Karlsruher Institut für Technologie

Eine Schnittstelle zur Vorhersage von Nutzeranfragen auf Datensätzen

Bachelorarbeit
von

Robin M. Maisch

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuer:	Dipl. Inform. Alexander Wachtel

Bearbeitungszeit: 01. Dezember 2018 – 31. März 2019

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 31. März 2019

. . . . . .

(**Robin M. Maisch**)

Inhaltsverzeichnis

Abstract

Diese Arbeit stellt eine Schnittstelle vor, die, eingebunden in ein Programm, die Aktionen eines Nutzers im Hintergrund auf dem lokalen Rechner verarbeitet und speichert, und versucht, aus den gesammelten Daten eine Vorhersage für die nächste Aktion zu ermitteln. Eine Nutzeraktion ist dabei durch die Daten eines Datensatzes definiert, die der Nutzer durch die Aktion abfragen möchte. Aus einer Reihe von Paaren (Nutzereingabe, extrahierte Daten), die automatisch über eine REST-Schnittstelle oder von Hand über eine grafische Nutzeroberfläche in das System eingespeist werden können, wird ein Modell gewonnen, das für jede Aktion alle unmittelbar nachfolgenden Aktionen als Bigramm speichert und ihre Häufigkeit zählt. Die Schnittstelle stellt eine Reihe von Vorhersagestrategien zur Verfügung. Eine davon nutzt ein künstliches neuronales Netz, das das System in die Lage versetzt, auch dann einen einen Aktionsvorschlag zu machen, wenn der Nutzer zuvor eine völlig unbekannte Aktion ausgeführt hat. Die Leistungsfähigkeit des neuronalen Netzes wurde an Beispieldaten getestet und evaluiert. Für die Evaluation wurden zufällig Modelle mit Aktionsfolgen generiert, die menschliches Verhalten nachahmen sollten. Bei einem stichprobenartigen Durchlauf, bei dem das Modell auf zweitausend generierte Nutzeranfragen trainiert wurde, konnte das System die Aktionen zu 54.2 Prozent replizieren, bei zweihundert Nutzeranfragen im Mittel zu 72.2 Prozent. Bei authentischem menschlichem Nutzerverhalten gibt es gute Gründe dafür, anzunehmen, dass die Vorhersage noch leistungsfähiger ist.

Kapitel 1

Einleitung

1.1 Motivation

Intelligente Systeme unterstützen uns heutzutage in fast jedem Aspekt unseres täglichen Lebens. Informationen, die andernfalls nur langsam zu beschaffen wären, sind nur einen Mausklick, ein Fingertippen oder eine Frage an einen persönlichen Sprachassistenten wie „Alexa“ entfernt. Wer womöglich anfangs noch nicht zugeben mag, dass diese neuen Formen der Informationsbeschaffung den Alltag erheblich vereinfacht, gewöhnt sich, sofern er sich darauf einlässt, schneller daran, als er glauben mag.
Für vielerlei Anwendungsfälle über bloße Informationsbeschaffung hinaus stehen dedizierte Programme zur Verfügung, die den Nutzer ohne entsprechende umfassende Ausbildung anfangs vor enorme Hürden stellen können. Meist gelingt es nach einiger Zeit, zu lernen, wo die (meist wenigen) Befehle zu finden können, die benötigt werden, und so wird solch ein Programm immer wieder auf sehr ähnliche Weise verwendet. Da der Ablauf aber nun doch nicht gleichförmig genug ist, um etwa ein *Makro* zu verwenden, hat der Nutzer keine Wahl, als sich stets aufs Neue durch die Menüführung zu arbeiten, was ihn ermüdet und viel Zeit kostet.
Für einen speziellen Anwendungsfall dieser Problemstellung schlägt diese Arbeit einen Lösungsansatz vor. Integriert in ein Klientsystem, ermöglicht es das vorliegende System, den Nutzer bei sich wiederholenden Arbeitsschritten zu unterstützen, indem es einen Vorschlag für die nächste Aktion aus der Aktionshistorie des Nutzers generiert und diesen an das Klientprogramm übermittelt. So kann die nächste Aktion in der grafischen Nutzeroberfläche des Klientprogramms dem Nutzer als Abkürzung oder Vorschlag angezeigt werden und seinen Arbeitsablauf somit erheblich beschleunigen und erleichtern.

1.2 Ziele dieser Arbeit

Ziel dieser Arbeit ist, ein eigenständiges System in Python zu entwickeln, das aus der Aktionshistorie eines lokalen Nutzers eine Vorhersage für die nächste Aktion generiert. Bei der Entwicklung sollten verschiedene Vorhersagestrategien implementiert werden, um deren Eignung zu vergleichen. Um vom bekannten Nutzungsverhalten auf den möglichen Nachfolger einer neuen Aktion schließen zu können, sollte das System auch Lernalgorithmen aus der Python-Bibliothek TensorFlow nutzen.
Es wurde festgelegt, sich darauf zu beschränken, für die Vorhersagen je nur die letzte Aktion der Sitzung zu berücksichtigen. Die Evaluation sollte zeigen, ob eine Bigrammanalyse auf Nutzeraktionen ein zufriedenstellendes Ergebnis liefert oder nicht.
Nach der Einrichtung der benötigten Python-Bibliotheken sollte das System nicht mehr auf Anbindung an das Internet angewiesen sein; alle Informationen über Datensätze sollten vom Klientsystem bereitgestellt und ggf. aktualisiert werden. Alle Daten, die das System speichert, werden ebenfalls lokal verwaltet und verarbeitet. Es fließen demnach auch keine Daten über das Nutzungsverhalten in zentrale Datenspeicher ein, um Erkenntnisse daraus zu gewinnen.

1.3 Abgrenzung der Ziele

Folgende Ziele wurden mit diesem Projekt *nicht* verfolgt:

Interaktion mit der Datenbank

Die Datenbank von Eurostat bietet, wie das vorliegende System, Interaktionen durch REST-Anfragen an, eine Python-API ist aktuell in Vorbereitung[1]. So wäre es möglich gewesen, die Datensätze regelmäßig zu aktualisieren. Da jedoch das Klientsystem ebenfalls von diesen Datensätzen abhängt, soll es auch dafür zuständig sein, die Datensätze zu beschaffen und aktuell zu halten, falls gewünscht. Auf die Anbindung an die Eurostat-API wurde also verzichtet.

Linguistische Analyse

Im Laufe der Entwicklung wurde in Betracht gezogen, die vorliegenden Nutzereingaben nach Stichwörtern zu durchsuchen. Wenn Wörter Dimensionswerten entsprechen, wäre es möglich gewesen, diese durch andere Dimensionswerte derselben Dimension auszutauschen, um weitere Nutzereingaben korrekt erkennen zu können. Aus der Eingabe *„Was war 2018 die Durchschnittsnote von Höherer Mathematik I?“* könnte also zum Beispiel die Eingabe *„Was war 2018 die Durchschnittsnote von Linearer Algebra I?“* einfach abgeleitet werden. Die Dimensionswerte liegen in den Datensätzen aber meist kodiert vor (HM1, LA1) oder entsprechen normierten Kodes (*ICCS01*: Handlungen mit Todesfolge oder Tötungsabsicht), das System müsste also Synonyme auflösen können, so wie es in [Sch17] ermöglicht wird, oder sich auf weitere Quellen wie Wortnetze oder die offiziellen Metadaten des Datensatzes berufen, um jegliche veränderte Formen der der Dimensionswerte, also etwa in anderen Sprachen oder konjugierte bzw. deklinierte Formen, zu erkennen. Diese Strategie wurde deshalb schließlich wegen ihrer hohen Komplexität verworfen.

[1] pyrostat auf GitHub: https://github.com/eurostat/pyrostat

Kapitel 2

Grundlagen

2.1 Eurostat

Das *Statistische Amt der Europäischen Union* (*Eurostat*) sammelt statistische Daten der Mitgliedsstaaten der Europäischen Union, die „Vergleiche zwischen Ländern und Regionen ermöglichen“ [ES][1] sollen. Die Aufzeichnungen umfassen mehr als 4600 Datensätze mit Daten aus verschiedensten Themenbereichen, die zum Teil bis zur Gründung von Eurostat 1953 und weiter bis in die 1920er Jahre reichen. Die Daten werden meist für jeden Mitgliedsstaat angegeben; wenige Regionalstatistiken enthalten Daten für kleinere Verwaltungseinheiten nach dem *NUTS*-Schlüssel, der etwa für Karlsruhe-Stadt den Kode DE122 vorsieht.

2.1.1 TSV-Datensätze

Eines der Formate, in denen (n-dimensionale) Datensätze von Eurostat zur Verfügung gestellt werden, ist TSV (*Tab-separated values*). Abbildung 2.1 zeigt ein Beispiel. TSV-Dateien sind folgendermaßen aufgebaut: Die erste Zelle der Datei enthält die Namen der Dimensionen der Tabelle. Die Werte der letzten Dimension dieser Liste sind auf der restlichen ersten Zeile abgetragen, es handelt sich dabei in der Regel um die Angabe der Jahreszahl (*time*). Für die anderen Dimensionen werden alle Kombinationen von Werten, für die Daten verfügbar sind, in der ersten Spalte angegeben. Für jede Zelle gelten die Dimensionswerte für die jeweilige Zeile und Spalte. Meist sind nicht alle Wertkombinationen verfügbar, leere Zeilen und Spalten werden verworfen. Fehlende Werte werden durch einen Doppelpunkt ”:” gekennzeichnet. Oft sind Kleinbuchstaben am Ende einer Zelle angegeben, die Auskunft über Hintergründe zu diesem Wert geben, diese spielen für diese Arbeit jedoch keine Rolle.

Anmerkung: „Tabelle“

Unter einer *Tabelle* wird bei Eurostat eine Auswahl an Daten verstanden, die die „beliebtesten Eurostat Daten“[ES][2] enthält. Im Gegensatz dazu steht eine vollständige *Datenbank*. In

[1] Über Eurostat > Übersicht, Pfad: */about/overview*, abgerufen am 09. März 2019.
[2] Datenbank > Informationen, Pfad: */data/database/information*, abgerufen am 14. März 2019.

unit,age,sex,geo\time	2017	2016	2015	2014	2013	2012
NR,TOTAL,F,AD	:	:	:	:	37408	38252
NR,TOTAL,F,AL	1423050	1417141	1424597	1430827	1437193	1444234
NR,TOTAL,F,AM	1567380	1569535	1571450	:	:	1684000
NR,TOTAL,F,AT	4460424	4427918	4384529	4352447	4328238	4309977
NR,TOTAL,F,AZ	4918771	4870002	4817181	4763571	4707690	4651601
NR,TOTAL,F,BA	:	:	:	:	:	1963655

Abbildung 2.1: Ausschnitt aus dem Datensatz *Einwohnerzahl am ersten Januar (demo_pjan)*

dieser Arbeit soll *Dankenbank* jedoch für eine Institution (wie Eurostat) stehen, *Datensatz* oder, um Verwechslung zu vermeiden, synonym *Tabelle* für eine vollständige Menge Daten zu einem Thema. Des Weiteren werden nach gewissen Regeln vorgenommene Auswahlen von Daten aus einer Tabelle als *Datenmenge* bezeichnet.

2.2 Maschinenlernen

An dieser Stelle soll ein knapper, oberflächlicher Einstieg in die Konzepte des Maschinenlernens gegeben werden, der auf [Wai] basiert und mit Schreibweisen und Formulierungen aus den weiteren Quellen ergänzt wurde, die in diesem Abschnitt genannt werden. Insbesondere werden einige Formelzeichen und Begriffe hier eingeführt, die später verwendet werden sollen.

2.2.1 Grundlegendes

Ziel ist es, eine Funktion h zu konstruieren, die bestmöglich eine Zielfunktion f nachbildet, deren allgemeine Form nicht vorliegt, von der wir jedoch Funktionswerte für bestimmte Argumente $(x_1, x_2, ...) =: X$ kennen.

- Diese Funktionswerte werden oft in der Natur *beobachtet*, die jeweiligen Argumente (*Merkmale*) *gemessen*, und in einer Liste festgehalten. Da f meist sehr umgangssprachlicher und nicht mathematischer Natur ist, werden Einträge dieser Liste gern (x, y) statt $(x, f(x))$ notiert.
- Da h in der Regel nur ein (begrenzt erfolgreicher) Versuch ist, f nachzubilden, heißt h *Hypothese* (Annahme).

Die allgemeine Form von h muss dem Problem entsprechend festgelegt werden; offen bleibt eine Menge Parameter $\vartheta := (\vartheta_1, \vartheta_2, ...)$ (*Konfiguration*, *Gewichtungsvektor*), die durch Optimierungsalgorithmen in vielen Iterationen immer besser gewählt werden sollen. Eine Wahl von ϑ bestimmt eine Hypothesenfunktion, deshalb schreibt man häufig h_ϑ.

Grundlage der Optimierung ist die Verlustfunktion $loss_\vartheta$[3], die angibt, wie stark h_ϑ von f abweicht, also $loss_\vartheta = \frac{1}{|X|} \sum_{x \in X} g(h_\vartheta(x) - f(x))$ für eine Funktion g. Im Kontext der konkreten Implementierung wird $loss$ aber auch gerne „lokal interpretiert“, nämlich als Funktion über ein Paar (Argument, Zielwert): $loss(\vartheta; x, y) = g(h_\vartheta(x) - y)$.

Für die ideale Konfiguration $\hat{\vartheta}$ wäre $loss(\hat{\vartheta}; x, y) = 0$ für alle bekannten Paare (x, y). Um von einem anderen, imperfekten ϑ aus zu einer besseren Konfiguration ϑ' zu gelangen (also $loss_{\vartheta'} < loss_\vartheta$), betrachte den Gradienten $grad(loss_\vartheta)$. Dieser gleicht einem Vektor entgegengesetzt der Richtung des steilsten Abstiegs (nach [WWS04, 21.35. (2)]) an der

[3] auch: L; im Kontext der Wirtschaft und der Statistik gelegentlich *Kostenfunktion* J_ϑ

Stelle ϑ. Wähle also $\Delta\vartheta := \alpha\frac{-grad(loss_\vartheta)}{\|grad(loss_\vartheta)\|}$ mit einer *Schrittweite* α und erhalte das bessere $\vartheta' := \vartheta + \Delta\vartheta$ (*Gradientenabstiegsverfahren*).

- Das Verfahren endet in einem lokalen Minimum von $loss_\vartheta$, wo $grad(loss_\vartheta) = 0$. Das ist im Allgemeinen nicht die beste Lösung, daher wird der Algorithmus mehrmals ausgeführt und die Ursprungskonfiguration ϑ_0 zufällig initialisiert.
- Die Güte der resultierenden Hypothese wird häufig an einer Teilmenge der bekannten Funktionsstellen X getestet, den *Testdaten* X_{Test}. X_{Test} wird bei der Optimierung nicht mit einbezogen und soll sicherstellen, dass die Verlässlichkeit auf den *Trainingsdaten* $X_{Training} = X \setminus X_{Test}$ erhalten bleibt.

2.2.2 Interpretation als Merkmalsvektor und Gewichtungsvektor

Als Vorüberlegung zu den Vorhersagestrategien, die Matrizenmultiplikation verwenden, betrachten wir Vektoren $x, w \in \mathbb{R}^n$. Die Einträge von x werden als Ausprägung eines Merkmals interpretiert, $y = xw^\top \in \mathbb{R}$ als eine Aussage über x. Der i-te Eintrag des *Gewichtungsvektors* w, w_i, bestimmt dabei, ob das i-te Merkmal x_i positiven, negativen oder keinen Einfluss auf y hat.
Eine Gewichtungsmatrix $w \in \mathbb{R}^{n\times k}$ kann auf dieselbe Weise k Aussagen auf einmal hervorbringen. Bei einer Merkmalsmatrix $w \in \mathbb{R}^{m\times n}$ werden entsprechend m Merkmalsvektoren auf dieselbe Weise verarbeitet.

2.2.3 Neuronale Netze

Ein neuronales Netz erweitert die Strategie, mehrere Merkmale aus dem Eingabevektor zu generieren, insoweit, als (1.) auf das Ergebnis ein konstanter Verzerrungsvektor (*bias*) addiert wird, (2.) auf das neue Ergebnis eine *Aktivierungsfunktion* angewendet wird, und (3.) dieses Verfahren mehrere Iterationen mit eigenen Gewichtsmatrizen, Verzerrungsvektoren und Aktivierungsfunktionen durchläuft.

Aktivierungsfunktionen

Die Aktivierungsfunktionen bilden das ursprüngliche Ergebnis auf eine Zielmenge ab, die in den weiteren Schritten besser verwendet werden kann. Einige häufig verwendeten Aktivierungsfunktionen sind im Folgenden angegeben.

$$\varphi_\vartheta(x) = \begin{cases} 1, & x \geq \vartheta \\ 0, & x < \vartheta \end{cases} \qquad \varphi_\vartheta(x) = \frac{1}{1+e^{-x}} \qquad \varphi_\vartheta(x) = \max(0, x) \qquad \varphi_\vartheta(x) = x$$

(a) *Threshold*-Funktion (b) *Sigmoid*-Funktion (c) *Rectifier*-Funktion (d) Identitätsfunktion

Abbildung 2.2: Aktivierungsfunktionen für neuronale Netze

2.2.4 Klassifikation

Die konkreten Problemstellungen, die durch Maschinenlernen gelöst werden sollen, sind häufig die der *Klassifikation*, also der Zuordnung eines Merkmalsvektors x zu einer Klasse c aus einer Menge Klassen C.

Beispiele.

(1) Ist dieser Bankkunde ein vertrauenswürdiger Kreditnehmer? (Gegeben möglicherweise: Einkommen, Alter)[4]

[4] Lernprobleme der Art *Ja/Nein* wie in (1) werden auch als *Konzept* bezeichnet.

(2) Wessen Gesicht ist das? (Zweidimensionales Array von Pixeln)

(3) Um welche Sorte Iris handelt es sich? (Länge und Breite der Kelch- und Blütenblätter, [Fis36])

Ergebnis von Klassifikationsproblemen ist oft ein Vektor der Länge $|C|$, der für jede mögliche Klasse c angibt, wie wahrscheinlich es ist, dass x c angehört. Das ist im Fall der neuronalen Netzwerke häufig genau das Anwendungsgebiet der *Sigmoid*-Funktion 2.2b.

