Freie Universität Berlin

Diplomarbeit

Entwurf eines verteilten Systems für die Klassifikation multivariater Zeitreihen im psychophysiologischen Kontext

Florian Schilling
3818846

Fachbereich für Mathematik und Informatik
Freie Universität Berlin

29. Februar 2012

Gutachter

Prof. Dr. Raúl Rojas
Fachbereich für Mathematik und Informatik
Freie Universität Berlin

Prof. Dr. Timo von Oertzen
Fachbereich für Psychologie
University of Virginia

In Zusammenarbeit mit dem

Max-Planck-Institut für Bildungsforschung
Max Planck Institute for Human Development

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit mit dem Titel „Entwurf eines verteilten Systems für die Klassifikation multivariater Zeitreihen im psychophysiologischen Kontext“ von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 29. Februar 2012

Florian Schilling

Zusammenfassung

In dieser Arbeit wird eine Brücke zwischen der Informatik und der Psychophysiologie geschlagen, indem die Grundlagen für ein System geschaffen werden, welches den Austausch von Daten und Methoden zwischen den beiden Gebieten unterstützt. Den Problemstellungen der psychologischen Forschung, die sich üblicherweise herkömmlicher, statistischen Verfahren als Methoden bedient, stehen Lösungsansätze aus dem Gebiet des maschinellen Lernens gegenüber. In einem einheitlichen Entwurf wird zunächst betrachtet, wie sich die beiden Themengebiete verbinden lassen und an welchen Stellen eine Zusammenarbeit hilfreich ist. Daraus ergibt sich eine Betrachtungsweise für die Analyse multivariater Zeitreihen im psychophysilogischen Kontext, aus der sich ein Funktionsumfang für ein übergreifendes System ableiten lässt, das von einer Anwendung umgesetzt werden kann.
In dieser Arbeit wird ein System entworfen, welches sowohl bei Entwicklungen im Bereich der Mustererkennung als auch bei der Beantwortung psychologischer Fragestellungen Unterstützung bietet. Um den Nutzen dieses Systems beurteilen zu können, wird am Ende dieser Arbeit eine Datenerhebung mit Unterstützung des Systems analysiert und die Ergebnisse dieser Auswertung vorgestellt. Dabei werden die Vorteile einer gegenseitigen Zusammenarbeit deutlich.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die Psychophysiologie ist ein Teilgebiet der Biopsychologie, in dem die Wechselwirkungen von psychischen Vorgängen und Körperfunktionen erforscht werden. Dabei werden unter anderem Themengebiete wie Emotionen, Schlaf oder Stress untersucht. Es wird davon ausgegangen, dass einer psychischen Aktivierung eine physische Körperreaktion folgt. Das heißt beispielsweise, dass eine Emotion, wie Angst oder Aufregung, eine physische Reaktion wie zittern oder schwitzen auslöst. Hierbei kann es sich aber auch viele andere Reaktionen wie eine Erhöhung Blutdrucks oder eine veränderte Leitfähigkeit der Haut handeln. Das Erkennen psychischer Zustände durch physiologische Gegebenheiten ist ein Gegenstand dieses Forschungsgebietes. Dabei ist ersichtlich, dass ein Rückschluss auf einen psychischen Zustand nicht durch eine einzelne, sondern nur durch das gleichzeitige Betrachten mehrerer physischer Eigenschaften geschlossen werden kann.
Durch Verfahren wie die *Elektrokardiographie* (EKG), *die Elektroenzephalographie* (EEG) oder die *Accelerometrie* lassen sich die Herzrate, die Aktivität des Gehirns, die Körperhaltung und viele andere Zustände erfassen. Über einen Zeitraum gemessen, stellen solche physischen Messgrößen *Zeitreihen* dar, beim gleichzeitigen Erfassen mehrerer Messgrößen *multivariate Zeitreihen.*
In der Forschung werden vermutete Wechselwirkungen physiologischer und psychischer Zuständen zunächst als Hypothesen formuliert die dann anhand empirischer Daten validiert werden können. Im Rahmen von Labor- und Feldstudien werden Daten erfasst, um die Aussagen der Hypothesen zu überprüfen, um diese somit bestätigen oder widerlegen zu können. Zum Formulieren dieser Hypothesen werden Kategorien von physiologischen Zuständen zugrunde gelegt, die für eine Beweisführung wichtig sind. Wird beispielsweise die These aufgestellt, dass sich ältere Menschen im Gegensatz zu jüngeren in bestimmten Situationen häufiger hinsetzen, werden neben *alt* und *jung* auch die Kategorien *sitzen* und *nicht sitzen* festgelegt.
Solche Kategorien werden in Studien zusammen mit Aufzeichnungen bestimmter physischer

Messgrößen erfasst. Die Messgrößen, die eine Trennung der Kategorien erlauben, zu bestimmen, stellt bereits ein Problem dar. So erscheint es in dem gegebenen Beispiel zwar sinnvoller die Beschleunigungen der Probanden aufzuzeichnen, als deren Herzfrequenz, aber ob diese Sensorkonfiguration genügt, um anhand der Daten unterscheiden zu können, ob eine Person sitzt oder steht, kann ohne vorheriges Testen nicht sichergestellt werden.
Doch auch in den Studien selbst sind diese Kategorien von Bedeutung: Ist beispielsweise in einer Laborstudie eine Sensorik gefunden worden, welche die gewünschten Rückschlüsse zulässt, wird die Hypothese in Feldstudien geprüft. In diesen Studien werden in der Regel nur noch die Daten, jedoch nicht die Kategorien erfasst und müssen im Rahmen einer Datenauswertung zugewiesen werden. Dies geschieht meist in einem aufwendigen Prozess, in dem die Daten manuell ausgewertet werden.
Dieser Prozess der Kategorisierung der Daten kann durch den Computer vereinfacht werden. Das Teilgebiet des maschinellen Lernens der Informatik umfasst den Bereich der Mustererkennung. Dieses beschäftigt sich mit der Klassifizierung von Daten anhand bestimmter Merkmale, deren Ausprägungen als Muster vom Computer genutzt werden um eine abstrakte Beschreibung für Klassen von Daten zu finden. Diese Beschreibungen der Klassen, werden dann genutzt, um der Daten entsprechenden Kategorien zuzuweisen.
Die Häufigkeit von Auf- und Abbewegungen, die beim Gehen aufgezeichnet werden, kann beispielsweise als ein Merkmal zur Beschreibung von Klassen wie *Gehen* und *Stehen* genutzt werden. Dem Computer würden dann, im Rahmen eines überwachten Schulungsprozesses, mehrere Werte dieses Merkmals zusammen mit den entsprechenden Klassen vorgeführt werden, wodurch dieser beispielsweise einen Schwellwert erlernt, der eine Abgrenzung zwischen diesen Klassen bildet. Ist das Training beendet, kann der Computer für eine neue Beobachtung des Merkmals vorhersagen, welcher Klasse diese angehört.

1.1 Motivation

Die Verfahren die für die Klassifikation von Daten genutzt werden bereits seit langem erforscht. Die S*upport* V*ector* M*achine* (SVM) und der *k*-N*earest*-N*eighbors-Algorithmus* (KNN), *Neuronale Netze* (NN) und *Entscheidungsbäume (Decission Trees)* stellen hier nur einige der grundlegenden Methoden dar. Neben diesen Verfahren existieren bereits Anwendungen die eine Vielzahl dieser Methoden nutzen, um die vielfältigsten Daten zu klassifizieren und zu analysieren. Jedoch sind die einzelnen Anwendungen meist auf ein Feld von Datenerhebungen spezialisiert und bieten somit nicht die Möglichkeit verschiedene Aufzeichnungsklassen mit einander zu verbinden. Ist eine Anwendung beispielsweise dafür geschaffen, Daten der *Magnet Resonanz Tomographie* (MRT) zu klassifizieren, ist sie nicht in der Lage zusammen mit diesen Daten auch Aufzeichnungen eines EKGs zu verarbeiten.
Systeme, welche in der Lage sind die Klassifizierung von Daten auf mehrere Teilsystem zu verteilen und somit den Gesamtprozess zu parallelisieren und zu beschleunigen stellen auch keine Neuerung dieses Themengebietes dar. Doch stellen diese Systeme in der Regel abgeschlossene Einheiten dar, die sich nicht ohne Weiteres in ihrer Funktionalität auf eine neue Problemstellung anpassen oder gar erweitern lassen. Für den Fall, dass sich ein System dennoch erweitern lässt, ist dies meist nur in der Programmiersprache möglich, in der das System

entworfen wurde. Das kann zu Folge haben, dass sich ein Entwickler, der sich diese Systems bedienen möchte, zuvor seine gewohnte Entwicklungsumgebung verlassen muss und sich einer neuen Sprache bemächtigen muss.
Betrachtet man die Daten, die in den Studien erhoben werden, stehen die meisten nur der erhebenden Gruppe selbst zur Verfügung und nicht unmittelbar auch anderen Gruppen. Die liegt meist nicht an datenschutzrechtlicher Gründen oder einem des Interessen der anderen Gruppierungen, sondern vielmehr an der zugrunde liegenden Infrastruktur, die es nur erlaubt die Daten selbst und keine Beschreibung der der Daten abzulegen. Durch das Fehlen einer solchen Beschreibungsstruktur, ist es ohne gezielte Informationsbeschaffung nicht möglich zu erfahren in welchem Kontext und mit welchen Zielen eine Datenerhebung stattgefunden hat. Somit die Daten im Vorfeld meist nicht zu verwerten.

1.2 Ziele

Das Hauptziel dieser Arbeit ist der Entwurf einer formale Beschreibung Muster erkennender Prozesse. Diese Formalisierung der Prozesse soll die Grundlage für das Verbinden verschiedener Anwendungen auf dem Gebiet der Mustererkennung bilden. Damit soll die bestehende Lücke zwischen verschiedenen Klassifizierungssystemen geschlossen werden, indem die Möglichkeit gegeben wird, diese in einem gemeinsamen System nutzen zu können.
Diese Forderung nach Interoperabilität bring nicht nur eine einfache Einbettung der System in verschieden Programmiersprachen mit sich, sondern auch eine Kommunikation mit Nutzern, die dass System lediglich für die Analysen von Daten nutzen wollen. Daher muss sichergestellt sein, dass Verfahren, die das System zur Verfügung stellt, auch Nutzern zugänglich ist, die mit den internen Strukturen zum bearbeiten der Auswertungen nicht vertraut sind. Das bedeutet, dass alle Methoden, die zu Verfügung gestellt werden so beschrieben werden, dass ihre Bedeutung leicht ersichtlich ist.
Ist die Beschreibung für die Kommunikation mit der System einmal in eine konkrete Programmiersprache aufgenommen wurden, soll nicht mehr als nötig in die laufenden Arbeitsprozesse eingreifen, sondern soll diese bei bedarf erweitern können. Zudem sollen die Daten, die von einer Implementierung des Systems erfasst werden, für jeden, dem ein Zugriff auf das System gestattet wird, ausgetauscht werden können. Damit dies problemlos möglich ist, soll auch an dieser Stelle Raum zum beschreiben dieser Daten gegeben werden.
Bei dem Entwurf des System muss darauf geachtet werden, dass dieser Entwurf laufend erweitert werden können muss, damit alle Beteiligte unmittelbar von neuen Entwicklungen profitieren können.

Damit in dieser Arbeit ein allgemein gültiger Entwurf für die Problemstellung der Mustererkennung gefunden werden kann, wird dieser Prozess zunächst formal erfasst und die grundlegenden Methoden vorgestellt. Danach werden die Zielstellungen analysiert und eine Architektur entworfen, die diesen Anforderungen genügt. Diesem Entwurf folgt die Beschreibung der Kommunikationsspezifikation, welche das Ergebnis diese Arbeit bildet. Abschließend wird ein Prototyp des vorgestellten Systems, anhand einer Feldstudie getestet und die Ergebnisse präsentiert.

KAPITEL 2

Grundlagen der Mustererkennung

In der Informatik beschreibt die Mustererkennung Verfahren, mit denen ein Computer Daten Kategorien zuordnet. Bei diesen Daten kann es sich beispielsweise um Texte, Bilder, Sprache oder Messdaten handeln. Die Kategorien, die diesen Daten zugeordnet werden sollen, können dann Themen, Worte oder Objektklassen, wie Gebäude, Gesichter oder allgemein Eigenschaften der Datenquellen umfassen. Der Prozess der Mustererkennung wird in sechs Teilschritte gegliedert (vgl. [7, S. 9ff]):

1. die Datenerhebung
2. eine Vorverarbeitung, in der die Daten aus dem ersten Schritt gegebenenfalls bereinigt werden (Normieren, skalieren, segmentieren, etc.)
3. das Festlegen von Merkmalen, anhand derer die Kategorisierung vorgenommen wird
4. die Wahl eines Modells zur Kategorisierung und dessen Konfigurierung
5. das Trainieren dieses Modells
6. die Bewertung eines Verfahrens

Die Beurteilung eines Verfahrens kann ergeben, dass die Ergebnisse nicht zufriedenstellend sind. Dies wird erkannt, wenn das gewählte Verfahren auf neuen Testdaten zu oft falsche Kategorien ermittelt. In diesem Fall können die Einstellungen jedes Teilschritts (2–5) vor der Bewertung verändert werden und das Modell neu evaluiert werden. Diese Iteration wird solange fortgesetzt, bis ein Modell gefunden wird, welches diese Daten bestmöglich beschreibt. In Abbildung 2.1 wird dieses Vorgehen schematisch dargestellt.

In den folgenden Abschnitten werden die einzelnen Teilschritte dieses Schemas betrachtet und formal definiert. Damit wird ein Überblick über die Problemstellung der Mustererkennung gegeben, der die Grundlage für ein verteiltes Klassifizierungssystem als Lösung der Problemstellung bietet bietet.

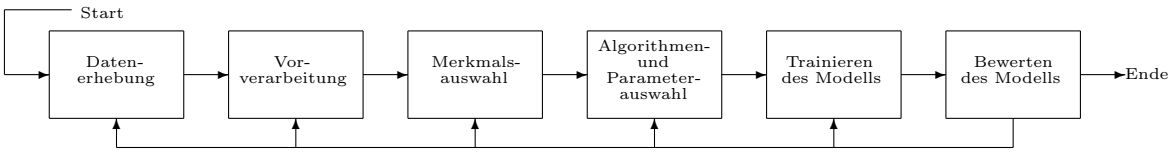


Abbildung 2.1: Allgemeines Schema Muster erkennender Systeme.

2.1 Datenerhebung

Im Rahmen dieser Arbeit bedeutet Datenerhebung, dass physiologische Daten von Testpersonen gemessen werden. Diese Messung umfassen beispielsweise:

- die Herzrate, eines Probanden die durch ein Elektrokardiographie (EKG) erfasst werden
- die Oberflächenspannung der Haut, Elektroenzephalographie (EEG)
- Beschleunigungsdaten oder Positionsdaten (Accelerometrie und Aktometrie)

Solche Werte werden über die Zeit erfasst, wodurch sich Zeitreihen der Messgrößen ergeben. Eine Erhebung solcher Zeitreihen wird als Datensatz bezeichnet. Die Zeitreihe einer Messgröße bildet einen Kanal des Datensatzes. Besteht ein solcher Datensatz aus mehreren Kanälen, kann dieser auch als ein *multivariater* Kanal angesehen werden.

Damit der Funktionsumfang des System formal beschrieben werden kann, wird zunächst eine Notation festgelegt. Zur Vereinfachung der Formalisierung wird davon ausgegangen, dass alle Kanäle zu den selben Zeitpunkten auf genommen werden und die zeitlichen Abstände zwischen diesen Zeitpunkten gleich sind. Dadurch lassen sich die Zeitpunkte der Messungen als Index $t \in \mathbb{N}$ betrachten und eine Messreihe von m Kanälen c_i über einen Zeitraum von d Zeitpunkten kann als

$$D_m^d = \begin{bmatrix} c_1(0), & \dots, & c_1(d-1) \\ \vdots & \ddots & \vdots \\ c_m(0), & \dots, & c_m(d-1) \end{bmatrix} = \begin{bmatrix} c_1^d \\ \vdots \\ c_m^d \end{bmatrix} \in \mathbb{R}^{m \times d} \tag{2.1}$$

geschrieben werden. Ein Kanal c_i der sich aus d Abtastwerten zusammensetzt wird als

$$c_i^d = \begin{bmatrix} c_i(0), & \dots, & c_i(d-1) \end{bmatrix} \text{ mit } c_i(x) \in \mathbb{R}, \forall x = 1, \dots, d \tag{2.2}$$

notiert. Dem entsprechend kann ein Segment des Kanals das d Zeitpunkte, beginnend bei Zeitpunkt t, umfasst durch

$$c_i^d(t) = \begin{bmatrix} c_i(t), & \dots, & c_i(t+d-1) \end{bmatrix} \tag{2.3}$$

beschrieben werden. Dies gilt analog auch für Datensätze

$$D_m^d(t) = \begin{bmatrix} c_1(t), & \dots, & c_1(t+d-1) \\ \vdots & \ddots & \vdots \\ c_m(t), & \dots, & c_m(t+d-1) \end{bmatrix} = \begin{bmatrix} c_1^d(t) \\ \vdots \\ c_m^d(t) \end{bmatrix} \tag{2.4}$$

Eine Spalte dieser Matrix entspricht somit einer *Stichprobe*, also einer Beobachtung aller Parameter zu einem Zeitpunkt, und die Veränderung eines Parameters über die Zeit entspricht einer Zeile der Matrix.

Neben dem Aufzeichnen der Messungen können auch Ereignisse erfasst werden, die während der Erhebung auftreten. Diese Ereignisse werden zusammen mit den Daten erfasst. Sie bilden Indikatoren für Kategorien die im weiteren Verlauf benötigt werden.

2.2 Vorverarbeitung

Bevor die Daten der Datenerhebung für eine Klassifizierung genutzt werden können, müssen sie gegebenenfalls vorverarbeitet werden. Dies bedeutet, dass

- unwichtige oder fehlerhafte Teile der Daten entfernt werden
- mögliches Rauschen reduziert wird
- die Daten normiert und skaliert werden

Diese Vorverarbeitungsschritte werden im Folgenden als Funktion $T : \mathbb{R}^{m \times n} \to \mathbb{R}^{m' \times n'}$ betrachtet. Diese Vorverarbeitungsfunktionen werden innerhalb dieser Arbeit als *Filter* bezeichnet. Da sie kombiniert werden können wird von einem einzigen Vorverarbeitungsschritt ausgegangen. Das bedeutet, dass sich eine Kette von Filtern $T_i : \mathbb{R}^{m_{i-1} \times n_{i-1}} \to \mathbb{R}^{m_i \times n_i}$ mit $i = 1 \dots k$ und einem $k \in \mathbb{N}^+$ zu einem Filter zusammenfassen lassen:

$$T : \mathbb{R}^{m_0 \times n_0} \to \mathbb{R}^{m_k \times n_k}$$

$$T = T_k \circ T_{k-1} \circ \dots \circ T_1$$

Die gefilterten Daten $X_m^d = T(D_{m'}^{d'})$ werden als die tatsächlichen Beobachtungen angesehen. Sie bilden die Basis für die weitere Verarbeitung.

2.2.1 Segmentierung

Die in einer Datenerhebung erfassten Daten werden in Datensätzen gespeichert. Diese enthalten neben den Daten auch Zusatzinformationen über die Aufzeichnungsbedingungen. Im Zusammenhang mit Testdaten stellen diese Information einen Zusammenhang zu den Kategorien der Aufzeichnungen her. Dabei führen Veränderungen der Aufnahmebedingungen, insbesondere Änderungen der vorgegebenen Kategorien, zu einer logischen Segmentierung der Daten. Diese und andere Segmentierungen werden im folgenden näher betrachtet.

Klassenzuweisungen

Daten, die unter der Bedingung einer vorgegebenen Kategorie aufgenommen wurden, bilden ein Datensegment. Datensegmente einer Kategorie bilden eine Klasse Daten dieser Kategorie. Einer der wichtigsten Vorverarbeitungsschritte ist das Zuweisen von Kategorien in Trainingsdaten.

Die erhobenen Daten bestehen nach der Aufzeichnung meistens aus einem Datensatz, in dem alle zu erfassenden Kategorien enthalten sind. Damit jedoch die Daten der Kategorien getrennt voneinander bearbeitet werden können, muss ein solcher Datensatz an den Stellen getrennt werden, an denen bei der Aufzeichnung eine vorgegebene Kategorie geändert wurde. Für diese Trennung sind zwei Verfahren vorgesehen:

- mit den Daten werden Zeitpunkte von Ereignissen erfasst, die durch eine Bezeichnung auf die vorgegebene Kategorie hinweisen
- die Daten werden bereits während der Datenerhebung in verschieden Datensätzen abgelegt und eine Bezeichnung wie beispielsweise ein Dateiname geben Aufschluss über die Zugehörigkeit zu einer Kategorie

In beiden Verfahren können die Bezeichnungen weitgehend frei gewählt werden. Eine Einschränkung ist jedoch dadurch gegeben, dass diese Bezeichnungen maschinell ausgewertet werden sollen. Um eine freie Gestaltung dieser Bezeichnungen weitgehend beibehalten zu können, wird die Klassenzugehörigkeit durch *reguläre Ausdrücke* [10, S. 93ff] bestimmt, die Angaben erzwingen, die für die spätere Nutzung unbedingt erforderlich sind.
Ein regulärer Ausdruck ist eine abstrakte Beschreibung, die eine Menge von Zeichenketten bzw. Bezeichnungen durch eine Abfolge syntaktischer Regeln beschreibt. Einfache Bespiele hierfür sind Regeln wie: *Eine Bezeichnung beginnt mit „Steht“* oder *Eine Bezeichnung enthält den Text „Gehen“.*

Damit den Datensätzen durch die regulären Ausdrücke Klassen $\mathcal{L} = \{l_1, \ldots, l_k\}$ zugewiesen werden können, werden für die beiden vorgestellten Verfahren drei solcher Ausdrücke eingesetzt:

- α_i und β_i für eine Unterteilung eines Datensatzes
- γ_i um einem gesamten Datensatz eine Klasse l_i zuzuweisen

Im ersten Verfahren wird in einem Datensatz D_m^d eine Markierung gesucht, die sich mit α_i beschreiben lässt. Der Zeitpunkt der mit dieser Markierung abgelegt wurde wird dann als Startzeitpunkt t_0 eines Segments festgelegt. Von diesem Zeitpunkt ausgehend wird eine weitere Markierung gesucht die sich durch β_i beschreiben lässt. Der entsprechende Zeitpunkt $t_n > t_0$ bezeichnet das Ende eines Segments und den Daten $D_m^{t_n - t_0}(t_0)$ wird die Klasse l_i zugeordnet. Dieses Vorgehen wird für alle Markierungen wiederholt. Das zweite Verfahren benutzt den Ausdruck γ_i um dem gesamten Datensatz $D_m^d(0)$ die Klasse l_i zuzuweisen, wenn sich die Bezeichnung des Datensatzes durch γ_i beschreiben lässt. Das Resultat dieser

Klassenzuweisungen ist eine Menge Tupeln, die sich aus einem Datensegment und einer Menge von Klassen zusammensetzen.

Klassenhierarchien Eine Kategorie kann eine Generalisierung mehrerer anderer Klassen sein. Beispielsweise können die Klassen *Liegen links*, *Liegen rechts*, *Liegen Bauch* und *Liegen Rücken* unter einer Klasse *Liegen* zusammengefasst werden. Um dies zu formalisieren gibt eine Funktion $p : \mathcal{L} \to \mathcal{L}$ die Oberklasse einer Klasse l_i an. Kann einer Klasse keine übergeordnete Klasse zugewiesen werden, gilt $p(l_i) = l_i$. Dieses hierarchischen Klassen bieten einem Anwender die Möglichkeit Datensätze wahlweise nach Ober- oder Unterklassen zu segmentieren.

