Das Gewinnen von Trainingsdaten für überwachtes Lernen mittels Crowdsourcing

Kogler Matthias

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Dezember 2012

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 3. Dezember 2012

Kogler Matthias

Inhaltsverzeichnis

Vorwort

Die vorliegende Arbeit ist im Rahmen meines Studiums an der Fakultät für Informatik, Kommunikation und Medien der FH OÖ entstanden und bildet den Abschluss eines in vielfacher Hinsicht interessanten und intensiven Lebensabschnitts. Ich möchte mich an dieser Stelle ausdrücklich bei meinen Professoren und den uns betreuenden Assistenten und Tutoren für ihr Engagement und ihre Unterstützung bedanken. Die fachliche Kompetenz und der Wille uns bei jeder auftretenden Fragestellung mit Rat und Tat zu Seite zu stehen, haben diesen Teil meiner universitären Bildung zu einem prägenden und ermutigenden Abschnitt gemacht. Die konstruktive, fast schon partnerschaftliche Stimmung die nicht nur zwischen den Studenten untereinander, sondern auch zwischen den Studenten und den Lehrbeauftragten der Fachhochschule vorherrscht, hebt sich in einer beispielhaften, positiven Weise von meinen Erfahrungen an anderen Instituten ab, ein Umstand, der seine Entsprechung auch in der Hilfsbereitschaft und Geduld der Administration, der Haustechnik und allen anderen involvierten Bereichen findet. Mein besonderer Dank gilt meinem Betreuer Prof. Dr. Andreas Stöckl für seine Hilfe bei der Themenauswahl, seiner fachlichen Unterstützung und der Geduld, die er bei unseren Treffen und Besprechungen an den Tag legte. Weiters möchte ich mich bei meinen Mitstudenten bedanken, die mich trotz des teils erheblichen Altersunterschieds von Anfang an als einen der ihren aufnahmen und zum Teil einer Gemeinschaft machten, wie man sie sich als Student nur erträumen kann. Und abschließend und an prominentester Stelle möchte ich mich bei meiner Familie und hier besonders bei meinen Eltern bedanken, die mir auch in schwierigen und herausfordernden Abschnitten meines Lebens nie ihre Unterstützung und ihren Zuspruch versagt und mir letztlich diesen Schritt erst ermöglicht haben.

Kurzfassung

Um das stetig anwachsende Informationsvolumen, welches sich uns heutzutage bietet, verarbeiten zu können, ist es eine gängige Praxis diese Daten mittels *überwachtem maschinellem Lernen* automatisch zu klassifizieren und zu bewerten. Das Generieren der hierfür benötigten Trainingsdaten ist ein zeitintensiver Prozess und wird in hohen Kosten resultieren, wenn dies durch Experten durchgeführt wird. Zusätzlich zum Kostenfaktor besteht die Gefahr von Fehleinschätzungen durch die Experten, sowohl in Bezug auf die Bewertung der Datensätze als auch bei der Identifizierung der *relevanten Attribute* dieser. Die vorliegende Arbeit schlägt als Lösungsansatz die Auslagerung dieser Aufgaben an eine Gruppe von Laien im Internet vor, präsentiert eine Architektur, um die für diesen Prozess notwendigen Bewertungen und Berechnungen an eine Reihe von *clients* zu verteilen und analysiert die Vorteile, die sich aus dieser Herangehensweise ergeben.

Abstract

In order to understand the continuously growing amount of information we face today, it has become a common practice to annotate this data with supervised machine learning techniques automatically. Obtaining the required trainingsdata is time consuming and will be expensive if performed by a small group of experts. Moreover, classifying this data and choosing the *most relevant features* of the underlying model may suffer from *expert bias*. This paper proposes to delegate this task to a group of non-experts in the internet, introduces an architecture to distribute the required annotations and calculations to the clients and analyses the implications and benefits that may derive from this approach.

Kapitel 1

Einleitung

Im Zeitalter der Informationsgesellschaft, insbesondere durch das wachsende Informationsangebot im Web, werden die Menschen zunehmend mit einer Datenflut konfrontiert, die ein Erfassen der wesentlichen Informationen aus der Unmenge der vorhandenen Daten schwierig bis unmöglich macht. Der Autor John Naisbitt diagnostiziert diesen Trend schon 1982 und formuliert [15]:

> We are drowning in information and starved for knowledge.

Ein weiterer Aspekt ist die Geschwindigkeit, mit der die Menschen mit neuen Informationen konfrontiert werden. John Freeman beschreibt in seinem Buch *The Tyranny of E-mail* wie der Mensch in der zunehmenden Informationsflut seiner Fähigkeit beraubt wird, den Alltag zu bewältigen [5]:

> Meanwhile, on flows the email down the screen, like a current with riptides and swirls (...). You paddle frantically and seem to get nowhere.

Die parallele Entwicklung moderner, computerisierter Systeme, die imstande sind diese gewaltige Menge an Rohdaten zu erfassen und verarbeiten legt es nahe, nach Ansätzen zu suchen aus dieser Datenmenge mittels automatisierter Prozesse verwertbare Informationen zu extrahieren. Aus diesem Ansatz hat sich im Laufe der letzten Jahre etwa das sogenannte *datamining*[1] entwickelt, das z.B. bei der Analyse von Diskussionsbeiträgen in *social communities* wie *Facebook* Verwendung findet[2]. In derart gestalteten Szenarien ist die Bewertung bzw. das Extrahieren der Informationen aufgrund des Datenumfangs und der Geschwindigkeit mit der diese generiert werden durch den Menschen einfach nicht mehr möglich. Ein in der aktuellen Entwicklung immer wieder anzutreffender Ansatz, dieser Problematik Herr zu werden ist der Einsatz von maschinellem Lernen bzw. dem *überwachtem maschinellen*

[1] Die automatisierte Extraktion von Informationen aus großen Datenbeständen.
[2] Man spricht in diesem Fall auch von *social media monitoring*.

Lernen, welches auch die Basis der vorliegenden Arbeit bildet. Maschinelles Lernen bezeichnet das Unterfangen, ein computerbasierendes Modell zur automatisierten Wissensgenerierung zu erzeugen. Beim *überwachtem maschinellen Lernen* werden Computerprogramme (genauer handelt es sich um Algorithmen) mit bewerteten Daten trainiert (man spricht in Folge auch von Trainingsdaten) aus denen es Regeln und Mechanismen ableitet, die imstande sind, unbewertete Daten zu klassifizieren. Die derart generierten Modelle können in Folge etwa verwendet werden, um die eingangs genannten Diskussionsbeiträge bestimmten Themengebieten zuordnen zu können oder unerwünschte *E-Mails* automatisch auszusortieren oder mit einem Warnhinweis zu versehen. Weitere Anwendungsgebiete sind etwa die *sentiment analysis* (das automatische Bewerten von Texten hinsichtlich der in ihnen vertretenen Stimmungslage) oder die gerade in unserer Zeit viel thematisierte automatische Vorhersage von Börsenkursen.

1.1 Problemstellung

Als einer der wichtigsten Schritte für das Erstellen eines Modells zur automatisierten Generierung von Wissen wurde bereits im eingehenden Abschnitt das Akquirieren von analysierten bzw. klassifizierten Trainingsdaten genannt. Dieser Daten müssen dabei zwei wesentliche Kriterien erfüllen:

1. **Umfang**: Die Datenmenge muss umfangreich genug sein, um den Analysevorgang des Computerprogramms mit allen notwendigen Informationen für fortführende Bewertungsprozesse zu versorgen.
2. **Qualität**: Für ein weitgehend fehlerfreies Generieren von neuem Wissen aus unbewerteten Daten durch das Modell ist es nicht nur notwendig, dass die initial dem System zugeführten Informationen richtig bewertet wurden, sondern auch, dass diese Trainingseinheiten durch jene Attribute repräsentiert werden, die für eine Bewertung am aussagekräftigsten sind. Findet diese Auswahl der Attribute (engl. feature selection) nicht statt, kann es geschehen, dass ein Modell falsche Regeln ableitet[3], durch zu starke Reduktion zur Übergeneralisierung neigt oder durch zu viele Attribute nicht imstande ist die vorgenommenen Bewertungen mit den Charakteristiken der Daten zu verknüpfen.

Die klassische Herangehensweise, diese Trainingsdaten durch einen oder eine Gruppe von Experten erstellen zu lassen, ist mit folgenden Problemen behaftet:

- Der Einsatz von Experten ist kostenintensiv.

[3] So könnte etwa ein System zur Identifizierung von geometrischen Figuren, welches mit farbigen Bildern trainiert wird versuchen, die in den Trainingsdaten identifizierten Formen mit den Farbinformation der Bilder zu verknüpfen.

- Die Ressourcen zur Generierung der Daten bzw. zum Evaluieren der erstellten Modelle sind durch die Zahl der Experten begrenzt.
- Experten (seien es nun Einzelpersonen oder Expertengruppen) neigen zu Fehleinschätzungen, sowohl in Bezug auf die Bewertung der Datensätze als auch bei der Identifizierung deren relevanten Attribute (auf diesen Umstand wird detailliert in Abschn. 4.1 eingegangen) .

Ein weiterer Umstand, der beim Experteneinsatz ins Hintertreffen gerät, ist, dass bei Systemen, die über einen langen Zeitraum Daten generieren (wie etwa beim eingangs erwähnten Extrahieren von Informationen aus *social communities*), ein kontinuierliches Erfassen von Daten und ein Evaluieren der Leistungsfähigkeit des Modells nicht gegeben ist.[4]

Schlussendlich wird auch die Starrheit der zumeist hochkomplexen Anwendungen, die im Bereich des maschinellen Lernens Anwendung finden kritisiert, da diese auf den verwendeten Rechnern unter Berücksichtigung spezifischer Anforderungen installiert werden müssen und gerade bei Beispielen wie dem *social media monitoring* ungeeignet erscheinen. Zusätzlich zu der geringen Flexibilität, die diese Systeme aufweisen, ist zumeist auch ein tiefgehendes Fachwissen der zugrundeliegenden Algorithmen für ihre Verwendung von Nöten.

1.2 Motivation

Im Sinne der zuvor definierten Problemstellungen erscheint es zum einen notwendig, die aktuelle Herangehensweise, das Identifizieren und Generieren von Trainingsdaten für überwachtes maschinelles Lernen bzw. die Auswahl der diese Daten repräsentierenden Attribute durch einen kleinen Kreis von Experten zu hinterfragen. Als Alternative zum Expertenwissen wird das Konzept der kollektiven Intelligenz vorgestellt und die Auslagerung der genannten Aufgaben an eine Masse von Internetnutzern (in Folge auch als crowd bezeichnet) auch unter Berücksichtigung des Kosten- und Ressourcenfaktors vorgeschlagen.

1.3 Zielsetzung und Abgrenzung

In der vorliegenden Arbeit soll analysiert werden, welche Vorteile sich aus der Verlagerung des klassischen Expertenbereichs zur kollektiven Intelligenz ergeben. Die verschiedenen involvierten wissenschaftlichen Bereiche (und die damit einhergehende zersplitterte Terminologie und Sichtweise) machen es notwendig, eine ganzheitliche Betrachtung herbeizuführen. Zielsetzung ist ist

[4] Da der Experte nur zu maximal zwei festgesetzten Zeitpunkten mit dem System interagiert, und zwar zum Zeitpunkt der Datenbewertung und in der Evaluierungsphase, wird die Dauer dazwischen nicht für den Bewertungsprozess genutzt und eine allfällige Fehleranfälligkeit des verwendeten Modells zur Laufzeit auch nicht überprüft.

es somit, eine Basis für weitere wissenschaftliche Forschung zu bilden und weiters eine Architektur zu entwerfen und umzusetzen, die geeignet ist zum Gewinnen von Trainingsdaten in einer Crowdsourcing Umgebung eingesetzt zu werden.

1.4 Gliederung

Einleitung: Dieses Kapitel soll einen grundlegenden Einblick in die behandelte Materie geben und die der Arbeit zugrundeliegenden Problemstellungen, Motivationsfaktoren und Zielsetzungen erläutern.

Begriffsdefinitionen: Zum allgemeinen Verständnis der weiterführenden Betrachtungen werden in diesem Kapitel die für die Arbeit grundlegenden Begriffe erläutert.

Kritik und begriffliche Neudefinition: Um den Begriff des Crowdsourcing in einen dieser Arbeit entsprechenden Kontext zu setzen, soll in diesem Abschnitt eine Betrachtung der klassischen Definition und anschließend eine Erweiterung im Sinne der Loslösung durch die verengende, ausschließlich unternehmerische Sichtweise präsentiert werden.

Die Klugheit der Gruppe: In diesem Kapitel soll eine Analyse für mögliches Fehlverhalten von Experten getätigt werden und die diesem Phänomen zugrunde liegenden Mechanismen untersucht werden. Als Alternativvorschlag zum Expertenwissen wird der Begriff der kollektiven Intelligenz vorgestellt sowie dessen Wirkungsweise eingehend beleuchtet.

Konzept und Idee: Entsprechend der eingangs getroffenen Zielsetzung wird an dieser Stelle das Konzept eines Prototypen entwickelt, sowie die wesentlichen Kriterien der dieser zugrunde liegenden Architektur definiert.

Projektumsetzung: Die Beschreibung des zuvor definierten Prototypen bildet das Kernstück dieses Abschnitts. Zu diesem Zweck werden eingangs die einzelnen Bestandteile und die dem Ansatz zugrunde liegende Terminologie erläutert. Anschließend erfolgt eine Detailbeschreibung der Projektumsetzung und die Implementierungsdetails der dieser zugrunde liegenden Funktionalitäten. Abschließend wird die exakte Arbeitsweise bzw. der Datenaustausch zwischen den *clients* und dem *server* veranschaulicht und auf die Aspekte Sicherheit und Datenvalidierung eingegangen.

Analyse: Die Analyse der zuvor erfolgten Projektumsetzung bildet schließlich den Inhalt des vorletzten Kapitels.

Schlussbemerkung: Im abschließenden Kapitel wird die vorliegende Arbeit zusammengefasst und ein Ausblick auf zukünftige Entwicklungen vorgenommen.

Kapitel 2

Begriffsdefinitionen

2.1 Überwachtes maschinelles Lernen

Maschinelles Lernen ist eine Technik aus dem Bereich der künstlichen Intelligenz um aus bestehenden Erfahrungswerten bzw. existierenden Daten neues Wissen zu generieren, indem ein Vorhersagemodell erstellt wird, welches automatisch Bewertungen vornimmt. Grundsätzlich wird zwischen den folgenden 3 verschiedenen Herangehensweisen unterschieden:

1. überwachtes Lernen (*engl. supervised learning*),
2. unüberwachtes Lernen (*engl. unsupervised learning*),
3. bestärkendes Lernen (*engl. reinforcement learning*).

Als überwachtes maschinelles Lernen bezeichnet man den Prozess aufgrund von bereits identifizierten Trainingsdaten eine Regel bzw. Funktion abzuleiten, die in weiterer Folge verwendet wird, um neue, nicht identifizierte Daten automatisch einer Bewertung zu unterziehen. Entsprechend der Art des durch diese Funktion generierten Resultats unterscheiden wir zwischen folgenden Funktionsschemata:

1. **Klassifizierung**: Dem durch die Funktion bewerteten Datensatz wird ein diskreter Wert zugeordnet (etwa der Einteilung eines Blogpostings in *spam* oder *nicht-spam*).
2. **Regression**: Die zu generierende Ausgabe ist durch einen kontinuierlichen Wert abbildbar (z.B. Prognose eines Temperatur oder eines Längenwertes).

Die der vorliegenden Arbeit zugrunde liegende Implementierung und das entsprechende Konzept verwendet ausschließlich die Technik des *überwachtes Lernens* und das Klassifizierungsschema (siehe Abschn. 6.2.1).

Die Anwendung von überwachtem Lernen auf eine Problemstellung kann in die folgenden Schritte untergliedert werden:

1. **Problemstellung**: Zunächst wird die Problemstellung formuliert und die betreffenden Dateninhalte identifiziert, sowie das benötigte Funk-

tionsschema gewählt.

2. **Definition und Wahl der Trainingsdaten**: Als nächsten Schritt wird ein repräsentativer Satz von Trainingsdaten ausgewählt und dieser (sofern etwa eine Klassifizierung nicht aufgrund von Messergebnissen schon vorliegt) durch Experten bewertet.
3. **Identifikation der relevanten Eingangsdaten**: In diesem Prozessschritt schließlich werden die aussagekräftigsten Informationen, welche in den für die Wissensgenerierung verwendeten Daten enthalten sind, identifiziert. Es ist wichtig sich des Umstandes bewusst zu sein, dass Daten oftmals eine Vielzahl an Informationen beinhalten, die für eine allfällige Bewertung nicht relevant oder sogar hinderlich sein können. Diese irrelevanten Daten (engl. *noise* bzw. *feature irrelevance*) können bei Texten etwa Satzzeichen und Anmerkungen sein[1] oder bei einem Satz von Messergebnissen Daten sein, deren Bedeutung für eine Bewertung nicht gegeben ist oder diese sogar verfälschen könnte. Ein weiteres Beispiel wären in einem Bild enthaltene Farbinformationen, die etwa für das Identifizieren von geometrischen Strukturen nicht benötigt werden. In der Fachliteratur wird das Reduzieren bzw. Selektieren der Parameter als *feature reduction* oder *feature selection* bezeichnet. Sollten diese Parameter (oft wird auch der Begriff *Attribute* verwendet) nicht identifizierbar sein, wird oftmals versucht, mittels eines *brute-force* Ansatzes alle zur Verfügung stehenden Daten dem Lernalgorithmus zur Verfügung zu stellen, in der Hoffnung, die wesentlichen Daten anschließend identifizieren und isolieren zu können. Diese Herangehensweise ist jedoch wie in der Fachliteratur beschrieben (siehe etwa [9, Kap. 2]) selten zielführend. Als gängige Herangehensweise findet diese Auswahl durch eine Gruppe von Experten statt, mit all den Problemstellungen, die sich durch diesen Ansatz ergeben können (siehe Abschn. 4.1). Ein Ziel der vorliegenden Arbeit ist es zu beweisen, dass dieser Schritt (und auch Prozessschritte wie das Generieren der Trainingsdaten bzw. das Evaluieren der erstellten Klassifizierungsfunktion) in der gleichen Art und Weise nicht nur durch eine Gruppe von Nicht-Experten durchgeführt werden kann, sondern dass diese Aufgabe auch in einer qualitativ besseren (und zudem kostengünstigeren) Art und Weise bewerkstelligt werden kann.

   An dieser Stelle ist anzumerken, dass die Zahl der einen Datensatz repräsentierenden Parameter zwar ausreichend sein soll, um die nötigen Informationen für eine Vorhersage zur Verfügung zu stellen, eine zu

[1] Ein oftmals beobachteter Fehler bei der Verwendung von Texten ist etwa der Umstand, dass die meisten Lernalgorithmen Groß- und Kleinschreibung unterscheiden, d.h. das etwa ein Wort welches aufgrund seiner Satzstellung entsprechend mit einem Grossbuchstaben oder einem Kleinbuchstaben beginnt als jeweils unterschiedliche Daten gehandhabt werden und somit den Parameterraum signifikant vergrößern.

große Menge an Parametern aber schnell zu einer dramatischen Abnahme der Vorhersagegenauigkeit führen kann (siehe [26]).

Die in diesem Abschnitt angeführten Überlegungen bezüglich der in einem Datensatz bereits definierten Parameter und ihrer Handhabung darf keineswegs darüber hinwegtäuschen, dass in den Daten oftmals auch Informationen enthalten sein können die durch eine Bearbeitung zu einer wesentlich verbesserten Vorhersagequalität der generierten Funktion führen können. So kann etwa das Umwandeln von kontinuierlichen Werten in stetige Werte mittels der Definition von Grenzwerten zu einer erheblichen Reduktion des Parameter Raums (engl.*feature space*) führen. Wesentlich interessanter ist hier aber das Generieren neuer Parameter aus sich möglicherweise gegenseitig beeinflussenden Informationen (engl.*feature interaction*) bzw. das Zusammenfassen redundanter Informationen (engl. *feature redundancy*). Diese Technik wird als *feature construction* bezeichnet.

4. **Auswahl des Algorithmus**: Hier erfolgt aufgrund der zuvor definierten Problemstellung bzw. des gewünschten Funktionsschemas und der Art der in der Wissensgenerierung verwendeten Daten die Auswahl eines passenden Lernalgorithmus, etwa *Support Vector Machine* oder Entscheidungsbaum (*engl. decision tree*).
5. **Training eines Modells**: Gemäß der zuvor getroffenen Entscheidungen werden die gesammelten Trainingsdaten in den ausgewählten Algorithmus gefüttert und eine Funktion generiert, die imstande ist, alle ihr zur Verfügung gestellten Trainingsdaten und deren entsprechende Resultate abzubilden.[2]
6. **Evaluierung und Reduktion der Eingangsdaten**: Nachdem der Trainingsvorgang abgeschlossen ist und die generierte Lernfunktion zur Bewertung einer Anzahl nicht identifizierter Inhalte herangezogen wurde, kann die Genauigkeit bzw. Zuverlässigkeit besagter Funktion etwa an einem Testset aus bereits bewerteten Daten gemessen werden und diese Evaluierung zu einer Reduktion bzw. Neustrukturierung der verwendeten Parameter oder einer Verbesserung des für das Training verwendeten Trainingsdatensatzes verwendet werden.

2.2 Crowdsourcing

Crowdsourcing bezeichnet das Auslagern von Aufgaben an eine Gruppe von Anwendern im Internet. Der Begriff, erstmals verwendet in einem im *Wired Magazine* veröffentlichtem Artikel [7], wurde vom amerikanischen Journalisten Jeff Howe geprägt und ist ein mittlerweile auch im alltäglichen Sprach-

[2] In der dieser Arbeit zugrundeliegenden Implementierung wird diese als Modellversion bezeichnet (siehe Abschn. 6.2.1).

gebrauch gängiger Neologismus[3], der eine Verschmelzung der beiden Begriffe Outsourcing und Crowd andeutet. Im Gegensatz zum herkömmlichen Outsourcing, dem Auslagern von Aufgaben durch Firmen an Drittunternehmen, erfolgt hier die Übertragung besagter Aufgaben an eine nicht näher spezifizierte, oftmals anonyme Gruppe von Usern. Christian Papsdorf definiert Crowdsourcing in der folgenden Weise [16]:

> Crowdsourcing ist die Strategie des Auslagerns einer üblicherweise von Erwerbstätigen entgeltlich erbrachten Leistung durch eine Organisation oder Privatpersonen mittels eines offenen Aufrufs an eine Masse von unbekannten Akteuren, bei dem der Crowdsourcer und/oder Crowdsourcees frei verwertbare und direkte wirtschaftliche Vorteile erlangen.

Aktuell existiert ein geradezu verwirrend hoher Diversifikationsgrad an Definitionen des Begriffs, wobei zahlreiche dieser nur bedingt imstande sind die Vielfalt der bestehenden Plattformen und deren Wirkungsweisen abzubilden. Aufgrund dieses Umstandes besteht auch das Bestreben, ein flexibles, vereinheitlichtes Schema zur Klassifizierung von *crowdsourcing* Systemen zu entwickeln (siehe etwa [13]). Ein zentrales Element, welches die klassische Definition und der überwiegende Teil der neu entwickelten Definitionen gemeinsam haben, ist der Fokus auf die Wirtschaftlichkeit des Prozesses oder der Versuch der Gewinnmaximierung.

Für eine Kritik dieser verengten Sichtweise, eine Begründung dieser bzw. den Versuch einer erweiterten Neudefinition des Begriffs wird auf Abschn. 3.1 verwiesen.

2.3 Google Prediction API

Mit der *Prediction API*[4] bietet *Google* Entwicklern ein *Cloud*-basierendes Service zur Verfügung um überwachtes maschinelles Lernen in Implementierungen verwenden zu können, ohne sich über die Auswahl der entsprechenden Lernalgorithmen oder Fragen wie Skalierbarkeit und Verfügbarkeit der benötigten Rechenleistung Gedanken machen zu müssen.[5]

[3] Ein neu eingeführter bzw. gebrauchter sprachlicher Ausdruck.

[4] https://developers.google.com/prediction/

[5] Da es sich hier um ein von der Firma *Google* zur Verfügung gestelltes Service handelt, ist künftig von einer Weiterentwicklung bzw. dem Implementieren von zusätzlichen Funktionalitäten zu rechnen. Als Entwicklungsstand für die dieser Arbeit zugrunde liegende Implementierung ist der Oktober 2012 anzusehen.

Kapitel 3

Kritik und begriffliche Neudefinition

3.1 Crowdsourcing

3.1.1 Notwendigkeit einer Neudefinierung

Crowdsourcing ist im Laufe der letzten Jahre vom reinen *Web 2.0* Schlagwort zu einem weltweit beachteten strategischen Modell geworden, um Interesse an Unternehmen und deren Tätigkeiten in einer viralen Art zu generieren, Problemlösungen zu bewältigen an denen selbst hoch qualifizierte Experten scheitern und neue, innovative Produkte und Designs zu erstellen. Trotz der Allgegenwärtigkeit und der (schon fast inflationär anmutenden) Verwendung des Begriffs im alltäglichen Sprachgebrauch kommt es immer wieder vor, dass die Partizipation der User an Crowdsourcing Projekte in einer unerwünschten Weise stattfinden. Der Redakteur des *Wired Magazine* Mark Robinson spricht in diesem Fall von *crowdsourcing backfire* bzw. im Fall des Auflehnens der Crowd gegen die ursprünglichen Projektziele als *crowdslapping*. Das Magazin *Zeitschrift für Marktforschung und Marketing* identifiziert drei wesentliche Motivationen für Crowdslapping [3]:

1. **Hijacking**: Das (zumeist bewusst böswillige manchmal aber auch humoristisch verstandene) Kapern eines Crowdsourcing Prozesses.
2. **Empörung**: Verärgerung der User (etwa aufgrund des Verdachts der Ausbeutung oder Missachtung bzw. der Manipulation durch die Initiatoren).
3. **Verselbstständigung**: Das Heranbilden von Netzwerken innerhalb der Community zum Zwecke der gegenseitigen Bevorzugung bzw. das Reagieren der User auf Beiträge anderer Community-Mitglieder und ein damit einhergehendes *Wegbewegen* von der ursprünglichen Zielsetzung der Initiatoren.

Beispiele hierfür sind etwa der mittlerweile schon zum Schlagwort gewordene *Bud-Spencer-Tunnel* [27] oder der von der Firma *Chevrolet* initiierte Ideenwettbewerb zur Bewerbung ihres Produktes *Tahoe*[1]. Zu einem überwiegenden Teil können 2 Gründe für derartige Fehlentwicklungen identifiziert werden:

1. Ein prinzipielles Unverständnis der Struktur einer Crowd als Community und die Nicht-Beachtung der Interaktionsmechanismen einer solchen.
2. Die Einschränkung der Crowd aufgrund einer zu starken Fokussierung auf den kommerziellen Aspekt des Prozesses und das damit einhergehende Gefühl der Ausnutzung bzw. der Unfähigkeit zur Gesamtheit des Gestaltungsprozesses beitragen zu können.

Im folgenden Abschnitt soll nun eine Kritik auf die oftmals ausschließliche Betrachtung der Wirtschaftlichkeit herkömmlicher Crowdsourcing Aktivitäten erfolgen und eine alternative Sichtweise präsentiert werden.

3.1.2 Alternative Sichtweisen

Die in Folge dargelegten Gedanken mit der Zielsetzung den Fokus weg von einer rein unternehmerischen, auf Gewinnmaximierung gerichtete Sichtweise hin zu einem gemeinschaftlichen Arbeiten in einer Gruppe von weitgehend gleichberechtigten Mitgliedern zu lenken darf natürlich nicht darüber hinweg täuschen, dass dieser Herangehensweise spätestens dann Grenzen gesetzt sind, sobald finanzbedürftige Unternehmungen aufgrund des Fehlens jedweder Möglichkeit eben diese Mittel zu lukrieren an den Rand ihrer Existenzmöglichkeit gedrängt werden. Der Internet-Pionier Jaron Lanier etwa kritisiert den Gedanken des Crowdsourcing und der *kollektiven Intelligenz* als Ausprägungen eines *digitalen Maoismus* [10] und prognostiziert, dass Entwicklungen wie etwa die Open Source-Bewegung die Möglichkeit der Mittelklasse Inhalte zu generieren auf lange Sicht hin zerstören wird. Die Strategie im Kollektiv etwas zu verbessern sei alleine aufgrund der Biologie des Menschen zum Scheitern verurteilt [28]. Wenngleich diese Form des Kulturpessimismus und diverse weitere Prognosen Laniers (so sprach er schließlich Unternehmungen wie *Facebook* oder *Twitter* jedwede Fähigkeit ab reale Geldmengen zu erwirtschaften) mittlerweile als widerlegt gelten dürften, so sollen die folgenden Ausführungen nicht dahin gehend interpretiert werden, dass eine auf Gewinnlukrierung orientierte Sichtweise bei Crowdsourcing Projekten vollständig falsch wäre. Alleine das vollständige Fokussieren auf den kommerziellen, an traditionelle Unternehmensstrukturen orientierten Ansatz soll durch eine ganzheitlichere Sicht der Zusammenhänge abgelöst werden.