2.2.5 Regression

Regression (dt. etwa: Rückführung) ist die Ermittlung einer mathematischen Funktion, die die gegebenen Paare $(x, f(x))$ bestmöglich annähert. Da die beobachteten Funktionswerte $f(x)$ i.d.R. störungsbehaftet sind, werden sie auch als punktweise von der „wahren" Zielfunktion abweichend interpretiert: $(x_i, f(x_i) + \varepsilon_i), |\varepsilon_i| < \varepsilon$ für „kleines" ε. Die allgemeine Form der Zielfunktion bestimmt das Vorgehen und oft die Länge des Parametervektors; entsprechend die Bezeichnungen lineare, polynomielle, exponentielle oder logistische Regression.

2.2.6 Maschinenlernen auf Graphstrukturen

Li, Cheng et al. geben in [LCW+10] eine Übersicht über mögliche Erweiterungen für die Verlustfunktion, um für verschiedene Datenstrukturen Zusammenhänge zwischen Merkmalen auszunutzen. Die Hypothese $h_{\mathrm{w}}(x) = \mathrm{w}x$ ist dabei stets ein Produkt aus Merkmalsvektor und Gewichtungsvektor, weshalb ab dieser Stelle die Schreibweise w statt ϑ bevorzugt wird. Für Graphstrukturen ist das Verfahren *GFLasso* [LCW+10, S. 25] angegeben. Die Zielfunktion lautet

$$\mathrm{w} = arg \min_{\mathrm{W}} \underbrace{loss(\mathrm{w}; X, y)}_{(1)} + \alpha \underbrace{\|\mathrm{w}\|_1}_{(2)} + (1-\alpha) \sum_{i,j} \underbrace{\mathrm{A}(i,j)|\mathrm{w}_i - sgn(r_{i,j}\mathrm{w}_j)|}_{(3)} \qquad \text{(Eq1)}$$

Idee: „Diejenige Gewichtungsmatrix w ist optimal, deren zugehörige Hypothese h_{w} die geringsten Abweichungen von der Zielfunktion hat (1), die gleichzeitig nicht zu große Einträge hat (2), und die für zusammenhängende Merkmale ähnliche (bei positiver Korrelation) bzw. verschiedene (bei negativer Korrelation) Gewichtungsvektoren aufweist (3).

Verlustfunktion loss

Als Verlustfunktion *loss* wird oft eine Funktion der Form
$loss_r(\mathrm{w}; X, y) := \|y - X\mathrm{w}\|^r$ für ein $r \in (0, \infty)$ verwendet. $loss_1$ heißt *Laplace-Verlust*, $loss_2$ *Gauß-Verlust*. $L_\varepsilon(\mathrm{w}; X, y) := \begin{cases} 1, & loss_1(\mathrm{w}; X, y) > \varepsilon, \\ 0, & loss_1(\mathrm{w}; X, y) \leq \varepsilon \end{cases}$ für ein $\varepsilon > 0$ heißt *0-1-Verlust* [Rü14, S. 20]. Durch sie werden fehlerhafte Vorhersagen „bestraft".

Regularisierungsterm

Der Strafterm $\|\mathrm{w}\|_1$ dient dazu, rauschbehaftete Trainingsdaten auszugleichen, die sich ansonsten in einer ebenso rauschbehafteten Vorhersagefunktion widerspiegeln könnten [FRK12]. Andererseits wird durch ihn „Merkmalsauswahl" realisiert: Es werden Gewichtungsmatrizen mit vielen Einträgen gleich 0 bevorzugt; idealerweise werden so Merkmale eliminiert, die für die Vorhersage irrelevant sind oder Abhängigkeiten mit anderen Merkmalen aufweisen, also redundant sind [LCW+10, S. 15].

Graphregularisierungsterm

Basis für den zweiten Strafterm ist die Annahme, dass der Zusammenhang zwischen verschiedenen Merkmalen aus dem Merkmalsraum $\mathcal{F}$ (bestehend aus $f := |\mathcal{F}|$ Merkmalen) als Kantenmenge E in einem ungerichteten Graphen $\mathcal{G} = (\mathcal{F}, E)$ gespeichert ist. Über $\mathcal{G}$ ist dann die Adjazenzmatrix $\mathrm{A} \in \{0,1\}^{f \times f}$ definiert durch $\mathrm{A}_{ij} = \begin{cases} 1, & (f_i, f_j) \in E \\ 0, & \text{sonst.} \end{cases}$

2.2.7 Der *Adam*-Algorithmus

Die Schrittweite α ist entscheidend dafür, dass das Gradientenabstiegsverfahren konvergiert. Bei zu großen Schrittweiten kreist ϑ unter Umständen um den Idealwert, zu kleine Schrittweiten zögern die Konvergenz unnötig hinaus. Der Algorithmus *Adam* (*Adaptive Momentum Estimation*) versucht, dieses Problem zu lösen, indem in jeder Iteration α neu und klug gewählt wird: Über den Gradienten g_t und seine Länge g_t^2 wird ein laufender Durchschnitt geführt, das *Moment* m_t und die *Geschwindigkeit* v_t. Diese Werte ergeben sich also aus der gewichteten Summe der vorigen Werte und des aktuellen Gradienten.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Die Gewichtungen β_1, β_2 sind dabei den Entwicklern von *Adam* zufolge groß zu wählen, die empfohlenen Werte sind $\beta_1 = 0.9, \beta_2 = 0.999$.

Es stellt sich nun heraus, dass β_1, β_2 nahe bei 1 dazu führen, dass m_t und v_t nur sehr zögerlich größer als ihr Initialwert 0 werden. Deshalb werden beide Werte in den ersten wenigen Iterationen um ein Vielfaches gestreckt.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Aus diesen korrigierten Werten ergibt sich schließlich der *Schritt* $\Delta\vartheta$:

$$\Delta\vartheta_t = \frac{-\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \qquad \vartheta_{t+1} = \vartheta_t + \Delta\vartheta_t$$

mit ε sehr klein, um zu verhindern, dass der Nenner 0 wird.

v_t (vor der Korrektur) steigt dabei recht langsam auf ein gewisses Niveau und nähert aich in jeder Iteration ein Stück der aktuellen Gradientenlänge an. m_t dagegen verändert sich rascher und passt sich neuen Richtungen in wenigen Schritten an. Die resultierende Schrittweite ist wegen der Korrektur zu Anfang sehr groß, nach einigen Iterationen stark proportional zu der Änderung der Steigung: Ebbt die Steigung in einem Gebiet allmählich ab, also etwa im Bereich eines lokalen Minimums, sinkt $\hat{m}_t$ im Vergleich zu $\hat{v}_t$ stark ab und die Schrittweiten werden schnell kürzer. Bleibt die Steigung über mehrere Iterationen konstant, nähert sich $\hat{v}_t$ $\hat{m}_t$ an und die Schritte verkürzen sich allmählich. Erreicht ϑ ein größeres Gefälle, so nimmt analog die Schrittweite zu.

So kann Adam im Vergleich zu Vorgängerstrategien schnell konvergieren und ist aktuell recht beliebt, auch als Basis für Weiterentwicklungen. [Rud17] gibt eine Übersicht über verschiedene Ausprägungen des Gradientenabstiegsverfahrens und zeigt die jeweils gegenüber den Vorgängeralgorithmen verbesserten Schwachpunkte auf. Ein solcher Schwachpunkt vieler dieser Verfahren soll zum Abschluss veranschaulicht werden.

Problemstellung der Sattelpunkte

Wie bei anderen Verfahren auch, stellen Sattelpunkte in der Verlustfunktion auch bei *Adam* eine Herausforderung dar. Der Gradient der Verlustfunktion nimmt an Sattelpunkten den Wert 0 an, aber es handelt sich weder um ein Maximum noch ein Minimum (indefinite Hessematrix), wie in Abbildung 2.3 an einem Beispiel gezeigt wird. Ist die Schrittweite so groß, dass ϑ in x-Richtung das Minimum überspringt, kommt es vor, dass ϑ in vielen Schritten zwischen den „Wänden“ des Sattels hin- und heroszilliert und nur wenig Fortschritt in y-Richtung macht, also in Richtung der potenziell optimalen Lösung. Im Extremfall (etwa, wenn ϑ im Bereich des Sattelpunkts initialisiert wird) nehmen die Schrittweiten und Gradienten immer weiter zu und *eskaliert* einige Iterationen lang weiter weg vom Ziel nach oben.

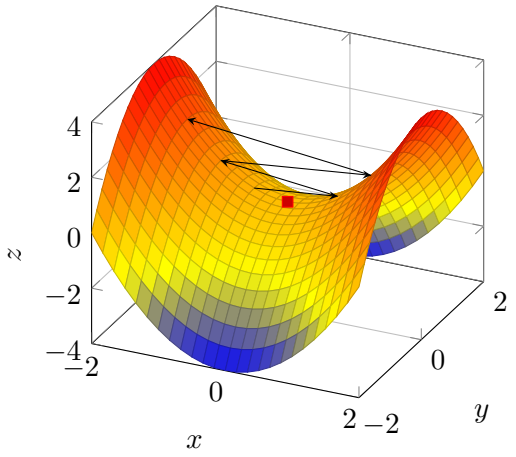


Abbildung 2.3: Sattelpunkt der Funktion $f(x, y) = x^2 - y^2$ beim Punkt (0,0,0) (rote Markierung), Eskalation von ϑ (Pfeile).

2.3 Bayes'sches Theorem, Bigramm-Analyse

Die verschiedenen Strategien, die das System zur Vorhersage benutzt, stützen sich im Wesentlichen auf das *Bayes'sche Theorem*

$$p(A_{n+1}|B) = \frac{p(B|A_{n+1})p(A_{n+1})}{p(B)} \left(= \frac{p(A_{n+1} \cup B)}{p(B)} = \frac{\#(BA_{n+1})}{\#B} \right),$$

mit einer infrage kommenden Folgeaktion A_{n+1} und einer gegebenen Aktionshistorie $B = (A_0, ..., A_n)$. Die Wahrscheinlichkeitsverteilung $p(B|A_{n+1})$ für alle möglichen A_{n+1} liegt aber in dieser Arbeit nicht explizit vor, deshalb wird auf die Umformung in der Klammer zurückgegriffen und die Wahrscheinlichkeitsverteilung aus der Häufigkeit der jeweiligen Ereignisse geschlossen. Im Folgenden sollen die auf der relativen Häufigkeit basierenden Vorhersagestrategien dennoch als *Bayes'sch* bezeichnet werden, um sie von jenen, die auf TensorFlow basieren, zu unterscheiden.
Nun stellt sich die Frage, wieviele zurückliegende Aktionen für die Bestimmung der Folgeaktion eine Rolle spielen. Eine valide Annahme wäre: Der Nutzer entscheidet sich stets spontan für eine Aktion, ohne dass die vorherigen Aktionen Einfluss auf die Entscheidung hätten.

$$p(A_{n+1}|B) = p(A_{n+1})$$

Dann wäre der Erwartungswert der folgenden Aktion schlicht die bisher häufigste Aktion. Nimmt man dagegen an, dass Nutzeraktionen doch zusammenhängend sind und bezieht

zumindest die letzte Aktion mit ein, bekommt man also eine Wahrscheinlichkeitsverteilung für jede vorangegangene Aktion.

$$p(A_{n+1}|B) = p(A_{n+1}|A_n)$$

Ist der Raum der statistischen *Ereignisse* Ω endlich und man betrachtet für die Vorhersage die letzten n Ereignisse, so ist es einfach, eine $|\Omega|^n$ Einträge große Statistik aufzustellen, die bei einer ausreichenden Datenmenge eine sehr gute Grundlage für die Vorhersage bieten sollte. Das ist in dieser Arbeit zwar der Fall, aber wegen der enormen Anzahl der Auswahlmöglichkeiten für Aktionen wohl keine gute Lösung.

2.4 Verwandte Arbeiten

An dieser Stelle sollen kurz einige Anwendungsfälle beschrieben werden, bei denen Verhaltensanalyse ebenfalls verwendet wird.

2.4.1 Agentenbasierte Systeme

Agentenbasierte Systeme nutzen sogenannte Agenten, Einheiten mit einem spezifierten Verhalten und einem eigenen Ermessens- und Entscheidungsspielraum. *Maxims* [Zhu01] etwa ist ein persönlicher Assistent zur Verwaltung des E-Mail-Verkehrs im E-Mail-Client *Eudora*, der das Verhalten des Nutzers erlernt. Für jede mögliche Aktion wird eine Konfidenz ermittelt. Übersteigt die Konfidenz den vom Nutzer bestimmten Schwellwert zur Aktionsfreigabe (*do-it level*), wird die Aktion ohne weiteren Einfluss des Nutzers ausgeführt. Andernfalls wird die Aktion dem Nutzer immerhin vorgeschlagen, wenn die Konfidenz das Mindestmaß zur Aktionsempfehlung (*tell-me level*) erreicht. Durch verschiedene Einstellungen dieser Schwellwerte kann der Grad der Autonomie angepasst werden.

2.4.2 Optimierung von Vorschlägen verwandter Suchanfragen

[KST12] beschäftigt sich mit Faktoren, die beeinflussen, als wie hilfreich ein Nutzer von Suchmaschinen eine vorgeschlagene verwandte Suchanfrage empfindet. Dabei sind einerseits die äußerliche Präsentation der Vorschläge, andererseits der Inhalt des Vorschlags und insbesondere das Verhältnis der vorgeschlagenen zur ursprünglichen Suchanfrage untersucht worden. Ergebnis der Untersuchungen ist der Prototyp einer Suchschnittstelle *SParQS* (*Specialization and Parallel movement Query Suggestion*), die verwandte Suchanfragen

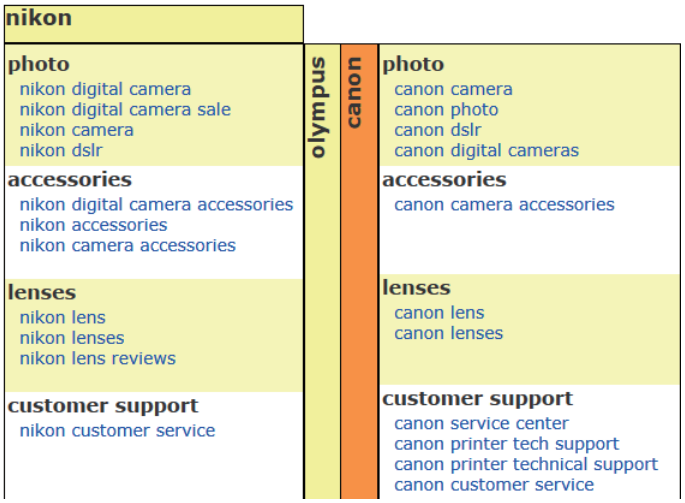


Abbildung 2.4: Die Schnittstelle von SParQS. Grafik übernommen aus [KST12]

nicht wie in den gängigen Suchmaschinen üblich als Liste, sondern gebündelt nach Kontext sortiert anzeigt. Im Beispiel führt die Suche nach dem Hersteller von Fotoapparaten *Nikon* zu einer strukturierten Übersicht beliebter Suchanfragen sortiert nach Produkttyp bzw. Dienstleistung, sowie dieselben Kategorien von Suchanfragen für die Konkurrenten *Olympus* und *Canon*. Eine Kombination der beiden Strategien *parallele Bewegungsrichtung* („nikon camera“ → „canon camera“) und *Spezialisierung* („nikon camera“ → „nikon camera sales“) sorgt dafür, dass die Probanden der Nutzerstudie die Suche mit *SParQS* im Mittel als einfacher, befriedigender und hilfreicher empfunden haben. Die Nutzerstudie zeigt aber auch, dass bei einfacher Informationsbeschaffung wie der Aufgabe *„Finden Sie Reiseinformationen über Peking“*, die Schnittstelle von *SParQS* die Nutzer eher von möglichen bereits relevanten Treffern ablenkt und sie dazu treibt, die Suchanfrage weiter anzupassen.

2.4.3 Softwaresicherheit und Sicherheitssoftware

Softwaresysteme, die Teile ihrer Dienste im Internet öffentlich machen, müssen gegen Angriffe vieler Art robust sein, da andernfalls große Schäden bei ihrem Betreiber entstehen können, solange das System funktionsunfähig bleibt. Eine wichtige Abwehrmaßnahme ist es, über Aktivitäten im Programm genügend Informationen aufzuzeichnen, um (fehlgeschlagene) Angriffe zu identifizieren und aus ihnen zu lernen. [McG04] empfiehlt, die Erkenntnisse über aktuelle Bedrohungen ständig in die eigenen Sicherheitsstrategien einzuarbeiten.
Gartner hat den Begriff der Nutzerverhaltensanalyse (*User Behaviour Analysis, UBA*) im Kontext der Cybersicherheit geprägt. Dabei wird ein Profil „normaler Nutzung“ erstellt und dem Betreiber Informationen über abweichendes, potenziell gefährliches Verhalten zur Verfügung gestellt. Besonderer Fokus liegt dabei auf der Identifikation von internen und externen Zugriffsberechtigten, die ihren Zugriff gezielt missbrauchen (*malicious insiders*). UBA-Systeme sind dabei nicht explizit dafür zuständig, solche Bedrohungen auch automatisch abzuwehren, sondern sollen helfen, die nötigen Maßnahmen zu ergreifen. Der Report [SLBP18] liefert eine Übersicht grundlegender Konzepte sowie aktueller Produkte und Entwicklungen, wie der zunehmenden Nutzung des Maschinenlernens.