2.2.2 Zeitblöcke

Datensätze oder Segmente können auch ohne Markierungen in Segmente unterteilt werden. Da es sich bei den Daten um Zeitreihen handelt entspricht eine Unterteilung einer Teilung in Zeitfenster. Eine Aufteilung eines Segments in solche Fenster kann dafür genutzt werden die Menge der Datensätze zu vergrößern.
Angenommen ein Segment enthält Daten die einer Klasse *Gehen* zugeordnet sind. Dann liefern die Daten dieses Segments ein Muster dieser Kategorie. Wird dieses Segment nun in mehrere Teile aufgespalten geben diese Teilsegment immer noch Daten die im Kontext *Gehen* aufgezeichnet wurden. Die Größe dieser Teilsegment kann jedoch nicht beliebig klein gewählt werden. Bei zu kleinen Fenstern können Charakteristika der Daten verloren gehen, die eine Zuordnung zu einer Klasse ermöglichen. Ein anschauliches Beispiel bieten EKG-Daten in denen die Ausschläge die den Kontraktionen des Herzmuskels vorausgehen das typische Bild eines EKGs ausmachen. Wird jedoch nur ein Ausschnitt diese Diagramms betrachtet in dem kein Ausschlag enthalten ist, lässt sich nicht erkennen, woher die Daten stammen. Somit ist eine optimale Fenstergröße von der Art der Aufgezeichneten Daten abhängig und sollte im Vorfeld bestimmt werden. Für eine Formalisierung dieser Segmentierung sei s die Abtastrate der Kanäle c_i eines Datensatzes D_m^d und $v \in \mathbb{R}$ eine Intervallbreite. Damit ergeben $k = \lfloor \frac{dv}{s} \rfloor$ die Größe, $n = \lfloor \frac{d}{k} \rfloor$ die Anzahl und $p = d - nk$ die Restgröße von Fenstern eines Segments. Dieses lässt sich somit in n Segmente der Größe k und ein Segment der Länge p unterteilen.

$$D_m^d = \begin{bmatrix} D_m^k(0k) & D_m^k(1k) & \dots & D_m^k((n-1)k) & D_m^p(nk) \end{bmatrix} \tag{2.5}$$

2.2.3 Kanal-Skalierung

Einige Verfahren setzen voraus, dass die Kanäle eines Datensatzes untereinander vergleichbar sind. Eine Voraussetzung hierfür ist, dass die Werte der Kanäle im gleichen Wertebereich liegen. Typischerweise handelt es sich hierbei um das Einheitsintervall $[0, 1]$. Im Allgemeinen wird ein Kanal c_i^d auf das Intervall $[0, s]$ skaliert, indem der größte c_i^+ und der kleinste Wert c_i^- des Kanals ermittelt werden

$$c_i^+ = \max_{x \in [0,\dots,d-1]} c_i(x) \text{ und } c_i^- = \min_{x \in [0,\dots,d-1]} c_i(x) \tag{2.6}$$

und die einzelnen Werte des Kanals durch

$$c_i'(t) = \begin{cases} s\frac{c_i(t)-c_i^-}{c_i^+-c_i^-} & \text{, wenn } c_i^+ \neq c_i^- \\ 0 & \text{, sonst} \end{cases} \tag{2.7}$$

skaliert werden. Der skalierte Kanal $\acute{c}_i^d$ setzt sich dann aus diesen Werten im Intervall $[0, s]$ zusammen.

$$\acute{c}_i^d = \left[c_i'(0), \quad \ldots, \quad c_i'(d-1)\right] \tag{2.8}$$

2.2.4 Kanal-Normierung

Ein Problem welches die Kanal-Skalierung mit sich bringt, sind starke Ausreißer der Messwerte. Ein solcher Messfehler hat zur folge, dass die übrigen Werte auf einen kleinen Bereich um 0 skaliert werden und die Zeitreihe durch diese Fehler dominiert wird. Ein Verfahren welches solchen Ausreißern gegenüber stabiler skaliert ist die Normierung. Für dieses Verfahren werden der Mittelwert und die Standardabweichung eines Kanal bestimmt und anhand dieser die Werte des Kanals skaliert. Dadurch können die Werte eines normierten Kanals als vielfache der Standardabweichung betrachtet werden. Für einen Kanal c_i^d ist der Mittelwert $\bar{c}_i$ definiert als arithmetische Mittel seiner Werte:

$$\bar{c}_i = \frac{1}{d}\sum_{j=1}^{d} c_i(j) \tag{2.9}$$

Entsprechend gilt für ein Segment dieses Kanals beginnend am Zeitpunkt t_0:

$$\bar{c}_i^d(t_0) = \frac{1}{d}\sum_{j=0}^{d-1} c_i(t_0+j) \tag{2.10}$$

Die Standardabweichung $\hat{c}_i^d$ des Kanalausschnitts ist dann definiert durch:

$$\hat{c}_i^d(t_0) = \sqrt{\frac{1}{d-1}\sum_{j=0}^{d-1}\left[c_i(t_0+j) - \bar{c}_i^d(t_0)\right]^2} \tag{2.11}$$

Für ein Segment beginnend zu einem Zeitpunkt t_0 ergeben sich die normierten Werte $\tilde{c}_i(t)$ durch

$$\tilde{c}_i(t) = \frac{c_i(t_0+t) - \bar{c}_i^d(t_0)}{\hat{c}_i^d(t_0)} \tag{2.12}$$

Damit wird der normierte Kanal $\tilde{c}_i^d$ wir folgt berechnet:

$$\tilde{c}_i^d = \left[\tilde{c}_i(0), \quad \ldots, \quad \tilde{c}_i(d-1)\right] \tag{2.13}$$

2.2.5 Gleitender Mittelwert

Die gleitenden Mittelwerte (engl. moving avarage) einer Zeitreihe dienen der Glättung der Daten. Diese Mittelwerte k-ter Ordnung bilden k aufeinanderfolgender Werte eines Kanals c_i auf ihr arithmetisches Mittel ab. Die Funktion $\mathrm{sma}_k : \mathbb{R}^d \to \mathbb{R}^{d-k+1}$, welche den Kanal glättet, ist definiert durch:

$$\mathrm{sma}_k(c_i) = \left[\bar{c}_i^k(0), \quad \ldots, \quad \bar{c}_i^k(d-k-1)\right] \tag{2.14}$$

Ferner kann die Funktion auch durch einen Gewichtsvektor $\omega \in \mathbb{R}^k$ modifiziert werden. Hierfür wird die Gleichung 2.9 wie folgt erweitert:

$$\mathrm{avg}_\omega^d(c_i, t) = \frac{1}{|\omega|} \sum_{j=0}^{d-1} \omega_j c_i(t+j) \tag{2.15}$$

Damit ergibt sich der gewichteten, gleitenden Mittelwert wma_k^ω in Abhängigkeit von dem Gewichtsvektor ω:

$$\mathrm{wma}_\omega^k(c_i, t) = \left[\mathrm{avg}_\omega^k(c_i, t), \quad \ldots, \quad \mathrm{avg}_\omega^k(c_i, t+d-k)\right] \tag{2.16}$$

2.3 Merkmalsraum

Nach der Vorverarbeitungsphase werden die Daten in den Merkmalsraum übertragen. Das bedeutet, dass ein physisches Muster in eine Darstellung überführt wird, in der es nur noch durch die Ausprägungen bestimmter Eigenschaften beschrieben wird und die zeitliche Komponente der Daten nicht mehr berücksichtigt wird. Diese Eigenschaften beschreiben Punkte Merkmalsraum $\mathcal{F} \subseteq \mathbb{R}^a$. Für diesen Schritt werden Merkmale definiert, anhand welcher die Muster erkannt werden sollen. Ein solches Merkmal wird als eine Funktion $f : \mathbb{R}^{m \times d} \to \mathbb{R}^a$ angesehen, die einem Datensegment eine Menge von a Merkmalsausprägungen $(f_{i,1}, f_{i,2}, \ldots f_{i,a_i})$ zuweist. Wie auch die Filterfunktionen kann eine Menge Merkmalsfunktionen $f_i : \mathbb{R}^{m \times n} \to \mathbb{R}^{a_i}$ mit $i = 1 \ldots k$ und $k \in \mathbb{N}^+$ als einzelne Funktion $f : \mathbb{R}^{m \times n} \to \mathbb{R}^j$ mit $j = \sum_1^k a_i$ betrachtet werden.

$$f(X_m^n) = (f_{1,1}, \ldots f_{1,a_1}, f_{2,1}, \ldots f_{k-1,a_{k-1}}, f_{k,1}, \ldots f_{k,a_k}) \in \mathbb{R}^j$$

Der so berechnete Vektor entspricht den Charakteristika der einzelnen Merkmale, die von einem Datensegment X extrahiert wurden.

2.3.1 Grundlegende Merkmale

Die ersten beiden Merkmale ergeben sich aus dem Vorverarbeitungsschritt der Kanalnormierung. Die Werte $\bar{c}_i^d$ und $\hat{c}_i^d$, die im Abschnitt 2.2.4 eingeführt wurden, geben charakteristische Werte der Kanäle an, die für eine Klassifizierung der Daten genutzt werden können.

Mittelwert

Die Mittelwerte aus Gleichung 2.9 bieten bereits Merkmale. Um diese von einem Datensatz X_m^d mit den Kanälen c_i zu erheben und in dem Mermalsraum zu überführen, wird die Funktion mean : $\mathbb{R}^{m\times d} \to \mathbb{R}^m$ definiert:

$$\text{mean}(D_m^d) = \left[\bar{c}_1, \ldots, \bar{c}_m\right]^T \tag{2.17}$$

Standardabweichung

Aus Gleichung 2.11 ergibt sich das Merkmal emphStandardabweichung. Dieses kann von den Kanälen c_i eines Datensatz D_m^d durch die Funktion stddev : $\mathbb{R}^{m\times d} \to \mathbb{R}^m$ erhoben werden:

$$\text{stddev}(D_m^d) = \left[\hat{c}_1, \ldots, \hat{c}_m\right]^T \tag{2.18}$$

Energie

Zur Bestimmung eines Energiewertes wird die Fläche die interpolierten Messkurve betrachtet. Die quadrierte Messwerte werden aufsummiert und auf einer logarithmischen Skala gemessen. Die Energie E eines Kanals c_i^d ist somit durch

$$E(c_i^d) = log \sum_{j=0}^{d-1} c_i(j)^2 \tag{2.19}$$

Der Merkmalsvektor eines Datensatzes D_m^d wird somit durch

$$\text{energy}(D_m^d) = \left[E(c_1^d), \ldots, E(c_m^d)\right]^T \tag{2.20}$$

bestimmt.

2.3.2 Kanalkorrelation

Werden die Kanäle eines Datensatzes als Zufallsvariablen betrachtet, können diese von einander abhängig sein. Die Korrelation ist ein Maß eine lineare Abhängigkeit und beschreibt die Stärke dieser Abhängigkeit. Absolute Korrelationswerte die Werte gegen 1 annehmen beschreiben einen starken linearen Zusammenhang zwischen zwei Kanälen. Werte um 0 dagegen sagen aus, dass die Variablen fast keinen linearen Zusammenhang aufweisen. In der Mustererkennung kann dieser Wert in verschieden Klassen unterschiedlich stark ausgeprägt sein und somit als Indikator für die Trennung von Klassen genutzt werden. Der Korrelationskoeffizient r für zwei Kanäle c_i^d und c_j^d gleicher Länge ist definiert durch

$$r(c_i^d, c_j^d) = \frac{\sum_{t=0}^{d-1} (c_i(t) - \bar{c}_i^d)(c_j(t) - \bar{c}_j^d)}{(d-1)\hat{c}_i^d \hat{c}_j^d} \tag{2.21}$$

Für die Bildung eines Merkmalsvektors werden die Korrelationskoeffizienten aller Kanalpaare zu einem Vektor zusammengefasst.

2.3.3 Dynamische Zeitverschiebung

Mit dem Begriff *Dynamic Time Warping* [19] wird eine Funktion beschrieben, welche die Zeitachse zweier Kanäle normiert. Ihren Ursprung findet diese Funktion in der Spracherkennung. Hier wird es genutzt um Worte die mit verschiedenen Geschwindigkeiten gesprochen wurden in die gleiche zeitliche Auflösung zu transformieren. Um die Verschiebung der beiden Zeitreihen zu berechnen bedienen sich die Autoren der Methode des *dymischen Programmierens*. Hierfür wird einen Matrix $W = [w_{i,j}] \in \mathbb{R}^{m+1 \times n+1}$ mit

$$W_{i,0} = \infty \text{ für } i = 1 \dots m \tag{2.22}$$

$$W_{0,j} = \infty \text{ für } j = 1 \dots n \tag{2.23}$$

$$W_{0,0} = 0 \tag{2.24}$$

initialisiert. Der Abstand zwischen zwei Zeitreihen $a = \left[a_1, \dots, a_m\right]$ und $b = \left[b_1, \dots, b_n\right]$ wird mit einer Kostenfunktion $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ durch den Wert $w_{m,n}$ beschrieben, nachdem die übrigen Werte von W durch die rekursive Funktion [1]

$$W_{i,j} = k(a_i, b_j) + \min \begin{cases} W_{i-1,j}, \\ W_{i,j-1}, \\ W_{i-1,j-1} \end{cases} \quad \text{mit } 1 \leq i \leq m \text{ und } 1 \leq j \leq n \tag{2.25}$$

berechnet wurden. Werden bei der Berechnung von $w_{m,n}$ die Indizes i und j gespeichert, die in Gleichung 2.25 das Minimum stellen, geben diese an welche Zeitpunkte i des ersten Kanals den Zeitpunkten j des weiten Kanals entsprechen. Die Grundlage dieser Idee bilden spracherkennende Systeme die für Worte einen bestimmten Wortschatzes jeweils Repräsentanten speichern und eingehende Muster mit diesen Prototypen vergleichen.
Zwar wird in dieser Arbeit keine explizite Spracherkennung betrieben, aber die Idee das sich die Muster einer Klasse stärker ähneln als die Muster verschiedener Klassen, scheint für einen Großteil von Daten offensichtlich. Ob sich diese Hypothese jedoch für konkrete Datenerhebungen bewahrheitet, bleibt die Aufgabe der Nutzer des in dieser Arbeit vorgestellten Systems.
Das eigentliche Merkmal beschreibt lediglich die Verschiebung zweier Kanäle zueinander:

$$\mathrm{dtw}(c_i^{d_i}, c_j^{d_j}) = W_{d_i,d_j} \tag{2.26}$$

Für die Abbildung eines Datensatzes in den Merkmalsraum werden die Verschiebung aller Kanalpaare durch die Funktion $\mathrm{DTW} : \mathbb{R}^{m \times d} \to \mathbb{R}^{\frac{m(m-1)}{2}}$ berechnet:

$$\mathrm{DTW}(X_m^d) = \left[\mathrm{dtw}(c_1, c_2), \mathrm{dtw}(c_2, c_3), \dots, \mathrm{dtw}(c_{m-1}, c_m)\right] \tag{2.27}$$

2.4 Modelle

Nach der Transformation der Daten in den Merkmalsraum können verschiedene Klassifikationsverfahren auf diesen Merkmalsdaten arbeiten. Aufgabe dieser Klassifikationsalgorithmen ist es Regelmäßigkeiten in dem Merkmalsraum zu erkennen und so einem Vektor von

[1] Diese Variante stellt den einfachsten Fall der *Warping-Funktion* dar. Sakoe u. Chiba 19 beschreiben und analysieren weitere symmetrische und asymmetrische Funktionen dieser Art.

Merkmalsausprägungen eine Klasse $l \in \mathcal{L}$ aus einer endlichen Menge möglicher Klassen $\mathcal{L}$ zuzuweisen. O.B.d.A. kann $\mathcal{L} \subset \mathbb{N}$ angenommen werden.
Wie diese Algorithmen Muster in den Merkmalsausprägungen erlernen und wie diese Muster erkannt werden ist von den Algorithmen selbst abhängig. Jedoch lassen sich die Lernstrategien der Klassifizierungsalgorithmen in drei wesentliche Kategorien fassen [7, S. 16f]:

Überwachtes Lernen ist eine Strategie zum trainieren von Klassifizierungsalgorithmen bei der ein Algorithmus in einer Trainingsphase eine Menge von Merkmalsvektoren mit bereits bekannten Kategorien vorgelegt bekommt um anhand dieser Daten die Verallgemeinerung der Kategorien durch die Merkmalsausprägungen zu erlernen. Ein solcher Trainingsdatensatz X_{train} besteht somit aus einer Menge von Tupeln $(F(X_i), l_i)$ mit $i = 1 \dots k$ und $k \in \mathbb{N}^+$.

Unüberwachtes Lernen Diese Klasse von Klassifizierungsalgorithmen benötigt keine im Voraus bekannten Kategorien sondern erstellt selbstständig eine Menge von Kategorien. Mit jedem Satz von Merkmalsausprägungen für den eine Kategorie vorausgesagt werden soll entscheidet der Algorithmus, welche der von ihm erzeugten Kategorien diesem an ehesten entspricht, passt die Beschreibung der Kategorie gegebenenfalls an und gibt diese zurück.

Bestärkendes Lernen Auch für diese Strategie des Trainierens wird eine Menge von Trainingsdaten mit bekannten Kategorien benötigt. Jedoch wird bei diesem Verfahren den die einzelnen Daten der Trainingsmenge zunächst eine Kategorie durch den Algorithmus bestimmt. Die kann somit entweder die *richtige* oder die *falsche* Kategorie zuweisen. Dieses Ergebnis wird dann von dem Algorithmus genutzt um die interne Beschreibung der Kategorie zu verbessern.

Eine besondere Form des überwachten Lernens stellt das *aktive Lernen*[20] dar. Die Grundidee dieser Strategie ist es, dass bereits kategorisierte Daten schwer zu beschaffen sind und es daher notwendig ist mit möglichst wenig Trainingsdaten ein Modell bestmöglich trainieren zu können. Dies soll dadurch erreicht werden, dass der zu trainierende Algorithmus zusammen mit einem *Orakel* arbeitet, welches die Frage nach den Kategorien von Daten beantworten kann. Das Orakel wird in der Regel durch einen Menschen realisiert, der die Daten gezeigt bekommt, daraufhin eine Kategorisierung der Daten vornimmt und das Ergebnis $\mathcal{T}$ an das Modell übermittelt. Abbildung 2.2 stellt diese Zusammenarbeit schematisch dar. Die Daten die durch das Orakel Klassifiziert werden sollen, werden durch das Modell bestimmt und folgen im wesentlichen durch zwei Strategien:

- Das erste Verfahren geht von einer großen Datenbasis $\mathcal{U}$ mit unbekannten Kategorien aus. Aus diesem mitunter sehr großen Pool von Daten werden nach einer festgelegten Strategie Exemplare gezogen und an das Orakel weiter gereicht.
- In der anderen Strategie werden gezielt Elemente ausgewählt, bei denen das Modell die ungenausten Vorhersagen treffen kann. Durch die Kategorisierung des Orakels werden

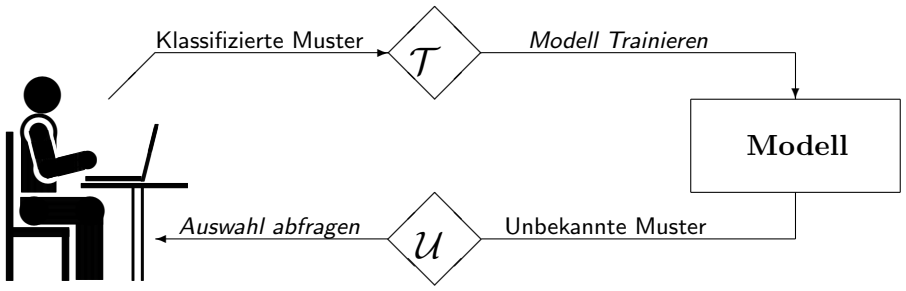


Abbildung 2.2: Grundschema des aktiven Lernens

in der nächsten Iteration andere Elemente des Pools besser durch das Modell beschrieben und der Pool von *unsicheren Kandidaten* verkleinert.

Damit die dass Modell von den erfragten Daten profitieren kann wird die Basis der Trainingsdaten um die neuen Beobachtungen erweitert und neu trainiert.

Ein Modell ist von drei Faktoren abhängig. Der erste ist die Wahl des Klassifizierungsalgorithmus $\mathcal{A}$. Da die meisten Algorithmen mit einigen Parametern auf die Daten eingestellt werden können ergibt sich als zweiter Faktor die konkreten Werte dieser Parameter $\Theta_{\mathcal{A}}$. Als letztes wird eine Menge von Merkmalen $\mathcal{F}$ festgelegt anhand derer der Algorithmus die Kategorien erkennen soll. Ein Modell $\mathcal{M}$ lässt sich somit formal durch ein Tupel $(\mathcal{A}, \Theta_{\mathcal{A}}, \mathcal{F})$ beschreiben. Im Folgenden werden die wichtigsten Klassifizierungsalgorithmen des überwachten Lernens beschrieben. Für die Beschreibung dieser Algorithmen gibt eine Funktion $f : \mathbb{R}^{m \times n} \to \mathbb{R}^a$ die Transformation der gefilterten Rohdaten in den Merkmalsraum an, deren Zielelemente $x_i = f(X_m^{d_i})$ die Merkmalsvektoren der zugrundeliegenden Daten sind. Zudem wird eine Trainingsmenge T zugrunde gelegt die sich aus n Trainingstupeln zusammensetzt:

$$T = \{(x_1, l_1), \ldots, (x_n, l_n) | l_i \in \mathcal{L}\} \tag{2.28}$$

Jedes Tupel dieser Menge entspricht einem Merkmalsvektor x_i und einer zugehörigen Klasse l_i. Eine solche Trainingsmenge bildet die Grundlage für das *überwachte Lernen* und somit für die folgenden Algorithmen.

2.4.1 Nächste Nachbarn

Der einfachste Weg eine Klassifikation von Daten vorzunehmen bietet der sogenannte k-Nearest-Neighbors. Damit die Klassenzugehörigkeit eines unbekannten Merkmalsvektors v bestimmt werden kann, betrachtet der Algorithmus die Elemente der Trainingsmenge, welche dem unbekannten Datum im Merkmalsraum am nächsten liegen. Die Anzahl der zu betrachtenden Nachbarn wird durch den Parameter k des Algorithmus festgelegt. Für die Formalisierung des Algorithmus wird eine Distanzfunktion $\delta : \mathbb{R}^a \times \mathbb{R}^a \to \mathbb{R}$ benötigt, die den Abstand zweier Merkmalsvektoren berechnet. In der Regel handelt es sich hierbei um den *euklidischen Abstand*: $\delta(x_1, x_2) = ||x_1 - x_2||_2$.

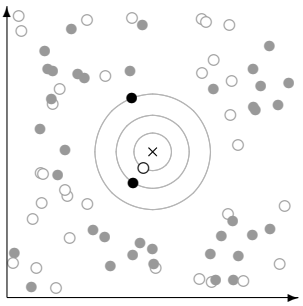


Abbildung 2.3: KNN mit $k = 3$. Gezeigt werden die 3 nächsten Nachbarn eines unbekannten Merkmalsvektors $\times$ in einer Trainingsmenge die aus den beiden Klassen $\circ$ und $\bullet$ besteht.

Die k nächsten Nachbarn N_k eines Merkmalsvektors x_i bezüglich der Trainingsmenge T sind dann definiert durch

$$N_0(x_i) = \emptyset \text{ und } N_k(x_i) = N_{k-1}(x_i) \cup \underset{(x,l)\in T\setminus N_{k-1}(x_i)}{\arg\min} \delta(x, x_i) \tag{2.29}$$

Für die Entscheidung welche Klasse $l_i \in \mathcal{L}$ dem Merkmalsvektor x_i zugewiesen wird, wird die Klasse gewählt welche in den benachbarten Elementen N_k am häufigsten vertreten ist. Sein also

$$r(l) = \{x|(x, l') \in N_k(x_i) : l' = l\} \tag{2.30}$$

die Vertreter einer Klasse l in der Nachbarschaft N_k von x_i. Dann ist die zugewiesene Klasse l_i definiert durch

$$l_i = g(x_i) = \arg\max_l \{|r(l)|\} \tag{2.31}$$

Zwar bietet diese Form der Klassifizierung im Allgemeinen sehr gute Ergebnisse, aber da dieser Algorithmus die gesamte Trainingsmenge halten muss, ist er für große Datenmengen ungeeignet. Um diesen Nachteil zu umgehen existieren zahlreiche Ideen, Die meisten dieser Ideen basieren darauf zu nächst zwei Klassen voneinander zu trennen.