Die weithin übliche Gliederung eines durch ein Unternehmen dominierten Wertschöpfungsprozesses in einen Akt der Produktion, ein Erstellen ei-

[1] http://news.cnet.com/1606-2_3-6056633.html

nes Produktes und einen anschließenden Konsum ist eine im Verständnis des industriellen Zeitalters verwurzelte, sich an der Schaffung konkreter, physischer Produkte orientierende Herangehensweise. Der in Harvard unterrichtende Professor Yochai Benkler bezeichnet in seiner 2006 erschienen Publikation *The Wealth of Networks: How Social Production Transforms Market and Freedom* [1] diese Betrachtungsweise als einen Anachronismus und prognostiziert aufgrund der Umstellung von einem zentralisierten Verteilungssystem von Information, Interaktion und Arbeit hin zu einem vernetzten System einen radikalen Wandel in der Art in der Information, Produktion und Interaktion kommerziell genutzt werden kann. Benkler illustriert das Ausmaß dieser Entwicklung mit dem folgendem Beispiel [29]: Die 1835 von James Gordon Bennett gegründete erste auf den Massenvertrieb ausgerichtete Zeitung in New York benötigte eine initiale Investition von 500 Dollar, was einem Wert von etwa 10.000 Dollar heute entspricht. 1850, also nur 15 Jahre später hätte das gleiche Unterfangen einen heutigen Wert von etwa 2.5 Millionen Dollar bedurft. Diese Entwicklung, die zunehmende Bedeutung (und Menge) von Kapital bzw. die Zentralisierung der Produktion und Distribution werde in der heutigen Zeit durch das Web 2.0 invertiert, das heißt das die Bedeutung von Kapital als Antriebsfaktor zunehmend durch intrinsische Motivationsfaktoren (wie etwa dem Wunsch nach Herausforderung) sowie durch soziale und emotionale Motive (z.B. das Streben nach Anerkennung oder Gemeinschaft) ersetzt oder vervollständigt werden. Eric von Hippel etwa stellt die (zugegeben provokante These) auf, dass eine Loslösung des Innovationsprozesses von rein kommerziell agierenden Unternehmen an sich stattfinden würde [6]:

> We conclude that innovation by individual users and also open collaborative innovation increasingly compete with—and may displace—producer innovation in many parts of the economy.

Der Wikipedia-Gründer Jimmy Wales betont, dass die Sichtweise Teilnehmer an gemeinschaftlichen Projekten (so etwa auch die Mitglieder der Crowd) als billige Arbeitskräfte zu sehen wesentliche Community Aspekte ignoriert und der Motivation deren Mitglieder nicht zuträglich ist. Diese, seien es nun Freiwillige oder auch explizit für eine Aufgabe verpflichtete Personen, sollten in ihrer Gesamtheit vielmehr als gleichberechtigte Partner (an einer anderen Stelle spricht er von Kunden) begriffen werden, deren Bedürfnisse nach Eigeninitiative und -verantwortung durch die Verwendung einer flachen hierarchischen Organisationsstruktur Rechnung zu tragen sei [30]. Der amerikanische Schriftsteller und Visionär Alvin Toffler postuliert in seinem 1980 erschienene Buch *The Third Wave* [24], dass in der *Prosumentenethik* (*Prosumer*, eine Wortverschmelzung der Begriffe *consumer* und *producer* bezeichnet Personen, die gleichzeitig zu Konsumenten und Produzenten werden[2]) nicht mehr Besitz, sondern die Tätigkeit der Menschen einen hohen

[2] Toffler erweitert hier Überlegungen, die schon von Marshall McLuhan und Barrington

Stellenwert haben wird. Von besonderem Interesse ist hier der Wandel des durch Toffler definierten Begriffs des *Prosumers* hin zu einer Bedeutungsverschmelzung der beiden Begriffe *professional* und *consumer*. Der Konsument wird hier nicht mehr nur als ein um die Rolle und Bedeutung des Produzierenden erweitertes, sondern vielmehr als ein mit den Fähigkeiten und Beurteilungsfähigkeiten des Professionellen ausgestattetes Individuum gesehen. Die Aggregation von Fähigkeiten und Wissen in der Crowd, wie sie ein elementarer Grundgedanke der *kollektiven Intelligenz* ist, führt hier also keineswegs zu einem (wie etwa durch den eingangs erwähnten Jaron Lanier geschmähten) *Mittelmaß des dumpfen Mobs*, sondern vielmehr zu einer echten Diversität von Expertenmeinungen und Ideen.

All diesen Ausführungen ist gemeinsam, dass zwar die wirtschaftliche Instrumentalisierung der Crowd nicht vollständig ausgeklammert wird, der hierarchische *top-down* Ansatz der klassischen Unternehmensstruktur aber zumindest als ein hinterfragenswerter Ansatz erscheint.

3.1.3 Neubewertung

Vor dem Hintergrund der genannten Betrachtungen wird ersichtlich, dass die klassische Definition nicht geeignet ist das Phänomen Crowdsourcing in seiner Gesamtheit abzubilden bzw. voll auszunutzen. Analysen erfolgreicher Projekte zeigen, dass der reine Fokus auf die Unternehmensperspektive weder zwingend notwendig ist noch ein unternehmens- bzw. auftragsbezogener Anstoß durch eine zentrale Autorität vorliegen muss. Das zentralistische Modell einer den Prozess kontrollierenden Person oder Gruppe steht im Widerspruch zu der Fähigkeit der Gemeinschaft selbsttätig tätig zu werden, sich zu organisieren und behindert durch sein starres hierarchisches Korsett die Entwicklung einer Vielfalt von Meinungen und die produktive Koexistenz von unterschiedlichsten Fähigkeiten. Gerade im Sinne einer Ausprägung des Web 2.0 und der dieser Entwicklung immanenten Informations- und Kommunikationstechnologien sowie deren dezentrale Interaktionsmechanismen muss bei der Betrachtung von Crowdsourcing das Festhalten an das Bild eines Inhalte, Produktionsmittel und Zielsetzungen kontrollierenden Produzenten als nicht zeitgemäß erscheinen.

Im Sinne der genannten Betrachtungen wird für die künftige Verwendung des Begriffes Crowdsourcing eine Redefinierung angedacht, die das interaktive Element des Prozesses betont. Die Crowd wird als ein Netzwerk von weitgehend frei miteinander kollaborierenden Mitgliedern begriffen, die sowohl die Art ihres Beitrages als auch ihre Position gemäß ihrer persönlichen Einschätzung definieren können und sich diese auch im Laufe des Prozesses ändern kann. Die Initiierung des Prozesses und die Definition der Erfolgskriterien kann sowohl aktiv durch die Gruppe selbst oder durch einen externen

Nevitt 1972 formuliert wurden [14].

Anstoß erfolgen. Die primäre Aufgabe eines allfälligen externen Initiators ist es hierbei die Crowd als Gruppe von gleichwertigen Individuen zu betrachten und deren Meinungen im Sinne einer flachen Hierarchie soweit als möglich Rechnung zu tragen.

Kapitel 4

Die Klugheit der Gruppe

4.1 Der Irrtum des Expertenansatzes

Die Annahme, dass Meinungen und Urteile direkt durch die von uns gewonnenen Eindrücke entstehen, dass also unser Begreifen der Welt kausal aus unserer Wahrnehmung folgt, ist eine irreführende oder zumindest vereinfachte Vorstellung. Mittels der Betrachtung optischer Täuschungen (siehe Abb. 4.1) lässt sich verdeutlichen, dass die gewonnenen Erkenntnisse über unsere Umwelt zu einem großen Teil durch bereits existierende Vorstellungen definiert sind. Es entspricht also weniger der Wahrheit, dass wir glauben bzw. erkennen was wir sehen, sondern vielmehr, dass wir oftmals nur jenes wahrnehmen, was bereits in unserer Vorstellung existiert oder die Interpretation der gewonnenen Eindrücke unseren Vorstellungen anpassen.

Um der gewaltigen Informationsmenge, die sich uns bietet, Herr zu werden, filtern wir die bereits aufgrund der eingeschränkten Möglichkeiten unserer Wahrnehmung reduzierte Signalmenge erneut durch unsere *Blickrichtung* bzw. Erwartungshaltung. Popper etwa vergleicht die Art der menschlichen Erkenntnis mit einem Scheinwerfer und spricht folglich vom *Scheinwerfermodell der Erkenntnis* [18]:

> Wir erfahren ja erst aus den Hypothesen, für welche Beobachtungen wir uns interessieren sollen, welche Beobachtungen wir machen sollen.

Jede Beobachtung ist somit theoriebeladen, Erkenntnis wird nicht (alleine) aus Erfahrung abgeleitet. Dieses Erkannte kann somit auch das Falsche sein. Der Biologe Rupert Riedl geht von einem biologischen Wissen aus, welches ein System aus Hypothesen und Vorurteilen enthält welche uns

> (...) im Rahmen dessen, wofür sie selektiert wurden, wie mit höchster Weisheit lenken, uns aber an dessen Grenzen vollkommen und niederträchtig in die Irre führen.

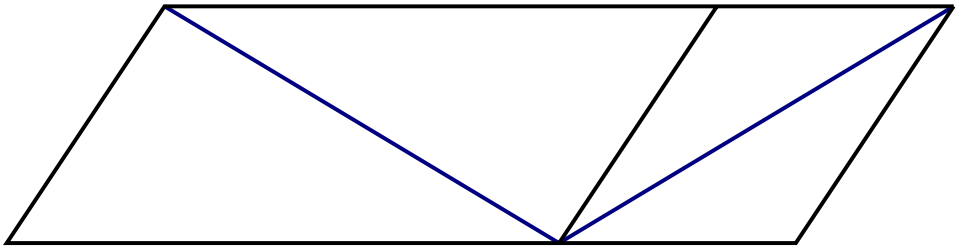


Abbildung 4.1: Das Sandersche Parallelogramm, benannt nach dem Deutschen Psychologen Friedrich Sander, ist eine optische Täuschung, die den Betrachter zur Annahme verleitet, dass die eingezeichneten diagonalen Linien verschiedene Längen aufweisen. Ein Erklärungsmodell besteht darin, dass die Figur in der realen Welt keine Entsprechung besitzt und vom Verstand als rechteckige Form interpretiert wird, in der die Diagonalen tatsächlich verschiedene Längen besitzen würden. Als mögliche Erklärung für den Versuch ein neues gedankliches Konstrukt durch ein bereits existierendes Verständnismodell zu interpretieren kann das Sparsamkeitsprinzip (engl. *principle of parisony*) gesehen werden. Der Mechanismus der Größenkorrektur wird *Größenkonstanz* [31] genannt. Bildquelle: [32].

Der Neurobiologe Elkhonon Goldberg reflektiert über diesen Prozess in einer positiven Weise und bezeichnet ihn als Weisheit, als eine Fähigkeit die Aufmerksamkeit auf das Wesentliche lenken zu können, die sich als ein Reifungsprozess aufgrund von Wiederholungen im Laufe seines Lebens manifestiert hat [21]:

> What I have lost with age in my capacity for hard mental work, I seem to have gained in my capacity for instantaneous, almost unfairly easy insight.

Seine biologische Analogie findet diese These im Prägungsverhalten von neuronalen Netzwerken im menschlichen Gehirn, das aufgrund seiner Beschaffenheit geradezu perfekt gestaltet ist, um Muster zu erkennen bzw. zu vervollständigen. Als Folge von Reizen und Impulsen die von uns verarbeitet werden (wie etwa Klänge, visuelle Eindrücke oder auch Emotionen), wird eine Reihe von Neuronen aktiviert und deren Verbindung untereinander gestärkt. Tritt derselbe Impuls vermehrt auf, wächst die Stärke besagter Verbindungen in einer entsprechenden Weise. Wir prägen uns also Verhaltens- und Denkmuster auf biologischer Ebene ein. Jeffrey M. Stibel veranschaulicht das mit der Aussage [21]:

> Cells that fire together, wire together.

Je nach dem Grad der Ausprägung ist es aber schließlich nicht mehr notwendig das gesamte Neuronennetz zu aktivieren, um einen Impuls zu identifizieren. Darin liegt auch die Fähigkeit unseres Gehirns, Muster zu vervollständi-

gen, fehlende Elemente anzunehmen bzw. Rückschlüsse auf eine Gesamtheit ohne die Kenntnis aller Einzelteile zu ziehen.

Diese Fähigkeit sei an dem folgendem Beispiel demonstriert[1]:

> Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rest can be a total mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Das semantische Erfassen dieses Textes, für den Computer eine nahezu unlösbare Aufgabe, erscheint uns als spielerisch und leicht. In diesem Umstand liegt auch eine der größten Stärken der Mustererkennungsfähigkeit des menschlichen Gehirns. Die Fähigkeit, komplexe Informationen und Sachverhalte in kleine Einheiten (wie in diesem Beispiel in wortähnliche Konstrukte) zu unterteilen, welche für sich nur soweit analysiert werden müssen, bis wir glauben, ein bekanntes Muster identifizieren zu können. Unsere Fähigkeit zur Abstraktion geht sogar so weit, dass wir nicht nur von Wort zu Wort springen, sondern vielmehr sogar nur einzelne Passagen eines Textes aufnehmen und die Lücken mit durch unsere Erfahrungen und Erwartungshaltungen definierten Inhalten füllen. Der Prozess, ein falsches derartig eingeprägtes Erkennungsmuster zu hinterfragen und redefinieren, ist somit nicht nur ein zutiefst intellektuell herausfordernder, er widerspricht sogar auf elementarer Ebene unserer *biologischen Strukturierung*. Betrachten wir nun unter den genannten Überlegungen die Fähigkeit des Experten einen Sachverhalt einschätzen bzw. Voraussagen treffen zu können, so werden potenzielle Unzulänglichkeiten dieses Ansatzes offensichtlich.

Anmerkung Es soll an dieser Stelle keineswegs bestritten werden, dass das intensive Studium eines Wissensgebietes in den *allermeisten* Fällen zu einer verbesserten Beurteilungsfähigkeit in Bezug auf diese Materie führen kann und zumeist auch führen wird. Das Ablehnen einer Erkenntnis an sich, wie es etwa durch den Philosophen Kratylos [33] postuliert wurde[2], ist nicht im Sinne des Autors. Vielmehr geht es an dieser Stelle darum die nach wie vor weitverbreitete These der *Unfehlbarkeit* des Experten kritisch zu hinterfragen und in weiterer Folge im Sinne der *kollektiven Intelligenz* ein diesen Ansatz erweiterndes Konzept aufzuzeigen.

[1] http://baheyeldin.com/writings/linguistics/language-recognition-the-brain-as-a-complex-pattern-recognition-machine.html

[2] Kratylos erweitert das Bild des Heraklit, mann könne nicht zweimal in denselben Fluss steigen, da sich dieser zwischenzeitlich verändert hätte dahin gehend, dass er die These aufstellt, es sei schon nicht möglich in besagten Fluss ein einziges Mal zu steigen, da dieser sich bereits währenddessen verändert.

Die Möglichkeit dieser Problematik zu begegnen, indem der einzelne Experte durch eine Gruppe von Experten ersetzt wird, ist ebenfalls nur in einem eingeschränkten Maße gegeben. Wie aktuelle Studien verdeutlichen, tendieren Wissenschaftler dazu, Gruppen von homogenen Meinungen und Ansichten zu bilden, ein Umstand, der sich etwa durch ihr Zitationsverhalten belegen lässt [19] und in der Psychologie oftmals als *confirmation bias* bezeichnet wird.[3] Seine Begründung findet dieses Verhalten in der menschlichen Natur Selbstbestätigung zu suchen, indem wir uns mit Gleichgesinnten umgeben (siehe etwa die Theorie der Selbstbestätigung bzw. engl. *self-affirmation* [34]) und in gruppendynamischen Prozessen wie dem Konformitätsverhalten. In der Psychologie wird dieses Phänomen als *Gruppendenken* bezeichnet [35], das besonders dann auftritt, wenn die Gruppe abgeschottet von äußeren Einflüssen und Meinungen agiert.

4.1.1 Eine Taxonomie der Verständnismechanismen

Die eingangs in diesem Kapitel angeführten Begründungen für ein allfälliges Fehlgeleitetwerden der menschlichen Erkenntnisfähigkeit soll in diesem Abschnitt zum Zwecke einer besseren Verständlichkeit um eine Taxonomie der evolutionär begründeten Denkprinzipien und Verhaltensweisen und der sich daraus ergebenden potenziellen Fehlern erweitert werden.

Denkprinzipien

Die folgenden Prinzipien ergeben sich aus der Notwendigkeit mit begrenzten geistigen Ressourcen[4] unser Umfeld begreifen zu können bzw. aus der evolutionären Notwendigkeit die benötigten Ressourcen zu minimieren.

- **Scheinwerferprinzip**: Der eingangs schon ausführlich beschriebene Umstand, dass unsere Wahrnehmung nicht nur durch die Limitierung unserer Sinnesorgane bzw. der uns zur Verfügung stehenden Kapazitäten beschränkt ist, sondern auch durch unsere Erwartungshaltung, unsere *Blickrichtung* (Popper spricht hier wie bereits erwähnt vom *Scheinwerfermodell der Erkenntnis*).

  Potenzielle Quellen für Fehlbeurteilungen:

  - Die *Bindung der Aufmerksamkeit*, etwa durch eine Ablenkung oder einen überproportional präsenten Reiz.
  - Das *Einschleifen falscher Grundannahmen und Hypothesen*, da unser Fokus nur die Aufnahme von Signalen zulässt, die in unserer Annahme existieren, widersprüchliche und neue Zusammenhänge

[3] In der genannten Studie wird infolge dieses Ballungsverhaltens auch ein Abnehmen der Innovationsfähigkeit diagnostiziert. Die Expertengruppe ist einfach nicht breit genug gestreut, um ein kritisches Hinterfragen bzw. die Neugenerierung von Wissen zu ermöglichen und den popperschen *Scheinwerferkegel* auf ein ausreichend großes Umfeld zu lenken.

[4] Etwa die Gedächtnisleistung und die Verarbeitungsfähigkeit unseres Gehirns.

können nicht aufgenommen werden. Die Prägung der Hypothese wird verstärkt.

- **Sparsamkeitsprinzip**: Das Sparsamkeitsprinzip (auch als ökonomisches Prinzip bezeichnet) liegt in der evolutionären Theorie begründet, dass Individuen (bzw. eine Spezies) durch den ökonomischen Einsatz ihrer Ressourcen einen Selektionsvorteil erhalten. Der Einfluss, den dieses Prinzip auf unsere Erkenntnisgewinnung ausübt, ist aber nicht nur durch den Versuch die verwendeten Energien zu minimieren gegeben, sondern vielmehr durch das Maß der uns zur Verfügung stehenden Ressourcen *während* des Entscheidungsprozesses. Der Psychologe und Nobelpreisträger für Wirtschaft Daniel Kahneman führt zur Verdeutlichung dieses Umstandes folgendes Beispiel an: Die Analyse von Richtersprüchen hat ergeben, dass der Zeitpunkt bzw. der energetische Zustand des Richters einen maßgeblichen Einfluss auf die Art des Urteils ausübt. Begnadigungen werden statistisch gesehen am häufigsten abgelehnt, wenn der Richter gerade ein hohes Arbeitspensum zu absolvieren hat (also ein hohes Quantum an Energie gebunden oder bereits verbraucht ist) oder die Mittagspause unmittelbar bevorsteht (und damit die letzte Nahrungs- bzw. Energieaufnahme schon länger zurückliegt). Ein überraschend hoher Anteil dieser Richtersprüche wurde übrigens revidiert oder zumindest im weiteren Instanzenweg adaptiert. Inwiefern der Entscheidungsprozess in diesen Fällen genau durch die genannten Umstände beeinflusst worden ist, kann an dieser Stelle nicht beurteilt werden. Wichtig ist nur die Erkenntnis, dass der Einfluss, den unsere Ressourcen auf unsere Entscheidungsfindung ausüben nicht nur durch den erwarteten Verbrauch, sondern auch durch das aktuell zur Verfügung stehende Ausmaß gegeben ist.

  Potenzielle Quellen für Fehlbeurteilungen:

  – Das *Ignorieren neuer Zusammenhänge*, da eine Problemanalyse durch existierende (und möglicherweise in weiterer Folge irreführende) Erkenntnismuster möglich scheint.

Angeborene Verhaltensweisen und Beurteilungsprinzipien

Konrad Lorenz bezeichnet diese Prinzipien in seinem Buch *Die Rückseite des Spiegels* [11] als die *angeborenen Lehrmeister* und nennt folgende Definition:

> Die angeborenen Lehrmeister sind dasjenige, was vor allem Lernen da ist und da sein muss, um Lernen möglich zu machen.

Als diese *Lehrmeister* können die folgenden Erkenntnismechanismen und Interpretationsprozesse identifiziert werden:

- **Strukturerwartung**: Der menschliche Verstand scheint von der Annahme einer objektiven, an klare Regeln gebundenen Welt auszugehen

und strebt danach, Strukturen und Zusammenhänge in dem von ihm Wahrgenommenen zu identifizieren.[5] In der Gestaltpsychologie wird dieser Umstand als *Gesetz der Prägnanz* bezeichnet.

Potenzielle Quellen für Fehlbeurteilungen:

- Die *Überbewertung* von Ordnungsmuster bzw. das *Konstruieren* von nicht existierenden Zusammenhängen, wie sie bei vielen optischen Täuschungen, aber auch bei groben (zum Teil manipulativen) Vereinfachungen und Argumenten vorkommt.

- **Kausalitätserwartung**: Das Streben für Erkenntnisse und Entwicklungen Ursachen zu identifizieren und die Erwartung, dass gleiche Resultate unter denselben Bedingungen zustande kommen, wird als Kausalitätserwartung bezeichnet. Die Ursache (oder der auslösende Faktor) hat damit eine direkte, eindeutige Auswirkung. In der Kognitionspsychologie wird zwischen *linearem Ursache-Wirkungsdenken* (Ursachen werden unabhängig voneinander betrachtet und führen zu einem entsprechenden Resultat) und *vernetztem Ursache-Wirkungsdenken* unterschieden.

  Potenzielle Quellen für Fehlbeurteilungen:

  - Aufgrund der menschlichen Bestrebung einfache Theorien von Zusammenhängen zu bilden, wird oftmals das *lineare Ursache-Wirkungsdenken* präferiert und die Suche nach Ursachen beim ersten zufriedenstellenden Resultat beendet. Dies kann etwa zu einer Fehleinschätzung bei der Analyse von Grenz- und Sonderfällen führen oder der Manipulation Vorschub leisten, indem etwa Szenarien aufgebaut werden die unsere Kausalerwartung instrumentalisieren.[6]
  - Eine weitere Möglichkeit der Irreführung unserer Reflexionsfähigkeit besteht darin, Einzelereignisse als eine kausal bedingte Abfolge von Geschehnissen zu interpretieren. Dieses Bedürfnis resultiert aus der leichteren Merkbarkeit derart ineinander *verzahnter* Geschichten und der leichteren Rekonstruktionsfähigkeit einzelner *Episoden* dieser. Eine praktische Anwendung findet dieser Mechanismus in zahlreichen Mnemotechniken. Fehlen Bausteine in dieser Struktur, so konstruieren wir uns diese gerne oder adaptieren unsere Wahrnehmung zum Zwecke des Erreichens eines *stimmigen* Gesamtbildes.

- **Die Anlage zur Induktion**[7]: Auch als *Befähigung zur Hypothesenbil-*

[5] Eine dieser Erwartungshaltungen äußert sich etwa in der Suche nach Symmetrie.

[6] Ein häufiger Vorwurf, der bei kritischen Reflexionen über die Gesundheitsindustrie auftaucht, durch welche Bedürfnisse geschaffen bzw. Wirkungen suggeriert würden, deren Ursache alleine in den Gewinnbestrebungen der Unternehmen liegt.

[7] Als *Induktion* wird das Ableiten einer Theorie aus empirischen Beobachtungen be-

dung bezeichnet, beschreibt dieser Mechanismus den Hang des menschlichen Geistes, mittels plausiblen Schließens aus unseren Beobachtungen Theorien abzuleiten und ist gemeinsam mit der zuvor definierten *Kausalitätserwartung* jenes Interpretationsmuster, welches uns das Erstellen von generalisierten Aussagen ermöglicht. Die derart abgeleiteten Regeln (in der Psychologie als *Heuristiken* bezeichnet) werden im Sinne des Sparsamkeitsprinzips ohne ein sorgfältiges Abwägen ihres Wahrheits- und Glaubwürdigkeitsgehalts zumeist als allgemeingültig angesehen.

Die potenziellen Fehleinschätzungen, die sich durch diese Art der Hypothesenbildung ergeben können, sind mannigfaltig, alleine schon aufgrund des Umstandes, dass wir im Sinne des Sparsamkeitsprinzips danach streben, mit einem Minimum an bereits existierenden Hypothesen unser Auslangen zu finden und in unserem Streben nach Selbstbestätigung den *Scheinwerfer* unserer Aufmerksamkeit primär auf Eindrücke lenken, die diese bestätigen. Voreilig getroffene Annahmen entwickeln somit eine Beharrlichkeit. Ein weiteres Beispiel für die Unzulänglichkeit der induktiven Schlussfolgerung ist die Tendenz Umkehrschlüsse als plausibel zu interpretieren. Diese gängige Fehleinschätzung, auch als *Scheitern am Modus Tollens* bezeichnet, sei in Folge an einem kurzen Beispiel demonstriert. Die Beobachtung, dass eine Straße nach dem Regen nass ist, führt zum Aufstellen der Hypothese „Aus dem Umstand, dass es geregnet hat, folgt eine nasse Straße“.

$$Regen \rightarrow \text{nasse Straße}$$

Der Umkehrschluss „Aus einer nassen Straße folgt die Erkenntnis, dass es geregnet hat“ ist nicht zulässig.

$$\text{nasse Straße} \rightarrow Regen$$

Vielmehr würde hier (unter Anwendung des genannten *Modus Tollens*) die Aussage „Aus einer nicht nassen Straße folgt die Erkenntnis, dass es nicht geregnet hat“ gültig sein.

$$\neg\text{nasse Straße} \rightarrow \neg Regen$$

- **Neugier- und Sicherheitstrieb**: Dass das Streben nach Sicherheit einer der ureigensten Triebe des Menschen ist und wesentliche Teile unseres Denkens und Handelns bestimmt, ist eine Erkenntnis, die an dieser Stelle keine weitere Ausführung bedarf. Interessant ist aber der Umstand, dass unsere Neugier bzw. unser Explorationstrieb als ausgleichendes Element ebenfalls biologisch seine Entsprechung findet. Felix von Cube bezeichnet die Exploration als eine „(...) Triebhandlung des Sicherheitstriebes, also die mit Anstrengungen verbundene

zeichnet (der Schluss vom Besonderen auf das Allgemeine), der gegenläufige Prozess wird als *Deduktion* bezeichnet.

Umwandlung der Unsicherheit in Sicherheit“ [2]. Dieser Explorationstrieb steht somit also im Konflikt mit dem Streben nach Sicherheit und dem Sparsamkeitsprinzip. Die sich aus diesem Neugierverhalten ergebenden Vorteile scheinen aber die entstehenden Risiken soweit zu überlagern, dass sich eine feste Verankerung in der menschlichen Biologie als Selektionsvorteil erwiesen hat. Exploration ist notwendig um sich die Welt vertraut zu machen und eventuelle weitere Risiken identifizieren und abschätzen zu können. Konrad Lorenz beschreibt in einem 1943 veröffentlichten Aufsatz wie gerade Tiere, die kaum an eine ökologische Nische angepasst sind (etwa Ratten, aber auch der Mensch), ein hohes Neugierverhalten und eine entsprechende Risikobereitschaft an den Tag legen. Neugierverhalten wird von Lorenz in diesem Zusammenhang auch als für den Erwerb mentaler Strukturen förderlich identifiziert, eine dem Explorationsverhalten dienliche Umgebung führe zu einer besseren Gehirnentwicklung. Interessant ist in diesem Zusammenhang auch der Umstand, dass Neugier nicht nur ein besseres Umgehen mit reizintensiven, herausfordernden Umgebungen ermöglicht, sondern auch, dass in der Psychologie von einer umgekehrten Wirkungsweise ausgegangen wird. Das oftmalige Konfrontiertwerden mit neuen Inhalten (so auch Thesen, Analysen oder Datenmaterialien) führt zu einem gesteigerten Explorationsverhalten. Betrachten wir nun die alltägliche Erfahrung und eingangs argumentativ unterlegte Überlegung, dass Triebe, Denkmuster bzw. Verhaltensweisen die Tendenz haben, sich bis zu einem Grad zu manifestieren und zu verfestigen, der ein Hinterfragen nahezu unmöglich macht, erkennen wir auch die Gefahr, in die der einzelne Experte in diesem Umfeld gerät. Ist das Festhalten an althergebrachten Ansätzen und Hypothesen eine im Sparsamkeitsprinzip bzw. dem Wunsch nach Selbstbestätigung begründete Verhaltensweise, so ist die Bereitschaft unbewiesene Standpunkte zu vertreten und neue Thesen alleine ihrer Neuwertigkeit wegen zu akzeptieren im Explorations- und Neugierstrieb begründet.