2.4.4 Videospiele

Vielleicht mehr als jede andere Industrie bedient die Videospielindustrie die Sehnsucht nach Erfolgserlebnissen ihrer Kunden. Eine Strategie ist dabei, mit einem erfolgreichen Titel Interesse an einem Nachfolger zu wecken, der an den ersten Teil anknüpft und so unter Umständen schließlich über Jahrzehnte hinweg eine mächtige Marke zu etablieren, deren Produkte die Spieler von damals und heute durch ihr Leben begleitet und sie emotional an sie bindet. Neuerscheinungen solcher Markenimperien auf dem Videospielmarkt, von denen es inzwischen unzählige gibt, sind geradezu automatische Verkaufserfolge; an der Spitze steht aktuell *The Pokémon Company* mit insgesamt umgerechnet 48 Milliarden Euro Umsatz seit der Gründung im Jahr 1996 (6 Trillionen Yen, Stand: März 2017 [Pok]). Ein jüngeres Verkaufsmodell, *Mobile* bzw. *Social Gaming* geht häufig aggressiver vor, indem es den Kunden mit Erfolgserlebnissen zunächst überhäuft, sie ihm später wiederum gezielt entzieht: das Spiel wird dann plötzlich eine bestimmte Zeit lang völlig unbrauchbar, der Nutzer ist gezwungen, zu warten. Die Wartezeit kann jedoch einfach mit bestimmten Gegenständen verkürzt werden, die innerhalb weniger Sekunden direkt im Spiel für „echtes Geld“ erhältlich sind. Umständliche Währungssysteme im Spiel verschleiern den tatsächlichen Preis, den es kostet, der Langeweile ein Ende zu setzen, die zu ertragen viele Spieler mit der Zeit verlernen.
Madigan bespricht in seinem Buch *Getting Gamers* [Mad16] die psychologischen Aspekte des Videospielentwurfs und -konsums. [SDB18] gibt einen Überblick über die aktuelle

Praxis des *Profiling*, durch das Spielern z.B. individuelle Angebote für kostenpflichtige Spielinhalte gemacht werden können, wenn sie im Begriff sind, das Spiel zu beenden.

2.4.5 Ermittlung von Aktionsvorschlägen

Zahlreiche *Office-/Productivity*-Suiten wie *Microsoft Office* und *Google Docs* bieten dem Nutzer an, die nächste gewünschte Funktion mittels eines Textfelds zu suchen, wenn er sie in der Menüstruktur nicht mehr finden kann. Bereits wenige Buchstaben eines Schlagwortes oder eines Synonyms können ausreichen, um eine Funktion mit der sogenannten *„Sie wünschen“*-Funktion bei Microsoft Office 2016 zu finden. Gelegentlich werden auch unaufgefordert Aktionen vorgeschlagen, etwa beim Öffnen eines Dokuments an die Stelle zu springen, die bei der letzten Sitzung als letzte bearbeitet wurde.
Kontextbezogene Aktionsvorschläge ohne Eingabe eines Suchwortes sind bisher auch bei Office 2016 nur selten zu finden; etwa wird die Aktion „Grafik zuschneiden“ vorgeschlagen, wenn eine Grafik markiert ist. Allerdings sind die wichtigsten Funktionen für Bilder, Tabellen und Ähnliches bereits übersichtlich im Kontextmenü zu finden. Vorschläge auf der Basis der Gewohnheiten des Nutzers reichen nur soweit, dass die letzten wenigen ausgewählten Funktionen in einer Liste anwählbar sind.

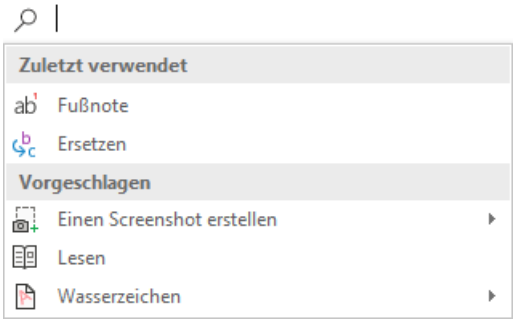


Abbildung 2.5: Das Suchfeld „Sie wünschen“ bei Word 2016.

Intelligente Persönliche Assistenten wie *Google Assistant*, *Apple Siri* und *Amazon Alexa* können dagegen auf ein breites Spektrum von Informationen zurückgreifen, um Gewohnheiten des Nutzers zu speichern und deren Basis hilfreiche Informationen in bestimmten Situationen anzubieten wie etwa:

- Fahrpläne des öffentlichen Nahverkehrs, kurz bevor der Nutzer zur Arbeit geht oder den Arbeitsplatz verlässt; Verbindungsinformationen von Fernverkehrsverbindungen, die gebucht worden sind
- Zwischen- und Endergebnisse der Lieblingsmannschaft, oder gerade dann nicht, wenn der Nutzer nicht will, dass ihm die Spannung verdorben wird
- das Wetter des kommenden Tages, spätabends

Da die Datenverarbeitung und -speicherung in der Regel nicht lokal stattfindet, setzt sich der Nutzer mit der Verwendung solcher Assistenzsysteme dem Risiko erheblicher Eingriffe in seine Privatsphäre aus. Verschiedene technische Fehler haben in der Vergangenheit private Daten in großem Umfang zugänglich gemacht.

Kapitel 3

Das System im Detail

Im folgenden Abschnitt soll das System näher erläutert werden. Um einen Überblick zu geben, wird zunächst erläutert, welche Funktionen die grafische Nutzeroberfläche anbietet. Danach werden die Komponenten der Architektur einzeln vorgestellt. Als drittes wird auf zentrale Entwurfsentscheidungen eingegangen, die das Kernstück dieser Arbeit bilden.

3.1 Nutzung des Systems mit der grafischen Benutzeroberfläche

Die grafische Benutzeroberfläche bietet Zugriff auf alle Funktionen, die die Schnittstelle des Systems zur Verfügung stellt. An dieser Stelle soll gezeigt werden, welche Schritte nötig sind, um das System über die Oberfläche zu steuern.

3.1.1 Verwaltung der Tabellen

Zunächst werden die Tabellen in den Datenspeicher importiert. Dazu wird im Dialog *"Add new table"* (siehe Abbildung 3.2) die entsprechende Datei in der Oberfläche angewählt, der Name der Datenbank angegeben, und ein Titel eingetragen, der den Inhalt der Tabelle sprechend beschreiben sollte. Ist die Tabelle vollständig eingelesen, ist die Verwendung unabhängig von der angegebenen Datei, da eine lokale Kopie angelegt wird.

Abbildung 3.1: Dialog zur Registrierung einer Tabelle

Für alle Tabellen, die im Datenspeicher hinterlegt sind, werden die angegebenen Informationen sowie Metadaten angezeigt. Diese sind eher weniger für den Nutzer des Klientsystems als vielmehr für Entwickler hilfreich, die die REST-Schnittstelle des Systems nutzen wollen und prüfen wollen, ob die Tabellen korrekt eingelesen worden sind.

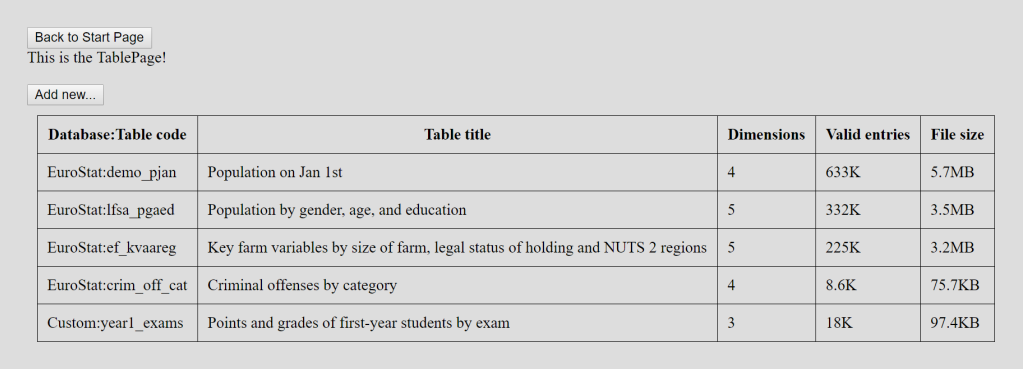


Abbildung 3.2: Übersicht über die bekannten Tabellen

3.1.2 Eintragen von Aktionen

Der lokale Nutzer arbeitet mit einem Klientensystem auf Datensätzen, von denen er gewisse Daten durch natürlichsprachliche Befehle anfordert. Um das Modell auf Aktionsfolgen zu trainieren, werden solche Paare *(Nutzereingabe, Datenausgabe)* an den Datenspeicher übergeben, zum Beispiel automatisch im Hintergrund eines Klientensystems. Jedes Mal, wenn das System gestartet wird, wird eine neue Sitzung angelegt, innerhalb der jede Aktion, die eingetragen wird, sowohl als Nachfolger der letzten Aktion als auch als Vorgänger der gegebenenfalls folgenden Aktion behandelt. Jeweils die erste Aktion einer Sitzung wird als Nachfolger von *NoData* behandelt.
Die Nutzereingabe wird als Klartext, die Antwort als Datenkodierung übergeben. *NoData* im Sinne einer fehlgeschlagenen Auswertung ist dabei kein gültiger Nachfolger, weil im Modell nur die Absichten des Nutzers zum Ausdruck kommen sollen, über die eine fehlgeschlagene Auswertung der Eingabe nichts verrät. Der Dienstgeber verhindert jedoch nicht, dass eigene Implementierungen der REST-Schnittstelle *NoData* als Aktion eintragen.

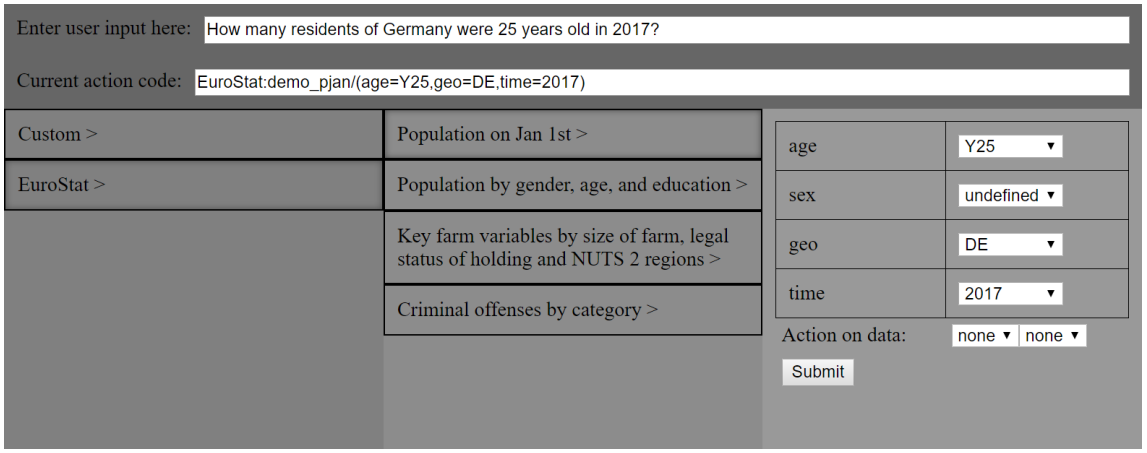
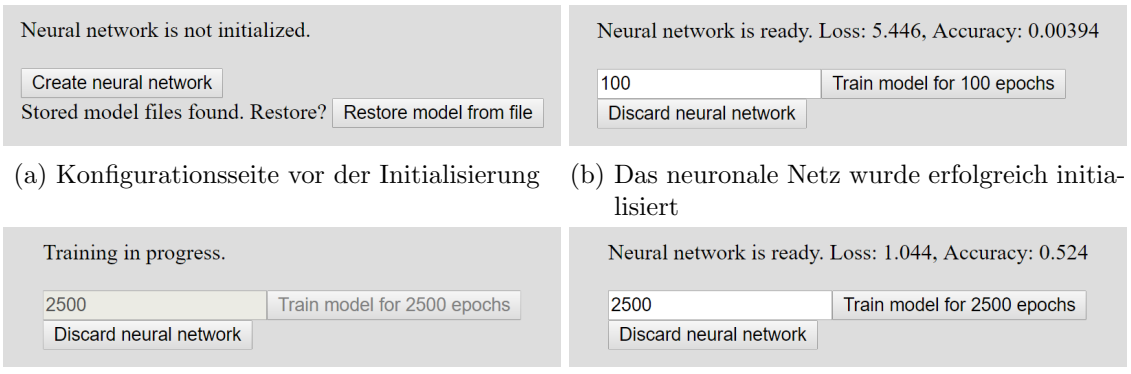


Abbildung 3.3: Beispielhafte Eingabe der Anfrage Q1 in die SubmitPage

3.1.3 Einrichtung des neuronalen Netzwerks

Das neuronale Netzwerk wird nicht automatisch eingerichtet, da das Training viel Rechenleistung und Arbeitsspeicher benötigt. Es kann bei erster Verwendung über die grafische

Oberfläche erstellt werden; danach wird nach jeder Änderung eine Sicherung angelegt. Wird eine Sicherung gefunden, kann der letzte Stand wiederhergestellt werden.
Ist ein Netzwerk geladen, werden die aktuellen Auswertungen der Verlustfunktion und die



(a) Konfigurationsseite vor der Initialisierung

(b) Das neuronale Netz wurde erfolgreich initialisiert

(c) Während des Trainings

(d) Nach einigen Trainingszyklen stellt sich ein Lernerfolg ein

Abbildung 3.4: Die Konfigurationsseite der grafischen Oberfläche in verschiedenen Stadien

Vorhersagegenauigkeit angegeben.
Ist die Bewertung des Modells nach der Einschätzung des Nutzers oder den Richtlinien des Klientsystems unzureichend, also etwa unmittelbar nach der Initialisierung oder nachdem neue Aktionen in das Modell eingetragen worden sind, kann es eine beliebige Anzahl Epochen trainiert werden. Nach dem Training können die neuen Bewertungen eingesehen werden.

3.1.4 Bildung einer Vorhersage

Nun kann mit einer gewählten Vorhersagestrategie eine Vorhersage getroffen werden. Standardmäßig wird die letzte Aktion der Sitzung als Referenzpunkt verwendet, es kann aber auch eine beliebige Datenkodierung als letzte Funktion angegeben werden, insbesondere zu Testzwecken. Sofern eine Nutzereingabe verfügbar ist, kann diese ebenfalls eingetragen werden; diese wird aber ausschließlich bei der Strategie *InputPrediction* verwendet.

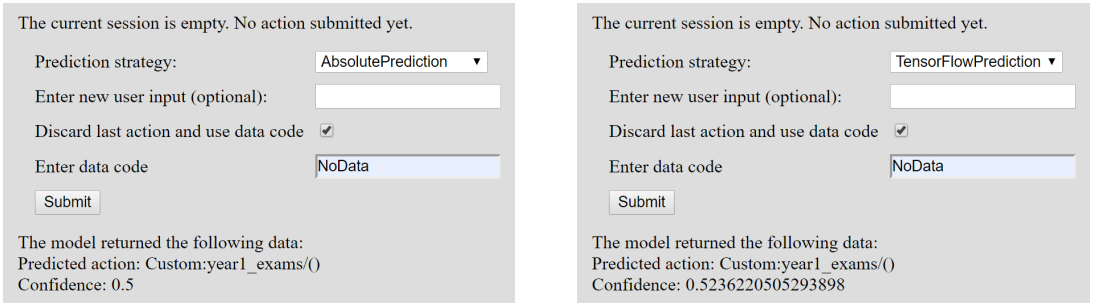


Abbildung 3.5: Ergebnisse der Vorhersage für verschiedene Vorhersagestrategien

3.2 Die Architektur des Systems

Das System besteht aus drei wesentlichen Komponenten: der Datenhaltung, der Datenverarbeitung und der REST-Schnittstelle. Die Interaktion zwischen den Komponenten ist in Abbildung 3.6 angedeutet. Der Aufbau und die Funktionen der einzelnen Komponenten werden im Folgenden erläutert.

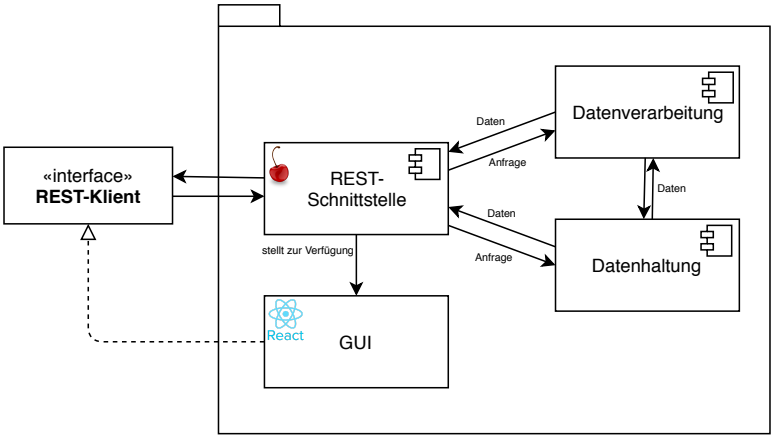


Abbildung 3.6: Überblick über die Komponenten und Schnittstellen des Systems

3.2.1 Datenhaltung

Ein *DataStore*-Objekt, das als Einzelstück implementiert ist, hält je eine Liste aller Tabellen und aller Datenbanken, sowie das Modell und bildet damit die Grundlage fast aller Anfragen. Außerdem sind in der Datenhaltungskomponente alle Klassen hinterlegt, deren Hauptzweck es ist, Daten zu speichern.

3.2.1.1 Das Modell

Das Modell ist baumartig aufgebaut und besteht auf jeder Ebene aus `dict`-Objekten mit einem Zähler. Die Schlüssel jeder Ebene sind `String`-Objekte. Die Ebenen haben folgende Bedeutung:

1. Das Modell selbst enthält Einträge für alle Datenmengen, für die Nachfolger bekannt sind.
2. Jeder Datenmenge werden die Datenbanken zugeordnet, aus denen die Nachfolger stammen.
3. Jede Tabelle verweist auf die Datenmengen, die aus dieser Tabelle stammen.
4. Der Eintrag für jedes Datenobjekt hält neben dem Zähler eine Liste von Nutzereingaben in natürlicher Sprache, denen dieses Datenobjekt zugeordnet wurde.

Wurde also die Sequenz (Q1, Q2) beobachtet, hat das Modell einen Eintrag

$$\text{Eurostat:demografie/(age=Y25,geo=DE,time=2017)},$$

von dem der Pfad

$$\text{Eurostat} \rightarrow \text{farm\_statistik} \rightarrow [(\text{index=TOTAL\_HA}) \rightarrow \text{avg(geo)}]$$

ausgeht.

3.2.1.2 Datenklassen

Die Datenklassen seien hier kurz aufgelistet und erklärt, wie sie verwendet werden. Zu beachten ist, dass in Python Mehrfachvererbung möglich ist. Abstrakte Datentypen stehen in *kursiver Monospace*-Schrift.

