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, TINF15/AI-BI
Ausbildungsfirma:	Atos Information Technology GmbH
Betreuer 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 9. August 2018

_____

Leonhard Applis

Abstract

Englisch Abstract to be done

title:	Machine-Learning and Prognosis
author:	Leonhard Applis
matriculation number:	2086307
class:	TINF15/AI-BI
supervisor DHBW:	???
supervisor Atos:	Jonas Mauer

Kurzfassung

Deutscher Abstract muss gemacht werden

Titel:	Machine Learning und Prognosen
Author:	Leonhard Applis
Matrikelnummer:	2086307
Kurs:	TINF15/AI-BI
Betreuer der Dualen Hochschule:	Prof. Dr. Rainer Colgen
Betreuer der Firma:	Jonas Mauer

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 auf 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 ins Besondere 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

Ziel dieser Arbeit ist 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 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 allgemeine Programme 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.

Der Abschnitt 2.5 behandelt die Verwendung der Programme aus 3.3 unter Bezugnahme auf das Fallbeispiel. Abschluss dieses Abschnittes bildet die Erzeugung eines Modells.

Abschnitt **??** benutzt das in Abschnitt 2.5 erzeugte Modell, um die in Abschnitt 4.1 vorgestellten Anforderungen zu bearbeiten. Hier befinden sich die eigentlichen Ergebnisse des Experiments sowie Programme die Prognosen durchführen.

Abschluss des Kapitels bildet Abschnitt **??** in dem Methoden zum Test eines Modells vorgestellt und durchgeführt werden.

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. Der Umfang hierbei entspricht dem Besuch der Vorlesung *Mathematik II*.

  Literaturempfehlung Lin.Alg

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

  Literaturempfehlung Stochastik/-Statistik

- **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 von R innerhalb des 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, das 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 gesondert 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 sind im Abschnitt 2.3.4 für Regressionen und im Abschnitt 2.4.2 für Klassifikation vorgestellt.

2.2 Bias

Wenn man mit Vorhersagemodellen 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 diese 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 ei-

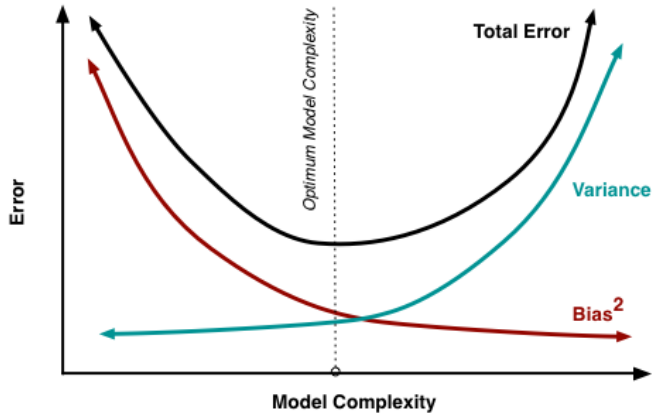

Abbildung 2.1: Bias-Variance-Dilemma

nes Modells wird der Fehler Verzerrung 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.

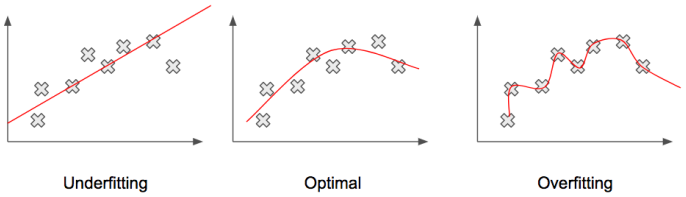

Abbildung 2.2: Über- und Unteranpassung

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*.

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 Trainings-Daten. 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 [1], wenn Anhand der Umfragen auf einer Messe für vegane Ernährung die Ernährungsgewohnheiten aller Deutscher interpretiert wird.

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

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 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 linearer Regression bereits sehr gute Ergebnisse erzielt werden.

[1] Es handelt sich hierbei um eine Vermutung

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 $\mathbf{x}$ und $\mathbf{y}$ das arithmetische Mittel der beiden Variablen darstellt, sowie s_x und s_y die Standart-Abweichungen. 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 unser Beispiel 2.3.1 haben wir ein 2-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 Erweiterung 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 Modell-Schätzung (immer) von den echten Werten abweicht. Innerhalb des F-Testes werden ebenfalls die Varianz der Grundgesamtheit in Relation zur Varianz der Modell-Werte betrachtet, um ein aussagekräftiges Ergebnis zu erzielen.

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 Ei-

genschaften 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 **Optimierungsfunktion**. 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 detailliertere 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.

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.

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

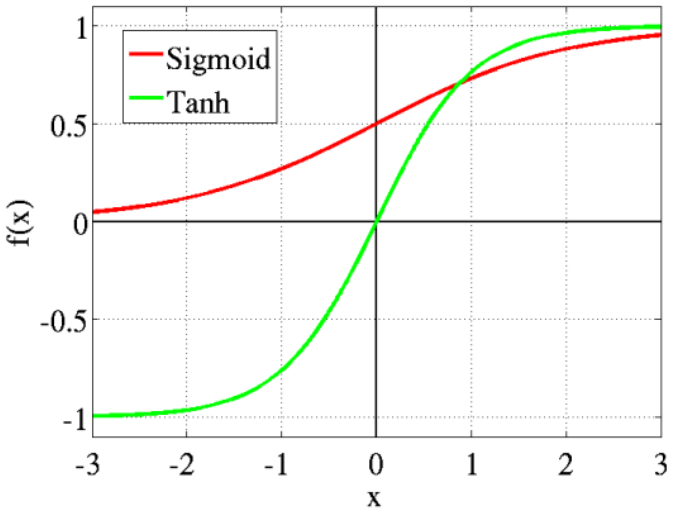

Abbildung 2.3: Sigmoid und Tanh

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 Ansätze, diesen Nachteil auszugleichen:

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

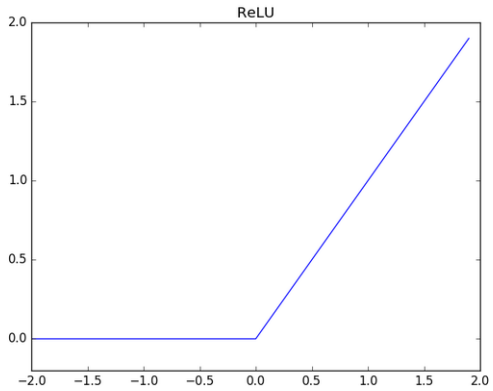

Abbildung 2.4: Plot der ReLU-Funktion

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 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[...]*).

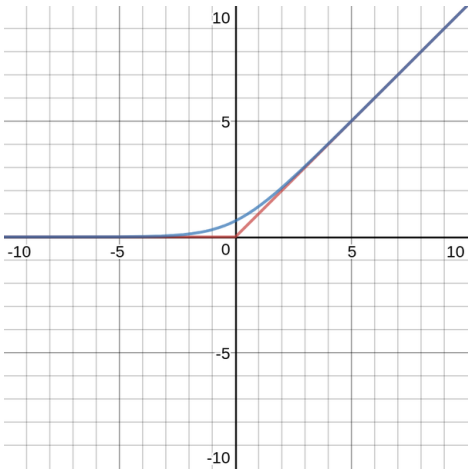

Abbildung 2.5: ReLU(Rot) und Softmax(Blau)

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.

Übliche Aktivierungsfunktionen sind das *Gradientenverfahren* sowie das *Stochastische Gradientenverfahren*.

2.4.5 Bewertung der Klassifizierung

Die Klassifizierung ist simpel dahingehend zu Bewerten, wieviele der Testdaten korrekt Klassifiziert wurden.

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

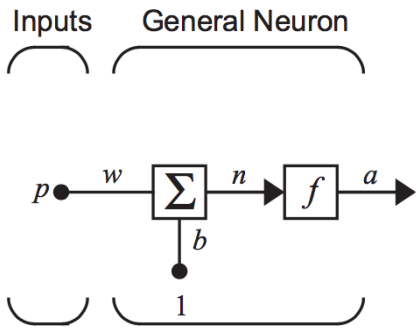

Abbildung 2.6: Einzelnes Neuron

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 2.6.
Diese Neuronen werden zu einen Graphen, welche sich als *Schichten* anordnen. Je-

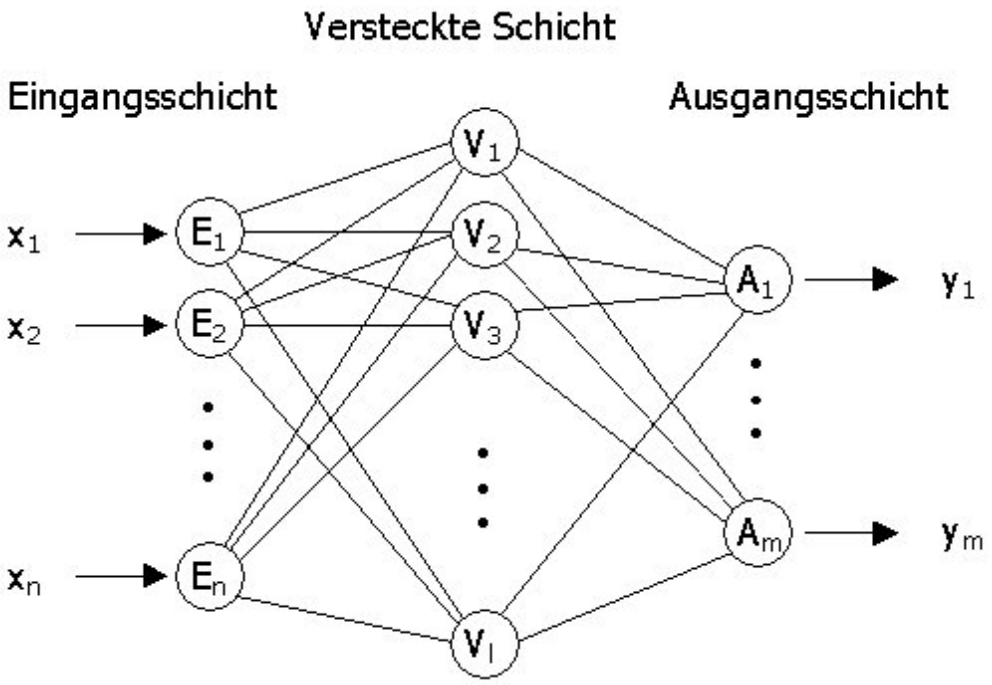

Abbildung 2.7: Modell eines Neuronalen Netzwerkes

des 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 2.7.

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, solange mit den vorliegenden Trainingsdaten zunächst Forward-Propagation durchzuführen, und 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 können die das Grundlegende Verfahren (Gradient Descent, SGC, Adadelta) gewählt werden, sowie Trainingsparameter (Lerngeschwindigkeit, Beschleunigung, Verfall) gesetzt 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 unter **??**.

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 grundlegend der SQL-Server 2017 vorgestellt, und anschließend werden vor dem Hintergrund Neuronaler Netze 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, seid 2017 sind sie nativ integriert.

Neben der reinen Datenhaltung bietet der SQL-Server folgende Schlüsselelemente:

1. (Web-) Schnittstellen um Remote auf Daten zu zugreifen
2. Möglichkeiten zur prozeduralen Programmierung
3. Unterstützende Tools für ETL-Prozesse und Reporting
4. Schnittstellen zu anderen Programmiersprachen, wie JS, .Net und Python
5. Security-Dienste zur Verschlüsselung der Daten sowie Nutzer und Rechtverwaltung

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.

3.1.1 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-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 barrierefreie Lösungen auf bestehenden Systemen erarbeitet werden.

Dennoch entstehen auch Probleme bei der Verwendung der Integrierten R-Services, konkret im Bereich der Entwicklung:

Bei Verwendung der R-GUI bzw. 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 als Skript in T-SQL liegt das Problem darin, dass das R-Skript als Text übergeben wird, und somit nicht in der Vorschau *compiled* 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 [1]. Optional bietet sich eine Auswahl nach bestehendem Vorwissen der Sprachen SQL und R an.

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

Broken Cite???

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

[1] 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.

Im Allgemeinen bezieht sich der Machine-Learning Server auf Projekte, die mit Python realisiert werden, sowie den ML-Bibliotheken von Microsoft.

3.2 Programmiersprache R

R ist eine Sprache und eine Entwicklungsumgebung für statistische Berechnungen und Grafiken. R ist ein GNU-Projekt und beruht auf der Sprache *S*, welche von John Chambers et Alii 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 Modellierung, Classification 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 Die Umgebung von 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.

Programmatische Besonderheiten

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. Ein Vektor kann dadurch *abgetrennt* werden.

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. Der Dimensionsvektor gibt hierbei eine Form für den Wert-Vektor dar, 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 Flale 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 *DataFrame* stellt eine besondere Form einer Liste dar, die folgende Eigenschaften erfüllen muss (Siehe [WNV18] Abschnitt 6.3 *DataFrames* Absatz 1f):

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

Data-Frames sind dahingehend wichtig, da der Import einer Tabelle aus dem SQL-Server als DataFrame erfolgt.

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.

Möglichkeiten in R Die Sprache R besitzt verschiedene Optionen 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 innerhalb des SQL-Servers bereits behandelt, sowie die gemeinsame Produktfamilie die einen einheitlichen Technik-Stack ergibt.

Möglichkeiten in Python Der SQL-Server 2017 unterstützt neben einem R-Server ebenfalls eine Instanz des Microsoft ML-Servers. Dieses Open Source Projekt zu finden auf Github [Mica] stellt eine Alternative zu den in R vorgestellten Methoden dar.

Der ML-Server ist in Python implementiert.

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 wiedergegeben 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 normiert. Diese Zeile ist nicht notwendig.

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

3.3.1 Lineare Regression

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

1. Alle Eingabewerte des Modells müssen (reelle)[2] Zahlen sein

[2]Es gibt andere Pakete, die komplexe Zahlen unterstützen

2. Texteingabewerte müssen zuvor über einen Faktor realisiert werden. Dieser Faktor muss eine festgesetzte Anzahl an Leveln besitzen.

3. 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 zu benutzen 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 Klassifikation mit RevoscaleR gelten folgende Bedingungen:

1. Alle Eingabewerte des Modells sind reelle Zahlen.
2. Texteingabewerte müssen zuvor über einen Faktor realisiert werden. Dieser Faktor muss eine festgesetzte Anzahl an Leveln besitzen.
3. Die Klasse stellt einen Faktor mit Level 2 dar.
4. Der Ausgabewerte ist eine Wahrscheinlichkeit, mit der die Ausprägung positiv ausfällt
5. 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.

Es werden an ein neuronales Netzwerk innerhalb von net# folgende Anforderungen 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 führen.
- 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. ReLU, 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:

```
netDefinition <- ("
  input Data auto;
  hidden Hidden[25] sigmoid from Data all;
  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 bis 4.7 die Ausarbeitung und Analyse verschiedener Prognosen mithilfe Neuronaler Netze.

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 Manhatten 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 dem Taxi an der Freiheitsstatue sagt *kurz 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.

  Eine Bewertung dieser Genauigkeit findet lediglich im Fazit statt.
- **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 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 wurden 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 gingen 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 das 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 am Ende jede Taxifahrt erweitert wird um die Wetterdaten.

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 v.A. von den 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 wurde.

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, wovon eine Probe von 10 000 entfernt wurde für die Überprüfung der Resultate.

4.3 Trinkgeldprognose

Zielsetzung

Versuchsaufbau

Netzaufbau und Einstellungen

Tuning/Verbesserung

Ergebnisse

4.4 Ratenerkennung

wie in Trinkgeld

4.5 Fahrtenaufkommen

wie in Trinkgeld

4.6 Umsatzvorhersage

wie in Trinkgeld

4.7 Passagieranzahl

wie in Trinkgeld

5 Fazit

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

[Micb] 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

[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