4.1.2 Die Fähigkeit zur Reflexion

Als Reflexion wird die Fähigkeit des Menschen bezeichnet (zumeist) intuitive Eingebungen mittels eines methodischen Denkansatzes zu korrigieren. Die Fähigkeit des Menschen, Schlussfolgerungen derart zu hinterfragen, steht im Widerspruch zu drei elementaren kognitiven Veranlagungen des Menschen:

1. **Dem Selbstbestätigungstrieb**: Eine Neubewertung wird vermieden bzw. werden die dieser zugrunde liegenden Annahmen sowie deren Resultate angepasst, um unser Selbstbildnis zu bestätigen.
2. **Dem Sparsamkeitsprinzip**: Der Mensch vermeidet Analysen gänzlich oder passt gewonnene Erkenntnisse an, um Ressourcen zu sparen.

3. **Dem Scheinwerferprinzip**: Die Blickrichtung der Aufmerksamkeit wird nur in unzureichendem Maße auf widersprüchliche Signale gelenkt, d.h. das Erfassen dieser ist oftmals gar nicht möglich.

Daniel Kahneman ordnet die Reflexion dem Bereich der *Deliberation* (dem logischen, sequenziellen und bewussten Denken) zu und spricht vom *lazy controller*, einem System, das nur dann aktiv wird, wenn „(...) intuitive Ideen ausbleiben, oder wenn die Ideen *bewussten* Regeln widersprechen“ [8]. Der Einfluss, den der Selbstbestätigungstrieb auf unsere Reflexionsfähigkeit ausübt, ist am leichtesten zu beschreiben und aus alltäglichen Beobachtungen bekannt. Widersprüchliche Erkenntnisse und Theorien werden nachträglich derart adaptiert, dass die Erkenntnis des eigenen Fehlers vermieden wird, wenn dies nicht gelingt, wird der Wahrheitsgehalt der dem Reflexionsprozess zugrunde liegenden Unstimmigkeiten angezweifelt. Der Physiker Jean Perdijon formuliert dies so [17]:

> Das wichtigste Kriterium betrifft die logische Stimmigkeit, d.h. das Fehlen von inneren Widersprüchen oder Gegenbeispielen. Dies hindert uns jedoch nicht daran (...) zusätzliche Parameter oder zweckmäßige Anfangsbedingungen einzuführen, um schließlich am Ende das Gewünschte herauszubekommen.

Subtiler arbeiten die zwei Denkprinzipien (siehe Abschn. 4.1.1), deren Wirkungsweise exemplarisch an zwei Beispielen erklärt werden soll.

Der Einfluss des Sparsamkeitsprinzips

Der Einfluss, der durch das Sparsamkeitsprinzip auf unser Reflexionsvermögen ausgeübt wird, kann durch ein Beispiel des Verhaltensökonomen Dan Ariely beschrieben werden, welches er anlässlich eines Vortrages bei der *EG Konferenz* 2008 in Monterey, Kalifornien präsentierte [36] und in dem eine Reihe von Medizinern mit einer (fiktiven) Fallstudie eines Patienten konfrontiert werden. Dieser leidet seit geraumer Zeit an Hüftproblemen. Einige Tage zuvor hätte eine Bewertung der bisherigen Behandlungsmethoden ergeben, dass trotz des Einsatzes aller zur Verfügung stehenden Behandlungsmethoden kein Erfolg ersichtlich ist und der Arzt hätte dem Patienten eine Operation und das Einsetzen eines Hüftimplantats angeraten. Während also nun der Patient auf dem Weg zur Operation ist, entdeckt der Arzt, dass ein Medikament noch nicht bezüglich seiner Wirksamkeit getestet wurde. Der Arzt wird vor die Wahl gestellt, den Patienten zurückzuberufen oder die Operation durchführen zu lassen. Erwartungsgemäß entscheidet sich ein Großteil der Mediziner dazu ihre zuvor getroffene Bewertung zu revidieren und den Patienten einer Behandlung mit dem verbliebenen Medikament zu unterziehen. Wird das Szenario aber dahin gehend verändert, dass zwei Medikamente noch nicht eingesetzt wurden und der Arzt vor die Wahl gestellt wird, die

Operation wie geplant durchführen zu lassen, oder den Patienten einer erneuten medikamentösen Behandlung zu unterziehen und diese Wahlmöglichkeit zugleich mit der Frage verbunden, welches der beiden Medikamente er verwenden würde, entscheidet sich ein signifikant größerer Anteil der Ärzte dazu ihre Entscheidung nicht zu revidieren. Das zweite Szenario stellt den Mediziner also vor die Wahl eine bereits getroffene Entscheidung bestehen zu lassen, oder sich einem wesentlich komplexeren Entscheidungsprozess auszuliefern, der Wahl sein Urteil zu revidieren *und* den neuen Behandlungsweg auszuwählen. Die Erwartung eines ressourcenintensiveren Entscheidungsprozesses führt somit laut Ariely zu einer Neubewertung der Situation die dazu führt, dass der Vorgabe (der Hüftoperation) der Vorzug gegeben wird.

Der Einfluss des Scheinwerferprinzips

Ein Beispiel für das Scheinwerferprinzip ist das von Kahneman als *narrow framing* bezeichnete Phänomen. Der menschliche Geist scheint die Angewohnheit zu haben, Problemstellungen gesondert bzw. abgegrenzt zu betrachten und nicht als Teil einer gleich gestalteten Reihe von Problemen oder getroffenen Entscheidungen, d.h. wir sind außerstande unsere Sichtweise auf die das zentrale Problem umgebenden Fakten und deren Einflüsse zu lenken. Er beschreibt dies folgendermaßen [37]:[8]

> (...) comes up in something that I call *narrow framing*, which is, you focus on the problem at hand and don´t see the class to which it belongs.

Dies bestätigt auch die in Abschn. 4.1.1 unter dem Punkt *Kausalitätserwartung* erwähnte Bevorzugung des *linearen Ursache-Wirkungsdenken.* Ein weiterer Effekt, der sich beim *narrow framing* übrigens beobachten lässt, ist die zeitliche Komponente des bewusst methodischen Reflexionsvorganges der uns aufgrund seiner sequenziellen Natur (wir bilden die uns unterstützende Argumentationskette zeitlich abfolgend nacheinander und in Richtung der aktuellen Problemstellung) dazu verleitet, länger zurückliegende Fehlannahmen als weitgehend unbeteiligt an einem aktuellen Widerspruch zu betrachten (beim Erreichen des aktuellen Glieds der Argumentenkette ist der Ansatz des zurückliegenden Gliedes vergessen). Weiters tendieren wir dazu, das Ausmaß des Einflusses, welches eine weiter zurückliegende Entscheidung auf die aktuelle Fragestellung ausübt, zu unterschätzen, da Fehler die Tendenz haben sich zu zerstreuen und ihre Ursache bei längeren kausal aufeinander abfolgenden Ketten schwerer zu lokalisieren ist (ein Umstand, der Programmierern bei der Fehleranalyse keineswegs neu ist).

[8] Im beigefügten Quellverweis ist ein vollständiges Beispiel Kahnemans für die Wirkungsweise von *narrow framing* angegeben.

4.1.3 Resümee

Aus den in diesem Abschnitt genannten, zumeist durch unsere Triebe definierten Verhaltens- und Interpretationsmuster scheint ein Umstand klar zutage zu treten. Der menschliche Verstand neigt dazu, Fehleinschätzungen vorzunehmen und hat die Tendenz diese auch manchmal geradezu beharrlich zu verteidigen. Die Begründung für diese Fehleranfälligkeit ist zu einem hohen Anteil im evolutionären Selektionsvorteil zu suchen, der dem Menschen aus diesen Verhaltensweisen erwachsen ist. Unsere Fähigkeit zur Reflexion scheint, sofern wir sie nicht aufgrund unseres Strebens nach einer Minimierung der für die Erkenntnisgewinnung verwendeten Ressourcen gänzlich vermeiden, nur in einem eingeschränkten Ausmaß imstande, intuitiv gewonnene Erkenntnisse und deren Hypothesen auf ihre Richtigkeit zu prüfen.

Vor allem benötigen wir aber für eine Prüfung bzw. Re-Evaluierung unserer Erkenntnisse und Theorien eine Erweiterung oder Neuausrichtung unseres Blickwinkels (des eingangs erwähnten popperschen *Scheinwerferkegels*) welche oftmals aus eigener Kraft aufgrund der folgenden Umstände nicht möglich scheint:

1. Dem Hang zur Bewahrung und Verteidigung von Thesen.
2. Dem Wunsch in unseren Erkenntnissen Bestätigung für unsere Ansichten zu finden.
3. Der Unmöglichkeit widersprüchliche Signale außerhalb unseres Blickfeldes wahrnehmen zu können.

Anmerkung Die gewonnenen Erkenntnisse sollen und dürfen natürlich nicht dazu führen, dass wir die Fähigkeit des Menschen zum Erkenntnisgewinn an sich infrage stellen und Heinrich Faust gleich der kognitiven Sackgasse durch die Hinwendung zu Aberglauben und Metaphysik zu entkommen trachten. Es erscheint aber angebracht über alternative Wege nachzudenken, die geeignet scheinen den klassischen Weg des *urteilenden Experten* zu hinterfragen und zu komplettieren.

Als ein Lösungsansatz für diese Problematik scheint das Hinzubringen neuer Meinungen und Sichtweisen angebracht, das Ersetzen des einzelnen Experten durch eine Gruppe von Experten. Wie am Beginn dieses Abschnitts durch eine Analyse des Zitationsverhaltens belegt wurde, ist diese Herangehensweise ebenfalls mit ihren Tücken versehen. Gerade Fachleute neigen dazu, sich mit einer Gruppe von Gleichgesinnten zu umgeben, die Gruppe ist einfach nicht breit genug gestreut um ein wirkliches Hinterfragen von Theorien und Ansichten zu gewährleisten. Als ein Alternativvorschlag soll im nächsten Abschnitt der Begriff der *kollektiven Intelligenz* vorgestellt und dessen Wirkungsweise beleuchtet werden.

4.2 Kollektive Intelligenz

Intelligenz leitet sich vom lateinischen *intellectus* ab, was soviel wie *Erkenntnis* oder *Einsicht* bedeutet. Der Psychologe William Stern[9] definiert sie als „(...) die allgemeine Fähigkeit eines Individuums, sein Denken bewusst auf neue Forderungen einzustellen." Diese gängige Definition der Intelligenz, welche sich auf Eigenschaften eines Einzelindividuums bzw. Leistungen, die dieses ohne die Mithilfe einer Zweitperson oder Gruppe erbringt, bezieht die Missachtung der spezifischen Fähigkeit des Menschen, seine intellektuellen Fähigkeiten an seine Umgebung anzuschließen und mit ihr in Wechselwirkung zu treten, wurde in der Vergangenheit vielfach einer berechtigten Kritik unterzogen. Das folgende Kapitel soll sich mit dem Begriff der *kollektiven Intelligenz* beschäftigen, deren Mechanismen analysieren und die Bedeutung des sozialen Faktors in diesem Kontext beleuchten.

Bereits Aristoteles formulierte in seiner *Summierungsthese*, dass Entschlüsse größerer Gruppen besser sind, als jene von wenigen Fachkundigen und Entscheidungsträgern.[10] Die Aufklärung sah in der *kritischen* Öffentlichkeit eine Möglichkeit, den Prozess des Loslösens vom Aberglauben zu beschleunigen und Irtümer zu beheben sowie neues Wissen zu generieren. Die Fähigkeit der Gruppe gemeinsam eine Erkenntnisfähigkeit zu entwickeln, die der des Einzelindividuums oftmals überlegen ist, sowie die Erkenntnis, dass Fehleinschätzungen und Trugschlüsse oftmals durch die Erweiterung des Blickwinkels unter Hinzufügung von weiteren, teils kontroversiellen Ansichten anderer Menschen überwunden werden können, ist also keineswegs neu. Als kollektive Intelligenz wird die Fähigkeit des Menschen bezeichnet, durch Aggregation von Meinungen, Erfahrungen und Wissen in der Gruppe zu einer besseren Entscheidung zu gelangen, als dies den Einzelpersonen möglich ist. Aktuelle Studien belegen hierbei nicht nur die Fähigkeiten der Menge aufgrund ihrer Erfahrungswerte bestehende Sachverhalte besser einschätzen zu können, sondern auch die Fähigkeit durch einen Austausch von Meinungen erfolgreicher bei der Behandlung neuer Problematiken zu sein [25]. Die Analyse vorhandener Fälle von kollektiver Intelligenz belegt, dass weniger das Expertenwissen der Einzelperson, sondern vielmehr die breite Streuung der Fähigkeiten und Ansichten innerhalb der Gruppe für dieses Phänomen verantwortlich ist [22].

Der Kybernetiker Francis Heylighen vergleicht die Gesellschaft an sich mit einem vielzelligen Organismus, deren Individuen die Rolle der Zellen übernehmen. Die Kommunikationskanäle, die diese verbinden, vergleicht er mit einem Nervensystem. Diese Analogie verwendet auch Jeffrey M. Stibel, der als die hoffnungsvollste Basis für das Auftreten von kollektiver Intelligenz das Internet ansieht und dieses mit dem menschlichen Gehirn vergleicht

[9] William Stern ist unter anderem bekannt für die Erfindung des Intelligenzquotienten.

[10] Bekannt ist diese These auch durch die Aussage „Das Ganze ist mehr als die Summe seiner Teile".

[21].[11] Die Fähigkeit zu einer Aggregation von *Klugheit* werde nicht durch die Leistungsfähigkeit moderner Maschinen ermöglicht, sondern vielmehr durch die Freiheit der Kommunikationsmedien, die das Web 2.0 den Menschen bietet. Ein System, das kollektive Intelligenz begünstigt, müsse sich weniger an der Exaktheit der binären Logik, sondern vielmehr an den Schwächen des menschlichen Geistes orientieren.[12] Der Psychologe Peter Kruse nennt als die wichtigste dieser Eigenschaften die bereits in Abschn. 4.1.1 thematisierte Fähigkeit des Menschen intuitiv zu agieren und Sachverhalte jenseits unserer rationalen Verständnisfähigkeit abzubilden. Der sich aus dieser Herangehensweise ergebenden Gefahr, Interpretationsmuster zu verwenden, die veraltert sind oder durch die zuvor erläuterten Verständnismechanismen auftreten, ist durch eine weitgehend große Vernetzung mit anderen Menschen und der Bildung von Kollektiven zu begegnen [38]. Durch die unterschiedlichen Teilnehmer dieses Kollektivs, wird das Auftreten von neuen Ansätzen und Aspekten begünstigt, welche die Problemstellungen besser erklären und lösen, sowie Vorurteile und Fehleinschätzungen bereinigen können. In dem 2004 veröffentlichtem Buch *The Wisdom of Crowds* [22] nennt Surowiecki als wesentliche Bedingungen für eine erfolgreiche Entwicklung von kollektiver Intelligenz diese Faktoren:

1. offene Kommunikation,
2. Meinungsvielfalt,
3. Unabhängigkeit,
4. Dezentralität,
5. neutrale Aggregation.

Diese Rahmenbedingungen, insbesonders die *Meinungsvielfalt* und *neutrale Aggregation* beschreiben auch einen wesentlichen Sachverhalt, der hilft, das Phänomen vom Begriff der *Schwarmintelligenz* [39] abzugrenzen. Die hochwertigsten, durch das Kollektiv getroffenen Entscheidungen, kommen nicht nur durch Kompromisse und Konsens zustande, sondern vor allem durch eine Vielfalt der Sichtweisen nebeneinander. Die Gefahr, die sich durch eine frühzeitige Konsensfindung ergeben kann, beschreibt Dirk Helbing in einer an der *ETH Zürich* durchgeführten Studie [12], in der eine Reihe von Studenten in drei Gruppen aufgeteilt und diese mit sechs identischen Fragestellungen konfrontiert wurden. Diese Vorgehensweise wurde fünfmal wiederholt und eine der Gruppen nach der erstmaligen Schätzung mit dem Mittelwert der Studienteilnehmer dieser Gruppe konfrontiert. Eine zweite Gruppe erhielt den Schätzwert aller Teilnehmer. Bei nahezu jeder Fragestellung stellte sich her-

[11] Stibel prägt dementsprechend den Begriff *Internet Intelligence*.

[12] Als eine dieser Unzulänglichkeiten des menschlichen Geistes wird etwa die Fähigkeit zur Ablenkung genannt. Der Erkenntnisprozess an sich ist laut Stibel ein sich wiederholender Prozess des Erfahrens und Hinterfragens, er verwendet hierfür den Begriff *loop-de-loop*, der durch unsere Fähigkeit zur Ablenkung erst die Möglichkeit zum Gewinnen neuer Sichweisen erlangen kann.

aus, dass die Qualität der Antworten im Verlauf der Studie abnahmen und zwar in dem Maße, in dem die Gruppe mit den durchschnittlichen Schätzwerten der Gesamtheit konfrontiert wurde. Helbing schliesst aus diesen Beobachtungen, dass das frühzeitige Verschwinden von Extremwerten durch eine vorzeitige Annäherung an einen Mittelwert für die Diversität der Ansichten ein Hinderniss ist. „Es ist wichtig, ein Meinungsspektrum zu kultivieren und nicht von Vornherein auf Konsens zu gehen.“. Das Vorhandensein von abweichenden Meinungen sei bedeutend, um der Gruppe ein kritisches Hinterfragen ihrer Ansichten zu ermöglichen. Als ein Weg dies zu gewährleisten, kann der aus dem Bereich der Rhetorik stammende Begriff des *Advocatus Diaboli* bezeichnet werden. Als einen solchen bezeichnet man eine Person, die Argumente und Positionen einer Gegenseite vertritt, ohne dass sie diese zwingend als richtig ansieht. Bruno S. Frey etwa fordert einen solchen Akteur in Unternehmen und in der Politik, um die Menschen zum Begründen und Hinterfragen von Vorschlägen zu motivieren [40]. Ähnlich lautende Überlegungen gehen übrigens sogar soweit, das Phänomen des Internet *Trolls* als ein wesentliches und stimulierendes Element einer lebendigen, kreativen Netzkultur zu bezeichnen [41].[13]

4.2.1 Der soziale Faktor

Der Autor Don Tapscott bezeichnet *openness* und *sharing*, also das freie Austauschen von Meinungen und Ideen, als einen der wichtigsten Faktoren für die Entfaltung der kollektiven Intelligenz [23]. Der am *Center for Collective Intelligence* des MIT unterrichtende Haym Hirsh sogar als dessen Grundlage. Die in diesem Prozess der Gruppe unterbreiteten Gedanken sollten durch sie einer kritischen Betrachtung unterzogen werden, die wiederum mit der Gruppe geteilt werden muss. Dies hat soweit als möglich unbeeinflusst vom Status stattzufinden, den die Akteure innerhalb des sozialen Gefüges haben.[14] Diese *aggregierte* Intelligenz tritt vor allem dann klar zutage, wenn in der Gruppe eine hohe Übereinstimmung der Ziel- und Wertevorstellun-

[13] Die Schädigung des funktionierenden sozialen Miteinanders, die derart agierende Personen einer Gruppe zufügen können, ist für das Auftreten der kollektiven Intelligenz natürlich hinderlich. Dementsprechend ist beim Einsatz bzw. Auftreten dieser Mechanismen immer auch ein Abgleichen der entstehenden Vor- und Nachteile notwendig.

[14] Wird die Stellung einer Person innerhalb einer Gruppe zu stark bei der Betrachtung ihrer Ideen berücksichtigt, kann es zu Verfälschungen in der Beurteilung kommen, etwa durch den sogenannten *Halo*-Effekt. Dieser vom Psychologen Edward Lee Thorndike geprägte Begriff beschreibt eine Verzerrung der kognitiven Wahrnehmung, die sich darin äußert, dass von bekannten Eigenschaften eines Akteurs auf unbekannte Eigenschaften geschlossen wird. Das Phänomen tritt etwa auf, wenn eine mit Autorität ausgestattete Person eine Ansicht vertritt und Personen des nahen Umfelds geneigt sind dieser eher Glauben zu schenken. In Folge kann es zu einer Zentralisierung der Meinungsbildung kommen, die die Richtung der künftigen Entwicklungen einschränkt. Der *Halo*-Effekt wird oft als einer der Gründe genannt, warum zentrale Autoritäten im Bereich der kollektiven Intelligenz nicht geeignet sind.

gen vorherrscht bzw. die Fähigkeit, kontroversielle Einstellungen im Diskurs zulassen zu können. Dementsprechend korreliert die Fähigkeit der Gruppe, Probleme lösen zu können, in einem höheren Maß mit den in der Gruppe vertretenen sozialen Fähigkeiten[15], als mit der in der Gruppe aggregierten Intelligenz (gemessen mittels einer Summierung der einzelnen *IQs*) [42]. Kollektive Intelligenz ist somit nicht nur durch große Diversität und einer breiten Verteilung von Fähigkeiten definiert, sondern auch und vor allem durch eine reibungslose soziale Interaktion innerhalb der Gruppe. Interessant ist in diesem Zusammenhang, dass ein hoher Anteil an Frauen in einer Gruppe der Fähigkeit die richtigen Entscheidungen zu treffen *durchwegs* zuträglich ist [43] und zwar nicht wie im Sinne einer (wie es die initiale Interpretation sein könnte) Vielfalt auch auf geschlechtlicher Ebene, sondern vielmehr als Folge einer *direkten Korrelation* zwischen dem Frauenanteil in der Gruppe und dem Maß an kollektiver Intelligenz. Ein Erklärungsmuster für diesen Umstand kann sein, dass oftmals der Selbstbestätigungstrieb (siehe Abschn. 4.1.1) bei Frauen als einer auf die Bestätigung des Selbstwertes innerhalb eines sozialen Kontext Gerichteter gesehen wird (das bedeutet, dass das Selbst *auch* in einer Verbundenheit mit anderen Menschen existiert) und Männer beim Streben nach Selbstbestätigung stärker die eigene Unabhängigkeit bzw. ihr Verschiedensein zu den Gruppenmitgliedern betonen. Die Psychologin Cordelia Fine [4] beschreibt diesen Sachverhalt und sieht in der gesteigerten sozialen Fähigkeit von Frauen keine biologische Ursache, sondern vielmehr den Wunsch, einer von der Gesellschaft gegebenen Erwartungshaltung gerecht zu werden, eben der, empathisch gegenüber anderen zu sein. Dementsprechend wird in der Studie schließlich auch der Eindruck widerlegt, dass Gruppen die ausschließlich aus Frauen bestünden die ideale Konstellation für das Entfalten der kollektiven Intelligenz seien. Wie die an der Studie beteiligte Anita Woolley feststellt, sind diese weniger kooperativ als Gruppen, in die zumindest ein Mann integriert ist. Woolley schließt mit der Erkenntnis, dass die ideale Gruppenzusammensetzung nicht steuerbar ist, vielmehr gelte es der Selbstorganisationsfähigkeit der Gruppe Rechnung zu tragen und diese weniger steuern zu wollen, sondern sie vielmehr zu begünstigen und zu ermöglichen.

[15] Der Psychologe und Evolutionsforscher Heinz-Martin Süß wählt für diese den Begriff der *sozialen Kompetenz* und subsumiert in diesem die Begrifflichkeiten *soziale Intelligenz*, *emotionale Intelligenz* und einen großen Teil der *praktischen Intelligenz* [20]. Dies auch um die Zersplitterung der Begrifflichkeiten und die Notwendigkeit einer Zusammenführung und Vereinfachung dieser zu illustrieren.

Kapitel 5

Konzept und Idee

Im Rahmen der in Abschn. 1.3 definierten Zielsetzung soll nun im folgenden Kapitel ein Konzept zur Entwicklung eines Prototypen entwickelt werden. Zu diesem Zweck gilt es zunächst die Anforderungen an dieses Konzept zu definieren und die veranschlagten Lösungsansätze für diese zu formulieren. Im Abschn. 6.2 wird anschließend die derart entwickelte Terminologie und deren Bedeutung ausführlich beschrieben, die Implementierungsdetails folgen im Abschn. 6.3 und Abschn. 6.4 veranschaulicht schließlich den Prozessablauf der Umsetzung und die Kommunikationsschritte zwischen den einzelnen entwickelten Komponenten.

5.1 Anforderungen

Bei der Analyse des Einsatzgebietes der zu erstellenden Umsetzung wurden folgende allgemeine Kriterien definiert:

1. **Verfügbarkeit und Ressourcen**: Die Applikation soll in einem Umfeld eingesetzt werden, in dem möglichst viele Menschen und Ressourcen angesprochen werden können.
2. **Sicherheit und Zuverlässigkeit**: Da das System einer breiten Masse von Anwendern zugänglich gemacht wird, soll die Verwendung des Systems einerseits sicher für die Benutzer sein und andererseits gewährleisten, dass eine Manipulation der generierten Daten verhindert wird.
3. **Flexibilität**: Die Verwendung des Systems soll an einer möglichst großen Zahl an Rechnern möglich sein, es sollen also keine speziellen Anforderungen bezüglich der installierten Software oder der Leistungsfähigkeit des Computers gestellt werden. Weiters soll die Verwendung des Systems ohne eine fachspezifisches Verständnis der verwendeten Lernalgorithmen möglich sein.

Entsprechend den zuvor definierten allgemeinen Kriterien werden für die Umsetzung des Prototypen folgende Festsetzungen getroffen:

1. Da das dieser Arbeit zugrunde liegende System in einer *crowdsourcing*tauglichen Umgebung eingesetzt werden soll und um eine möglichst große, breit gestreute Personengruppe ansprechen zu können, wird im Konzept die Entwicklung einer *Webanwendung* festgesetzt. Den Problematiken, die sich etwa durch die hierbei verwendeten Webtechnologien ergeben[1], wird in der Folge der Konzeptentwicklung Rechnung zu tragen sein. Diese Herangehensweise bietet zusätzlich den Vorteil, dass die Anbindung an externe Quellen und *web services* zum Erfragen und Bearbeiten von Daten möglich ist.
2. Um eine Kompromittierung des Systems zu verhindern, soll eine Funktionalität implementiert werden, die jede einzelne Anfrage bezüglich ihrer Zugriffsberechtigungen überprüft und eventuelle nicht legitimierte Zugriffe in einer log Datei vermerkt. Für eine vollständige Auflistung der Sicherheitsmaßnahmen wird auf Abschn. 6.5 und Abschn. 6.3.1 verwiesen. Um die Validität der generierten Daten sicherzustellen, werden die von den *clients* generierten Daten mit den am *server* gespeicherten Validierungsdaten bei jedem Schritt überprüft. Für eine genaue Beschreibung dieses Mechanismus wird auf Abschn. 6.4 verwiesen.
3. Entsprechend der zuvor getroffenen Entscheidung, den Prototypen als Webanwendung zu konzeptionieren, ist es für die Anwender nicht notwendig, zusätzliche Komponenten zu installieren. Der *client*seitige Teil des Systems ist mittels des Browsers abrufbar. Der *server*seitige Teil der Implementierung wird mit der in den meisten Szenarien verfügbaren Scriptsprache PHP umgesetzt und als Datenbank MySQL gewählt. Mit dieser Entscheidung geht allerdings die folgende Problematik einher: Aufgrund der verringerten Performance, die zumeist durch PHP erreicht werden kann und den durch die Programmiersprache an sich gegebenen Einschränkungen, ist die Umsetzung der für das maschinelle Lernen notwendigen Algorithmen nur in eingeschränktem Maße möglich, zusätzlich ergibt sich das Problem einer schlechteren Skalierbarkeit. Um diesem Problem zu begegnen, wird der für die Handhabung besagter Algorithmen notwendige Bereich an ein externes Service, das Google Prediction Service ausgelagert. Diese Herangehensweise bietet folgende Vorteile:
   (a) Es ist nicht der Einsatz eines hochperformanten *servers* notwendig. Die für das maschinelle Lernen benötigte Rechenleistung wird durch die von Google zur Verfügung gestellten clustern geliefert.

[1] Wie eine eingeschränkte Funktionalität und Performance im Vergleich zu *Desktopanwendungen* aber auch die Notwendigkeit einer Reihung und Strukturierung der Kommunikation multipler *clients* mit einem *server*.

(b) Das System skaliert auch bei einer intensiven Verwendung ausreichend.

(c) Durch die *black box* der Google Prediction API (das Service ermittelt selbst aus den ihm zur Verfügung gestellten Daten den besten Algorithmus) ist eine fachspezifische Kenntnis der verwendeten Algorithmen nicht notwendig.

Für die Funktionalität des Systems werden folgende Anforderungen definiert:

1. Das System soll die Fähigkeit besitzen, Modelle für verschiedene Datentypen und Vorhersagemechanismen bzw. mehrere Modelle für denselben Datentyp und verschiedene Entwicklungsstadien dieser Modelle verwalten können. Diese Entwicklungsstadien werden als *Modellversion* bezeichnet (siehe Abschn. 6.2.1).
2. Das System soll eine flexible Möglichkeit besitzen, Funktionalitäten zu definieren, um die Attribute der Daten extrahieren bzw. neue Attribute generieren zu können. Die hierfür einem Modell zugewiesenen Funktionen werden *Modelfunctions* genannt (siehe Abschn. 6.2.1).
3. Das System soll die Wiederverwertbarkeit der bewerteten Originaldatensätze ermöglichen, um diese für andere Modelle bzw. alternative Modelle (etwa zur Validierung und zum Vergleich) verwenden zu können. Dies macht die Trennung der Originaldaten und der aus diesen generierten Attribute notwendig. Die Orginaldaten werden in der folgenden Terminologie als *Sources* bezeichnet, die Speicherung der Attribute (bzw. Parameter) erfolgt in den sogenannten *Predictions* (siehe Abschn. 6.2.2).
4. Das System soll im Sinne der Dezentralisierung das Generieren bzw. Extrahieren der die Datensätze repräsentierenden Attribute an die *clients* delegieren können. Diese Aufgaben werden in der verwendeten Terminologie als *Jobs* bezeichnet (siehe Abschn. 6.2.4), deren Erteilung, Verarbeitung und Rückgabe während der *Harvest* Phase erfolgt. Diese wird in Abschn. 6.4.2 beschrieben.[2]
5. Das System soll es den Anwendern bzw. der *crowd* ermöglichen, kontinuierlich neue Datensätze bewerten zu können bzw. eine Möglichkeit bieten, die Meinung der *crowd* zu aggregieren und damit eine Validierung der Bewertung dieser Daten und infolge des Modells durchführen zu können. Die durch einen Anwender abgegebene Bewertung wird in Folge als *Userlabel* bezeichnet, die Gesamtheit aller Bewertungen eines Datensatzes als *Status* (siehe Abschn. 6.2.5).