Klasse	Basistyp	Verwendung
`Table`	Liste von `Dimension`en	Speichert alle Dimensionen einer Tabelle und den Namen ihrer Datenbank.
`Dimension`	Liste von `String`	Speichert alle Werte, die eine Dimension annehmen kann, als Zeichenketten.
`TFData`	*`Serializable`*	Bietet Methoden zur Verarbeitung der Daten in TensorFlow.
`EurostatData`	*`TFData`*, `dict: Dimension→String`	Speichert zu spezifizierten Dimensionen deren Werte.
`DataSelection` `DataReduction`	`EurostatData` `DataSelection`	Speichern zusätzlich zu Dimensionswerten eine `DataOperation`, die Ziel-Dimension sowie ein Basisdatenobjekt.

`DataOperation`
 	
`SelectOperation`
`ReduceOperation`
	-
	
	`DataOperation`
	`SelectOperation`
		Speichert eine Operation, die aus mehreren Daten mit gleichen Dimensionswerten
		...je ein Datum auswählt.
		...je ein Datum generiert.

`Action`	-	Speichert eine Nutzereingabe als `String` und die zurückgegebene Datenmenge als `EurostatData`.

3.2.1.3 Sicherung des Datenspeichers in einer JSON-Datei

Bei jeder Änderung wird eine JSON-Kodierung des Datenspeichers in eine Datei geschrieben, die nach dem Neustart des Systems insbesondere das Modell wiederherstellen kann. Die Kodierung wird durch die Standardbibliothek `jsonpickle` verwirklicht.
Bei der Dekodierung traten im Laufe der Entwicklung des Systems häufig Probleme auf; im Kern scheint es `jsonpickle` nicht möglich zu sein, `dict`-Objekte, die sowohl objektwertige Schlüssel als auch Attribute besitzen (so wie etwa die `ModelEntry`-Klasse), in derselben Reihenfolge abzuarbeiten wie bei der Kodierung, was dazu geführt hat, dass Querverweise, die Dopplungen identischer Objekte vermeiden sollten, falsch aufgelöst wurden. Eigene `Handler`-Klassen für die Dekodierung sowie die Implementierung eines Stellvertreter-Modus für `EurostatData`-Objekte haben das Problem schließlich behoben.

3.2.2 Datenverarbeitung

Die Klassen der Datenverarbeitungskomponente sind insbesondere dafür zuständig, relevante Daten aus der Tabelle einzulesen und die in Abschnitt 3.3.3 erläuterten Datenoperationen umzusetzen. Die Umsetzung dieser Funktionen wird hier knapp beschrieben.

3.2.2.1 Einlesen von Eurostat-Tabellen

Wird eine Tabelle über die Schnittstelle registriert, wird eine lokale Kopie angelegt, weil sie sonst nicht mehr zugreifbar wäre, sobald das übergebene Objekt gelöscht wird. Außerdem werden die Namen der Dimensionen und alle Dimensionswerte aufgelistet. Das Ergebnis ist ein Objekt der Klasse `EurostatTable`, das im Datenspeicher abgelegt wird.

3.2.2.2 Datenextraktion aus Eurostat-Tabellen

Die Klasse `EurostatData` und ihre Unterklassen verfügen über die Funktion `extractFromTable()`, die die entsprechenden Daten aus der Tabelle extrahiert. Dafür werden die Dimensionswerte jeder Zeile mit denen des Zieldatenobjekts verglichen und geprüft, ob die Zeile relevante Einträge enthält. Eine genaue Beschreibung der notwendigen Bedingungen ist in 3.3.4 zu finden.

Diese Funktion wurde implementiert, um das Verhalten des Klientsystems zu simulieren, das die konkreten Daten aus Datensätzen extrahieren muss, um sie dem Nutzer als Antwort auf seine Anfrage zu liefern. Sie ist jedoch recht teuer in der Laufzeit und kann aktuell nicht in der Schnittstelle angesprochen werden.

3.2.2.3 Datenoperationen

Datenoperationen auf extrahierten Daten *data* aus einer Tabelle t werden entlang einer Dimension dim_{op} ausgeführt. Dabei sind möglicherweise in der *Basisdatenmenge* (also der Datenmenge vor Anwendung der Datenoperation und aller folgenden Schritte) bereits die Werte einiger Dimensionen $dims_{spec}$ bereits spezifiziert worden und stimmen somit in allen Einträgen von *data* überein, übrig bleiben die unspezifizierten Dimensionen $dims_{unspec} = t.dims \setminus (dims_{spec} \cup \{dim_{op}\})$.
Zunächst werden die Einträge von *data* nach ihren Dimensionswerten der $data_{unspec}$ sortiert. Anschaulich entspricht das einer Transformation der Tabelle t derart, dass nun die Werte der Dimension dim_{op} entlang der ersten Zeile aufgelistet werden, wie in der folgenden Tabelle dargestellt. Die Operation wird nun für jede Zeile angewendet.

age,time\geo	AD	AL	AM	AT	AZ	BA
Y1,2017	:	1423050	1567380	4460424	4918771	:
Y1,2016	:	1417141	1569535	4427918	4870002	:
Y1,2015	:	1424597	1571450	4384529	4817181	:
Y1,2014	:	1430827	:	4352447	4763571	:
Y1,2013	37408	1437193	:	4328238	4707690	:
Y1,2012	38252	1444234	1684000	4309977	4651601	1963655

Abbildung 3.7: Der nach der Dimension *geo* transformierte Tabellenausschnitt 2.1. Die Dimension *sex* wurde auf den Wert F festgelegt. Jede offene Wertekombination steht nun in einer Zeile.

Das `DataOperation`-Objekt, das die gewählte Operation umsetzen soll, erhält ein `dict`-Objekt, das für jede Zeile eine Liste von Tupeln $(Datenmenge, Wert)$ enthält. Aus den Werten wird das Ergebnis bestimmt, also etwa der Mittelwert oder das Minimum. Vom Typ der Dataenoperation hängt ab, welche Dimensionswerte mit zurückgegeben werden: Bei `SelectOperation`en wird der gewählte Wert für dim_{op} mit zurückgegeben. Bei `ReduceOperation`en wird das Ergebnis aus allen Werten der Zeile berechnet, somit gibt es keinen passenden Wert für dim_{op}.

3.2.3 REST-Schnittstelle

Die REST-Schnittstelle wurde geschaffen, um es so einfach wie möglich zu machen, die Funktionen des Systems zu nutzen. Der Dienstgeber beruht auf der Python-Bibliothek *CherryPy*, die REST-Anweisungen an einem Port entgegennimmt und in den Aufruf einer Python-Methode übersetzt.

age,time	max(geo)	avg(geo)
Y1,2017	((age=Y1, time=2017, geo=EEA31), 2518890)	((age=Y1, time=2017), 332091)
Y1,2016	((age=Y1, time=2017, geo=EEA31), 2532190)	((age=Y1, time=2017), 340367)
Y1,2015	((age=Y1, time=2017, geo=EEA31), 2508395)	((age=Y1, time=2017), 336422)
Y1,2014	((age=Y1, time=2017, geo=EEA31), 2556722)	((age=Y1, time=2017), 360342)
Y1,2013	((age=Y1, time=2017, geo=EEA31), 2580068)	((age=Y1, time=2017), 344599)
Y1,2012	((age=Y1, time=2017, geo=EEA31), 2638007)	((age=Y1, time=2017), 345449)

Abbildung 3.8: Ergebnisse für die Selektionsoperation max(geo) und die Reduktionsoperation avg(geo) auf den Daten von Abbildung 3.7. EEA31 ist der Europäische Wirtschaftsraum mit 31 Mitgliedsstaaten (2013 bis 2018), siehe auch 5.1.5.

Aufruf:
http://localhost:2107/getTable?database=Eurostat&filename=demography

Angesprochene Python-Methode:

```
@cherrypy.expose
def getTable(database, filename):
  ...
```

Antwort: JSON-Objekt
{'database': 'Eurostat', 'filename': 'demography', 'dimensions': [...], ...}

Die REST-Schnittstelle ist aber nicht nur in der Lage, REST-Anfragen zu bedienen, sondern zeigt unter der URL '/' eine grafische Benutzeroberfläche (GUI) an.

3.2.3.1 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche nutzt die Bibliothek *React* und stellt eine *Single-Page-Webanwendung* dar, alle Ereignisse werden also auf derselben Seite verarbeitet. Während der gesamten Nutzung bleibt die URL dieselbe. Hier wird der Nutzer dabei unterstützt, den gesamten Prozess im Browser zu durchlaufen, siehe Abschnitt 3.1.

3.3 Zentrale Überlegungen bei der Entwicklung

3.3.1 Datenmengen

Die Antwort einer Abfrage durch den Nutzer ist eine Teilmenge der Daten einer Tabelle, für die in der Regel einige Dimensionswerte festgelegt worden sind. So ist die Antwort auf die Anfrage

„Wieviele 25-jährigen lebten 2017 in Deutschland?“ (Q1)

in einer entsprechenden Tabelle in einer Zelle mit den Dimensionswerten ($age : Y25$, $geo : DE, time : 2017, sex : TOTAL$) zu finden. Sind n Dimensionswerte nicht in der Frage angegeben, ist das passende Ergebnis eine n-dimensionale Teilmenge der Daten der Tabelle.

3.3.2 NoData

NoData-Objekte sind Platzhalter, die kennzeichnen, dass keine passenden/noch überhaupt keine beobachteten Datenmengen verfügbar sind. Der jeweils ersten Aktion in einer Sitzung wird NoData als Vorgängerdatum zugeordnet.

3.3.3 Datenreduktionen und -selektionen

In Anlehnung an [Fuc17] sollen auch Reduktions- und Selektionsoperationen (entlang einer Dimension) auf den Daten unterstützt werden. Bei Selektionsoperationen ist das Ergebnis einer der ursprünglichen Zellwerte, bei Reduktionsoperationen wird ein Wert ermittelt, der nicht notwendigerweise in einer der Zellen stand. Folgende Operationen sind aktuell enthalten:

Maximum - Minimum - Median - Mittelwert - Zählen

Durch diese Erweiterung ist es möglich, Datenmengen zu modellieren, die Fragen der Art

„Wieviele Hektar Land umfassen Landwirtschaftsbetriebe durchschnittlich?“ (Q2)

beantworten.

3.3.4 Teilmengenbeziehung auf mehrdimensionalen Daten

Bei der Extraktion von Daten aus Tabellen werden die Dimensionswerte der jeweiligen Zeile $data_{line}$ mit denen des Datenobjekts $data_{obj}$ verglichen und geprüft, ob sie sich widersprechen. Erlaubt ist es demnach, wenn die Zeile eine Dimension spezifiziert, die in $data_{obj}$ offengelassen wurde. Diese Teilmengenbeziehung lässt sich auch als Halbordnung $\geq$ auf Datenmengen interpretieren, wobei gerade gilt:

$$data_{obj} \geq data_{line} :\Leftrightarrow \forall dim \in table : \begin{cases} data_{line}\,[dim] = data_{obj}\,[dim] \text{ oder} \\ dim \text{ nicht spezifiziert in } data_{obj} \end{cases}$$

Die beiden Fälle und die spezielle Rolle der Dimension entlang der ersten Zeile (*„Y-Dimension“*) wird in Abbildung 3.9 an einigen Beispielfällen verdeutlicht. Ein Ausschnitt der Tabelle ist in 2.1 abgedruckt.

Zeilenwerte $data_{line}$	**Vorgegebene Werte** $data_{obj}$	$data_{line}$ **Teil des Ergebnisses?**
geo=DE, age=Y25, sex=M	*geo=DE, age=Y30*	Nein, da die Werte von *age* konfligieren
geo=DE, age=Y30, sex=M	*geo=DE, age=Y30*	Ja
geo=DE, age=Y30, sex=M	*geo=DE, age=Y30, time=2015*	Ja, aber nur die Spalte mit *time=2015*
geo=DE, age=Y30, sex=M	*<keine>*	Ja

Abbildung 3.9: Beispielfälle für den Vergleich zwischen Datenobjekt und Zeilenwerten bei der Extraktion von Daten aus Tabellen

3.3.5 Kodierung von Datenmengen

Für verschiedene Zwecke sind in diesem Projekt mehrere verschiedene Kodierungen von Datenmengen nötig geworden. Die allgemeine Form und die Eigenschaften der Kodierungen sollen hier kurz erläutert werden.

3.3.5.1 Zeichenkettenkodierung

Zunächst wurde eine Kodierung als Zeichenkette gesucht, um Datenmengen zwischen der grafischen Benutzeroberfläche und der Serveranwendung zu transportieren. Die allgemeine Form lautet:

```
Datenbank:Tabelle/(DimA=valA,DimB=valB,...)->operationName(dimension)->...
```

Die Kodierung besteht aus der Angabe der Datenbank und der Tabelle, und darauffolgend einer Liste von Spezifikationen. Spezifiziert wird eine Datenmenge, indem entweder mindestens ein Dimensionswert festlegt wird, oder eine Reduktions- bzw. Selektionsoperation auf den Daten ausgeführt wird. Da die Reihenfolge keine Rolle beim Festlegen von Dimensionswerten spielt, sollen unmittelbar aufeinanderfolgenden Spezifikationen von Dimensionswerte immer zusammengefasst werden.
NoData-Objekte erhalten die Kodierung *NoData*.

Diese Kodierung ist recht gut menschenlesbar, einigermaßen kompakt und gut dazu geeignet, als Zeichenkette zwischen den Teilen des Systems übertragen zu werden.

3.3.5.2 Tensorkodierung

Um TensorFlow auf Datenmengen trainieren zu können, wurde eine Darstellung als Tensor, in diesem Fall konkret als Matrix, entwickelt. Diese Darstellung ist abhängig vom Zustand des persönlichen Profils, also nicht zwischen verschiedenen Nutzern sinnvoll austauschbar.

$$\begin{bmatrix} db & t & v_{0,1} & ... & v_{0,d} \\ ai_1 & ad_1 & v_{1,1} & ... & v_{1,d} \\ & & ... & & \end{bmatrix}$$

mit db, t Index der Datenbank bzw. der Tabelle innerhalb der jeweiligen Liste, $v_{i,j}$ Index des gewählten Dimensionswerts für die j-te Dimension in Schritt i, oder -1, falls diese Dimension nicht im i-ten Schritt gewählt wird; ai_i Index der Operation vor der i-ten Auswahl von Dimensionswerten, ad_i Index der Dimension, auf der die Operation ausgeführt werden soll.
NoData-Objekte erhalten die Kodierung $\begin{bmatrix} -1 & -1 \end{bmatrix}$.

Es ergeben sich einige Beschränkungen für gültige Datenmatrizen:

- Sei DM eine Datenmenge aus der Tabelle T, $\#dims$ die Anzahl der Dimensionen von T, und $\#ops$ die Anzahl der Reduktions-/Selektionsoperationen, die auf DM ausgeführt werden. Dann hat die Tensordarstellung von DM gerade $\#dims + 2 \times \#ops + 1$ Einträge.
- Da für eine Dimension nur einmal ein Wert festgelegt werden kann, ist $v_{i,j}$ für festgelegtes j nur für höchstens ein i *nicht* -1. Wird über Dimension j reduziert oder selektiert, ist also $ad_i = j$ für ein i, so ist $v_{i,j} = -1$ für alle i.

3.3.5.3 Beispiel

Sei die Liste der Datenbanken im Speicher [Eurostat, Eigene], und die Liste der Tabellen [Eurostat:demografie, Eurostat:farm_statistik, Eigene:temperatur]. Dann ist die Antwort auf Anfrage Q1:

`Eurostat:demografie/(age=Y25,geo=DE,time=2017)` $\begin{bmatrix} 0 & 0 & 25 & 8 & 1 \end{bmatrix}$,

da *Eurostat* und *demografie* jeweils den Index 0 haben und die Werte Y25, DE und 2017 die Indizes 25, 8, 1 in den jeweiligen Listen der Dimensionswerte. Die Antwort auf Q2 lautet:

`Eurostat:farmStatistik/(index=TOTAL_ HA)->avg(geo)` $\begin{bmatrix} 0 & 1 & 2 & -1 & -1 & -1 \\ 3 & 1 & -1 & -1 & -1 & -1 \end{bmatrix}$

Hier ist abzulesen, dass die Datenoperation mit Index 1 durchgeführt wird, die die Dimension mit Index 3 betrifft; somit ergeben sich auch zwei Zeilen. Bereits vor der Datenoperation wird die Dimension *index* (Index 0) mit dem Wert *TOTAL_HA* (Index 2) spezifiziert.

Die verwendeten Indizes sind willkürlich und die angegebenen Daten dienen nur dem Beispiel.

3.3.6 Bayes'sche Vorhersagestrategien

Anhand des vorgestellten Modells und der letzten Aktion A_n der Sitzung $S = (A_1, ..., A_n)$ sollten nun Strategien zur Vorhersage implementiert werden. Später wurde für alle Strategien ein Konfidenzmaß entwickelt, das durch eine Zahl im Intervall $[0, 1]$ zum Ausdruck bringen sollte, wie sicher die Vorhersage ist. Eine Konfidenz von 0 würde bedeuten, dass die Vorhersage kein sinnvolles Ergebnis hervorbringen konnte, etwa weil A_n keinen bekannten Nachfolger im Modell hat. Eine Konfidenz von 1 würde bedeuten, dass bisher der Nachfolger von A_n stets derselbe war. Es wird jedoch nicht berücksichtigt, wie oft A_n überhaupt vorgekommen ist; ist also die Sequenz (A_n, A_{n+1}) bisher genau einmal beobachtet worden und liegt kein anderer Nachfolger von A_n vor, wird die Konfidenz 1 ausgegeben.

3.3.6.1 Gestaffelte Vorhersage

Idee: „Zu einer gegebenen Aktion A_n ist A_{n+1} der wahrscheinlichste Nachfolger, wenn die Datenbank von A_{n+1}, die Tabelle innerhalb dieser Datenbank, und die Datenmenge innerhalb der Tabelle jeweils am häufigsten nach A_n beobachtet wurden."
Umsetzung: Wähle im Modell iterativ den Kindknoten mit dem größten Zählerwert.

