eine Wahrscheinlichkeit, mit der die Ausprägung positiv ausfällt
3. Es kann gleichzeitig nur eine Klasse überprüft werden

Der R-Code verhält sich parallel zum Code der linearen Regression:

```
1 formel <- rain ~ temperature+humidity;
2 logitmodel <- rxLogit(formula = form, data = TrainingsData);
3 rainPropability <- rxPredict(model, data.frame(TestData));
```

Als Beispiel wurde hierbei die Voraussage gewählt ob es regnet, anhand von Temperatur und Luftfeuchtigkeit.

3.3.3 Neuronale Netze

Es ist möglich, die im Abschnitt 2.5 vorgestellten Konzepte direkt in R umzusetzen. Ein gutes Tutorial liefert hierbei [Sel], welcher eine Schritt-Für-Schritt Anleitung und Erklärung bietet ein eigenes neuronales Netz zu entwerfen. Das Tutorial von Selby setzt einen ähnlichen Blogeintrag von Denny Britz (Siehe [Bri]) in R um.

Innerhalb dieser Arbeit wird allerdings das Paket *MicrosoftML* verwendet.

Netz-Definition

Eine der wichtigsten Einstellung stellt die Definition des neuronalen Netzes dar. Für diese wird innerhalb der Microsoft-Umgebung (Innerhalb des ML-Servers, Azure und R-Services) einheitlich eine Definition in *Net#* verwendet. Diese Notation definiert das gesamte neuronale Netz, und stellt einen einheitlichen und übertragbaren Standard in der Microsoft Umgebung dar. Ein einfaches Beispiel:

```
1 netDefinition <- ("
2  input Data auto;
3  hidden Hidden[25] sigmoid from Data all;
4  output Result[2] from Hidden all;
5 ")
```

In Zeile Zwei wird die Eingabeschicht mit dem Namen *Data* und einer automatisch erkannten Größe erstellt.

In Zeile Drei wird die versteckte Schicht *Hidden* mit 25 Knoten, einer Verbindung zu allen Knoten in Data und der Aktivierungsfunktion *Sigmoid* gewählt.

In Zeile Vier wird die Ausgabeschicht *Result* mit zwei Ausgabeknoten definiert. Es handelt sich um eine binäre Klassifikation. Die Aktivierungsfunktion wird auf den Standardwert *sigmoid* gesetzt.

Optional ist es möglich, die Größe eines hidden Layers in der Form [X,X,Y] anzugeben. Dies bedeutet, das zunächst zwei Layer mit X Knoten und anschließend ein Layer mit Y Knoten vorliegt, welche eine Einheit bilden. Die anderen Parameter, z.B. die Aktivierungsfunktion, werden für alle Teilschichten übernommen.

Es werden an ein neuronales Netzwerk innerhalb von net# folgende Anforderun-

gen gestellt:

- Jedes neuronale Netz besitzt mindestens eine Eingabeschichte und genau eine Ausgabeschicht.
- Die Anzahl der Knoten der Ausgabeschicht entspricht der Klasse des neuronalen Netzes (Ein Ausgabeknoten für Regression, zwei für binäre Klassifikation, n für eine Klassifikation von n-Labeln).
- Verbindungen müssen azyklisch sein, anders ausgedrückt, sie dürfen keine Kette von Verbindungen bilden, die zurück zum ursprünglichen Quellknoten führt.
- Um eine Vorhersage mit dem Modell zu machen, werden bei den Eingabedaten mindestens alle in der Formel angegebenen Features benötigt.

Nach diesem Schema lassen sich beliebig komplexe neuronale Netze definieren. Es gibt weitere Möglichkeiten, die Netzdefinition anzupassen:

- Auswahl von Aktivierungsfunktionen (z.B. Sigmoid, Softmax, Linear)
- Deklaration Konvolutionsbündeln, d.h. Schichten definieren, welche sich mit zusätzlichen Gewichten gegenseitig beeinflussen.
- Deklaration von Selektionsbündeln, d.h. Auswahlkriterien, nach welchen die Schichten verknüpft werden.
- Deklaration von Poolingbündeln, d.h. Schichten und Teilnetze, welche eine ähnliche Funktion erfüllen, allerdings nicht trainiert werden.

Regression

Ein neuronales Netz mithilfe des Paketes zu erstellen, ist ähnlich einfach wie ein normales Modell hierfür:

```
1 netDefinition <- ("
2   input Data auto;
3   hidden Hidden[25] sigmoid from Data all;
4   output Result[1] linear from Hidden all; ")
```

```
form <- C ~A+B;
model <- rxNeuralNet(
    formula = form,
    data = TrainingsData,
    type = "regression",
    netDefinition = netDefinition,
    numIterations = 100,
    normalize = "yes"
);
```

Zunächst wird ein Netz definiert, welches als Ausgabeschicht einen einzelnen Knoten mit linearer Ausgabefunktion besitzt. Anschließend wird die Formel aus dem obigen Beispiel für lineare Regression definiert, und das Modell mit der Funktion *rxNeuralNet(...)* erstellt. Diese erhält gegenüber anderen Modellen zusätzliche Parameter für die Netzdefinition, die Iterationen und den Typ des neuronalen Netzes.

Des Weiteren können Einstellungen über die Optimierungsfunktion, Initialisierung der Gewichte, sowie Lerngeschwindigkeit, Verfall und Beschleunigung vorgenommen werden.

Multiclass-Labeling

Um eine Multiklassen-Klassifizerung vorzunehmen, benötigt man einen ähnlichen Aufbau:

```
netDefinition <- ("
    input Picture auto;
    hidden Hidden[250] sigmoid from Picture all;
    output Species[4] softmax from Hidden all; ")
form <- Species ~Sepal.Length+Sepal.Width+Petal.Length+Petal+Width;

model <- rxNeuralNet(
    formula = form,
    data = TrainingsData,
    type = "multiclass",
    netDefinition = netDefinition
);
```

Als Beispiel ist die Klassifizierung eines Bildes in eine von 4 Lotus-Spezien gewählt. Zu betonen ist, das die Ausgabe der Vorhersage 5 Werte erzeugt: Einen für die wahrscheinlichste Spezies, und für jede Spezies die Wahrscheinlichkeit.

4 Fallbeispiel: Prognose von Taxifahrten

Innerhalb dieses Kapitels wird das Fallbeispiel der Taxidaten behandelt. Zunächst erfolgt eine Zielsetzung, anschließend in dem Abschnitt 4.2 eine Vorstellung des Versuchsaufbaus und der zugrunde liegenden Daten.

Hauptteil dieses Kapitels bildet in den Abschnitten 4.3.1 bis 4.3.3 die Ausarbeitung und Analyse verschiedener Prognosen mithilfe neuronaler Netze.

Abschließend findet sich in diesem Kapitel eine Übersicht über die herausgearbeiteten *Best Practices* bei der Arbeit mit neuronalen Netzen im SQL-Server.

Auf die Aufnahme des Quellcodes wird aus Umfang verzichtet. Dieser findet sich in der angehängten CD.

4.1 Ziele und Anforderungen

Ziel des Fallbeispieles ist es, *lohnenswerte* Prognosen anhand von realistischen Daten zu erheben und die Qualität der verwendeten Algorithmen objektiv zu bewerten.