Um eine weitgehend flexible Verwendung zu ermöglichen, werden die in diesem Konzept vorgestellten Funktionalitäten als Bibliothek umgesetzt. Mittels dieses Ansatzes soll es den Entwicklern ermöglicht werden, die automa-

[2] Arbeitsschritte auf Seiten des *servers* werden hingegen als *Tasks* bezeichnet, deren Bearbeitung in der *Task* Phase (Abschn. 6.4.1) erfolgt.

tisierte Klassifizierung von Inhalten bzw. das Generieren von Trainingsdaten für den Klassifizierungsalgorithmus in Plattformen und *CMS* Systeme ihrer Wahl zu integrieren. Eine exemplarische Integrierung in das *CMS Wordpress* wird dementsprechend nach der Erstellung der Bibliothek durchgeführt und diese auch zum Zweck der abschließenden Evaluierung verwendet.

Kapitel 6

Projektumsetzung

Die in den Abschn. 6.1 und 6.4 erläuterten Begriffe, Funktionalitäten und Abläufe werden zum Zwecke der besseren Verständlichkeit mit Beispielen aus einem Szenario, indem es gilt, Postings als *Spam* oder zur Thematik passende Einträge (*Ham*) zu identifizieren, versehen. Diese Beispiele sind als auf die wesentlichen Aspekte reduziert anzusehen.[1] Weiters erfolgt im Abschn. 6.3 eine schematische Beschreibung des implementierten Sourcecodes und eine Auflistung der entsprechenden Funktionalitäten, Kommentare und Begründungen zur Implementierungsweise.

6.1 Verwendete Sprachen und Technologie

Eines der wesentlichsten Kriterien bei der Auswahl der verwendeten Technologien, ist deren allgemeine, breite Verfügbarkeit und die Möglichkeit diese in einem alltäglichen Szenario verwenden zu können, ohne die Notwendigkeit zusätzliche Gegebenheiten auf den Rechner zu schaffen bzw. Installationen vornehmen zu müssen. Für die vorliegende Projektumsetzung wurde die Auswahl dahin gehend getroffen.

1. ***Client*seitige Entwicklung**:
   - Javascript,
   - JQuery 1.8.2 und JQuery UI 1.8.23.
2. ***Server*seitige Entwicklung**:
   - PHP 5.3.0,
   - MySQL,
   - RedBeanPHP 3.3.[2]

[1] Als wesentlich werden nur die für das prinzipielle Verständnis notwendigen Strukturen und Abläufe angesehen. Für die vollständigen Implementierungsdetails wird auf den dokumentierten Quellcode der Projektumsetzung verwiesen.

[2] http://redbeanphp.com/

Die Umsetzung wurde auf den folgenden Browsern getestet:

1. Internet Explorer,
2. Firefox Mozilla,
3. Safari,
4. Chrome,
5. Opera.

Diese Tests werden später noch im Kapitel 7 ausführlich beschrieben.

6.2 Terminologie

6.2.1 Modell, Modellversion und Modellfunktion

Modell

Ein Modell beinhaltet in seinem Kern eine fest definierte Verkettungsstruktur von Funktionen, anhand derer aus einer Anzahl von Eingangswerten (*Source*) eine Anzahl von Ausgangswerten (*Prediction*) generiert werden (siehe Abschn. 6.2.2). Modelle dienen als eine Art Filter, um die wesentlichen, für die Generierung einer Vorhersage notwendigen Parameter aus einer Reihe von *Noise*[3] behafteter Einträge zu extrahieren. Wird ein Modell mit einem Trainingsdatensatz (siehe Abschn. 6.2.3) trainiert, so wird eine neue Modellversion (siehe Abschn. 6.2.1) generiert. Multiple Versionen entsprechen somit verschiedenen Entwicklungszuständen des Vorhersagemechanismus eines Modells. Weitere sind in einem Modell folgende Eigenschaften definiert:

- **Eingangswerte**: Typ und Anzahl der durch die Funktionskette zu verarbeitenden Eingangsparameter (das bedeutet die Werte und deren Typ einer entsprechenden *Souce* mit diesen übereinstimmen müssen).
- **Ausgangswerte**: Typ und Anzahl der durch das Modell generierten Ausgangswerte, wobei diese den generierten Parametern[4] einer *Prediction* entsprechen.
- **Klassifizierungen**: Definition der möglichen, durch den Vorhersagealgorithmus generierten Klassifizierungen.

Sowohl Eingangs- als auch Ausgangswerte können vom Typ *string*, *number* oder *boolean* sein.

Ein Modell wie jenes im Beispielszenario könnte somit etwa folgende Eigenschaften besitzen:

[3] In dem vorliegenden Szenario können etwa zahlreiche Informationen in einem Posting enthalten sein, die für die Zuverlässigkeit einer Aussage entweder nicht von Nutzen oder sogar hinderlich sein können, wie etwa *stopwords*, der exzessive Einsatz von Satzzeichen oder eine wechselnde Groß- und Kleinschreibung, da etwa die Wörter *STOP* und *stop* zwei verschiedene Werte für den Vorhersagealgorithmus darstellen.

[4] In der Terminologie des maschinellen Lernens entsprechen diese Parameter den Werten des *Eingangsvektors*.

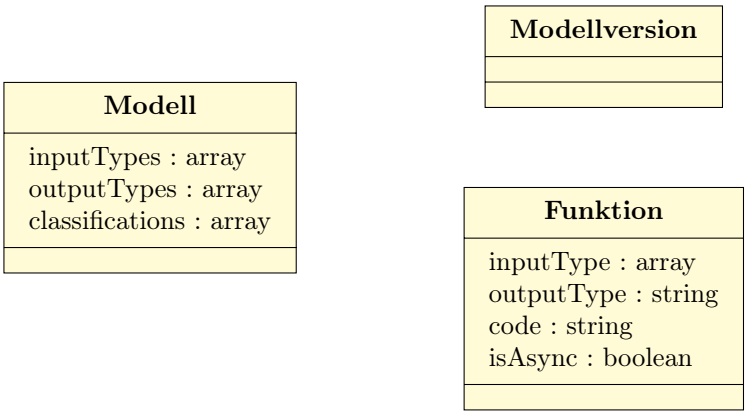


Abbildung 6.1: Beziehung zwischen einem Modell und den ihm zugeordneten Modellversionen und Funktionen.

- **Eingangswerte**
  1. Postingtext (vom Typ *string*).
  2. *URL* zur Überprüfung der im Posting enthaltenen Verweise mittels der *Google Safe Browsing API* (vom Typ *string*).
- **Ausgangswerte**
  1. Die Anzahl der im Posting enthaltenen Verweise (vom Typ *number*).
  2. Anzahl der als schädlich identifizierten Links (vom Typ *number*).
  3. Der aufbereitete Text des Postings (vom Typ *string*).
- **Klassifizierungen**
  1. Array mit den Werten *spam* und *ham*.

Für ein Beispiel der derart gestalteten Eingangsdaten wird auf Tab. 6.2 verwiesen (der erwähnte Link auf die *Google Safe Browsing API* wird wie im Prog. 6.1 ersichtlich manuell definiert) und für die daraus generierten Ausgangswerte auf Tab. 6.3.

Modellversion

Eine Modellversion repräsentiert *einen* trainierten Zustand eines Modells.[5] Ist das Training einer Modellversion mithilfe der initialen oder durch die Anwender generierten Trainingsdaten (siehe Abschn. 6.2.3) abgeschlossen, ermöglicht sie die Klassifizierung von neuen, unbewerteten Einträgen.

[5] In der Terminologie des maschinellen Lernens wird hier von einer *Klassifizierungsfunktion* gesprochen, also einer Funktion die versucht alle ihr zur Verfügung gestellten Trainingsdaten und deren entsprechende Resultate abzubilden.

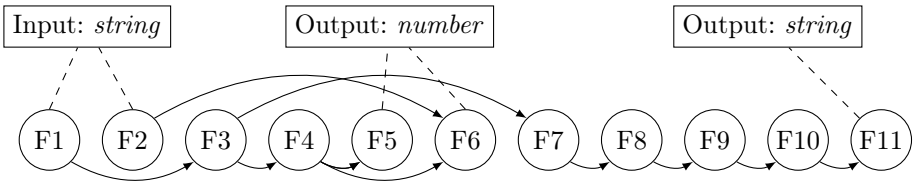


Abbildung 6.2: Verkettung der in Tab. 6.1 aufgelisteten Funktionen.

Modellfunktion

Modellfunktionen stellen die einzelnen Arbeitsprozesse dar, die gemäß der Modellspezifikationen in einer eindeutig definierten Reihenfolge mit Eingangsparametern durch das Modell aufgerufen werden und einen oder mehrere definierte Rückgabewerte generierten. Eine Modellfunktion beinhaltet folgende Eigenschaften:

- **Eingangswerte**: Typ und Anzahl der durch die Funktion zu verarbeitenden Eingangsparameter.[6]
- **Ausgangswert**: Typ eines durch die Funktion generierten Ausgabewertes.[6]
- **Code**: Javascript Programmcode, der beim Laden der Funktion vonseiten des *clients* validiert, kompiliert und im Falle einer Asynchronität mit einem entsprechenden *callback* versehen wird.
- **Synchronitätsverhalten**: Ist eine Funktion asynchroner Natur[7], wird der normale Ablauf der Funktionskette des Modells an dieser Stelle unterbrochen und mittels eines *callbacks* der Ablauf nach erfolgreichem Rückgabewert fortgesetzt.

Zum besseren Verständnis des Zusammenspiels der einem Modell zugeordneten Funktionen, ist in Tab. 6.1 eine schematische Auflistung der im Beispielszenario verwendeten Modellfunktionsstruktur angeführt. Eine grafische Darstellung dieser Verkettung findet sich in Abb. 6.2.

6.2.2 *Source* und *Prediction*

Dem Begriff *Source* bezeichnet einen zu bewertenden Eintrag oder Datensatz, als *Prediction* wird die Summe der aus diesem Datensatz generierten Parameter sowie eine durch das System generierte Klassifizierung bezeichnet. In Tab. 6.2 werden zwei als *Source* gespeicherte Postings und in Tab. 6.3 die

[6] Im Gegensatz zu den im Modell existierenden Typen *string*, *number* und *boolean* können Funktionen auch multiple Ein- und Ausgabewerte des entsprechenden Typs handhaben. Dies wird in Folge durch die Kennzeichnung *single* und *array* gekennzeichnet (also etwa *single string*).

[7] Etwa um externe Ressourcen wie Webservices abzufragen.

Tabelle 6.1: Funktionen des *Spam Identifikations Modells*.

ID	*Aufgabe*	*Eingangswerte*	*Ausgangswert*
F1	Einspeisen des Postingtextes.	*single string*	*single string*
F2	Einspeisen des *Google Safe Browsing API Endpoints*.	*single string*	*single string*
F3	Dekodieren aller *HTML Entities*.	*single string*	*single string*
F4	Extrahieren aller validen URLs.	*single string*	*array string*
F5	Rückgabe der Länge eines Arrays.	*array string*	*single number*
F6	Ermitteln der schädlichen Links.	*single string array string*	*single number*
F7	Entfernen aller *HTML Tags*.	*single string*	*single string*
F8	Entfernen der Satzzeichen.	*single string*	*single string*
F9	Entfernen aller Wörter kürzer als 4 Zeichen.	*single string*	*single string*
F10	Text in Kleinbuchstaben umwandeln.	*single string*	*single string*
F11	*Whitespace* entfernen.	*single string*	*single string*

Tabelle 6.2: *Sources* mit unverändertem Text.

ID	*Text*
1	great page...im sure i'll come back...best regards Alex
2	Ok, I will sign your blog. I really love your site. <a http://somemalewarelink.com>Casino

entsprechenden, mittels des *Spam Identifikations Modells* generierten *Predictions* exemplarisch dargestellt.

Eine *Source* kann aus einer beliebigen Anzahl an Werten bestehen, die jeweils vom Typ *string* oder *number* sein müssen. So haben *Sources* des vorliegenden Szenarios etwa als Eingangsparameter zwei Textfelder, wobei der erste Parameter der unveränderte Text des Postings und der zweite Parame-

Tabelle 6.3: Aus *Source* Einträgen generierte *Predictions* mit Klassifizierung und den Parametern URLs *(Anzahl der URLs)*, BadURLs *(schädliche links)* und dem aufbereiteten Text.

ID	*Class*	*URLs*	*BadURLs*	*formatierter Text*
1	ham	0	0	great page sure come back best regards alex
2	spam	1	1	will sign your blog really love your site casino

Programm 6.1: Speichern eines Postings as *Source*.

```
1       $model = ModelManager::getModel(array("unique_string" => $modelID)
      );
2       ...
3       $sourceValues = array(array($postText, "string"), array(
      $safebrowsingEndpoint, "string"));
4       $source = PUtility::createBean("source", array($uniqueSourceID,
      $is_validation, $is_inital, $sourceValues));
5       R::store($source);
6
```

ter eine Zieladresse ist, unter der die extrahierten *URLs* mittels der *Google Safebrowsing API* bezüglich einer eventuellen Schädlichkeit einer Überprüfung unterzogen werden. Weiters können *Source* Einträge als Validierungsdaten[8] bzw. als zum initialen Trainingsdatensatz gehörig markiert werde. Eine vereinfachte Darstellung des Speichervorgangs eines solchen Postings wird in Prog. 6.1 gezeigt.

Predictions entsprechen den systeminternen, aufbereiteten Repräsentationen einer *Source* und der durch das System vorgenommenen Klassifizierung. Die in Tab. 6.3 angeführten Werte *URLs*, *BadURLs* und *formatierter Text* sind adäquat zu den im Maschinellen Lernen bezeichneten *input vector features* und bilden die durch das *Modell* extrahierten, für eine Bewertung als maßgeblich erachteten Werte. Eine *Prediction* ist somit immer genau mit einer *Source* und einem *Modell* verknüpft. *Predictions* können des weiteren mit *Jobs* versehen werden (siehe Abschn. 6.2.4).

6.2.3 Trainingsdaten

Um Einträge mittels eines Modells einer Klassifizierung unterziehen zu können, ist es, wie eingangs bereits erwähnt, notwendig eine Modellversion mit

[8] Validierungsdaten werden nicht zum Trainieren neuer Modellversionen herangezogen und dienen in der Evaluierungsphase zur Bewertung der Zuverlässigkeit dieser.

entsprechenden Trainingsdaten zu trainieren. Diese Datensätze stellen eine Anzahl von bereits klassifizierten Einträgen dar und existieren in der vorliegenden Projektimplementierung in zwei verschiedenen Formen:

- **Initiale Trainingsdaten**: Um den Vorhersagealgorithmus eines Modells initial mit Daten zu füttern, wird ein bereits klassifizierter Datensatz (der einer beliebigen Anzahl von bewerteten *Source* Einträgen entspricht) in die Applikationslogik eingespielt, *client*seitig entsprechende *Predictions* daraus generiert und anhand derer ein zum Training der anfänglichen Modellversion verwendeter Trainingsdatensatz erzeugt.[9]
- **Anwendergenerierte Trainingsdaten**: Um eine neue Modellversion zu erzeugen, werden die existierenden *Predictions* gemäß der durch die Anwender erfolgten Bewertung (siehe Abschn. 6.2.4) überprüft, gegebenenfalls neue Klassifizierungen zugewiesen[10] und ein neuer Trainingsdatensatz erzeugt. In Folge findet sich ein exemplarischer Auszug der im Beispielszenario generierten Trainingsdaten im *CSV* Format.

```
1  "spam",2,1,"very useful comments good read online casinos online
     casino choice"
2  "spam",1,0,"excellent that really well explained helpful business
     grants"
3  "ham",0,1,"difference degree becomes difference kind perhaps that
     precisely what long trail graph depicts"
```

6.2.4 *Job* und *Task*

*Client*seitiger *Job*

Um einen gespeicherten *Source* Eintrag mittels eines Modells in eine *Prediction* umwandeln zu können, ist es notwendig, diesen mit einem *Job* zu versehen. *Jobs* stellen Aufgaben dar, die von der *server*seitigen Applikationslogik an den *client* delegiert werden. Es existieren zwei verschiedene Arten:

- **Parametergenerierung eines klassifizierten Eintrags**: Ist ein Eintrag bereits bewertet (etwa durch Einspielen eines klassifizierten, initialen Trainingsdatensatzes), so werden die mit diesem Auftragstyp versehenen Einträge an den *client* übermittelt, die entsprechenden *features* durch die im Modell definierte Logik extrahiert, im Erfolgsfall dieses Resultat an den *server* zurückgegeben und von diesem in der Form einer *Prediction* gespeichert. Applikationsintern wird diese Art des Auftrags als *Input Job* bezeichnet.
- **Klassifizierung und Parametergenerierung**: Unbewertete Einträge (im Beispielszenario etwa neue Postings die von der Applikation

[9] Für eine detaillierte Ablaufbeschreibung dieser Prozesse wird auf Abschn. 6.4 verwiesen.

[10] Diese *Neubewertung* erfolgt durch ein Vergleichen der existierenden Bewertung mit den aufsummierten und entsprechend gewichteten Bewertungen der Anwender.

als *Source* Einträge gespeichert werden) können mittels dieses, intern als *Prediction Job* bezeichneten Typs einer Klassifizierung unterzogen werden. Die Generierung der *Prediction features* erfolgt in einer dem zuvor genannten Auftragstyps entsprechenden Weise, erweitert um eine Klassifizierungsanfrage der *server*seitigen Applikationslogik im Falle einer erfolgreichen Durchführung des Auftrages.

Erteilte Aufgaben werden für eine durch das Modell definierte Zeitdauer blockiert. Wird ein *Job* nicht innerhalb dieser Frist vom *client* an den *server* zurückgeliefert, wird der erteilte Auftrag gelöscht und an einen anderen Anwender delegiert. Erfolgt die Rückgabe der abgearbeiteten Aufgabe innerhalb der Frist, wird das Resultat validiert, der *Job* vom entsprechenden Eintrag entfernt und die Dauer des Prozesses sowie einige anwenderspezifische Daten für spätere Evaluierungszwecke gespeichert. Die Anzahl der in einem Schritt erteilten Aufträge richtet sich nach der Art des *Jobs* und den am *client* vorhandenen Javascript *features*.[11] *Prediction Jobs* werden bei der Aufgabenerteilung gegenüber *Input Jobs* bevorzugt behandelt, liegt ein *Prediction Jobs* mit der *ID* des auf dem *client* aktuell angezeigten Eintrags vor, so wird diese präferiert.

*Server*seitiger *Task*

Arbeitsschritte, die nicht vom *client* durchgeführt werden können und deren Abarbeitung ausschließlich dem *server* obliegt, werden als *Tasks* bezeichnet.[12] Bei jeder Anfrage des *clients* an den *server* wird überprüft, ob aktuell ein *Task* vorliegt und wenn dies der Fall ist, dessen Abarbeitung initiiert. *Tasks* dienen primär dazu eine zu lange Scriptlaufzeit am *server* zu vermeiden und längere, komplexere Arbeitsschritte in kleine Einheiten zu unterteilen.

6.2.5 *Userlabel* und *Status*

Userlabel

Um die Richtigkeit einer Vorhersage durch die Applikationslogik zu prüfen, haben Anwender die Möglichkeit Einträgen eine der gemäß des Modells definierten Klassifizierungsmöglichkeiten zuzuweisen, die sie für die treffende erachten. Diese Bewertungen, als *Userlabels* bezeichnet, können grundsätzlich in zwei verschiedene Klassen unterteilt werden:

[11] Verfügt der *client* etwa die Möglichkeit mittels des *Html5 WebWorkers* multiple Javascript Funktionen in einer threadähnlichen Weise parallel auszuführen, wird eine größere Zahl von *Jobs* erteilt.

[12] Das Erstellen einer neuen Modellversion beinhaltet etwa das Generieren einer neuen *CSV* Trainingsdatei, dem Hochladen dieser Datei auf *Google Storage*, dem Initiieren des Trainingsprozesses mittels der *Google Prediction API* und dem Pollen des aktuellen Trainingszustandes. Eine genaue Ablaufbeschreibung findet sich im Abschn. 6.4.

Anonyme Bewertungen: Der die Bewertung abgebende Anwender ist nicht angemeldet.

Authentifizierte Bewertungen: Die Bewertung wird von einem angemeldeten Anwender abgegeben.

Authentifizierte Bewertungen werden immer einem Anwender zugeordnet. Diese können beliebig oft geändert werden, jede Änderung wird aber zu Evaluierungszwecken gespeichert und die zuletzt abgegebene Klassifizierung als die dem Eintrag Zugeordnete angezeigt.

Status

Als *Status* bezeichnet man die Gesamtheit der einem Eintrag zugeordneten Bewertungen und der aktuellen Klassifizierung, welche durch die drei folgenden Prozedere entstehen kann:

1. Durch die im *initialen Datensatz* definierte Klassifizierung.
2. Durch den Vorhersagealgorithmus generierte Bewertung bei *zur Laufzeit gespeicherten Einträgen.*
3. Durch eine Analyse der vorhandenen *Userlabels* und der bestehenden Klassifizierung bei der *Erstellung einer neuen Modellversion.*

6.3 Programmiertechnische Umsetzung

Die der vorliegenden Arbeit zugrunde liegende Umsetzung besteht aus den folgenden 3 Bereichen:

1. Die Funktionsbibliothek (im Projekt als *Prediction Library* bezeichnet).
2. Eine *standalone* Applikation zum Erstellen und Verwalten von Modellen.
3. Eine exemplarische Integrierung in das *content management system Wordpress.*

Die Beschreibung der Abschn. 6.3.1 und 6.3.2 besteht aus folgenden Teilabschnitten:

1. **Funktionsweise**: Eine grundsätzliche Beschreibung der Funktionsweise.
2. **Detailbeschreibung** / *client*seitig bzw. *server*seitig: Eine Detailbeschreibung der erstellten Dateien und der in ihnen enthaltenen Funktionalitäten.

Für die Integrierung in *Wordpress* wurde das in dieser Arbeit zu Veranschaulichungszwecken verwendete *Spam* Klassifizierungsmodell erstellt und verwendet. Eine Darstellung der fertigen Umsetzung ist in Abb. 6.3, Abb. 6.4 und Abb. 6.5 ersichtlich. Aufgrund des Umfangs dieser Umsetzung können nicht alle Funktionalitäten und Implementierungsdetails angeführt werden.

Abbildung 6.3: Der Administrationsbereich der *Wordpress* Integrierung.

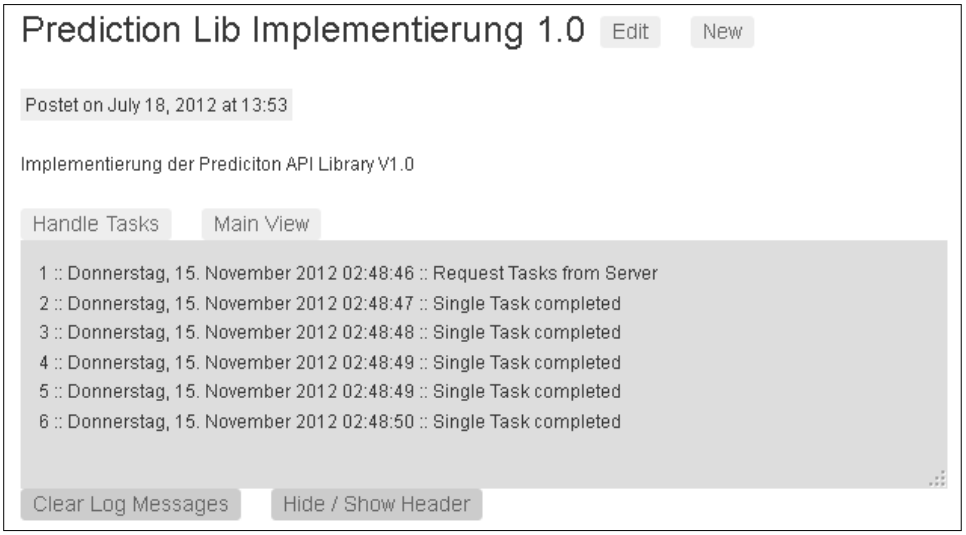


Abbildung 6.4: Die *Wordpress* Integrierung in der *Taskphase*.

Es wird dementsprechend versucht sich auf die wesentlichen Aspekte zu konzentrieren, Dateien, die für das Allgemeinverständnis nicht notwendig sind (d.h. auch *CSS* Dateien und verwendete Bilder), werden nicht angeführt. Besonderes Augenmerk wurde bei der Entwicklung auf folgende Aspekte gelegt:

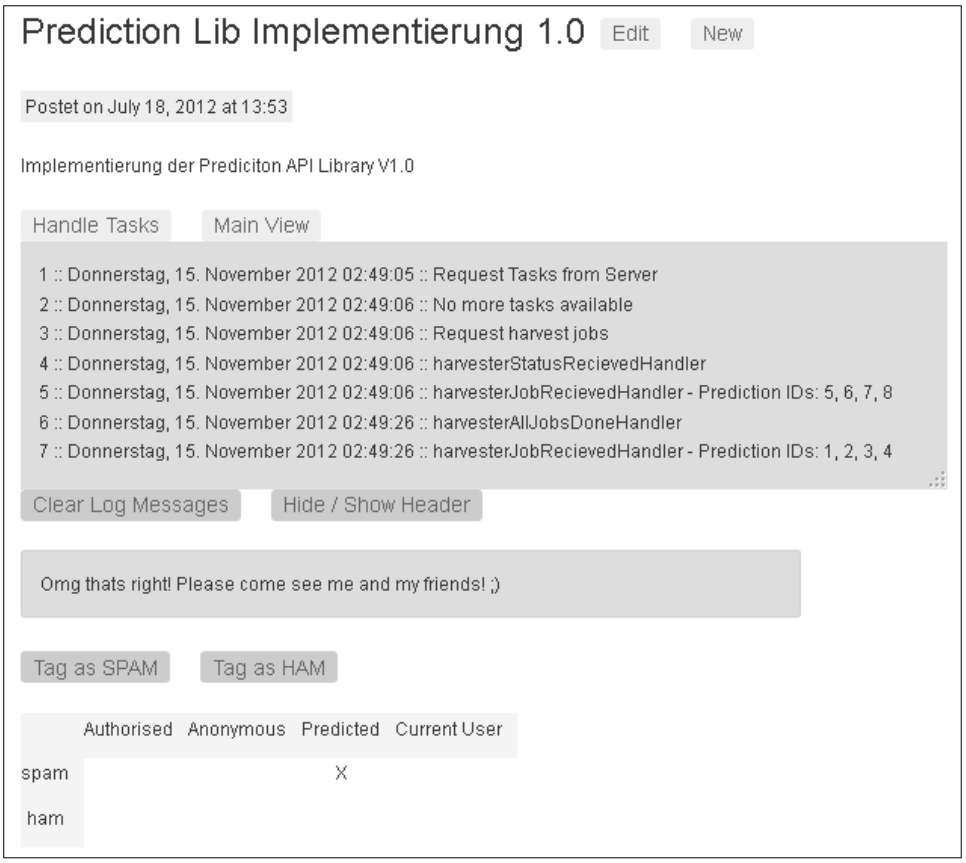


Abbildung 6.5: Die *Wordpress* Integrierung in der *Harvestphase*.

*Client*seitige Umsetzung

1. Geschwindigkeit und Speicherbedarf:
   - Durch die Vermeidung von *closures* innerhalb von Schleifen und strikter Beachtung der korrekten Javascript *scope chain* wird der Speicherbedarf minimiert und *leaks* vorgebeugt.
   - Durch die parallele Verwendung von Arrays und Javascript Objekten in Strukturen, etwa in den durch die Funktion *construct_ Map* erzeugten Listen (siehe Abschn. 6.3.1), wird der verlangsamte Zugriff auf Eigenschaften von Javascript Objekten (wie er in manchen Browsern auftritt) umgangen.
   - Durch das Verwenden von statischen Eigenschaften und Methoden der Klassen wird ein unnötiger Speicherverbrauch vermieden.
   - Die Resultate aller durchgeführten Operationen (z.B. Validierung einer Funktionskette, die *topologische* Sortierung derselben oder das Generieren eines Funktionsobjekts aus dem geladenen Funk-

tionstext) werden im *cache* abgelegt und bei Bedarf von dort geladen.

2. Kapselung:
   - Durch ein Definieren von sensiblen Parametern innerhalb einer *closure scope chain* werden diese *privat* gehalten. Der Zugriff erfolgt mittels *Gettern* und *Settern*.