Diese Strategie schien zunächst die naheliegendste zu sein, allerdings zeigte sich, dass diese Strategie nicht notwendigerweise die *tatsächlich* häufigste Nachfolgeaktion bestimmt. Betrachte folgendes Modell:

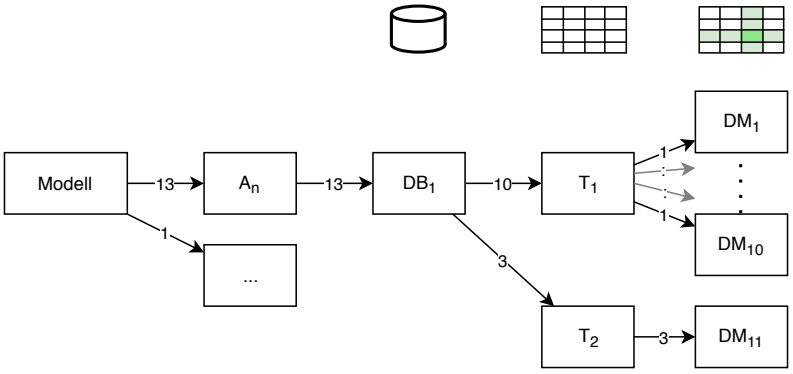


Abbildung 3.10: Bei diesem Modell gibt die gestaffelte Vorhersage nicht das Ergebnis aus, das potenziell zu erwarten wäre.

Die häufigste Aktion nach A_n ist mit drei Treffern Datenmenge 11. Jedoch wurde Tabelle 2 seltener ausgewählt als Tabelle 1, somit gibt die gestaffelte Vorhersage somit stattdessen eine der Datenmengen 1 bis 10 zurück.

Konfidenz: Das Konfidenzmaß sollte solche möglichen Streuungen berücksichtigen.

$$conf_{Stag} := \prod_{i=1}^{3} \frac{countMax_i}{count_{i-1}}$$

Die Konfidenz im Modell in Abbildung 3.10 für Datenmenge 1 wäre $conf_{Stag}(DM_1) = \frac{13}{14}\frac{13}{13}\frac{10}{13}\frac{1}{10} \approx 7.14\%$, für Datenmodell 11 hingegen $conf_{Stag}(DM_{11}) = \frac{13}{14}\frac{13}{13}\frac{3}{13}\frac{3}{3} \approx 21.43\%$.

3.3.6.2 Absolute Vorhersage

Idee: „Zu einer gegebenen Aktion A_n ist A_{n+1} der wahrscheinlichste Nachfolger, wenn keine andere Nachfolgeaktion von A_n häufiger vorgekommen ist als A_{n+1}."
Umsetzung: Durchsuche den Baum des Modells nach dem Blattknoten mit dem größten Zählerwert und verfolge nur vielversprechende Kanten.
Konfidenz: Hier wurde das Korrelationsmaß *corr* aus Unterabschnitt 3.3.7 herangezogen und linear vom Wertebereich $[-1, 1]$ auf $[0, 1]$ transformiert.

$$conf_{Abs} := \frac{corr(this, other) + 1}{2}$$

Analog zu oben ist $conf_{Abs}(DM_1) \approx 48.02\%$, $conf_{Abs}(DM_{11}) \approx 64.08\%$.

3.3.6.3 Vorhersage anhand der Eingabe

Idee: „Wenn A_n und die nächste Nutzeranfrage q_{n+1} gegeben sind, dann ist A_{n+1} der wahrscheinlichste Nachfolger, wenn A_{n+1} bereits einmal mit einer Nutzeranfrage aufgerufen wurde, die q_{n+1} mehr ähnelt als jede andere Nutzereingabe nach A_n."
Umsetzung: Berechne die Levenshtein-Distanz zwischen A_{n+1} und allen beobachteten auf A_n folgenden Nutzereingaben und wähle die Nutzereingabe mit der geringsten Distanz (relativ zu der Eingabelänge).
Die Levenshtein-Distanz $lev_{A,B}$ zwischen zwei geordneten Behältern (Wörter, Sätze, Listen) A und B gibt die kleinste Anzahl an Operationen an, die nötig ist, um A an B anzugleichen. Zur Wahl stehen dabei die Operationen *Einfügen*, *Löschen* und *Ersetzen*. Da *Einfügen* und *Löschen* dual sind sowie *Ersetzen* selbstdual ist, ist *lev* symmetrisch.

Da nun $lev_{A,B} \leq max(|A|, |B|)$, wird bei der Suche nach der ähnlichsten Nutzeranfrage $1 - \frac{lev_{q_{n+1},q}}{max(|q_{n+1}|, |q|)} =: similarity_{A_{n+1},A}$ als Vergleichsindikator verwendet.

Konfidenz: Da $similarity_{(\cdot,\cdot)} \in [0, 1]$, eignet es sich gut als Konfidenzwert.

$$conf_{Input} := similarity$$

3.3.7 Vorhersage mittels TensorFlow

Die aufgezeichneten Aktionen werden nur selten ausreichen, um für beliebige Paare (DM_A, DM_B) von Datenmengen einzuschätzen, wie wahrscheinlich es ist, dass DM_B nach DM_A auftritt. Um für Aktionen, für die noch kein Nachfolger beobachtet wurde, eine Vorhersage treffen zu können, soll TensorFlow zu Hilfe genommen werden. Ziel war es dabei, *GFLasso* (siehe 2.2.6) zu implementieren. Für die einzelnen Terme der Zielfunktion wurden die nun folgenden Überlegungen angestellt.

3.3.7.1 Korrelation zwischen Datenmengen

Dieses Maß ist aus einem Irrtum entstanden: die Korrelationsmatrix A sollte nicht etwa den Zusammenhang zwischen Datenmengen, sondern zwischen *Merkmalen* beschreiben. Da das Maß später dennoch verwendet wurde (siehe 3.3.6.2), wird es hier dennoch aufgeführt.

Die Korrelation $corr_{(\cdot,\cdot)}$ zweier Datenmengen DM_A, DM_B sollte mit einer Zahl im Intervall $[-1, 1]$ angeben, wie wahrscheinlich es ist, dass DM_B nach DM_A auftritt. Also:

$$corr(DM_A, DM_B) = \begin{cases} 1 - \delta \\ 0 \\ -1 + \delta \end{cases} \implies \text{Dass } DM_B \text{ nach } DM_A \text{ auftritt, ist...}$$

$$\begin{cases} \text{mit großem Abstand wahrscheinlicher als alle Alternativen,} \\ \text{ähnlich wahrscheinlich wie die anderen bekannten Nachfolger von } DM_A, \\ \text{bisher im Vergleich zu den Alternativen äußerst selten vorgekommen.} \end{cases}$$

Um diesen Wert zu berechnen, liegt der Unterbaum von DM_A aus dem Modell vor, aus dem abgelesen werden kann, wie oft DM_A und alle Nachfolger aufgerufen worden sind. Betrachten wir zunächst:

$$rating(DM_A, DM_B) = \frac{DM_B.count \cdot \#succ}{DM_A.count} - 1$$

Für einen Nachfolger DM_B aus der Menge aller Nachfolger $succ$, für den gilt: $DM_B.count = 0$, ist die Bewertung -1, für „durchschnittlich häufige“ DM_B, also $DM_B.count = \frac{DM_A.count}{\#succ}$, ist die Bewertung 0; insoweit sind die Kriterien für $corr$ erfüllt. Für große $DM_B.count$ übersteigt die Bewertung jedoch den Wert 1 möglicherweise. Die rechnerisch größtmögliche Bewertung ergibt sich, wenn $DM_A.count$ groß, $DM_B.count$ annähernd so groß wie $DM_A.count$, $\#succ$ klein, zu $\#succ - 1$.

Deshalb wurde in einem zweiten Schritt diese Bewertung so transformiert, dass die Abbildung auf -1 und 0 erhalten bleiben, aber $\#succ - 1$ auf 1 abgebildet wird.

Das wurde mit einer Potenzfunktion bewerkstelligt. Für die Basisfunktion $f_a(x) = x^a$ gilt für alle $a > 0$: $f_a(0) = 0, f_a(1) = 1$. f_a durchquert gewissermaßen den quadratischen Bereich zwischen $(0, 0)$ und $(1, 1)$ auf eine bestimme Weise, die durch a angepasst werden kann. Dieser quadratische Bereich wird nun auf den Bereich zwischen $(-1, -1)$ und $(\#succ - 1, 1)$ gedehnt:

$$f_a(x) = 2 \cdot \left(\frac{x+1}{\#succ}\right)^a - 1$$

Zum Schluss wird a so gewählt, dass $f_a(0) = 0$, nämlich nach trivialer Umformung $\hat{a} = log_{\#succ}2$. Das liefert uns das gewünschte Korrelationsmaß

$$corr := f_{\hat{a}} \circ rating$$

$$corr(DM_A, DM_B) = 2 \cdot \left(\frac{DM_B.count}{DM_A.count}\right)^{log_{\#succ}2} - 1$$

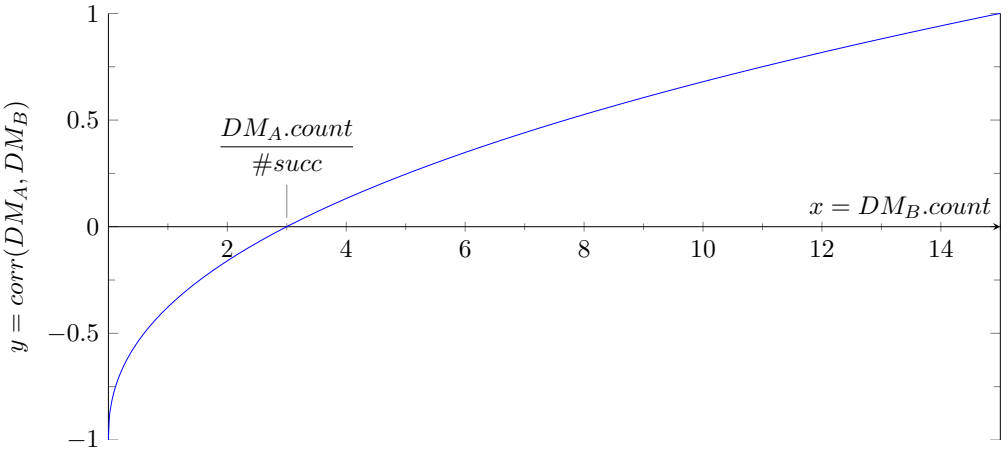


Abbildung 3.11: Schaubild für die Korrelation zweier Datenmengen, beispielhaft mit $DM_A.count = 15, \#succ = 5$

Zu beachten ist, dass im Gegensatz zu Korrelationskoeffizienten im klassischen mathematischen Sinne $corr$ nicht symmetrisch ist.

3.3.7.2 Matrixmultiplikation I

Der erste Ansatz war, die Vorhersage in einer Matrixmultiplikation xw = y umzusetzen, wobei x und y Tensordarstellungen für Datenmengen (siehe 3.3.5.2) sind, und w eine trainierbare Matrix. Für das Training sollten konkret alle Einträge des Modells als x und nur die jeweils absolut häufigste Nachfolgeaktion als y verwendet werden, um die Anzahl der Paare zu verringern und so das Training zu beschleunigen. Da w die passenden Dimensionen für alle x, y haben sollte, mussten die Dimensionen der Tensordarstellungen vereinheitlicht werden.

Sei im Folgenden $\hat{d} = \max_{t \in T} |t.dims|$ die größte Anzahl an Dimensionen aller bekannten Tabellen. Die größtmögliche Matrix, die eine Datenmenge kodiert, hat $\hat{d} + 1$ Zeilen[1] und $\hat{d} + 2$ Spalten. Somit werden alle Matrizen auf diese Dimensionen erweitert und die neuen Einträge mit -1 aufgefüllt. Die Dimensionen von w sind $\hat{d} + 2 \times \hat{d} + 2$.

Mit den Überlegungen in 2.2.2 wird jedoch schnell klar, dass die Matrix w ab einer gewissen Anzahl Paare x, y keine sinnvollen Ergebnisse mehr produzieren kann, da die Spalten von x verschiedene Bedeutungen haben. Da sie jeweils mit denselben Gewichtungsvektoren multipliziert werden, kann keine gültige oder sinnvolle Datenmatrix als Ergebnis erwartet werden.

3.3.7.3 Matrixmultiplikation II

Um die Dimension der Matrix w und somit ihre Fähigkeit, Zusammenhänge zu erlernen, zu vergrößern, wurden die Matrizen aus dem vorigen Schritt in Zeilenvektoren mit $\ell := (\hat{d} + 1) \cdot (\hat{d} + 2)$ Einträgen umgewandelt. Für eine gültige Matrixmultiplikation hat w nun ℓ Zeilen und Spalten.

Jeder Eintrag in x kann nun Einfluss auf jeden Eintrag in y nehmen, weshalb gegenüber 3.3.7.2 deutlich verbesserte Ergebnisse erwartet wurden. Für wenige Paare x, y hat diese Strategie sehr gute Ergebnisse gezeigt und auch gültige Vorhersagen gemacht. Bereits an einem Beispiel von zehn Datenpaaren ohne erkennbaren Zusammenhang ist sie jedoch gescheitert.

Wiederum aus den Vorüberlegungen 2.2.2 ist zu erkennen, dass diese Strategie nur funktionieren kann, wenn der Zusammenhang zwischen allen Paaren aufeinanderfolgender Datenmengen recht gleichförmig ist. Eine beliebige Abbildung zwischen Datenmengen kann dagegen nicht realisiert werden.

3.3.7.4 Zwischenfazit

Da sich alle bisherigen Versuche, TensorFlow zur Vorhersage einzusetzen, als wenig Erfolg versprechend herausgestellt haben, wurde auch die Implementierung von *GFLasso* verworfen. Da die Zusammenhänge der Gewichte im neuronalen Netzwerk komplexer sind als in den vorigen Versuchen, ist *GFLasso* nicht mehr anwendbar. Im Folgenden werden in TensorFlow standardmäßig verfügbare Maße verwendet. Als Verlustfunktion soll die *mittlere absolute Abweichung*, als Genauigkeitsmaß der *mittlere Anteil übereinstimmender Einträge* eingesetzt werden.

3.3.7.5 Einfaches neuronales Netzwerk

Um komplexere bzw. beliebige Zusammenhänge zwischen Nachfolgern zu trainieren, wurde schließlich ein neuronales Netz entworfen. Es wurde nach dem Vorbild von [Bro16] strukturiert, die Aktivierungsfunktionen wurden hingegen angepasst und die Anzahl der Knoten dynamisch in Abhängigkeit von $\hat{d}$ gewählt. Das neuronale Netz besteht aus drei Schichten:

[1] Damit bietet sie genug Einträge, um eine Datenoperation auf jeder einzelnen der $\hat{d}$ vielen Dimensionen zu kodieren.

1. Voll vernetzte Schicht mit $\frac{3}{2}\ell$ Knoten, Aktivierungsfunktion: *Rectifier*
2. Voll vernetzte Schicht mit ℓ Knoten, Aktivierungsfunktion: *Rectifier*
3. Voll vernetzte Schicht mit ℓ Knoten, Aktivierungsfunktion: *Identität*

Die insgesamt $\frac{7}{2}\ell$ Knoten haben je einen Gewichtungsfaktor für jeden Knoten in der vorherigen Schicht (im Fall der ersten Schicht: die Eingabe) sowie genau einen Verzerrungsskalar (*bias*). Das ergibt $4\ell^2 + \frac{7}{2}\ell$ Parameter, also ein Vielfaches mehr als in der vorigen Strategie. Diese Strategie ist nun als einzige der Verfahren, die TensorFlow nutzen, in der Schnittstelle verfügbar. Sie stellt damit die einzige Wahlmöglichkeit dar, wenn die gegebene Vorgängeraktion dem Modell nicht bekannt ist und kein bekannter Input gegeben worden ist. Sie wurde in der Evaluation ausführlich getestet, siehe Kapitel 4.

Kapitel 4

Evaluation

Die Evaluation soll zeigen, ob das System die Aufgabestellung bewältigen kann, für die es entworfen wurde. Da die Bayes'schen Vorhersagestrategien keine zufälligen Komponenten haben, sondern ihr Ergebnis bei gegebenen Parametern allein vom Zustand des Modells abhängig ist, wurde ausschließlich die Strategie, die auf einem neuronalen Netz basiert, einem umfangreichen Test unterzogen. Im Folgenden wird das Vorgehen bei der Evaluation des Systems erläutert, die Ergebnisse zusammengefasst und interpretiert. Die ausführlichen Ergebnisse aller Testläufe sind im Anhang 5.1 zu finden.

4.1 Beliebige Aktionsfolgen

In einem ersten Schritt wurde die Klasse TensorFlowPrediction auf beliebige Aktionsfolgen trainiert. Bei jedem Durchgang wurden Modellgrößen von $n = 10, 20, ..., 200$ in zufälliger Reihenfolge gewählt, das Modell mit je n zufälligen Datenmengen initialisiert und schließlich in Sätzen von 2500 Epochen solange trainiert, bis zehn Sätze in Folge keine neue höchste Vorhersagegenauigkeit erzielt wurde. Der genaue Ablauf wird in der Quelltextskizze 5.4.3 dargestellt. Wie zufällige Datenmengen konstruiert werden, ist in 5.4.1 nachzulesen.

Dieser Versuchsaufbau soll den *Worst Case* simulieren: Da es die Aufgabe des neuronalen Netzwerks ist, Zusammenhänge zwischen aufeinanderfolgenden Aktionen zu erlernen, ist anzunehmen, dass die Vorhersage dann am seltensten die Nachfolgeaktion korrekt schätzen kann, wenn die aufeinander folgenden Aktionen der Historie paarweise keinen Zusammenhang haben, sondern zufällig aneinandergereiht sind. Statt der Zusammenhänge könnte das Netzwerk also höchstens die Paarungen „auswendig“ lernen.
Das Ergebnis soll zeigen, ob und in welchem Maße das Modell dazu in der Lage ist, unzusammenhängende Paarungen zu erlernen. Es stellt die Grundlage für die zweite Evaluation dar, in der untersucht wurde, ob das Modell auf authentischen (bzw. jedenfalls nicht zufälligen) Aktionsfolgen eine bessere Leistung zeigen kann.

4.1.1 Ergebnis

Abbildung 4.1 zeigt die Ergebnisse der zehn Testläufe. Trotz der großen Rolle der Zufälligkeit ist ein Trend zu erkennen: Die Vorhersagegenauigkeit $acc(n)$ ist mit $r = -0.8086$ mit

der Anzahl der Aktionen n korreliert. Die Linie zeigt eine logarithmische Regressionskurve, die gewählt wurde, weil sie aus allen klassischen Verfahren die höchsten Korrelation zwischen n und $acc(n)$ erreicht.
Für fast alle $n \geq 100$ erreicht $acc(n)$ im Mittel nicht 50 Prozent, stagniert aber ab $n = 120$ bei Werten um 40 Prozent.