2.4.2 Zwei-Klassen-Problem

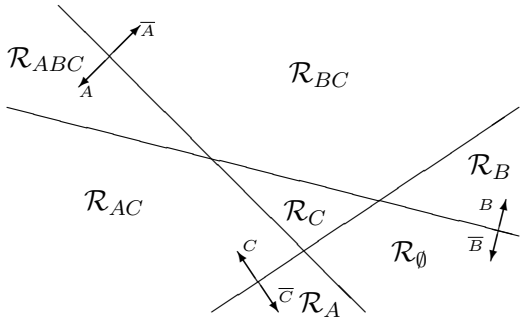
Klassifizierungsalgorithmen die nativ nur zwei Klassen, o.B.d.A.: $\mathcal{L} = \{-1, +1\}$, voneinander trennen, können durch verschiedene Methoden auf Mehr-Klassen-Probleme erweitert werden. Eine Trennung zweier Klassen bedeutet, dass der Merkmalsraum $\mathbb{R}^a$ in zwei Halbräume unterteilt wird. Eine lineare Unterteilung des Raumes kann durch eine Linearkombination der Merkmale beschrieben werden, indem ein Vektor $\omega \in \mathbb{R}^a$ die Gewichte der einzelnen Merkmale und ein Skalar b einen Schwellwert beschreibt. Vgl. Duda u. a. 7, S.216 ff und Bishop 5, S.181 ff. Geometrisch betrachtet entspricht b dem Abstand der Ebene zum Ursprung und ω der Neigung der Ebene. Für eine lineare Diskriminanzfunktion g ergibt sich die Form:

$$g(x) = \omega^T x + b \tag{2.32}$$

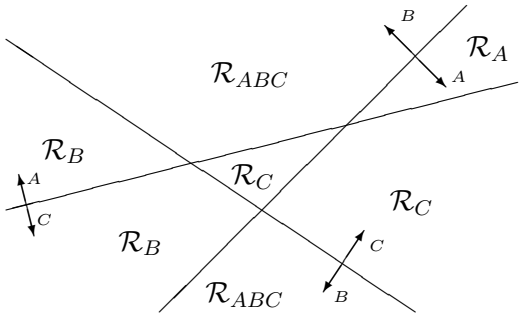
Betrachtet man nun die Lage eines Merkmalsvektors y zu der durch g beschriebenen Hyperebene, kann dieser, wenn er sich nicht auf der Trennebene selbst befindet, nur auf einer der beiden Seiten der Ebene befinden und mit Hilfe der Funktion

$$l_i = l(x_i) = sign\left(\frac{g(x_i)}{||\omega||_2}\right) \tag{2.33}$$

kann die Klassenzugehörigkeit $l_i \in \{-1, +1\}$ eines Merkmalsvektors x_i bestimmt werden. Hierbei notieren $||\omega||_2$ die L_2-Norm des Vektors ω und $sign$ die Signumfunktion. Damit diese Algorithmen für die Trennung mehrerer Klassen genutzt werden können gibt es zwei Lösungsansätze:



(a) **Eine-Gegen-Alle**: Die drei Trennebenen trennen jeweils eine Klasse x gegen alle anderen $\bar{x}$, so dass vier Klassen eindeutig erkannt werden können



(b) **Paarweise**: Die drei Trennebenen trennen jeweils zwei Klassen von einander, so dass eine Mehrheitsentscheidung eine Klassifizierung zu lässt.

Abbildung 2.4: Gezeigt werden die Trennebenen der Klassen A, B und C. Die Regionen $\mathcal{R}_x$ geben an in welchen Bereichen die Klassifizierung zu Gunsten einer Klasse x ausfällt. $\mathcal{R}_\emptyset$ bezeichnet Regionen in denen eine Klassifizierung nicht möglich ist.

Eine-Gegen-Alle

Die erste Möglichkeit einen Algorithmus von einem Zwei-Klassen-Problem auf ein Mehr-Klassen-Problem zu erweitern ist die Variante *Eine-Gegen-Alle*. Die Idee dieser Erweiterung ist es, für jede der insgesamt s Klassen einen Algorithmus zu Trainieren der eine Aussage darüber Treffen kann, ob ein unbekannter Merkmalsvektor zu einer Klasse l_s gehört oder nicht. Jedes Exemplar des Algorithmus trennt den Mehrmalsraum seinerseits in zwei Halbräume. Dadurch wird der gesamte Merkmalsraum in Regionen aufgeteilt, die durch die begrenzenden Trennebenen Aufschluss über die Klassenzugehörigkeit eines unbekannten Merkmalsvektors geben. In Abbildung 2.4(a) wird ein Beispiel einer solchen räumlichen Aufteilung wird gezeigt.
Für eine Region innerhalb des aufgeteilten Raumes gilt, dass sich diese für jede Trennebene entweder auf der *positiven*, der Klasse zugehörigen Seite, oder der *negativen*, der Klasse nicht angehörigen, Seite befindet. Da sich Elemente auf mehreren positiven Seiten der Trennebenen befinden können, kann einen eindeutige Klassenzugehörigkeit für diese Elemente nicht

voraus gesagt werden. Gleiches gilt für diejenige Region $\mathcal{R}_\emptyset$, die sich auf der negativen Seite aller Trenngeraden befindet. Daraus folgt, dass eine eindeutige vorhersage einer Klasse nur für Regionen möglich ist, die sich auf der positiven Seite von genau einer Trennebene befinden.

Paarweises Trennen

Ähnlich der *Eine-Gegen-Alle*-Trennung werden auch bei der Idee des *Paarweisen-Trennens* mehrere Exemplare von Klassifizierungsalgorithmen dafür genutzt den Merkmalsraum in verschiedene Regionen auf zuteilen. Jedoch werden bei dieser Variante die Algorithmen dafür genutzt jeweils zwei Klassen der insgesamt s Klassen von einander zu trennen. Daraus ergeben sich $\frac{n(n-1)}{2}$ Trennebenen die den Merkmalsraum unterteilen. Auch in diesem Verfahren legt eine Region in der Regel die Zuweisung mehrerer Klassen fest. Jedoch können verschiedene Trennebenen eine Klasse für eine Region mehrfach definieren. Befindet sich ein Element in einer solchen Region, wird die Klassenzugehörigkeit durch eine Mehrheitsentscheidung getroffen. Dieses Verfahren wird in Abbildung 2.4(b) veranschaulicht.
Einen ausführlichen Vergleich dieser Verfahren speziell für SVM bietet die Arbeit von Tsujinishi u. a. [24], die als Ergebnis ihrer Untersuchungen die paarweise Trennung von Klassen vorschlagen.

2.4.3 Regression

Die Regressionsanalyse umfasst Verfahren, mit denen ein Zusammenhang zwischen mindestens einer unabhängigen und einer abhängigen Variable durch eine Funktion $g : \mathbb{R}^m \to \mathbb{R}$ zu beschreiben. Im Kontext dieser Arbeit entsprechen die unabhängigen Variablen den Ausprägungen der Merkmale und die abhängige Variable der Klasse, welche diesen zugeordnet wird. Im Allgemeinen wird die gesuchte Funktion mit Hilfe von Parametern beschrieben. Diese Parameter werden mit dem Vektor $\omega \in \mathbb{R}^a$ bezeichnet. Die gesuchte Funktion entspricht somit der Form

$$g(x_i, \omega) + \epsilon \tag{2.34}$$

wobei ϵ einen Fehlerterm und $x_i = f(X_m^{d_i})$ einen Merkmalsvektor in $\mathbb{R}^a$ beschreiben. Auf Grund dieser Definition wird zunächst von einem reelwertigen Merkmalsraum $\mathcal{L}^* = \mathbb{R}$ ausgegangen.

Lineare Regression

Im Spezialfall der linearen Regression wird davon ausgegangen, dass es sich der Zusammenhang der unabhängigen Variablen durch eine lineare Funktion der Form

$$g_{linear}(x_i, \omega) = b + \omega^T x_i + \epsilon_i \tag{2.35}$$

ausdrücken lässt. Die Fehlerterme ϵ_i einen Fehler beschreiben, der sich auf eine einzelne Beobachtung x_i bezieht. Die Bezeichnung *linear* bezieht sich hierbei lediglich auf die Parameter ω der Funktion, was nicht lineare Terme im Bezug auf die Merkmalsvektoren zulässt.

Damit die Funktion unbekannte Daten bestmöglich voraussagen kann müssen die Gewichte ω so gewählt werden, dass das Ergebnis einer Verlustsfunktion L minimiert wird. Eine solche Verlustsfunktion beschreibt, wie Falschaussagen durch das Modell bewertet werden. Ein gängiges Beispiel ist die *Squared-error-loss*-Funktion L_2 wertet eine Falschaussage mit dem quadratischen Abstand zum korrekten Wert. Sie ist definiert durch

$$L_2(T, g) = \frac{1}{2} \sum_{(x_i, l_i) \in T} [g(x_i) - l_i]^2 \tag{2.36}$$

Da die die Regressionsfunktion auf den Raum der reellen Zahlen $\mathbb{R}$ abbildet, kann sei nicht direkt für eine diskrete Klassenzuweisung genutzt werden. Deshalb muss eine weitere Funktion g^* gefunden werden, die das Ergebnis den diskreten Raum der Kategorien $\mathcal{L}$ abbildet. Die einfachste Möglichkeit bietet ein Schwellwertverfahren. Für ein solches Verfahren wird ein Schwellwert t festgelegt und die Funktion

$$g^*(x_i) = \begin{cases} l_1 & \text{wenn } g(x_i, \omega) > t \\ l_0 & \text{sonst} \end{cases} \tag{2.37}$$

eine von zwei Klassen zuweist. Jedoch können Ausreißer der Merkmalsvektoren, in Verbindung mit der Fehlerfunktion L_2 zu Problemen führen. Da ein solcher Ausreißer zu einem großen Fehler führt, kann es passieren, dass die übrigen Daten in der Optimierungsfunktion ausgestochen werde. Ein Ansatz dieses Problem zu lösen, bietet die logistische Regression.

Logistische Regression

Wird ein Zwei-Klassen-Problem mit $\mathcal{L} = \{0, 1\}$ betrachtet, kann die Wahrscheinlichkeit, dass für einen Merkmalsverktor x_i die Klasse $l_i = 1$ vorhergesagt wird, durch $P(l_i = 1|x_i)$ angeben, die Gegenwahrscheinlichkeit mit $P(l_i = 0|x_i) = 1 - P(l_i = 1|x_i)$. Das Verhältnis von einer Wahrscheinlichkeit zu seiner Gegenwahrscheinlichkeit wird als *odds* bezeichnet und ist definiert durch

$$odds = \frac{p(l_i = 1|x_i)}{p(l_i = 0|x_i)} = \frac{p(l_i = 1|x_i)}{1 - p(l_i = 1|x_i)} \tag{2.38}$$

Da diese Funktion durch 0 beschränkt ist, werden die sogenannten *Logits* definiert, um auf $\mathbb{R}$ abzubilden.

$$\log odds = \log \left(\frac{P(Y = 1|x)}{P(Y = 0|x)} \right) \tag{2.39}$$

mit

$$g(x_i, \omega) = \omega^T x_i = \log \left(\frac{p(Y = 1|x)}{p(Y = 0|x)} \right) \tag{2.40}$$

ergibt sich die logistische Sigmoidfunktion

$$g(x_i, \omega) = P(l_i = 1|x_i) = \frac{exp(\omega^T x_i)}{1 + exp(\omega^T x_i)} \tag{2.41}$$

Diese lässt sich durch die Schwellwertfunktion g^* wieder auf ein diskretes Zwei-Klassen-Problem zurückführen.

2.4.4 Stützvektormaschine

Die Grundidee der SVM ist die Trennung zweier Klassen durch eine Hyperebene im Merkmalsraum. Die Trennebene die durch das Trainieren einer SVM gefunden wird soll die beiden Klassen bestmöglich von einander Trennen. Das bedeutet, dass eine Hyperebene gefunden werden muss die zu beiden Klassen einen maximalen Abstand b hält. Diejenigen Merkmalsvektoren welche genau diesen Abstand von der Trennebene einhalten werden dann als die *Stützvektoren* bezeichnet. Um diese Begriffe zu veranschaulichen wird in Abbildung 2.5 wir eine optimale Trennebene mit den entsprechenden Stützvektoren gezeigt. Das Für linear separierbare Daten ergibt sich mit den Gleichungen 2.32 und 2.33 das Optimierungsproblem der SVM:

$$\text{Minimieren von:} \quad \tfrac{1}{2}||\omega||_2^2 \tag{2.42}$$

$$\text{Unter der Bedingung:} \quad l_i \frac{g(x_i)}{||\omega||_2} \geq 1 \quad \text{für alle i=1}\ldots\text{n} \tag{2.43}$$

Da die Daten in der Regel nicht linear separierbar sind, wird die Problemstellung um eine Menge von n Nebenbedingungen erweitert welche Fehler in den Daten modellieren. Diese Fehlertherme $\xi_i \geq 0$ und ihre Kosten $c \geq 0$ erweitern das Optimierungsproblem folgendermaßen:

$$\text{Minimieren von:} \quad \tfrac{1}{2}||\omega||_2^2 + c\sum_{i=1}^{m} \xi_i \tag{2.44}$$

$$\text{Unter der Bedingung:} \quad l_i \frac{g(x_i)}{||\omega||_2} \geq 1 - \xi_i \quad \text{für alle i=1}\ldots\text{n} \tag{2.45}$$

Durch die Aufnahme der Fehlerterme in die zu minimierende Funktion wird auch Fehler minimiert, der durch die Verletzung der Nebenbedingungen entsteht.

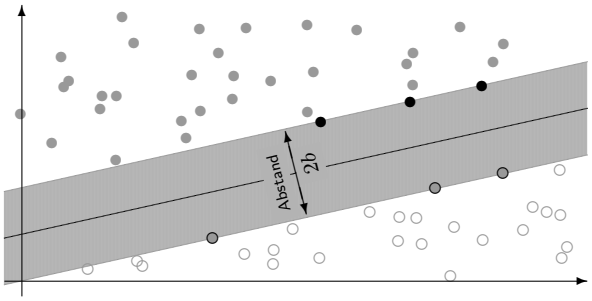


Abbildung 2.5: Zwei Klassen, • und ∘, werden durch eine Hyperebene von einander getrennt. Die dunkel gekennzeichneten Repräsentanten der Klassen stellen die Stützvektoren der Trennebene mit Abstand b dar.

2.5 Bewertung von Modellen

Ein Modell dient dazu die Daten die es klassifizieren soll zu beschreiben. Jedoch werden die Modelle nur auf Daten trainiert die im Vorfeld bekannt sind und es zunächst nicht vorhergesagt werden, wie eine konkrete Modellierung unbekannte Daten Klassifiziert. Damit dennoch eine Abschätzung über die Güte des Modells abgegeben werden kann werden Verfahren benötigt, die es erlauben, unabhängig von einem Modell ein Maß für Güte bieten. Der zunächst einfachste Ansatz wäre es ein Modell auf einem Satz von Beobachtungen zu trainieren und danach die Kategorien der gleichen Daten durch das Modell zu bestimmen. Jedoch kann dieser Ansatz kein geeignetes Maß für die Güte bieten, da das Modell die Trainingsdaten auswendig lernen könnte und somit auf den Testdaten perfekte Vorhersagen treffen würde, aber an realen Daten scheitern würde. Dieses Problem der Überanpassung (Overfitting) kann auch im zweiten Ansatz nicht ausgeschlossen werden, in dem die Trainingsdaten in zwei Teile, die Trainingsdaten selbst und ein Anteil Testdaten, getrennt wird. Denn auch hier kann es bei einer zufälligen Aufteilung dazukommen, dass die Testdaten den Trainingsdaten zu ähnlich sind. Dennoch kann dieser Ansatz dadurch verbessert werden, dass die zufällige Aufteilung in zwei Datensätze mehrfach wiederholt wird und für jede dieser Aufteilungen eine Schätzung der Güte abgegeben wird und der Mittelwert dieser Ergebnisse als Maß für die Güte des Modells bietet. Diese sogenannte *Kreuzvalidierung* bietet ein zuverlässiges Testverfahren und wird im Kapitel 2.5.2 näher beschrieben.

2.5.1 Konfusionsmatrix

Eine genauere Betrachtung der Klassifizierungsergebnisse bietet die sogenannte Konfusionsmatrix. Sie stellt die vorhergesagte Ergebnisse einer Testmenge und die tatsächlichen Klassen gegenüber. Für eine Trainingsmenge $T = \{(x_1, l_1), \ldots, (x_n, l_n)\}$ und zwei Klassen $p, t \in \mathbb{L}$ ist

$$\mathcal{C}_g(p,t) = |\{x_i | (x_i, t) \in T : g(x_i) = p\}| \tag{2.46}$$

Die Anzahl der Beobachtungen aus der Testmenge die der Klasse t angehören und als Klasse p eingeordnet wurden. Die Trennschärfe (accuracy) einer Klasse gegenüber den übrigen Klassen ergibt sich dann als

$$q_{g,t} = \frac{\mathcal{C}_g(t,t)}{\sum_{l \in \mathcal{L}-t} \mathcal{C}_g(l,t)} \tag{2.47}$$

Die Güte des Modells ist dann durch das arithmetische Mittel der Trennschärfen definiert:

$$Q(g) = \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} q_{g,l} \tag{2.48}$$

2.5.2 Kreuzvalidierung

Die Kreuzvalidierung bietet ein Verfahren die Güte eines Modells zu bestimmen. Dafür wird ein Testdatensatz benötigt, der neben der Merkmalsvektoren auch die zugehörigen Klassen

enthält. Ein solcher Testdatensatz $T = \{(x_1, l_1), \ldots, (x_n, l_n)\}$ wird in $k \in \mathbb{N}^+$ disjunkte Teildatensätze aufgespalten die annähernd gleich groß sind:

$$T = \dot{\bigcup_{i=1\ldots k}} T_i \text{ mit } T_i \cap T_j = \emptyset \text{ für } i \neq j \tag{2.49}$$

Diese werden wiederum zu k Datensätzen $T \setminus T_i$ zusammengesetzt, die für das Training eines Modells g_i benutzt werden. Für jedes dieser trainierten Modelle wird die Güte anhand des entsprechenden Testdatensatzes bestimmt. Das arithmetische Mittel $\mathcal{X}_k$ dieser Werte gibt die Güte des Modells an:

$$\mathcal{X}_k(g) = \frac{1}{k} \sum_{i=1}^{k} Q(g_i) \tag{2.50}$$

Das Problem dieser Form der k-fachen-Kreuzvalidierung besteht darin, dass die Elemente der Teildatensätze zufällig bestimmt werden. Damit kann es geschehen, dass eine Klasse von Daten innerhalb eines Testdatensatzes T_i kaum vertreten ist und das Modell somit nicht auf diese Klasse trainiert wird. Damit dieses Problem umgangen wird, bietet die *stratifizierte Kreuzvalidierung* eine Lösung, indem bei der Aufteilung der Datensätze darauf geachtet wird, dass die Verteilung der Klassen innerhalb der Testdatensätze der Verteilung des Gesamtdatensatzes entspricht.

KAPITEL 3

Systementwurf

Die Grundlagen, die im vorangegangenem Kapitel, erfasst wurden, bilden die Basis für ein System, das Aufgaben im Rahmen des maschinellen Lernens lösen soll. Im Rahmen dieser Arbeit soll ein solches System von Psychologen und Informatikern gemeinsam genutzt werden.
Die Informatiker werden in hier als Entwickler von Verfahren angesehen, die Auswertungen von Daten ermöglichen. Praktisch gesehen umfasst dies, die Entwicklung und Verbesserung sowohl von Methoden zur Klassifikation, als auch Verfahren, die diese Methoden auswerten.
Die Psychologen benötigen ein Werkzeug, das ihnen erlaubt diese Verfahren zu nutzen um die von ihnen erhobenen Daten auswerten und die Ergebnisse mit herkömmlichen statistischen Verfahren weitergehend analysieren zu können.
Bisher arbeiten beide Parteien größtenteils isoliert voneinander, könnten jedoch gegenseitig von den Forschungen der jeweils anderen Partei profitieren. So bieten Daten, die innerhalb einer psychologischen Studie erhoben werden, eine eine Möglichkeit für die Entwickler von Klassifizierungsalgorithmen, selbige mit reellen Daten zu testen und die Ergebnisse nutzen, um die Verfahren zu verbessern. Andererseits können diese Ergebnisse auch genutzt werden, um die Verfahren zu verbessern, die für Datenerhebung genutzt wurden.
Ein weiterer Nutzen, der aus diesem Projekt gezogen werde soll, ist das Verbessern der Zusammenarbeit innerhalb der einzelnen Disziplinen. So beschäftigen sich mehrere Arbeitsgruppen mit Analysen von verschiedenen Datengrundlagen wie z.B. MRT-, EEG-, EKG-Aufzeichnugen, Beschleuninigungsdaten, Motion-Capture-Daten, etc. Da sich viele dieser Daten auf die selbe Art beschreiben lassen, können die verschiedenen Vorgehensweisen beim Auswerten und Analysieren auch auf andere Daten angewendet werden. Zudem bietet eine Schnittmenge von Probanden, welche an verschiedenen Studien teilgenommen haben, eine Möglichkeit, diese Studien gemeinsam zu betrachten und somit die Grundlage einer neuen Studie zu schaffen.

3.1 Anforderungen

Ein System, das mit dem die gestellten Aufgaben gelöst werden sollen, muss zunächst eine Vielzahl von Datenformaten unterstützten. Denn auch wenn sich die abstrakte Beschreibung der Daten ähnelt, werden die Daten selbst von verschiedenen Geräten aufgezeichnet, die jeweils spezielle Formate für die Speicherung nutzen. Ebenso bringen neue Technologien und Weiterentwicklungen von alten System meist neue Datenformate mit sich, die von dem System nutzbar sein sollten. Somit muss das System entweder die Möglichkeit bieten, diese Formate zu konvertieren oder sie intern verarbeiten können.
Zudem ist es leicht ersichtlich, dass die gemeinsame Verwaltung von Daten verschiedener Gruppierungen, Studien und Formaten schnell zu Speicherproblem führen kann. Es muss also gewährleistet werden, dass das System mit großen Datenmengen umgehen kann. Das heißt, diese zu verwalten und verarbeiten zu können.

Auf Grund der Vielfalt von Daten und der verschiedenen Handhabungen, die sich durch das Zusammenführen von Arbeitsgruppen ergeben, müssen diese Daten einfach und mit einheitlichen Zusatzinformationen versehen gespeichert werden können. Damit soll der Austausch der Daten vereinfacht werden. Informationen, die den Daten beigefügt werden, sollten Beschreibungen der Daten, wie Umstände und Ziele der Erhebung enthalten. Damit soll es den Benutzern, welche die Daten fern der eigentlichen Erhebung nutzen wollen, im Voraus prüfen können, ob sie diese verwerten können.
Ein weiterer Aspekt der bei der Entwicklung des System berücksichtigt werden muss, ist das Weiterverarbeiten der Ergebnisse. Es muss davon ausgegangen werden, dass Berechnungen, die von dem System ausgeführt werden, auf Grund des Datenvolumens, viel Zeit in Anspruch nehmen und ein Benutzer des Systems somit nicht für die gesamte Dauer der Berechnungen mit dem System verbunden sein wird. Daraus folgt, dass Ergebnisse von Berechnungen von dem System gespeichert werden müssen. Ein weiterer Punkt, der diese Forderung unterstützt, ist das mehrfache Verwenden solcher Ergebnisse. Zudem müssen auch Ergebnisse von Berechnungen von den Nutzern weiterverarbeitet werden können.
Aus der Problemstellung selbst ergeben sich zunächst zwei Sichtweisen. Zum einen die der *Anwender*, welche Daten erheben und analysieren wollen und zum anderen die der *Entwickler*, welche neue Modelle entwickeln und diese mittels erhobener Daten testen und vergleichen wollen. Diese beiden Sichten müssen innerhalb eines solchen Systems vereint werden, um möglichst wenig in die aktuellen Arbeitsweisen einzugreifen.

Anwender

Den Anwendern muss ein einfacher Zugang zu dem System geboten werden, damit schnell klar ist, wie das System zu nutzen ist und welche Vorteile geboten werden. Zunächst müssen die Anwendern eine Möglichkeit erhalten, die Rohdaten einer Datenerhebung selbstständig in das System einzuschleusen, um sie mit dem System analysieren zu können. Die damit zur Verfügung stehenden Daten müssen dann durch den Anwender für die Analysen aufbereitet werden können. Konkret bedeutet dies, dass die Daten in ein für das System gültiges Format gebracht werden müssen und bereits in Kategorien eingeteilt worden sein. Dieser Schritt

der Bearbeitung sollte jedoch auch durch die Anwendung geboten werden, da diese unter Umständen Hilfestellungen leisten kann.
Im weiteren Verlauf müssen die Anwender verschiedene Modelle erstellen können, um ihre Daten zu analysieren. Hierbei sollte es möglich sein, verschiedene Modelle parallel zu testen. Die Ergebnisse dieser Tests müssen den Nutzern nach ihrer Berechnung im einem Format zur Verfügung gestellt werden, in dem sie weiter verarbeitet werden können. Hierbei muss berücksichtigt werden, dass verschiedene Anwender auf unterschiedliche Weisen mit den Ergebnissen arbeiten und sich deren Systeme grundlegend unterscheiden können.

Entwicker

Die Sicht der Entwickler auf das System unterscheidet sich grundlegend von der Sicht der Anwender. Zwar sind die ersten Möglichkeiten, die einem Entwickler durch das System geboten werden sollen, die gleichen. Doch müssen sich die Funktionen innerhalb einer Entwicklungsumgebung befinden. Für die Entwickler soll der Zugriff auf das System innerhalb der jeweiligen Programmiersprache geboten werden. Da Entwickler mit verschiedenen Programmiersprachen arbeiten, muss das System diese Umgebungen verbinden können, um die unterschiedlichen Entwicklungen an die Anwender weiterleiten zu können. Auch den Entwicklern anderer Arbeitsgruppen muss die Nutzung fremder Systeme ermöglicht werden.
Die Entwickler benötigen ein Werkzeug (Framework), welches ihnen erlaubt einzelnen Methoden des Klassifizierens von Daten zu entwickeln und außerdem auf bereits entwickelte Komponenten anderer Entwickler zugreifen zu können. Ein solches Rahmenwerk muss also Schnittstellen für die Entwicklung von Klassifizierungsalgorithmen, Merkmalen, Filtern und Analysen zur Verfügung stellen und einfach in die bereits vorhandenen Umgebungen eingebunden werden können. Es ist darauf zu achten, dass sich diese Framework in beliebigen Programmiersprachen umsetzen lassen soll.

3.2 Entwurf

Durch die Aufteilung in die verschiedenen Sichtweisen und das auf Teilen in mehrere Systeme, wird eine zentrale Komponente notwendig, die diese Teilsysteme miteinander verbindet und die Daten des Gesamtsystems verwaltet. Diese Komponente wird im Folgenden als *Master*-Komponente bezeichnet. Anderen Komponenten, wie die Entwicklersysteme und Anwendungen, die auf das System zugreifen, werden als Entwicklerknoten und Anwendungsknoten innerhalb einer Kommunikationsstruktur betrachtet. Damit diese Komponenten Daten und Funktionsaufrufe untereinander austauschen können, muss ein Protokoll gefunden werden, das diese Aufgabe übernimmt.

3.2.1 Protokoll

Eine grundlegende Technik, die diese Forderungen erfüllt, sind sogenannte R*emote* P*rocedure* C*alls* (RPC) [26, 13, 23]. Sie bieten eine Möglichkeit, Funktionen von verschieden Systemkomponenten her aufzurufen und Daten auszutauschen. Die Technik selbst ist dabei ein so

erfolgreiches Prinzip, dass es bereits eine Vielzahl von Implementierungen gibt. Doch sind diese Implementierungen meist nicht untereinander kompatibel und sondern auf ihr spezielles Einsatzgebiet beschränkt. So bringen die meisten Programmiersprachen eine eigene Implementierung dieser Interprozesskommunikation mit, die zwar innerhalb dieser Sprache Funktionsaufrufe und Datenaustausch zulassen, aber keine Schnittstelle zu anderen Sprachen zulassen. Die Remote Method Invocation (RMI) für Java™ oder das Python-Pendant Remote Python Calls (RPyCs) [4] sind Beispiele für Implementierungen, die einen Austausch von Funktionsaufrufen und Daten innerhalb des jeweiligen Systems zulassen. Für einen Austausch zwischen heterogenen Systemen, wie sie innerhalb dieses Projektes benötigt werden, sind diese Umsetzungen jedoch nicht nutzbar.
Es existieren jedoch andere Implementierungen, die den Austausch zwischen heterogenen Systemen zulassen. Die wohl bekanntesten Beispiele hierfür sind die Common Object Request Broker Architecture (CORBA) [18, 16, 17], das Simple Object Access Protocol (SOAP) [14, 9, 15] und XML-RPC [3]. Die CORBA bietet bereits eine mächtige Architektur für den geforderten Austausch von Informationen aber gerade diese macht es schwer Erweiterungen innerhalb eines bestehenden Systems umzusetzen. Das Simple Object Access Protocol bietet hier eine durchaus flexiblere Lösung an. Zum Repräsentieren der Daten werden XML-Dokumente genutzt und es wird eine Architektur vorgeschlagen, die den Anforderungen weitestgehend entspricht. Jedoch sorgt diese allgemeine Lösung durch ihre Definition der Daten für beträchtliche kosten beim Übertragen einfacher Daten. Eine wesentlich einfachere Umsetzung dieser Idee bietet XML-RPC. Diese Spezifikation basiert ebenfalls auf dem Austausch von Daten mittels XML-Dokumenten. Dabei wird der Aufbau von Anfrage- und Antwortdokumenten einfach gehalten und es wird lediglich eine geringe Anzahl von primitiven Datentypen spezifiziert. Es wird hierbei auf einige Systemdefinition, wie einen Weitertransport von Daten mittels Zwischenknoten, Nachrichtenpfaden und einigem mehr, verzichtet. Zudem wird im Folgenden wird ein Protokoll aufgebaut, welches sich an dieser Lösung orientiert und mit ihr kompatibel bleibt.

Nachrichtenaustausch

Der Austausch von Nachrichten basiert auf dem Client-Server-Modell. Die Server-Seite, welche die Dienste im Netzwerk verteilen soll, stellen sowohl der Master-Knoten als auch die Entwicklerknoten dar. Jede Anfrage innerhalb des Systems wird durch einen Client initialisiert und beginnt mit der Übertragung eines Anfragedokuments durch den Client. Hat der Server ein solches Dokument erhalten, ist die Übertragung von der Client-Seite her beendet. Das bedeutet, dass in dieser Richtung keine weiteren Daten mehr übertragen werden. Danach wird die Anfrage durch den Server bearbeitet und aus dem Ergebnis ein Antwortdokument erstellt, welches dann an den Client übertragen wird. Nach der Übertragung dieses Antwortdokuments wird die Verbindung von beiden Seiten beendet.
Ein wichtiger Aspekt bei diesen Aufrufen ist die Verarbeitung und Weiterleitung von Funktionsaufrufen und Ergebnissen durch den Master. Denn dadurch ist es möglich, dass der Master Informationen, die innerhalb des Systems gespeichert werden, konsistent hält und gegebenenfalls Ergebnisse zwischenspeichern kann, um sie bei einer erneuten Anfrage nicht

neu berechnen zu müssen.
Eine besondere Form in diesem Nachrichtenaustauschs stellen die eingehend erwähnten Remote Procedure Calls dar. In diesem Fall bedeutet die Verarbeitung durch den Server, dass dieser selbst als Client eine Verbindung zu einer anderen Komponente aufbaut, um eine erforderliche Unterstützungsfunktion zu erfragen.

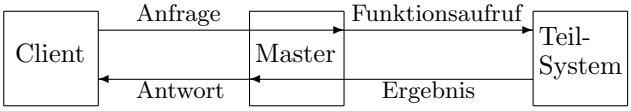


Abbildung 3.1: Remote Procedure Call Prinzip: Eine beliebige Client-Komponente greift auf eine Funktion zu, welche von einer Entwicklerkomponente zur Verfügung gestellt wird. Der Master übernimmt dabei die Weiterleitung des Aufrufs und des Ergebnisses.

Funktionsregistrierung

Damit Funktionen, die von einer Komponente des Systems zur Verfügung gestellt werden, von anderen Komponenten genutzt werden können, müssen diese Registriert werden. Das bedeutet, dass dem System eine solche Funktion bekannt gegeben werden muss. Hierfür schickt die Entwicklerkomponente, welche eine Funktion nutzbar macht, eine Registrierungsanfrage an den Master. Dieser speichert dann sowohl die Kennung des registrierenden Systems als auch eine lokale Kennung der Funktion und generiert eine globale Kennung mit der die Funktion dann im gesamten System eindeutig identifiziert werden kann. Andere Komponenten können dann mit Hilfe dieser Kennung den Master veranlassen, diese Funktion auf dem entsprechenden Knoten ausführen lassen. Die lokale Kennung identifiziert eine Funktion eines Teilsystems und wird bei Funktionsaufrufen durch die Master-Komponente benutzt.

Datenregistrierung

Ähnlich wie die Registrierung von Funktionen müssen auch Datensätze, mit denen gearbeitet werden soll, registriert und von dem System übernommen werden. Jedoch werden die Daten nicht von einzelnen Komponenten zur Verfügung gestellt. Sie werden vielmehr zentral verwaltet, damit sie nicht außerhalb des Systems verändert können. Dies ist notwendig, damit die Konsistenz der Daten nicht verletzte werden kann. Wird ein Datensatz registriert, bedeutet dies, dass eine Kopie dieses Datensatzes mit einigen Zusätzlichen Informationen wie Name und Beschreibung an den Master übertragen, von diesem gespeichert und eine systemweit eindeutige Kennung zurückgegeben wird. Mit dieser Kennung wird der Datensatz zukünftig identifiziert.

3.2.2 Architektur

Die Master-Komponente stellt den Kern des Systems dar. Sie ist dafür zuständig alle Daten des Systems zu erfassen, zu verwalten und die Kommunikation zwischen den einzelnen Komponenten zu ermöglichen. Ebenso sorgt sie für die Speicherung der Daten und gibt die

Möglichkeit zusätzliche Informationen mit den Daten zu speichern um diese aus Sicht eines Benutzers leichter identifizieren zu können und aus Anwendungsperspektive zu nutzen um Rechenaufwand und Übertragungsvolumen gering halten zu können. Abbildung 3.2 zeigt die Grundidee dieser Architektur.

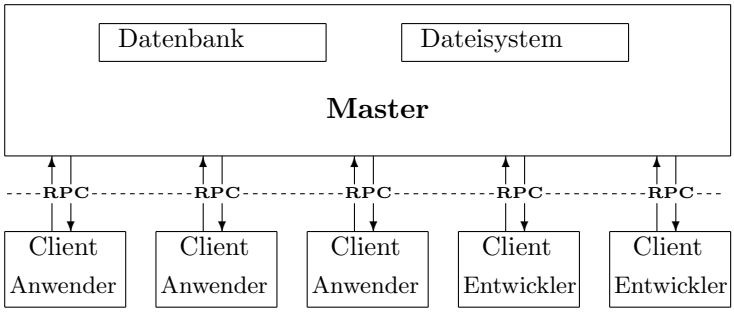


Abbildung 3.2: Grundstruktur des Systems. Die einzelnen Teilsysteme (Clients) kommunizieren mittels Remote Procedure Calls mit der Master-Komponente . Der Master verwaltete die Ressourcen auf einem Dateisystem und die verwaltet Zusatzinformationen mit einer Datenbank

Dateisystem

Die Menge von Daten, die vom Master verwaltet wird, ist zwar nicht festgelegt, aber es ist leicht zu erkennen, dass diese schnell ein Volumen annehmen kann, dass von einem einfachen Speichersystem nicht mehr verwaltet werden kann. Aus diesem Grund wird bei der Entwicklung des Systems davon ausgegangen, dass die Datensätze auf einem verteilten Dateisystem abgelegt werden. Auch wenn ein solches Dateisystem in der Regeln für die zugreifenden Komponenten transparent sind, können hier Seiteneffekte auftreten die es zu vermeiden gilt. Ein Beispiel hierfür ist das automatische Durchsuchen und Indexieren der Dateistruktur, welches schnell die Grenzen eines einzelnen Systems erreichen kann.

Datenbank

Neben den eigentlichen Daten muss die Master-Komponente auch andere Informationen speichern und effizient auf diese zugreifen können. Dies sind zum einen Metainformationen der erfassten Daten und zum anderen die Kennungen von Datensätzen und Funktionen, die von dem Master verwaltet werden. Die Hauptkomponente des entwickelten Systems nutzt ein Datenbanksystem, um diese Informationen zu effizient zu speichern und abzurufen zu können.

Die Nutzug des Systems erfolgt durch Anwender und Entwickler. Diese beiden Benutzergruppen sollten aus Erfahrung [8] von einander getrennt betrachtet werden. Aus diesem Grund werden für Clients, die sich mit dem Master verbinden, zwei Arten von Knoten unterschieden: Entwicklerknoten und Anwenderknoten.

Entwicklerknoten

Die sogenannten Entwicklerknoten stellen vollständige Untersysteme des Masters dar. Das bedeutet, dass innerhalb eines solchen Systems die Entwicklung von Methoden der Mustererkennung ohne Nutzung des Masters stattfinden kann. Jedoch soll es durch Einbinden des Masters möglich sein, die dort zu Verfügung gestellten Methoden nutzen zu können. Ebenso soll die Funktionalität dieser Systeme auf dem Master registriert werden können, um einen Zugriff von Außen auf diese Systeme nutzen zu können. Damit wird eine Möglichkeit geschaffen, neue Entwicklungen untereinander breit zustellen und zu nutzen.

Anwenderknoten

Für die Anwender ist das Erweitern der Funktionalität unerheblich. Sie sollen die Funktionen der einzelnen Systeme lediglich im Rahmen der Analysen nutzen und die Datenbasis des Systems erweitern können. Hierfür müssen die Funktionen und die Methodik so aufbereitet werden, dass ein Anwender ohne tiefere Kenntnisse der Mustererkennung, mit den arbeiten kann.

Zwar werden die Nutzer an dieser Stelle in zwei Gruppen unterteilt, aber die Grundfunktionen, die zur Verfügung gestellt werden, sind im wesentlichen identisch. Der größte Unterschied besteht in der Erweiterung der Funktionalität, die einem Anwender nicht geboten werden muss. Daher ergibt sich für den Nachrichtenaustausch der Systemkomponenten kein Unterschied. Jedoch besteht in der Aufbereitung der Informationen ein wichtiger Unterschied, der berücksichtigt werden muss. Für die Entwicklung des Systems ergibt sich somit neben dem Entwurf eines Protokolls und der Master-Komponente auch die Aufgabe des Entwurfs von Rahmenwerken für verschiedene Programmiersprachen und eine leicht bedienbare Anwendung, die jeweils den Zugriff auf die zentrale Komponente ermöglicht.

Nachrichten-Bypass

Einige Berechnungen die das System ausführen soll benötigen eine viel Zeit, bis ein Ergebnis geliefert werden kann. Diese Tatsche kann innerhalb der Verbindungen zu unerwünschten Nebenwirkungen wie einem Verbindungsabbruch (Timeout) führen. Damit dieses Problem umgangen werden kann und die Ergebnisse solcher Berechnungen dennoch zeitnah übertragen werden, wird ein Bypass zur Verfügung gestellt. Als unbestätigter Dienst dient dieser Kommunikationskanal lediglich einem Austausch von Statusmeldungen zwischen einem Teilsystem und der Master-Komponente. Die Basis dieses Bypasses bildet das User Datagram Protocol.

KAPITEL 4

Schnittstellen

Damit die Aufgaben der Mustererkennung auf verschiedene Systeme verteilt werden kann, ist es notwendig, einheitliche Schnittstellen festzulegen. Durch diese Schnittstellen wird der Austausch von Daten und Funktionsaufrufen zwischen den einzelnen System-Komponenten definiert. Sie basieren auf dem in Kapitel 3.2.1 beschriebenen RPC-Protokoll.

Die Extendable Markup Language (XML) ist eine Repräsentation für Daten, die zwischen verschieden Computersystemen ausgetauscht werden sollen. Die wesentlichen Grundziele dieser Sprache sind neben der einfachen Handhabung und der Unterstützung einer Vielzahl von Anwendungen auch die Lesbarkeit durch Menschen und eine knappe, formale Beschreibung von Daten (Vgl. Bray u. a. 6). Aus diesen Zielen ist 1998 eine Auszeichnungssprache vom World Wide Web Consortium (W3C) vorgeschlagen worden, die für die Übertragung von Daten *XML-Dokumente* nutzt. Diese Dokumente bestehen jeweils aus genau einem *XML-Element*. Sie stellen die übertragbare Grundeinheit dar.
Ein XML-Element selbst setzt sich wiederum aus einer Menge von *Attributen* zusammen, die jeweils aus einer Kombination von einem Schlüsselwort und einem Wert gebildet werden. Diese Elemente können beliebig viele Kind- oder Text-Elemente enthalten. Sie sind durch ein *Start-* und ein *End-Tag* begrenzt. Die *Entitäten* (Kind- und Text-Elemente) zwischen diesen Begrenzungen werden als untergeordneten Element betrachtet. Dadurch entsteht eine eindeutige, hierarchische Baumstruktur.

Durch die Extendable Markup Language wird die Sprache der Schnittstellen beschrieben, für eine vollständige Definition der Schnittstellen müssen jedoch auch der Aufbau der Elemente und Dokumente festgelegt werden. Für diese Festlegung empfiehlt das W3C die sogenannten XML Schema Definitions (XSDs) [25]. Diese Definitionen legen Strukturen [22] fest, mit denen der Aufbau von Datentypen im XML-Format beschrieben werden kann. Das bedeutet, dass durch XSDs unter anderem festgelegt wird, von welchen Typ ein Attribute eines Elements zu sein hat, ob Attribute optional oder obligatorisch sind und welche untergeordneten Elemente ein Element enthält. Damit ist es auch möglich ein XML-Dokument auf

seine Gültigkeit zu prüfen. Neben den Definition für den Aufbau von Elementen bietet XSD einen Satz bereits definierter Datentypen, die für den Aufbau komplexer Datentypen [12] genutzt werden können. Mit Hilfe dieser Definitionen werden die Schnittstellen des System einheitlich festgelegt. Sie bieten dadurch die Grundlage für die Umsetzung in verschieden Umgebungen.
Ein Vorteil der XSDs besteht in der Möglichkeit, die Definitionen der Schnittstellen aufzugreifen und mit Programmen, wie beispielsweise dem Projekt *XMLBeans*[1] der Apache Software Foundation[2], automatisch in eine konkreten Programmiersprache zu überführen. Dadurch lässt sich der sich der Entwicklungsaufwand drastisch reduzieren lässt.

In diesem Kapitel wird betrachtet, welche Funktionen und Daten im Detail übertragen und wie diese im XML-Format repräsentiert werden. Hierbei handelt es sich zunächst um die semantischen Beschreibungen, deren Repräsentation durch Beispiele beschrieben werden. Die vollständigen Definitionen werden im Anhang A in der XML-Beschreibungssprache XSD angegeben. Diese Beschreibungen gliedern sich in drei Teile. Im ersten Teil werden die Datentypen beschrieben, welche von den Systemkomponenten verarbeitet und in eine interne Darstellung übersetzt werden. Auf diesen Datentypen aufbauend, werden im zweiten Teil Funktionen definiert, die von einzelne Teilsystemen implementiert werden können. Der letzte Satz von Definitionen umfasst die Beschreibung des Protokolls. Es werden Arbeitsschritte definiert, welche durch einen Nutzer beschrieben und durch das System bearbeitet werden.

4.1 Universalattribute

Eine der wichtigsten Eigenschaften, die das System erfüllen soll, ist das in Abschnitt 3.1 beschriebene Wiederverwenden und Reproduzieren von Daten, Ergebnissen und Analysen. Damit dies gewährleistet werden kann, muss das System in der Lage sein informelle Beschreibungen der Daten und Funktionen zu speichern und zu Übertragen. Solche Metainformationen sollen von einem Nutzer des System zu jedem erfassten Datum abgelegt werden. Sie sollen es einem Anwender erlauben, die Daten zu identifizieren und in einen Kontext einzuordnen. Aus diesem Grund werden die Datentypen, die nach außen sichtbar sind, d.h. auf die ein Anwender zugreifen kann, grundsätzlich mit zwei Attributen für einen Namen (`name`) und eine Beschreibung (`desc`) versehen. Zudem wird ein drittes Attribut (`id`) genutzt, um die Daten innerhalb des Systems zu identifizieren. Werden Daten für eine Maschine-Maschine-Übertragung verwendet, soll auf die informellen Attribute *Name* und *Beschreibung* verzichtet werden, um die Anonymität der Daten weitestgehend zu gewährleisten.

4.2 Datentypen

Zunächst werden die Datentypen betrachtet, die zwischen den Systemen ausgetauscht werden müssen, um die Parameter von Funktionen übertragen zu können. Bei diesen Parametern

[1] XMLBeans: http://xmlbeans.apache.org
[2] Apache Software Foundation: http://www.apache.org

handelt es sich um die erhobenen Daten, die Merkmale, die Modelle und und ihre Konfiguration die für die Klassifizierungsprozesse benötigt werden.

4.2.1 Datensätze und Kanäle

Datensätze stellen die Basis der Analysen dar. Im Rahmen dieser Arbeit repräsentieren sie Zeitreihen. Diese stellen in der Übertragung die wohl größte Herausforderung dar, denn sie können mit sehr hohen Abtastraten und über sehr lange Zeiträume aufgenommen worden sein. Sie können daher ein sehr großes Datenvolumen in Anspruch nehmen. Für eine Aufzeichnung der Daten einer Person, die mit acht Sensoren ausgestattet wurde, einer Abtastrate von 256Hz und einer Auflösung von typischerweise 32bit aufgezeichnet wurde, ergibt sich für eine Aufzeichnungsdauer von 24h ein Datenvolumen von mehr als 700Mb. Zudem nimmt dieses Volumen in der Darstellung des XML-Formats zu, da dieses Format die Daten in Textform repräsentiert.

Kanäle

Die Zeitreihen bestehen aus Kanälen, welche wiederum aus Werten in $\mathbb{R}$ zusammengesetzt sind. Diese Werte müssen durch das Protokoll übertragen werden und auf allen Teilsystem gleichermaßen interpretiert werden. Die gängigsten Systeme arbeiten mit einer Fließkomma-Darstellung, welche sich an dem Entwurf des Standards IEEE 754 [21] orientiert. Diese Darstellung wiederum basiert auf einer Repräsentation in binären Formaten und kann zwischen den meisten Systemen ausgetauscht werden. Jedoch kann diese Darstellung nicht unmittelbar in das durch XML definierte Format übernommen werden. Diese Darstellung der Werte kann Zeichen enthalten die mit dem zugrundeliegenden Format nicht konform sind. Aus diesem Grund werden die Werte in ein Format überführt, das es ermöglicht, diese Daten in XML-Elemente einzubinden. Die gängigsten Kodierungen die dieses Kriterium erfüllen sind `Base16`, `Base32` und `Base64` [11]. Diese beschreiben die Kodierung und Dekodierung von Zeichenketten die auf verschiedenen Alphabeten beruhen. Die Ziel-Alphabete dieser Kodierungen bauen entsprechend auf 16, 32 und 64 Zeichen auf, die alle innerhalb von XML-Elementen erlaubt sind.

Durch die beiden Umformungen wird eine Darstellung geschaffen, die sich in XML beschreiben lässt und damit zwischen verschiedenen Systemen ausgetauscht werden kann. Für die vollständige Beschreibung eines Kanals im Übertagungsformat werden einem Element, neben den Universalattributen, ein Attribut `encoding` für die Angabe der Kodierung und ein Attribut `type` für die Angabe der Werte Darstellung beigefügt. Zudem werden die Elemente um Attribute der beiden physikalischen Größen Abtastrate `samplingrate` in *Hz* und die Einheit `unit` der Zeitreihe selbst ergänzt.

Markierungen

Neben den Kanälen eines Datensatzes muss jeder Datentyp auch Markierungen von Zeitpunkten übertragen können. Eine solche Markierung trägt im wesentlichen zwei Informationen: den Zeitpunkt `position`, an dem ein Ereignis aufgetreten ist, und eine Bezeichnung `name` des

Ereignisses. Die Bezeichnung wird für die weitere Verarbeitung der Datensätze verwendet. Daher werden diese Bezeichnungen innerhalb von verschiedenen Datensätzen gleich belegt und geben nur eine einheitliche Kurzform des Ereignisses an. Um die Handhabung der Markierungen besser gestalten zu können, wird zudem das Beschreibungsattribut `desc` definiert. Da die Kanäle unter Umständen mit verschiedenen Abtastraten aufgezeichnet wurden, wird das Positionsattribut in Sekunden angegeben. Die Elemente, die einen Datensatz beschrei-

```
<DataObject id="DS001P1S0" name="Person1" desc="Obstacle course1...">

  <Channel id="xaccel" name="X-Axis" samplingrate="10"
           unit="m/s^2" desc="lateral acceleration"
           type="float" encoding="base64" >
     AAAAABXvwz7zBDU/XoNsPwAAgD9eg2w/8wQ1PxXvwz4=
  </Channel>

  <Channel id="yaccel" name="Y-Axis" samplingrate="10"
           unit="m/s^2" desc="anteroposterior acceleration"
           type="float" encoding="base64" >
     AACAP16DbD/zBDU/Fe/DPjIxjSQV78O+8wQ1v16DbL8=
  </Channel>

  <Marker position="0.00" name="Event1" desc="Step1... "/>
  <Marker position="0.05" name="Event2" desc="2nd Task..."/>
  <Marker position="0.10" name="Event3" desc="End of..."/>
</DataObject>
```

Quellcode 4.1: Beispiel eines Datensatzes. Die Definition wird in Anhang 1.3 spezifiziert

ben, setzen sich somit aus Kanal-Elementen und Markierungs-Elementen zusammen. Die Attribute entsprechen lediglich den Universalattributen. Ein vollständiges Beispiel eines solchen Elements ist im XML-Auszug 4.1 angegeben.
Die Daten dieses Typs sind in der Regel zu Umfangreich, um bei jedem Aufruf komplett übertragen zu werden. Aus diesem Grund sind zwei Abwandlungen vorgesehen. Diese Abwandlungen sind zum einen für die Auflistung von Datensätzen und zum anderen für eine partielle Übertragung gedacht.

Segmente repräsentieren nur Ausschnitte eines Datensatzes. Das bedeutet, dass sie ausgewählte Kanäle enthalten. Die Kanäle wiederum enthalten nicht die gesamte Aufzeichnung sondern nur einen zusammenhängenden Ausschnitt der Zeitreihe des Originaldatensatzes. Dieser Datentyp ist für die interne Kommunikation der System-Komponenten gedacht und ist nicht für den Anwender sichtbar. Aus diesem Grund werden die Universalattribute `id`, `name` und `desc` nicht benötigt. Jedoch werden zwei optionale Attribute definiert, die einen Verweis auf den Ursprungsdatensatz enthalten. Das Attribut `dataobject` gibt, falls vorhanden, die Kennung dieses Datensatzes an. Die Position des Segments innerhalb dieses Datensatzes kann durch das Attribut `position` in Sekunden angegeben werden. Ein solches

`Segment` wird in Quellcode 4.2 angegeben.

```
<Segment dataobject="DS001P1S0" position="0.0">
  <Channel id="xaccel" name="X-Axis" samplingrate="10"
           unit="m/s^2" desc="lateral␣acceleration"
           type="float" encoding="base64" >
     AAAAABXvwz7zBDU/XoNsPwAAgD9eg2w/8wQ1PxXvwz4=
  </Channel>

  <Channel id="yaccel" name="Y-Axis" samplingrate="10"
           unit="m/s^2" desc="anteroposterior␣acceleration"
           type="float" encoding="base64" >
     AACAP16DbD/zBDU/Fe/DPjIxjSQV78O+8wQ1v16DbL8=
  </Channel>
</Segment>
```

Quellcode 4.2: Beispiel eines Datensegments. Die Definition wird in Anhang 1.1 spezifiziert

Blinddatensätze stellen keine echten Datensätze dar, da sie keine Werte der Kanäle enthalten. Sie tragen lediglich die Basisinformationen über einen Datensatz, wie seine Identifikation, Namen, Beschreibung, eine Liste der Kanäle, welche ebenfalls nur die Basisinformationen enthalten, und die Markierungen. Das Beispiel 4.3 zeigt die Blinddaten des Datensatzes aus 4.1.

```
<DataObjectInfo id="DS001P1S0" name="Person1" desc="Obstacle␣course" >
  <Channel id="xaccel" name="X-Axis" samplingrate="10"
              unit="m/s^2" desc="lateral␣acceleration" />
  <Channel id="yaccel" name="Y-Axis" samplingrate="10"
              unit="m/s^2" desc="anteroposterior␣acce..." />
  <Marker position="0.00" name="Event1" desc="Step1...␣"/>
  <Marker position="0.05" name="Event2" desc="2nd␣Task..."/>
  <Marker position="0.10" name="Event3" desc="End␣of..."/>
</DataObjectInfo>
```

Quellcode 4.3: Beispiel eines Blinddatensatzes. Die Definition wird in Anhang 1.13 spezifiziert

Klassen Die Zuweisung von Klassen innerhalb der Datensätze erfolgt über die Bezeichnungen der Markierungen. Hierfür wird ein Elementtyp `Label` definiert, der sich aus sechs Attributen zusammen setzt. Neben den drei Universalattributen wird ein Attribut `dataobject` definiert, dessen Wert ein regulärer Ausdruck ist, der eine Klasse von Markierungsnamen beschreibt. Erfüllt der Name eines Datensatzes die durch den Ausdruck beschriebenen Bedingungen, wird dem gesamten Datensatz die durch das Element beschriebene Klasse zugewiesen. Die letzten beiden Attribute werden benötigt, um Segmente innerhalb eines Datensatzes zu klassifizieren. Sie tragen die Bezeichnungen `start` und `end` und bestimmen ebenfalls durch

reguläre Ausdrücke, an welcher Position eine Klasse beginnt und endet. Trifft der durch das Attribut `start` beschrieben Ausdruck auf den Namen einer Markierung in einem Datensatz zu, wird die nächste Markierung gesucht, die dem regulären Ausdruck entspricht, der durch das Attribut `end` beschrieben wird. Den Daten, die sich zwischen den entsprechen Markierungen befinden, werden dann der Klasse zugeordnet, die durch das Element beschrieben wird. Durch ein weiteres Attribut `parent` wird eine übergeordnete Klasse referenziert, die wie im Abschnitt 2.2.1 beschrieben ebenfalls zugewiesen wird. Im Quellcode 4.4 wird eine Klassendefinition beschrieben.

```
1 <Label id="label0" parent="label0" name="Walk" desc="Person␣walking"
2      dataobject=".*P1.*" start=".*[Ww]alk.*" end=".*"/>
```

Quellcode 4.4: Beispiel einer Klassendefinition *Walk*. Diese Klasse wird einem Datensatz zugeordnet dessen Name die Zeichenkette „P1“ oder eine Markierung mit dem Teilwort „walk“ enthält. Die Definition wird in Anhang 1.8 spezifiziert

4.2.2 Merkmalsräume und Ausprägungen

Die Beschreibung der Merkmalsräume stellt den nächsten wichtigen Datentyp dar. Er repräsentiert die Zusammensetzung von Merkmalen und Kanälen, von denen diese erhoben werden. Da nicht jedes Merkmal auf jeden Kanal eines Datensatzes angewendet werden soll, enthält jedes Merkmalselement neben der Kennung des Merkmals eine Liste der Kanalkennungen, die verarbeitet werden sollen. Eine Liste solcher Merkmalselemente wird zu einem Element zusammengefasst, welches einen Merkmalsraum beschreibt. Bei der Beschreibung eines Merkmalsraums handelt es sich, wie bei dem Datentyp `Segment`, um Daten, die nur für die interne Kommunikation der System-Komponenten genutzt werden. Daher benötigt dieser Datentyp keines der Universalattribute. Im Beispiel 4.6 werden die Merkmalsausprägungen des Beispiel-Merkmalsraumes 4.5 gezeigt.

```
1  <FeatureSpace>
2          <Feature id="stddev">
3                  <Channel id="xaccel" />
4                  <Channel id="yaccel" />
5          </Feature>
6          <Feature id="dtw">
7                  <Channel id="xaccel" />
8                  <Channel id="yaccel" />
9          </Feature>
10 </FeatureSpace>
```

Quellcode 4.5: Beispiel einer Beschreibung eines Merkmalsraumes im XML-Format. Definiert wird ein Merkmalsraum der auf zwei Merkmalen *stddev* und *dtw* basiert. Anhang 1.7 liefert die XSD dieses Datentyps

Es können zwei Typen von Merkmalen unterschieden werden: Merkmale, die auf einem

einzelnen Kanal erhoben werden und Merkmale die Zusammenhänge mehrerer Kanäle beschreiben. Diese Betrachtung wird von dem System selbst nicht berücksichtigt, sondern den Entwicklern der Merkmale überlassen.
Während der Datentyp des Merkmalsraums eine Beschreibung der zu erfassenden Merkmale definiert, muss ein weiterer Datentyp eine Möglichkeit bieten, deren Ausprägungen zu übertragen. Diese Ausprägungen der Merkmale werden durch ein Element `Characteristics` beschrieben. Dieses enthält

- eine Liste von Kind-Elementen des Typs `Characteristic`
- die globale Kennung `id` der merkmalerhebenden Funktion
- den Wert `value` des Mermals
- und die Liste von Kanälen `Channel` pro Merkmal, welche die Basis der Berechnung angibt.

Quellcode 4.6 zeigt ein Beispiel eines solchen Elements.

```
<Characteristics>
     <Characteristic id="stddev" value="1.0">
       <Channel>xaccel</Channel>
     </Characteristic>
     <Characteristic id="stddev" value="1.0">
       <Channel>yaccel</Channel>
     </Characteristic>
     <Characteristic id="dtw" value="1.0">
       <Channel>xaccel</Channel><Channel>xaccel</Channel>
     </Characteristic>
</Characteristics>
```

Quellcode 4.6: Beispiel einer Beschreibung der Werte einer Merkmalsberechnung im XML-Format. Anhang 1.20 liefert die XSD dieses Datentyps

4.2.3 Modelle und Konfigurationen

Einige Funktionen, insbesondere die Klassifizierungsalgorithmen oder Modelle, benötigen eine Konfiguration, in der die Parameter des Algorithmus festgelegt werden. Diese Konfigurationen von Parametern bilden sowohl die Schnittstelle zwischen verschiedenen Entwicklersystemen, als auch die Verbindung von Entwicklern und Anwendern. Insbesondere für die Anwender ist es wichtig, eine einfache Möglichkeit zu erhalten, die Parameter einer Konfiguration zu setzten und somit die Algorithmen an die Problemstellung anzupassen.
Die Anwendung stellt einige Grundelemente bereit, mit denen das Erstellen einer Konfiguration ermöglicht wird. Diese Elemente unterstützen nur einige Basistypen von Werten und erlauben keine Definition komplexerer Datentypen.

Basis-Elemente

Eine Konfiguration setzt sich aus Basis-Elementen zusammen, die es den Entwicklern erlauben, die benötigten Parameter einer Konfiguration zu beschreiben. Ein Benutzter soll diese dann nur noch mit gültigen Werten belegen können. Damit dies erreicht wird, werden alle Elemente dieses Typs mit den Universalattributen ausgestattet.

`Integer`	für ganzzahlige numerische Werte
`Float`	für Fließkommazahlen
`String`	für kurze textuelle Beschreibungen
`Text`	für längere Texte
`Boolean`	für das optionale setzten eines festen Wertes

Zudem können die nummerischen Typen `Float` und `Integer` drei weitere optionale Attribute enthalten:

`min`	für den kleinsten Wert den ein Parameter annehmen kann
`max`	für die Obergrenze eines Wertes
`step`	für bestimmte Werte innerhalb eines festgelegten Intervalls

Auswahlelemente

Für einige Konfigurationen ist es unter Umständen notwendig, dem Nutzer eine Auswahl aus einer Menge von möglichen Werten zu bieten. Ein solcher Parameter wird durch den Datentyp `Choice` definiert. Dieser Datentype enthält verschiedene Parameter-Elemente, von denen *eines* durch den Nutzer ausgewählt wird und dessen Identifikation als Wert des Auswahl-Parameters gesetzt wird. Analog definiert der Datentyp `Selection` einen Parameter der als Rückgabe eine Untermenge einer vordefinierten Menge von Parametern liefert. Eine solche Konfiguration wird in Quellcode 4.7 gezeigt. Eine gültige Belegung für diese Konfiguration zeigt 4.8

```
1 <Config id="SVM" name="SVM" desc="Support␣Vector␣Machine">
2   <Float id="C" name="TradeOff" min="0.0" max="1.0" desc="trade..."/>
3
4   <Choice name="kernel␣type" id="Kernel" desc="Type␣of...">
5       <String id="linear" name="Linear" desc="Linear␣Kernel" />
6       <String id="poly" name="polynomial" desc="Polynomial␣Kernel" />
7   </Choice>
8 </Config>
```

Quellcode 4.7: Beispiel einer Konfigurationsbeschreibung für eine SVM. Es werden zwei Parameter definiert: `C` für den Trade-Off und `kernel` für den Typ des Kernels. Letzter kann entweder den Wert `linear` oder `polynomial` annehmen. Anhang 1.18 liefert die XSD dieses Datentyps

```
<ParameterSet id="SVM" name="Test␣SVM" desc="Example...">
  <Value id="C">0.5</Value>
  <Value id="Kernel">linear</Value>
</ParameterSet>
```

Quellcode 4.8: Beispiel einer Konfiguration für eine SVM. Die Beschreibung wird in Quellcode 4.7 und Anhang 2.32 liefert die XSD dieses Datentyps

4.3 Funktionen

Da die Systeme, auf denen Funktionen implementiert sind, die nicht homogen sein müssen, kann ein Austausch dieser Funktionen selbst nicht stattfinden. Jedoch werden eine Kennung zur Identifikation, der Name und die Beschreibung übertragen, um eine Funktion auf der Master-Komponente zu registrieren. Für diese Registrierung werden zwei Kennungen benötigt:

1. **Die lokale Kennung** stellt die Kennung innerhalb des implementierenden Systems dar. Ihre Notwendigkeit liegt darin begründet, dass ein solches System eine Vielzahl von Funktionen implementieren kann und bei einem Aufruf durch den Master diesen lokal an die entsprechende Funktion weiter leiten muss.

2. **Die globale Kennung** wird bei einer Funktionsregistrierung vom Master vergeben und ist systemweit eindeutig. Diese Kennung wird bei Aufrufen verwendet, welche sich von einem Client an den Master richten.

Für die Registrierung einer Funktion werden von dem Client neben den Universalattributen, die lokale Kennung, die Adresse und der Port des Teilsystems an den Master übertragen. Der Master antwortet daraufhin mit der globalen Kennung der Funktion.

```
<Register host="127.0.0.1" port="9876">
 <Filter id="transform" name="test␣Filter"desc="..." />
 <Feature id="stddev" name="standard␣deviation" desc="..."/>
 <Feature id="mean" name="mean" desc="..."/>
 <Class id="knn" name="KNN" desc="K-Nearest...">
  <Integer id="k" min="1" name="K" desc="..." >3</Integer>
 </Class>
</Register>
```

Quellcode 4.9: Beispiel einer System-Registrierung. Die Definition wird in Anhang 3.44 durch den Datentypen `SystemRegistration` gegeben

4.3.1 System-Registrierung

Damit ein Teilsystem die Funktionalität des Gesamtsystems, wie in Abschnitt 3.2.2 angegeben, erweitern kann, muss dieses System registriert werden. Um dies zu erreichen verbindet

```
<Register host="127.0.0.1" port="9876">
 <Filter id="transform" name="test␣Filter"desc="..." />
 <Feature id="stddev" name="standard␣deviation" desc="..."/>
 <Feature id="mean" name="mean" desc="..."/>
 <Class id="knn" name="KNN" desc="K-Nearest...">
  <Integer id="k" min="1" name="K" desc="..." >3</Integer>
 </Class>
</Register>
```

Quellcode 4.10: Beispiel einer System-Registrierungsantwort. Definiert wird dieser Datentyp in Anhang 3.47 unter der Bezeichnung `RegistrationResponse`

sich ein Teilsystem mit dem Master und überträgt neben der Adresse, `host` und `port`, unter der das System erreichbar ist, die Basisdaten der Funktionen, die es zur Verfügung stellt.
Der Master antwortet auf diese Registrierung mit einer Liste der registrierten Funktionen, in der sowohl die lokale `cid` als auch die globale Kennung `id` übertragen werden. Zusätzlich wird über ein Attribut `bypass` ein Port angegeben, auf dem das Teilsystem Statusmeldungen an den Master übertragen kann. Die Protokollauszüge Quellcode 4.9 und 4.10 zeigen eine solche Registrierung und eine entsprechende Antwort.

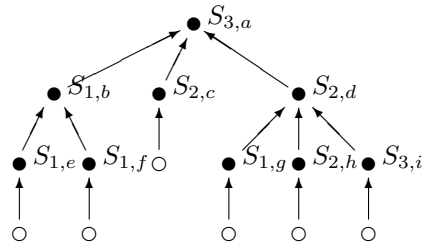
4.3.2 Datensatz-Registrierung

Damit das Gesamtsystem arbeiten kann, sind zunächst Daten notwendig, auf denen die Klassifizierungsalgorithmen arbeiten können. Um diese Daten in das System aufnehmen zu können, ist als erste Möglichkeit das Laden von Daten vorgesehen. Der hierfür benötige Befehl `Upload` entspricht in seiner Definition dem Datentyp `DataObject`. Das Attribut `id` dieser Elements entspricht bei diesem Vorgang einer Kennung, die nur für das Erstellen einer Antwort benötigt wird. Da in der Regel mehrere Datenobjekte innerhalb einer Verbindung an den Master übertragen werden, nutzt dieser die Kennung, um in seiner Antwort eine Verknüpfung zwischen den Datenobjekten und der globalen Kennung herzustellen.
Die Übermittlung der Daten auf diesem Weg ist jedoch in vielen Fällen nicht wünschenswert, da es sich auf Grund des Kodierungsverfahrens (s. Abschnitt 4.2.1) und der zu übertragenden Datenmenge um einen sehr zeitaufwendigen Prozess handelt. Damit dieser Prozess abgekürzt werden kann, ist als zweite Möglichkeit ein direkter Zugriff auf das Dateisystem vorgesehen, in dem es dem Anwender gestattet wird, die Daten in einem Arbeitsverzeichnis der Master-Komponente abzulegen. Pfad und Dateiname des Datensatzes bilden in diesem Fall die Kennung und je nach Format der Datensätze übernimmt der Master Namen und Beschreibungen der Datensätze.

4.3.3 Filterung

Für die Vorverarbeitung der Daten werden durch das System sogenannte Filterfunktionen bereit gestellt. Diese Filterfunktionen können geschachtelt werden und enthalten dadurch sowohl Datensegmente als auch weitere Filterelemente und Konfigurationsparameter. Quell-

code 4.11 zeigt eine solche Filterungsaufforderung. Der Prozess des Datenfilterns ist durch das Verschachteln sehr aufwendig: Wird das System angewiesen, Daten auf diesem Weg zu verarbeiten, werden die einzelnen Stufen des Filterungsprozesses sequenziell bearbeitet. Die Bearbeitung beginnt in der tiefsten Ebene, indem der Master die angeforderten Daten an das entsprechende Teilsystem überträgt und eine Antwort abwartet. Ist die Berechnung durch das Teilsystem beendet, werden die Daten wieder an den Master übertragen und von dort aus an das nächste Teilsystem übertragen. Durch das Verschachteln kann eine Filteranfrage als Baum aufgefasst werden. Damit diese Übertragungen zwischen den Systemen so gering wie möglich gehalten wird, sorgt die Master-Komponente dafür, dass Sequenzen, die von einem einzelnen Teilsystem berechnet werden können, innerhalb einer Verbindung übertragen werden. Hierfür wird die Baumstruktur der Filter per Tiefensuche durchlaufen. Wird bei dieser Suche ein Teilbaum gefunden dessen Elemente, abgesehen von Datensegmenten, nur Filterelemente enthält, die sich auf ein Teilsystem beziehen, wird der gesamte Teilbaum an das entsprechende System übertragen und bearbeitet. Ein Beispiel dieser Optimierung zeigt Abbildung 4.1.



System	S_1	S_2	S_3
Stufe I	b(e, f)	c	i
	g		
Stufe II		d(g, h, i)	
Stufe III			a(b, c, d)

Abbildung 4.1: Ein Filterbaum mit Anfragen auf drei Systemen S_1 bis S_3. ∘ markiert Datensegmente und • die Filterfunktionen a bis i. Wird die Anfrage an das System übermittelt wird sie zunächst in vier Teilanfragen der Teilbäume b, c, g und i zerlegt. Sind die Anfragen g und i beantwortet wird der Teilbaum d durch das System S_2 bearbeitet. Nach Beendigung der Teilbäume b, c und d wird die Wurzel a bearbeitet.

Eine weitere Struktur des Filterbaumes, die eine Optimierung erlaubt, sind Teilbäume, welche von der Wurzel ausgehende Pfade enthalten, die mit Elementen beginnen, die auf dem gleichen System, wie das Wurzelelement bearbeitet werden. In diesem Fall werden zunächst alle Elemente bearbeitet die nicht von dem entsprechenden Teilsystem stammen. Nachdem diese bearbeitet und das Ergebnis an die Master-Komponente übertragen wurde, werden die bearbeiteten Segmente und die verbliebenen Pfade als Teilanfrage an das Teilsystem übertragen.

4.3.4 Merkmalsextraktion

Die Klassifizierung von Daten findet nicht auf Roh- oder gefilterten Daten statt, sondern basiert auf Merkmalen die diesen Daten entnommen werden. Um festzulegen, welche Merkmale den Daten entnommen werden sollen, wird ein Element definiert, das einen Merkmalsraum beschreibt. Jedes Merkmalsfunktion dieses Raumes wird auf eine Menge von Datenkanälen angewendet.

```
<Register host="127.0.0.1" port="9876">
 <Filter id="transform" name="test␣Filter"desc="..." />
 <Feature id="stddev" name="standard␣deviation" desc="..."/>
 <Feature id="mean" name="mean" desc="..."/>
 <Class id="knn" name="KNN" desc="K-Nearest...">
  <Integer id="k" min="1" name="K" desc="..." >3</Integer>
 </Class>
</Register>
```

Quellcode 4.11: Beispiel einer Filteraufforderung. Definiert wird dieser Datentyp in Anhang 2.35

Für die Erhebung der Ausprägungen der Merkmale wird die Anforderung der Transformation der Daten in den Merkmalsraum von einem Client an den Master übertragen. Der Master unterteilt den Merkmalsraum in Teilräume, welche jeweils von einem Teilsystem bearbeitet werden können, und leitet diese Definitionen an die entsprechenden Komponenten weiter. Nach der Bearbeitung aller Teilräume übertragen die Teilsysteme ihre Ergebnisse an den Master, der sie zu dem Ergebnis des gesamten Merkmalsraumes zusammensetzt. Die Nutzung dieser Methode liegt darin begründet, dass so für jede Transformation nur ein Aufruf pro Teilsystem erfolgen muss.

```
<FeatureExtraction>
    <FeatureSpace>
      <Feature id="dtw">
        <Channel id="xaccel"/>
        <Channel id="yaccel"/>
      </Feature>
    </FeatureSpace>

    <Segment>
      <Channel id="xaccel" name="X-Axis" samplingrate="10"
               unit="m/s^2" desc="lateral␣acceleration"
               type="float" encoding="base64" >
         AAAAABXvwz7zBDU/XoNsPwAAgD9eg2w/8wQ1PxXvwz4=
      </Channel>

      <Channel id="yaccel" name="Y-Axis" samplingrate="10"
           unit="m/s^2" desc="anteroposterior␣acceleration"
           type="float" encoding="base64" >
       AACAP16DbD/zBDU/Fe/DPjIxjSQV78O+8wQ1v16DbL8=
      </Channel>
    </Segment>
</FeatureExtraction>
```

Quellcode 4.12: Anfrage einer Merkmalsberechnung. Definiert wird dieser Datentyp in Anhang 2.39

4.4 Arbeitsabläufe

Durch die Formalisierung der Datentypen und Funktionen ergeben sich Arbeitsschritte, die das System bearbeiten soll. Diese Arbeitsschritte stellen die Protokollebene dar, auf der ein Nutzer mit dem System kommuniziert und somit auch den Funktionsumfang, den eine Anwendung für einen Nutzer aufbereiten muss, um eine attraktive Arbeitsumgebung zu schaffen.
Bei den Arbeitsabläufen handelt es sich in der Regel um Prozesse, die viel Zeit in Anspruch nehmen. Daher wird die Aufforderung einen dieser Abläufe zu beginnen, lediglich mit einer Kennung `id` der Auftrages quittiert. Diese optional um einen Text `message` und eine Fortschrittswert `status` erweitert werden. Ein Beispiel dieser Definition gibt der XML-Auszug 4.13.
Durch das Senden eines leeren `Job`-Elements kann ein Teilsystem zu jeder Zeit den aktuellen Stand eines Auftrags erfragen. Ist ein Auftrag beendet, liefert diese Anfrage das Ergebnis der Berechnungen.

```
1 <JobStatus id="Analyse1" message="Extracting␣Features" status="99.0"/>
```

Quellcode 4.13: Auftragsbestätigung. Definiert wird dieser Datentyp in Anhang 3.48

4.4.1 Training

Das Training eines Modells zur Klassifizierung stellt der ersten wichtigen Arbeitsschritt dar. Hierfür muss zunächst eine Modell angegeben werden, das trainiert werden soll. Dafür muss mindestens eine Konfiguration von Parametern zusammen mit der Kennung des Modells, in einem Element `Model` angegeben werden, dessen Attribut `id` das Modell angibt.
Die zweite notwendige Information ist der Merkmalsraum, in dem das Training erfolgen soll. Diese werden durch ein Elemente `FeatureSpace` angegeben und entspricht der Definition 1.7 im Anhang. Um den Trainingsdaten Klassen zuweisen zu können, müssen mindesten zwei `Label`-Element angegeben werden, die durch das Attribut `id` auf die Klasseninformationen verweisen, die durch die Master-Komponente verwaltet werden.
Den Klassen-Definitionen folgt eine Liste von Daten, die für das Training genutzt werden sollen. Eine Definition eines solchen Trainings wird im XML Auszug 4.14 gezeigt. Mitunter werden mehrere Klassifizierungen auf der Basis des gleichen trainierten Modells vorgenommen. Deshalb muss das System eine Möglichkeit bieten, diese Tainierten Modelle zu speichern. Da die Klassifizierungsalgorithmen auf anderen Systemen implementiert sind, wird nicht das trainierte Modell selbst, sondern die Konfiguration des Trainings durch den Master gespeichert.

4.4.2 Klassifizierung

Neben den System-, Datensatz- und Funktions-Registrierungen bildet das Klassifizieren von Daten eine weitere Kategorie von Arbeitsschritten, die durch das System bearbeitet werden.

```
<Training>
  <Model id="SVM">
    <Value id="c">0.1</Value>
  </Model>
  <FeatureSpace>
        <Feature id="mean">
                <Channel id="xaccel" />
                <Channel id="yaccel" />
        </Feature>
  </FeatureSpace>
  <Label id="Walk0"/>
  <Label id="Sit0"/>
  <Segment>....</Segment>
  <Segment>....</Segment>
  <Segment>....</Segment>
</Training>
```

Quellcode 4.14: Beispiel für das Training einer SVM. Im Anhang 2.41 wird die XSD dieses Datentyps angegeben

Im Allgemeinen besteht die Aufgabe der Klassifizierung darin, Datensätze und Segmente, deren Kategorien nicht bekannt sind, eben diese zuzuordnen. Die folgenden Abschnitte beschreiben drei Verfahren die genau dies leisten.

Manuelles Klassifizieren

Da das überwachte Lernen nicht ohne bereits Klassifizierte Daten auskommt, muss das System eine Schnittstelle bieten, um Rohdaten klassifizieren zu können. Hierfür wir einem Nutzer die Möglichkeit geboten, Datensätze und Segmente durch ihre Kennung anzugeben, die Klassifiziert werden sollen. Zudem wird eine Blockgröße angegeben, die eine maximale Dauer der zu klassifizieren Segmente bestimmt. Das System schickt dann ein Segment an die Anwendung des Nutzers, welche das Segment darstellen kann und die Möglichkeit bietet, Markierung festzulegen. Diese Markierungen werden dann an die Master-Komponente übertragen und gespeichert. Dach wird das nächste Segment an den Nutzer Übertragen. Dieser Kreislauf wird solange wiederholt, bis alle ausgewählten Datensätze durch den Nutzen mit Markierungen aufbereitet wurden.

Teilautomatisches Klassifizieren

Angelehnt an das aktive Lernen bietet das System eine Unterstützung für das Klassifizieren von unklassifizierten Daten. Die Übertragung entspricht der des manuellen Klassifizierens, jedoch mit dem Unterschied, dass der Nutzen ein Modell und einen Merkmalsraum angibt. Mit jeder manuellen Klassenzuweisung wird das Training des Modells erweitert und dem Nutzer ein Klassifizierungsvorschlag für das nächste Segment übermittelt. Der Hintergedanke hierbei ist es, dass das zugrunde liegende Modell nur die Segmente an den Nutzer überträgt,

die als unsicher gelten und somit den Aufwand für den Nutzer möglichst gering zu halten. Die Definition des Anfrage-Elements erweitert sich sich somit um ein Attribut `model`, welches die Kennung eines Klassifizierungsalgorithmus trägt. Die Parameter des Algorithmus, die Beschreibung eines Merkmalsraumes und die Vorverarbeitung der Daten, die für das Training des Modells notwendig sind, werden in die Definition der Kindelemente mit aufgenommen. Der Datentyp `ActiveSession` ist im Anhang 2.42 definiert und ein Beispiel für den Beginn einer solchen Sitzung wird im XML-Auszug 4.15 gezeigt.

```
<ActiveSession model="SVM" name="Test-Session" desc="...">
        <Value id="c">0.1</Value>
        <Value id="kernel">linear</Value>
        <FeatureSpace>
                <Feature id="stddev">
                        <Channel id="xaccel"/>
                        <Channel id="yaccel"/>
                </Feature>
                <Feature id="dtw">
                        <Channel id="xaccel"/>
                        <Channel id="yaccel"/>
                </Feature>
        </FeatureSpace>
        <Filter id="normalize">
                <Channel id="xaccel"/>
                <Channel id="yaccel"/>
        </Filter>
        <DataObject id="DS01.01" />
</ActiveSession>
```

Quellcode 4.15: Eröffnen eine computergestützten Klassifizierung. Das unterstützende Modell ist eine lineare SVM die auf einem dreidimensionalen Merkmalsraum arbeitet. Im Anhang 2.42 wird die XSD dieses Datentyps angegeben

Vollautomatisches Klassifizieren

Das automatische Klassifizieren bietet schließlich ein Verfahren, in dem die Klassenzuweisung voll ständig durch das System erfolgt. Hierfür werden die folgenden Informationen durch den Befehl Klassifizierungs-Befehl `Classification` (s. Anhang 2.40) an das System übertragen:

- Das Modell welches zu Klassifizieren genutzt werden soll
- Die Parameter mit denen dieses Model konfiguriert wird
- Der Merkmalsraum in den die Daten überführt werden müssen
- Die Daten mit denen das Modell trainiert wird
- Eine Liste von Daten die klassifiziert werden sollen

Das Resultat dieses Befehls ist eine segmentweise Auflistung der Vorhersagen durch den Klassifizierungsalgorithmus.

4.4.3 Analyse

Im Gegensatz zur vollautomatischen Klassifizierung von Daten arbeitet die Analyse mit zwei klassifizierten Datensätzen. Einem Trainingsdatensatz und einem Testdatensatz. Wie die Namen bereits implizieren, wird der erste Datensatz genutzt, um ein Modell zu trainieren und die Klassen des Testdatensatzes durch das Modell vorhergesagt. Da die Klassen des Testdatensatzes bekannt sind, kann somit erkannte werden, an welchen Daten das Modell bei einer Klassifizierung scheitert.
Das Ergebnis eines Analyse-Auftrags (s. Anhang 2.38) wird sowohl als Konfusionsmatrix als auch als Einzelauflistung der Vorhersagen des Modells, übertragen. In der XML-Repräsentation der Konfusionmatrix (s. Anhang 1.27) werden die Zeilen der Testklassen durch `Target`-Element (s. Anhang 1.29) beschrieben, die durch das Attribut `label` eine Klasse der Testdaten referenzieren. Die Kind-Elemente dieser Elemente bilden die in Anhang 1.28 definierten `Prediction`-Elemente, die durch das `label`-Attribute eine Klasse der Trainingsdaten angeben.

4.4.4 Kreuzvalidierung

Die Kreuzvalidierung ist ein Verfahren für das Bestimmen der Güte eines Modells. Für diese Analyse wird ein Testdatensatz mit bekannten Klassen und ein parametrisiertes Modell benötigt. Zudem muss die Anzahl der Faltungen innerhalb der Analyse angegeben werden. Das Befehls-Element `Crossvalidation` (s. Anhang 2.30) referenziert, mit Hilfe des Attributs `model`, einen Klassifizierungsalgorithmus und über die Kindelemente des Typs `ParameterSet` werden die Parameter der Modells festgelegt. Das Attribut `folds` enthält die Anzahl der Faltungen der statifizierten Kreuzvalidierung.
Das Ergebnis dieses Befehls besteht auch einem einfachen Element `Accuracy`, das durch die beiden Attribute `id` und `value` die Berechnungskennung und die durch Kreuzvalidierung ermittelte balancierte Trennschäfte angibt.

4.5 Fehlermeldungen

Bei der Verarbeitung von Befehlen kann es passieren, dass Fehler auftreten. Ob diese durch falsche Eingaben von Nutzern, Fehler in der Berechnungen, das Versagen der Infrastruktur oder durch andere Ursachen bedingt sind, muss in den Schnittstellendefinitionen nicht unterschieden werden. Da diese dennoch entstehen, soll eine Benachrichtigung, soweit es möglich ist, an einen Nutzer der Systems geleitet werden. Hierfür wird im Anhang 3.49 ein `Exception`-Element definiert, dass neben einer Fehlerkennung `id` auch eine Fehlermeldung `message` enthält. Diese Elemente werden genutzt, um eine Systemübergreifende Fehlerbehandlung zu ermöglichen.

KAPITEL 5

Auswertung

In der bisherigen Arbeit wurde die Struktur eines verteilten Systems zur Klassifizierung multivariater Zeitreihen entworfen. Auf Grund einer Anforderungsanalyse am Max–Planck–Institut für Bildungsforschung, sind Implementierungen dieses Systems entstanden. Diese umfassen eine Master- und eine Entwickler-Komponente, welche mittels Python [2] realisiert wurden. Zudem wurden Schnittstellendefinition in $Java^{TM}$.
Im Folgenden wird der Aufbau des Systems beschrieben. Danach wird eine Studie vorgestellt, in der dieses System benutzt wird, um die Datenerhebungen auszuwerten. Abschließend ein Anwendungsfall beschrieben, der aufzeigt welchen Nutzen das System bietet.

5.1 Systembeschreibung

Für die Realisierung des Entwurfs ist zunächst die Protokollebene implementiert worden. Diese setzt sich aus zwei Teilen zusammen:

- Die Kommunikationsschnittstellen der Master-Komponente. Dies umfasst XML-Parser für die eingehenden Anfragen und das generieren der entsprechende Antworten
- Die Kommunikation der Teilsysteme mit der Master-Komponente

Die Schnittstelle zwischen den Teilsystemen und der Master-Komponente ist in in zwei Sprachen verfasst worden, zum einen in der Programmiersprache *Python* zum anderen in Java. Die Python-Schnittstelle bildet die Grundlage für Anbinden von Teilsystemen, die das Gesamtsystem erweitern. Die Java-Schnittstelle bildet die Basis für eine graphische Benutzeroberfläche (GUI), die Psychologen einen Zugang zu dem System zu gewährleistet.
Für das Testen des Systems sind die Aufgaben der Klassifizierung auf drei Systeme verteilt worden. Das erste Teilsystem stellt die Klassifizierungsalgorithmen zur Verfügung. Hier ist die Python-Schnittstelle genutzt worden, um um das *PyMVPA*-Framework [1] in das System

mit aufzunehmen. Das institutsinternes Klassifizierungs-Framework *PyFreshFruit* stellt das zweite Teilsystem und übernimmt die Transformation der Rohdaten in den Merkmalsraum. Das letzte Teilsystem übernimmt die Vorverarbeitung und Analyse der Daten. Somit stellt dieses den ersten und den letzten Schritt der Datenverarbeitung dar.
Die graphische Oberfläche bietet zwar die Möglichkeit die Daten in das System aufzuneh-

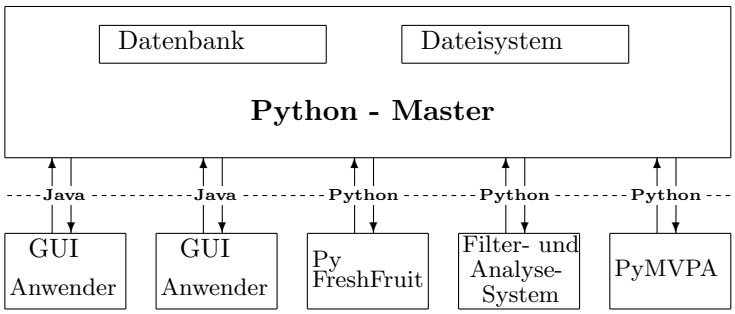


Abbildung 5.1: Aufbau des Systems

men, aber auf Grund der Übertragungsgeschwindigkeit und des Volumens der Daten, ist Master-Komponente so konfiguriert worden, dass sie direkt auf dem Dateisystem arbeitet, auf dem die erfassten Daten abgelegt wurden.

5.2 Anwendungsbeispiel

Diese Realisierung des Entwurfs ist genutzt worden, um die Daten einer Studie zu analysieren und soll an dieser Stelle genutzt werden, um die Anwendung des Entwurfs zu beschreiben und die Fähigkeiten des System demonstrieren. Grundlage hierfür bietet eine am Max–Planck–Institut für Bildungsforschung erhobene Studie der Arbeitsgruppe *Affekt im Lebensverlauf* unter M. Riedigers. Ziel der Studie ist es heraus zu finden, ob die Fähigkeit, kognitive und motorische Aufgaben gleichzeitig zu lösen, im Alter schwerer fällt, als in der Jugend.
Um diese Hypothese zu prüfen, wurden zunächst neun Bewegungskategorien festgelegt die es zu erfassen galt. Diese wurden dann, wie in Tabelle 5.1 angegeben, zu vier Bewegungsprofilen zusammengefasst. Zum Erfassen der Profile sind die Bewegungen der Probanden durch drei Sensorengruppen aufgezeichnet worden. Zwei dieser Gruppen zeichneten die Beschleunigungskräfte in den Richtungen sagittal (X), transversal (Y) und longitudinal (Z) auf, eine die Orientierung (O) des Oberschenkels in der Sagittalebene.
Die erste Beschleunigungssensorik wurde am Brustbein befestigt, die zwei als Tasche in der Umgebung des Bauches getragen, die dritte am Oberschenkel befestigt. Bereits während der ersten Datenerhebung haben sich die nicht befestigte Sensoren als untauglich erwiesen, da diese nicht nur individuell unterschiedlich getragen wurden, sondern auch währen der Bewegungen ihre Position änderten. Deshalb wurden die Daten nur mit den verbleibenden vier Messgrößen analysiert.
Die Studie setzte sich aus 92 Teilnehmern im Alter zwischen 14 und 84 Jahren zusammen.

Profil	Unterkategorie	Beschreibung	Verteilung
Gehen & Treppe	normales Gehen	Die Person geht	51,0%
	schnelles Gehen	Die Person läuft	
	Treppe rauf	Eine Treppe nach oben gehen	
	Treppe runter	Eine Treppe nach unten gehen	
Stehen	Stehen	Der Proband steht	12,2%
Sitzen	Sitzen	Die Person sitzt	12,2%
Liegen	Links Liegen	Auf der linken Seite liegen	24,6%
	Rechts Liegen	Auf der rechten Seite liegen	
	Rücken Liegen	Auf dem Rücken liegen	

Tabelle 5.1: Bewegungsprofile der Feldstudie

Einen Überblick über die Altersgruppen und deren Verteilung innerhalb der Studie bietet die Tabelle 5.2.

Gruppe	Personen	Durchschnittsalter	m/w
14 – 18	10	17.0	5/5
18 – 30	18	22.5	7/11
30 – 40	16	33.7	8/8
40 – 50	14	44.1	5/9
50 – 60	12	55.1	5/7
60 – 70	15	65.3	6/9
70 – 84	7	75.1	5/2
14 – 84	92	42.4 ($\sigma = 19.0$)	41/51

Tabelle 5.2: Stichprobenmerkmale der Probanden

5.2.1 Modelle und Merkmalsräume

Die Auswahl der Modelle beschränkt sich auf die in dieser Arbeit vorgestellten Algorithmen k-Nearest-Neighbors (KNN), Support Vector Machine (SVM) und die logistische Regression. Für die ersten beiden Modelle müssen die Parameter k bzw. c gefunden werden, für die eine bestmögliche Trennung der Klassen geboten wird. Da die logistische Regression ein parameterloses Modell bildet wird dieses lediglich auf den verschiedenen Merkmalsräumen getestet. Anhand der Daten lässt sich zunächst keine Aussage darüber treffen, in welchem Merkmalsraum eine bestmögliche Trennung der Klassen stattfinden kann. Deshalb werden die Daten der Studie in sechs Merkmalsräume (s. Tabelle 5.3) transformiert und in jeden dieser, eine Serie von Parametern der Modelle getestet. Diese Merkmale sind so gewählt, dass die eine möglichst simple Beschreibung der Daten bieten und somit dem Sparsamkeitsprinzip (vgl. [7, S. 464f]) entsprechen, nachdem ein einfaches Modell einem komplexeren vorzuziehen ist.
In den ersten drei Testserien werden nur Merkmale genutzt, die von den Kanälen X, Y und

Kanäle	X	Y	Z	O	XY	XZ	YZ	XO	YO	ZO
1. Serie	Sa.	Sa.	Sa.	—	—	—	—	—	—	—
	Mw.	Mw.	Mw.							
2. Serie	Sa.	Sa.	Sa.	—	—	—	—	—	—	—
	Mw.	Mw.	Mw.							
	Energie	Energie	Energie							
3. Serie	Sa.	Sa.	Sa.	—	DTW	DTW	DTW	—	—	—
	Mw.	Mw.	Mw.							
	Energie	Energie	Energie							
4. Serie	Sa.	Sa.	Sa.	Sa.	—	—	—	—	—	—
	Mw.	Mw.	Mw.	Mw.						
5. Serie	Sa.	Sa.	Sa.	Sa.	—	—	—	—	—	—
	Mw.	Mw.	Mw.	Mw.						
	Energie	Energie	Energie	Energie						
6. Serie	Sa.	Sa.	Sa.	Sa.	DTW	DTW	DTW	DTW	DTW	DTW
	Mw.	Mw.	Mw.	Mw.						
	Energie	Energie	Energie	Energie						

Tabelle 5.3: Merkmalsräume der Testserien

Z erhoben werden, um die Modelle einfach zuhalten. In der ersten Serie wird die Trennbarkeit der Klassen auf den Merkmalen *Standardabweichung* und *Mittelwert* getestet. Die Merkmale werden gewählt, da davon aufzugehen ist, dass sie bereits Auskunft über die Lage einer Person geben können. Aufrechte Positionen lassen auf den Achsen andere Mittelwerte erwarten als liegende. Die Standardabweichung sollte in Profilen, in denen sich die Probanden bewegen (z.B. *Gehen*) höher ausfallen, als in Profilen, in denen wenig Bewegung zu erwarten ist (z.B. *Stehen*). In der zweiten Testserie wird das Merkmal *Energie* mit aufgenommen. Dieses Merkmal wird dafür genutzt die Geschwindigkeiten der einzelnen Richtungen zu bestimmen. Die Zeitverschiebung DTW zwischen den Kanälen wird in der dritten Serie gemessen. Da diese ein Maß für die Ähnlichkeit zweier Messreihen bietet, kann durch sie ein Zusammenhang zwischen Bewegungsabläufen bestimmt werden.
Nach Auswertung dieser drei Testserien werden diese um die Merkmale der Oberschenkelorientierung O erweitert, um zu erfahren, ob sich die Ergebnisse signifikant verbessern. Kann eine sichere Klassifizierung ohne diesen Sensor erfolgen, wäre es möglich die Klassifizierung beispielsweise durch Mobiltelefone zu erfassen. Dadurch könnten unter anderem weitere Studien kostengünstig und flächendeckend erhoben werden.

5.2.2 Parameter

Für das Finden der optimalen Parameter der Algorithmen, werden die erhobenen Daten *normalisiert*, anhand der Klassen in Segmente aufgeteilt und jedes diese Segmente in $1s$-Blöcke zerlegt. Jedes dieser Segmente wird dann in den Merkmalsraum überführt und so generierten Merkmalsverktoren danach in zwei Datensätze aufgeteilt. Die Verteilung der Klassen innerhalb dieser Datensätze entspricht dabei der des Ursprungsdatensatzes (s. Tabelle 5.1).

Der erste Datensatz enthält 2/3 der Segmente und wird genutzt um die Güte der parametrisierten Modelle durch eine 10-fache stratifizierte Kreuzvalidierung zu bestimmen. Der zweite Datensatz, der aus 1/3 der Daten zusammengesetzt ist, dient dann zum Testen der Parameter der Modelle. Das entsprechende Modell wird auf dem ersten Datensatz trainiert und genutzt um die Klassen des zweiten Datensatzes vorherzusagen.

5.2.3 Ergebnisse

Für die Suche nach einem geeigneten Parameter k für der k-Nearest-Neighbors-Algorithmus werden die ungeraden Werte zwischen 0 bis 10 gewählt. Da dieses Verfahren eine Klassifizierung anhand der im Merkmalsraum benachbarten Elemente vornimmt, können große Werte dazu führen, dass die Nachbarschaftsbeziehung auf andere Klassen ausgedehnt wird, die mehr Repräsentanten umfasst, als die korrekte Klasse. Wird der Parameter zu klein gewählt, können Ausreißer das Ergebnis zu stark beeinflussen und gerade Werte können zu einem Unentschieden bei der Klassenabstimmung führen.

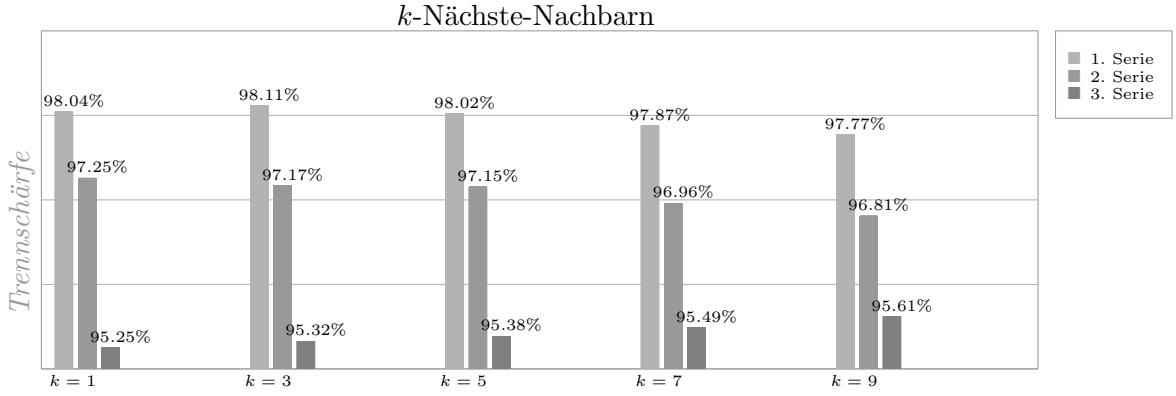


Abbildung 5.2: Trennschärfe der Bewegungsprofile durch den KNN-Algorithums in den Testserien 1–3 in Abhängigkeit des Parameters k

Das beste Klassifizierungsergebnis des KNN-Algorithmus im ersten Merkmalraum liefert der Parameter $k = 3$ mit einer balancierten Güte von $98,11\%$. In der zweiten Testserie werden bereits schlechtere Ergebnisse erzielt als im vorherigen Testlauf erzielt. Das Optimum für KNN liegt hier bei $k = 1$ mit einer Trennschärfe von $97,25\%$. Die Ergebnisse er dritten Serie fallen insgesamt noch schlechter aus: Der beste Wert liegt hier nur noch bei $96,61$ für $k = 9$. Eine Übersicht über die Ergebnisse, die durch die Veränderung des Parameters k in den ersten drei Testserien erzielt wurden, bietet die Abbildung 5.2.

Durch die Support Vector Machine wird versucht ein lineares Modell zur Beschreibung der Klassen zu finden. Das dieses Verfahren zunächst nur zwei Klassen von einander Tren-

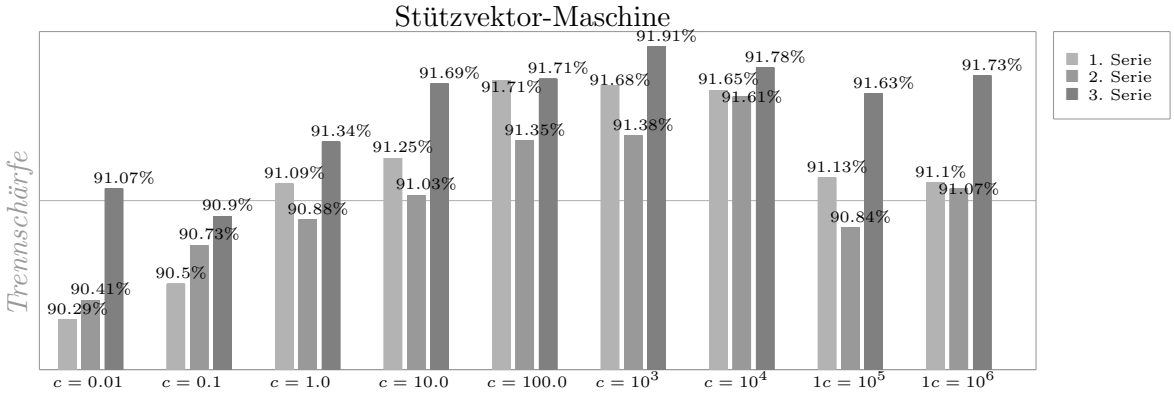


Abbildung 5.3: Trennschärfe der Bewegungsprofile durch die SVM in den Testserien 1–3 in Abhängigkeit der Kosten der Fehlerterme c

nen kann, wird es durch die Methode der *paarweisen Trennung* (s. Abschnitt 2.4.2), auf ein Verfahren für die Trennung mehrerer Klassen, erweitert. Für den Parameter c, durch den die Kosten einer Missklassifikation festgelegt werden, werden die Werte $c = 10^i$ mit $i = -2 \dots 6$ gewählt. Dadurch soll der Bereich einer optimalen Trennung grob eingegrenzt werden, um zu erfahren, wie gut sich die Daten mit einem linearen Modell trennen lassen.
In der ersten drei Testserien trennt die SVM die Klassen wesentlich schlechter voneinander als der KNN-Algorithmus. Innerhalb der ersten drei Testserien wird der beste Wert von $91,91\%$ bei $c = 1000$ in der dritten Serie erzielt. Die Ergebnisse dieser Parametersuche werden in Abbildung 5.3 gezeigt.
Auch mit dem letzten Verfahren, der logistischen Regression, werden schlechte Ergebnisse erzielt. Wie in Abbildung 5.4 abzulesen ist, übersteigt die Güte des Verfahrens die $91,26\%$ in der ersten drei Testserien nicht. Jedoch kann hier beobachtet werden, dass die Güte der linearen Trennung mit Hinzunahme weiterer Merkmale zunimmt.

Zwar ist durch KNN-Algorithmus in den ersten drei Testserien eine $98,12\%$-ige Trennung der Klassen möglich, werden aber die Konfusionmatrizen in Tabelle 5.4 betrachtet, lässt sich erkennen, dass zwar die Klasse *Liegen* sehr gut erkannt wird, die Trennung der übrigen Klassen ist jedoch relativ ungenau. Insbesondere die Unterscheidung der Klassen *stehen* und *sitzen*. Dies lässt sich damit erklären, dass die Lage der am Brustkorb angebrachten Sensoren in diesen Körperhaltungen nahe zu identisch ist. Wird die Orientierung des Oberschenkels mit in die Analysen mit aufgenommen, sind weitaus bessere Ergebnisse zu erwarten.
Wie sich aus den Ergebnissen der letzten drei Testreihen ablesen lässt, ergibt sich durch den KNN-Algorithmus eine Trennschärfe der Klassen von $99,68\%$ in der vierten Testserie bei $k = 1$. Abzulesen ist dies in der Übersicht über die Parameterauswertung des KNN-Algorithmus in Abbildung 5.5.
Auch die Ergebnisse der logistischen Regression (Abbildung 5.4) und der Support Vector

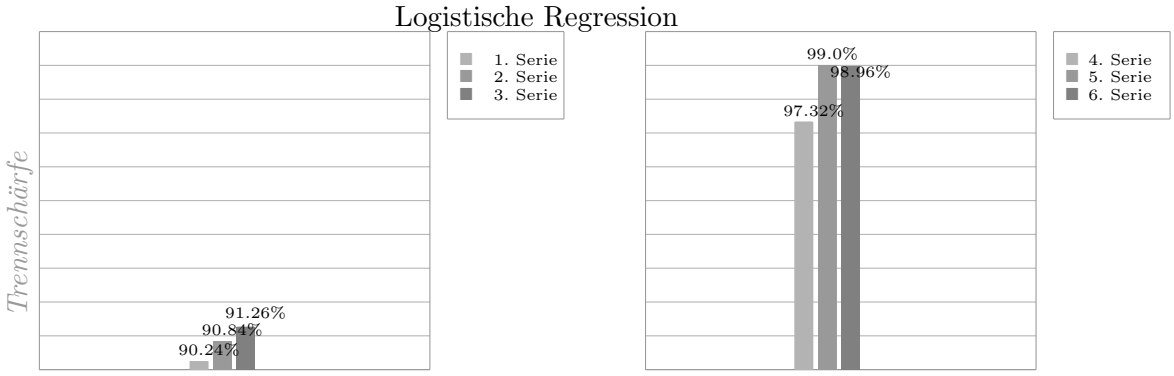


Abbildung 5.4: Trennschärfe der Bewegungsdaten durch die logistische Regression in den Testserien 1–6

Machine (Abbildung 5.6) zeigen eine Verbesserung der Trennschärfe. Das beste Ergebnis wird hier durch die SVM mit $c = 100$ und einer $99,59\%$-igen Trennschärfe erzielt. Die Auflistung der Gesamtergebnisse aus den Testserien 4–6 sind der Tabelle 5.5 zu entnehmen.

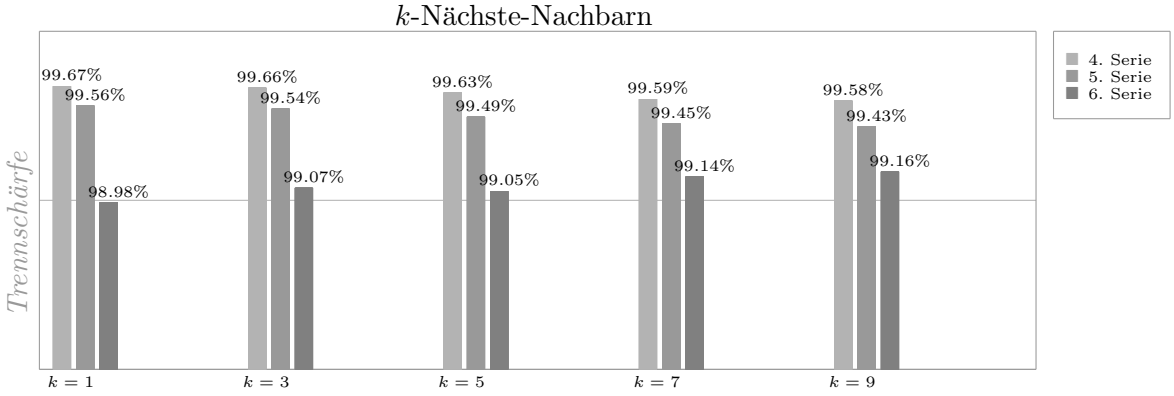


Abbildung 5.5: Trennschärfe der Bewegungsprofile durch den KNN-Algorithums in den Testserien 4–6 in Abhängigkeit des Parameters k

Die Auswertung der Testserien zeigt, dass für eine Trennung der festgelegten Bewegungsprofile alle vier Sensoren benötigt werden. Die Modellierung dieser Profile durch den KNN-Algorithmus ($k = 1$) bietet hierbei im einfachsten Merkmalsraum (3. Serie) die besten Ergebnisse. Jedoch lassen sich die Daten offenbar auch linear gut voneinander Trennen, wie die Ergebnisse der SVM ($c = 100$) zeigen. Hierfür wird aber ein wesentlich komplexerer Merk-

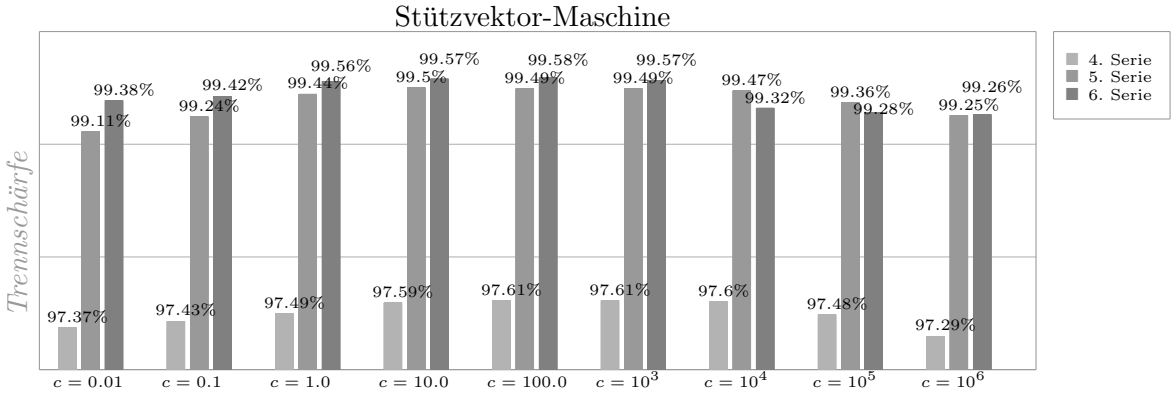


Abbildung 5.6: Trennschärfe der Bewegungsprofile durch die SVM in den Testserien 1–3 in Abhängigkeit der Kosten der Fehlerterme c

malsraum (6. Serie) benötigt. Zwar fällt das Analyseergebnis der SVM gegenüber dem KNN-Algorithmus schlechter aus, aber ein optimaler Wert für den Parameter c ist noch nicht bestimmt worden. Dieser Wert muss durch weitere Testreihen im Intervall $(10, 1.000)$ bestimmt werden.

	Klasse	Gehen & Treppe	Liegen	Stehen	Sitzen	bal. Trennschärfe
1. Serie	**k-Nearest-Neighbors ($k = 3$)**					98,12%
	Gehen & Treppe	50,77%	0,00%	0,17%	0,06%	99,55%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,09%	0,00%	11,36%	0,77%	92,96%
	Sitzen	0,13%	0,00%	0,66%	11,41%	93,51%
	Support Vector Machine ($c = 100$)					91,71%
	Gehen & Treppe	50,79%	0,00%	0,17%	0,04%	99,59%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,05%	0,00%	10,49%	1,68%	85,85%
	Sitzen	0,11%	0,00%	6,24%	5,85%	47,97%
	Logistische Regression					90,25%
	Gehen & Treppe	50,79%	0,00%	0,16%	0,05%	99,59%
	Liegen	0,00%	24,38%	0,00%	0,20%	99,18%
	Stehen	0,08%	0,00%	8,52%	3,62%	69,74%
	Sitzen	0,12%	0,09%	5,43%	6,56%	53,76%
2. Serie	**k-Nearest-Neighbors ($k = 1$)**					97,26%
	Gehen & Treppe	50,67%	0,00%	0,14%	0,18%	99,36%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,11%	0,00%	10,93%	1,17%	89,44%
	Sitzen	0,12%	0,01%	0,99%	11,08%	90,77%
	Support Vector Machine ($c = 10.000$)					91,61%
	Gehen & Treppe	50,71%	0,00%	0,21%	0,08%	99,44%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,02%	0,00%	9,23%	2,97%	75,53%
	Sitzen	0,08%	0,00%	5,03%	7,10%	58,14%
	Logistische Regression					90,84%
	Gehen & Treppe	50,77%	0,00%	0,12%	0,11%	99,55%
	Liegen	0,00%	24,50%	0,00%	0,09%	99,65%
	Stehen	0,07%	0,00%	8,04%	4,11%	65,83%
	Sitzen	0,13%	0,00%	4,54%	7,54%	61,74%
3. Serie	**k-Nearest-Neighbors ($k = 9$)**					95,62%
	Gehen & Treppe	50,78%	0,02%	0,13%	0,07%	99,57%
	Liegen	0,16%	24,38%	0,02%	0,02%	99,18%
	Stehen	0,11%	0,03%	10,73%	1,35%	87,88%
	Sitzen	0,12%	0,01%	2,35%	9,72%	79,66%
	Support Vector Machine ($c = 1000$)					91,91%
	Gehen & Treppe	50,78%	0,00%	0,11%	0,11%	99,57%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,10%	0,00%	9,68%	2,44%	79,28%
	Sitzen	0,06%	0,01%	5,27%	6,87%	56,26%
	Logistische Regression					91,26%
	Gehen & Treppe	50,73%	0,00%	0,14%	0,12%	99,48%
	Liegen	0,02%	24,48%	0,00%	0,09%	99,57%
	Stehen	0,11%	0,00%	8,24%	3,87%	67,47%
	Sitzen	0,11%	0,00%	4,29%	7,81%	64,01%

Tabelle 5.4: Analyseergebnisse der ersten Testreihe. Aufgelistet sind die Konfufionsmatrizen der Modelle die bei der Parameterauswertung die beste Güte lieferten

	Klasse	Gehen & Treppe	Liegen	Stehen	Sitzen	bal. Trennschärfe
4. Serie	**k-Nearest-Neighbors ($k = 1$)**					99,68%
	Gehen & Treppe	50,85%	0,00%	0,15%	0,00%	99,70%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,09%	0,00%	12,10%	0,03%	99,06%
	Sitzen	0,05%	0,00%	0,01%	12,15%	99,53%
	Support Vector Machine ($c = 100$)					97,61%
	Gehen & Treppe	50,76%	0,00%	0,23%	0,01%	99,53%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,08%	0,00%	12,13%	0,01%	99,30%
	Sitzen	0,06%	0,00%	2,01%	10,14%	83,10%
	Logistische Regression					97,33%
	Gehen & Treppe	50,77%	0,00%	0,23%	0,00%	99,55%
	Liegen	0,00%	24,48%	0,00%	0,11%	99,57%
	Stehen	0,08%	0,00%	11,80%	0,33%	96,64%
	Sitzen	0,07%	0,00%	1,86%	10,28%	84,19%

	Klasse	Gehen & Treppe	Liegen	Stehen	Sitzen	bal. Trennschärfe
5. Serie	**k-Nearest-Neighbors ($k = 1$)**					99,56%
	Gehen & Treppe	50,72%	0,00%	0,27%	0,01%	99,46%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,09%	0,00%	12,11%	0,02%	99,14%
	Sitzen	0,03%	0,00%	0,03%	12,15%	99,53%
	Support Vector Machine ($c = 10$)					99,50%
	Gehen & Treppe	50,78%	0,00%	0,22%	0,00%	99,57%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,03%	0,00%	12,17%	0,02%	99,61%
	Sitzen	0,04%	0,00%	0,19%	11,98%	98,12%
	Logistische Regression					99,01%
	Gehen & Treppe	50,65%	0,00%	0,34%	0,00%	99,33%
	Liegen	0,01%	24,57%	0,00%	0,00%	99,96%
	Stehen	0,05%	0,00%	12,16%	0,01%	99,53%
	Sitzen	0,06%	0,00%	0,53%	11,62%	95,23%

	Klasse	Gehen & Treppe	Liegen	Stehen	Sitzen	bal. Trennschärfe
6. Serie	**k-Nearest-Neighbors ($k = 9$)**					99,17%
	Gehen & Treppe	50,78%	0,03%	0,16%	0,03%	99,57%
	Liegen	0,07%	24,47%	0,04%	0,01%	99,53%
	Stehen	0,08%	0,06%	12,01%	0,07%	98,36%
	Sitzen	0,09%	0,05%	0,16%	11,91%	97,57%
	Support Vector Machine ($c = 100$)					99,59%
	Gehen & Treppe	50,83%	0,00%	0,17%	0,00%	99,66%
	Liegen	0,00%	24,58%	0,00%	0,00%	100,00%
	Stehen	0,05%	0,00%	12,14%	0,03%	99,37%
	Sitzen	0,04%	0,00%	0,12%	12,04%	98,67%
	Logistische Regression					98,97%
	Gehen & Treppe	50,65%	0,00%	0,33%	0,01%	99,33%
	Liegen	0,01%	24,57%	0,00%	0,00%	99,96%
	Stehen	0,01%	0,00%	12,15%	0,06%	99,45%
	Sitzen	0,07%	0,00%	0,54%	11,59%	94,99%

Tabelle 5.5: Analyseergebnisse der zweiten Testreihe. Aufgelistet sind die Konfufionsmatrizen der Modelle die bei der Parameterauswertung die beste Güte lieferten

KAPITEL 6

Fazit

In dieser Arbeit sollten die Grundlagen für ein System geschaffen werden, welches die Zusammenarbeit in den Disziplinen Psychophysiologie und Informatik im Rahmen der Klassifikation multivariater Zeitreihen unterstützen kann. Bei dieser Kooperation sind die Psychophysiologen als Nutzer von Klassifizierungsverfahren und -methoden zu sehen, während die Informatiker das Entwickeln und Testen diese Methoden und Verfahren übernehmen. Gemeinsam nutzen sie die Daten aus psychophysiologischen Studien.
Um die Aufgabe eines solchen Systems festzulegen, wurden zunächst die grundlegenden Prozesse der Klassifizierung formal erfasst, mathematisch beschrieben und in Arbeitsschritte aufgeteilt. Daraufhin wurde eine Systemarchitektur entworfen,die in der Lage ist, diese Arbeitsschritte auf verschiedene Systeme zu verteilen, Dadurch sollte erreicht werden, dass die unterschiedlichen Nutzergruppen ihre Arbeiten in ihren gewohnten Arbeitsumgebungen weiterführen können.

Es wurde eine Client-Server-Architektur vorgeschlagen, die XML-Dokumente für den Austausch von Daten nutzt. Einer Master-Komponente sind hierfür Koordinationsaufgaben und eine konsistente Datenhaltung zugeordnet worden. Die Teilsysteme entsprechen den gewohnten Arbeitsumgebungen der Nutzer. Sie kommunizieren asynchron mit der Master-Komponente, um Daten und Funktionsaufrufe zu übermitteln. Die Spezifikation für diese Kommunikationsprotokoll ist durch XML Schema Definitions beschrieben worden. Dadurch kann das System leicht um neue Funktionen und Arbeitsabläufe erweitert werden.
Die Master-Komponente ist für die Bearbeitung von Klassifizierungsaufgaben nicht auf den eigenen Funktionsumfang begrenzt, sondern verteilt die Berechnungen vielmehr andere System-Komponenten. Dadurch können Entwicklersysteme dem Gesamtsystem neue Verfahren und Methoden hinzufügen und anderen Nutzern zur Verfügung stellen. Dadurch benötigen deren Systeme keine weiter Anpassungen.

Auf Grund der vorgeschlagenen Architektur wurden am Max–Planck–Institut für Bildungsforschung die Schnittstellen für Java™ und Python umgesetzt und eine Master-Komponente entwickelt. Der Aufbau dieses Systems wurde beschrieben und eine Studie vorgestellt, in der es genutzt wurde um Datenerhebungen auszuwerten. Ein Anwendungsfall dieser Studie ist anschließend beschreiben und ausgewertet worden.
Mit diesen Auswertungen wurde gezeigt, dass sich ein System anhand dieses Entwurfs implementieren lässt, das für Segmentklassizierungsaufgaben im Rahmen des überwachten Lernens geeignet ist.
Darüber hinaus zeigt sich im Praxistest, dass Personen ohne weitgreifende Kenntnis über das Themengebiet der Mustererkennung, einen interaktiven Zugang zum System nutzen können, um Daten zu analysieren. Damit schließt das System nicht nur die Lücken zwischen verschieden Anwendungen im Feld der Mustererkennung, sondern es bietet auch einen wesentlichen Vorteil gegenüber lokalen Anwendungen.
Auf Grund dieser Vorteile befindet sich das System bereits im Einsatz und wird in mehreren Forschungsprojekten der Informatik und Psychophysiologie aktiv genutzt. Zudem soll es in naher Zukunft auch in der Lehre genutzt werden, um den Umgang mit statistischen Verfahren zu schulen.

Wie die Erfahrung im Umgang mit Projekten zeigt, bieten diese nach ihrem Abschluss viel Raum für viele neue Ideen. Durch die Wahl der Beschreibungssprache XSD und XML-Dokumenten und durch den Entwurf einer einfachen, allgemeinen Systemstruktur eine Grundlage geschaffen worden, die es erlaubt, viele solcher Ideen im vorgegebenen Rahmen umzusetzen und zu testen.

Abkürzungsverzeichnis

CORBA Common Object Request Broker Architecture

DTD Document Type Definition

DTW Dynamic Time Warping

EEG Elektroenzephalographie

EKG Elektrokardiographie

EMG Elektromyographie

GUI Graphical User Interface

KNN k-Nearest-Neighbors

MRT Magnet Resonanz Tomographie

RMI Remote Method Invocation

RPC Remote Procedure Call

RPyC Remote Python Call

SOAP Simple Object Access Protocol

SVM Support Vector Machine

TCP Transmission Control Protocol

UDP User Datagram Protocol

W3C World Wide Web Consortium

XML Extendable Markup Language

XSD XML Schema Definition

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[1] *Multivariate Pattern Analysis in Python*. http://www.pymvpa.org. Version: Dez. 2011, Abruf: 19. Dezember 2011

[2] *Python Programming Language*. http://python.org. Version: Dez. 2011, Abruf: 19. Dezember 2011

[3] *XML-RPC*. http://xmlrpc.scripting.com/default.html. Version: Nov 2011, Abruf: 30. 11. 2011

[4] *RPyC*. http://rpyc.sourceforge.net/. Version: Feb. 2012, Abruf: 28. 2. 2012

[5] Bishop, Christopher M.: *Pattern Recognition and Machine Learning*. 1st, 2006. Corr 2nd printing. Springer, 2007

[6] Bray, Tim ; Paoli, Jean ; Maler, Eve ; Yergeau, François ; Sperberg-McQueen, C. M.: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. November 2008. – http://www.w3.org/TR/2008/REC-xml-20081126/

[7] Duda, Richard O. ; Hart, Peter E. ; Stork, David G.: *Pattern classification*. 2nd. Wiles&Sons, 2001

[8] Fowler, Martin: Separating User Interface Code. In: *IEEE Software* 18 (2001), S. 96–97. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/52.914754. – DOI http://doi.ieeecomputersociety.org/10.1109/52.914754

[9] Gudgin, Martin ; Hadley, Marc ; Mendelsohn, Noah ; Lafon, Yves ; Moreau, Jean-Jacques ; Karmarkar, Anish ; Nielsen, Henrik F.: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Version: apr 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, Abruf: 23

[10] Hopcroft, John E. ; Motwani, Rajeev ; Ullman, Jeffrey D.: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. 2. Aufl. Pearson Studium, 2002

[11] Josefsson, S.: *The Base16, Base32, and Base64 Data Encodings.* RFC 4648 (Proposed Standard). http://www.ietf.org/rfc/rfc4648.txt. Version: Oktober 2006 (Request for Comments)

[12] Malhotra, Ashok ; Biron, Paul V.: *XML Schema Part 2: Datatypes Second Edition.* Oktober 2004. – http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[13] Microsystems, Sun: *RPC: Remote Procedure Call Protocol specification: Version 2.* RFC 1057 (Informational). http://www.ietf.org/rfc/rfc1057.txt. Version: Juni 1988 (Request for Comments)

[14] Mitra, Nilo ; Lafon, Yves: *SOAP Version 1.2 Part 0: Primer (Second Edition).* http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Version: Apr 2007

[15] Moreau, Jean-Jacques ; Gudgin, Martin ; Hadley, Marc ; Mendelsohn, Noah ; Lafon, Yves ; Karmarkar, Anish ; Nielsen, Henrik F.: *SOAP Version 1.2 Part 2: Adjuncts (Second Edition).* Version: apr 2007. http://www.w3.org/TR/2007/REC-soap12-part2-20070427/

[16] Object Management Group (Hrsg.): *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1, Part 1.* OMG Headquarters, 140 Kendrick Street, Building A, Suite 300 Needham, MA 02494, USA: Object Management Group, Aug. 2011. http://www.omg.org/spec/CORBA/3.1.1/Interfaces/PDF

[17] Object Management Group (Hrsg.): *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1, Part 2.* OMG Headquarters, 140 Kendrick Street, Building A, Suite 300 Needham, MA 02494, USA: Object Management Group, Aug. 2011. http://www.omg.org/spec/CORBA/3.1.1/Interoperability/PDF/

[18] Object Management Group (Hrsg.): *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1, Part 3.* OMG Headquarters, 140 Kendrick Street, Building A, Suite 300 Needham, MA 02494, USA: Object Management Group, Aug. 2011. http://www.omg.org/spec/CORBA/3.1.1/Components/PDF/

[19] Sakoe, Hiroaki ; Chiba, Seibi: Dynamic Programming Algorithm Optimization for Spoken Word Recognition. In: *EEE Transactions on Acoustics, Speech and Signal Processing* 26 (1978), Feb., Nr. 1, S. 43–49

[20] Settles, Burr: Active Learning Literature Survey / University of Wisconsin–Madison. 2009 (1648). – Computer Sciences Technical Report

[21] Stevenson, D.: A Proposed Standard for Binary Floating-Point Arithmetic. In: *Computer* 14 (1981), S. 51–62. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/C-M.1981.220377. – DOI http://doi.ieeecomputersociety.org/10.1109/C–M.1981.220377. – ISSN 0018–9162

[22] Thompson, Henry S. ; Maloney, Murray ; Beech, David ; Mendelsohn, Noah: *XML Schema Part 1: Structures Second Edition.* Oktober 2004. – http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[23] Thurlow, R.: *RPC: Remote Procedure Call Protocol Specification Version 2.* RFC 5531 (Draft Standard). `http://www.ietf.org/rfc/rfc5531.txt`. Version: Mai 2009 (Request for Comments)

[24] Tsujinishi, Daisuke ; Koshiba, Yoshiaki ; Abe, Shigeo: Why pairwise is better than one-against-all or all-at-once. In: *IEEE International Joint Conference on Neural Networks* Bd. 1, 2004. – ISSN 1098–7576, S. 693–698

[25] Walmsley, Priscilla ; Fallside, David C.: *XML Schema Part 0: Primer Second Edition.* Oktober 2004. – http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

[26] White, J.E.: *High-level framework for network-based resource sharing.* RFC 707. `http://www.ietf.org/rfc/rfc707.txt`. Version: Dezember 1975 (Request for Comments)

ANHANG A

XML Schema Definitions

In diesem Anhang werden die XML Schema Definitions der Datentypen des Systems angegeben. Diese Definitionen werden in drei Teile getrennt:

- `Mano.xsd` definiert die Basistypen der Systems und wird mit dem *Namespace-Prefix* `mango` ausgezeichnet
- `Workflows.xsd` definiert die Arbeitsabläufe, die das Systems verarbeitet ihnen wird `work` vorangestellt
- `Protocol.xsd` definiert die eigentliche Kommunikations-Ebene des Systems

1 Mango.xsd

```
20 <complexType name="Segment">
21   <sequence maxOccurs="unbounded" minOccurs="0">
22     <sequence maxOccurs="unbounded" minOccurs="1">
23         <element name="Channel" type="mango:Channel"/>
24     </sequence>
25     <sequence maxOccurs="unbounded" minOccurs="0">
26       <element name="Marker" type="mango:Marker"/>
27     </sequence>
28   </sequence>
29   <attribute name="dataobject" type="string"/>
30   <attribute name="position" type="float" use="optional"/>
31 </complexType>
```

Quellcode 1.1: Segment

```
<complexType name="DataSource">
  <sequence maxOccurs="unbounded" minOccurs="1">
    <choice>
      <element name="Segment" type="mango:Segment"/>
      <element name="DataObject" type="mango:DataObject"/>
      <element name="Filter" type="work:Filter"/>
      <element name="Reference" type="mango:DataObjectRef"/>
    </choice>
  </sequence>
</complexType>
```

Quellcode 1.2: DataSource

```
<complexType name="DataObject">
  <complexContent>
    <extension base="mango:MangoElement">
      <sequence maxOccurs="unbounded" minOccurs="0">
        <sequence maxOccurs="unbounded" minOccurs="1">
          <element name="Channel" type="mango:Channel"/>
        </sequence>
        <sequence>
          <element name="Marker" type="mango:Marker"/>
        </sequence>
      </sequence>
      <attribute name="id" type="string" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

Quellcode 1.3: DataObject

```
<complexType name="Channel">
  <complexContent>
    <extension base="mango:MangoElement">
      <sequence maxOccurs="unbounded" minOccurs="0">
        <any></any>
      </sequence>
      <attribute name="samplingrate" type="string" use="required" />
      <attribute name="id" type="string" use="required" />
      <attribute name="type" type="mango:channeltype" use="required"
          />
      <attribute name="encoding" type="mango:encoding" use="required"
          />
    </extension>
  </complexContent>
</complexType>
```

Quellcode 1.4: Channel

```
70 <complexType name="MangoElement">
71 <attribute name="desc" type="string" use="required"/>
72   <attribute name="name" type="string" use="required"/>
73 </complexType>
```

Quellcode 1.5: MangoElement

```
74 <complexType name="Feature">
75   <complexContent>
76     <extension base="mango:MangoElement">
77       <sequence maxOccurs="unbounded" minOccurs="1">
78         <element name="Channel" type="IDREF"/>
79       </sequence>
80       <attribute name="id" type="string"/>
81     </extension>
82   </complexContent>
83 </complexType>
```

Quellcode 1.6: Feature

```
84 <complexType name="FeatureSpace">
85   <sequence maxOccurs="unbounded" minOccurs="1">
86     <element name="Feature" type="mango:FeatureDef"/>
87   </sequence>
88 </complexType>
```

Quellcode 1.7: FeatureSpace

```
89 <complexType name="Label">
90   <complexContent>
91     <extension base="mango:MangoElement">
92       <attribute name="start" type="string"/>
93       <attribute name="end" type="string"/>
94       <attribute name="dataobject" type="string"/>
95       <attribute name="id" type="string"/>
96       <attribute name="parent" type="string"/>
97     </extension>
98   </complexContent>
99 </complexType>
```

Quellcode 1.8: Label

```
100 <complexType name="DataObjectList">
101   <sequence maxOccurs="unbounded" minOccurs="1">
102     <element name="Object" type="mango:DataObject"/>
103   </sequence>
104 </complexType>
```

Quellcode 1.9: DataObjectList

```
105 <complexType name="FeatureDef">
106   <sequence maxOccurs="unbounded" minOccurs="1">
107     <element name="Channel" type="mango:ChannelDef"></element>
108   </sequence>
109   <attribute name="id" type="string" use="required"></attribute>
110 </complexType>
```

Quellcode 1.10: FeatureDef

```
111 <complexType name="LabelRef">
112   <attribute name="id" type="string" use="required"></attribute>
113 </complexType>
```

Quellcode 1.11: LabelRef

```
114 <complexType name="ChannelDef">
115   <attribute name="id" type="mango:ChannelRef" use="required"/>
116   <attribute name="alias" type="string" use="optional"/>
117 </complexType>
```

Quellcode 1.12: ChannelDef

```
118 <complexType name="DataObjectInfo">
119     <sequence maxOccurs="unbounded" minOccurs="1">
120       <sequence maxOccurs="unbounded" minOccurs="1">
121         <element name="Channel" type="mango:ChannelDef"/>
122       </sequence>
123       <sequence maxOccurs="unbounded" minOccurs="0">
124         <element name="Marker" type="mango:Marker"/>
125       </sequence>
126     </sequence>
127     <attribute name="id" type="string" use="required" />
128 </complexType>
```

Quellcode 1.13: DataObjectInfo

```
129 <complexType name="Marker">
130   <attribute name="name" type="string"></attribute>
131   <attribute name="position" type="double"></attribute>
132   <attribute name="desc" type="string"></attribute>
133 </complexType>
```

Quellcode 1.14: Marker

```
134 <complexType name="Parameter">
135   <complexContent>
136     <extension base="mango:MangoElement">
137       <attribute name="id" type="string" use="required"/>
138     </extension>
139   </complexContent>
140 </complexType>
```

Quellcode 1.15: Parameter

```
141 <complexType name="floatParameter">
142   <complexContent>
143     <extension base="mango:Parameter">
144       <attribute name="min" type="double" use="optional"/>
145       <attribute name="max" type="double" use="optional"/>
146       <attribute name="step" type="double" use="optional"/>
147 
148     </extension>
149   </complexContent>
150 </complexType>
```

Quellcode 1.16: floatParameter

```
151 <complexType name="intParameter">
152   <complexContent>
153     <extension base="mango:Parameter">
154       <attribute name="min" type="long" use="optional"/>
155       <attribute name="max" type="long" use="optional"/>
156       <attribute name="step" type="long" use="optional"/>
157 
158     </extension>
159   </complexContent>
160 </complexType>
```

Quellcode 1.17: intParameter

```
<complexType name="Config">
  <complexContent>
    <extension base="mango:MangoElement">
      <sequence maxOccurs="unbounded" minOccurs="0">
        <choice>
          <element name="Integer"
            type="mango:intParameter">
          </element>
          <element name="Float" type="mango:floatParameter"/>
          <element name="String" type="mango:StringParameter"/>
          <element name="Choice" type="mango:Config"/>
          <element name="Selection" type="mango:Config"/>
        </choice>
      </sequence>
      <attribute name="id" type="string" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

Quellcode 1.18: Config

```
<complexType name="Characteristic">
  <sequence maxOccurs="unbounded" minOccurs="1">
    <element name="Channel" type="string"/>
  </sequence>
  <attribute name="id" type="string"/>
  <attribute name="value" type="float"/>
</complexType>
```

Quellcode 1.19: Characteristic

```
<complexType name="Characteristics">
  <sequence maxOccurs="unbounded" minOccurs="1">
    <element name="Characteristic" type="mango:Characteristic"/>
  </sequence>
</complexType>
```

Quellcode 1.20: Characteristics

```
<complexType name="StringParameter">
  <complexContent>
    <extension base="mango:Parameter"/>
  </complexContent>
</complexType>
```

Quellcode 1.21: StringParameter

```
<complexType name="ParameterValue">
  <simpleContent>
    <extension base="string">
      <attribute name="id" type="string" use="required"/>
    </extension>
  </simpleContent>
</complexType>
```

Quellcode 1.22: ParameterValue

```
<complexType name="LabeledSet">
  <sequence maxOccurs="1" minOccurs="1">
    <element name="Label" type="mango:LabelRef"
             maxOccurs="1" minOccurs="1"/>
    <element name="Label" type="mango:LabelRef"
             maxOccurs="unbounded" minOccurs="1" />
    <sequence maxOccurs="unbounded" minOccurs="1">
      <choice>
        <element name="Filter" type="work:Filter"/>
        <element name="Segment" type="mango:Segment"/>
        <element name="DataObject" type="mango:DataObject"/>
      </choice>
    </sequence>
  </sequence>
</complexType>
```

Quellcode 1.23: LabeledSet

```
218 <complexType name="ParameterSet">
219   <sequence>
220     <element name="Value" type="mango:ParameterValue"/>
221   </sequence>
222   <attribute name="id" type="string" use="required"/>
223 </complexType>
```

Quellcode 1.24: ParameterSet

```
224 <complexType name="FilterDef">
225   <sequence maxOccurs="unbounded" minOccurs="1">
226     <choice>
227       <element name="Filter" type="mango:FilterDef"/>
228       <element name="Channel" type="mango:ChannelDef"/>
229     </choice>
230   </sequence>
231   <attribute name="id" type="string"/>
232 </complexType>
```

Quellcode 1.25: FilterDef

```
233 <complexType name="DataObjectRef">
234   <attribute name="id" type="string" use="required"/>
235   <attribute name="start" type="float" use="optional"/>
236   <attribute name="end" type="float" use="optional"/>
237 </complexType>
```

Quellcode 1.26: DataObjectRef

```
238 <complexType name="ConfusionMatrix">
239   <sequence>
240     <element name="Target" type="mango:Target"
241              maxOccurs="unbounded" minOccurs="0"/>
242   </sequence>
243 </complexType>
```

Quellcode 1.27: ConfusionMatrix

```
244 <complexType name="Prediction">
245   <simpleContent>
246     <extension base="integer">
247       <attribute name="label" type="string"/>
248     </extension>
249   </simpleContent>
250 </complexType>
```

Quellcode 1.28: Prediction

```
251 <complexType name="Target">
252   <sequence>
253     <element name="Prediction" type="mango:Prediction"
254              maxOccurs="unbounded" minOccurs="0"/>
255   </sequence>
256   <attribute name="label" type="string"/>
257 </complexType>
```

Quellcode 1.29: Target

2 Workflows.xsd

```
<complexType name="Crossvalidation">
  <complexContent>
    <extension base="work:JobDef">
      <attribute name="folds" use="required">
        <simpleType><restriction base="int">
                    <minInclusive value="1"/>
                    </restriction>
        </simpleType>
      </attribute>
    </extension>
  </complexContent>
</complexType>
```

Quellcode 2.30: Crossvalidation

```
<complexType name="JobDef">
    <sequence maxOccurs="unbounded" minOccurs="1">
      <element name="Model" type="work:ParameterSet"
               maxOccurs="unbounded" minOccurs="1" />
      <element name="FeatureSpace" type="base:FeatureSpace"
               maxOccurs="unbounded" minOccurs="1" />
        <element name="Label" type="base:LabelRef"
                 maxOccurs="1" minOccurs="1"/>
        <element name="Label" type="base:LabelRef"
                 maxOccurs="unbounded" minOccurs="1"/>

        <choice maxOccurs="unbounded" minOccurs="1">
          <element name="DataObject" type="base:DataObjectInfo"
                   maxOccurs="unbounded" minOccurs="1" />
          <element name="Filter" type="work:Filter"
                   maxOccurs="unbounded" minOccurs="1" />
          <element name="Segment" type="base:Segment"
                   maxOccurs="unbounded" minOccurs="1" />
          <element name="Referenze" type="base:DataObjectRef" />
        </choice>

    </sequence>
    <attribute name="name" type="string"/>
</complexType>
```

Quellcode 2.31: JobDef

```
294 <complexType name="ParameterSet">
295   <sequence maxOccurs="unbounded" minOccurs="0">
296     <element name="Value" type="work:Value"/>
297   </sequence>
298   <attribute name="id" type="string"/>
299 </complexType>
```

Quellcode 2.32: ParameterSet

```
300 <complexType name="Value">
301   <attribute name="id" type="string"/>
302 </complexType>
```

Quellcode 2.33: Value

```
303 <complexType name="FilterValues">
304   <sequence maxOccurs="unbounded" minOccurs="0">
305     <choice>
306       <element name="Filter" type="work:FilterValues"/>
307       <element name="Value" type="work:Value"/>
308     </choice>
309   </sequence>
310   <attribute name="id" type="string"/>
311 </complexType>
```

Quellcode 2.34: FilterValues

```
312 <complexType name="Filter">
313   <sequence maxOccurs="unbounded" minOccurs="1">
314     <choice>
315       <element name="DataObject" type="base:DataObjectInfo"/>
316       <element name="Filter" type="work:Filter"/>
317       <element name="Segment" type="base:Segment"/>
318       <element name="Reference" type="base:DataObjectRef"/>
319     </choice>
320   </sequence>
321   <attribute name="id" type="string" use="required"/>
322 </complexType>
```

Quellcode 2.35: Filter

```
<complexType name="CrossvalidationResponse">
  <sequence>
    <element name="Filter" type="work:FilterValues"/>
    <element name="Model" type="work:ModelDef"
             maxOccurs="unbounded" minOccurs="1"/>
  </sequence>
</complexType>
```

Quellcode 2.36: CrossvalidationResponse

```
<complexType name="ModelDef">
  <complexContent>
    <extension base="work:ParameterSet">
      <attribute name="accuracy" type="double"/>
    </extension>
  </complexContent>
</complexType>
```

Quellcode 2.37: ModelDef

```
<complexType name="Analysis">
  <complexContent>
    <extension base="work:JobDef"/>
  </complexContent>
</complexType>
```

Quellcode 2.38: Analysis

```
<complexType name="FeatureExtraction">
  <sequence>
    <element name="FeatureSpace" type="base:FeatureSpace"
             minOccurs="1" maxOccurs="unbounded"/>
      <choice maxOccurs="unbounded" minOccurs="1">
        <element name="Filter" type="work:Filter"
                 maxOccurs="unbounded" minOccurs="0"/>
        <element name="DataObject" type="base:DataObject"
               maxOccurs="unbounded" minOccurs="0" />
        <element name="Segment" type="base:Segment"
                 maxOccurs="unbounded" minOccurs="0"/>
    </choice>
  </sequence>
</complexType>
```

Quellcode 2.39: FeatureExtraction

```
<complexType name="Classification">
  <complexContent>
    <extension base="work:JobDef">
      <sequence maxOccurs="unbounded" minOccurs="1">
        <element name="TestData" type="base:DataSource"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Quellcode 2.40: Classification

```
<complexType name="Training">
  <sequence>
    <element name="Model" type="base:ParameterSet"
             maxOccurs="unbounded" minOccurs="1"/>
    <element name="FeatureSpace" type="base:FeatureSpace"
             maxOccurs="1" minOccurs="1"/>
    <element name="Label" type="base:LabelRef"
             maxOccurs="1" minOccurs="1"/>
    <element name="Label" type="base:LabelRef"
             maxOccurs="unbounded" minOccurs="1" />
    <sequence maxOccurs="unbounded" minOccurs="1">
      <choice>
        <element name="Filter" type="work:Filter"/>
        <element name="DataObject" type="base:DataObject" />
        <element name="Segment" type="base:Segment"/>
      </choice>
    </sequence>
  </sequence>
</complexType>
```

Quellcode 2.41: Training

```
384 <complexType name="ActiveSession">
385   <complexContent>
386     <extension base="base:MangoElement">
387       <sequence>
388         <element name="FeatureSpace" type="base:FeatureSpace"
389                  maxOccurs="1" minOccurs="1" />
390         <element name="Filter" type="base:FilterDef"
391                  maxOccurs="1" minOccurs="1" />
392         <element name="DataObject" type="base:DataObjectRef"
393                  maxOccurs="unbounded" minOccurs="1" />
394       </sequence>
395       <attribute name="model" type="string"/>
396     </extension>
397   </complexContent>
398 </complexType>
```

Quellcode 2.42: ActiveSession

3 Protocol.xsd

```
399 <complexType name="JobSet">
400   <complexContent>
401     <extension base="work:JobDef">
402       <sequence>
403         <element name="Job" type="work:JobDef"/>
404       </sequence>
405     </extension>
406   </complexContent>
407 </complexType>
```

Quellcode 3.43: JobSet

```
408 <complexType name="SystemRegistration">
409   <sequence maxOccurs="unbounded" minOccurs="0">
410     <choice>
411       <element name="Feature" type="proto:FunctionRef"/>
412       <element name="Filter" type="proto:FunctionRef"/>
413       <element name="Class" type="mango:Config"/>
414       <element name="DataObject" type="mango:DataObject"/>
415     </choice>
416   </sequence>
417   <attribute name="host" type="string" use="required"/>
418   <attribute name="port" type="int" use="required"/>
419 </complexType>
```

Quellcode 3.44: SystemRegistration

```
420 <complexType name="FunctionRef">
421   <complexContent>
422     <extension base="mango:MangoElement">
423       <attribute name="id" type="string" use="required"/>
424       <attribute name="cid" type="string" use="required"/>
425     </extension>
426   </complexContent>
427 </complexType>
```

Quellcode 3.45: FunctionRef

```
428 <complexType name="FunctionRegistration">
429   <attribute name="id" type="string" use="required"/>
430   <attribute name="cid" type="string" use="required"/>
431 </complexType>
```

Quellcode 3.46: FunctionRegistration

```
432 <complexType name="RegistrationResponse">
433   <sequence>
434     <element name="Filter" type="proto:FunctionRegistration"/>
435     <element name="Class" type="proto:FunctionRegistration"/>
436     <element name="Feature" type="proto:FunctionRegistration"/>
437     <element name="DataObject" type="proto:FunctionRegistration"/>
438   </sequence>
439   <attribute name="bypass" type="integer"></attribute>
440   <attribute name="id" type="string"></attribute>
441 </complexType>
```

Quellcode 3.47: RegistrationResponse

```
442 <complexType name="JobStatus">
443   <attribute name="id" use="required">
444     <simpleType>
445       <restriction base="string">
446         <minLength value="1"/>
447       </restriction>
448     </simpleType>
449   </attribute>
450   <attribute name="message" use="optional">
451     <simpleType>
452       <restriction base="string">
453         <minLength value="1"/>
454       </restriction>
455     </simpleType>
456   </attribute>
457   <attribute name="status">
458     <simpleType>
459       <restriction base="float">
460         <minExclusive value="0"/>
461         <maxExclusive value="100"/>
462       </restriction>
463     </simpleType>
464   </attribute>
465 </complexType>
```

Quellcode 3.48: JobStatus

```
466 <complexType name="Exception">
467     <sequence><any/></sequence>
468     <attribute name="id" type="string"/>
469   <attribute name="message" type="string"/>
470 </complexType>
```

Quellcode 3.49: Exception