_Server_seitige Umsetzung

1. Sicherheit:
   - Jede vom *client* an den *server* versendete Anfrage wird bezüglich der Zugriffsrechte des Anwenders validiert (siehe Abschn. 6.5).
   - Durch das Verwenden von *prepared statements* sowie dem Entfernen von schädlichen Elementen und der Überprüfung mittels eines *whitelist* Ansatzes[13] werden die für den Eintrag in die Datenbank bestimmte Daten des *clients* sicher gehandhabt.
   - Die Tabellen, welche die von der Datenbank zu ladenden Funktionen (bzw. den Funktionstext) beinhalten, werden durch einen separaten *MySQL user* verwaltet, d.h. der vom *client* anfragende Anwender hat nur Leserechte für diese Tabelle.
   - Potentiellen Angriffsmöglichkeiten wie dem *session hijacking* oder dem *session fixation* wird durch Sicherheitsmaßnahmen Rechnung getragen.
2. Vermeidung von Kollisionen:
   - *Inkonsistenzen* in der Datenbank, durch etwa zeitgleich gelieferte Anfragen oder Antworten des *clients* werden mit den folgenden Mechanismen verhindert:
     - Reihung der gesamten Anfrage bzw. Antwort mittels einer *queue*.
     - Sperren einer Datenbank oder einer einzelnen Tabelle und Reihung der Zugriffsanfragen (inklusive eines *commit / rollback* Mechanismus).
   - Implementierung eines Mechanismus um zu verhindern, dass vom selben *client* multiple *browser* oder *browser* Instanzen mit dem *server* kommunizieren.

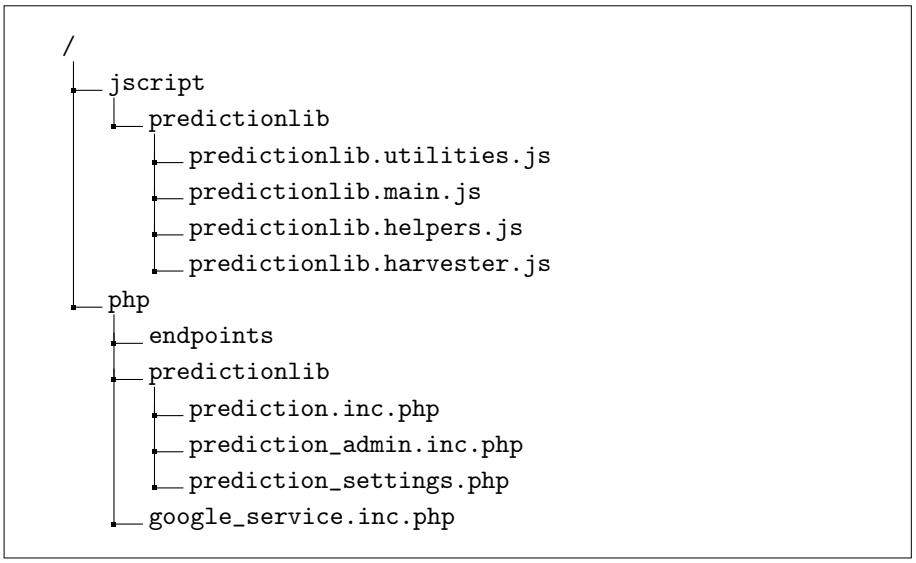


Abbildung 6.6: Dateien der Funktionsbibliothek. Die verwendeten *ajax endpoints* werden aufgrund von Platzgründen weder in dieser Liste noch in der Detailbeschreibung angeführt. Im wesentlichen werden diese durch die in Abschn. 6.4 dargestellten Prozessschritte bzw. jede dort angeführte Anfrage des *clients* an den *server* beschrieben, so etwa das Laden von Modellen und Funktionen, das Erfragen von *Task*, dem Status oder *Jobs*.

6.3.1 Funktionsbibliothek

Funktionsweise

Die in der Funktionsbibliothek (umsetzungsintern als *Prediction Library* bezeichnet) enthaltenen Klassen, *factories* und Funktionen sind das eigentliche Herzstück der Anwendung, eine detaillierte Beschreibung der Funktionsweise ist im Abschn. 6.4, die Filestruktur in Abb. 6.6 ersichtlich.

Detailbeschreibung / *client*seitig

- **predictionlib.utilities.js**: Sammlung von *utility* Klassen und Konstruktorfunktionen der Funktionsbibliothek.
  - **construct_PredictionError** / Funktion: Erzeugt ein Javascript Fehlerobjekt, welche die Art des Fehlers, den Ort des Auftretens und weitere Informationen beinhaltet.
  - **construct_Map** / Funktion: Erzeugt eine Liste mit *gettern*, *settern* und Hilfsmethoden. Verwendet zur Steigerung der Zugriffsgeschwindigkeit parallel ein Array (zum Speichern der Indizes)

[13] Als *whitelist* bezeichnet man eine Liste von *erlaubten* Zeichen. Der gegensätzliche Ansatz, alles zu verbieten was nicht explizit erlaubt ist wird als *blacklist* Ansatz bezeichnet.

und Objekt (zum Speichern der Werte).

- **construct_FunctionQueue** / Funktion: Erzeugt eine verkettete Liste von Funktionsobjekten und führt die in diesen enthaltenen Funktionen nacheinander aus. Der Ablauf dieser Exekution ist in Abb. 6.2 ersichtlich. An jedem Abfolgeschritt werden die generierten Werte gespeichert und in die folgenden Funktionen gemäß der definierten Verkettungsstruktur eingespeist. Ist die entsprechende Funktion *synchron*, wird direkt die nächste Funktion aufgerufen, ist sie *asynchron* wird der Ablauf bis zum Aufruf des entsprechenden *callbacks* unterbrochen. Die Liste selbst erlaubt das Definieren zweier *callbacks*, mit denen folgende Phasen der Funktionsaufrufe kontrolliert werden können:

  1. Vor dem Aufrufen der Funktion und dem Einspeisen der Funktionsparameter.
  2. Nach dem erfolgreichen Ausführen und generieren des Rückgabewertes.

  Die einzuspeisenden Funktionsparameter bzw. der generierte Rückgabewert wird hierbei an den entsprechenden *callback* übergeben. Mittels dieser Funktionalität können etwa Abbruchbedingungen geprüft und Werte geändert werden. Die Resultate der für die Ausgangswerte des Modells zuständigen Funktionen werden gespeichert und als Array an den Aufrufer zurückgegeben.
- **Settings** / Klasse: Speichert die *endpoint URLs*, die *IDs* der verwendeten Modelle sowie die *client* Daten, welche bei jeder Anfrage an den *server* identisch sind (wie etwa den in Abschn. 6.5 beschriebenen *MD5 hash*).
- **Utility** / Klasse: Sammlung von Hilfsmethoden, die etwa bei der Ausgabe von Fehlern und Nachrichten an den Anwender bzw. der Validierung von durch *Ajax*anfragen übermittelten Daten verwendet werden.
- **AjaxUtil** / Klasse: Kapselung der *ajax* Funktionalitäten sowie Fehler- und Ablaufhandhabung.

- **predictionlib.main.js**: Definition der primären Funktionalitäten, Klassen und *factories* der *Prediction Library*.
  - **FunctionMap** / Klasse: Speichert die von der Datenbank geladenen Funktionen (siehe Abb. 6.7), die zuvor mittels der *FunctionFactory* in Funktionsobjekte umgewandelt wurden.
  - **ModelMap** / Klasse: Speichert die von der Datenbank geladenen Modelle (siehe Abb. 6.8), die zuvor mittels der *ModelFactory* in Modellobjekte umgewandelt wurden.
  - **ToposortManager** / Klasse: Mittels dieser Manager Klasse wird

eine Reihe von Funktionsobjekten *topologisch* sortiert, das bedeutet in einer Reihenfolge, die gewährleistet, dass alle definierten Abhängigkeiten der Verkettungsstruktur erfüllt sind. Dieser Vorgang lässt sich am besten durch eine Verletzung dieser Abhängigkeiten veranschaulichen. Wenn die Funktionen A, B und C gegeben sind und B die durch A generierten Werte benötigt sowie C die durch B generierten Werte wäre folgende Verkettungsstruktur zulässig:

$$A \rightarrow B \rightarrow C$$

Würde allerdings diese Struktur geändert, käme es zu einer Verletzung dieser Abhängigkeiten:

$$A \rightarrow C \rightarrow B$$

Dies ist natürlich nur ein einfaches Beispiel zu Illustrationszwecken. In der Praxis sind diese Strukturen wesentlich komplexer und deren *topologische* Sortierung relativ kompliziert.

- **ModelManager** / Klasse: Lädt Modelle und die benötigten Funktionen (sofern diese nicht bereits in der *FunctionMap* enthalten sind). Der *ModelManager* ist eine der zentralen Klasse dieses Bereichs (der Funktionsbibliothek) und kapselt alle Bereiche, die zum Ausführen der Modellfunktionen notwendig sind.
- **FunctionFactory** / Klasse: Generiert aus den von der Datenbank geladenen Funktionswerten Funktionsobjekte. Die *FunctionMap* bedient sich etwa dieser Klasse. Neben den Validierungsoperationen generiert die *FunctionFactory* aus dem geladenen *code* eine ausführbare Javascript Funktion, in die die Eingangsparameter und (wenn es sich um eine *asynchrone* Funktion handelt) der *callback* eingeschliffen werden.
- **ModelFactory** / Klasse: Erzeugt aus den geladenen Modelldaten Modellobjekte, welche in der *ModelMap* gespeichert werden. Die *ModelMap* ist es auch, die die geladenen Funktionen einer *topologischen* Sortierung unterzieht.

- **predictionlib.helpers.js**: Funktionssammlung, die sowohl zum Testen als auch im direkten Betrieb der Applikation verwendet wird. Sie beinhaltet vor allem *wrapper* Funktionen um Befehle konsistent auf den verschiedensten Browsern ausführen zu können und Hilfsfunktionen zum Manipulieren von Listen, Objekten und Arrays.
- **predictionlib.harvester.js**: Definition aller Klassen, die für das Erfragen von *Jobs* durch den *client* bzw. deren Verarbeitung und Rückgabe an den *server* zuständig sind und der Handhabung von *Userlabel* bzw. *Status* eines *Source* Eintrages. Das Aktivieren von *Tasks* wurde (aufgrund weitgehender Überschneidungen mit der Funktionalität der *Job* Handhabung) ebenfalls in dieser Datei implementiert.

  - **Harvester** / Klasse: Zuständig für das Erfragen von *Jobs* durch den *client*, der Bearbeitung und Rückgabe dieser an den *server*. Weiters verantwortlich für das Verarbeiten von durch den *server* übermittelten *Status* Objekten und dem Versenden von *Userlabels* an den *server* sowie das *Anstoßen* von *Tasks* auf dem *server*.
  - **JobDataMap** / Klasse: Speichert die vom *server* an den *client* delegierten *Jobs*, die zuvor mittels der *JobDataFactory* formatiert wurden.
  - **JobDataFactory** / Klasse: Formatiert und validiert die durch den *server* übermittelten *Jobs*.

Detailbeschreibung / *server*seitig

- **prediction.inc.php**: Beinhaltet alle *serverseitigen* Funktionalitäten.[14]
  - **PredictionSecurity** / Klasse: Für die Sicherheit des *serverseitigen* Teils der Funktionsbibliothek zuständige Klasse. Beinhaltet etwa Methoden gegen *session fixation* oder um sicherzustellen, dass nur Anfragen der zuletzt gestarteten Browser Instanz vom *client* als gültig anerkannt werden. Dies ist etwa beim Erteilen von *Jobs* durch den *server* wichtig, da die Authentifizierungsdaten des *clients* mittels *session* und *cookie* gespeichert werden. Wird ein *Job* vom *server* erteilt, werden diese Authentifizierungsdaten und den *IDs* der jeweiligen Einträge zu Validierungszwecken in der Datenbank gespeichert und nach erfolgreicher Bearbeitung des Auftrags durch den *client* die zurückgegebenen Werte mit den Validierungsdaten der Datenbank verglichen. Startet ein Anwender nun mehrere Instanzen eines Browsers, können multiple *Jobs* an den *client* überwiesen werden und deren nicht der Reihung der Erteilung entsprechende Rückgabe zu einer Inkonsistenz der Datenbank führen. Weiters ist diese Klasse für die Handhabung der in Abschn. 6.5 beschriebenen Sicherheitsmaßnahmen zuständig.
  - **PUtility** / Klasse: Hilfsklasse zum Kapseln wesentlicher durch die Modelle und deren Manager verwendeter Methoden wie Validierung und Vergleich. Weiters sind in dieser Klasse die Selektionsmethoden und *factory* Methoden, die von den Managerklassen verwendet werden, enthalten.
  - **Model_Base** / Klasse: Basisklasse der verwendeten Modelle mit den durch diese verwendeten Methoden.
  - **FunctionManager** / Klasse: Manager Klasse, die für die Handhabung der in der Datenbank gespeicherten Funktionen zuständig ist. Eine Funktion besteht intern aus einem Eintrag in der *function*

[14] Wie bereits zuvor erwähnt wurde, bedient sich dieser Teil der Umsetzung der ORM (*object relational mapping*) Bibliothek *RedBeanPHP*.

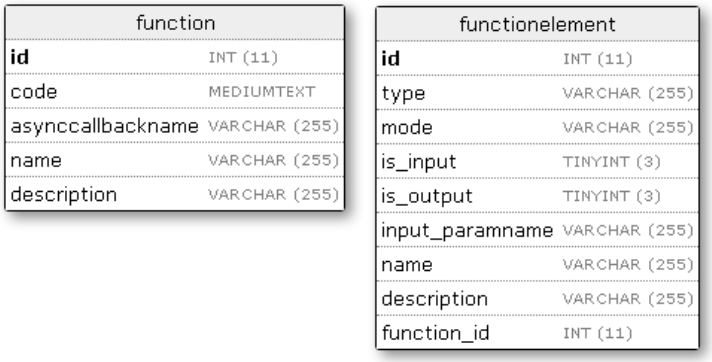


Abbildung 6.7: Die für das Speichern von *Funktionen* verwendete *MySQL* Tabellen.

Tabelle und multiplen Einträgen in der *functionelement* Tabelle (welche die Eingangs- und Ausgangsparameter der Funktion abbilden). Die Datenbankstruktur ist in Abb. 6.7 ersichtlich. Die verwalteten Modelle sind:

  * **Model_Function** / Klasse,
  * **Model_Functionelement** / Klasse.

– **ModelManager** / Klasse: Zuständig für die Verwaltung der in der Datenbank gespeicherten Modelle. Ein Modell besteht intern aus einem Eintrag in der *model* Tabelle, multiplen Einträgen in der *modelversion* Tabelle, multiplen Einträgen in der *modelresult* Tabelle (die möglichen Klassifizierungen des Modells) und multiplen Einträgen in den Tabellen *modelelement* und *modelelementinput* (diese zwei Tabellen repräsentieren die Verkettungsstruktur der Modellfunktionen). Die entsprechende Datenbankstruktur ist in Abb. 6.8 ersichtlich. Die verwalteten Modelle sind:

  * **Model_Model** / Klasse,
  * **Model_Modelversion** / Klasse,
  * **Model_Modelresult** / Klasse,
  * **Model_Modelelement** / Klasse,
  * **Model_Modelelementinput** / Klasse.

– **SourceManager** / Klasse: Manager Klasse, die für die Handhabung der in der Datenbank gespeicherten *Source* Einträge zuständig ist. Eine *Source* besteht intern aus einem Eintrag in der *source* Tabelle und multiplen Einträgen in der *sourcevalue* Tabelle. Die Datenbankstruktur ist in Abb. 6.9 ersichtlich. Die verwalteten Modelle sind:

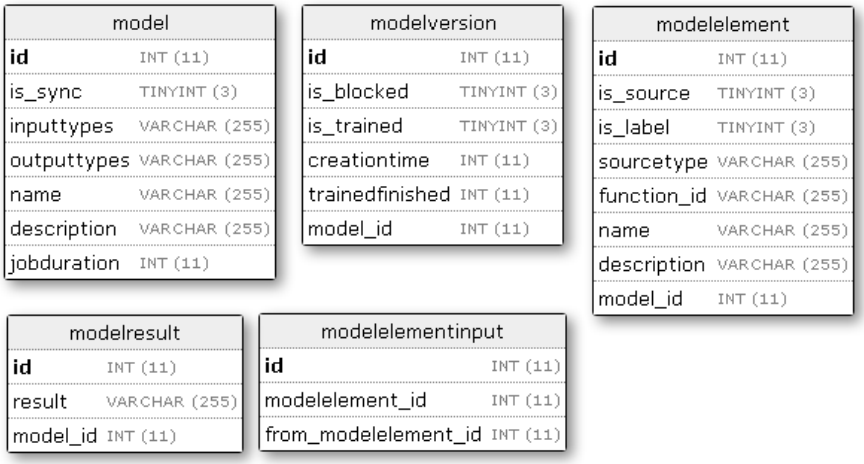


Abbildung 6.8: Die für das Speichern von *Modellen* verwendete *MySQL* Tabellen.

  - **Model_Source** / Klasse,
  - **Model_Sourcevalue** / Klasse.

- **PredictionManager** / Klasse: Zuständig für die Verwaltung der in der Datenbank gespeicherten *Predictions*. Weiters zuständig für das Erfragen des aktuellen Status einer solchen, die Auswahl von *Jobs*, das Speichern der durch den *client* generierten *Job* Resultate und das Einspeisen dieser Werte in den Vorhersagealgorithmus der *Google Prediction API*. Die hierbei gewonnenen Klassifizierungen werden in den jeweiligen *Predictions* gespeichert, die Spezifikationen des die Rückgabewerte übermittelnden *clients*, die *Jobs* Daten und die Dauer der Bearbeitung zu Evaluierungszwecken in der *harvest* Tabelle gespeichert. Eine *Prediction* besteht intern aus einem Eintrag in der *prediction* Tabelle und multiplen Einträgen in der *predictioninput* Tabelle. Ein *Harvest* Eintrag wird in der gleichnamigen Tabelle gespeichert. In Abb. 6.9 ist die Datenbankstruktur einer *Prediction* und in Abb. 6.10 jene eines *Harvest* Eintrags ersichtlich. Die verwalteten Modelle sind:
  - **Model_Prediction** / Klasse,
  - **Model_Predictioninput** / Klasse,
  - **Model_Harvest** / Klasse.
- **UserManager** / Klasse: Zuständig für die Handhabung der in der Datenbank gespeicherten *User* Einträge. Handhabt weiters das Speichern der Validierungsdaten eines an den Anwender delegierten *Jobs* und die Daten, die vom System generiert werden,

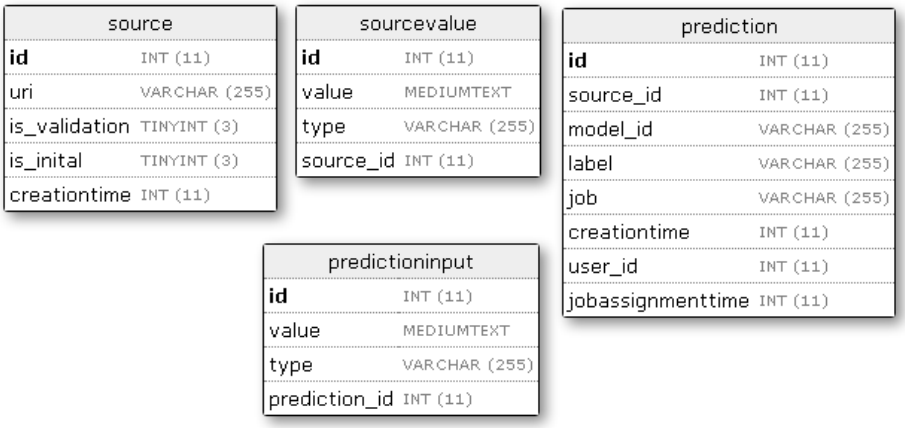


Abbildung 6.9: Die für das Speichern von *Source* und *Prediction* Einträgen verwendete *MySQL* Tabellen.

um sicherzustellen, dass nur die zuletzt geladene Browser Instanz Anfragen an den *client* versenden darf. Die Datenbankstruktur eines *User* Eintrags ist in Abb. 6.11 ersichtlich. Das verwaltete Modell ist:

  * **Model_User** / Klasse.

– **UserLabelManager** / Klasse: Zuständig für die Handhabung der in der Datenbank gespeicherten *Userlabels*, d.h. der Klassifizierung, die ein Anwender einer *Prediction* zuordnet. Wie bereits zuvor erläutert, kann ein Anwender seine Bewertung beliebig oft ändern, allerdings fließt nur die zuletzt gespeicherte Klassifizierung in den für den Eintrag generierten Status ein. Anonyme Klassifizierungsvorschläge werden ebenfalls als *Userlabels* gespeichert, die *user_id* ist in diesem Fall nicht definiert. Die Datenbankstruktur eines *Userlabels* ist in Abb. 6.11 ersichtlich. Das verwaltete Modell ist:

  * **Model_Userlabel** / Klasse.

– **TaskManager** / Klasse: Zuständig für das Verwalten von *serverseitigen Tasks*. Die Datenbankstruktur eines *Tasks* ist in Abb. 6.10 ersichtlich. Das verwaltete Modell ist:

  * **Model_Task** / Klasse.

– **PredictionLogger** / Klasse: Hilfsklasse, die zum *loggen* von Informationen und Fehlern verwendet wird, sowie zum Analysieren der generierten *log* Dateien. Weiters beinhaltet diese Klasse eine

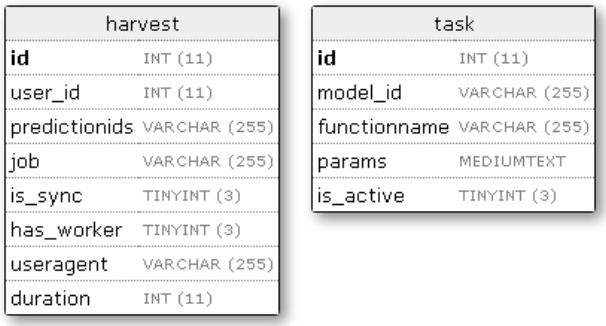


Abbildung 6.10: Die für das Speichern von *Harvest* und *Task* Einträgen verwendeten *MySQL* Tabellen.

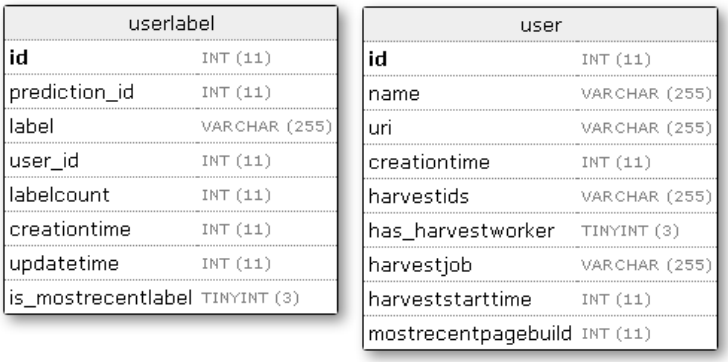


Abbildung 6.11: Die für das Speichern von *User* und *Userlabel* Einträgen verwendeten *MySQL* Tabellen.

Methode, um die für ein Training eines Modells[15] durch die *Google Prediction API* benötigte Trainingsdatei zu generieren. Da das Erzeugen dieser Dateien ein relativ umfangreicher Prozess ist (es können ohne Weiteres tausende *Predictions* analysiert werden, d.h. deren Input Daten gelesen und die bestehende Klassifizierung mit den durch den entsprechenden Status definierten Bewertungen der Anwender verglichen werden), wird dieser Prozess in einzelne *Task* Schritte untergliedert.[16] Wird nun einer dieser Schritte aufgrund eines Fehlers unerwartet unterbrochen, kann dies zu In-

[15] In der der vorliegenden Umsetzung zugrunde liegenden Terminologie sprechen wir natürlich von einer *Modellversion*.

[16] In jedem einzelnen dieser Schritte wird eine bestimmte Anzahl an Einträgen analysiert, und die hierbei generierten Werte wie in Abschn. 6.2.3 ersichtlich zeilenweise in der neuen Trainingsdatei gespeichert.

konsistenzen in der generierten Textdatei führen (etwa kann der Schreibvorgang mitten in der Zeile unterbrochen und dadurch die Struktur für die *Google Prediction API* unleserlich gemacht werden). Die in dieser Klasse enthaltene Methode erkennt also in Folge, ob ein vorhergehender Taskschritt derart unterbrochen wurde und korrigiert diese Fehler in der generierten Textdatei.

  - **RequestQueue** / Klasse: Hilfsklasse, die zur Vermeidung von Kollisionen bei der Anfragebeantwortung durch den *server* verwendet wird. Erzeugt bzw. verwendet intern Dateien, die beim Userzugriff gesperrt und erst nach Abarbeitung der Anfrage wieder freigegeben werden. Erfolgt eine Anfrage durch einen unterschiedlichen *client*, wird diese gereiht und periodisch überprüft, ob die gesperrte Ressource bereits durch den ersten *client* freigegeben wurde. Anfragen werden abgewiesen, wenn innerhalb einer maximalen Zeitdauer kein Zugriff auf die Ressource möglich ist.
  - **CurrentDBStatus** / Klasse: Hilfsklasse, die intern zur Überprüfung des Datenbankzustandes dient.

- **prediction_admin.inc.php**: Beinhaltet die Klasse **PredictionAdmin** in der die einzelnen *Tasks* definiert sind bzw. die durch diese verwendeten Funktionen. Die *Tasks* sind:
  - Das Einfügen der initialen Trainingsdaten in die Datenbank.
  - Das Generieren einer neuen Modellversion bzw. einer entsprechenden *CSV* Trainingsdatei.
  - Das Hochladen dieser Trainingsdatei mittels des *Google Storage Service* (siehe *google_service.inc.php*).
  - Das Initieren des Trainings der Modellversion mittels der *Google Prediction API* (siehe *google_service.inc.php*).
  - Das periodische Erfragen (*pollen*) des aktuellen Trainingszustandes des zuvor initiierten Prozesses.
- **prediction_settings.php**: Beinhaltet die Klasse **PredictionSettings** in der alle relevanten Setupdaten der Funktionsbibliothek gespeichert werden. Die *wesentlichsten* dieser Daten sind:
  - Die Anzahl der pro *Job*anfrage zurückgegebenen Einträge (dieser Wert errechnet sich auch aus dem Umstand, ob das verwendete Modell eine *asynchrone* Funktion beinhaltet oder der *client* mehrere Javascript *threads* parallel mittels des *HTML5 webworkers* ausführen kann).
  - Die Datenbank Verbindungsdaten.
  - Die Ordner, in denen generierte *log* Dateien und Trainingsdateien erzeugt bzw. von denen existierende Dateien gelesen werden.

  - Die Zeitdauer, die der *RequestQueue* für das Beantworten von Fragen zur Verfügung steht.
  - Definitionen, an welchen Stellen des *serverseitigen work-flows* Informationen und Fehler in die *log* Dateien geschrieben werden sollen.
  - Die in der Datei *google_ service.inc.php* verwendeten Parameter.

  Weiters beinhaltet die Klasse dieser Datei die Methoden, die für die Verbindung zur Datenbank zuständig sind und Hilfsmethoden, die zum Erfragen von Daten des aktuell die Anfrage stellenden Anwenders verwendet werden.
- **google_service.inc.php**: Klassen, die die Verbindung mit den benutzten *Google* Services herstellen bzw. das Ansprechen deren Funktionalitäten kapseln.
  - **GoogleAPIClient** / Klasse: Handhabt die Verbindungsherstellung zu den Services unter Verwendung eines *accessTokens*. Ist dieses im Begriff abzulaufen (die von *Google* erteilten *accessTokens* haben nur eine zeitlich begrenzte Gültigkeit), wird mittels eines *refreshTokens* ein neues *accessTokens* beantragt.
  - **StorageService** / Klasse: Handhabt das Hochladen sowie Trainingsdateien mittels des *Google Cloud Storage Service* (dies ist eine Notwendigkeit um ein Modell durch die *Google Prediction API* trainieren zu können). Hierfür wird zuerst überprüft, ob ein *bucket* mit der *ID* des Modells existiert, wenn nicht (dies ist beim Training der ersten Modellversion der Fall) wird ein solcher erzeugt. Anschließend wird die Trainingsdatei der zu trainierenden *Modellversion* in diesen hochgeladen.
  - **PredictionService** / Klasse: Zuständig für alle Interaktionen mit der *Google Prediction API*. Beinhaltet Methoden für das Trainieren, das Abfragen des aktuellen Trainingszustandes (diese Methode wird etwa beim *polling Task* verwendet) und dem Absetzen einer Klassifizierungsanfrage an den Vorhersagealgorithmus der *Google Prediction API*.

6.3.2 Modell Tool

Funktionsweise

Mittels des *Modell Tools* können Modelle erstellt und verwaltet werden, und diese mit einer Reihe von *Klassifizierungen* und einer Anzahl von *Eingangswerten* versehen werden. Die Implementierung erlaubt es, die in der Datenbank definierten Funktionen nach Namen zu durchsuchen, eine entsprechende Auswahl zu laden und diese in die Verkettungsstruktur des Modells (siehe Abb. 6.2) einzufügen, sowie deren Position in der Verkettung mittels *drag-*

Abbildung 6.12: Dateien des *Modell Tools*. Die durch dieses verwendeten Dateien der Funktionsbibliothek (siehe Abschn. 6.3.1) werden hier nicht erneut angeführt.

and-drop zu verändern. Die Verknüpfung der Funktionen untereinander[17], sowie das Definieren der Ausgangswerte des Modells wird mittels entsprechender Navigationselemente in den Elementen der Liste ermöglicht. Um die Übersicht zu gewährleisten kann der Anwender die Funktionen bzw. die entsprechenden Listenelemente minimieren oder maximieren. Jede Funktion besitzt eine Detailansicht in der der Anwender eine Kurzbeschreibung (mit einem Verweis auf ihr Synchronitätsverhalten), den Programmcode sowie die Art und Anzahl der Eingangs- und Ausgangswerte einsehen kann.