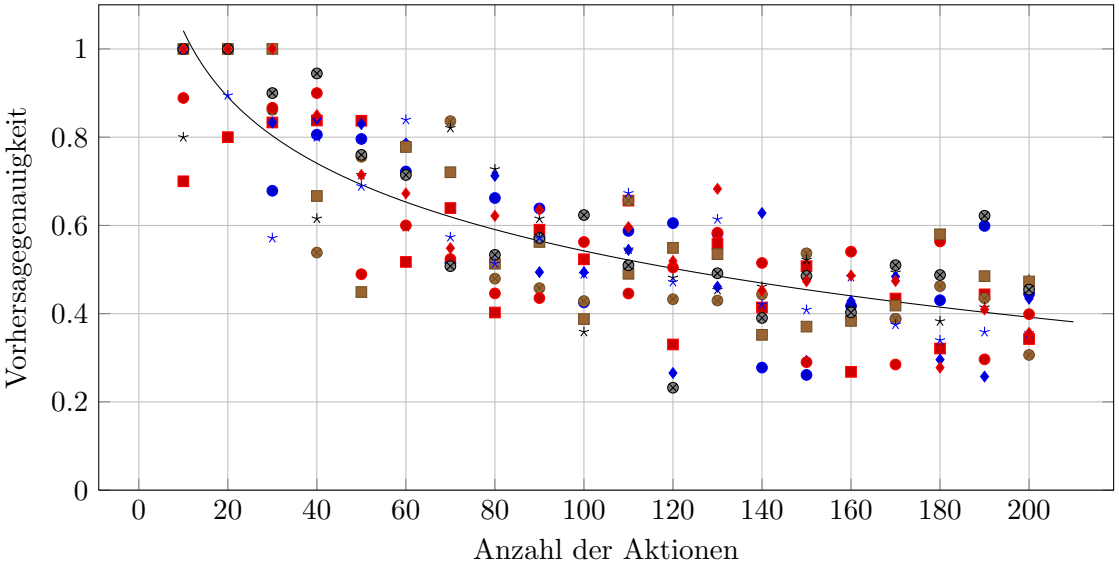


Abbildung 4.1: Die Vorhersagegenauigkeit der Strategie TensorFlowPrediction bei zufälligen Datenfolgen, abgetragen gegen die Modellgrößen.

4.2 Simulation menschlichen Nutzungsverhaltens: Zusammenhängende Aktionsfolgen

Nun sollte das System an Datenfolgen getestet werden, die menschlichem Verhalten näher kommen. Es war nicht realistisch, im Rahmen einer Nutzerstudie eine Menge echter Nutzungsdaten zu erheben, da die Daten der Datenbank für die Probanden nicht von Natur aus relevant sind. Die Aufgabestellung, sich einige Datenmengen angeben zu lassen, hätte demnach wenig Ähnlichkeit mit natürlicher, sinnvoller Nutzung eines Klientsystems gehabt. Da die Nutzer außerdem vermutlich dieselbe Aktion nur höchstens wenige Male durchgeführt hätten, hätte man nicht von erkennbaren Verhaltensmustern sprechen können, und es wäre wohl auch kein befriedigendes Vorhersageergebnis zustande gekommen. Stattdessen wurden Aktionsfolgen mit folgenden Annahmen generiert:

- Nutzeraktionen sind in kurzen Folgen zusammenhängend, etwa wie bei typischem Verhalten im Browser.
- Der Zusammenhang von Aktionen kann als Distanz der Tensordarstellung der zu den Aktionen gehörenden Datenmengen verstanden werden.

Die Länge der zusammenhängenden Aktionsfolgen $A_i = (a_{i0}, a_{i1}, ...)$ wurde normalverteilt generiert: $|A_i| \sim \mathcal{N}(6, 2)$, als Distanzfunktion wurde $loss_2$ gewählt (siehe 2.2.6), also in diesem Fall die Summe der quadrierten eintragsweisen Differenzen.
Als maximale Distanz zweier Datenmengen, die als zusammenhängend gelten sollen, wurde 250 festgelegt. Bei einer Messung mit zufälligen Datenpaaren haben durchschnittlich 29 Prozent diese Schwelle unterschritten (siehe 5.2.2).

4.2.1 Ergebnis

Mit den Erkenntnissen des ersten Durchgangs wurde der Test erneut zehn Mal durchlaufen. Im Schaubild 4.2 ist deutlich zu erkennen, dass die Messreihen deutlich langsamer abfallen und die Stagnation schneller eintritt als im ersten Durchgang. Die Werte sind weniger gestreut, für ein n bewegen sich die $acc(n)$ also in kleineren Intervallen als in 4.1. Bei $n = 200$ ist die Bewertung im Mittel um rund 78 Prozent besser.

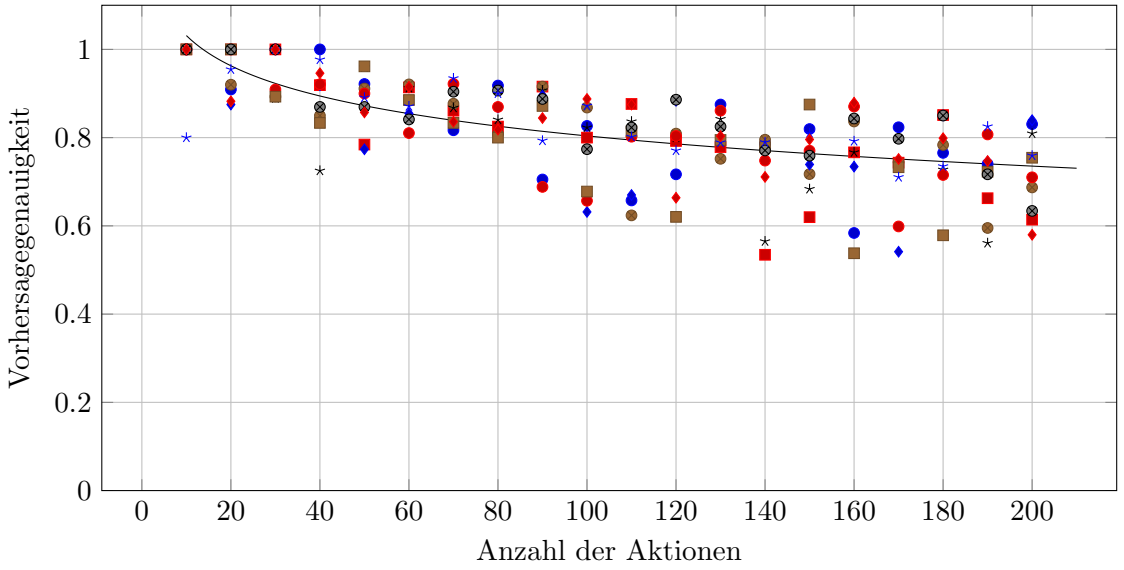


Abbildung 4.2: Die Vorhersagegenauigkeit der Strategie TensorFlowPrediction bei zusammenhängenden Datenfolgen, abgetragen gegen die Modellgrößen.

4.3 Interpretation der Ergebnisse

Vergleich der Durchgänge, Aussagekraft der Regressionskurve

Den logarithmischen Regressionskurven für Evalutation 1 $reg_1(n)$ bzw. 2 $reg_2(n)$ zufolge ist bei den zusammenhängenden Daten bereits ab $n = 105$ ein um die Hälfte besseres Ergebnis im Vergleich zu beliebigen Daten zu erwarten, ab $n = 232$ ein um den Faktor 2 besseres Ergebnis. Für Werte $n > 200$ können die Werte der Kurven nur noch bedingt als aussagekräfig interpretiert werden: $reg_1(n)$ nimmt für $n > 1220$ negative Werte an ($reg_2(n)$: für $n > 342919$), während für die Messwerte $acc(n) \geq 0$ stets gilt. Der oben verwendete „Faktor“ $\frac{reg_2(n)}{reg_1(n)} - 1$ nimmt somit unglaubwürdig hohe und schließlich negative Werte an.

Fällt $acc(n)$ für große n auf 0 ab?

Es ist nicht zu erwarten, dass die wahren Werte $acc(n)$ für große Werte n und authentischen Aktionshistorien gegen 0 streben. Da die Tensordarstellungen von Datenmengen 3.3.5.2 derart konstruiert sind, dass viele Einträge den Wert -1 annehmen, könnte das neuronale Netz, wenn es aufgrund zu komplexer Datenzusammenhänge keine zufriedenstellenden Vorhersagen mehr treffen kann, darauf zurückfallen, viele Werte -1 zu erzeugen. So würde im Mittel das Genauigkeitsmaß dennoch ein gewisses Niveau nicht unterschreiten. Nichtsdestoweniger wären die konkreten Aktionsvorschläge somit nicht mehr allzu hilfreich.
Ein Umstand, der dazu betragen könnte, dass die Genauigkeit nicht beliebig kleine Werte annimmt, wäre, dass sich die Komplexität authentischer Nutzeranfragen in Grenzen hält. Dies kann an dieser Stelle nur vermutet und nicht bewiesen werden. Eine Anfrage in natürlicher Sprache zu formulieren, die mehr als zwei Datenoperationen enthält, ist bereits

derart umständlich[1], dass die Sprachverarbeitung des Klientsystems damit Schwierigkeiten haben könnte, wenn sie nicht zulässt, dass die Eingabe in mehreren Schritten erfolgt (*„Zeige die Anzahl männlicher Einwohner aller EU-Mitgliedsstaaten,* und davon *nur die größte Altersgruppe,* und davon *den Mittelwert über...“*). Diese Annahme wird in 5.1.1 nochmals aufgegriffen und ist auch in die Wahl der Parameter eingeflossen, die die Anzahl der Spezifikationen bzw. Datenoperationen bei der Konstruktion zufälliger Daten steuern, siehe 5.4.1.

Bestmögliche Leistung für gegebenes n

Die Ergebnisse der Evaluation zeigen lediglich einen Zwischenstand des Trainingsfortschritts. Ungünstige Gebiete im Parameterraum, wie in 2.2.7 dargestellt, können verhindern, dass innerhalb von zehn Iterationen wie gefordert ein verbesserter Höchstwert für $acc(n)$ erzielt wird, was im gewählten Trainingsablauf zum Abbruch des Trainings führt. In realen Anwendungsfällen könnte es sich also lohnen, das Training fortzusetzen.

Leistung für große Modellgrößen

Um einen Eindruck davon zu bekommen, welche Genauigkeit das System für Modellgrößen $n > 200$ erreichen kann, wurde der Test aus Evaluation 1 und 2 mit $n = 2000$ ausgeführt. Wegen der enormen Laufzeit von mehr als vier Stunden wurde der Test jedoch nur einmal (und nicht zehn Mal) durchlaufen, und ist damit nicht Teil der repräsentativen Ergebnisse. Dennoch seien sie hier erwähnt:

- Zufällige Aktionsfolgen: $acc(2000) = 0.26133019$, Laufzeit: 4:39 Stunden
- Zusammenhängende Aktionsfolgen: $acc(2000) = 0.54164206$, Laufzeit: 4:10 Stunden

Fazit der Evaluation

Abschließend die Frage: Was sagen die Werte $acc(n)$ genau über das System aus? $acc(n)$ beschreibt, wie gut das neuronale Netz die (bereits vorher bekannten) Trainingsdaten replizieren kann, nachdem es in für die Evaluation angemessenem Umfang trainiert worden ist. Falls der Nutzer eine unbekannte Eingabe macht, kann aktuell keine Aussage darüber getroffen werden, wie sinnvoll oder hilfreich die durch das neuronale Netz vorgeschlagene Aktion für den Nutzer ist.

4.3.1 Verzerrung früherer Ergebnisse

Bei der ersten Durchführung der zweiten Evaluation war zu erkennen, dass die Durchgänge 4 und 7 auffällig viele im Vergleich schlechte Werte hervorgebracht haben als die anderen acht. Bei einem verkürzten Test wurde festgestellt, dass die Leistungsfähigkeit des Modells stark abnimmt, wenn nicht jeder Testlauf in einem eigenen Python-Prozess abläuft. Daraufhin wurden beide Evaluationen erneut durchgeführt. Die jeweils zweiten Durchläufe hatten eine deutlich kleinere mittlere Abweichung beziehungsweise eine deutlich größere Korrelation, weshalb sie als getreueres Abbild der Leistungsfähigkeit des Modells eingestuft wurden. Bei den in 4.1.1 und 4.2.1 vorgestellten Ergebnissen handelt es sich dementsprechend um die zweiten Durchläufe. Die Ergebnisse der ersten Durchläufe sind jedoch ebenfalls im Anhang abgedruckt 5.1. Wie vorgegangen wurde, um das Verhalten der Testläufe in demselben Prozess mit dem in einzelnen Prozessen zu vergleichen, sowie die Ergebnisse sind in 5.2.1 näher erläutert.

[1] Erschwerend kommt es dazu, wenn Nutzer nicht dazu ausgebildet sind, solche Anfragen präzise und strukturiert zu formulieren. Die Grenze zum Diktat eines Programms statt natürlichsprachlicher Programmierung würde hier fast erreicht.

Kapitel 5

Fazit

In dieser Arbeit wurde ein System vorgestellt, das Klientsysteme dazu befähigen soll, dem Nutzer eine Aktion vorzuschlagen, die er zu einem Zeitpunkt in seinem Ablauf typischerweise auswählt. Falls dem System keine solche Aktion bekannt ist, kann durch die Vorhersagestrategie, die sich ein künstliches neuronales Netz zunutze macht, eine Aktion ermittelt werden, die dem Nutzungsverhalten des Nutzers entsprechen könnte, obwohl er sie bisher noch nie verwendet hat. Während die Vorhersagestrategien, die allein die Zählerwerte im Modell zur Vorhersage verwenden, gewissermaßen deterministisch sind, da sie bei demselben Zustand des Modells und derselben Vorgängeraktion stets dieselbe Nachfolgeaktion ermitteln, ist die Vorhersage durch das neuronale Netzwerk an seinen Trainingsfortschritt, also seine Gewichte geknüpft und wurde daher ausführlich getestet. Die Evaluation hat gezeigt, dass das neuronale Netz dazu in der Lage ist, das Nutzungsverhalten des Nutzers zu erlernen, und dabei eine wesentlich bessere Leistung zeigt, wenn die Aktionen und die jeweiligen Nachfolgeaktionen sich bezüglich ihrer Tensordarstellung recht ähnlich sind.

5.1 Ansätze zur Weiterentwicklung

Einige Ideen, die bei der Entwicklung des Systems entstanden sind, konnten nicht mehr aufgegriffen werden, könnten aber Ansätze zur Weiterentwicklung des Systems oder Anhaltspunkte zum Entwurf eines Klientsystems liefern.

5.1.1 Automatische Generierung von Fragestellungen zu Datenmengen

Sollen die Vorhersagen des Systems in einem Klientsystem dazu genutzt werden, dem Nutzer eine Aktion vorzuschlagen, wird es wohl nicht genügen, dem Nutzer die Kodierung einer Datenmenge oder die bereits aus der Tabelle extrahierte Ergebnisdatenmenge vorzulegen. Für einen Vorschlag wäre interessant, ob es möglich ist, aus der Semantik der Tabelle, etwa „Einwohnerzahl nach EU-Mitgliedsstaat, Alter, Geschlecht und Jahr“ und einer kodierten Datenmenge, etwa *Eurostat:demografie/(geo=DE, age=Y25, sex=F, time=2015)*, eine passende Fragestellung *„Wieviele 25-jährigen, weiblichen Einwohner hat Deutschland?“* zu generieren. Das stellt sogar für einen Menschen eine Herausforderung dar, wenn die Datenmenge mehrere Datenoperationen enthält.

Für Aktionen, die bereits im Modell (als Folgeaktion) hinterlegt sind, würden vorerst die gespeicherten Fragestellungen des Nutzers genügen, die auch von großem Nutzen sind, wenn das Modell hinreichend groß ist.

5.1.2 n-Grammanalyse

Die implementierten Datenstrukturen sehen keine Verallgemeinerung von Bigrammen auf n-Gramme vor. Für Verhaltensmuster der Form $A, B, C, B, A, B, C, B...$ würden die aktuellen Strategien scheitern (wenn nicht eine Nutzereingabe vorliegt), da zur Vorhersage die Kenntnis der letzten zwei Aktionen nötig ist. Zeigt sich, dass solche und für $n > 2$ analoge Verhaltensmuster häufig Teil authentischen Nutzerverhaltens ist, wäre es sinnvoll, die Auswertung von mehr als einer vorangegangenen Aktion zu unterstützen.

5.1.3 Kombination der Vorhersagestrategien

Die Vorhersagestrategien sind aktuell nicht so entworfen, dass für eine beliebige gegebene Datenmenge eine Konfidenz ermittelt werden kann, sondern nur im Zuge der Vorhersage für die letztendlich vorhergesagte Datenmenge. Wäre das möglich, so könnten die Vorhersageergebnisse jeder einzelnen Strategie mit den anderen Strategien bewertet werden, um aus einer gewichteten Kombination der Bewertungen die beste Vorhersage zu ermitteln. Konkrete Anwendungsfälle könnten auch nahelegen, diejenige Aktion zu wählen, die von den meisten der bestehenden Vorhersagestrategien als der wahrscheinlichste Nachfolger ausgewählt wird. Geradezu ein Baukastensystem zur Kombination von Vorhersagestrategien wäre als Teil dieses Systems oder auch eines möglichen Klientsystems denkbar, das womöglich selbst noch weitere Kriterien heranzieht, um eine Vorhersage zu entwickeln.

5.1.4 Erweiterung der Datenoperationen

Die Datenoperationen können nur eine beschränkte Menge Nutzeranfragen abbilden. Je nach Anwendungsfall könnten seitens des Klientsystems weitere Datenoperationen implementiert werden, denkbar wäre ein Typ MapOperation, der eine Abbildung realisiert, zu der jedoch eine Methode eingegeben werden müsste, sowie ein Typ AbstractOperation, der es ermöglichte, einen durch eine SelectOperation ausgewählten Dimensionswert in einer anderen Anfrage zu verwenden. Ein Beispiel wäre die Anfrage:

„Gib für den Mitgliedsstaat der EU mit der größten Einwohnerzahl dessen Anteil an der Gesamtfläche an.“

mit der möglichen erweiterten Datenkodierung:

```
Eurostat:demografie/max(geo)->abstract(geo)
->lookup(Eurostat:area/map(percentage(geo)))
```

Mit zunehmender Vielfalt möglicher Befehle und Erweiterung der Syntax stößt die Datenkodierung jedoch an ihre Grenzen, die ursprünglich Menschenlesbarkeit zum Ziel hatte.

5.1.5 Vorverarbeitung der Tabellen

Wie in 3.2.2.2 geschildert, ist die Extraktion von Daten aus Tabellen zwar im System integriert, aber bisher nicht über die Schnittstelle ansteuerbar. Da in den Datensätzen von Eurostat die Daten nämlich nicht überschneidungsfrei sind, sondern verschiedene Gruppierungen ebenfalls erfasst werden, wie etwa der Europäische Wirtschaftsraum *EEA31*, werden diese z.B. auch in die Berechnung der durchschnittlichen Einwohnerzahl der aufgeführten Regionen miteinbezogen und verfälschen diese. Würden die Regionen etwa als Objekte mit Relationen [Fuc17] modelliert, wäre es möglich, die korrekten Werte auszugeben – vorausgesetzt, die Werte für alle Regionen sind verfügbar. Alternativ könnten die

Tabellen automatisch um solche Überschneidungen bereinigt werden, wenn bekannt ist, welche Dimensionswerte welcher Granularität entsprechen (Zusammenfassung von Mitgliedsstaaten bzw. einzelner Mitgliedsstaat (NUTS 0) bzw. größere Regionen (NUTS 1), ...).
Grundsätzlich soll jedoch weiterhin das Klientsystem dafür Sorge tragen, dass die Tabellen für die Verarbeitung geeignet sind, sofern die Funktionen zur Verarbeitung eingesetzt werden sollen, die dieses System bereitstellt, siehe 1.2.

Literaturverzeichnis

[Bro16] Brownlee, Jason: *Develop Your First Neural Network in Python With Keras Step-By-Step*. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/, 2016. – Hochgeladen am 24. Mai 2016, abgerufen am 24. Februar 2019

[ES] Statistisches Amt der Europäischen Union: *Eurostat - Ihr Schlüssel zur europäischen Statistik*. https://ec.europa.eu/eurostat/, . – Abgerufen am 09. März 2019

[Fis36] Fisher, Ronald A.: The Use of Multiple Measurements in Taxonomic Problems. 7 (1936), Sept., S. 179–188

[FRK12] Fan, Yue ; Raphael, Louise ; Kon, Mark: Feature vector regularization in machine learning. (2012). https://arxiv.org/abs/1212.4569

[Fuc17] Fuchß, Dominik T.: *Verarbeitung natürlich-sprachlicher Beziehungsbeschreibungen zwischen Objekten*, Karlsruher Institut für Technologie (KIT) - IPD Tichy, Bachelorarbeit, 2017

[KST12] Kato, Makoto P. ; Sakai, Tetsuya ; Tanaka, Katsumi: Structured Query Suggestion for Specialization and Parallel Movement: Effect on Search Behaviors. In: *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012, S. 389–398

[LCW+10] Li, Jundong ; Cheng, Kewei ; Wang, Suhang ; Morstatter, Fred ; Trevino, Robert P. ; Tang, Jiliang ; Liu, Huan: Feature Selection: A Data Perspective. 9 (2010), März, Nr. 4. – Artikel 39

[Mad16] Madigan, James: *Getting Gamers: The Psychology of Video Games and Their Impact on the People who Play Them*. Rowman & Littlefield, 2016. – ISBN 978–1–4422–3999–9

[McG04] McGraw, Gary: Software Security. 2 (2004), März-April, 80 - 83. https://www.garymcgraw.com/wp-content/uploads/2015/11/bsi1-swsec.pdf

[Pok] The Pokémon Company: *Business Summary - Pokémon in Figures*. https://www.pokemon.co.jp/corporate/en/services/, . – Abgerufen am 10. März 2019

[Rud17] Ruder, Sebastian: An overview of gradient descent optimization algorithms. (2017). https://arxiv.org/abs/1609.04747v2. – Version 2, 2017

[Rü14] Rüschendorf, Ludger: *Mathematische Statistik*. Springer-Verlag, 2014. – ISBN 978–3–642–41996–6

[Sch17] Schulz, Sophie: *Erweiterung der Sprachdomäne durch das Erlernen von Synonymen basierend auf dem Dialogsystem JustLingo*, Karlsruher Institut für Technologie (KIT) - IPD Tichy, Bachelorarbeit, 2017

[SDB18] Sifa, Rafet ; Drachen, Anders ; Bauckhage, Christian: Profiling in Games: Understanding Behaviour from Telemetry. (2018)

[SLBP18] Sadowski, Gorka ; Litan, Avivah ; Bussa, Toby ; Phillips, Tricia: Market Guide for User and Entity Behavior Analytics / Gartner, Inc. 2018. – Forschungsbericht

[Wai] Waibel, Alexander: *Kognitive Systeme, Vorlesung 06: Machine Learning*

[WWS04] Winkler, Daniel ; Westphal, Markus ; Senger, Christian: *Höhere Mathematik für Informatiker*. Version: 2004. `http://danielwinkler.de/hm/`. – HM2 Revision 291

[Zhu01] Zhu, Hong: Formal Specification of Agent Behaviour through Environment Scenarios. In: Rash, James L. (Hrsg.) ; Rouff, Christopher A. (Hrsg.) ; Truszkowski, Walt (Hrsg.) ; Gordon, Diana (Hrsg.) ; Hinchey, Michael G. (Hrsg.): *Formal Approaches to Agent-Based Systems: First International Workshop April 2000 - Lecture Notes in Artificial Intelligence* Bd. 1871, Springer-Verlag Berlin Heidelberg, 2001, S. 263–276

Anhang

Es folgen detailliertere Informationen aus verschiedenen Bereichen der Entwicklung des vorliegenden Systems. Zunächst sind die vollständigen Ergebnisse der Evaluationstestläufe angegeben, dann folgen einige Auswertungen, aufgrund derer Entscheidungen über den Entwurf und zum weiteren Vorgehen getroffen worden sind. Abschließend sind Quelltextskizzen angegeben, die verschiedene Abläufe innerhalb des Programms erläutern sollen.

5.1 Ergebnisse der Evaluation

5.1.1 Beliebige Aktionsfolgen: Durchgang 1

Achtung, die gültigen Ergebnisse der Evaluation sind die der *Durchgänge 2*, nicht die auf dieser Seite, siehe 4.3.1.

n	1	2	3	8	10	4A	5A	6A	7A	9A
10	1.00000000*	1	0.7	0.69999999**	1	1	1	0.89999998	1	1.00000000
20	1.00000000*	0.95	1	1	0.8947368	0.85000002	0.80000001	0.69999999**	1	0.80000000
30	0.8	0.79310346	0.65517240**	0.81481481	1.00000000*	0.82758623	1.00000000*	0.79310346	0.9285714	0.86206895
40	0.8717949	0.8	0.9	0.875	0.7222222	0.78378379	0.68421053	0.775	0.92105263*	0.30000000**
50	0.7708333	0.79166670*	0.72	0.46938776**	0.5	0.73999999	0.72000001	0.75510205	0.71428572	0.72727275
60	0.71428573	0.6440678	0.7222222	0.82758621*	0.5535714	0.28571428**	0.5762712	0.80000001	0.67924529	0.69642860
70	0.43478260**	0.5970149	0.80303030*	0.58064516	0.6615385	0.64705882	0.5942029	0.796875	0.62318841	0.71875000
80	0.5584416	0.5375	0.29870130**	0.50632912	0.6363636	0.51282051	0.73333334*	0.46835443	0.52631579	0.43421054
90	0.5121951	0.5731707	0.5421687	0.5595238	0.5730337	0.42045455	0.38372093**	0.49382717	0.75581396*	0.59036140
100	0.4516129	0.36458334	0.58762884	0.47872341	0.5494506	0.51136363	0.51578947	0.62499999*	0.40425532	0.25263157**
110	0.27184466	0.3611111	0.27000000**	0.43	0.63157890*	0.47524753	0.60606061	0.59803922	0.60784314	0.40196080
120	0.37383178	0.39130434	0.63302750*	0.24107143**	0.57009345	0.55652174	0.3	0.43478261	0.4375	0.45045045
130	0.4919355	0.48333332	0.38400000**	0.50806452	0.40983605	0.5338983	0.40983607	0.52941177	0.58119659*	0.44736840
140	0.26923078**	0.62121210*	0.4274809	0.55284552	0.5112782	0.32330827	0.41538462	0.42647059	0.47580645	0.39062500
150	0.41095892	0.54347825*	0.16666667**	0.47368421	0.51428574	0.33823529	0.37588653	0.45652174	0.25179856	0.27941176
160	0.34228188	0.27516780**	0.50993377	0.27777778	0.4054054	0.39583333	0.52941177	0.43506494	0.54421769*	0.44680852
170	0.38961038	0.41401273	0.36774194	0.28571429**	0.57861640*	0.4025974	0.48666667	0.44585987	0.41025641	0.54362416
180	0.5411765	0.41764706	0.21951220**	0.40490798	0.34355828	0.33532934	0.5	0.41040462	0.31578947	0.55900620*
190	0.39181286	0.36516854	0.43820226	0.4047619	0.30113637**	0.40782123	0.40571429	0.47159091*	0.3220339	0.40935670
200	0.31491712	0.29508197	0.32972974	0.26630435	0.31491712	0.43575419	0.36781609	0.20555556**	0.2688172	0.51351350*

Die jeweils besten Ergebnisse einer Zeile sind mit *, die schlechtesten mit ** markiert. Durchgänge mit A wurden mit Anaconda Python durchgeführt, die übrigen mit CPython.

Einige statistische Kennzahlen:

Logarithmische Regression: $acc(n) = A + B \cdot ln(x)$ mit $A = 1.5276627 37, B = -0.21959933$

$r(n, acc(n)) = -0.79801870 3$

$r(ln(n), acc(n)) = -0.83488503 9$

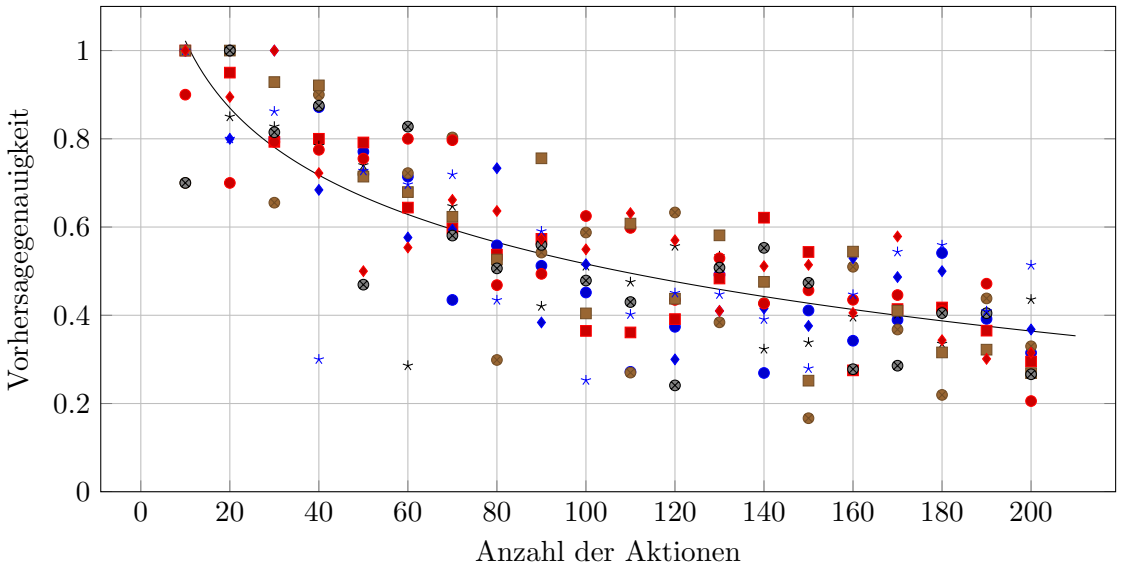


Abbildung 5.1: Die durch ungleiche Ausgangsbedingungen verzerrte Vorhersagegenauigkeit der Strategie TensorFlowPrediction bei zufälligen Datenfolgen, abgetragen gegen die Modellgrößen.

5.1.2 Zusammenhängende Aktionsfolgen: Durchgang 1

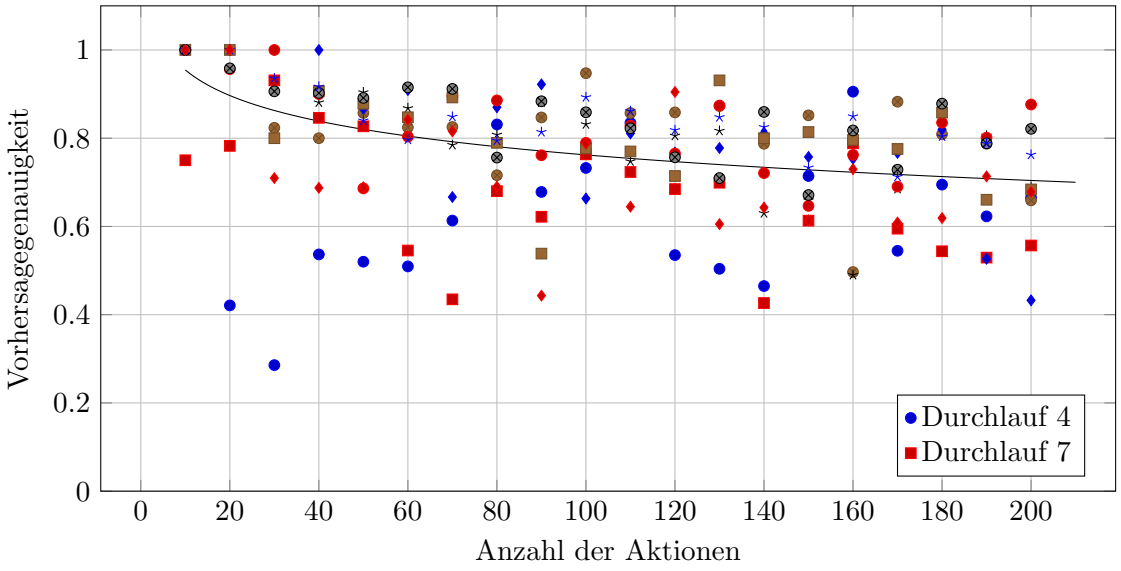


Abbildung 5.2: Die durch ungleiche Ausgangsbedingungen verzerrte Vorhersagegenauigkeit der Strategie TensorFlowPrediction bei zusammenhängenden Datenfolgen, abgetragen gegen die Modellgrößen.

Achtung, die gültigen Ergebnisse der Evaluation sind die der *Durchgänge 2*, nicht die auf dieser Seite, siehe 4.3.1.

n	0	1	2	3	4	5	6	7	8	9
10	1.00000000*	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	0.75000000**	1.00000000	1.00000000
20	1.00000000*	1.00000000	1.00000000	0.95652175	0.42105263**	1.00000000	0.95833331	0.78260869	1.00000000	1
30	0.82352941	1.00000000*	1.00000000	1.00000000	0.28571429**	0.80000000	0.90625000	0.93103451	0.93548387	0.709677398
40	0.80000000	0.88095238	1.00000000*	0.90000000	0.53658537**	0.90697675	0.90243903	0.84615385	0.91666667	0.6875
50	0.85714286	0.90384615*	0.86666667	0.68627452	0.52000001**	0.87804878	0.89130435	0.82692309	0.84000000	0.688888897
60	0.82456140	0.86792453	0.90909090	0.80357143	0.50943397**	0.84745764	0.91525425*	0.54545454	0.79661018	0.842105268
70	0.82539683	0.78461538	0.66666667	0.89552239	0.61333334	0.89230769	0.91176471*	0.43478261**	0.84848485	0.815384615
80	0.71604939	0.80769231	0.87012987	0.88571429*	0.83116884	0.78947368	0.75609756	0.68000000**	0.79452055	0.690140848
90	0.84705882	0.88095238	0.92207792*	0.76136364	0.67816093	0.53846155	0.88372094	0.62195122	0.81395350	0.443181815**
100	0.94736842*	0.83146068	0.66336634**	0.78947369	0.73255814	0.77659575	0.85869564	0.76344087	0.89285715	0.787234037
110	0.85714286	0.74757282	0.81081081	0.82857143	0.83495146	0.77000000	0.82242991	0.72340426	0.86138614*	0.644859816**
120	0.85849057	0.80555556	0.76724138	0.76521739	0.53508772**	0.71428572	0.75675676	0.68468469	0.81818182	0.904761905*
130	0.87288135	0.81666666	0.77777778	0.87401574	0.50400000**	0.93103448*	0.70967742	0.69911505	0.84761905	0.605263155
140	0.78740157	0.62992126	0.81060606	0.72093023	0.46491228	0.80000001	0.85950413*	0.42635659**	0.82500000	0.642857139
150	0.85185185*	0.65891473	0.75757576	0.64661654	0.71428572	0.81395349	0.67123288	0.61313869**	0.73333333	0.613636364
160	0.49664430	0.48993289**	0.75342466	0.76223776	0.90540540*	0.79605263	0.81756756	0.78846154	0.84931507	0.72972973
170	0.88275862*	0.68484848	0.76666666	0.69032258	0.54482759**	0.77564103	0.72857143	0.59493671	0.71250000	0.608974361
180	0.80921052	0.85526316	0.81987578	0.83522727	0.69480519	0.85806451	0.87837838*	0.54375000**	0.80392157	0.619047616
190	0.78750000	0.80571429*	0.52662722**	0.80120482	0.62285714	0.66049383	0.78846154	0.52941176	0.78735632	0.713483147
200	0.65909091	0.68965517	0.43258427**	0.87634408*	0.66486486	0.68390805	0.82122905	0.55681818	0.76243094	0.678160918

Die jeweils besten Ergebnisse einer Zeile sind mit *, die schlechtesten mit ** markiert. Alle Durchgänge wurden mit Anaconda Python durchgeführt.
Einige statistische Kennzahlen:
Logarithmische Regression: $acc(n) = A + B \cdot ln(x)$ mit $A = 1.147111845, B = -0.083592442$
$r(n, acc(n)) = -0.421359899$
$r(ln(n), acc(n)) = -0.465856225$

5.1.3 Zufällige Aktionsfolgen: Durchgang 2

n	10	11	12	13	14	15	16	17	18	19
10	1.00000000*	0.69999999**	1.00000000	0.80000001	1.00000000	0.88888890	1.00000000	1.00000000	1.00000000	1.00000000
20	1.00000000*	0.80000001**	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	1.00000000	0.89473683	1.00000000
30	0.67857140	0.83333331	0.86206895	0.86206895	0.83333331	0.86666667	1.00000000*	0.89999998	0.57142860**	1.00000000
40	0.80555556	0.83783784	0.53846154**	0.61538462	0.84210526	0.90000000	0.66666667	0.94444444*	0.80000000	0.85000000
50	0.79591837	0.83720930*	0.75555556	0.71428572	0.82978724	0.48936170	0.44897959**	0.76000001	0.68888889	0.71428572
60	0.72222221	0.51724139**	0.78181817	0.59649123	0.78571429	0.59999999	0.77777778	0.71428571	0.83928571*	0.67241380
70	0.51562500	0.63934427	0.83636362*	0.82089552	0.52459016	0.52380952	0.72058824	0.50769231**	0.57352941	0.54838709
80	0.66216217	0.40259740**	0.47945205	0.72727273*	0.71232877	0.44594595	0.51315789	0.53333334	0.51351352	0.62162162
90	0.63855422*	0.59036145	0.45783133	0.61538462	0.49411765	0.43529412**	0.56250000	0.57142858	0.57142857	0.63529412
100	00.42553191	0.52325581	0.42857142	0.35869565**	0.49397590	0.56250000	0.38775510	0.62352942*	0.48958333	0.52222223
110	0.58762887	0.65625000	0.65656566	0.54545455	0.54455446	0.44565217**	0.49038462	0.50961538	0.67307692*	0.59615385
120	0.60526316*	0.33018868	0.43269231	0.48148148	0.26548673	0.50476191	0.54901961	0.23214286**	0.47169811	0.51923077
130	0.58119659	0.55855856	0.42982456**	0.45378151	0.46086957	0.58333333	0.53508772	0.49152543	0.61403509	0.68292684*
140	0.27777778**	0.41406250	0.44360902	0.46093750	0.62809918*	0.51492537	0.35200000	0.39024390	0.42105263	0.45238095
150	0.26119403**	0.50757576	0.53676471*	0.52173913	0.29285714	0.29007634	0.37062937	0.48550725	0.40875912	0.47368421
160	0.41721854	0.26811594**	0.38926175	0.41496599	0.42857143	0.54054054*	0.38356164	0.40277778	0.48299320	0.48630137
170	0.38815789	0.43421053	0.38815789	0.49315069	0.48407643	0.28481013**	0.41830065	0.50993377*	0.37500000	0.47402598
180	0.43037975	0.32098765	0.46250000	0.38271605	0.29585799	0.56363636	0.57988166*	0.48765432	0.33962264	0.27777778**
190	0.59883721	0.44378698	0.43502825	0.41420118	0.25748503**	0.29651163	0.48502994	0.62195122*	0.35838150	0.40963855
200	0.44505494	0.34254144	0.30645161**	0.47701149*	0.43406594	0.39890710	0.47305389	0.45454545	0.35555556	0.35632184

Die jeweils besten Ergebnisse einer Zeile sind mit *, die schlechtesten mit ** markiert. Alle Durchgänge wurden mit Anaconda Python durchgeführt.

Einige statistische Kennzahlen:

Logarithmische Regression: $acc(n) = A + B \cdot ln(x)$ mit $A = 1.540446173, B = -0.216755427$

$r(n, acc(n)) = -0.808621369$

$r(ln(n), acc(n)) = -0.850421062$

5.1.4 Zusammenhängende Aktionsfolgen: Durchgang 2

n	10	11	12	13	14	15	16	17	18	19
10	1.000000000*	1.000000000	1.000000000	1.000000000	1.000000000	1.000000000	1.000000000	1.000000000	0.800000012**	1.000000000
20	0.909090936	1.000000000*	0.920000017	1.000000000	0.875000000**	1.000000000	1.000000000	1.000000000	0.954545438	0.882352948
30	1.000000000*	1.000000000	0.892857134	0.888888889**	1.000000000	0.909090909	0.892857134	1.000000000	1.000000000	1.000000000
40	1.000000000*	0.918918921	0.850000000	0.725000000**	0.918918921	0.921052632	0.833333328	0.869565217	0.976744186	0.945945948
50	0.921568636	0.784313720	0.911111112	0.959183673	0.773584911**	0.899999993	0.961538462*	0.869565210	0.890909083	0.857142861
60	0.918032798	0.913793106	0.920634907*	0.912280692	0.854838700	0.810344819**	0.885245906	0.841269829	0.870967730	0.915254240
70	0.816666679**	0.861111111	0.876923077	0.867647059	0.911764706	0.921875000	0.833333333	0.904109590	0.934426230*	0.836065567
80	0.918032778*	0.825000000	0.807692309	0.840579710	0.868421056	0.869565217	0.800000004**	0.906666666	0.900000000	0.818181819
90	0.705128208	0.915662647	0.915662656*	0.906976751	0.895348844	0.688311690**	0.871794870	0.887640443	0.793103449	0.844444446
100	0.826530612	0.800000004	0.868131864	0.822916667	0.631578949**	0.656862745	0.677777783	0.773809521	0.872340418	0.887640449*
110	0.657657658	0.876288660*	0.623762377**	0.836538462	0.670000000	0.802083333	0.814432990	0.822916667	0.801886794	0.875000000
120	0.716814162	0.792452830	0.809090910	0.800000001	0.883495147	0.803738318	0.620370370**	0.885964909*	0.770642203	0.663716816
130	0.875000004*	0.778625954	0.751999999**	0.842105259	0.862068970	0.861111111	0.794642857	0.825000000	0.788990826	0.803418809
140	0.786259542	0.534482758**	0.795275591*	0.564885496	0.774193548	0.747899162	0.778625954	0.770992366	0.789473685	0.710937500
150	0.819444444	0.619718311**	0.717391303	0.683453237	0.739130436	0.770370371	0.875000000*	0.759124088	0.810218980	0.795774646
160	0.583892620	0.766666666	0.836601303	0.766666663	0.734265735	0.870503599	0.537931037**	0.843283581	0.791946311	0.879194632*
170	0.823529412*	0.743243243	0.745098038	0.730769232	0.541401273**	0.598684211	0.732919255	0.797385619	0.710344828	0.751592359
180	0.765432099	0.851190476*	0.783439493	0.727272728	0.850299402	0.715328468	0.578651683**	0.850000000	0.734939758	0.798742139
190	0.740331493	0.662650602	0.595238095	0.560975610**	0.745562130	0.807228915	0.722891567	0.716867471	0.825301204*	0.746987951
200	0.830601093	0.614035087	0.687150838	0.809248556	0.839572197*	0.710059173	0.754385964	0.633879780	0.759358291	0.579787235**

Die jeweils besten Ergebnisse einer Zeile sind mit *, die schlechtesten mit ** markiert. Alle Durchgänge wurden mit Anaconda Python durchgeführt.

Einige statistische Kennzahlen:

Logarithmische Regression: $acc(n) = A + B \cdot ln(x)$ mit $A = 1.258708566, B = -0.098759019$

$r(n, acc(n)) = -0.713808073$

$r(ln(n), acc(n)) = -0.709844905$

5.2 Auswertungen

5.2.1 Verhalten mehrerer Testläufe in demselben Prozess

Nachdem die Ergebnisse der Evaluation auf zusammenhängenden Datenmengen zunächst teilweise unerwartet schlecht ausgefallen sind, wurde geprüft, ob es einen Einfluss darauf hat, wenn mehrere Testläufe in demselben oder in verschiedenen Python-Prozessen ablaufen. Dafür wurde der Testlauf auf fünf Durchgänge mit $n = 10$ gekürzt. Die folgenden Quelltextfragmente sollen das Vorgehen verdeutlichen.

```
# evaluation.bat
python -m ujepl.eval.evaluation

# evaluation.py:
for i in range(10):
    evaluation(modelSizes=[10],
        trialsEach=[5])
```

Iteration	$acc(n)$					Mittelwert
1.1	1.0	1.0	1.0	0.9	1	0.98
1.2	1.0	1.0	0.78	0.89	1	0.934
1.3	1.0	1.0	1.0	0.6	0.85	0.89
1.4	0.88	1.0	0	0.89	1	0.754
1.5	0.6	1.0	0.86	1.0	1	0.892
1.6	1.0	0.5	0.54	0.92	0.46	0.684
1.7	0.77	0.6	1.0	0.83	0.91	0.822
1.8	0.53	0.2	0.73	1.0	0.5	0.592
1.9	0.85	0.36	1.0	0.87	0.27	0.67
1.10	0.8	0.47	0.69	0.62	0.14	0.544

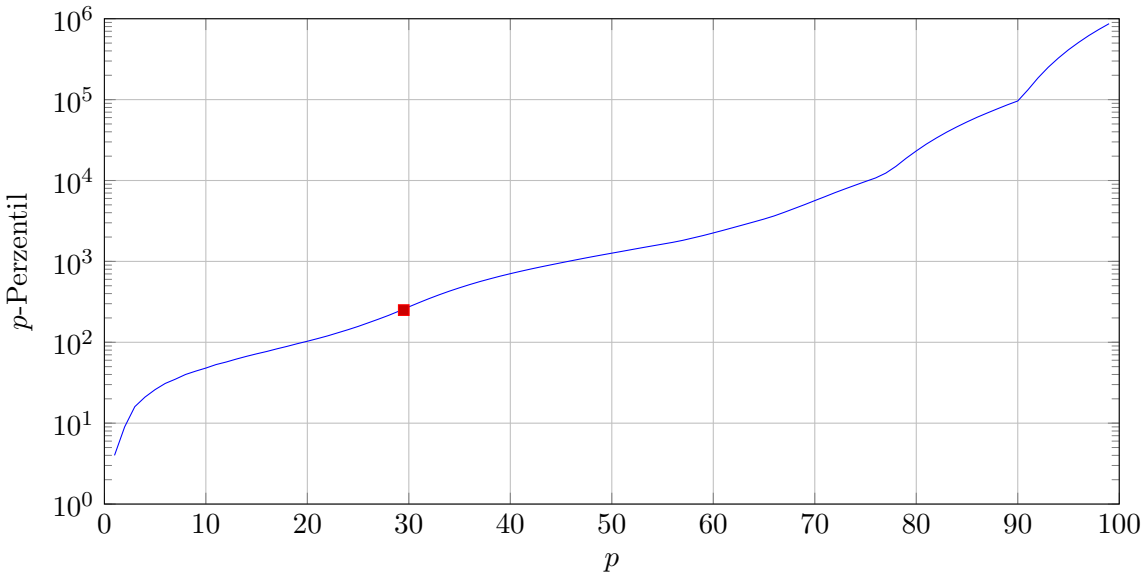
```
# evaluation.bat
FOR %I IN (0 1 2 3 4 5 6 7 8 9) DO
    python -m ujepl.eval.evaluation

# evaluation.py:
evaluation(modelSizes=[10],
    trialsEach=[5])
```

Iteration	$acc(n)$					Mittelwert
1.1	1.00	1.00	1.00	1.00	1.00	1.000
2.1	1.00	0.83	1.00	1.00	0.93	0.952
3.1	1.00	1.00	1.00	0.80	1.00	0.960
4.1	1.00	0.71	0.92	1.00	1.00	0.920
5.1	1.00	1.00	1.00	0.73	0.90	0.920
6.1	1.00	1.00	0.92	1.00	1.00	0.984
7.1	1.00	1.00	0.75	1.00	1.00	0.950
8.1	1.00	1.00	0.86	1.00	1.00	0.972
9.1	1.00	1.00	0.94	1.00	1.00	0.988
10.1	1.00	1.00	1.00	1.00	1.00	1.000

Die deutliche Abweichung hat einen zweiten Durchgang mit weiteren zehn Testläufen in eigenen Prozessen vertretbar gemacht. Grund für die Abweichung ist wahrscheinlich, dass das TensorFlow-Modul zwischen den Durchgängen nicht ordnungsgemäß zurückgesetzt wurde.

5.2.2 Differenz zufälliger Daten

In zwei Durchläufen mit je 10^6 zufälligen Datenpaaren wurde mit großer Übereinstimmung gemessen, wie häufig verschiedene Differenzen auftreten. Die Kurve in Abbildung 5.3 zeigt für eine dieser Stichproben für $p = 1, ..., 100$ den $(\frac{p}{100} \cdot 10^6)$-t größten Eintrag der sortierten Liste bzw. das p-Perzentil auf einer logarithmischen Skala.
Die Markierung zeigt den gewählten Höchstwert 250 für Datenpaare, die als *zusammenhängend* gelten sollen.



$[10^0, 10^1)$: 03%, $[10^1, 10^2)$: 17%, $[10^2, 10^3)$: 26%, $[10^3, 10^4)$: 30%, $[10^4, 10^5)$: 15%, $[10^5, 10^6)$: 09%

Abbildung 5.3: Die Verteilung der Differenz zweier zufällig generierter Daten

5.3 Liste der Befehle der REST-Schnittstelle

Befehl	Argumente, Standardwerte	Beschreibung der JSON-Antwort
/tables	database='any'	Liste von Objekten mit den folgenden Attributen: *database, name, dimCount, valCount, fileSize.*
/tableDimensions	name, database	Liste von Strings, Name der Dimensionen der gegebenen Tabelle. Als Parameter *name* ist das Attribut *name* vom Befehl /tables zu verwenden.
/registerTable	part, title	String, der anzeigt, ob der Befehl erfolgreich war. 'part' ist ein *dateiähnliches*, lesbares Objekt, von dem eine lokale Kopie als Datei angelegt werden kann.
/databases	-	Liste von Strings, die die Namen aller bekannten Datenbanken enthalten.
/dataActions	-	List von Strings, die die Kodenamen für alle bekannten Datenoperationen enthalten.
/submitAction	input, data	POSTet die Aktion in das Modell und ergänzt sie am Ende der aktuellen Sitzung. Das Argument 'data' ist eine Zeichenkettenkodierung für Datenmengen.
/lastAction	-	Objekt mit den Attributen *input* und *data.*
/predictionStrategies	-	Liste von Strings, Bezeichner der bekannten Vorhersagestrategien.
/getPrediction	strategy, data=None, input=""	Objekt mit den Attributen *data*, *confidence* und möglicherweise *error.*

5.4 Quelltextskizzen

Im Folgenden sind Quelltextpassagen angegeben, die Abläufe innerhalb des Systems deutlich machen sollen. Sie entsprechen damit nicht zwangsläufig dem wahren Quelltext, sondern können irrelevante Implementationsdetails vorenthalten.

5.4.1 Konstruktion zufälliger Datenmengen

```
def randomData(probSpec=0.25, probOp=0.15):
    table = random.choice(dataStore.getTables())
    data = Data(table)
    somethingHappened = True
    while somethingHappened:
        somethingHappened = False
        for dimension in table:
            if dimension not in data.unspecifiedDimensions():
                continue
            r = random.random() # aus dem Intervall [0, 1)
            if r < probSpec:
                # Dimension wird spezifiziert
                value = random.choice(dimension.getValues())
                data = data.specify(dimension, value)
                somethingHappened = True
            elif r > 1 - probOp:
                # Datenoperation auf Dimension
                operation = random.choice(DataOperation.getAvailableOperations())

                operations.add((operation, dimension))
                somethingHappened = True
        for (operation, dimension) in operations:
            data = operation.apply(data, dimension)
    return data
```

Anmerkung: Im echten Quelltext wird der Methode `createModel` eine Methode übergeben, die zur Konstruktion der Datenmengen verwendet wird, also entweder zufällig oder zusammenhängend.

5.4.2 Extraktion konkreter Datenmengen aus Datensätzen

```
# Klasse EuroStatData
def extractFromFile(self):

res = EuroStatDataContainer()
dims = []   # enthaelt Dimensionen in richtiger Reihenfolge
xData = None  # enthaelt vorgegebene x-Dimensionswerte
yDim = None  # Dimension, deren Werte in der ersten Zeile stehen
with open(self.table.basepath) as file:
    for i, line in enumerate(file):
        # Kopfzeile enthaelt Informationen ueber Dimensionen
        if i == 0:
            # [...] initialisiere dims, xData, yDim
            continue

        lineData = EuroStatData([...]) # Werte der x-Dimensionen in dieser Zeile
        # Gehoert diese Zeile zum Ergebnis?
        if xData.isSuperSet(lineData):
            # Ist der Wert der y-Dimension vorgegeben?
            if yDim in self:
                for j, cell in enumerate(line):
                    if yDim.values[j] == self[yDim]:
                        # Fuege Zellenwert cell zu res hinzu
            else:
                # fuege alle Werte der Zeile zu res hinzu

    return res

# Klasse DataSelection/DataReduction
def extractFromFile(self):
    baseData = self.base.extractFromFile()
    unk = self.base.unspecifiedDims()
    unk.remove(self.dim)
    dataByAttr = dict()
    # Nach nicht vorgg. Dimensionswerten sortieren
    for k, v in baseData.items():
        attrTuple = tuple((k[dim] for dim in unk))
        if attrTuple not in dataByAttr:
            dataByAttr[attrTuple] = []

        dataByAttr[attrTuple].append((k, v))

    # Je einen Wert aus der Liste auswaehlen/berechnen
    pairs = [self.op(v) for k,v in dataByAttr.items()]

    # Spezifikationen anwenden
    pairs = list(filter(lambda pair: self.isSuperSet(pair[0]), pairs))

    return EuroStatDataContainer(pairs)
```

5.4.3 Ablauf der Evaluationstests

```
MAX_NO_PROGRESS_BEFORE_ABORT = 10
NUM_EPOCHS = 2500
def evaluation():
    modelSizes = random.shuffle(range(10, 210, step=10))
    res = []
    for size in modelSizes:
        model = createModel(size=size)
        noProgressCounter = 0; maxAccuracy = 0.0
        while noProgressCounter < MAX_NO_PROGRESS_BEFORE_ABORT:
            tfp.trainModel(optimizer="adam", epochs=NUM_EPOCHS)
            accuracy = tfp.getScore()
            if accuracy > maxAccuracy:
                maxAccuracy = accuracy
                noProgressCounter = 0
            else:
                noProgressCounter += 1

        res.append((size, maxAccuracy))
    return res
```