User Stories

Als lohnenswert werden hierbei Fragestellungen bezeichnet, welche für ein Unternehmen einen Mehrwert darstellen. Konkret werden folgende User Stories behandelt:

- Wie viele Taxis brauche ich kommenden Samstagmittag am Time-Square, wenn es sonnig wird?
- Wie viel Umsatz werde ich am ersten Oktoberwochenende machen?
- Zu welchem Ort wird eine Person an einem regnerischen Morgen aus Manhattan fahren?
- Am 23.12 um 01:00 endet die Weihnachtsfeier im Trump-Tower. Wieviele Passagiere wird das Taxi haben?
- Gibt es ein Muster, nach welchem mehr Trinkgeld gegeben wird?
- Wie viel Trinkgeld werden 3 Fahrgäste geben, wenn eine relativ kurze Strecke vom JFK-Airport gefahren wird?
- Wie lange wird ein Fahrgast brauchen, wenn er den Taxifahrer an der Freiheitsstatue bittet *kurz auf ihn zu warten*?
- Am 21.Juni um 14:30 stehen zwei Personen am Central Park bei Nebel. Werden Sie ein grünes oder ein gelbes Taxi nehmen?

Es ist anzunehmen, das einige Prognosen deutlich bessere Ergebnisse liefern als andere. Dennoch sollen bewusst auch die Grenzen von Machine-Learning gezeigt werden.

Die vorgestellten User Stories werden in allgemeinere Modelle umgewandelt.

Anforderungen

Um eine Objektive Bewertung vorzunehmen, werden folgende Kriterien an die Durchführung der Experimente gestellt:

- **Harte Kriterien:** Die Experimente liefern als Resultat eine Genauigkeit.
- **Wiederholbarkeit:** Eine Wiederholung der Tests muss dieselben Resultate liefern
- **Nachstellbarkeit:** Mithilfe dieses Experimentes muss der Leser im Stande sein, die gezeigten Ergebnisse selbst nachstellen zu können

4.2 Eigenschaften der Daten

Innerhalb dieses Abschnittes werden zunächst die Daten vorgestellt, die dem Fallbeispiel zugrunde liegen.

Die vorgestellten Daten haben bereits einen ETL-Prozess durchlaufen. Dieser besteht im Wesentlichen darin, die CSV-Dateien dahingehend aufzubereiten, das amerikanische Nummerierungen (z.B. Angabe von Dezimalzahlen mit '.' anstelle von ',') auf europäische Normen gebracht werden. Prinzipiell entfällt dieser Schritt für eine rein amerikanische Umgebung.

4.2.1 Taxifahrten

Zunächst werden die Daten der Taxifahrten erläutert.

Diese Daten stammen von der Stadt New York [Gova] und wurde von der *Taxi and Limousine Commission* (Kurz: TLC) bereitgestellt.

Die TLC stellt einen Dachverband mehrerer Taxiunternehmen dar und veröffentlicht die Daten - die Erhebung erfolgt in einzelnen, anonymisierten Kleinunternehmen.

Zusätzlich teilen sich die Fahrten in zwei Kategorien auf: *Green* und *Yellow*. Bei grünen Fahrten handelt es sich um Fahrzeuge mit einer anderen Lizenzierung (vgl. [Giu] Absatz 5ff) und besonderen Auflagen. Im Allgemeinen verhalten sich die Fahrten allerdings gleich, insofern werden lediglich Unterschiede aufgelistet, falls diese bestehen.

Attribute und Datentypen

Die folgende Übersicht entspricht der von der NYC bereitgestellten Übersicht (Siehe [Govb]), die Beschreibung wurde übersetzt und eine Spalte für den Datentyp [1] ergänzt.

[1] Wie sie innerhalb des SQL-Servers bezeichnet werden

Name	Beschreibung	Datentyp
VendorID	Ein Code für das Taxiunternehmen, welches die Daten bereitstellt	smallint
pickup_datetime	Uhrzeit und Datum, wann die Fahrt begann	datetime
dropOff_datetime	Uhrzeit und Datum, wann die Fahrt endete	datetime
Passenger_count	Anzahl der Fahrgäste	smallint
store_and_fwd_flag	Angabe, ob die Fahrt direkt hochgeladen wurde, oder ob die Fahrt zwischengespeichert wurde vor einem Upload	bit
RatecodeID	Ein Code für die Rate, welche für die Taxifahrt bezahlt wurde	smallint
PULocationID	Ein Code für die Zone, in welcher die Fahrt begann	smallint
DOLocationID	Ein Code für die Zone, in welcher die Fahrt endete	smallint
trip_distance	Distanzangabe des Taximeters	real
fare_amount	Der Fahrpreis berechnet aus Zeit und Distanz	
extra	Verschiedene Zuschläge auf den Fahrpreis	real
MTA_tax	Aufschlag, automatisch erhoben bei entsprechender Rate	real
improvement_surcharge	Aufschlag, automatisch erhoben in bestimmten Zonen	real
payment_type	Angabe des Zahlungsmittels als Code	smallint
tip_amount	Höhe des Trinkgeldes	real
tolls_amount	Summierter Betrag von Zuschlägen dieser Fahrt	real
total_amount	Gesamtbetrag der Fahrt **ohne Trinkgeld**	real

Die Daten der Grünen Taxis sind erweitert um einen Code für den *Trip_Type* (Ob eine Fahrt von einem Taxistand begann oder ob die Gäste an der Straße abgeholt

wurden).

Alle Distanz-Angaben entsprechen amerikanischen Meilen (1 mile→1,6 km), alle Währungsangaben Dollar.

Für die Angaben der Codes sind ebenfalls Dictionaries bereitgestellt, diese spielen allerdings für den Machine-Learning-Aspekt dieser Arbeit keine Rolle.

Umfang

Aus Ressourcengründen wurde ausschließlich das Jahr 2017 betrachtet.

Es gibt **113 Millionen** Einträge für gelbe Fahrten, welche insgesamt knapp **8,1 GB Speicher** benötigen. Zusätzlich wurden Indizes angelegt mit weiteren 9,4 GB Speicher (Um schnelle Anfragen auf Uhrzeiten und Orte zu ermöglichen).

Es gibt **11,7 Millionen** Einträge für grüne Fahrten, mit insgesamt **900 MB Speicherplatz**. Es wurden zusätzlich Indizes mit 1,1 GB Speicher erstellt.

Zusammen gibt es aus dem Jahr 2017 also fast **125 Millionen Einträge**, welche insgesamt 19,5 GB Speicher belegen.

Anomalien

Innerhalb der Daten traten einige Ungewöhnlichkeiten auf - zum Beispiel gibt es Fahrten, die von Ort A nach Ort A verliefen und keine Strecke zurückgelegt haben. Bei einer genaueren Untersuchung ergab sich allerdings, dass diese Fahrten meist wenige Minuten dauerten und ebenfalls keinen Passagier hatten.

Es ist anzunehmen, das die Taxis an dieser Stelle auf ihre Passagiere gewartet haben. Zunächst wurde versucht, die Anomalien mit in die Machine-Learning-Algorithmen aufzunehmen. Erste Ergebnisse zeigten allerdings desaströse Resultate, welche v.a. daher rührten, dass extreme Reichweiten der Eingabedaten auftraten.

Es wurden außerdem weitere Anomalien gefunden, welche kurz genannt werden:

- Fahrten mit Negativkosten
- Fahrten außerhalb von 2017
- Fahrten mit extrem hohen Trinkgeld ($\sim$ 100$) oder extrem hohen kosten ($\sim$ 300$)
- Fahrten, die wenige Sekunden gedauert haben
- Fahrten, welche mehrere Stunden gedauert haben und dabei nur kurze Strecken zurücklegen

Die Daten wurden nach Maßgabe des Autors auf *gesunde* Werte gekürzt (keine Fahrten über 150$, nicht mehr als 40% Trinkgeld, nur positive Kosten, nur Fahrten in 2017).

4.2.2 Wetteraufzeichnungen

In diesem Unterabschnitt werden die Wetterdaten sowie ihr Umfang vorgestellt.

Die Wetterdaten stammen von der *National Oceanic and atmospheric Administration* [NOA] (Kurz: NOAA), welche verschiedene Klimadaten sammelt. Für dieses Fallbeispiel wurden die Wetterdaten der Wetterstation des JFK-Airports für das Jahr 2017 abgefragt.

Attribute und Datentypen

Im Gegensatz zu den Taxidaten werden in diesem Paragraphen lediglich die verwendeten Attribute vorgestellt. Es werden ebenfalls der Bezeichner, eine kurze Beschreibung und der Datentyp innerhalb des SQL Server vorgestellt.

Name	Beschreibung	Datentyp
Date	Der Tag, an welchem der Datensatz erhoben wurde	date
Hour	Die Stunde, an welcher der Datensatz erhoben wurde	smallint
DryBulbTemp	Die Trockenkugeltemperatur	real
WetBulbTemp	Die Feuchtkugeltemperatur (Messung unter Berücksichtigung von Verdunstungskälte)	real
DewPointTemp	Die Höhe des Taupunktes	real
RelativeHumidity	Die gemessene Luftfeuchtigkeit	real
Visbility	Sichtweite in Meilen	real
WindSpeed	Windgeschwindigkeit	real
WindDirection	Windrichtung in Grad	int
Sunrise	Uhrzeit des Sonnenaufgangs	datetime
Sunset	Uhrzeit des Sonnenuntergangs	datetime

Die Windgeschwindigkeiten sind hierbei in Meilen/Stunde angegeben, die Temperaturen in Grad Celcius auf eine Nachkommastelle gerundet. Die Windrichtung ist als Grad angegeben, wobei 360° Norden und 180° Süden entsprechen.

Es gab keine nennenswerten Anomalien.

Umfang

Es gibt **13351 Datensätze** die 1,4 MB Speicher benötigen. Zusätzlich gibt es 0,6 MB Indizes.

4.2.3 Machine-Learning-Sicht und Rich-Sicht

Die in den vorhergehenden Unterabschnitten vorgestellten Daten sind für die Verwendung als Sicht zusammengefasst, so das die Taxifahrten um die jeweiligen Wetterdaten angereichert werden.

Insgesamt wurden vier Sichten erstellt, je zwei für die grünen und gelben Taxis:

Rich-View Enthält alle Daten in lesbarer Form, Locations wurden nach *Borough* und *Zone* aufgeteilt. In gleichem Maße sind die HändlerID's, Zahlungsmittel und Raten als Text aufgelöst.

Diese Sicht wurde erstellt, um Anomalien zu erkennen und den Import der Daten zu überprüfen. Für Machine Learning ist Sie im Allgemeinen unbrauchbar.

Machine-Learning-View Enthält die für das Machine Learning benötigten Trainingsdaten. Jeder Satz der gefilterten Taxi-Daten erhält eine Erweiterung um die aktuell herrschenden Wetterdaten.

Die Orte, Raten und Zeiten liegen als numerischer Faktor vor.

Ausschnitt-Tabellen Neben den Views werden des weiteren kleinere Auschnitte entnommen. Dies liegt daran, dass das zufällige Entnehmen einer Stichprobe der gelben Taxidaten mehrere Minuten benötigt.

Die Ausschnitte wurden aus den ML-Sichten erstellt, indem jeder Datensatz einen zufälligen neuen Hashwert bekam, nach welchem er sortiert ist.

Für Prognosen, die für ihr Training aggregierte Daten benötigen (z.B. Umsatzprognose), wurde ebenfalls eine Aggregats-Tabelle erstellt.

Diese gruppiert die Machine-Learning-View nach Datum, Stunde, Startort,Zielort und aggregiert verschiedene Attribute. Dies führt zu einer Menge von ca. 1 Millionen Aggregat-Daten, hiervon wurden 10000 entfernt als Kontrollgruppe.

4.3 Prognosen

Dieser Abschnitt widmet sich der Umsetzung der verschiedenen Prognosen. Die Reihenfolge wurde nach aufbauender Komplexität der Fälle und vor Allem des *Tunings* gewählt - d.h. die Verbesserungen der zweiten Prognose *fußen* auf denen der ersten.

4.3.1 Trinkgeldprognose

Als erstes Beispiel wird die Vorhersage des gegebenen Trinkgeldes behandelt.

Dies stellt zwar nicht unbedingt ein für das Unternehmen hochgradig relevantes Thema dar, bietet jedoch einen guten Einstieg in das Thema Regression mit Hilfe von *Deep Learning*:

Es ist anzunehmen, dass Passagiere nach einem gewissen Schema Trinkgeld geben, etwa um Beträge aufzurunden oder eine schnelle Fahrt zu *belohnen*. Ebenso kann es Kriterien geben, welche nicht innerhalb der Daten erfasst sind, wie die Stimmung der Passagiere oder die Freundlichkeit des Fahrers. Genauso treten eventuell Kriterien auf, welche zu einem Ausbleiben des Trinkgeldes führen, eine lange Fahrt, eine Panne, oder ein Taxifahrer, der *Extrarunden dreht*.

Dennoch lassen sich diese potenziellen Kriterien schwer in einen anwendbaren Katalog zusammenfassen, nachdem man logisch das Trinkgeld erschließen kann. Vor diesem Hintergrund bietet sich die Verwendung eines tiefen neuronalen Netzes an:
Die erste versteckte Schicht extrahiert (uns unbekannte, nicht genauer definierte) Kriterien automatisch, und die zweite Schicht gewichtet diese.

Zielsetzung

Das Modell soll vorhersagbare Attribute erhalten, um für eine geplante Fahrt das Trinkgeld zu schätzen.

Ziel ist es eine gute Prognose mit realistischen, d.h. abschätzbaren Features zu erzeugen.

Aus unternehmerisches Sicht ist dieser Use-Case relevant, da Fahrer voraussichtlich die Fahrten mit höherem Trinkgeld bevorzugen werden, obwohl sich diese nicht mit dem größten Nutzen des Unternehmens decken müssen.

Versuchsaufbau

Die Fahrten liegen in gefilterter, aber nicht weiter aufbereiteter Form vor.

Als Eingabefeatures werden die Fahrtdauer in Minuten, der Start- und Ziel-Ort, die Kosten der Fahrt, die gefahrene Strecke und die Anzahl der Passagiere gewählt.

Die Ausgabe ist eine Schätzung des Trinkgeldes in Dollar.

Netzaufbau und Einstellungen

Das neuronale Netz wurde mit zwei Schichten á 100 Knoten definiert. Die Schichten sind vollvermascht und verwenden beide die Sigmoidfunktion. Als Ausgabefunktion wurde *Linear* gewählt.

Die Lernrate wurde auf 5% gesetzt und als Optimierungsfunktion stochastic gradient descent gewählt. Es werden jeweils 500 Epochen durchgeführt.

Tuning und Verbesserung

Wesentliche Unterschiede in der Genauigkeit dieses Beispiels verbucht die Auswahl der Features:

- Modelle, welche nicht die Gesamtkosten der Fahrt als Eingabe erhalten, erzielen nur Genauigkeiten von 75%.
- Ein Modell, welches sich auf ausschließlich die Kosten der Fahrt stützt, erzielte eine Genauigkeit von ca. 40%.
- Werden die im Versuchsaufbau genannten Features um (grobe) Wetterdaten und eine Tageszeit erweitert, verbessert sich die Genauigkeit des Modells auf bis zu 98%.
- Werden die im Versuchsaufbau genannten Features um die Passagieranzahl gekürzt wird nur eine Genauigkeit von ca. 80% erzielt - die Passagieranzahl scheint also ein wesentliches Feature darzustellen.

- Weitere Features, welche sich auf den Preis beziehen (z.B. die Rate der Fahrt, Steuerliche Zulagen, diverse Aufschläge), welche bereits im Gesamtpreis enthalten sind, bringen keine Veränderung der Genauigkeit.

Im Rahmen der Zielsetzung wurden aber lediglich die unter dem Versuchsaufbau gewählten Attribute verwendet. Die Vorhersage des Wetters gestaltet sich weitestgehend schwierig für ein Taxiunternehmen bzw. den Fahrer.

Weitere Unterschiede erzeugten Veränderungen in der Netzdefinition:

Eine Verkleinerung einer der beiden Schichten unter 50 Knoten führte zu drastisch schlechteren Ergebnissen (max. 60% Genauigkeit).

Eine Vergrößerung der Schichten über 100 Knoten führte zu keinem Gewinn an Genauigkeit, jedoch zu längeren Laufzeiten (Ein Trainingsdurchlauf mit je 200 Knoten dauert ca. 4 mal so lange).

Eine Verwendung der Aktivierungsfunktion *tanh* führte zu keinen Unterschieden, eine Verwendung der Softmax-Aktivierungsfunktion führte zu einer um ca. 20% längeren Laufzeit und ähnlichen Genauigkeiten.

Ergebnisse

Die folgenden Ergebnisse wurden mithilfe der unter **Netzaufbau und Einstellungen** genannten Optionen erzielt:

Trainingsdaten (in Tausend)	R^2-Wert	Zeit (in Min)
1	0.38	1
5	0.74	2
10	0.78	3
50	0.83	4
100	0.85	5
500	0.88	36
1000	0.90	71

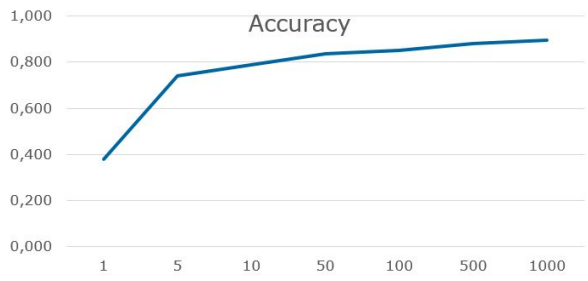

Abbildung 4.1: Ergebnisse der Trinkgeldprognose

Die Ergebnisse der Modelle sind sehr zufriedenstellend: Unter Verwendung absehbarer Daten können signifikante Modelle erstellt werden, ebenfalls verhält sich

der Trainingserfolg (sowohl unter Zuwachs der Epochen, als auch an der Menge der Trainingsdaten), wie man ihn erwartet.

Die Entwicklung des Modells ist allerdings mit mehreren Entscheidungen verbunden: Vor allem die Auswahl der Features hatte in diesem Beispiel große Auswirkungen auf die Genauigkeit. Während schwer abzuschätzende Parameter, wie das Wetter, exzellente Ergebnisse ermöglichen, können hiermit keine zuverlässigen Aussagen über Fahrten in der kommenden Woche getroffen werden. Sollte das Modell allerdings nur für den jeweiligen Tag (oder kürzere Intervalle) eingesetzt werden, und das Wetter relativ absehbar, ist eine Aufnahme dieser Features durchaus denkbar.

Ein wichtiges Nebenergebnis stellt die gleichbleibende Genauigkeit bei größeren hidden Layern dar. Durch die wachsende Trainingszeit sollte die optimale Größe des Netzes zunächst mit einer kleineren Menge an Trainingsdaten evaluiert werden.

4.3.2 Ratenerkennung

Als zweites Beispiel wird die Erkennung der Fahrrate behandelt. Dieser Fall ist weitestgehend trivial, wurde dennoch als einfache Referenz für Multiklassenklassifikation aufgenommen.

Im Gegensatz zur Trinkgeldprognose sind die Kriterien, nach welchen sich die Fahrtkosten berechnen, klar dokumentiert: Je nach Rate kostet jede Meile und jede Minute einen bestimmten Betrag. Zusätzlich kommt je nach Gebiet (z.B. JFK-Airport) ein Aufschlag hinzu. Dahingehend lässt sich diese Formel zur Erkennung der Rate umformen.

Interessant wird dieser Use-Case vor dem Hintergrund, dass die Raten bzw. ihre zugehörigen Verrechnungssätze nicht innerhalb der Daten auftreten, und das Modell erkennen muss, welche Beträge Aufschläge sind, und welche Beträge die Raten ausmachen.

Das Modell soll also in diesem Fall einen für den Menschen eindeutigen, klar nachvollziehbaren Sachverhalt reproduzieren.

Versuchsaufbau

Die Fahrten liegen in gefilterter, aber nicht weiter aufbereiteter Form vor.

Als Eingabefeatures werden die Fahrtdauer in Minuten, der Start- und Ziel-Ort, die Kosten der Fahrt, die gefahrene Strecke sowie die Anzahl der Passagiere gewählt.

Die Ausgabe ist die Rate der Taxifahrt als ID.

Netzdefinition

Das neuronale Netz wurde mit zwei Schichten á 100 Knoten definiert. Die Schichten sind vollvermascht und verwenden beide die Sigmoidfunktion. Als Ausgabefunktion wurde *Softmax* gewählt.

Die Lernrate wurde auf 5% gesetzt und als Optimierungsfunktion Stochastic Gradient Descent gewählt. Es werden jeweils 250 Epochen durchgeführt.

Tuning und Verbesserung

Innerhalb der Ratenerkennung ließen sich wenig Verbesserungen vornehmen: Grund hierfür ist die bereits anfänglich hohe Genauigkeit.

Eine Verbesserung der Geschwindigkeit lässt sich mit einem kleineren Netz erzielen. Eine Netzdefinition mit 60 und 10 Knoten in den Hidden Layern erzielte bei größeren Trainingsmengen ebenfalls eine Genauigkeit von 99%, was für die meisten Anwendungen ausreichend ist. Es benötigt allerdings nur ca. ein Fünftel der Zeit für das Training.

Ergebnisse

Die unter Netzdefinition genannten Optionen erzielten folgende Werte:

Trainingsdaten (in Tausend)	Accuracy	Zeit (in Min)
1	0.975	0.1
5	0.993	0.2
10	0.994	3
50	0.995	3
100	0.997	11
500	0.998	71

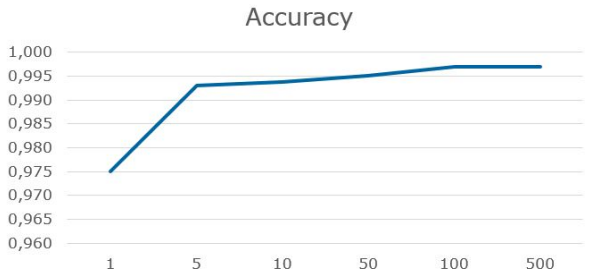

Abbildung 4.2: Ergebnisse der Ratenerkennung

Ein wichtiges Teilergebnis ist die Veränderung des ersten Versuches mit 1000 Trainingsdaten zu dem mit 5000 Trainingsdaten:
Zwar steigt die Genauigkeit nur minimal, dennoch werden erst ab 5000 Trainingsdaten die weniger vertretenen Daten korrekt erkannt. Das Modell für 1000 Trainingsdaten gab immer die Rate *Standard* aus - welche zu 97% auftritt.

Von den (zuletzt) nicht erkannten Daten besitzen 75% die Rate *Negotiated*.

Als Nebenergebnis ist zu nennen, das die Trainingsverfahren meist nach ca. 100 Epochen aufhören, weil keine weiteren Änderungen in der Genauigkeit erzielt werden.

4.3.3 Passagieranzahl

Ein weiteres Beispiel ist die Vorhersage der Anzahl der Passagiere, welche an einer Fahrt teilnehmen.

Dieser Use-Case ist eher für technische Belange interessant, denn die Verteilung der Passagieranzahl[2] über die Fahrten stellt eine Besonderheit dar:

Abbildung 4.3: Verteilung der Passagieranzahl über alle Fahrten

Wie zu erkennen ist, besitzen die meisten Fahrten einen einzelnen Passagier. Diese machen rund 70% der Fahrten aus.

Von den verbleibenden Fahrten machen Zwei-Passagier-Fahrten ca. 60% aus. Dies hat sich im Verlauf der Entwicklung als überaus herausfordernd herausgestellt.

Zielsetzung

Ziel des Modells ist es, für einen gegebenen Ort, Datum und Stunde einer Fahrt die Anzahl der Passagiere vorherzusagen.

Versuchsaufbau

Für diesen Use-Case wurden verschiedene Versuche durchgeführt. Gemeinsamkeit bilden die eingegebenen Daten: Datum, Tageszeit und Ort des Fahrtbeginns.

1. In *Variante 1* wurde das Modell mit Regression erstellt.

[2] Es handelt sich hierbei um die ungefilterten Daten

2. In *Variante 2* wurde das Modell für Multiklassen-Klassifizierung erstellt.

3. In *Variante 3* wurden Einzelfahrten gefiltert und ein Regressionsmodell erstellt.

4. In *Variante 4* wurden ebenfalls Einzelfahrten gefiltert und ein Multiklassen-Modell generiert.

Ausgabe war, abhängig von der Variante, die Anzahl der Fahrten als Ganzzahl (1 & 3) oder Klasse (2 & 4).

Es wurden verschiedene Netzdefinitionen durchgespielt (was Tiefe, Schichtgröße und Aktivierungsfunktionen betrifft), dennoch erzielten diese für alle 4 Varianten die selben Ergebnisse:

Ergebnisse

Variante 1 & 2 erreichten, bei größeren Trainingsmengen (ca. 10000), eine Genauigkeit von 70% [3]. Während dies objektiv ein akzeptabler Wert ist, gibt das Modell für jede Eingabe stets eine *Ein-Personen-Fahrt* aus. Diese Annahme stimmt in 70% aller Fälle.

Variante 3 & 4 zeigten dieselbe Schwäche für *Zwei-Personen-Fahrten*. Für kleinere Trainingsdaten wurden Modelle erzeugt, welche andere Ausgaben erzeugten, allerdings wurden mit zunehmenden Iterationen und Trainingsdaten wieder nur die statistisch häufigsten Werte ausgegeben.

Die dabei entstehenden Modelle zeichnen sich alle durch verschwindend geringe Gewichte aus. Die einzige Änderung geschieht innerhalb des Bias, welcher in beiden Modellen (nach Abschluss des Trainings) dem Ergebnis entspricht.

Da auch die Verwendung von der *Tanh*-Funktion und der Linear-Funktion in keinem befriedigendem Maße Ergebnisse zeigte (Welche sich für stark von Faktoren abhängige Schichten besser zu eignen scheinen, Siehe 4.3.4), ist anzunehmen, dass elementare Features fehlen. Ein oder mehrere ausschlaggebende Kriterien, um die

[3] Variante 1 nur bei einer Rundung auf Ganzzahl, ansonsten etwa 65%

Passagieranzahl zu ermitteln, wurde innerhalb der Trainingsdaten noch nicht abgebildet/berücksichtigt.

Ein Nebenergebnis des Versuches ist der direkte Vergleich der Geschwindigkeiten: Variante 1 und 3, sowie Variante 2 und 4 verhielten sich in ihrer Trainingsdauer jeweils analog.

Die Multiklassen-Varianten benötigten aber ca. die 3-fache Zeit ihrer Regressions-Gegenstücke.

Trainingsdaten (in Tausend)	Accuracy	Zeit (in Min)
1	0.31	1
5	0.58	3
10	0.69	6
50	0.71	11
100	0.71	31

Abbildung 4.4: Ergebnisse der Passagiervorhersage (Variante 2)

Die oben stehenden Ergebnisse in Abbildung 4.4 entsprechen den Zeiten von 250 Iterationen bei einem einzelnen *tanh*-Layer mit 250 Knoten und einer Multiklassen-Ausgabe.

4.3.4 Fahrtenaufkommen

Als nächstes Beispiel wird die Prognose des Fahrtenaufkommens behandelt.

Diese Prognose stellt die Erste für aggregierte Daten dar, und besitzt somit wesentlich weniger Trainingsdaten.

Eine weitere Schwierigkeit für das Modell besteht darin, das die Testdaten aus der Menge der Trainingsdaten entfernt wurden, und somit die Gruppierungen der Testdaten in dieser Form nicht vorliegen.

Das Modell muss also, um die Fahrten des 31. August um 12:00 vom JFK-Airport vorherzusagen,welche in dieser Kombination nicht im Training vorkommen, einen Transfer auf unbekannte Daten leisten. Das neuronale Netz kann sich in diesem Fall nicht *erinnern*.

Eine letzte Schwierigkeit für dieses Modell stellt die große Reichweite der Werte dar: Während es in den äußeren Gebieten wenige oder keine Fahrten gibt, werden Börsenviertel und Flughäfen stark frequentiert.

Hierbei gibt es sowohl in den Orten als auch in den kalendarischen Daten starke Ausschläge, welche innerhalb der gruppierten Daten zu einer breiten Streuung führen.

Zielsetzung

Ziel des Modells ist es, für einen gegebenen Ort und eine gegebene Stunde eines Tages die Anzahl der **ausgehenden** Fahrten vorherzusagen.

Dieses Modell hilft dem Unternehmen, seine Taxi-Kontingente strategisch zu verteilen bzw. kann dem Taxifahrer helfen, vor einer Rushhour am richtigen Taxistand einzutreffen.

Versuchsaufbau

Die Taxifahrten liegen in aggregierter Form vor: Sie wurden nach Datum, Tageszeit auf Stunde gerundet und Startort gruppiert. Die daraus resultierenden Trainingsdaten sind Wertepaare aus der Anzahl der Fahrten, Ort, Stunde und Datum.

Die Gruppierungskriterien werden in R als Faktoren repräsentiert.

Ausgabe des Modells ist eine Schätzung der Fahrten, welche auf eine Ganzzahl gerundet wird.

Netzdefinition

Für die endgültige Netzdefinition wurde ein einzelner Layer mit 300 Knoten und der *tanh*-Aktivierungsfunktion gewählt. Die Ausgabe erfolgt über die *Linear*-Funktion.

Die Lernrate wurde auf 15% gesetzt und als Optimierungsfunktion Stochastic Gradient Descent gewählt. Es wurde ein Verfall von 5% gewählt. Es werden jeweils 500 Epochen durchgeführt.

Tuning und Verbesserung

Die Prognose des Fahrtenaufkommen hat in den ersten Versuchen mit zwei Hidden Layern keine Ergebnisse erzielt (genau genommen negative r^2-Werte [4]).

Das erzeugte Modell gab in jedem Fall den Mittelwert der Fahrten aus, die Gewichte innerhalb des Modells hatten verschwindend geringe Werte und der Mittelwert wurde als Bias addiert.

Hierfür gibt es zwei mögliche Erklärungen: Die erste ist Overfitting. Die zweite ist eine mögliche, noch zu geringe Komplexität bzw. fehlende Features.

Um das Problem zu analysieren wurden dahingehend verschiedene Netzparameter durchgespielt, von [10,10] bis [1000,1000]-Netzen wurde eine große Bandbreite getestet. Auch die Veränderung der Aktivierungsfunktionen zu Sigmoid, Tanh, Softmax und Linear, bzw. Kombinationen dieser, erzeugte keine besseren Ergebnisse.

Die ersten Ergebnisse wurden erzielt, als die zweite versteckte Schicht entfernt wurde. Das neue Modell mit einer einzelnen, 100 Knoten großen Schicht zeigte das zu erwartende, trainierbare Verhalten. Hier zeigten größere Trainingsdaten sowie mehr Epochen eine positive Wirkung auf die erstmals positive Genauigkeit. Ebenso zeigten unterschiedliche Aktivierungsfunktionen verschiedene Ergebnisse

[4] negative r^2-Werte bedeuten, das eine Gerade bessere Werte erzielt, als das Modell

(Linear sehr schlechte, Softmax und Sigmoid gute, Tanh sehr gute).

Dieses Verhalten legt nahe, das es sich um eine Art von Overfitting gehandelt hat.

Die besseren Ergebnisse der Tanh-Aktivierungsfunktion gegenüber der Sigmoid- und Softmax-Funktion sind voraussichtlich auf das Verhalten um die Nullstelle zurückzuführen: Während die $sig(0) = 0.5$ ist, gilt $tanh(0) = 0$. Somit werden Null-Werte, wie sie in den faktorisierten Werten (z.B. Ort) häufig auftreten, besser abgebildet und werden unabhängig vom angelegten Gewicht nicht in die weiteren Berechnungen aufgenommen.

Unter Verwendung der Sigmoid-Funktion werden bei vielen Epochen und Trainingsdaten zwar ebenfalls akzeptable Ergebnisse erzielt, dennoch liegt das i.A. daran, dass die Gewichte der faktorisierten Daten gegen Null reduziert werden. Dies führt dazu, das faktorisierte Features nicht (stark) in die Prognose aufgenommen werden, und durch den *Wegfall* der Features ein schwächeres Modell erzeugt wird.

Eine Verbesserung der Genauigkeit hat sich ebenfalls durch das Erhöhen der Neuronenanzahl sowie der Anzahl der Epochen ergeben. Die in Netzdefinition genannten Einstellungen ergaben hierbei die kleinsten Werte mit der höchsten erzielten Genauigkeit.

Eine potenzielle Verbesserung würde eine komplexere Netzdefinition darstellen, welche zwei (oder drei) parallele Schichten besitzt: Eine für die numerischen Daten mit der Sigmoid- oder Softmax-Funktion sowie eine getrennte Schicht für die faktorisierten Daten. Dies wurde aufgrund der technischen und terminlichen Limits der Arbeit nicht umgesetzt (Zu hoher Speicherverbrauch sowie lange Trainingszeiten).
Was ebenfalls eine wesentliche Verbesserung darstellen kann, ist die Aufnahme eines zweiten Jahres in die Trainingsdaten.

Ergebnisse

Die unter Netzdefinition genannten Optionen erzielten folgende Werte:

Trainingsdaten (in Tausend)	R^2-Wert	Zeit (in Min)
1	0.1	0.1
5	-1.5	0.4
10	-0.3	1.5
50	0.37	3.5
100	0.5	6
500	0.7	34
1000	0.73	69

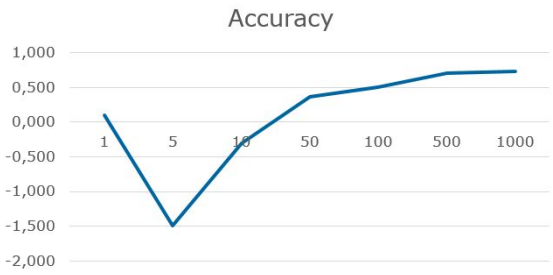

Abbildung 4.5: Ergebnisse der Fahrten-Prognose

Dies stellt zwar nicht die besten Ergebnisse dar, dennoch stellen die unter **Tuning** vorgestellten Maßnahmen die wichtigsten praktischen Erkenntnisse dieser Arbeit dar.

Bei eintausend Trainingsdaten gibt das Modell lediglich den Mittelwert aus - es liegen noch zu wenig Trainingsdaten vor. Bei Fünf- und Zehntausend liegt zunächst ein starker Einbruch vor, was darauf zurückzuführen ist, dass sich das Modell an den (wenigen) Trainingsdaten ausrichtet, die inhaltlich zu weit entfernt sind von den tatsächlichen Testdaten (Es könnten z.B. keine Daten für den Oktober, oder keine Daten für Nachtfahrten ins Training eingeflossen sein).

Werden die Ergebnisse nicht gerundet (auf *ganze Fahrten*) so liegen die r^2-Werte ca. 2% höher (in allen Versuchen).

4.3.5 Umsatzvorhersage

Als letztes Beispiel wird die Umsatzvorhersage behandelt. Dieses stellt für das Unternehmen offensichtlich den wichtigsten Use-Case dar.Sie gibt dem Unternehmen nicht nur eine Übersicht über den Ist-Zustand, sondern zeigt ebenfalls detailliert, an welchen Punkten noch gearbeitet werden muss.

Ein gut funktionierendes Modell kann ggfs. auch zur Prognose von Änderungen erweitert werden.

Zielsetzung

Ziel des Modells ist es, für einen gegebenen Ort und eine gegebene Stunde eines Tages die Summe der Fahrtenkosten auszugeben.

Versuchsaufbau

Der Gesamtumsatz wird aus den Einzelfahrten aufsummiert: Sie wurden nach dem Datum, der Tageszeit (auf Stunde gerundet) und dem Startort gruppiert. Die daraus resultierenden Trainingsdaten sind Wertepaare aus dem Gesamtumsatz, Ort, Stunde und Datum.

Die Gruppierungskriterien werden in R als Faktoren repräsentiert.

Ausgabe des Modells ist eine Schätzung des Umsatzes, welche auf zwei Nachkommastellen gerundet wird.

Tuning und Verbesserung

Für diesen Use-Case wurden ebenfalls zwei Versuche durchgeführt. Gemeinsamkeit bilden die eingegebenen Daten: Datum, Tageszeit und Ort des Fahrtbeginns.

Hierbei ist die Ausgabe von Variante 1 der aufsummierte Umsatz aller Fahrten innerhalb der Stunde, und bei Variante 2 der durchschnittliche Umsatz pro Fahrt in dieser Stunde.

Es wurden verschiedene Netzdefinitionen durchgespielt (was Tiefe, Schichtgröße und Aktivierungsfunktionen betrifft), dennoch erzielten (im Wesentlichen) alle vergleichbare Ergebnisse. Die unten aufgeführten Ergebnisse sind die zuletzt mit

einem einzelnen hidden Layer mit 250 Knoten und der *tanh*-Aktivierungsfunktion erzielten Genauigkeiten.

Ergebnisse

Trotz intensiver Versuche erreichte Variante 1 nie positive r^2 Werte.

Da allerdings verschiedene Netzeinstellungen, insbesondere die Erkenntnisse aus den anderen Use-Cases, ebenfalls eingeflossen sind, lässt sich Overfitting im Allgemeinen ausschließen.

Für Variante 2 wurden mithilfe der Erkenntnisse aus 4.3.4 bezüglich des Overfittings ein r^2-Wert von 31% erzielt.

Die Ergebnisse legen nahe, das es ähnlich wie bei den Passagieranzahlen noch fehlende, dringend benötigte Features gibt. Potenziell denkbar wären Feiertags-Kalender oder beispielhafte Angaben eines Tourismus-Büros über erhöhtes Touristenaufkommen.

Nachdem die Schätzung des Fahrtenaufkommens zufriedenstellend umgesetzt werden konnte, muss die Schwierigkeit explizit bei der Berechnung des Umsatzes liegen.

Die hierfür wahrscheinlichste Ungewissheit stellt die Unberechenbarkeit des Zielortes dar. Innerhalb der Daten gibt es keine Ausschläge der Verteilung *Fahrten pro Startort nach Zielort* - es gibt entsprechend ebenfalls keine (wahrscheinliche) Aussage darüber, wohin eine einzelne Fahrt geht. Da der Umsatz einer Fahrt, und entsprechend die Summe über alle Fahrten stark von den jeweiligen Distanzen und Zeiten abhängt, kann ein Modell keine genauen Ergebnisse erzielen.

Insgesamt konnte kein zufriedenstellendes neuronales Netz mithilfe der vorliegenden Daten erzeugt werden. Dennoch gibt es Alternativen um eine (zufriedenstellende) Prognose zu erreichen: Diese liegen vor Allem in der klassischen Statistik. Dennoch könnten auch modernere, technische Ansätze, wie eine Time-Series-Datenbank Aufschlüsse und signifikantere Auswertungen ergeben.

4.4 Best Practices bei neuronalen Netzen im SQL-Server

Im Rahmen des Praxisteils haben sich im Wesentlichen 3 Best Practices herauskristallisiert, welche nun genauer vorgestellt werden.

4.4.1 Auschnitts-Tabellen und Aggregatstabellen

Im Rahmen des Trainings mussten zufällige Daten ausgewählt werden. Dies wird innerhalb von SQL über die Sortierung einer zufälligen *ID* realisiert.

Dieses Verfahren dauert für die vorliegenden Daten (113 Millionen Datensätze) ca. 26 Minuten [5]. Während diese Zahl bei Modellen, die Stunden trainieren, eher unbedeutend ist, stellte sie gerade für die ersten *Testmodelle* einen Großteil der benötigten Zeit dar.

Eine wesentliche Verbesserung ergab sich durch die Erzeugung einer (kleineren) Tabelle, die bereits zufällig sortiert ist, und ein Training mithilfe dieser Tabelle.

Vor allem für kleinere Modelle, welche lediglich mit wenigen Tausend Daten trainiert wurden, führte dies zu einer markanten Beschleunigung.

Bei den Modellen, welche sich auf aggregierte Daten stützen, fiel dieser Effekt noch größer aus: Eine Aufsummierung des Umsatzes nach Ort und Stunde dauerte ca. eine Stunde. Hierfür wurde ebenfalls eine (redundante) Tabelle erstellt, welche bereits (mehrere) aggregierte Werte hält.

4.4.2 ML-Templates

Nach den ersten Schritten mit den neuronalen Netzen zeigte sich ein Muster, welches sich auf alle Use-Cases übertragen lies.

Dieses Muster lässt sich mithilfe von vier SQL-Dateien darstellen:

[5] Bei Verwendung eines einfachen Desktop-PCs

1. Eine Datei zur Erstellung einer Prozedur, welche das neuronale Netz erstellt, trainiert und abspeichert.

   Hier findet sich die Netzdefinition, Trainingsparameter und ggfs. eine Aufbereitung der Trainingsdaten.

2. Eine Datei zur Erstellung einer Prozedur, welche alle Testdaten mithilfe des neuronalen Netzes vorhersagt.

   Hier findet sich eine analoge Aufbereitung der Testdaten zu den Trainingsdaten.

3. Eine Datei zur Erstellung einer Prozedur, welche die Vorhersage bewertet. Je nach Art des Netzes wird die Genauigkeit berechnet und eine Stichprobe der Vorhersagen genommen.

   Diese Prozedur lässt sich soweit vereinfachen, das es jedes beliebige Modell bewertet.

4. Eine Datei, welches die Prozeduren ausführt.

Mithilfe dieser Templates ließen sich sehr schnell Use-Cases umsetzen und die Modelle sowie Prozeduren waren einheitlich und übersichtlich.

Für eine reelle Anwendung sollten noch zwei weitere Prozeduren aufgenommen werden: Eine zum *Weitertrainieren* eines Modells, sowie eine zur konkreten Vorhersage eines einzelnen Datensatzes.

4.4.3 Speicherung der Modelle

Für die Organisation, den Vergleich der Modelle und die allgemeine Verwendung wurden die serialisierten Modelle in einer eigenen Tabelle abgespeichert.

Während diese zunächst nur das Binärfile und den Namen hielten, war dies nicht wirklich hilfreich bei der Wiederverwendung. Deswegen wurde die Tabelle um folgende Eigenschaften erweitert, welche innerhalb der oben genannten Prozeduren mitgesetzt werden:

1. Das Erstellungsdatum

2. Die Art des Modells (Klassifizerung oder Regression)
3. Die Anzahl der Trainingsdaten
4. Die Genauigkeit
5. Einen Kommentar

Die ersten 3 werden hierbei von der Trainings-Prozedur erstellt, die Genauigkeit durch die Bewertungs-Prozedur.

Diese erweiterte Tabelle stellte sich als wesentlich benutzerfreundlicher heraus und lässt sich über das Template problemlos warten.

5 Fazit

Im Rahmen der Arbeit haben sich verschiedene, übergreifende Ergebnisse herauskristallisiert welche nun abschließend aufbereitet werden.

In Bezug auf die Benutzung der Algorithmen gilt: ***Easy to learn, hard to master***.

Moderne Frameworks und Bibliotheken ermöglichen es *kinderleicht* komplexe Modelle zu erstellen und zu benutzen. Sogar die Aufbereitung der Daten erfolgt einfach mithilfe optischer Werkzeuge oder wird sogar automatisch übernommen (z.B. die Normierung und Faktorisierung innerhalb von RevoScaler).

Dennoch liefern diese Modelle nicht unbedingt zufriedenstellende Ergebnisse:

Entweder sind die Standard-Einstellungen nicht geeignet, oder die Aufbereitung der Daten produziert neue Fehler (z.B. wenn die Orte als ID übergeben werden, sind diese nicht faktorisiert). Auch entstehen häufig Fehler bei automatisch faktorisierten Daten, etwa weil innerhalb der Trainingsmenge eien Merkmalsausprägung gefehlt hat, welche in den Testdaten vorkommt und nicht korrekt auf den Eingabevektor übertragen werden kann.

Fehler dieser Art zeigen sich erst bei der konkreten Anwendung des Modells auf separate Testdaten: Werden die *korrumpierten* Trainingsdaten und Einstellungen in Standardeinstellungen verwendet, so zeigt das Modell bei zunehmenden Training ebenfalls Erfolg - es spiegelt nur nicht zwangsläufig echte Erfolge wieder.

Insofern sind die Frameworks und Standardeinstellungen zwar angenehm für den Einstieg, sollten allerdings für die Umsetzung eines Projektes genau inspiziert werden. Vor allem die Aufbereitung der Trainingsdaten und die Kontrolle des Modells sollte der Entwickler selbst übernehmen.

Ein häufiger Satz, welcher v.A. in Internetforen auftritt, ist ***building neural networks is more art than science***.

Diesen Satz sollte man nicht unreflektiert hinnehmen - viele Änderungen, Designentscheidungen und Trainingserfolge wurden bewusst durch die wissenschaftlichen Eigenschaften der einzelnen Komponenten erzielt.

Konkret zeigte sich dies bei der Benutzung der *tanh*-Funktion gegenüber der Sigmoid-Funktion in Abschnitt 4.3.4 welche Aufgrund ihrer Eigenschaften mit negativen Werten bessere Ergebnisse erzielte.

Auch ist die Analyse des Datenbestandes, im Speziellen die Suche nach Anomalien, und das Design von Features im Bereich der Wissenschaft anzusiedeln.

Zwar sind viele Fortschritte, z.B. das beheben von Over- und Underfitting, experimentell entstanden, dennoch können grundlegende Probleme auftreten, welche sich nicht (zeitnah) experimentell lösen lassen, im Rahmen dieser Arbeit konkret Abschnitt 4.3.5 zur Schätzung des Umsatzes und 4.3.3 für das Passagieraufkommen. Hier haben selbst umfangreiche Experimente keine Ergebnisse erzielt.

Als letzter, großer Themenblock *pro science* ist ebenfalls das Performancetuning zu nennen:

Alle Hyperparameter der Optimierungsfunktion haben einen technischen Ursprung, und ein Verständnis der Algorithmen ist notwendig, um die Parameter sinnvoll zu wählen. Zwar sind die Standardeinstellungen hier meistens hinreichend, allerdings können durch bewusste Anpassung ebenfalls markante Besserungen in der Trainingsgeschwindigkeit erzielt werden.

So zeigten höhere Lern- und Verfalls-Parameter bei den weitestgehend homogenen Trinkgelddaten ein wesentlich schnelleres Terminieren das Trainings bei gleichbleibender Genauigkeit.

Für andere Experimente, z.B. dem Fahrtenaufkommen, zeigten diese Einstellungen katastrophale Ergebnisse - die Daten waren einfach zu verschieden, um mit diesen Parametern behandelt zu werden.

Große Teile sind allerdings rein experimentell - vor Allem die optimale Größe der

einzelnen Hidden Layer zu finden gestaltet sich im Wesentlichen durch Versuche.

Insgesamt kann zwar jedes Problem rein experimentell gelöst werden, allerdings stellt dies oft mehr einen *Glückstreffer* dar.

Ein Verständnis der Algorithmen, ihrer Eigenschaften und der verwendeten Funktionen ermöglicht nicht nur eine bessere Ausgangslage für das Tuning, sondern ist eine Grundvoraussetzung für die Fehleranalyse und Problembehebung.

Insofern ist der Ansatz, die Herangehensweise *a priori* als Kunst zu bezeichnen nicht gerechtfertigt. Die Arbeit mit neuronalen Netzen entpuppte sich als gesunde Mischung aus Kunst und Wissenschaft.

Ausblick

Innerhalb der Arbeit ließen sich zwei weiterführende Themen ausmachen:

Der Fokus auf Alternativen bei der Umsatzprognose sowie die Übertragung und Verallgemeinerung der erarbeiteten Netze. Diese beiden Ansätze werden kurz ausgearbeitet.

Die Umsatzprognose stellt nach wie vor den wichtigsten Use-Case für viele Unternehmen dar, und konnte ihm Rahmen dieser Arbeit mit neuronalen Netzen nicht gelöst werden. Im Zuge einer weiteren Arbeit kann der Hauptfokus auf der Umsatzprognose liegen, und verschiedene Formen der Vorhersage benutzt und verglichen werden um Ergebnisse zu erzielen.

Möglichkeiten stellen hierbei klassische statistische Methoden dar, oder eine Vertiefung anderer technischer Ansätze wie etwa Timeseries-Datenbanken oder Modellierung innerhalb einer Mehrdimensionalen-Datenbank.

Ein weiterer Punkt ist die Portierung der in dieser Arbeit erzeugten neuronalen Netze auf ähnliche Anwendungsgebiete, konkreter auf Taxidaten einer anderen Stadt.

Hierbei zu lösen sind Schwierigkeiten bezüglich unterschiedlicher Eingabevektoren, vergleichbare Tests zu erzeugen sowie die technische Umsetzung innerhalb des SQL-Servers.

Literaturverzeichnis

[Bri] BRITZ, Denny: *Building a neural Network from scratch*. http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/. – Erstellung eines NN mit Python Schritt für Schritt - benutzt als Quelle bei Selby NN

[Dar84] DARLEY, John M.: A Hypothesis-Confirming Bias in Labeling Effects. 44 (1984), S. 20–33

[Fou] FOUNDATION, The R.: *What is R?* https://www.r-project.org/about.html

[FR] FORTMANN-ROE, Scott: *Understanding the Bias-Variance Tradeoff*. http://scott.fortmann-roe.com/docs/BiasVariance.html

[Giu] GIUFFO, John: *NYC's New Green Taxis: What You Should Know*. https://www.forbes.com/sites/johngiuffo/2013/09/30/nycs-new-green-taxis-what-you-should-know/#5ca3d25732a2. – Erklärung was es mit den Grünen Taxis auf sich hat

[Gova] GOV, NYC: *TLC Trip Record Data*. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. – Quelle der Taxidaten, angegebenes Date = Letztes Errata

[Govb] GOV, NYC: *TLC Trip Record Data*. http://www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf. – Data-Dictionary der gelben Taxidaten, angegebenes Date = Letztes Errata

[Ham] HAMMAK, Daniel: *Why does mean normalization help in gradient descent?* https://www.quora.com/Why-does-mean-normalization-help-in-gradient-descent

[Ins] INSTITUTE, National C.: *NCI Dictionary of Cancer Terms*. https://www.cancer.gov/publications/dictionaries/cancer-terms/def/selection-bias

[Mica] MICROSOFT: *Microsoft SQL Server 2017*

[Micb] MICROSOFT: *ML-Server*. https://github.com/Microsoft/ML-Server. – Github Repository des ML Servers. Weiterführende Links. Date=Letzer Commit

[Micc] MICROSOFT: *Welcome to Machine Learning Server*. https://docs.microsoft.com/en-us/machine-learning-server/what-is-machine-learning-server

[NOA] NOAA: *Land-Based Station Data*. https://www.ncdc.noaa.gov/data-access/land-based-station-data. – Quelle der Wetterdaten, angegebenes Date = Letztes Update

[Ola] OLAH, Christopher: *Calculus on Computational Graphs: Backpropagation*. http://colah.github.io/posts/2015-08-Backprop/

[Pat] PATNIA, Abhishek: *Why is softmax activate function called ßoftmax"?* https://www.quora.com/Why-is-softmax-activate-function-called-softmax

[Pfe] PFEIFER, Stella: *B wie Bias*. https://blog.eoda.de/2018/05/08/b-wie-bias/#more-4991

[Rei] REIF, Roberto: *Importance of Feature Scaling in Data Modeling (Part 2)*. https://www.robertoreif.com/blog/2017/12/21/importance-of-feature-scaling-in-data-modeling-part-2

[Rud] RUDER, Sebastian: *Understanding the Bias-Variance Tradeoff*. http://ruder.io/optimizing-gradient-descent/

[Sel] SELBY, David: *Building a neural Network from scratch in R*. https://selbydavid.com/2018/01/09/neural-network/. – Schritt für Schritt Erklärung von NN's + Codebeispiele in R ohne Package

[Sha] SHARMA, Sagar: *Activation Functions: Neural Networks*. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[Ste] STEEN, Heidi: *RevoScaleR package*. https://docs.microsoft.com/en-us/machine-learning-server/r-reference/revoscaler/revoscaler. – Haupt-R Paket von MS

[Str] STROETMANN, Prof. Dr. K.: *Artificial-Intelligence*. https://github.com/karlstroetmann/Artificial-Intelligence/tree/master/Lecture-Notes

[WNV18] W. N. VENABLES, D. M. S.: *An Introduction to R*. 2018

[Woo16] WOODY, Buck: *Data Science with Microsoft SQL Server 2016*. 2016

[Zei] ZEILER, Matthew D.: *AdaDelta: an adaptive learningrate method*