Da eine *topologische* Sortierung der Funktionen zur Laufzeit zu ressourcenintensiv wäre (um *circular dependencies* in der Listenstruktur zu vermeiden), gilt folgende Konvention: Die Eingangswerte der Funktion müssen aus einem Listenelement (entweder einem Eingangswert des Modells oder einem Ausgangswert eines anderen Funktionselements) kommen, welches über dem Zielelement liegt und die Ausgangswerte der Funktion dürfen nur auf Funktionselemente verweisen, welches sich unterhalb des Elements befindet. Bei folgenden Handlungsschritten des Anwenders wird vom System eine Überprüfung der Validität der Aktion bzw. eine Auflistung der möglichen Optionen angeboten:

- Signalisiert der Anwender die Absicht, mittels einer ziehenden Bewegung (*drag*) vom Pfeil Element ausgehend, die aktuell geladene Funktion der Liste hinzuzufügen, werden alle möglichen Positionen (*drop*) der Liste angezeigt, d.h. diejenigen, die ein verknüpfbares Element über der Zielposition haben. Alle übrigen Positionen werden blockiert und ausgegraut.
- Signalisiert der Anwender den Wunsch ein Listenelement zu verschie-

[17] Woher bezieht die Funktion ihre Eingangswerte und wohin verweist ihr Ausgangswert.

ben, werden wie zuvor beschrieben, die möglichen Positionen angezeigt und weiters überprüft, ob die neue Position die eingangs getroffene Konvention verletzen würde.

- Will der Anwender ein Listenelement löschen, so überprüft das System, ob es mit anderen Elementen verknüpft ist.
- Versucht der Anwender das Modell zu speichern, wird überprüft ob:
  - Alle Funktionen korrekt verknüpft wurden.
  - Alle Eingangsparameter des Modells verknüpft sind.
  - Alle notwendigen Modell Parameter definiert wurden.
  - Mindestens eine Funktion als Ausgangswert für das Modell festgelegt wurde.

Will ein Anwender die Verknüpfungskette einer Funktion angezeigt bekommen[18], ist dies mittels eines weiteren Navigationselements innerhalb der Listenelemente möglich. Gespeicherte Modelle können schließlich mittels des ihnen zugeordneten Namens bzw. ihrer *ID* durch die *Prediction Library* verwendet werden, um *Source* Einträge zu formatieren und automatisierte Klassifizierungen vorzunehmen. Die Filestruktur ist in Abb. 6.12 ersichtlich.

Das in Abb. 6.13 gezeigte Modell ist jenes, welches auch im Beispielszenario (der Identifizierung von *Spam* Einträgen) zur Anwendung kommt, die Funktionselemente der Liste entsprechen der Verkettungsstruktur, welche in Abb. 6.2 beschrieben wird. Die geöffneten Listenelemente sind:

1. Der zweiten Eingangswert des Modells.[19]
2. Die sich an letzter Stelle befindliche Funktion, welche zugleich den letzten Parameter der mittels Modell aus der *Source* generierten *Prediction* liefert (in Tab. 6.3 als *formatierter Text* bezeichnet).

Detailbeschreibung / *client*seitig

- **jquery.ui.loadinganimation.js**: Setzt während des Ladevorgangs den Anwender von diesem in Kenntnis und unterbindet Aktionen von ihm.
- **jquery.ui.callbacktabs.js**: *Widget* zur Darstellung von Eigenschaften (wie z.B. Eingangsdaten, Ausgangsdaten, Beschreibung oder *code*) selektierter Funktionen. Beinhaltet Funktionalitäten zum Erzeugen neuer Tabulatoren, Selektieren bzw. Freischalten und Sperren von Tabulatoren. Kommunikation mit anderen Elementen erfolgt mittels des *jQuery observer patterns*. Verwendet die *jQuery UI widget factory* zum

[18] Damit ist die Abfolge der Funktionen definiert, von der bzw. an die Werte *durchgeschliffen* werden.

[19] Eingangswerte sind, wie bereits zuvor beschrieben, die Parameter eines *Source* Eintrages. Die im *screenshot* gezeigte *URL* ist zwecks besserem Verständnis angegeben und wird natürlich wie in Prog. 6.1 ersichtlich, nicht dem Modell, sondern dem entsprechenden *Source* Eintrag zugeordnet.

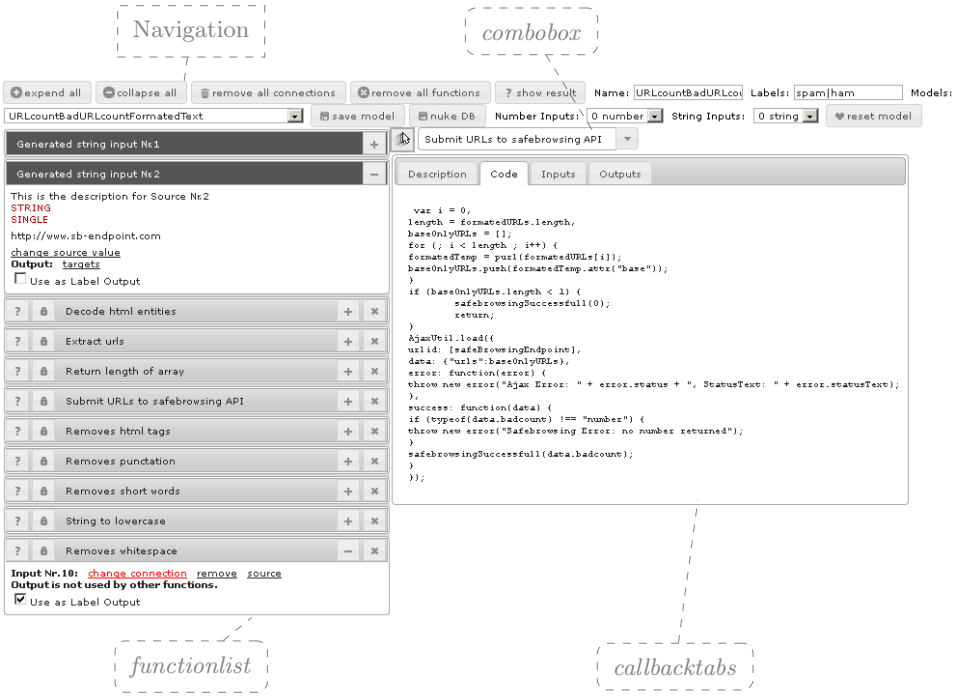


Abbildung 6.13: Das mittels des *Modell Tools* erstellte *Spam* Klassifizierungsmodell. Der Bildschirmausschnitt unterteilt sich in **Navigation** (den am oberen Rand ersichtlichen Navigationsbereich), das ***combobox widget*** (das *dropdown* Menü und das rechts davon befindliche Pfeilsymbol), das ***functionlist widget*** (die Elemente beginnend mit den dunkel hinterlegten Eingangsparametern des Modells) und dem ***callbacktabs widget*** (das Tabulator Feld mit der aktuellen Selektion *code*). Die verwendeten *widgets* werden in Abschn. 6.3.2 beschrieben.

Registrieren und Erzeugen. In Abb. 6.13 ist aktuell der *code* Tabulator selektiert, der den Funktionstext der in der *combobox* ersichtlichen Funktion anzeigt.

- **jquery.ui.combobox.js**: *Widget* zur Auswahl von Funktionen. Erlaubt die Auswahl aus einer *dropdown* Liste aller Funktionen sowie durch eine Zeicheneingabe, wobei die vorhandenen Funktionen mittels einer *regular expression* durchsucht werden und entsprechende Treffer als Auswahl zur Verfügung gestellt werden. Kommunikation mit anderen Elementen erfolgt mittels des *jQuery observer patterns*. Verwendet die *jQuery UI widget factory* zum Registrieren und Erzeugen. In Abb. 6.13 ist aktuell die Funktion mit der Kennung *Submit URLs to safebrowsing API* ausgewählt, es wird keine *dropdown* Liste der vorhandenen Funktionen angezeigt.
- **jquery.ui.functionlist.js**: *JQuery UI widget* zum Anordnen und Ver-

knüpfen von Funktionen eines Modells, sowie zum Definieren dessen Eingangsdaten (den dunkel hinterlegten Listenelementen in Abb. 6.13) und Ausgangsdaten (in den Elementen als *label output* bezeichnet[20]). Verwendet intern die *jQuery UI widgets sortable* und *dragable*. Erlaubt mittels einer ziehenden Bewegung (*drag*) vom Pfeil Element ausgehend, die zuvor mittels des *jQuery UI widgets combobox* ausgewählte Funktion der Liste hinzuzufügen. Derart erzeugte Funktionsinstanzen werden als grafische Repräsentation in Form eines *clones* des *DOM* Elements *Function List Modell* (siehe *index.php*) dargestellt und sind mit folgenden Funktionalitäten versehen:

1. **Information** (Fragezeichen): Übermittelt mit dem *observer pattern* die *ID* der Funktion des Listenelements an das *callbackTab widget* und *combobox widget* und selektiert die entsprechende Funktion in diesen. Ermöglicht ein einfaches Wechseln zwischen Funktionen innerhalb der definierten Funktionsstruktur, um etwa Detailinformationen wie Eingänge oder Ausgänge einer Funktion zu überprüfen.
2. **Lock / Unlock** (Schlosssymbol): Um das Listenelement an eine neue Position verschieben zu können, ist es notwendig dieses freizuschalten. Das System erkennt an welche Positionen das derart ausgewählte Element verschoben werden kann, um keine bestehenden Verknüpfungen zu verletzen und zeigt dem Anwender diese an, alle anderen Positionen werden blockiert und ausgegraut.
3. **Expend / Collapse** (Plus- oder Minussymbol): *Toogle* Funktion, um den Körper des Listenelements (mit den entsprechenden Verknüpfungsoptionen) anzeigen bzw. verstecken zu können.
4. **Close** (Kreuzsymbol): Entfernt das Element aus der Liste. Überprüft zuvor ob Abhängigkeiten mit anderen Funktionen bestehen und gibt in diesem Falle dem Anwender einen Warnhinweis aus.

Jedes Listenelements bietet dem Anwender die Möglichkeit im geöffneten Elementkörper (in Abb. 6.13 etwa das letzte Element) die für die Funktion notwendigen Eingangsparameter zu definieren bzw. bereits bestehende Verknüpfungen anzuzeigen, zu ändern oder zu entfernen. Als Quelle kann dafür der Ausgangswert einer anderen Funktion oder einer der Eingangswerte des Modells dienen, sofern die Typen übereinstimmen. Folgende Typen stehen zur Verfügung:

1. *string / single*,
2. *string / array*,
3. *number / single*,

[20] Im genannten Bild ist etwa das letzte Listenelement mittels der selektierten *checkbox* entsprechend markiert.

4. *number / array*,
5. *boolean / single*,
6. *boolean / array*.

Versucht ein Anwender einen Eingang einer Funktion zu verknüpfen, oder eine bestehende Verknüpfung zu ändern, durchsucht das System alle passende Ausgangswerte von Elementen, die in der Liste vor dem selektierten Funktionselement liegen und bietet die Option an diese zu verbinden. Erfolgt eine Verknüpfung, werden im Elementkörper die Optionen hinzugefügt diese zu ändern oder zu löschen. Verknüpfungen haben Einfluss auf die möglichen Verschiebepositionen. Funktionselemente können in der Liste niemals über das tiefstliegende Eingangselement gezogen werden bzw. unter das höchst liegende Ausgangselement, mit dem es verknüpft ist. Besitzt eine Funktion einen Ausgangswert vom Typ *string / single* oder *number / single*, kann sie mittels einer *checkbox* als ein Ausgangsparameter des Modells definiert werden.

- **jquery.createmodel.js**: Hauptdatei des *Modell Tools*. Beinhaltet folgende Funktionalitäten:
  1. Initialisiert die modale Ladeanimation (siehe *jquery.ui.loadinganimation.js*).
  2. Lädt die zur Verfügung stehenden Funktionen mittels der *AjaxUtil* Klasse (siehe Abschn. 6.3.1 / *predictionlib.utilities.js*).
  3. Lädt die bereits erstellten Modelle mittels der *AjaxUtil* Klasse.
  4. Initialisiert die *jQuery UI widgets combobox*, *callbackTabs* und *functionList* und weißt sie den *DOM* Elementen zu.
  5. Definiert die Optionen, *styles*, *IDs*, Klassen, *event handler* und *callback* Funktionen für die *widgets*.
  6. Weist den Navigationselementen folgende Funktionen zu:[21]
     (a) expand all / collapse all: Alle Funktionen der Liste werden auf- bzw. zugeklappt.
     (b) remove all connections: Alle Verbindungen der Listenelemente werden entfernt.
     (c) remove all functions: Alle Listenelemente, bis auf die Eingangsdaten des Modells werden entfernt.
     (d) show results: Validiert die aktuelle Listenstruktur, versucht eine *topologische* Sortierung, wendet die Funktionskette auf Zufallswerte an und zeigt den generierten *Prediction* Wert.[22]

[21] Die zugewiesenen Funktionen werden mit der Namensbezeichnung des entsprechenden Navigationselements angeführt. Die Übermittlung des Aufrufs und allfälliger Daten erfolgt mittels des *jQuery observer patterns*.

[22] Welcher aus mehreren Werten, den Ausgangswerten des Modells besteht und natürlich noch *nicht* klassifiziert ist. Hierfür müssten die Werte der *Prediction*, die in der Begriffser-

(e) save model: Führt die zuvor genannten Validierungsschritte durch und speichert das Modell.

(f) nuke DB: Löscht alle definierten Modelle.

(g) reset model: Entfernt alle Listenelemente und setzt die aktuellen Modellparameter zurück.

- **index.php**: Dient dazu die benötigten Javascript Files zu laden und die entsprechenden (mittels *jQuery* und *jQuery UI* angesprochenen) *DOM* Elemente anzulegen. Die primäre Funktionalität ist in den entsprechenden Javascript Dateien implementiert.
  1. Laden der *jQuery* und *jQuery UI* Bibliothek, der verwendeten CSS files, der *Prediction Library* und der *jQuery UI GUI* Elemente.
  2. Definition der *endpoints* und der *client* Daten mittels der *Settings* Klasse (siehe Abschn. 6.3.1 / *predictionlib.utilities.js*).
  3. Definition eines *Function List* Modells (siehe *jquery.ui.functionlist.js*).
  4. Definition der primären Navigationselemente.
  5. Definition eines Interfaces für das Ansprechen von globalen Funktionalitäten des *widgets jquery.ui.functionlist.js*.
  6. Definition von *DOM* Elementen für die *jQuery UI widgets*:
     - *jquery.ui.functionlist.js*,
     - *jquery.ui.combobox.js*,
     - *jquery.ui.callbacktabs.js*.

Detailbeschreibung / *server*seitig

Das Modell Tool bedient sich der in der Funktionsbibliothek definierten *server*seitigen Strukturen und *endpoints*, die an dieser Stelle nicht erneut erläutert werden sollen.

6.4 Ablauf

Im folgenden Abschnitt werden die *client*- und *server*seitigen Programmabläufe erläutert und die Abfolge dieser einzelnen Prozesse in einen Kontext gestellt. Als Basis dient hierfür der in Abb. 6.14 gezeigte, auf dem *client* ausgeführte Aufrufzyklus. Der in Folge näher beleuchtete Prozessablauf wird initiiert, sobald ein Anwender eine mit den *client*seitigen Javascriptfunktionalitäten der Projektimplementierung versehene *website* aufruft. Die benötigten Javascriptdateien gelten also zum Zeitpunkt der folgenden Betrachtung als

klärung als Eingangsdaten bezeichneten Werte, in den Vorhersagealgorithmus eingespeist werden.

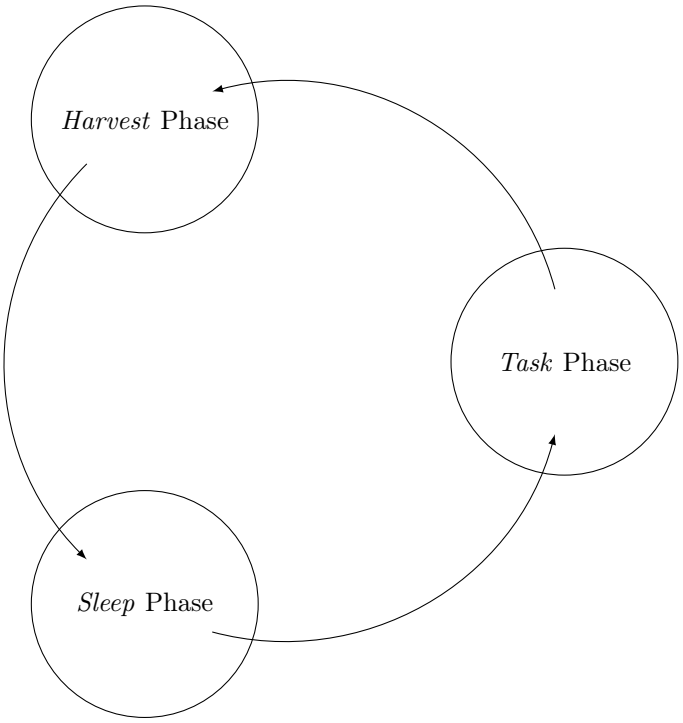


Abbildung 6.14: Zyklus der drei am *client* aufeinanderfolgenden Prozessphasen.

geladen und die benötigten Parameter wie der Authentifizierungsstatus des Anwenders als generiert. Der am *client* in Folge initiierte Aufrufzyklus setzt sich aus den folgenden drei Phasen zusammen (wobei die *Task* Phase an erster Stelle steht):

1. *Task*,
2. *Harvest*,
3. *Sleep*.

Tritt ein behandelbarer Fehler in einer dieser Phasen auf, wird die Applikation in die *Sleep* Phase versetzt, unbehandelbare Fehler führen, wie in Javascript üblich, zu einem Programmabbruch.

6.4.1 *Task* Phase

Wie bereits im Abschn. 6.2.4 erläutert, stellen *Tasks server*seitige Aufgaben dar, die von einem *client* aktiviert werden und sowohl von der Applikationslogik selbstständig als auch durch einen Administrator zu gegebenem Zeitpunkt definiert werden können. Wie in Abb. 6.15 ersichtlich besteht diese

Phase aus folgenden Ablaufschritten:[23]

1. **Phase initiieren ?** / *client*: Ist dies der initiale Aufruf oder hat der *server* in der zuvor erfolgten Antwort das Existieren eines *Tasks* signalisiert, wird der Ablauf initiiert. Ansonsten wird in die *Harvest* Phase gewechselt.
2. **Sende *pushTask* Anfrage** / *client*: Eine Anfrage wird vom *client* an den *server* übermittelt, um einen allfälligen *Task* zu aktivieren bzw. fortzusetzen. Diese Anfrage wird intern als *pushTask* bezeichnet.
3. **Anfrage verarbeiten** / *server*: Die Anfrage wird empfangen und ihre Legitimität überprüft.
4. **Task existiert und nicht aktiv ?** / *server*: Überprüft, ob ein Task existiert bzw. dieser nicht aktuell ausgeführt wird und startet diesen bzw. setzt ihn fort. Ansonsten wird das Nichtvorhandensein weiterer *Tasks* mittels einer leere Antwort dem *client* kenntlich gemacht.
5. **Starte *Task*** / *server*: Startet den *Task* mittels des in der Datenbank gespeicherten *Taskzustandes* und entfernt ihn nach erfolgreicher Abarbeitung.[24]
6. ***Task* aktiv oder weiterer *Task* ?** / *server*: Wurde die Aufgabenstellung noch nicht erfolgreich abgearbeitet bzw. existieren weitere *Tasks*, wird eine entsprechende Antwort an den *client* generiert. Ansonsten erfolgt eine leere Antwort an den *client* zur Kenntlichmachung dieses Umstandes.
7. **Generiere Antwort an *client*** / *server*: Antwort wird generiert, um den *client* vom Vorhandensein weiterer *Tasks* zu informieren.
8. **Retourniere Antwort** / *server*: Die erstellte Antwort bzw. ein leeres Antwortobjekt wird an den *client* zurückgeschickt.

Beispiel

Im vorliegenden Beispielszenario existieren etwa folgende *Tasks*:[25]

1. **Einspielen der initialen Trainingsdaten**: Durch den Administrator initiiert, wird ein initialer Trainingsdatensatz (siehe Abschn. 6.2.3) validiert und dessen Einträge als *Sources* gespeichert. Dieser Task untergliedert sich in die Arbeitsschritte:

[23] Diese werden wiederholt durchgeführt, bis eine durch die Applikationslogik definierte maximalen Aufrufanzahl erreicht ist, oder der *server* das Fehlen weiterer anstehender *Tasks* meldet.

[24] Beim erstmaligen Starten eines *server*seitigen *Task* ist dieser Zustand nicht definiert. Folgeaufrufe werden mit den Zuständen (dem *Fortschritt*) des letzten *Taskaufrufes* fortgesetzt.

[25] Die angeführten *Tasks* sind gemäß des Zeitpunkts ihrer Definition und Abarbeitung aufgelistet.

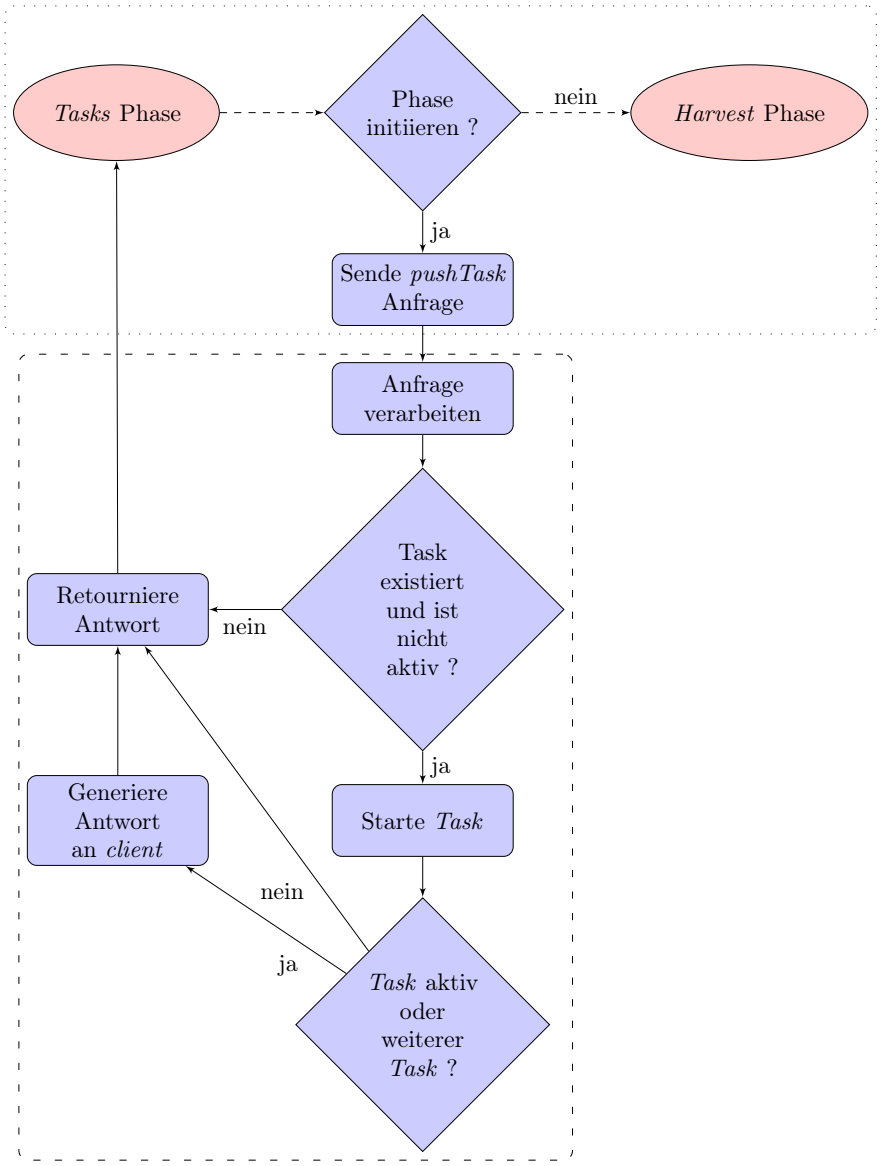


Abbildung 6.15: Ablauf der *Task* Phase und Übergang in die *Harvest* Phase. Der punktiert umrandete Bereich stellt die Prozesse des *clients* dar, der gestrichelt umrandete Bereich die Prozesse aufseiten des *servers*.

(a) **Validieren der Trainingsdaten**: Sämtliche Trainingsdaten werden gemäß der Korrektheit ihrer Klassifizierung bzw. der Anzahl und des Typs ihrer Eingangsdaten einer Überprüfung unterzogen. Ist diese erfolgreich, wird dieser Umstand als *Taskzustand* gespeichert und der *client* vom aktiven Status des *Tasks* in Kenntnis gesetzt, ansonsten wird die Aufgabe entfernt und eine leere Ant-

wort an den *client* generiert.

(b) **Segmentiertes Übertragen der Trainingsdaten**: Speichert pro Aufruf des *Tasks* eine definierte Anzahl an Trainingsdaten als *Source* Einträge und die Zeilennummer des zuletzt extrahierten Trainingsdatensatzes als *Taskzustand*. Die generierten *Sources* werden jeweils mit einem *Input Job* (siehe Abschn. 6.2.4) versehen, sodass in der folgenden *Harvest* Phase Aufträge an die *clients* zur Generierung der *Predictions* übermittelt werden können.[26] Wurden alle Trainingsdaten erfolgreich verarbeitet, wird der Task entfernt.

2. **Generieren einer neuen *Modellversion***: Nach einer definierten Dauer (oder durch den Administrator initiiert) wird mittels dieses *Tasks* eine neue Modellversion erzeugt. Er besteht aus folgenden Arbeitsschritten:

   (a) **Segmentierte Reklassifizierung der *Predictions* und Generierung einer Trainingsdatei**: In diesem Schritt wird der Status (siehe Abschn. 6.2.5) einer Reihe nicht mit einem *Job* versehenen *Predictions* ermittelt, aufgrund dessen eine Reklassifizierung der entsprechenden Einträge vorgenommen[27] und diese in eine neu erstellte Trainingsdatei (im *CSV* Format) geschrieben. Dieser Prozess gibt ähnlich dem zuvor genannten *segmentierten Übertragen der Trainingsdaten* einen *Taskzustand* (die aktuelle *ID* der zuletzt verarbeiteten *Prediction)* an Folgeaufrufe weiter.

   (b) **Hochladen der Trainingsdatei auf *Google Storage***: Nach erfolgreichem Erstellen der Trainingsdatei wird diese in einen dynamisch erzeugten *bucket* auf *Google Storage* hochgeladen.[28]

   (c) **Initiieren des Trainingsvorganges**: Nach erfolgreichem Hochladen der Trainingsdatei wird das Trainieren einer neuen Modellversion mithilfe der *Google Prediction API* initiiert.

   (d) **Pollen des Trainingszustandes**: Mittels dieses Taskschrittes

[26] Senden multiple *clients* Anfragen an den *server* und wurde etwa ein derartiger *Task* am *server* durch den ersten *client* zum wiederholten Mal initiiert, so läuft eine Anfrage von einem zweiten *client* ins Leere (der *server* liefert eine leere Antwort zurück, da ein *Tasks* aktiv ist), dieser wechselt in die *Harvest* Phase und kann bereits diejenigen *Jobs* vom *server* erfragen, die vom ersten *client* zuvor definiert wurden. Dies nur als Beispiel, dass die einzelnen Phasen in den beteiligten *clients* nicht in einer identen Abfolge verlaufen werden.

[27] Hierfür werden die einzelnen Bewertungen mit einer gemäß ihrer Art (*anonym*, *authentifiziert* und *aktuell*) versehenen Gewichtung multipliziert und die resultierende, höchst gewertete Klassifizierung übernommen.

[28] Dies ist eine Notwendigkeit zum Verwenden der *Google Prediction API*. Sowohl der *bucket* als auch der Name der Trainingsdatei entsprechen dem Namen des verwendeten Modells. Die Trainingsdatei hat eine der Modellversion entsprechende Seriennummer im Namen beigefügt.

wird der Trainingszustand der aktuellen Modellversion vom *Google Prediction Service* erfragt. Ist das Training abgeschlossen, wird die Modellversion applikationsintern als trainiert markiert, dies bedeutet, dass ab diesem Zeitpunkt *Prediction Jobs* (siehe Abschn. 6.2.4) erstellt bzw. an *clients* delegiert werden können.

6.4.2 *Harvest* Phase

Die *Harvest* Phase ist der Teil des Ablaufzyklus, indem die eigentliche Distribution der Arbeitsaufträge an die verbundenen *clients* stattfindet, bzw. die durch diese generierten Ergebnisse an den *server* zurückgegeben werden. Die in Abb. 6.16 dargestellten Ablaufschritte werden bis zum Eintreten einer Abbruchbedingung wiederholt und sind:

1. **Status erhalten ?** / *client* +
2. **Status verarbeiten** / *client*: Überprüfung, ob der *server* in einer zuvor erfolgten Antwort einen Status an den *client* übermittelt hat und Verarbeitung dessen mittels eines *callbacks* (etwa indem die Darstellung aktualisiert wird).
3. ***Jobs* erhalten ?** / *client* +
4. ***Jobs* verarbeiten** / *client*: Wurden in einer zuvor erfolgten Antwort *Jobs* an den *client* übermittelt, werden die in der Abb. 6.17 dargestellten Einzelschritte durchgeführt:
   (a) **Modell laden**: Lädt die für die *Jobs* benötigten Modelle und den Trainingszustand deren aktuellster *Modellversion*.[29] Ist die aktuellste *Modellversion* nicht trainiert, werden erteilte *Prediction Jobs* ignoriert.
   (b) **Funktionen laden**: Lädt die für die Modelle benötigten *Modellfunktionen*[29] und führt folgende Operationen durch.
      i. Validieren der Funktion (*Eingangs-* und *Ausgangswerte*).
      ii. *Topologisches* Sortieren der Funktionen gemäß der Modellsspezifikationen.
      iii. Kompilieren der Funktionen und Einfügen der *Eingangs-* und *Ausgangswerte* sowie des bei asynchronen Funktionen definierten *Callback*.
   (c) ***Jobs* verarbeiten**: Iteriert über die erteilten Aufträge, fügt die Eingangswerte in die Funktionskette (siehe Abb. 6.2) des Modells ein und generiert die entsprechenden Rückgabewerte. Die Abarbeitung der einzelnen Aufträge erfolgt je nach Funktionstyp synchron oder asynchron und *single threaded*, wenn der *client* keinen

[29] Diese Daten werden nur einmal vom *server* geladen und in weiterer Folge aus dem *cache* generiert.

Zugriff auf den *Html5 WebWorker* hat, bzw. parallel, wenn diese Funktionalität zur Verfügung steht.

(d) **Generiere *Job*antwort**: Generiert ein Antwortobjekt an den *server* mit den *Job* Resultaten, d.h. den generierten Werten der *Prediction* Einträge und den *IDs* der jeweiligen Aufträge.

5. **Initial oder (*Jobs* erhalten und im Limit) ?** / *client* +
6. **Generiere *Job*anfrage** / *client*: Ist dies der initiale Aufruf des *Harvest* Zyklus bzw. hat ein zuvor erfolgter Aufruf Aufträge vom *server* zur Bearbeitung erhalten und die maximale Anzahl an *Harvest* Iterationen ist nicht überschritten, wird eine Anfrage für weitere *Jobs* an den *server* generiert.
7. **Initial oder kontinuierlicher Status ?** / *client* +
8. **Generiere Statusanfrage** / *client*: Beim erstmaligen (initialen) Aufruf des *Harvest* Zyklus, bzw. wenn eine Kennung (systemintern als *continuousStatusreport* bezeichnet) gesetzt ist, wird eine Statusanfrage generiert.
9. ***Job*anfrage oder Antwort existiert ?** / *client* +
10. **Anfrage senden** / *client*: Wurden Anfragen generiert bzw. wurden erteilte *Jobs* erfolgreich abgearbeitet, werden diese an den *server* übermittelt, ansonsten wechselt die Applikation in die *Sleep* Phase.
11. **Anfrage verarbeiten** / *server*: Die an den *server* geschickten Daten werden validiert und die Authentifizierung des *clients* überprüft. Eine Ausgliederung dieses Prozessschrittes ist in Abb. 6.18 ersichtlich.

    (a) **Hat *Job*antwort ?** +

    (b) **Validiere und speichere *Jobs***: Hat der *client Job* Resultate an den *server* zurückgegeben, wird überprüft, ob diese *Jobs* an den *client* delegiert wurden, indem die retournierten *Jobtypen* und die *IDs* der delegierten Einträge mit den in der Datenbank gespeicherten Werten verglichen werden und die *Jobwerte* auf die Richtigkeit ihrer Anzahl und ihrer Typen geprüft werden. Sind die Aufträge in der geforderten Zeit retourniert worden, werden diese als neue *Predictions* gespeichert bzw. bestehende *Predictions* angepasst.

    (c) **Statusanfrage oder Klassifizierung der aktuellen *Source* ?** +

    (d) **Generiere Statusantwort**: Wurde vom *client* eine Statusanfrage gestellt bzw. wird der dem Anwender aktuell angezeigte Eintrag einer (Re-)Klassifizierung unterzogen, wird der Status der aktuell angezeigten *Source* ermittelt und eine Antwort generiert.

    (e) **Ist eine *Job*anfrage ?** +

    (f) **Generiere *Job*antwort**: Wurden vom *client* neue Aufträge angefordert, so wird eine Anzahl *Jobs*, die nicht an andere *clients* de-

legiert wurden, ausgewählt und diese Werte (einschließlich deren *ID* und *Jobtyp*) in ein Antwort Objekt gespeichert. Diese erteilten Aufträge werden für eine durch das Modell definierte Dauer gesperrt und die Daten der Auswahl zu Validierungszwecken in der Datenbank gespeichert.

12. **Retourniere Antwort** / *server*: Sendet allfällige, generierte Antworten an den *client* bzw. eine leere Antwort, falls keine solchen erzeugt wurden.

Beispiel Nr.1

Eine *Harvest* Phase des vorliegenden Beispielszenarios könnte etwa (in einer vereinfachten Weise dargestellt) folgendermaßen aussehen:

Vorbedingung Zum aktuellen Zeitpunkt gelten die initialen Trainingsdaten als teilweise eingespielt, ein anderer *client* hat eben eine Anfrage (*pushTask*) erfolgreich übermittelt, um weitere *Tasks* zum Einspielen der restlichen Trainingsdaten zu starten. Die von dem für dieses Beispiel betrachteten *client* versendete Anfrage (*pushTask*) blieb unbeantwortet (da der *server* aktuell mit der Ausführung eines Tasks beschäftigt ist) woraufhin er in die *Harvest* Phase wechselt. Der *client* führt somit den *Harvest* Zyklus zum ersten mal (initial) aus, eine kontinuierliche Information über den Status des aktuell angezeigten Eintrags ist nicht gefordert (*continuousStatusreport*) und das Limit der maximal durchgeführten *Job*anfragen ist mit 10 festgelegt.

1. **client**
   (a) Der *client* hat weder einen Status *Job* noch Aufträge erhalten (initialer Aufruf der *Harvest* Phase).
   (b) Der *client* generiert eine *Job*anfrage (initialer Aufruf).
   (c) Der *client* generiert eine *Status*anfrage (initialer Aufruf).
   (d) Es wurde eine oder mehrere Anfragen definiert, d.h. diese werden an den *server* geschickt.
2. **server**
   (a) Der *server* überprüft, ob der *client* die Berechtigung hat eine Anfrage an den *server* zu übermitteln[30] und verarbeitet diese.

[30] Beim Generieren der HTML Seite wird vom *server* ein *hash* aus verschiedenen Werten (etwa dem Browser des *clients*, einem sich zu bestimmten Zeiten ändernden, im Sourcecode definiertem Text bzw. dem aktuellen *timestamp*) generiert, in den *HTML code* als Javascript Variable eingefügt und mittels der Anfrage an den *server* übertragen. Des weiteren generiert der *client* einen Teil dieser Daten selbstständig und sendet diese ebenfalls an den *server*. Hat dieser nun eine Anfrage erhalten, wird der intern generierte *hash* mit dem übertragenen *hash* und einem aus den *Client*daten generierten *hash* verglichen und überprüft, ob die Anfrage innerhalb eines Zeitfensters versendet wurde. Ist die Anfrage

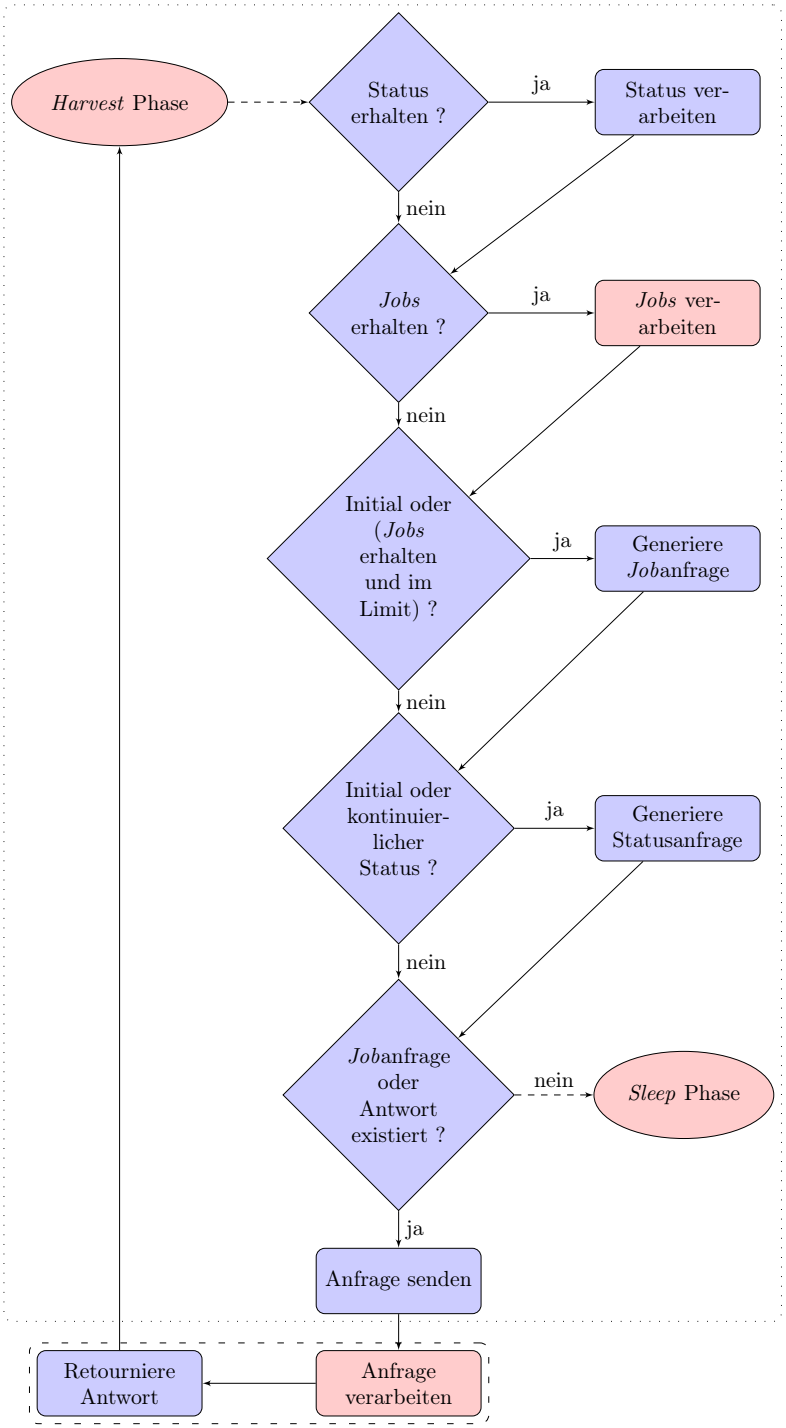


Abbildung 6.16: Schematische Darstellung der *Harvest* Phase und Übergang in die *Sleep* Phase. Der punktiert umrandete Bereich stellt die Prozesse des *clients* dar, der gestrichelt umrandete Bereich die Prozesse aufseiten des *servers*. Die Knoten *Jobs verarbeiten* und *Anfrage verarbeiten* werden in der Abb. 6.17 bzw. 6.18 gesondert dargestellt.

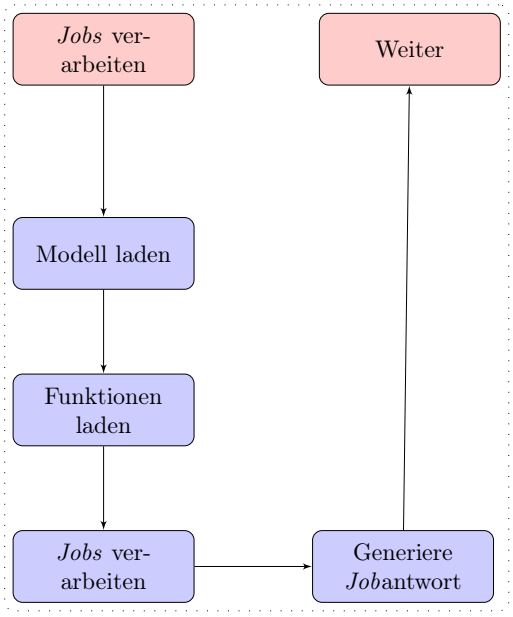


Abbildung 6.17: Teilstruktur des in der Abb. 6.16 definierten Knotens: *Jobs verarbeiten.*

(b) Es wurden keine fertigen *Jobs* vom *client* verschickt.

(c) Der *client* hat eine Statusanfrage mit der *ID* des aktuell angezeigten Eintrags geschickt. Da dieser Eintrag allerdings noch nicht bewertet wurde (die Modellversion ist zum aktuellen Zeitpunkt nicht trainiert und erlaubt damit noch keine Klassifizierungen) führt diese Anfrage zu keinem Ergebnis und liefert ein leeres Status Objekt zurück.

(d) Der *client* hat eine *Job*anfrage an den *server* übermittelt. Obwohl ein anderer *client* zum aktuellen Zeitpunkt noch *Tasks* zum Einfügen weiterer initialer Trainingsdaten am *server* startet, sind einige dieser Daten bereits gespeichert und mit *Input Jobs* versehen. Die Applikationslogik wählt die Einträge mit den *IDs*: *4*, *5* und *7* aus (der Eintrag mit der *ID 6* wurde an einen anderen *client* delegiert und ist für eine durch das Model definierte Dauer gesperrt), speichert diesen Auftrag (unter anderem die erteilten *IDs* und den Startzeitpunkt des Auftrags in der Datenbank) und generiert eine Antwort an den *client* mit den zu bearbeitenden Werten der *Source* Einträge und den auf sie verweisenden, nicht klassifizierten *Predictions* (die ja, wie bereits erwähnt, immer ge-

nicht valide, wird sie vom *server* abgewiesen und dieser Versuch zu Evaluierungszwecken in der *log* Datei vermerkt, ist die maximale Anfragedauer überschritten muss die Seite erneut vom *server* geladen werden.

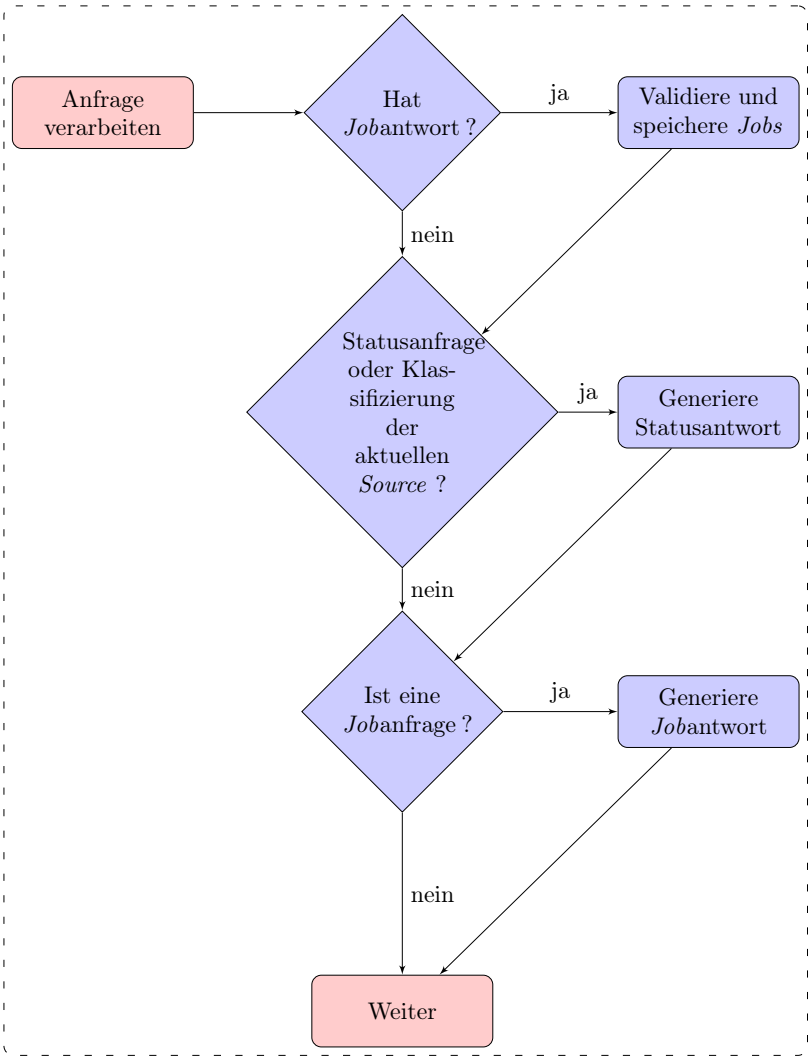


Abbildung 6.18: Teilstruktur des in der Abb. 6.16 definierten Knotens: *Anfrage verarbeiten*.

nau eine *Source* und ein Modell referenzieren).

(e) Der *server* retourniert die generierten Antworten an den *client*.

3. **Client**

(a) Der *client* hat keinen Status erhalten (bzw. ein leeres Status Objekt).

(b) Es wurden vom *server* *Jobs* an den *client* übermittelt, die dieser nun bearbeitet.

i. Der *client* lädt das verwendete Modell vom *server* (es wurde

noch nicht geladen und befindet sich somit nicht im *cache*).

ii. Die für das Modell benötigten Funktionen werden geladen (und diese wie bereits beschrieben validiert, sortiert und kompiliert).

iii. Die Werte der übermittelten *Source* Einträge werden in die Funktionskette eingespeist und die relevanten Daten für die *Predictions* daraus generiert.

iv. Alle so generierten Werte werden (inklusive der entsprechenden, durch den *server* übermittelten *IDs*) in ein Antwort Objekt verpackt.

(c) Der *client* generiert eine *Job*anfrage (Es wurden *Tasks* vom *server* erteilt und die Anfrageanzahl befindet sich noch im definierten Limit von 10 Anfragen).

(d) Da dies nicht mehr der initiale Aufruf ist und der *client* nicht kontinuierlich den Status des aktuellen Eintrags anfordert, wird keine Statusanfrage generiert.

(e) Es wurde eine oder mehrere Anfragen definiert, d.h. diese werden an den *server* geschickt.

4. **server**

(a) Der *server* überprüft die Berechtigung des *clients* in der zuvor beschriebenen Art und Weise.

(b) Es wurden erledigte Aufträge an den *server* übermittelt. Dieser validiert sie folgendermaßen:

i. Wurde ein Auftrag an den *client* mit diesen *IDs* erteilt ?

ii. Entsprechen die *Jobtypen* der zurückgegebenen Daten denen die zuvor in der Datenbank zu Validierungszwecken gespeichert wurden ?

iii. Stimmen die generierten Werte eines zurückgegebenen Auftrags mit den im Modell definierten bezüglich ihrer Anzahl und ihres Typs überein ?

iv. Ist die Rückgabe der Aufträge an den *server* innerhalb der erlaubten Zeit erfolgt ?

Im Erfolgsfall speichert der *server* die Daten in den bereits durch den initialen Trainingsdatensatz klassifizierten *Predictions* und entfernt deren *Input Jobs*.

(c) Es wurde keine Statusanfrage übermittelt.

(d) Als Reaktion zu der übermittelten *Job*anfrage wird von der Applikationslogik ein Antwort Objekt mit Aufträgen zu den *IDs*: *6*[31],

[31] Der zuvor an einen anderen *client* erteilte Auftrag wurde nicht innerhalb des definierten Zeitrahmens zurückgegeben.

8 und *9* generiert.

(e) Der *server* retourniert die generierte Antwort an den *client*.

Beispiel Nr.2

Ein weiteres Beispiel für den Verlauf einer *Harvest* Phase im Rahmen des Beispielszenarios, welches die Klassifizierung von neuen, zur Laufzeit gewonnen Daten veranschaulichen soll, wird in Folge angeführt.

Vorbedingung Hier gelten die initialen Trainingsdaten als vollständig eingespielt, alle mit ihnen verknüpften *Input Jobs* als erledigt und die aktuelle Modellversion des verwendeten Modells als trainiert, was bedeutet, dass neue Einträge klassifiziert werden können. Zum aktuellen Zeitpunkt der durch den *client* durchgeführten Anfrage liegen weder *Tasks* noch *Jobs* vor, die an *clients* delegiert werden können. Beim Laden der Seite durch den *client*[32] wird eine eindeutige *ID* (die *URL* der entsprechenden Seite) mit den in der Datenbank existierenden Einträgen verglichen und der als neu identifizierte Eintrag in Form einer *Source* mit besagter *ID* und dem Text des Postings als Eingangswert generiert.[33] Des weiteren wird eine *Prediction* ohne Klassifizierung und Werte erzeugt welche die zuvor erzeugte *Source* und das verwendete Modell referenziert und mit einem *Prediction Job* versehen.

Der in Folge beschriebene *workflow* ist in wesentlichen Bereichen dem zuvor Beschriebenen ähnlich. Dementsprechend werden auch nur die wesentlichen Unterschiede beschrieben.

1. **client**
   Im initialen Aufruf der *Harvest* Phase generiert der *client* wie im zuvor beschriebenen Beispiel eine *Job*anfrage und übermittelt diese an den *server*.
2. **server**
   Der *server* empfängt und validiert die Anfrage des *clients*, generiert ein leeres Statusobjekt (der aktuell angezeigte Eintrag aufseiten des *clients* ist noch nicht bewertet) und verarbeitet die *Job*anfrage. Aufgrund der in Abschn. 6.2.4 dargelegten Priorisierung wird der zuvor definierte *Prediction Job* an den *client* delegiert. Der weitere *server*seitige Ablauf entspricht dem im Beispiel zuvor Beschriebenen.
3. **client**
   Die vom *server* übermittelte Antwort wird vom *client* in exakt der gleichen Art wie zuvor geschildert verarbeitet, die entsprechende *Job*antwort

[32] In diesem Beispiel wird ein neuer Posting Eintrag aufgerufen.

[33] Wie in Prog. 6.1 ersichtlich, wird der für die Generierung einer durch das verwendete Modell klassifizierbaren *Source* notwendige zweite Eingangswert (der Wert *$safebrowsingEndpoint*) manuell hinzugefügt. Die erwähnte *ID* ist hier durch den Parameter *$uniqueSourceID* symbolisiert.

und eine erneute Anfrage generiert (der *client* generiert wie zuvor keine erneute Statusanfrage) und diese Daten an den *server* transferiert.

4. **server**
   Die Handhabung der durch den *client* übertragenen Daten wird vom *server* wie bereits beschrieben vorgenommen. Der wesentliche Unterschied liegt hier im Speichern der durch den *client* übertragenen *Job* Daten. Da der *server* den zuvor an den *client* delegierten Auftrag als *Prediction Job* identifiziert, wird vor dem Speichervorgang eine Anfrage an das *Google Prediction Service* übermittelt und die hierbei generierte Klassifikation sowie die durch den *client* erzeugten *Job*daten in der entsprechenden *Prediction* gespeichert. Da eine Klassifizierung des aktuellen am *client* ersichtlichen Eintrags vorgenommen wurde, wird trotz des Umstandes, dass keine Statusanfrage übermittelt wurde ein entsprechendes Status Objekt generiert und schließlich eine Antwort aller Daten an den *client* übermittelt.

6.4.3 *Sleep* Phase

Abschließend bildet diese Phase den inaktiven Teil des am *client* sich wiederholenden Prozess Zyklus. Der Wechsel in diese Phase kann unter den folgenden Bedingungen stattfinden:

1. Auftreten eines behandelbaren Fehlers aufseiten des *clients* (siehe Abschn. 6.4).
2. Die maximale Anzahl an *Job*anfragen in der *Harvest* Phase wurde überschritten, es wurden keine Aufträge vom *server* an den *client* delegiert und keine *Job* Daten zur Rückmeldung an den *server* generiert (siehe Abb. 6.16).

Die Dauer dieser Phase kann wahlweise durch den *client* definiert oder ein von der Applikationslogik vorgegebener Wert übernommen werden. Sinn dieser Ruhephase ist primär das Vermeiden eines Programmabbruchs durch den verwendeten Browser aufgrund einer überschrittenen maximalen Scriptlaufzeit. Ist die Wartefrist der *Sleep* Phase abgelaufen, wechselt der *client*seitige Prozess Zyklus erneut in die *Task* Phase.

6.5 Sicherheit

Wie bereits in den vorhergegangenen Kapiteln angemerkt wurden zahlreiche Sicherheitsvorkehrungen getroffen, um die durch den *client* übermittelten Daten als valide zu verifizieren bzw. um Anfragen des *clients* an die *server*seitige Applikationslogik auf ihre Legitimität zu prüfen.

Authentifizierung der Anfrage

Sobald ein *client* eine Seite lädt, wird ein Authentifizierungswert (in der Form eines *MD5 hash*[34]) generiert und dem *client*seitigen Teil der Applikation als Variable übergeben (dieser Wert wird nicht in der *session* gespeichert). Die für die Generierung verwendeten Werte sind:

1. **Ein durch den *client* definierter Wert**: Eine Identifikation des die aktuelle Anfrage verschickenden Browsertyps (*ermittelt aus dem HTTP header*).
2. **Ein statischer *server*seitiger Wert**: Ein definierter Text (als *shiflett* bezeichnet), welcher von einer Textdatei die sich außerhalb des *root* Verzeichnisses befindet geladen wird.
3. **Ein aus der Datenbank geladener Wert**: Die aktuell verwendete Modell *ID*.
4. **Ein zur Laufzeit durch den *server* generierter Wert**: Die *ID* des zu diesem Zeitpunkt angezeigten Eintrags (*server*seitig als *Source gespeichert*).
5. **Die Gültigkeitsdauer der Verbindung**: Ein Wert, der den Zeitraum weiterer valider Anfragen definiert. Die Berechnung dieses Wertes erfolgt mittels der Gleichung

$$valid = \left\lceil \frac{time}{duration} \right\rceil,$$

wobei *time* der aktuelle *UNIX timestamp* in Sekunden ist und *duration* die Gültigkeitsdauer der Anfragen in Sekunden definiert. Der resultierende Wert *valid* wird bei beim initialen Aufruf der Applikationslogik generiert und als Parameter zur Generierung des Authentifizierungswertes verwendet.

Versendet der *client* eine Anfrage an den *server*, so wird besagter Authentifizierungswert sowie die Modell *ID* und die Identifikation der aktuell angezeigten *Source* in der Anfrage ebenfalls übermittelt. Der *server* generiert nun erneut zwei verschiedene Versionen des Authentifizierungswertes mit den oben genannten Parametern (der Browsertyp wird erneut aus dem *HTTP header* ermittelt) und dem Parameter *valid*, sowie einem Wert der $valid - 1$ entspricht. Stimmt der durch den *client* übermittelte *MD5 hash* mit einem dieser beiden überein, wird die Anfrage des *clients* angenommen. Das maximale Zeitfenster, in dem eine Anfrage als gültig bewertet wird ist somit definiert als $[duration, duration \cdot 2[$.

[34] Die Verschlüsselung mittels *MD5 hash* ist eine typische Herangehensweise, um zu verhindern, dass aus als Klartext übertragenen Daten ein Rückschluss über deren Generierung gewonnen werden kann. Ein einmal derart verschlüsselter Text kann (zumindest in der Theorie) nur sehr schwer wieder rückwärts entschlüsselt werden. Wird ein zweites Mal der selbe Text in einen *MD5 hash* umgewandelt ist dieser jedoch mit dem ersten Resultat identisch.

Sessions

1. **Session *Fixation*** Um zu verhindern, dass ein potentieller Angreifer einen anderen Anwender mit einer zuvor generierten *session ID* vortäuscht, wird bei jedem Aufruf der *server*seitigen Applikationslogik auf das Vorhandensein einer *session* Kennung geprüft und die Sitzung, sollte diese Kennung nicht existieren, mittels *session_ regenerate_ id()* regeneriert.
2. **Session *Hijacking*** Sollte ein Angreifer an eine valide *session ID* gelangt sein, wird ein (wie eingangs beschrieben) aus multiplen Daten generierter valider Authentifizierungswert vom *server* für die Bearbeitung der Anfrage vorausgesetzt. Indem dieser Wert sowohl aus statischen Daten, als auch dynamisch generierten Daten (*client-* und *server*seitig) besteht und die Gültigkeitsdauer einer Verbindung beschränkt ist, wird dieser Bedrohung Rechnung getragen.
3. **Session *Prediction*** Der durch PHP generiert 32-stellige *session* Schlüssel ist mittlerweile relativ einfach mittels *brute-force* zu knacken. Durch die komplexe Struktur des übermittelten Authentifizierungswertes und die eingeschränkte Gültigkeit der Sitzung wird dieser Gefahr Rechnung getragen.

Validierung der durch den *client* generierten Daten

Hat ein *client Job* Daten generiert und werden diese an den *server* zurückgegeben, so werden die *IDs* der retournierten *Jobs* mit den bei der Auftragserteilung gespeicherten *IDs* verglichen und die Ergebnisse nur dann gespeichert, wenn diese übereinstimmen.

Datenbankstruktur und Konventionen

1. *Source* Einträge werden ausschließlich durch die Applikationslogik erzeugt und gespeichert (keine durch den Anwender generierte Daten).
2. Die durch den *client* generierten *Prediction* Werte dürfen nur aus *alphanumerischen* Zeichen bestehen und die Klassifizierungen müssen durch das Modell erlaubt sein.
3. Liefert ein Anwender häufig fehlerhafte Daten (oder versucht er etwa *Jobs* mit *IDs* zurückzuliefern, die ihm nicht zuvor erteilt wurden), so wird dieser Umstand in der *log* Datei vermerkt und dieser Anwender kann mit einer Sperre belegt werden.
4. Modelle und Modellfunktionen befinden sich in einem getrennten Bereich der Datenbank, für dessen Änderung und Bearbeitung ein eigener *MySQL user* mit entsprechenden Rechten definiert wurde.
5. Sämtliche Datenbank Einträge werden auf schädliche Scriptelemente überprüft und mittels *prepared statements* eingefügt.

Kapitel 7

Analyse

Um das entwickelte System hinsichtlich seiner Verwendbarkeit und Funktionalität bzw. den in dieser Arbeit veranschlagen Lösungsansatz zu prüfen, wurden drei verschiedene Aspekte der vorliegenden Umsetzung evaluiert:

1. Die Erwartungshaltung und das Interesse der Entwickler bzw. ihren Wunsch einen derartigen Mechanismus zu implementieren.
2. Die Funktionalität und Leistungsfähigkeit des Systems auf verschiedenen Plattformen.
3. Die Qualität der durch das System generierten Klassifizierungen.

Die im Rahmen dieses Prozesses verwendete Herangehensweise soll in den folgenden drei Abschnitten erläutert und die hierbei gewonnen Erkenntnisse diskutiert und in grafischer bzw. tabellarischer Form dargestellt werden. An dieser Stelle ist den an der Studie beteiligten Personen noch der ausdrückliche Dank für ihre Bereitschaft zur Mitarbeit auszusprechen und im Zuge dessen besonders die Mitarbeit von Stefan Moosbauer und Christoph Zeller hervorzuheben, ohne die die Evaluierung der verschiedenen Plattformen nicht möglich gewesen wäre.

7.1 Erwartungshaltung der Entwickler

7.1.1 Methodik und Vorgehen

In Anbetracht der in Kapitel 5 getroffenen Entscheidung, das System als *Webanwendung* umzusetzen, gilt es zunächst zu prüfen, ob ein derart gestaltetes System geeignet ist, ein Bedürfnis der Entwickler zu befriedigen. Dieser Teil der Evaluierung zielt also nicht auf die technischen Aspekte der Implementierung ab, sondern soll die Erwartungshaltung von Anwendern ergründen. Zu diesem Zweck wurde eine Befragung von Entwicklern anhand eines Fragenkatalogs durchgeführt. Als Form wurde eine *geschlossene Fragestellung* gewählt, bei der die Teilnehmer aus einer Reihe von vorgegebenen Antworten auswählen können. Zum Zweck der statistischen Messbarkeit der

Ergebnisse kam eine sogenannte *Likert Skala* zum Einsatz [44], bei der die Zustimmung zu einer durch den Studienverantwortlichen definierten Aussage gemessen wird. Im vorliegenden Fragebogen wurde eine fünfstufige Skala mit je zwei zustimmenden und zwei ablehnenden Antwortvorgaben verwendet, sowie einer neutralen Option. Durch diese Auswahl wurde ein breites Spektrum der möglichen Antwortvarianten abgedeckt und eine erzwungene Positionierung der Teilnehmer im zustimmenden oder ablehnenden Bereich durch eine neutrale Wahlmöglichkeit verhindert. Um ein Maß für den Grad der insgesamt abgegebenen Zustimmung bestimmen zu können, wurden die abgegebenen Bewertungen im Sinne eines Schulnotensystems interpretiert und das *arithmetische Mittel* aus diesen Werten generiert. Dies ist zwar insofern mathematisch nicht korrekt, als es sich streng genommen bei der *Likert Skala* um eine *Ordinalskala* handelt, die beschriebene Interpretationsweise ist aber eine gängige Methode um derartige Umfragen zu bewerten, unter der Bedingung, dass die Antwortvorgaben annähernd eine gleiche Wertentfernung haben, was bei Likert Ausprägungen (engl. *items*) ja auch der Fall sein sollte. Der derart ermittelte Durchschnittswert wird auf eine Kommastelle kaufmännisch gerundet angegeben.

7.1.2 Teilnehmer und Testablauf

Die befragte Personengruppe umfasst 12 Entwickler mit einem Durchschnittsalter von 26 Jahren und setzt sich aus 1 Frau bzw. 11 Männern zusammen. Die Studienteilnehmer wurden so ausgewählt, dass sie durchwegs Erfahrungen im Bereich der Webentwicklung generell und mit den in der Implementierung verwendeten Technologien im Besonderen besaßen. Im Vorfeld der Befragung wurden die beteiligten Akteure mit der grundsätzlichen Terminologie vertraut gemacht und ihnen anhand der im Rahmen des Projektes erstellten exemplarische Integrierung der Funktionsbibliothek in das *CMS Wordpress* die Funktionalität erläutert. Die Beschreibung möglicher Einsatzszenarien wurde zum Zweck der leichteren Verständlichkeit auf die *automatisierte Klassifikation von Textinhalten* im Internet verengt.

In weiterer Folge wurden die Teilnehmer mit den folgenden Aussagen konfrontiert und gebeten den Grad ihrer Zustimmung zu diesen abzugeben:

1. Die angebotene Funktionalität ist nützlich.
2. Ich würde ein solches System verwenden.
3. Einem automatisierten Bewertungsmechanismus vertraue ich.

Die eingangs bereits erwähnte fünfstufige *Likert Skala* umfasste folgende Antwortmöglichkeiten:[1]

1. Ich stimme der Aussage *voll* zu (entsprechend der Note 1).
2. Ich stimme der Aussage *eher* zu (entsprechend der Note 2).

[1] Entsprechend der eingangs getroffenen Konvention wird die der Aussage zugeordnete Benotung zwecks der Veranschaulichung beigefügt.

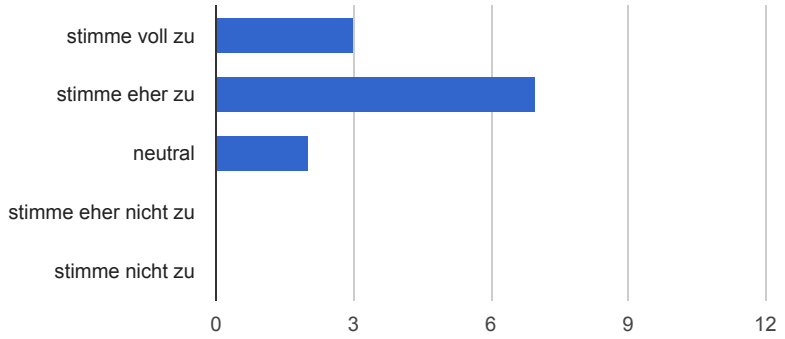


Abbildung 7.1: Antworten zur Fragestellung: *Die angebotene Funktionalität ist nützlich* ($\varnothing = 1.9$).

3. Ich vertrete eine *neutrale* Position (entsprechend der Note 3).
4. Ich stimme der Aussage *eher nicht* zu (entsprechend der Note 4).
5. Ich stimme der Aussage *nicht* zu (entsprechend der Note 5).

7.1.3 Resultate der Befragung

Die in Kapitel 1 identifizierten Problematiken äußern sich ja auch vor allem in dem Umstand, dass die Verwendung automatisierter Klassifizierungsmechanismen aufgrund der geforderten zeitlichen, finanziellen und technischen Ressourcen für die meisten Entwickler und Organisationen nicht möglich ist. Die dementsprechend formulierte Anforderung, eine plattformübergreifende Lösung zu entwickeln und die Expertentätigkeit an eine Schar von Internetbenutzern auszulagern, bedingt natürlich vorweg, dass potenzielle Anwender dieses Systems ein Interesse an dessen Funktionalität haben bzw. den Wunsch verspüren, diese auch einzusetzen.

Wie in Abb. 7.1 ersichtlich ist, erscheint die Nützlichkeit eines automatisierten Klassifizierungssystems für den überwiegenden Teil der befragten Entwickler gegeben. Die in Abb. 7.2 abgebildeten Ergebnisse zeichnen ein entsprechendes Bild und veranschaulichen ein vorhandenes Interesse der Entwickler ein derart gestaltetes System zu verwenden. Bei der Fragestellung bezüglich des Vertrauens, das die Studienteilnehmer den derart generierten Klassifizierungen entgegenbringen, zeichnet sich schon ein differenzierteres Bild ab (siehe Abb. 7.3). Die Bereitschaft, diese etwa auch ohne manuelle Prüfung zu verwenden scheint nicht in demselben Ausmaß gegeben wie das zuvor ermittelte Interesse. Ein Interpretationsansatz für diesen Umstand kann sein, dass die Technik der automatischen Klassifizierung zwar im alltäg-

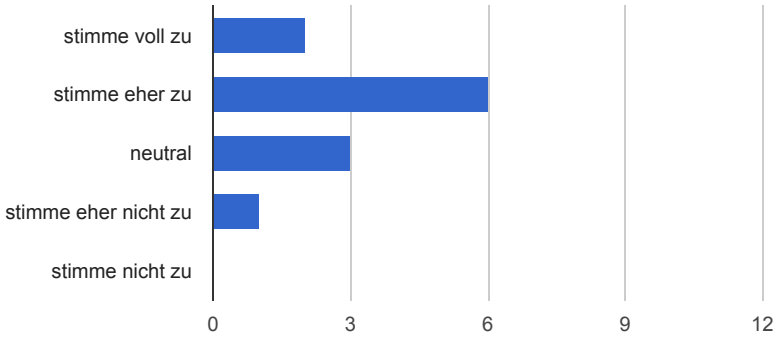


Abbildung 7.2: Antworten zur Fragestellung: *Ich würde ein solches System verwenden* ($\varnothing = 2.3$).

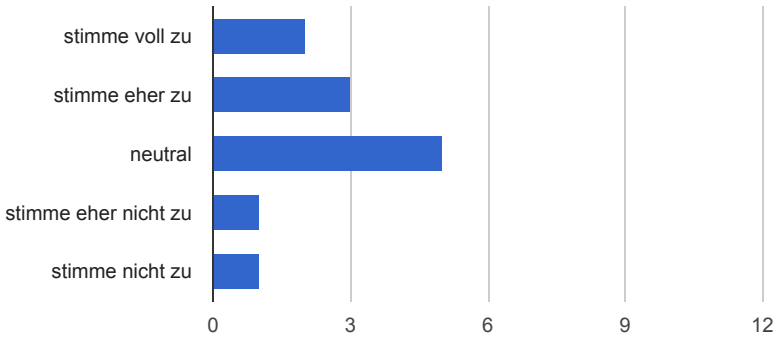


Abbildung 7.3: Antworten zur Fragestellung: *Einem automatisierten Bewertungsmechanismus vertraue ich* ($\varnothing = 2.7$).

lichen Gebrauch, etwa bei Suchmaschinen Einzug gehalten hat, die wenigsten Entwickler aber Erfahrungen mit der Verwendung derartiger Systeme in ihren Arbeiten haben und dementsprechend noch ein gewisses Misstrauen besteht.

Tabelle 7.1: Überprüfung der Funktionalität auf verschiedenen Betriebssystemen und Browsern mittels 50 an den *client* delegierten *Jobs*.

Betriebssystem	*Browser*	*Fehlerhafte Jobs*	*Durschschnittliche Dauer*
Win7	IE 9	0	0.53 sec
Win7	FF 16	0	0.51 sec
WinXP	IE 8	0	0.67 sec
WinXP	FF 16	0	0.55 sec
WinXP	Chrome 19	0	0.53 sec
Mac OS X	Chrome 23	0	0.35 sec
Mac OS X	Safari 6.0	0	0.32 sec
iPhone OS 5	Mobile Safari 5.1	0	0.73 sec
Android 4.0	Default	0	0.62 sec
Android 4.0	Opera 9.8	0	0.71 sec
Android 4.0	Chrome 18	0	0.84 sec

7.2 Funktionalität auf verschiedenen Systemen

Um die im Rahmen dieser Arbeit definierten Anforderung, eine auf multiplen Plattformen funktionsfähige Webanwendung umzusetzen zu prüfen und die Leistungsfähigkeit des Systems zu evaluieren, wurde es auf den aktuell am meisten verwendeten Browsern und Betriebssystemen getestet.[2] Zu diesem Zweck wurde die erstellte Integrierung der Funktionsbibliothek in das *CMS Wordpress* und das dabei verwendete *Spam Identifikations Modell* (siehe Kapitel 6) unter den verschiedensten Systemen aufgerufen und in der *Harvest* Phase insgesamt 200 Einträge an den *client* delegiert, wobei pro *Job* Anfrage 4 *Sources* übermittelt wurden, d.h. dem *client* 50 Aufträge zur Bearbeitung erteilt wurden. Die in Tab. 7.1 dargestellten Werte wurden zum Teil aus den für diesen Zweck gespeicherten Validierungsdaten der *Harvest* Tabelle entnommen (siehe *PredictionManager*, Abschn. 6.3.1) und sind:

1. **Betriebssystem**: Das verwendete Betriebssystem.
2. **Browser**: Der verwendete Browser.
3. **Fehlerhafte Jobs**: Die Anzahl der Aufträge, die nicht ordnungsgemäß durch den *client* abgearbeitet oder retourniert wurden.

[2]Diese sind etwa unter http://www.w3schools.com/browsers/browsers_stats.asp und http://www.w3schools.com/browsers/browsers_os.asp ersichtlich.

Tabelle 7.2: Überprüfung der Klassifizierungszuverlässigkeit des *Spam Identifikations Modells* mittels 200 Validierungsdaten und Vergleich mit einem durch unveränderte *Postingtexte* trainierten Modell.

Modell	*Fehlgeschlagene Klassifizierung*	*Prozentsatz*
Spam Identifikations Modell	4	2.5%
Referenz Modell	19	9.5%

4. **Durschschnittliche Dauer**: Die durchschnittliche Dauer eines Auftrages, gemessen von Auftragserteilung durch den *server* bis zur erfolgreichen Verarbeitung des *servers* der durch den *client* generierten Werte.

Aus den mittels dieser Testreihe gewonnenen Ergebnissen ist klar ersichtlich, dass die erstellte Funktionsbibliothek erwartungsgemäß auf den gängigsten Betriebssystemen und Browsern fehlerfrei funktioniert. Die durchschnittliche *Job* Dauer ist dabei natürlich aufgrund der verschiedenen Übertragungsgeschwindigkeiten der Internetverbindung nur bedingt ein repräsentativer Wert, ist aber ausreichend, um die performante Arbeitsweise des Systems zu verdeutlichen.

7.3 Qualität der generierten Klassifizierungen

Um die Qualität der durch das System generierten Klassifizierungen zu überprüfen, wurde exemplarisch am *Spam Identifikations Modell* eine Evaluierung mittels eines *Validierungsdatensatzes* durchgeführt. Zu diesem Zweck wird ein Teil der zur Verfügung stehenden Trainingsdaten nicht zum eigentlichen Trainingsvorgang herangezogen, sondern sie bzw. die ihnen zugrunde liegenden Attribute durch das trainierte Modell klassifiziert und diese Resultate mit den bereits bestehenden Klassifizierungen des Datenkorpus verglichen. Der zur Übersichtlichkeit des Vorgangs verwendete Datensatz ist auf der Internetseite der ILPS Gruppe[3] verfügbar. Weiters wurde ein *Referenzmodell* erstellt, zu dessen Training nicht die durch das vorliegende System extrahierten Attribute der *Postingtexte* (siehe die Erläuterungen zum Beispielszenario in Abschn. 6.2.1) verwendet wurden, sondern die unveränderten Originaltexte.

Die in Tab. 7.2 dargestellten Testergebnisse dokumentieren die Zuverlässigkeit der generierten Klassifizierungen zum einen und zum anderen auch die Überlegenheit des durch das System verwalteten Modells gegenüber jenem Modell, welches in herkömmlicher Art und Weise trainiert wurde.

[3] http://ilps.science.uva.nl/resources/commentspam

Kapitel 8

Schlussbemerkung

8.1 Zusammenfassung

In der vorliegenden Arbeit wurde der Begriff des *überwachten maschinellen Lernens* zwecks der automatisierten Klassifizierung von Datensätzen thematisiert und die gängige Herangehensweise, die hierfür benötigten Trainingsdaten durch Experten zu generieren kritisiert. Als Alternativvorschlag wurde die Delegierung dieser Aufgabe an eine Gruppe von Laien im Internet vorgestellt und die diesem Vorschlag zugrunde liegende Motivation beschrieben. Zu diesem Zweck wurden eingangs die mit der Materie verbundenen Fachbegriffe erläutert und der für diese Auslagerung verwendete Begriff *Crowdsourcing* in einen entsprechenden Kontext gesetzt. In weiterer Folge wurde das neben dem Zeit- und Kostenfaktor zentrale Element dieses Prozesses, die *kollektive Intelligenz* der die Daten generierenden Gruppe beleuchtet und im Vergleich dazu die Gründe für fehlerhafte Expertenbeurteilungen analysiert. In den folgenden Abschnitten erfolgte die Konzeptionierung eines Systems, welches sowohl imstande ist den veranschlagten Prozess durchzuführen, als auch im Sinne des *Dezentralisierungsgedankens* der *kollektiven Intelligenz* die Berechnung der benötigten Datenattribute an die verbundenen Benutzer zu delegieren. Im abschließenden Teil wurde das zuvor entwickelte System einer Prüfung unterzogen und der in dieser Arbeit beschrittene Lösungsweg analysiert.

8.2 Fazit und Ausblick

Das Generieren von Trainingsdaten für *überwachtes maschinelles Lernen* ist, sowohl aufgrund der benötigten Menge an Daten, als auch der in diesem Vorgang involvierten Entscheidungsprozessen ein hochkomplexer, teurer und zeitintensiver Vorgang. Der in der vorliegenden Arbeit veranschlagte Weg, diesen durch eine Auslagerung an eine Schar von Internetbenutzern durchführen zu lassen, scheint geeigneter die Verwendung von automatisierten

Klassifizierungsmechanismen schlussendlich auch einem größeren Kreis von Organisationen zur Verfügung zu stellen. Wie gezeigt werden konnte, ist das Web nicht nur aufgrund der einfachen Erreichbarkeit der dafür benötigten Personengruppe geeignet, sondern auch durch die breite Streuung dieser und der damit verbundenen *kollektiven Problemlösungsfähigkeit*. Die durch diesen Prozess erfolgende Wissensgenerierung erscheint als ein geeignetes Mittel, um die zunehmende Datenflut der modernen Informationsgesellschaft zu verarbeiten, kann aber auch selbst zu einer Verschärfung der Problematik führen, wenn das Instrument in einer unreflektierten Art eingesetzt wird. Wenn der Fokus nicht auf einer *Reduktion* der Informationen auf ihre wesentlichen Bestandteile liegt, sondern vielmehr wahllos neue Erkenntnisse produziert werden, werden schlussendlich auch die leistungsfähigsten Klassifizierungsalgorithmen den *information overflow* nicht verhindern können.

Anhang A

Inhalt der CD-ROM/DVD

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 Masterarbeit

Pfad: /

DA.pdf	Masterarbeit (Gesamtdokument)
/hgb-thesis-20120510 .	TeXnicCenter Projekt (LaTeX-Quelldateien)

A.2 Online-Quellen (PDF)

Pfad: /online

likert_skala.pdf	Artikel (Wikipedia)
female_smarter.pdf . .	Interview (Harvard Business Review)
social_over_iq.pdf . . .	Artikel (Harvard Business Review)
trolle.pdf	Artikel (Zeit Online)
advocatus_diaboli.pdf .	Artikel (Neue Zürcher Zeitung)
swarm_intelligence.pdf	Artikel (Wikipedia)
narrow_framing.pdf . .	Blogposting
gruppendenken.pdf . . .	Artikel (Wikipedia)
self_affirmation.pdf . .	Artikel (Wikipedia)
kratylos.pdf	Artikel (Wikipedia)
sander_illusion.pdf . . .	Artikel (Wikipedia)
groessen_konstanz.pdf	Artikel (Wikipedia)
interview_faz_lanier.pdf	Interview (F.A.Z.)
bud_spencer_tunnel.pdf	Artikel (Spiegel Online)
curse_of_dimensionality.pdf	Artikel (Wikipedia)

A.3 Projekt

Pfad: /projekt

/modell_tool	Modell Tool
/wordpress_integrierung	Wordpress Integrierung der Funktionsbibliothek

A.4 Sonstiges

Pfad: /hgb-thesis-20120510/images

*.pdf	PDF Dateien
*.png	PNG Dateien
*.svg	SVG Dateien

Quellenverzeichnis

Literatur

[1] Y. Benkler. *The Wealth of Networks: How Social Production Transforms Market and Freedom*. Yale: Yale University Press, 2006.

[2] F. Cube. *Gefährliche Sicherheit – Die Verhaltensbiologie des Risikos*. Stuttgart: Hirzel, 1990.

[3] Catharina van Delden. „Crowdslapping: Wenn Crowdsourcing schiefläuft". In: *planung & analyse* 5 (2012), S. 83–85.

[4] C. Fine und S. Held. *Die Geschlechterlüge: Die Macht der Vorurteile über Mann und Frau*. Klett-Cotta, 2012.

[5] J. Freeman. *The Tyranny of E-mail: The Four-Thousand-Year Journey to Your Inbox*. Simon & Schuster, 2009.

[6] E. Hippel und C Baldwin. „Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation". In: *Organization Science* 22 (Dezember 2011), S. 1399–1417.

[7] Jeff Howe. „The Rise of Crowdsourcing". In: *Wired Magazine* 6 (2006), S. 1–4.

[8] D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus und Giroux, 2011.

[9] Sotiris B. Kotsiantis. „Supervised Machine Learning: A Review of Classification Techniques". In: *Emerging Artificial Intelligence Applications in Computer Engineering*. Amsterdam: IOS Press, 2007, S. 3–24.

[10] J. Lanier. *You Are Not a Gadget: A Manifesto*. New York: Alfred A. Knopf, 2010.

[11] Konrad Zacharias Lorenz. *Die Rückseite des Spiegels: Versuch einer Naturgeschichte menschlichen Erkennens*. München, Zürich: Piper, 1973.

[12] J. Lorenza u. a. „How social influence can undermine the wisdom of crowd effect". In: *Proceedings of the National Academy of Sciences*. Hrsg. von Burton H. Singer. National Academy of Sciences, Mai 2011, S. 9020–9025.

[13] N. Martin, S. Lessmann und S. Voß. „Crowdsourcing: Systematisierung praktischer Ausprägungen und verwandter Konzepte“. In: *Multikonferenz Wirtschaftsinformatik 2008*. (München). Berlin: GITO mbH Verlag, 2008, S. 273–274.

[14] M. McLuhan und B. Nevitt. *Take today; the executive as dropout*. Harcourt Brace Jovanovich, 1972.

[15] J. Naisbitt. *Megatrends: Ten New Directions Transforming Our Lives*. Grand Central Publishing, 1982.

[16] Christian Papsdorf. *Wie Surfen zur Arbeit wird: Crowdsourcing im Web 2.0*. Frankfurt am Main/New York: Campus Verlag, 2009.

[17] J. Perdijon. *Das Mass in Wissenschaft und Philosophie: Ausführungen zum besseren Verständnis*. Übers. von I. Flegel. Lübbe, 2001.

[18] K. Popper und I. Fleischmann. *Objektive Erkenntnis : ein evolutionärer Entwurf*. Hamburg: Hoffmann und Campe, 1995.

[19] K. Rost. „Der Einfluss von Erfindernetzwerken auf die Relevanz von Patenten“. In: *ZfbF Schmalenbachs Zeitschrift für betriebswirtschaftliche Forschung* (2006), S. 363–389.

[20] H.-M. Süß. „Eine Intelligenz – viele Intelligenzen ? Neuere Intelligenztheorien im Widerstreit“. In: *news&science* 15 (Jänner 2007), S. 18–27.

[21] Jeffrey M. Stibel. *Wired for Thought: How the Brain is Shaping the Future of the Internet*. Boston, Massachusetts: Harvard Business Press, 2009.

[22] J. Surowiecki. *The Wisdom of Crowds*. Knopf Doubleday Publishing Group, 2005.

[23] D. Tapscott. *Growing up digital: the rise of the net generation*. McGraw-Hill, 1998.

[24] A. Toffler. *The Third Wave*. Bantam Books, 1980.

[25] Anita W. Woolley u. a. „Evidence for a Collective Intelligence Factor in the Performance of Human Groups“. In: *Science* 330 (Okt. 2010), S. 686–688.

Online-Quellen

[26] Kopie auf CD-ROM (Datei online/curse_of_dimensionality.pdf). URL: http://en.wikipedia.org/wiki/Curse_of_dimensionality (besucht am 24. 10.2012).

[27] Kopie auf CD-ROM (Datei online/bud_spencer_tunnel.pdf). URL: http://www.spiegel.de/panorama/bud-spencer-tunnel-in-schwaben-der-dampfhammer-kommt-a-775983.html (besucht am 11.09.2012).

[28] Kopie auf CD-ROM (Datei online/interview_faz_lanier.pdf). URL: http://www.faz.net/aktuell/feuilleton/debatten/netzkultur-der-digitale-maoismus-ist-zu-ende-1913101.html (besucht am 20.10.2012).

[29] URL: http://www.ted.com/talks/yochai_benkler_on_the_new_open_source_economics.html (besucht am 26.10.2012).

[30] URL: http://www.elektrischer-reporter.de/index.php/site/film/43/ (besucht am 24.10.2012).

[31] Kopie auf CD-ROM (Datei online/groessen_konstanz.pdf). URL: http://de.wikipedia.org/wiki/Gr%C3%B6%C3%9Fenkonstanz (besucht am 12.10.2012).

[32] Kopie auf CD-ROM (Datei online/sander_illusion.pdf). URL: http://en.wikipedia.org/wiki/Sander_illusion (besucht am 14.10.2012).

[33] Kopie auf CD-ROM (Datei online/kratylos.pdf). URL: http://de.wikipedia.org/wiki/Kratylos_%28Philosoph%29 (besucht am 11.10.2012).

[34] Kopie auf CD-ROM (Datei online/self_affirmation.pdf). URL: http://en.wikipedia.org/wiki/Self-affirmation (besucht am 12.10.2012).

[35] Kopie auf CD-ROM (Datei online/gruppendenken.pdf). URL: http://de.wikipedia.org/wiki/Gruppendenken (besucht am 11.10.2012).

[36] URL: http://www.youtube.com/watch?v=9X68dm92HVI&feature=related (besucht am 11.10.2012).

[37] Kopie auf CD-ROM (Datei online/narrow_framing.pdf). URL: http://rouvelle.com/?p=142 (besucht am 11.10.2012).

[38] URL: http://www.youtube.com/watch?v=oQVAy_wR1d0 (besucht am 12.10.2012).

[39] Kopie auf CD-ROM (Datei online/swarm_intelligence.pdf). URL: http://en.wikipedia.org/wiki/Swarm_Intelligence (besucht am 15.10.2012).

[40] Kopie auf CD-ROM (Datei online/advocatus_diaboli.pdf). URL: http://www.bsfrey.ch/articles/E_111_02.pdf (besucht am 12.10.2012).

[41] Kopie auf CD-ROM (Datei online/trolle.pdf). URL: http://www.zeit.de/digital/internet/2012-06/trolle-internet (besucht am 11.10.2012).

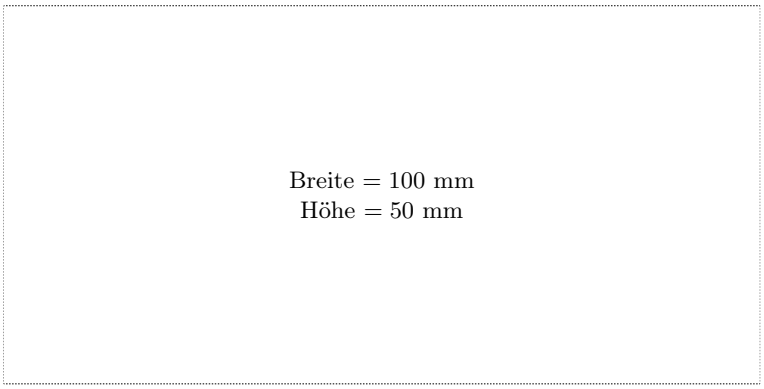
[42] Kopie auf CD-ROM (Datei online/social_over_iq.pdf). URL: http://blogs.hbr.org/cs/2012/10/collective_intelligence_and_th.html (besucht am 15.10.2012).

[43] Kopie auf CD-ROM (Datei online/female_smarter.pdf). URL: http://hbr.org/2011/06/defend-your-research-what-makes-a-team-smarter-more-women/ar/1 (besucht am 15.10.2012).

[44] Kopie auf CD-ROM (Datei online/likert_skala.pdf). URL: http://de.wikipedia.org/wiki/Likert-Skala (besucht am 17.10.2012).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —