

Integration von Clustering-/Classification-Techniken in eine objektrelationale Datenbankumgebung



Diplomarbeit

Universität Rostock
Institut für Informatik

angefertigt von: Lars Milewski
geboren am: 16. September 1977 in Crivitz
Erstgutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
Zweitgutachter: Prof. Dr. rer. nat. habil. Adelinde M. Uhrmacher
Betreuer: Dipl.-Inf. Ilvio Bruder
Dipl.-Ing. Temenushka Ignatova
Abgabedatum: 11. März 2004

Zusammenfassung

Im 18. Jahrhundert wurden Kompositionen überwiegend auf dem handschriftlichen Wege vervielfältigt und verbreitet. Solche historischen Notenhandschriften sind ein Teil des kulturellen Erbes der Menschheit. Diese Dokumente so lange wie möglich zu konservieren und gleichzeitig als Informationsquelle für die Musikforschung bereitzustellen, ist eine wichtige und schwierige Aufgabe mit vielen Teilbereichen. Dazu gehört unter anderem die Entwicklung eines Systems, das automatisch Notenhandschriften identifizieren kann.

In dieser Diplomarbeit wird zum einen die Eignung von Data-Mining-Verfahren (Klassifikation und Clustering) zur automatischen Notenhandschriftidentifikation untersucht und zum anderen werden die Integrationsmöglichkeiten solcher Techniken in objektrelationale Datenbanksysteme diskutiert. Ziel ist es, ein Konzept für eine Datenbank-integrierte Klassifikations-/Clustering-Analyse von Notenhandschriftcharakteristiken zu erstellen, dieses innerhalb des eNoteHistory-Projektes prototypisch zu implementieren, Testmethoden für das System zu entwickeln und anzuwenden.

Abstract

Music works in the 18th century were mainly disseminated by handwritten copies. These manuscripts of music scores are considered to be part of the cultural heritage of mankind. Therefore it is an important task to conserve these documents for a long time and to make them available as sources for music research. One of the problems which needs to be solved is the identification of writers of music scores based on their handwriting characteristics.

In the scope of this thesis data mining techniques (classification and clustering) have been surveyed for their suitability for identifying note handwriting on the one hand. On the other hand the possibilities of integrating such techniques into an object-relational database system have been discussed. The goal is to provide a concept for a database-integrated classification/clustering analysis of note handwriting characteristics, to implement this concept within the eNoteHistory project prototypically, to develop methods for testing the system and to conduct these tests.

CR-Klassifikation

H.2.4 Systems

H.2.8 Database Applications

H.3.3 Information Search and Retrieval

H.3.7 Digital Libraries

Keywords

writer identification, knowledge base, digital archive, feature classification with k-nearest-neighbor

Inhaltsverzeichnis

1	Einleitung	9
I	Grundlagen	11
2	Grundlagen	12
2.1	Musikwissenschaftliche Grundlagen	12
2.1.1	Schriftmerkmale	12
2.1.2	Feature Base (Feature Dictionary)	15
2.2	Data Mining	17
2.2.1	Datenrepräsentation	19
2.2.2	Klassifikation	20
2.2.3	Clustering	25
2.2.4	Allgemeine Data-Mining-Verfahren und Begriffe	28
3	Stand der Forschung und vorhandene Programme	31
3.1	Schreibererkennung	31
3.1.1	Schreibererkennung mit dem K-Nearest-Neighbor- Verfahren	31
3.1.2	Information-Retrieval-basierte Schreibererkennung	32
3.2	Data-Mining-Tools	34
4	Einsatzszenarios und Anforderungen	38
4.1	Einsatzszenarios	38
4.2	Anforderungen	40
4.2.1	Ausnutzung geeigneter Data-Mining-Verfahren	40
4.2.2	Datenbankintegration	41
4.2.3	Unterstützung von Nullwerten	41
4.2.4	Mehrere Werte für ein Feature	41
4.2.5	Fehlertoleranz	41
II	Konzeption und Umsetzung	43
5	Problembeschreibung und Voraussetzungen	44
5.1	Bewertung vorhandener Data-Mining-Verfahren	44
5.1.1	Klassifikationsverfahren	44

5.1.2	Clustering-Verfahren	46
5.2	Formale Problembeschreibung	47
5.2.1	Grundlagen	47
5.2.2	Feature Base	47
5.2.3	Distanzfunktion	49
5.2.4	Die Feature-Distanzfunktion	49
5.2.5	Ähnlichkeiten	51
5.2.6	Features mit komplexen Werten	52
5.2.7	Nullwerte	53
6	Umsetzung	55
6.1	Systemarchitektur	55
6.2	Verfahren, Methoden und Strukturen	56
6.2.1	Ähnlichkeiten	57
6.2.2	Distanzfunktionen	58
6.2.3	Nullwerte	61
6.2.4	Features mit komplexen Werten	62
6.3	Bewertungsmaße	62
6.3.1	Bewertung der Partition	62
6.3.2	Bewertung der Antwortmenge A_{FV}	64
6.3.3	Bewertung der Antwortmenge A_S	65
6.3.4	Einfluss von Nullwerten	65
7	Implementierung	67
7.1	Nutzen vorhandener Tools	67
7.2	Integration in die Datenbank	69
7.2.1	Stored Procedures	71
7.2.2	Relationenschema des Schreibererkennungssystem	72
7.3	Prototypische Implementierung	73
7.3.1	Import Tools	74
7.3.2	Klassifikationskomponente	76
8	Test und Optimierung des Systems	80
8.1	Priorität der Features	80
8.2	Wahl der Distanzfunktion	83
8.3	Gewichte in der Distanzfunktion	84
8.4	Optimierung der Distanzmatrizen	86
8.5	Bewertung der Antwortmengen in Zusammenhang mit Nullwerten	87
8.5.1	Precision und Recall	87
8.5.2	Nullwerte	87
8.6	Features mit mehreren Werten	88
9	Zusammenfassung und Ausblick	90
9.1	Umsetzung der Anforderungen	90
9.2	Ausblick	91

A Tabellen	95
A.1 Priorität der Features aus musikwissenschaftlicher Sicht	95
A.2 Testergebnis: Priorität der Features	98
A.3 Testergebnis: Distanzfunktion	99
A.4 Testergebnis: Gewichte der Features	100
A.5 Testergebnis: Ähnlichkeitswerte	101
A.6 Testergebnis: Schwellwert, Precision, Recall, K und Nullwerte . .	102
A.7 eNoteHistory Modell	103
A.8 Abkürzungs- und Symbolverzeichnis	105

Kapitel 1

Einleitung

Die Arbeit eines Musikwissenschaftlers gleicht bisweilen der eines Kriminalisten, der aus wenigen Indizien einen ganzen Sachverhalt rekonstruieren muss. Besonders ist dies der Fall bei älteren Notenhandschriften: Die Fragen, wer beispielsweise eine bestimmte Komposition wann, wo, warum und für wen geschrieben oder abgeschrieben hat, lässt sich oftmals nur mit detektivischem Spürsinn beantworten, und Indizien sind beispielsweise die individuelle Handschrift des Schreibers, das verwendete Papier, die Tinte oder das Wasserzeichen. Sofern man diese Merkmale aber sicher zuordnen kann, ist die Summe der Indizien dann fast genauso zuverlässig wie ein Fingerabdruck. [43]

Genau wie der Kriminalist heute Computer-Technologie zur Indiziensuche und -auswertung benutzt, möchte sich auch die moderne Musikwissenschaft die Vorteile der schnellen und automatischen Computer-Analyse zunutze machen. So entstand eine Kooperation zwischen dem Institut für Musikwissenschaft der Universität Rostock, dem Datenbanklehrstuhl der gleichen Universität, dem Fraunhofer Institut für Grafische Datenverarbeitung (IGD) und anderen Einrichtungen der Universität Rostock. Ziel des Projektes eNoteHistory ist es, Notenhandschriftdokumente zu konservieren und als Informationsquelle und Material für die Musikforschung bereitzustellen [26]. Ein Aspekt dabei ist die automatische Analyse von Notenhandschriften, um Rückschlüsse auf den Schreiber der Noten zu ziehen. Dafür gibt es zwei unabhängige Ansätze. Der erste wird vom Fraunhofer Institut für Grafische Datenverarbeitung verfolgt. Er basiert darauf, die Handschriftcharakteristik mittels Bildverarbeitungsalgorithmen zu extrahieren und zu klassifizieren. Der zweite Ansatz stammt vom Datenbanklehrstuhl zusammen mit dem musikwissenschaftlichen Institut und beruht auf einer manuellen Extraktion der Handschriftcharakteristik mit anschließender automatischer Klassifizierung. Die vorliegende Diplomarbeit verfolgt genau diesen Ansatz.

Ausgangspunkt für diese Arbeit ist eine Hypothese der Musikwissenschaftler:

Jeder Schreiber hat eine für ihn typische, eindeutige Schreibercharakteristik, die auf einer Menge von Schriftmerkmalen basiert.

Diese Eigenschaft soll ausgenutzt werden, um die Handschriften von Notenschreibern automatisch zu klassifizieren. Die einzelnen Teilschritte, die zu diesem Ziel führen, werden durch den Titel der Arbeit gut umrissen:

Integration von Clustering-/Classification-Techniken in eine objektrelationale Datenbankumgebung

Der Titel gibt die Methoden und Techniken vor, die zur Umsetzung eines Schreibererkennungssystems untersucht und genutzt werden sollen. In dieser Arbeit werden die Data-Mining-Verfahren Klassifikation und Clustering daraufhin untersucht, inwieweit sie für die Schreiberhandklassifikation geeignet sind. Außerdem soll ein Prototyp entwickelt werden, der in eine objektrelationale Datenbankumgebung integriert werden kann.

Diese Diplomarbeit besteht aus zwei Teilen. Der Leser, der bereits mit dem Projekt eNoteHistory und den Data-Mining-Grundlagen vertraut ist, kann den ersten Teil ‘Grundlagen’ überspringen und gleich auf Seite 44 mit dem zweiten Teil ‘Konzeption und Umsetzung’ beginnen. Im Detail besteht der erste Teil aus drei Kapiteln. Kapitel 2 erläutert die musikwissenschaftlichen Grundlagen und jene aus dem Bereich des Data-Mining (DM). In Kapitel 3 werden der aktuelle Stand der Technik auf dem Gebiet der Handschrifterkennung und existierende DM-Tools vorgestellt. Der Grundlagenteil endet mit einer Beschreibung der Anwendungsmöglichkeiten und der Anforderungen an das zu entwickelnde Schreibererkennungssystem.

Der zweite Teil beginnt mit einer formalen Beschreibung der musikwissenschaftlichen Grundlagen und der geforderten Eigenschaften des Schreibererkennungssystems in Kapitel 5. Kapitel 6 zeigt dann, mit welchen Strukturen und Algorithmen diese Forderungen und Eigenschaften praktisch umgesetzt werden können. In Kapitel 7 wird die Implementierung dieser Strukturen und Algorithmen in Form eines Prototypen dargestellt. Der Prototyp wird in Kapitel 8 getestet und das zugrunde liegende Modell mit Hilfe der Testergebnisse optimiert. Die Diplomarbeit endet mit einer Zusammenfassung und einem Ausblick auf den Fortgang des Projektes in Kapitel 9. Um dem Leser das Verständnis der Arbeit zu erleichtern, befindet sich in Anhang A.8 auf Seite 105 eine Übersicht aller Abkürzungen und Symbole, die im Folgenden benutzt bzw. eingeführt werden.

Teil I

Grundlagen

Kapitel 2

Grundlagen

2.1 Musikwissenschaftliche Grundlagen

In diesem Abschnitt sollen die Begriffe und Zusammenhänge aus dem Bereich der Musikwissenschaft eingeführt werden, die notwendig sind, um den Inhalt dieser Diplomarbeit nachvollziehen zu können. In Kapitel 5.2 wird daraus eine formale Beschreibung abgeleitet, die für die Formulierung und Untersuchung von Verfahren zur Schreibererkennung geeigneter ist.

Für dieses Projekt liegt eine Sammlung von Notenblättern vor, die so genannte ‘Rostocker Sammlung’ [42]. Dabei handelt es sich um Notenblätter aus verschiedenen Gebieten Europas, die hauptsächlich aus dem 18. Jahrhundert stammen. In dieser Zeit war es typisch, Noten per Hand zu kopieren. Diese Arbeit wurde von so genannten Schreibern oder Kopisten erledigt. Die Namen der Kopisten sind nur selten bekannt. Darum ist es interessant, Schreiber anhand ihrer Schrift zu identifizieren. Vom Kopisten sind Rückschlüsse auf seine Nähe zum Komponisten, den Gebrauchszusammenhang der Materialien (Vorlage für weitere Kopien, Material zum Studium oder zur Aufführung) und andere Kopisten möglich. Normalerweise schreibt ein Kopist für mehrere Komponisten. Es gibt aber solche, die dem Komponisten sehr nahe stehen und nur für ihn schreiben. Basierend auf allgemeinen Merkmalen, wie zum Beispiel dem Papier, können Rückschlüsse auf die regionale Nähe zweier Kopisten möglich sein. Anhand der Notenschreiber können darüber hinaus Aussagen über die Verbreitung und die Wege der Verbreitung eines Stückes geschlossen werden.

Die Schreiber sollen aufgrund einer für sie typischen Schreibercharakteristik unterschieden werden. Diese setzt sich aus einer Menge von Schriftmerkmalen zusammen, die im folgenden Abschnitt erläutert werden.

2.1.1 Schriftmerkmale

Die Notenschrift versucht, Musik lesbar zu fixieren. Sie beschreibt die verschiedenen Parameter der Musik mit unterschiedlichen Mitteln: Tonhöhe und Tondauer durch Höhe und Form der Noten, Tempo, Lautstärke, Ausdruck, Artikulation usw. durch zusätzliche Zeichen und Wörter [4]. Eine individuelle Notenschrift wird durch diese verschiedenen Eigenschaften charakterisiert. Dazu



Abbildung 2.1: Verschiedene Noten und Pausen

gehören einerseits, die Art und Weise wie der Schreiber bestimmte Noten, Pausen und andere Symbole schreibt, aber andererseits auch darüber hinaus gehende Merkmale, wie zum Beispiel Informationen über das Papierformat. Ein Merkmal, das der Beschreibung einer Handschrift dient, wird im folgenden auch **Feature** genannt.

Es gibt ungefähr 80 verschiedene Merkmale, zum Beispiel das Feature 'Papierformat' oder das Feature 'C-Schlüssel', welches darüber informiert, wie der Kopist den C-Schlüssel schreibt.

Die Merkmale können in dreizehn Gruppen eingeteilt werden, die nun kurz vorgestellt werden.

Form der Notenköpfe Die Form der Note bestimmt die Tondauer. Ausgangspunkt für die Einteilung der Notenwerte ist die Ganze Note. Kleinere Notenwerte haben Hälse rechts aufwärts oder links abwärts und darüber hinaus die im folgenden Punkt beschriebenen Fähnchen. Abbildung 2.1 zeigt in der oberen Reihe verschiedene Noten. Notenköpfe unterscheiden sich darin, ob sie ausgefüllt sind (schwarz) oder nicht (weiß), ob der Kreis geschlossen ist und welche Form der Kreis hat. Die oben genannte Lage der Hälse entspricht dem modernen Standard. Gerade in diesem Punkt gab es bis ins 18. Jahrhundert viele Freiheiten, die auf den Schreiber hindeuten können.

Fähnchen Kleinere Notenwerte haben Hälse, an dessen Enden sich Fähnchen befinden. Diese sind stets nach rechts gerichtet. In Abbildung 2.1 sind die Fähnchen gut zu erkennen. Die Achtelnote hat ein, die Sechzehntel zwei Fähnchen usw. Die Fähnchen unterscheiden sich zum Beispiel bei den verschiedenen Schreibern in ihrer Form und dem Winkel zum Notenhals. Darüber hinaus gilt wie bei den Notenköpfen, dass es noch im 18. Jahrhundert mehr Freiheiten bei der Gestaltung der Fähnchen gab, als dies heute der Fall ist.

Bebalkung Noten mit Fähnchen können nach gewissen Regeln durch Balken zu Gruppen verbunden werden. Anstatt dass jede Achtelnote ein einzelnes Fähnchen bekommt, werden einfach alle aufeinanderfolgenden durch einen



Abbildung 2.2: G-Schlüssel, C-Schlüssel und F-Schlüssel

Balken verbunden, bei Sechzehnteln entsprechend durch zwei Balken. Die Lage und Form dieser Balken variiert von Kopist zu Kopist.

Pausen Eine Pause beschreibt eine bestimmte Zeitdauer, in der keine Musik erklingt. Jedem Notenwert entspricht ein Pausenwert. Abbildung 2.1 veranschaulicht dies. Entsprechend einer Ganzen Note gibt es auch eine Ganze Pause. Die Pause, welche die größte Individualität in Bezug auf die grafische Gestalt zulässt, ist die Viertelpause. Darum wird auch nur sie zur Schreibererkennung benutzt.

Vorzeichen Vorzeichen dienen der Veränderung der Tonhöhe (Alteration). Sie können entweder am Anfang oder direkt vor einer einzelnen Note stehen. Vorzeichen sind das Be (b), das Doppel-Be (bb), das Kreuz (\sharp) und das Doppelkreuz. Das Auflösungszeichen (\natural) macht die Veränderung rückgängig. Ein Schreiber kann daran erkannt werden, wie er die Zeichen selbst schreibt und wie er sie mit anderen Symbolen verbindet.

Schlüssel Der Schlüssel ist ein Zeichen, das am Beginn des Notensystems steht, um die Tonhöhe einer seiner Linien anzugeben und durch diesen Bezugspunkt alle übrigen Positionen des Systems festzulegen. Es werden der C-, F- und G-Schlüssel betrachtet. Abbildung 2.2 zeigt jeweils eine Variante dieser Schlüssel. Der Beginn des F- und G-Schlüssels und die Mitte des C-Schlüssels markieren jeweils die entsprechende Linie. Schlüssel sind die komplexesten Objekte und es gibt viele Variationen von ihnen. Darum werden sie bei der Schreiberidentifikation eine entscheidende Rolle spielen.

Taktvorzeichen Taktvorzeichen geben die nachfolgende Taktart an, zum Beispiel $\frac{4}{4}$ -Takt oder $\frac{3}{4}$ -Takt. Es werden die Symbole '1', '2', '3', '4', '8' und 'C' (als Abkürzung für den $\frac{4}{4}$ -Takt) betrachtet. Außerdem ist die Form des Bruchstrichs von Bedeutung, sowie die Frage, ob er überhaupt geschrieben wird.

Schriftneigung In der Merkmalsgruppe Schriftneigung wird für jede Notenart beschrieben, wie sich ihr Hals neigt.

Kaudierung In dieser Gruppe wird für die verschiedenen Notenwerte angegeben, an welcher Stelle der Notenhals am Notenkopf sitzt. Der Hals kann sich rechts, links oder auch mittig befinden. Die Notenwerte werden getrennt nach aufwärts und abwärts angesetzten Notenhälsen betrachtet.

Schlusszeichen Schlusszeichen markieren das Ende eines Werkes oder eines Teils von einem Werk (Satz). Sie bestehen im Notendruck aus einer dünnen und einer dicken senkrechten Linie, die durch das System gezogen werden. Kopisten verzieren dieses Zeichen häufig individuell. Zuweilen bringen sie dort auch ihre Initialen unter. Die Art dieser Verzierung kann gelegentlich einen Hinweis auf die Identität des Kopisten liefern.

Laufweite Hierbei geht es darum, wie weit die Noten voneinander entfernt sind. Es muss die Frage beantwortet werden, ob der Notenabstand enger, gleich oder weiter als die Notenkopfgröße ist.

Rastral Das Rastral ist ein Gerät zum Ziehen von Notenlinien. Es enthält fünf (oder Vielfache von fünf) Ausziehfedern, so dass mehrere Linien gleichzeitig gezogen werden können. Zur Schreibererkennung werden Eigenschaften des Rastrals bestimmt, beispielsweise der Abstand der äußeren Linien und damit die Breite des Rastrals.

Schreibgewohnheiten Zu den Schreibgewohnheiten zählen zum Beispiel das Papierformat, das Vorhandensein von Schlüsseln und Tonartvorzeichen und die Frage, ob Kustoden vorhanden sind. Kustoden sind früher gebräuchliche Zeichen, welche am Ende der Zeile die erste Note der folgenden Zeile anzeigen.

Mehr Informationen zu den Merkmalen sind in [4], [3], [6] und [56] zu finden. Eine ausführliche Liste aller Merkmale liegt im Anhang A.1 vor. Die Merkmale werden allerdings nicht nur in die dreizehn Gruppen aufgeteilt, sondern zusätzlich in jeder Gruppe in hierarchischer Form angeordnet. Diese Gruppen und Hierarchien bilden zusammen die **Feature Base**, die auch **Feature Dictionary** genannt wird.

2.1.2 Feature Base (Feature Dictionary)

In der Feature Base werden zum Beispiel alle Fähnchen zu einer Hierarchie zusammengefasst. Dort wird dann weiter zwischen Achtel- und Sechzehntelfähnchen unterschieden. Diese beiden Merkmale werden jeweils noch in aufwärts und abwärts kaudiert aufgeteilt. Dadurch entsteht die in Abbildung 2.3 dargestellte Hierarchie.

Die Abbildung zeigt die vier Merkmale, welche zur Gruppe der Fähnchen gehören:

1. 4.1.1. Achtelfähnchen aufwärts
2. 4.1.2. Achtelfähnchen abwärts
3. 4.2.1. Sechzehntelfähnchen aufwärts
4. 4.2.2. Sechzehntelfähnchen abwärts

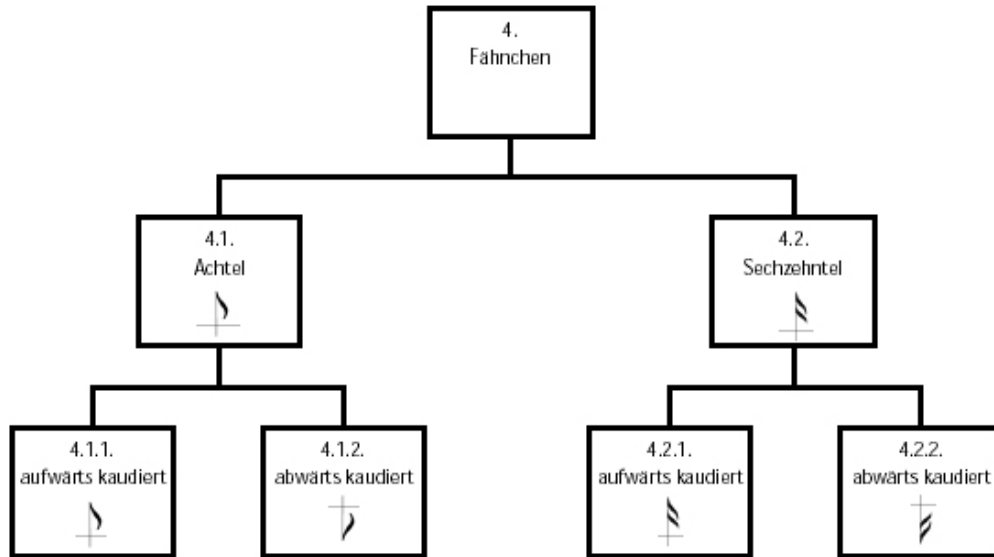


Abbildung 2.3: Die Merkmalsgruppe der Fähnchen.

Jedes dieser vier Merkmale beschreibt die Fähnchen in dem entsprechenden Fall. Auf diese Weise verteilen sich die zur Zeit 78 Merkmale auf die 13 Feature-Gruppen, die zusammen die Feature-Base bilden.

An dieser Stelle sei auch auf die Nummerierung der Merkmale hingewiesen. Sie beginnt mit der Nummer der Gruppe, in der das Feature liegt (siehe Abschnitt 2.1.1). Für alle Fähnchen-Features ist das die Vier. Die nächste Zahl bezieht sich auf die darunter liegende Hierarchieebene, wo die möglichen Alternativen durchnummeriert sind. Nach diesem Prinzip wird weiter verfahren, bis das entsprechende Feature in der Hierarchie erreicht ist. Diese Notation - im Folgenden **Punktnotation** genannt - beschreibt also den Pfad von der Wurzel in der Hierarchie bis hin zu einem Feature. Der Name eines Merkmales in Punktnotation wird **Feature-Code** oder **Feature-Pfad** genannt.

Die Feature Base enthält nicht nur die Merkmale selbst, sondern auch die möglichen Werte, die jedes einzelne Feature annehmen kann.

Wertebereich der Features

Die Menge aller möglichen Werte ist - analog zu den Features selbst - hierarchisch aufgebaut. Die Wertehierarchie ist darauf ausgerichtet, die manuelle Handschrifterkennung so einfach wie möglich zu gestalten. Die Werte für das Feature '4.1.1. Achtfähnchen aufwärts' sind sehr einfach und liegen alle auf einer Ebene. Sie geben an, wie der Kopist das Fähnchen geschrieben hat. Abbildung 2.4 veranschaulicht dies. Dort ist auch zu erkennen, dass die Wertehierarchie direkt unter dem Feature liegt und somit die Feature-Hierarchie fortsetzt. Die Punktnotation für einen Wert beginnt mit der Wurzel der Feature-Hierarchie. So kommt es, dass alle Werte eines Features in ihrem Pfadnamen in Punktnotation den Feature-Code als Präfix enthalten. Zum Beispiel besitzen alle Werte des Merkmales '4.1.1. Achtfähnchen aufwärts' in Abbildung 2.4 das

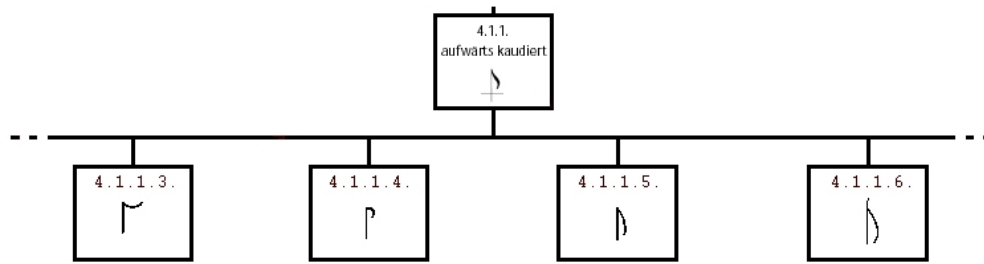


Abbildung 2.4: Die Werte des Merkmales 4.1.1.

Präfix ‘4.1.1.’

Die Mehrheit der Wertebereiche wird aber nicht wie bei den Fähnchen durch eine Ebene dargestellt, sondern durch Hierarchien mit mehreren Ebenen. In einigen Fällen folgen diese Hierarchien dem Verlauf beim Schreiben eines Symbols. Streng genommen darf nur beim G-Schlüssel von ‘folgen’ gesprochen werden, da nur dessen Hierarchie dem Schreibverlauf wirklich folgt. Andere Merkmale beschreiben lediglich Teilstücke eines Symbols. Die Beschreibung einer Viertelpause hat zum Beispiel drei Ebenen. Eine Ebene beschreibt den oberen, eine den mittleren und die letzte den unteren Teil der Viertelpause. Beim Bestimmen des Wertes muss der Nutzer zuerst den Wert auf der obersten Ebene bestimmen, dann in dem entsprechenden Teilbaum eine Ebene tiefer fortfahren und so weiter. Je komplexer ein Symbol ist, desto größer wird auch die Hierarchie. Zur Beschreibung eines G-Schlüssels werden beispielsweise bis zu 18 Ebenen benötigt. Wenn ein vorliegender Kopist allerdings einen sehr einfachen G-Schlüssel schreibt, der wenig Verzierungen hat, kann die Bestimmung auch schon einige Ebenen vor der 18. beendet werden, da die tieferen Ebenen Details beschreiben, die dieser Kopist nicht schreibt. Die Feature-Werte müssen also nicht immer die maximale Anzahl an Ebenen umfassen.

Auf die in diesem Kapitel beschriebene Weise kann jedem Symbol ein Wert in der Feature Base zugeordnet werden. Nach der Hypothese, auf der diese Arbeit beruht, unterscheidet die Menge dieser Merkmale einen Kopisten von allen anderen. Die Feature Base wird in Abschnitt 5.2 formal eingeführt. Im folgenden Kapitel werden die Grundlagen aus dem Bereich des Data Mining eingeführt.

2.2 Data Mining

In den letzten Jahrzehnten hat sich die Informationstechnologie enorm entwickelt. In allen Bereichen des täglichen Lebens werden Informationen digital abgespeichert. Dabei entstehen oft riesige Datenmengen, die wertvolle Informationen in Form von impliziten Mustern und Vernetzungen enthalten können, die nicht einfach zu durchschauen sind.

Unter dem Schlagwort **Data Mining**¹ (kurz: DM) werden Modelle, Da-

¹Eine adäquate Übersetzung für Data Mining ist schwer zu finden. Das englische Wort ‘mining’ bedeutet soviel wie ‘Bergbau’, ‘Bergbau betreiben’, ‘graben’. Ein Übersetzungsversuch wie ‘Nach Daten graben’ deutet den Umfang von Data Mining nur an. In der Literatur (z.B.

Instanz	Schreiber	Schriftgröße	Schriftzug geschlossen	Neigung
γ_1	Schreiber 1	9	Nein	Links
γ_2	Schreiber 1	8	Ja	Links
γ_3	Schreiber 1	7	Ja	Links
γ_4	Schreiber 2	7	Nein	Rechts
γ_5	Schreiber 2	5	Ja	Rechts
γ_6	Schreiber 2	6	Nein	Rechts

Tabelle 2.1: Konstruiertes Beispiel mit Schriftmerkmalen von zwei Schreibern

tenstrukturen und Algorithmen zusammengefasst, die dem Ziel dienen, aus großen Datenbeständen sinnvolle Zusammenhänge zu extrahieren, die vorher noch nicht bekannt waren. Damit können Abhängigkeiten oder Ähnlichkeiten der Datensätze erkannt und Vorhersagen getroffen werden.

Diese Arbeit beschäftigt sich mit der Einordnung von Handschriften, deshalb spielt hier besonders das Erkennen von Ähnlichkeiten eine Rolle. Zwei Teilgebiete, auch Konzepte genannt, die dies leisten, sind hier von besonderem Interesse:

- Das Clustering und
- Die Klassifikation.

Diese zwei Data Mining-Konzepte werden in den folgenden Abschnitten eingeführt.

Zuvor soll aber ein Beispiel vorgestellt werden, mit dessen Hilfe die verschiedenen Verfahren illustriert werden können. Es handelt sich dabei um einen konstruierten Datensatz, der zwei Schreiber und drei ihrer typischen Eigenschaften enthält. Er hat noch gar nichts mit den Datensätzen gemein, um die es in dieser Arbeit gehen wird, sondern enthält wenige Werte, die gut geeignet sind, die DM-Verfahren zu veranschaulichen.

Tabelle 2.1 zeigt die Merkmale, die über die beiden Schreiber bekannt sind. Es handelt sich dabei um die Größe ihrer Schrift, gemessen an einem bestimmten Buchstaben, die Geschlossenheit des Schriftzuges, die angibt, ob der Schreiber den Stift innerhalb eines Wortes abgesetzt hat, und schließlich, die Neigung der Schrift. Es ist deutlich zu erkennen, dass die Schriftgrößen bei beiden Schreibern zwar variieren, aber sich doch klar voneinander unterscheiden, da der erste Schreiber eine größere Schrift hat als der zweite. Die Geschlossenheit des Schriftzuges ist nicht ganz eindeutig, jedoch ist die Tendenz zu erkennen, dass der erste Schreiber im Beispiel eher zu geschlossenen Schriftzügen neigt, währenddessen Schreiber 2 den Stift des Öfteren absetzt. Die Schriftneigung hingegen kann uneindeutig den Schreibern zugeordnet werden.

Anhand dieses Beispiels wird im Folgenden gezeigt, wie die Eingabedaten repräsentiert werden können, es werden Klassifikations- und Clustering-Grundlagen vermittelt und abschließend einige grundlegende Data-Mining-Konzepte beschrieben.

[19]) taucht sogar das Wort 'minieren' als deutsches Äquivalent zu 'mining' auf.

2.2.1 Datenrepräsentation

Die Eingabe für einen Data-Mining-Algorithmus ist eine Menge von Datensätzen, die auch **Instanzen**² heißen. Diese Instanzen sollen auf Zusammenhänge hin untersucht werden. Man kann sie sich als Zeilen einer Tabelle vorstellen. Eine Instanz wird in dieser Arbeit als ein Vektor

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

dargestellt, wobei v_1 bis v_n die einzelnen Werte dieser Instanz sind. Die Werte werden jeweils einem **Attribut** (auch **Feature** genannt) zugeordnet. Ein Attribut entspricht einer Tabellenspalte.

Zur Veranschaulichung der Daten wird in dieser Arbeit folgendes Modell benutzt. Die Wertebereiche der Attribute werden auf je eine Skala abgetragen. Die Skalen bilden die Achsen eines n-dimensionalen Raumes. Es wird also durch n Attribute ein n-dimensionaler Raum aufgespannt. Instanzen, die als Vektoren notiert werden, bestimmen genau einen Punkt in diesem Raum. Eine Operation auf diesen Vektoren wird von ganz besonderem Interesse sein: Die Abstandsfunktion, auch Distanzfunktion genannt.

Es existieren verschiedene **Distanzfunktionen**. Die Wahl der besten muss von Fall zu Fall getroffen werden. Sie ist von der Art der Eingabedaten, ihren Datentypen und der allgemeinen Problemstellung abhängig. Bekannte Abstandsfunktionen ($d(\gamma_1, \gamma_2)$) für die Feature-Vektoren $\gamma_1 = (v_1^1, \dots, v_n^1)$ und $\gamma_2 = (v_1^2, \dots, v_n^2)$ sind:

$$\text{Hamming-Distanz: } d(\gamma_1, \gamma_2) = |v_1^1 - v_1^2| + \dots + |v_n^1 - v_n^2|$$

$$\text{Euklidische Distanz: } d(\gamma_1, \gamma_2) = \sqrt{(v_1^1 - v_1^2)^2 + \dots + (v_n^1 - v_n^2)^2}$$

In Variationen dieser Formeln können die einzelnen Summanden Gewichte bekommen, was die entsprechenden Attribute höher gewichtet. Außerdem kann die Distanz normiert werden, zum Beispiel auf den Bereich $[0,1]$. Eine formale, tiefer gehende Diskussion der Distanzfunktion, die für die vorliegende Arbeit geeignet ist, folgt in Kapitel 5.2.3.

Nachdem gezeigt wurde, wie die Daten repräsentiert werden können, sollen nun konkrete Data-Mining-Verfahren vorgestellt werden. Zuerst werden Klassifikationsverfahren und im Anschluss Clustering-Verfahren beschrieben. Gegen Ende des Kapitels werden dann einige wichtige Basismethoden erläutert, die im Data Mining angewendet werden.

²In [57] heißt es 'individuelle, unabhängige Instanzen'. Diese Eigenschaft soll hier aber nicht weiter vertieft werden.

2.2.2 Klassifikation

Die Ausgangslage bei der Klassifikation ist, dass eine Menge von Instanzen vorliegt, von denen bekannt ist, zu welcher Klasse sie gehören. Die Aufgabe von Klassifikationsalgorithmen ist es, herauszufinden, warum genau diese Instanzen in genau dieser Klasse liegen. Anders gesagt sollen die Eigenschaften der Instanzen gefunden werden, aufgrund derer sie zu einer bestimmten Klasse gehören. Die Phase, in der sich das System das Wissen aneignet, wird Test- oder Lernphase genannt. Dazu wird eine Trainingsmenge mit Instanzen benötigt, aus denen automatisch die signifikanten Eigenschaften abgeleitet werden.

Das Beispiel aus Tabelle 2.1 entspricht der Frage, wieso die Instanzen γ_1 , γ_2 und γ_3 vom ersten Schreiber sind und was typisch für die Charakteristiken des zweiten Schreibers (γ_4 , γ_5 und γ_6) ist. Die Trainingsmenge, anhand der das Data-Mining-System die Regeln ableiten soll, ist hier die Menge $\{\gamma_1, \dots, \gamma_6\}$. Mit Hilfe der berechneten Regeln kann dann für neue Instanzen, deren Klassenzugehörigkeit unbekannt ist, die entsprechende Klasse bestimmt werden.

Es gibt viele Möglichkeiten dieses Wissen darzustellen. Bevor einige Klassifikationsalgorithmen vorgestellt werden, soll zuerst erläutert werden, welche Arten der Wissensrepräsentation existieren.

Wissensrepräsentation

Das Ergebnis der Klassifikationsverfahren soll ein Regelwerk sein, das beschreibt, wie die Instanzen den Klassen zugeordnet werden. Dafür gibt es verschiedene Strukturen. Die bekanntesten sind:

- Klassifikationsregeln
- Entscheidungsbäume
- die Repräsentation durch Hyperebenen im n-dimensionalen Raum und die
- Instanzbasierte Repräsentation.

Eine **Klassifikationsregel** besteht aus einer Menge von Bedingungen, die mit dem logischen Operator AND verknüpft werden und aus einer Schlussfolgerung, wie zum Beispiel:

IF Schriftgröße = 7 AND Schriftzug geschlossen = Ja THEN Schreiber 1

Die Ausgabe eines Klassifikationsalgorithmus kann nun eine Menge solcher Regeln sein. Es werden immer die Regeln gefeuert, deren Bedingungen zutreffen. Dabei kann es passieren, dass zwei Regeln widersprüchliche Schlussfolgerungen ziehen. In diesem Fall können entweder beide ignoriert werden, oder es wird die populärere gefeuert. Die populärere Regel ist jene, welche auf den Trainingsdaten häufiger angewendet wurde. Die beiden Möglichkeiten führen natürlich zu unterschiedlichen Ergebnissen.

Die Klassifikationsregeln können noch um Ausnahmen und Relationen erweitert werden. Das Beispiel von oben könnte auch mit Hilfe einer Ausnahme

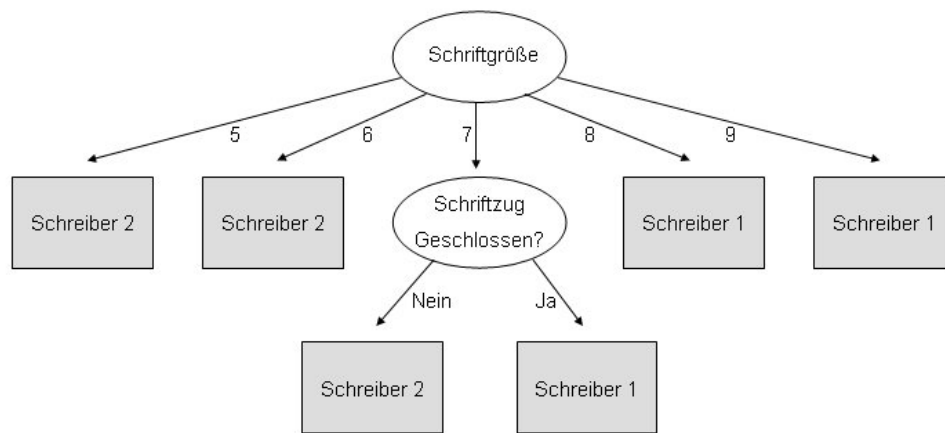


Abbildung 2.5: Ein (nicht optimaler) Entscheidungsbaum

(engl. Exception) formuliert werden:

IF Schriftgröße = 7 THEN Schreiber 1 EXCEPT Schriftzug geschlossen = Nein

Das bedeutet, dass Schriften mit der Schriftgröße 7 vom ersten Schreiber stammen, es sei denn, der Schriftzug ist nicht geschlossen.

Relationen sind Vergleiche von zwei Attributen. Bisher waren in Bedingungen nur Vergleiche von Attributen und ihren Werten erlaubt. Es können selbstverständlich nur passende Features miteinander verglichen werden. Für die Schriftgröße und ihre Neigung ist das nicht möglich. Relationen können in einem Datensatz sinnvoll sein, in dem es zwei Attribute für einen Minimal- und einen Maximalwert gibt. Dann wäre folgende Regel möglich:

IF Minimum = Maximum THEN Klasse 1

Ein **Entscheidungsbaum** ist ein Baum, an dessen Knoten die Attribute liegen. An den Knoten werden die Features auf ihren Wert hin getestet und je nach Ausgang des Tests wird eine Kante zu einem Kindknoten verfolgt. Die Bedingungen dafür werden an den entsprechenden Kanten notiert. Knoten von Attributen mit nominalem Wertebereich haben üblicherweise genauso viele Kindknoten wie mögliche Werte.

Die Blätter des Baumes stellen dann jeweils eine Klasse dar, entsprechen also der Schlussfolgerung von Klassifikationsregeln. Abbildung 2.5 zeigt einen möglichen Entscheidungsbaum für das Beispiel aus Tabelle 2.1. Für jede Instanz, für die die Klasse bestimmt werden soll, in die sie gehört, wird der Baum von der Wurzel zu den Blättern hin durchlaufen und an jedem Knoten das Attribut getestet. Das Blatt, in dem der Durchlauf endet, gibt dann über die Klasse Auskunft.

Attribute	Regeln	Fehler	Totaler Fehler
Schriftgröße	5,6 → Schreiber 1	0/2	1/6
	7 → Schreiber 1	1/2	
	8,9 → Schreiber 2	0/2	
Schriftzug geschlossen	Ja → Schreiber 1	1/3	2/6
	Nein → Schreiber 2	1/3	
Neigung	Links → Schreiber 1	0/3	0/6
	Rechts → Schreiber 2	0/3	

Tabelle 2.2: Untersuchung der Attribute des Schreiberbeispiels

Der Entscheidungsbaum in Abbildung 2.5 wurde zur Veranschaulichung konstruiert. Von einem Klassifikationsverfahren, das so einen Baum erzeugen würde, ist abzuraten. Der optimale Entscheidungsbaum für das Beispiel hätte als Wurzelknoten das Attribut Neigung, als ersten Kindknoten für den Wert ‘Links’ die Klasse ‘Schreiber 1’ und als zweiten Kindknoten die Klasse ‘Schreiber 2’.

Eine abstraktere Möglichkeit, die Ausgabe eines Klassifikationsalgorithmus darzustellen, basiert auf der Darstellung der Instanzen als Punkte im **n-dimensionalen Raum**, wie sie bereits in Abschnitt 2.2.1 beschrieben wurde. Dort können Hyperebenen bestimmt werden, die den Raum in mehrere Teilräume aufteilen. Diese Teilräume entsprechen den Klassen. Sie können maximal n-dimensional sein, aber auch nur einzelne Punkte darstellen.

Als letzte Form der Wissensrepräsentation sei an dieser Stelle die **instanzbasierte Repräsentation** genannt. Dafür muss kein Algorithmus aus einer Trainingsmenge Regeln ableiten, sondern die Instanzen selbst dienen zur Beschreibung der Wissensbasis. Die Klasse einer neuen Instanz wird bestimmt, indem die im n-dimensionalen Raum nächstgelegene Instanz zu Rate gezogen wird. Eine genauere Beschreibung des Verfahrens ist im nächsten Abschnitt zu finden.

Nachdem die grundlegenden Strukturen der Klassifikation eingeführt wurden, sollen im Folgenden Verfahren vorgestellt werden, die diese Wissensrepräsentanten erzeugen können.

Klassifikationsverfahren

Das einfachste Verfahren heißt ‘**1-rule**’ (**1R**). Es erstellt eine Regel, die die Klasse nur mit Hilfe eines Attributs bestimmt. Das Verfahren erzielt oft gute Ergebnisse. Das mag daran liegen, dass in der echten Welt Zusammenhänge häufig sehr einfach sind und nur von einem Parameter abhängen.

Die Idee ist, Regeln für genau ein Attribut zu erstellen, wobei jedem Wert die Klasse zugeordnet wird, in der dieser Wert am häufigsten vorkommt. Für jede Regel wird der Fehler berechnet. Dieser ergibt sich aus dem Quotienten der nicht korrekt zugeordneten Instanzen und der Anzahl aller Instanzen, die für das Attribut den betrachteten Wert haben. Daraus lässt sich dann der Totale Fehler für die Regeln ableiten, die für das Feature bestimmt wurden. Als Regeln werden jene gewählt, die den kleinsten Totalen Fehler haben.

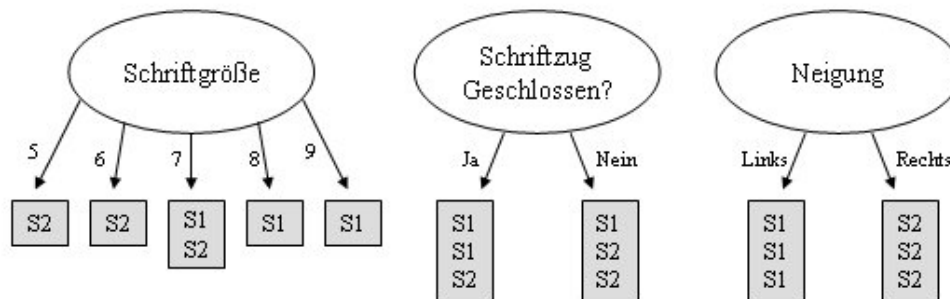


Abbildung 2.6: Mögliche Wurzeln des Entscheidungsbaums

Tabelle 2.2 zeigt die Werte für das Schreiberbeispiel. Zu Erklärung diene die zweite Regel der ersten Zeile 'Schriftgröße'. Es existieren zwei Instanzen (γ_3, γ_4) , welche die Schriftgröße '7' haben. Eine Instanz gehört in die Klasse 'Schreiber 1' die andere in 'Schreiber 2'. Es muss also eine der beiden Klassen zufällig ausgewählt werden, in Tabelle 2.2 ist das der erste Schreiber. Der Fehler beläuft sich damit auf $1/2$, denn es wird genau eine von zwei Instanzen falsch zugeordnet.

Das Attribut mit dem kleinsten Fehler ist die Neigung. Diese beiden Regeln wären darum die Ausgabe des Algorithmus'. In diesem Beispiel kann allein dieses Attribut die Klassenzugehörigkeit fehlerlos bestimmen.

Als nächstes soll ein Verfahren zur Erzeugung von Entscheidungsbäumen betrachtet werden. Es wurde von Ross Quinlan unter dem Namen **ID3** (Induction of Decision Trees, [50]) entwickelt.

Wenn ein Baum von der Wurzel her aufgebaut wird, stellt sich natürlich jeweils die Frage, welches Attribut als nächstes in den Baum eingefügt werden soll. Oder anders ausgedrückt, welches Feature die Menge der Instanzen anhand von Wertevergleichen am besten aufteilt. Abbildung 2.6 zeigt, wie die einzelnen Attribute als Wurzel des Baumes aussehen würden. Das Attribut 'Neigung' teilt die Menge der Instanzen in zwei reine Teilmengen auf, jeweils für 'Schreiber 1' und 'Schreiber 2'. Im Falle des Attributs 'Schriftzug geschlossen?' müsste an beiden entstandenen Knoten noch einmal gesplittet werden - solange, bis nur reine Knoten entstehen.

Die Entscheidung über die Wahl des Attributs wird mit Hilfe der Reinheit (auch Informationswert) der entstehenden Knoten getroffen. Dieser Wert ist gleich null, wenn nur eine einzige Klasse in der Menge enthalten ist und sie damit von maximaler Reinheit ist. Der Wert ist gleich eins, wenn die Anzahl aller enthaltenen Klassen gleich ist. Daraus wird für jedes Attribut der so genannte Gewinn (engl. gain) berechnet. Das Attribut, das den größten Gewinn verspricht, wird als nächster Knoten gewählt. Die Formeln sollen an dieser Stelle nicht weiter vertieft werden. Sie können zum Beispiel in [57] nachgeschlagen werden. Hier zum Vergleich die Werte für die Beispiele aus Abbildung 2.6:

$$\text{gain}(\text{Schriftgröße}) = 0,666$$

$$\begin{aligned} \text{gain}(\text{Schriftzug Geschlossen}) &= 0,082 \\ \text{gain}(\text{Neigung}) &= 1 \end{aligned}$$

Damit wird, wie nicht anders zu erwarten, das Attribut Neigung ausgewählt. Es wurde ja schon im vorherigen Abschnitt gezeigt, dass dieses Attribut die Klassen ohne Fehler voraussagt.

Eine Verbesserung dieses Verfahrens ist der Algorithmus **C4.5**³. Er wurde von Ross Quinlan über eine lange Periode (beginnend in den siebziger Jahren) entwickelt und in [51] beschrieben. C4.5 berücksichtigt zusätzlich numerische Attribute und Nullwerte. Darüber hinaus wendet er Pruning-Verfahren an, die den Baum wieder reduzieren. Dies kann sinnvoll sein, um Overfitting zu vermeiden (vgl. Abschnitt 2.2.4), also das Problem, dass der Entscheidungsbaum so speziell ist, dass er nur für die Trainingsmenge funktioniert, aber nicht mehr für neue Instanzen.

Ein weiteres, sehr mächtiges Klassifikationsverfahren stellt die **Support Vector Machine (SVM)** dar. Das Prinzip wurde von Boser, Guyon und Vapnik in [25] erstmals vorgestellt. Die bisherigen Modelle erlauben es, im durch die Attribute aufgespannten Raum die Cluster-Grenzen linear zu beschreiben. SVMs lassen darüber hinaus nichtlineare Grenzen zu.

Die Grundidee basiert jedoch zunächst auch auf einem linearen Modell. Für je zwei Klassen wird eine trennende Hyperebene⁴ gesucht. Die Instanzen mit der minimalen Distanz zur Hyperebene sind die Support-Vektoren. Alle anderen Instanzen der Klassen werden bei der Berechnung der Hyperebene nicht benötigt.

Nichtlineare Grenzen werden erzeugt, indem die ursprünglichen Eingabedaten mittels nichtlinearem Mapping transformiert werden. Der Raum, in dem die Instanzen liegen, wird damit in einen neuen transformiert und dort wird das eben vorgestellte Grundverfahren angewendet. Eine gerade Linie in dem neuen Raum erscheint im ursprünglichen nicht mehr gerade und somit sind nichtlineare Klassen-Grenzen möglich. Einen genaueren Einblick in die Thematik gibt [57] ab Seite 118.

Alle bisher vorgestellten Verfahren haben spezielle Entscheidungsstrukturen erstellt, um das gewonnene Klassifikationswissen zu speichern. Andere Verfahren tun dies nicht, sondern speichern einfach nur die Instanzen der Trainingsmenge, um später für neue Instanzen die Entscheidung der Klassenzugehörigkeit mit Hilfe dieser zu treffen. Solche Methoden werden instanzbasiert genannt. Hierbei wird die eigentliche Arbeit nicht im Vorhinein erledigt, sondern erst beim Klassifizieren neuer Instanzen. Ein solches Verfahren ist **K-Nearest-Neighbor**. Zur Bestimmung der Klasse einer Instanz werden dabei die Klassen ihrer k nächsten Nachbarn im n -dimensionalen Raum zu Rate gezogen. Dafür muss eine Distanzfunktion auf den Instanzen in diesem Raum definiert sein, wie sie in Abschnitt 2.2.1 eingeführt wurde. Verfahren für $k = 1$ und $k = 2$ wurden von Aha Anfang der neunziger Jahre in [20] und [21] vorgestellt. Es wird sich

³Es existiert ein kommerzieller Nachfolger: C5.0

⁴Diese wird Maximum Margin Hyperplane genannt und liegt senkrecht zur kürzesten Verbindungsgeraden beider Klassen. Siehe auch [57].

später zeigen, dass dieser Algorithmus für die vorliegende Diplomarbeit von entscheidender Bedeutung ist.

Nachdem die für diese Arbeit bedeutenden Klassifikationsverfahren umrissen wurden, sollen in nächsten Abschnitt Clustering-Methoden vorgestellt werden.

2.2.3 Clustering

Die Ausgangslage beim Clustering ist eine andere als bei der Klassifikation. Hier sind die Klassen der Instanzen nicht bekannt. Aufgabe von Clustering-Verfahren ist es, eine Menge von Instanzen automatisch in Klassen, die hier Cluster genannt werden, einzuteilen. Dafür sind zwei Eigenschaften von Clustern entscheidend:

- Instanzen eines Clusters liegen im Raum nah beieinander.
- Instanzen verschiedener Cluster haben einen großen Abstand.

Ziel des Clusterings ist es, eine Partition der Menge von Instanzen zu finden, so dass diese beiden Eigenschaften optimiert werden.

Algorithmen

MacQueen stellte 1967 das **K-Means**-Verfahren [46] vor. Dieses grundlegende Clustering-Verfahren teilt die Instanzen in k Cluster auf, die durch ihre Zentren repräsentiert werden. K-Means basiert auf einer Distanzfunktion $d(i, j)$, die den Abstand von zwei Punkten i und j im Raum angibt. Der Algorithmus läuft nun folgendermaßen ab:

Zu Beginn werden k Cluster-Zentren zufällig bestimmt.

1. Dann wird mit Hilfe der Distanzfunktion für jede Instanz der Cluster bestimmt, zu dessen Zentrum sie den geringsten Abstand hat. Die Instanz wird dann genau diesem Cluster zugeordnet.
2. Wenn jede Instanz einem Cluster zugewiesen wurde, wird das neue Zentrum (engl.: mean) des Clusters aus allen enthaltenen Instanzen berechnet. Dieser Wert ersetzt den alten Wert, der den Cluster repräsentierte.
3. Die Schritte 1) und 2) werden nun solange ausgeführt, bis die resultierenden Cluster stabil bleiben.

Auf diese Weise wird allerdings nur ein lokales Minimum erreicht. Es ist nicht sichergestellt, dass so die global beste Lösung gefunden wird. Ein beliebtes Beispiel, das diese Eigenschaft auf einfache Weise verdeutlicht, sind vier Punkte, die die Eckpunkte eines nicht-quadratischen Rechtecks bilden. Wenn zwei Cluster berechnet werden sollen, ist zu erwarten, dass die jeweils gemeinsam an einer der beiden kürzeren Kanten des Rechtecks liegenden Punkte in einen Cluster fallen. Werden die zufällig bestimmten Cluster-Zentren aber genau auf die Mitten der beiden längeren Seiten gelegt, ergeben sich sofort zwei stabile Cluster, die jeweils die beiden Endpunkte der längeren Kanten enthalten.

	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6
a) C_1	0	2	3	5	8	6
C_2	8	6	5	3	0	2

	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6
b) C_1	2	0	1	4	5	5
C_2	5	5	4	1	2	0

Tabelle 2.3: Distanzen im ersten (a) und zweiten (b) Durchlauf des K-Means-Verfahrens

Das Verfahren soll mit dem Beispiel aus Tabelle 2.1 von Seite 18 veranschaulicht werden. Als Distanzfunktion diene eine Hamming-Distanz:

$$d(x, y) = |x_1 - y_1| + |x_2 - y_2| + 2|x_3 - y_3|$$

Die Differenz der nominalen Werte sei null, wenn sie identisch sind und sonst eins. Außerdem ist das dritte Attribut (Neigung) stärker gewichtet, da sonst einerseits die Schriftgröße die Distanz dominieren würde. Andererseits wurde ja schon gezeigt, dass die Schriftneigung ein sehr aussagekräftiges Feature ist. Die initialen Cluster-Zentren seien:

$$C_1 = (9, \text{Nein}, \text{Links})^T \text{ und } C_2 = (5, \text{Ja}, \text{Rechts})^T$$

Im ersten Durchlauf wird für jede Instanz der Abstand zu allen Clustern berechnet. Tabelle 2.3 (a) zeigt die berechneten Abstände. Für jede Instanz wurde der am nächsten liegende Cluster mit fetter Schrift markiert. Aus den Instanzen der Cluster werden die neuen Cluster-Zentren berechnet:

$$C_1 = (8, \text{Ja}, \text{Links})^T \text{ und } C_2 = (6, \text{Nein}, \text{Rechts})^T$$

Im nächsten Durchlauf ergeben sich die Abstände, die in Tabelle 2.3 (b) dargestellt sind. Die Cluster-Zuordnung hat sich seit dem letzten Durchlauf nicht mehr geändert und damit ist der Algorithmus beendet. Es wurden genau die Schriften als ähnlich erkannt, die auch von den gleichen Schreibern sind. Es wird aber auch deutlich, wie wichtig die Wahl der Distanzfunktion ist. Hätte zum Beispiel das zweite Attribut eine höhere Gewichtung bekommen, dann wäre auch eine andere Partition entstanden.

Um das Ergebnis des K-Means-Verfahrens zu verbessern, ist es möglich, mehrere Läufe mit verschiedenen Startkonfigurationen durchzuführen und das beste Ergebnis auszuwählen.

Ein anderer Ansatz, der auf der Distanz zwischen zwei Instanzen basiert, kommt aus dem Bereich der **hierarchischen Cluster-Bildung**. Dafür werden die Abstände zwischen allen Instanzen in eine Tabelle eingetragen. Am Anfang liegt jede Instanz in einem eigenen Cluster. Dann werden jeweils die beiden Cluster vereinigt, die den geringsten Abstand haben. Für den neu entstanden Cluster muss darauf hin der Abstand zu allen anderen Clustern neu berechnet werden. Eine einfache Lösung dafür ist es, den minimalen Abstand aller enthaltenen Instanzen zu denen des anderen Cluster zu wählen:

	{ γ_1 }	{ γ_2 }	{ γ_3 }	{ γ_4 }	{ γ_5 }	{ γ_6 }
{ γ_1 }	-	2	3	4	7	5
{ γ_2 }	2	-	1	4	5	5
{ γ_3 }	3	1	-	3	4	4
{ γ_4 }	4	4	3	-	3	1
{ γ_5 }	7	5	4	3	-	2
{ γ_6 }	5	5	4	1	2	-

	{ γ_1 }	{ γ_2, γ_3 }	{ γ_4, γ_6 }	{ γ_5 }
{ γ_1 }	-	2	4	7
{ γ_2, γ_3 }	2	-	3	5
{ γ_4, γ_6 }	4	3	-	2
{ γ_5 }	7	5	2	-

Tabelle 2.4: Abstandstabelle nach dem ersten (a) und dritten (b) Durchlauf des Single-Linkage-Verfahrens

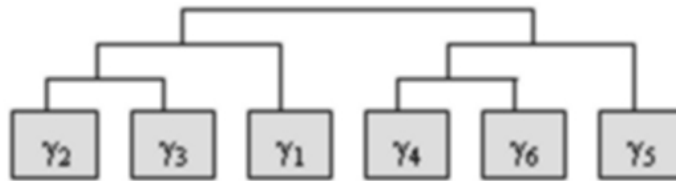


Abbildung 2.7: Dendrogramm für das Schreiberbeispiel

$$d(C_i, C_j) = \min_{x,y}(d(x,y) | x \in C_i, y \in C_j)$$

Diese Variante nennt sich **Single-Linkage**. Die Verschmelzungen von Clustern werden in einem Baum mitprotokolliert. Dieser Baum wird Dendrogramm genannt.

Für das Schreiberbeispiel mit der gleichen Distanzfunktion wie im vorherigen Abschnitt entsteht die Abstandstabelle aus Tabelle 2.4 (a). Die beiden Cluster-Paare $\{\gamma_2, \{\gamma_3\}$ und $\{\gamma_4, \{\gamma_6\}$ haben jeweils den minimalen Abstand. Es würde also erst ein Paar zu einem Cluster verschmolzen werden und dann das andere, so dass die neue Tabelle 2.4 (b) entsteht. Der Vorgang wird solange wiederholt, bis nur noch ein Cluster existiert und das Dendrogramm in Abbildung 2.7 entstanden ist.

Der Vorteil dieser Methode ist, dass im Vorhinein nicht feststehen muss, wie viele Cluster der Algorithmus erzeugen soll. Je nachdem, welche Ebene des Dendrogramms betrachtet wird, ergeben sich verschieden viele Cluster.

Während die bisherigen Verfahren iterativ auf der gesamten Datenmenge operieren, behandelt die folgende Methode die Instanzen nacheinander, also inkrementell. Darum spricht man auch vom **Inkrementellen Clustering**. Das Verfahren nutzt dabei keine Distanzfunktion, sondern die so genannte Kategoriennützlichkei, die nun eingeführt werden soll.

Kategoriennützlichkei

Die Kategoriennützlichkei (CU, Category Utility) wurde von Gluck und Corter im Jahr 1985 in [32] vorgestellt. Sie gibt die Qualität einer Partition von Instanzen in Cluster an. Die Grundidee ist, dass eine Eigenschaft eines Clusters darin besteht, die Werte der Attribute von Instanzen dieses Clusters vorauszusagen. Die folgende Formel basiert auf dieser Idee:

$$CU(C_1, C_2, \dots, C_k) = \frac{\sum_l P[C_l] \sum_i \sum_j (P[a_i=v_{ij} | C_l]^2 - P[a_i=v_{ij}]^2)}{k}$$

Die innere Doppelsumme wird über allen Attributen und ihren möglichen Werten berechnet. $P[a_i = v_{ij}|C_l]$ gibt die Wahrscheinlichkeit an, dass das Feature a_i im Cluster C_l den Wert v_{ij} annimmt. Es wird also die Frage beantwortet, wie gut der Wert eines Attributs aufgrund der Cluster-Zugehörigkeit vorhergesagt werden kann. Von dem Quadrat dieser Wahrscheinlichkeit wird das Quadrat von $P[a_i = v_{ij}]$ subtrahiert. Dieser Ausdruck beschreibt die Wahrscheinlichkeit, dass a_i irgendwann den Wert v_{ij} annimmt, nicht nur wenn es in einem bestimmten Cluster liegt. Sollte diese Differenz nun negativ sein, folgt daraus: $P[a_i = v_{ij}|C_l] < P[a_i = v_{ij}]$. Das bedeutet wiederum, dass der Wert für das Attribut auch mit Kenntnis des Clusters nicht besser vorhergesagt werden kann. Damit ist der Cluster selbst nicht sehr nützlich.

Die äußere Summe addiert die Werte für alle Cluster, gewichtet nach der Größe der Cluster. Das Ergebnis wird am Ende durch die Anzahl der Cluster (k) geteilt, weil sonst genau die Partition den größten Wert für CU und damit die beste Kategoriennützlichkeith hätte, die jede Instanz einem eigenen Cluster zuordnen würde.

Mit CU existiert ein Maß zur Bestimmung der Qualität einer Partition. Ein Algorithmus, der dieses Maß ausnutzt, wurde 1987 von Fisher [30] vorgestellt. Der Algorithmus ist Teil des von Fisher entwickelten Clustering-System COBWEB. Bei diesem Verfahren wird eine Hierarchie von Clustern erzeugt, indem die Instanzen nacheinander in einen Baum eingefügt werden. Beim Einfügen wird der Baum von der Wurzel zu den Blättern durchlaufen und für mögliche Positionen der einzufügenden Instanz die Kategoriennützlichkeith berechnet. Es werden darüber hinaus die Operationen Insert, Create, Merge und Split von Knoten des Baumes berücksichtigt. Dadurch erlaubt COBWEB, bereits erlerntes Wissen zu revidieren, wenn es die neuen Instanzen erfordern. So können zum Beispiel zwei Cluster, die zuvor getrennt wurden, mit der Merge-Operation wieder vereinigt werden.

Die vorgestellten Clustering-Verfahren sollen als Überblick über die Methoden, die in dieser Arbeit diskutiert werden, genügen. Weiterführende Literatur wird am Ende dieses Kapitels genannt. Im nächsten Abschnitt soll es darum gehen, einige wichtige, grundlegende Data-Mining-Techniken und Begriffe zu klären.

2.2.4 Allgemeine Data-Mining-Verfahren und Begriffe

Im Folgenden wird die Thematik des Overfitting eingeführt, Methoden zur Optimierung der Eingabedaten vorgestellt und Verfahren zur Bewertung der DM-Algorithmien und ihrer Ergebnisse erläutert.

Overfitting

In diesem Kapitel wurden Algorithmen anhand des Schreiberbeispiels (vgl. Tabelle 2.1) veranschaulicht. Die daraus resultierenden Regeln oder Entscheidungsbäume konnten jeweils fehlerlos auf die Trainingsmenge angewendet werden. Es kann aber leicht passieren, dass eine neue Instanz mit Hilfe der

Regeln nicht mehr klassifiziert werden kann, weil die Regeln zu speziell sind und nur für die Trainingsmenge funktionieren. Dieser Effekt wird **Overfitting** genannt. Im Schreiberbeispiel hat sich eine ganz einfache Regel herauskristallisiert:

IF Neigung = Links THEN Schreiber 1

Sollte dieser Schreiber aber einmal seine Schrift nicht nach Links neigen, würde er auch nicht mehr erkannt werden. Darum wurden in diesem Kapitel Verfahren vorgestellt, die dem Overfitting entgegenwirken. Weitere Verfahren werden im übernächsten Abschnitt erläutert.

Optimierung der Eingabedaten

Data-Mining-Algorithmen sind häufig so angelegt, dass sie mit den verschiedensten Arten von Eingabedaten arbeiten können. Die Behandlung von Nullwerten, die Unterstützung von nominalen und numerischen Attributen oder die Auswahl geeigneter Attribute sind meist schon in den Algorithmen enthalten.

Trotzdem kann es sinnvoll sein, die Eingabedaten schon vorher bezüglich dieser Anforderungen zu optimieren. Einige DM-Verfahren setzen beispielsweise voraus, dass die Attribute unabhängig voneinander sind. Solche Abhängigkeiten in den Eingabedaten können erkannt und durch Entfernen eines Attributs beseitigt werden.

Geschieht die Auswahl der besten Attribute vor der Ausführung des DM-Algorithmus', wird dies Filtermethode genannt, ansonsten Wrapper-Methode. Es gibt dafür mehrere Ansätze. Ein Ansatz ist es, die Ergebnisse von DM-Verfahren zu benutzen. Zum Beispiel kann die Menge der Features auf jene reduziert werden, die tatsächlich für den Entscheidungsbaum benutzt werden. Diese Einschränkung würde sich dann beispielsweise bei der Berechnung des nächsten Nachbarn bemerkbar machen.

Ein anderer Ansatz besteht darin, alle Untermengen der Attributmenge zu 'minieren' und die Resultate dann mit Cross-Validation zu überprüfen. Dieser Ansatz ist allerdings sehr kostenintensiv. Cross-Validation ist eine Methode zur Bewertung und Validierung der Ergebnisse von Data-Mining-Verfahren. Diese und andere Methoden sollen im folgenden Abschnitt vorgestellt werden.

Bewertung

Im Data-Mining-Bereich im Allgemeinen und in dieser Diplomarbeit im Speziellen reicht es nicht aus, einen passenden Algorithmus auszuwählen. Es sollten immer mehrere Verfahren auf die Daten angewendet und das beste bestimmt werden. Doch wie kann festgestellt werden, wie gut ein Verfahren ist?

Eine Methode, die Ergebnisse zu überprüfen, ist es, von der Trainingsmenge eine so genannte Testmenge abzunehmen. Der DM-Algorithmus wird dann nur noch auf der verbliebenen Trainingsmenge ausgeführt und mit der Testmenge wird überprüft, ob das Ergebnis auch für diese stimmt. Auf diese Weise kann Overfitting vermieden werden.

Cross-Validation ist eine Weiterentwicklung dieser Methode. Dabei wird die Trainingsmenge in mehrere Teilmengen unterteilt. Jeweils eine davon fungiert als Testmenge, die restlichen als Trainingsmenge. Dies wird so oft wiederholt, bis jede Teilmenge einmal als Testmenge diente, also bei n Teilmengen n mal.

Eine weitere Möglichkeit, Algorithmen und ihre Ausgaben zu bewerten, sind **Scoring-Funktionen**. Einige Verfahren nutzen eine Scoring-Funktion implizit, indem sie damit das jeweilige Zwischenergebnis bewerten und das beste auswählen. So gibt zum Beispiel die Kategoriennützlichkeit, die beim Inkrementellen Clustering benutzt wird, die Qualität des jeweiligen Clustering-Ergebnisses an. Eine andere einfache Clustering-Bewertungsfunktion wird in [35] beschrieben und basiert auf zwei Maßen: der 'within cluster variation' $wc(C)$ und der 'between cluster variation' $bc(C)$, wobei C die Partition einer Menge in Cluster ist. Die Idee ist, mit diesen Maßen die beiden typischen Clustering-Eigenschaften zu messen: Die Instanzen eines Clusters sind sich sehr ähnlich (kleines $wc(C)$) und die Cluster selbst unterscheiden sich deutlich (großes $bc(C)$). Es gibt verschiedene Möglichkeiten $wc(C)$ und $bc(C)$ zu berechnen. Eine davon ist:

$$wc(C) = \sum_{k=1}^K wc(C_k) = \sum_{k=1}^K \sum_{x \in C_k} d(x, r_k)^2$$

$$bc(C) = \sum_{1 \leq j < k \leq K} d(r_j, r_k)^2$$

Das Maß $wc(C)$ berechnet sich aus den Werten $wc(C_k)$ für alle Cluster C_k . C_k wiederum ist die Summe der Quadrate der Distanzen $d(x, r_k)$ von jeder Instanz x des Clusters zum Cluster-Zentrum r_k . $wc(C)$ ist somit ein Maß dafür, wie weit die Instanzen durchschnittlich vom Cluster-Zentrum entfernt sind. $bc(C)$ hingegen gibt die Summe der Quadrate der Distanzen zwischen jeweils zwei Cluster-Zentren an. Die Bewertungsfunktion für C ist dann:

$$SF(C) = \frac{bc(C)}{wc(C)}$$

Aufgrund der oben genannten Eigenschaften hat eine gute Partition einen möglichst großen Wert für SF .

Damit soll der Überblick über den Bereich des Data Mining, der für das Verständnis der vorliegenden Arbeit benötigt wird, beendet sein. Als Startpunkt für die Vertiefung der Thematik seien [57] und [35] empfohlen. Einen vergleichenden Überblick über Clustering-Verfahren gibt [24].

Kapitel 3

Stand der Forschung und vorhandene Programme

3.1 Schreibererkennung

Beim Erkennen von Schreibern werden zwei Basisalgorithmen unterschieden:

- Identifikation
- Verifikation

Die Aufgabe der Verifikation besteht darin, für eine gegebene Handschrift zu überprüfen, ob sie von einem bestimmten Schreiber stammt. Sie kann zum Beispiel eingesetzt werden, um eine Unterschrift auf ihre Echtheit zu überprüfen. Das Ziel der Identifikation ist es, eine vorliegende Handschrift einem Schreiber zuzuordnen. Es muss also der ähnlichste Schreiber aus einer Menge von Schreibern gefunden werden. Die Identifikation eines Schreibers ist eine Aufgabe dieser Arbeit. Es soll ein Notenschreibererkennungssystem entwickelt werden, das bestimmt, zu welchem Schreiber eine gegebene Handschrift gehört.

Es gibt viele Ansätze, die sich dem Thema der Schreibererkennung widmen. In diesem Abschnitt sollen solche vorgestellt werden, die Verfahren benutzen, die auch in dieser Arbeit diskutiert werden:

- Schreibererkennung mit dem K-Nearest-Neighbor-Verfahren
- Information-Retrieval-basierte Schreibererkennung

Diese Ansätze sollen im Folgenden erläutert werden.

3.1.1 Schreibererkennung mit dem K-Nearest-Neighbor-Verfahren

Eine Umsetzung stammt von Bin Zhang, Sargur N. Srihari und Sangjik Lee aus dem 'Computer Science and Engineering Department' der 'State University of New York' in Buffalo. Sie haben ihr Verfahren in [59] vorgestellt.

Das Verfahren isoliert zuerst die einzelnen Symbole und legt dann für alle Symbole Micro-Features an, die sich schon in [28] und [53] bewährt haben. Ein Micro-Feature ist ein dreigeteilter Vektor, bestehend aus 512 Bits. Er enthält Informationen über den Gradienten, die Struktur und die Wölbung des Symbols, das dazu in jeweils 4×4 Regionen unterteilt wird. Für die Menge Ω dieser n-dimensionalen binären Vektoren wird dann eine Distanzfunktion

$$D^b : \Omega \times \Omega \rightarrow \mathbb{R}$$

definiert, welche die Distanz zweier Symbole angibt. Für $X, Y \in \Omega$ gilt:

$$D^b(X, Y) = \frac{1}{2} - \frac{S_{11}S_{00} - S_{10}S_{01}}{\sqrt{2(S_{10} + S_{11})(S_{01} + S_{00})(S_{11} + S_{01})(S_{00} + S_{10})}},$$

wobei S_{ij} ($i, j \in \{0, 1\}$) die Anzahl der Vorkommen angibt, bei den i in X und j in Y an genau der gleichen Stelle auftreten. Für zwei Dokumente A und B , die l Paare gleicher Symbole enthalten, lautet die Distanz:

$$D(A, B) = \frac{1}{l} \sum_{i=1}^l D_i^b,$$

wobei D_i^b nach oben genannter Vorschrift berechnet wird.

Für die Schreiberidentifikation wurde das k-Nearest-Neighbor-Verfahren eingesetzt, das die beschriebenen Distanzfunktionen nutzt. Es hat sich gezeigt, dass das Verfahren mit $k = 2$ nicht unbedingt besser war, als jenes mit $k = 1$. In [59] wurden 62 verschiedene Symbole auf ihre Relevanz für die Schreiberidentifikation getestet:

- '0' bis '9'
- 'a' bis 'z' und
- 'A' bis 'Z'.

Es hat sich gezeigt, dass Zahlen dafür im Allgemeinen am schlechtesten geeignet sind. Die Zahl '1' erreichte die schlechteste Performance, gefolgt von der '0'. Aber auch Buchstaben wie 'C' und 'X' ließen wenig Individualität erkennen. Werden nur die Symbole der ersten Gruppe für die Berechnung der Distanz genutzt, kann das System knapp 74,8% der Schreiber richtig identifizieren. Dieser Wert erhöht sich bei Hinzunahme der Symbole der zweiten Gruppe auf 94,5% und erreicht bei Ausnutzung aller 62 Symbole sogar mehr als 97,8%.

Die Autoren fassen zusammen, dass sie mit Ihrer Arbeit einen ersten Schritt gemacht haben, die Bedeutung und Gewichtung einzelner Symbole für die Schreibererkennung zu bestimmen.

3.1.2 Information-Retrieval-basierte Schreibererkennung

Die Universität von Rouen in Frankreich beschäftigt sich schon seit längerer Zeit mit Methoden der Schreibererkennung. Ihre Ansätze werden in [49], [22], [48]

und [23] erläutert. In der jüngsten Veröffentlichung [23] wird eine Methode vorgestellt, welche die Schreiberidentifikationsmethoden auf Information-Retrieval-Techniken (IR) überträgt.

Für dieses Verfahren wird ein handgeschriebenes Dokument (D_j) in Grapheme zerlegt. Ein Graphem ist die kleinste, bedeutungsunterscheidende Einheit der geschriebenen Sprache, zu vergleichen mit einem Symbol aus dem vorherigen Abschnitt. Es wurde darüber hinaus in [49] gezeigt, dass jede Handschrift redundante Muster enthält, die **Writer's Invariants** genannt werden. Aufgrund dieser Redundanzen kann die Datenmenge um den Faktor vier reduziert werden, ohne große Verluste in der Genauigkeit der Ergebnisse in Kauf nehmen zu müssen.

Der ursprünglich Ansatz definiert die Menge aller Grapheme (x_i) eines handgeschriebenen Dokuments als:

$$D_j = \{x_i, i \leq |D|\}$$

Ein Ähnlichkeitsmaß für ein unbekanntes Dokument Q und ein Referenzdokument D in der Datenbank ist:

$$SIM(Q, D) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{Max}_{y_j \in D} (\text{sim}(x_i, y_j)),$$

wobei $\text{sim}(x_i, y_j)$ das Ähnlichkeitsmaß zweier Grapheme ist. Der Schreiber des unbekanntes Dokuments Q ist dann der Schreiber des Dokumentes D_j , das Q am ähnlichsten ist:

$$\text{Writer}(Q) = \text{Writer}(\text{Argmax}_{D_j \in \text{base}} SIM(Q, D_j))$$

Dieses Modell soll auf ein IR-Modell übertragen werden. Dazu wird in [23] das Vektorraummodell beschrieben. Die Dokumente und die Anfrage (Query) werden durch einen hoch-dimensionalen Feature-Raum beschrieben. Für jedes Dokument wird in der Indexphase ein Vektor in diesem Raum erstellt, der das Dokument beschreibt. In der Retrieval-Phase wird dann nur noch die Relevanz eines jeden Dokuments für die Anfrage berechnet. Formal seien ein Dokument D_j und eine Anfrage Q die Vektoren:

$$D_j = (a_{0,j}, \dots, a_{m-1,j})^T$$

$$Q = (b_0, \dots, b_{m-1})^T,$$

wobei $a_{i,j}$ und b_i die Gewichte eines binären Features φ_i ($1 \leq i \leq m$) sind. Sie berechnen sich wie folgt:

$$a_{i,j} = FF(\varphi_i, D_j) \cdot IDF(\varphi_i)$$

$$b_i = FF(\varphi_i, Q) \cdot IDF(\varphi_i)$$

FF ist die Feature-Häufigkeit in dem entsprechenden Dokument und IDF die inverse Dokumenthäufigkeit. Die Idee dahinter ist, dass ein Feature umso relevanter für die Beschreibung eines Dokumentes ist, je häufiger (relativ betrachtet) es in diesem Dokument im Vergleich zu allen anderen vorkommt. Die Ähnlichkeit zwischen Q und D_j wird durch den Kosinus des Winkels zwischen den beiden Vektoren angegeben:

$$\cos(Q, D_j) = \frac{\sum FF \cdot IDF_{\varphi_i, D_j} FF \cdot IDF_{\varphi_i, Q}}{\sqrt{\sum FF \cdot IDF_{\varphi_i, D_j}^2 \sum FF \cdot IDF_{\varphi_i, Q}^2}}$$

Dieses IR-Modell wird auf die Schreiberidentifikation übertragen, indem die **Writer's Invariants** als binäre Features betrachtet werden und so ein allgemeiner Feature-Raum über der Menge aller handschriftlichen Dokumente aufgespannt wird. Das so aufgestellte System erreicht eine korrekte Identifikation in 93,3% aller Testfälle. [23] endet mit der Schlussfolgerung, dass die IR-Methode ähnliche Ergebnisse erzielt, wie die zuvor entwickelten Ansätze, sich aber durch ihre lineare Zeitkomplexität hervorhebt.

Die in diesem Kapitel vorgestellten Verfahren unterscheiden sich in einem entscheidenden Punkt von dem zu entwickelnden Notenschreibererkennungssystem. Sie setzen mit der Analyse einer Handschrift direkt bei der Handschrift an und können die Repräsentation dieser Handschrift selbst wählen. Das Notenschreibererkennungssystem setzt eine Ebene höher an, da die Repräsentation der Symbole durch Feature und ihre Werte in der Feature-Base schon festgelegt ist. Nichtsdestotrotz können allgemeine Verfahren, wie zum Beispiel das k-Nearest-Neighbor-Verfahren, übernommen werden und die Testergebnisse mit denen der anderen Systeme verglichen werden.

Nachdem in diesem Abschnitt bereits existierende Verfahren zur Schreibererkennung beschrieben wurden, sollen im nächsten Abschnitt Data-Mining-Tools aus dem kommerziellen und Forschungsbereich vorgestellt werden.

3.2 Data-Mining-Tools

Es existieren viele Tools, welche Data-Mining-Technologien umsetzen und dem Nutzer in einfacher Form anbieten, so dass er sie für seine Zwecke nutzen kann. In diesem Abschnitt sollen die für diese Arbeit interessanten vorgestellt werden. In Kapitel 7.1 wird dann untersucht, inwieweit diese Programme für das zu entwickelnde Schreibererkennungssystem genutzt werden können.

Es gibt bereits Paper und Berichte, welche die folgenden Programme nach allgemeinen Kriterien bewerten. Die Informationen in diesem Kapitel stammen aus solchen Berichten ([38], [31], [47], [33]) und aus den im Folgenden direkt genannten Quellen.

Clementine von der Firma SPSS basiert auf einer visuellen Programmierschnittstelle, die Datenzugriff, Manipulation, Visualisierung und DM-Technologie verbindet. Als Datenquelle können Datenbanken mit ODBC-

Schnittstelle, Text-, SPSS- und SAS-Dateien¹ dienen. Die Ausgabe ist über ODBC, Text-, SPSS-, SAS-, und Microsoft-Excel-Dateien möglich. Die unterstützten Klassifikationsmodelle sind Entscheidungsbäume, Regelinduktion, Neuronale Netze und lineare Regression. Für das Clustering wird zum Beispiel der Algorithmus K-Means angeboten. Clementine unterstützt den Standard CRISP-DM². Die gelernten Regeln und Netze können als C-Code exportiert werden. Außerdem bietet Clementine die Möglichkeit, eine Schnittstelle von anderen Programmen aus aufzurufen. Auf der Homepage von SPSS (www.spss.com) kann eine Demo heruntergeladen werden. Clementine kann zur Zeit direkt über <http://www.spss.com/spssbi/clementine> erreicht werden.

Darwin von Oracle bietet eine grafische Oberfläche und Wizards, um dem Nutzer das Zusammentragen der Daten, das Erstellen der DM-Modelle und die Interpretation der Ergebnisse zu erleichtern. Die Daten können aus Datenbanken, Text- und SAS-Dateien gelesen werden. Die Modelle können in C oder Java generiert und von anderen Programmen genutzt werden. Darwin unterstützt unter anderem Entscheidungsbäume, den K-Means- und den Nearest-Neighbor-Algorithmus. Der Hersteller ist im Internet unter www.oracle.com zu erreichen. Dort gibt es auch die folgenden ausführlichen Dokumentationen: [7], [5], [9] und [8].

Enterprise Miner ist ein Produkt von SAS. Es zeichnet sich durch eine komfortable grafische Oberfläche aus, die dem Nutzer vielfältige Möglichkeiten der Vorverarbeitung der Daten und des Testens der Data-Mining-Modelle gibt. Als Datenquelle werden mehr als 50 verschiedene Dateiformate oder ein Data-Warehouse unterstützt. Angebotene DM-Technologien sind unter anderem Entscheidungsbäume, Regelinduktion und der K-Means-Algorithmus. Der Enterprise Miner ist zur Zeit auf den SAS-Seiten (www.sas.com) unter <http://www.sas.com/technologies/analytics/datamining/miner/> verlinkt.

Die Intelligent Miner von IBM sind eine Ansammlung von Data-Mining-Tools, zum Extrahieren von Daten und zum Erstellen, Anwenden, Testen und Visualisieren von DM-Modellen. Die Quelldaten können aus einer DB2-Datenbank oder einer Datei gelesen werden. Der Nutzer hat die Wahl zwischen einem externen Tool mit grafischer Oberfläche und einer direkten Datenbankerweiterung. Die Intelligent Miner (IM) unterstützen PMML³ als XML-basiertes Austauschformat für DM-Modelle und den ISO-Standard 'SQL MM Part 6' für Data Mining. Die Intelligent Miner werden von besonderer Bedeutung sein, da sie direkt in IBMs Datenbank

¹Die Firmen SPSS [2] und SAS [1] haben eigene Formate entwickelt, um die Daten zu speichern.

²Bei CRISP-DM handelt es sich um ein Projekt zur Entwicklung eines industrie- und tool-unabhängigen Data-Mining-Prozess-Modells. An diesem Projekt sind die Firmen Teradata, SPSS, Daimler Chrysler und OHRA Verzekering en Bankk beteiligt.

³Beteiligte Firmen: Angoss Software Corp.; IBM Corp.; Insightful Corp.; KXEN; Magnify, Inc.; Microsoft; MINEit; NCDM; NCR Corp.; Oracle Corporation; Quadstone; SAP; SAS, Inc.; Salford Systems, SPSS Inc.; StatSoft, Inc.; StatSoft, Inc.; Xchange, Inc.;

DB2 integriert werden können und DB2 die Datenbank ist, auf dessen Basis das Schreibererkennungssystem laufen soll. Von den aus Kapitel 2.2 bekannten DM-Verfahren werden Entscheidungsbäume, Regelinduktion und diverse Clustering-Verfahren angeboten.

Eine Reihe ausführlicher Bücher und Referenzen über die Intelligent Miner, wie zum Beispiel [19], [15], [16], [17] und [18] können auf der Homepage von IBM (www.ibm.com) heruntergeladen werden. Die aktuelle URL der Intelligent Miner ist <http://www-306.ibm.com/software/data/iminer/>.

MLC++ ist eine C++ Bibliothek, die von der University of Stanford entwickelt wurde und bis zu der Version 1.3.X frei verfügbar ist. Neuere Versionen werden von Silicon Graphics, Inc. (sgi) vertrieben. SGI's Programm **MineSet** basiert auf dieser Bibliothek. MLC++ bietet von den in der Einleitung genannten Modellen und Algorithmen folgende an: Entscheidungsbäume, Regelinduktion, instanzbasierte Klassifikationsverfahren und lineare Klassifikation. Es werden Methoden bereitgestellt, um verschiedene Quelldatenformate einzulesen. MLC++ wird in [40], [41] und [45] genauer beschrieben. Es existiert mit [39] auch eine einfache Einführung in MLC++. SGI ist unter www.sgi.com zu finden, wo MLC++ aktuell als <http://www.sgi.com/tech/mlc/> verlinkt ist.

Model 1 ist ein Softwarepaket von Unica. Es ist auf einfache Anwendbarkeit ausgelegt. Model 1 bietet dem Nutzer Wizards an, um Probleme zu lösen. Darüber hinaus wendet es automatisch verschiedene Algorithmen auf ein gegebenes Problem an. Die Daten können aus einer Textdatei oder einer Datenbank gelesen werden. Von den aus Abschnitt 2.2 bekannten Verfahren setzt es Entscheidungsbäume und den K-Means-Algorithmus ein. Seine Vorteile liegen nach [38] eindeutig in der einfachen Bedienung. Der Hersteller ist im Internet unter <http://www.unica-usa.com> erreichbar.

Weka ist ein Akronym für Waikato Environment for Knowledge Analysis. Es handelt sich dabei um ein System, bestehend aus mehreren Tools, welche die in [57] vorgestellten Verfahren und Algorithmen implementieren. Weka ist komplett in Java umgesetzt. Der Nutzer kann die DM-Methoden entweder über Kommandozeilenbefehle aufrufen oder die Weka-API direkt aus eigenen Programmen heraus nutzen. Die Kommandozeilen-Tools benötigen die Eingabedaten in Form von ARFF-Dateien. Dieses Format wird genauer in [57] erläutert. Weka implementiert alle in Kapitel 2.2 vorgestellten Verfahren. Es bietet außerdem die Möglichkeit der Vorverarbeitung der Daten und des Testens und Visualisierens der DM-Modelle. Das Programm kann aktuell unter www.cs.waikato.ac.nz/ml/weka heruntergeladen werden.

Die hier vorgestellten Data-Mining-Tools werden in Kapitel 7.1 darauf hin untersucht, inwieweit sie bei der Umsetzung des Schreibererkennungssystems genutzt werden können.

In diesem Kapitel wurde der aktuelle Stand in den Bereichen der Schreiberidentifikation und der Data-Mining-Tools präsentiert. Das nächste Kapitel wird einen Überblick darüber geben, was das zu entwickelnde System leisten und welche Anforderung es erfüllen soll.

Kapitel 4

Einsatzszenarios und Anforderungen

Als Einleitung in die Thematik dieser Arbeit sollen in dem nun folgenden Kapitel die Einsatzszenarios der zu entwickelnden Technologie dienen. Im ersten Abschnitt werden die möglichen Anwendungen erläutert. Im darauf folgenden Abschnitt werden daraus allgemeine Anforderungen an die zu entwickelnden Algorithmen und Strukturen abgeleitet und diese durch darüber hinausgehende Anforderungen ergänzt.

4.1 Einsatzszenarios

Die in diesem Kapitel beschriebenen Szenarios geben einen guten Überblick über den Gegenstand und das Anliegen der Arbeit. Das allgemeine Ziel ist es, Methoden, Algorithmen und Strukturen zur Schreibererkennung in eine Datenbankumgebung zu integrieren. Diese sollten aber nicht nur auf genau einen Einsatzschwerpunkt optimiert sein, sondern verschiedene Anwendungen unterstützen. Dabei handelt es sich um folgende Anwendungen:

- Die Initialisierungsphase,
- Änderungsoperationen, die das Hinzufügen von neuen Schreibern ermöglichen und
- Anfrageoperationen, die Anfragen nach dem Schreiber erlauben.

Diese drei Szenarios werden im Folgenden genau beschrieben.

Initialisierungsphase

Die Initialisierungs- oder Testphase ist bereits aus Kapitel 2.2 bekannt. Ihr Ziel ist es, mit Hilfe von Data-Mining-Techniken Zusammenhänge, Abhängigkeiten und Regeln zu finden, die es erlauben, die Feature-Vektoren den Kopisten zuzuordnen. Dafür wird eine große Menge von Feature-Vektoren benötigt, für die jeweils bekannt ist, zu welchem Schreiber sie gehören.

Nach der Lernphase ist das System einsatzbereit für den laufenden Betrieb in einer Datenbankumgebung. Darauf beziehen sich die folgenden zwei Szenarios.

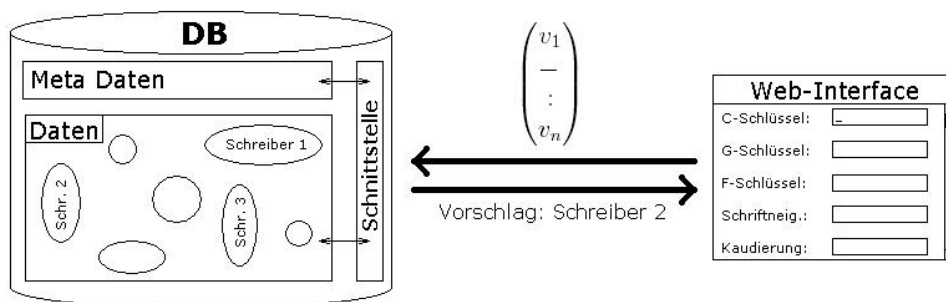


Abbildung 4.1: Einfügen einer neuen Schreibercharakteristik

Änderungsoperationen

Datenbankoperationen werden in Änderungs- und Anfrageoperationen unterteilt. Das Hinzufügen von Daten gehört in die erste Kategorie. Es umfasst alle Vorgänge, die nötig sind, um eine neue Schreibercharakteristik in die Datenbank einzufügen.

Der Nutzer soll in der Lage sein, die Werte der einzelnen Schreibermerkmale über ein Web-Interface einzugeben. Abbildung 4.1 veranschaulicht die Übermittlung des Feature-Vektors vom Web-Interface aus an die Datenbank. Wie in der Lernphase sollte der Vektor genug Informationen und nicht übermäßig viele Nullwerte enthalten. Wenn der Schreiber bekannt ist, muss der Feature-Vektor in die entsprechende Klasse eingeordnet werden. Ist der Schreiber jedoch nicht bekannt, so muss zuerst eine Klasse bestimmt werden. Das System sollte automatisch feststellen, um welchen Schreiber es sich handelt, bzw. dem Nutzer eine Liste mit den wahrscheinlichsten Kopisten anbieten. In Abbildung 4.1 wird zum Beispiel festgestellt, dass es sich bei dem Schreiber um 'Schreiber 2' handelt und dieses Ergebnis wird an das Web-Interface gemeldet.

Darüber hinaus sollte das System auch in dieser Phase noch dazulernen und das durch die neu hinzukommende Schreibercharakteristik gewonnene Wissen in seine Wissensbasis aufnehmen. Neben einer Änderungsoperation soll, wie bereits erwähnt, auch eine Anfrageoperation unterstützt werden.

Anfragen nach Schreibern

Dieses Szenario ist dem Änderungsszenario sehr ähnlich, wie zum Beispiel Abbildung 4.2 verdeutlicht. Der Nutzer kann eine Anfrage wieder über das Web-Interface stellen. Da es sich aber nur um eine Anfrage handelt, sollte er dabei nicht gezwungen sein, alle ihm bekannten Merkmale anzugeben. Stattdessen sollte er im besten Falle nach Eingabe eines jeden neuen Merkmales ein Zwischenergebnis bekommen, das angibt, auf wie viele Kopisten seine Beschreibung noch zutrifft. So kann er die Eingabe abbrechen, wenn er ein eindeutiges oder genügend genaues Ergebnis bekommt.

Im Gegensatz zu den ersten beiden Szenarios bedeutet dies, dass der Feature-Vektor sehr viele Nullwerte enthalten kann. Zu Beginn wird überhaupt nur ein einziges Feature einen Wert haben. Dafür wird die Ergebnismenge am

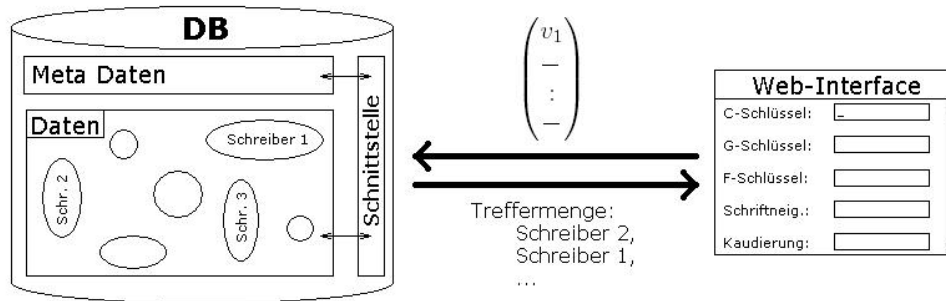


Abbildung 4.2: Anfrage nach einem Schreiber

Anfang sehr groß sein und mit steigender Zahl der vom Nutzer eingegebenen Merkmale immer mehr eingeschränkt werden.

Über diese drei vorgestellten Szenarios hinaus sind natürlich noch weitere möglich, die aber letztendlich nur Erweiterungen der vorgestellten Beispiele sind. Es wurde zum Beispiel bisher davon ausgegangen, dass der Nutzer die Schreibercharakteristik per Hand eingibt. Dabei wäre es besser, wenn das Web-Interface diese automatisch vom Notenblatt einlesen würde. An dieser Stelle reichen aber die drei grundlegenden Szenarios aus, da sich aus ihnen die Anforderungen an das Schreibererkennungssystem herauskristallisieren.

4.2 Anforderungen

Zu den Anforderungen, die sich aus dem vorangegangenen Abschnitt ableiten lassen, kommen solche, die sich aus der Aufgabenstellung der vorliegenden Diplomarbeit ergeben:

- Ausnutzung geeigneter Data-Mining-Verfahren
- Datenbankintegration
- Unterstützung von Nullwerten
- Mehrere Werte für ein Feature
- Fehlertoleranz

Zunächst sei eine grundlegende Forderung erläutert: Die Ausnutzung geeigneter Data-Mining-Verfahren zur Schreibererkennung.

4.2.1 Ausnutzung geeigneter Data-Mining-Verfahren

Das zu entwickelnde System soll auf einem gut geeigneten Data-Mining-Verfahren basieren. Das kann auch bedeuten, dass ein existierendes Verfahren erweitert wird, um die weiteren Anforderungen dieses Kapitels zu erfüllen. Es

wird in dieser Arbeit hauptsächlich darum gehen, ein bezüglich der Anforderungen und Gegebenheiten optimales Modell zu finden und weniger darum, eine optimale Implementierung zu entwickeln. Die Strukturen und Algorithmen werden aber prototypisch implementiert und in Kapitel 7.3 erläutert. Das gesamte Kapitel 6 beschäftigt sich zuvor mit der Eignung von existierenden Data-Mining-Verfahren, mit der Diskussion von Erweiterungen dieser und mit der Einführung neuer, optimierter Strukturen und Algorithmen.

4.2.2 Datenbankintegration

Der zweite wichtige Aspekt des zu entwickelnden Systems ist die Integration in eine Datenbankumgebung. Dies bedeutet einerseits, dass die Strukturen und die Funktionalität eines Datenbanksystems ausgenutzt werden. Andererseits aber auch, dass die neue Schreibererkennungskomponente für den Nutzer transparent ist. Die Abbildungen 4.1 und 4.2 zeigen zum Beispiel, dass der Nutzer über eine Schnittstelle mit dem System kommuniziert und nicht direkt auf die Daten und Methoden zugreift.

Als Datenbanksystem für die Data-Mining-Komponente ist DB2 von IBM vorgesehen. Im Rahmen dieser Arbeit soll untersucht werden, inwieweit die in Abschnitt 3.2 vorgestellten Produkte zur Umsetzung genutzt werden können. Diese Thematik wird in Abschnitt 7.1 diskutiert.

4.2.3 Unterstützung von Nullwerten

Die Notwendigkeit der Behandlung von Nullwerten resultiert aus den drei Szenarios aus Abschnitt 4.1, die diesbezüglich ein gegensätzliches Verhalten haben. Im dritten Szenario, welches Anfrageoperationen beschreibt, enthalten die Feature-Vektoren extrem viele Nullwerte. Diese bedeuten nicht wie in den ersten beiden Beispielen, dass kein Wert bekannt ist, sondern nur, dass er noch nicht eingegeben wurde. Die Algorithmen müssen in beiden Fällen gute, wie in Abschnitt 4.1 beschriebene, Ergebnisse liefern. Die Bedeutung von Nullwerten wird in Kapitel 5.2.7 vertieft.

4.2.4 Mehrere Werte für ein Feature

In einigen Fällen kann es auftreten, dass für ein Feature mehrere Werte bekannt sind. Das ist zum Beispiel der Fall, wenn ein Kopist auf einem Notenblatt ein Symbol auf zwei eindeutig verschiedene Weisen schreibt. Das Schreibererkennungssystem muss dieses mehrfache Vorkommen mit passenden Verfahren behandeln. Die Thematik wird in Abschnitt 5.2.6 diskutiert.

4.2.5 Fehlertoleranz

Es gibt mehrere Stellen an denen Fehler auftreten können, wie zum Beispiel beim manuellen Bestimmen der Feature-Werte in der Feature-Base. Es kann leicht passieren, dass zwei Musikwissenschaftler für ein und dieselbe Note zwei zwar sehr ähnliche, aber dennoch verschiedene Werte bestimmen. Das System muss diese Ähnlichkeit berücksichtigen. Das Gleiche gilt für ähnlich aussehende

Schriftsymbole.

Im ersten Teil dieser Arbeit wurden die musikwissenschaftlichen und Data-Mining-Grundlagen eingeführt, der neueste Stand der Forschung und aktuelle DM-Tools beschrieben und die Anforderungen an das zu entwickelnde System formuliert. Im zweiten Teil der Arbeit wird die Konzeption und die Umsetzung des System diskutiert.

Teil II

Konzeption und Umsetzung

Kapitel 5

Problembeschreibung und Voraussetzungen

Bevor vorhandene Konzepte, Verfahren und Programme genauer untersucht werden, soll in diesem Kapitel eine Vorauswahl getroffen werden. So gibt es für einige Verfahren Gründe und Kriterien, die deren Einsatz von vornherein ausschließen. Hier soll die Tauglichkeit der existierenden Algorithmen, Strukturen und Programme bezüglich der Anforderungen an das zu entwickelnde Schreibererkennungssystem diskutiert werden.

5.1 Bewertung vorhandener Data-Mining-Verfahren

In Kapitel 2.2 wurden bereits die beiden Data-Mining-Konzepte Klassifikation und Clustering eingeführt. Es soll nun untersucht werden, inwieweit sie für die Notenschreibererkennung geeignet sind.

5.1.1 Klassifikationsverfahren

Klassifikationsverfahren dienen dazu, die zugehörige Klasse einer Instanz aus einer großen Datenmenge zu bestimmen. Induktive Klassifikationsverfahren lernen dafür einmalig (One-Shot-Lernen) anhand einer Trainingsmenge eine Menge von Regeln. Im Gegensatz dazu klassifizieren instanzbasierte Verfahren eine Instanz mit Hilfe der bereits in der Datenbank befindlichen Instanzen. Induktive Verfahren werden in den verschiedensten Anwendungen eingesetzt. Dazu zählen zum Beispiel 1-rule, ID3, C4.5 oder SVM, die alle in Abschnitt 2.2 beschrieben wurden. Für das Schreibererkennungssystem sind sie aber nicht geeignet. Der Grund dafür ist, dass Ähnlichkeiten von Instanzen für das Schreibererkennungssystem eine entscheidende Rolle spielen. In Kapitel 5.2.5 wird ausführlich erklärt, dass diese Ähnlichkeiten keiner herkömmlichen Metrik entsprechen. Darum benötigt das zu entwickelnde Modell ein eigenes Ähnlichkeitsmaß. Verfahren, die auf Entscheidungsbäumen oder Klassifikationsregeln basieren, können nicht eingesetzt werden, weil sie für den Attribut-/Wertevergleich nur die Vergleichsoperatoren $<$, \leq , $=$, \neq , \geq , $>$ zulassen, mit denen keine unscharfen Ähnlichkeiten realisiert werden können. Modelle, die auf einem Vektorraum

basieren (z.B. SVM) sind nicht verwendbar, da die Attribute der Feature Base nominal skaliert sind und somit keinen Raum aufspannen können. Das Problem des eigenen Ähnlichkeitsmaßes kommt hier noch dazu, denn die Metrik, die in so einem Raum gilt, gilt im Falle des Schreibererkennungssystems nicht.

Die induktiven Klassifikationsverfahren haben drei weitere Nachteile. Sie

- sind nicht dafür geeignet, Fachwissen zu integrieren
- neigen zu Overfitting und
- sind One-Shot-Lern-Verfahren.

Die drei Punkte werden im Folgenden genauer erläutert.

Integration von Fachwissen

Induktive Klassifikationsverfahren sind nicht darauf ausgelegt, Fachwissen zu berücksichtigen. Aufgrund des Fachwissens der Musikwissenschaftler ist zum Beispiel schon bekannt, welche Features aussagekräftiger sind als andere. Da dies nicht unbedingt aus der Trainingsmenge hervorgehen muss, sollte eine Vorgabe möglich sein. Ein weiteres Beispiel ist das eigene Ähnlichkeitsmaß, welches benötigt wird. Es beschreibt die Ähnlichkeiten aller in der Feature Base vorhandenen Werte. Ein induktives Klassifikationsverfahren kann aber nur Ähnlichkeiten erkennen, die es gelernt hat. Dazu müssten schon in der Trainingsmenge alle möglichen Werte vorhanden sein, was unmöglich ist, da die Feature Base auch Werte enthält, zu denen bisher noch kein Schreiber bekannt ist, der so schreibt.

Overfitting

Aus diesem Zusammenhang ergibt sich ein weiterer Nachteil: Die Gefahr des Overfitting. Die Klassifikationsverfahren leiten die Gewichte, bzw. Nützlichkeit, der Features und ihrer Werte aus der Trainingsmenge ab. Dem kann zwar mit Methoden zur Vermeidung von Overfitting Einhalt geboten werden, ganz verhindert werden kann es allerdings nicht. Im Fall dieser Arbeit ist die Gefahr des Overfittings zudem in besonderem Maße gegeben, weil die vorliegende Trainingsmenge nur sehr wenige Instanzen enthält.

One-Shot-Lernen

Induktive Verfahren lernen ihr Wissen einmalig und wenden es dann nur noch auf neue Instanzen an. Dieses Vorgehen widerspricht aber dem in Abschnitt 4.1 genannten Prinzip des lebenslangen Lernens. Es gibt jedoch Verfahren, die genau dieses Prinzip umsetzen: die instanzbasierten Verfahren.

Instanzbasierte Klassifikationsverfahren, wie zum Beispiel k-Nearest-Neighbor, treffen die Entscheidung der Klassenzugehörigkeit aufgrund der in der Datenbank bereits vorhandenen Instanzen. Damit lernen sie auch mit jeder neuen Instanz dazu. Das k-Nearest-Neighbor-Verfahren benötigt nur eine Distanzfunktion, die jeweils den Abstand zweier Instanzen angibt. Diese wird,

basierend auf dem gesamten Wissen, das über die Features und ihre Werte vorhanden ist, erstellt. Somit ist die Umsetzung eines eigenen Ähnlichkeitsmaßes möglich. In bereits existierenden Systemen zur Schreibererkennung, wie [59] und [23], die in Kapitel 3.1 beschrieben wurden, hat sich das k-Nearest-Neighbor-Verfahren bereits bewährt.

Die Herausforderung besteht darin, die Eignung der gewählten Distanzfunktion zu überprüfen, sie zu optimieren und Alternativen zu untersuchen. Für diese Aufgabenstellung spielen Clustering-Verfahren eine entscheidende Rolle.

5.1.2 Clustering-Verfahren

Die Idee des Clustering entspricht nicht der Zielstellung des Schreibererkennungssystem. Übertragen auf die Notenschreibererkennung würde ein Clustering-Verfahren eine Menge von Schreibercharakteristiken deren Schreiber unbekannt sind, automatisch in Teilmengen ähnlicher Handschriften partitionieren. Das kann eine interessante Untersuchung sein, ist aber nicht Teil dieser Diplomarbeit. Trotzdem werden Methoden aus dem Bereich des Clustering in dieser Arbeit Anwendung finden. Der Grund dafür soll im Folgenden kurz skizziert und in Kapitel 6.3.1 detailliert beschrieben werden.

Das Grundprinzip des Clusterings stimmt mit den Eigenschaften überein, denen auch die Distanzfunktion für das k-Nearest-Neighbor-Verfahren genügen soll:

- Instanzen eines Clusters/einer Klasse haben einen minimalen Abstand zueinander.
- Instanzen verschiedener Cluster/Klassen haben einen großen Abstand.

Da die Distanzfunktion den Hauptanteil am Gelingen des Clustering-Verfahrens darstellt, können Bewertungsmaßstäbe für Clustering-Verfahren auf die Distanzfunktion übertragen werden. Es wird ausgenutzt, dass eine schlecht bewertete Partition der Menge von Feature-Vektoren in Cluster darauf schließen lässt, dass die Distanzfunktion und deren Parameter schlecht gewählt wurden. Die Parameter können somit optimiert werden, bis sie eine bessere Bewertung bewirken.

In diesem Abschnitt wurde begründet, warum das k-Nearest-Neighbor-Verfahren zur Schreibererkennung eingesetzt werden wird. Die Distanzfunktion wird im Abschnitt 5.2.3 diskutiert und das Ähnlichkeitsmaß in den Abschnitten 5.2.5 und 6.2.1. Die Vorgehensweise beim Optimieren der Distanzfunktion mit Hilfe der Bewertung von Clustering-Verfahren wird in Kapitel 6.3.1 ausführlich beschrieben. Nachdem ein Data-Mining-Verfahren festgelegt wurde, soll im folgenden Kapitel die daraus resultierende Problemstellung formal beschrieben werden.

5.2 Formale Problembeschreibung

5.2.1 Grundlagen

Es sollen nun die Grundbegriffe formal eingeführt werden. Eine Handschrift wird durch einen Feature-Vektor γ beschrieben. Er enthält Werte für die Features f_1 bis f_n und hat die Form $\gamma = (v_1, \dots, v_n)^T$, wobei v_i ein Wert aus dem Wertebereich W_i des Features f_i ist. Die Menge aller Feature-Vektoren sei $\Gamma = \{\gamma | \gamma = (v_1, \dots, v_n)^T, \forall i : v_i \in W_i\}$. Die Menge aller Features sei $F = \{f | f \text{ ist ein Feature}\}$. Die Mengen F und W_i werden im nächsten Abschnitt genauer definiert.

5.2.2 Feature Base

Die Feature Base wurde schon in Abschnitt 2.1.2 eingeführt. Im Folgenden wird ein formales Modell beschrieben, das im weiteren Verlauf einen einfacheren Bezug auf die Feature Base erlaubt.

In Abschnitt 2.1.2 wurde erläutert, dass die Feature Base (FB) Informationen über die Features und ihre möglichen Werten enthält. Sie beschreibt welche Features es gibt und welche Wertebereiche die einzelnen Features haben. Die Informationen werden in einem gerichteten, azyklischen, knotenmarkierten Graphen (bzw. Baum) dargestellt:

$$FB = (V, E, \mu, \nu, \tau), \text{ wobei}$$

$$V = \{v_1, \dots, v_n\} \text{ eine Menge von Knoten ist und}$$

$E \subseteq V \times V$ eine Menge von gerichteten Kanten, die diese Knoten verbinden. Durch Definition der Feature Base in Form eines einzigen Baumes, wird eine gemeinsame Wurzel für alle Feature-Gruppen¹ unterstellt. Diese Annahme ist für das Modell aber völlig ausreichend.

Jedem Knoten wird eine Bezeichnung $\mu : V \rightarrow B$ aus einer Menge B von Knotenbezeichnern zugeordnet. Außerdem bekommt jeder Knoten eine laufende Nummer, die ihn von seinen Geschwistern unterscheidet: $\nu : V \rightarrow \mathbb{N}$. Für den Knoten '4.1.2. Achtfähnchen aufwärts kaudiert' (x) aus Abbildung 2.3 auf Seite 16 sind $\mu(x) = \text{'aufwärts kaudiert'}$ und $\nu(x) = 2$.

Für die Definition der dritten Funktion τ ist es hilfreich, zuvor folgende Begriffe einzuführen: Für alle $x, y \in V$ gilt:

1. Ein Knoten y heißt Kindknoten von x (kurz: $x \mapsto y$) $\Leftrightarrow (x, y) \in E$, das bedeutet, es existiert eine gerichtete Kante von x nach y .
2. Des Weiteren sei y ein Nachkomme von x (kurz: $x \mapsto \mapsto y$) $\Leftrightarrow (x \mapsto y) \vee (\exists z \in V : x \mapsto z \wedge z \mapsto \mapsto y)$
3. Die Menge aller Nachkommen von x sei $\Lambda_x = \{y | x \mapsto \mapsto y\}$

¹vgl. Abschnitt 2.1.1, Seite 12

4. $LEAVES(FP) = \{x \in V \mid \nexists y \in V : x \mapsto y\}$ ist die Menge aller Blätter des Feature-Base-Baumes. Diese Definition wird etwas später zur Beschreibung der Eigenschaften der Distanzfunktion benötigt.

Damit kann nun die Funktion τ definiert werden als $\tau : V \rightarrow Type$. Sie weist jedem Knoten einen Typ aus der Menge $Type = \{\text{'prefix'}, \text{'feature'}, \text{'value'}\}$ zu. Für den Typ gilt folgendes:

1. $\forall x \in V : \tau(x) = \text{'feature'} \Rightarrow \forall y \in \Lambda_x : \tau(y) = \text{'value'}$
2. $\forall x \in V : \exists y \in \Lambda_x : \tau(y) = \text{'feature'} \Rightarrow \tau(x) = \text{'prefix'}$

Ein Knoten x mit $\tau(x) = \text{'feature'}$ heißt Feature. Der darunter liegende Teilbaum Λ_x ist der Wertebereich dieses Features. Jeder Knoten dort hat nach Regel 1 den Typ 'value', weil er einen möglichen Wert für dieses Feature darstellt. Im oben genannten Beispiel mit dem Knoten aus Abbildung 2.3 ist $\tau(x) = \text{'feature'}$. Alle Knoten, die über x in der Hierarchie liegen, haben den Typ 'prefix', alle darunter den Typ 'value'.

Damit können nun auch die Definitionen für die Menge aller Features F und für den Wertebereich W_i aus der Einleitung dieses Kapitels verfeinert werden.

$$F = \{f \mid f \in V \wedge \tau(f) = \text{'feature'}\}$$

$$W_i = \Lambda_{f_i}, \text{ für alle } f_i \in F$$

Während der Entwicklung des Schreibererkennungssystems hat sich eine Verfeinerung der Definition herauskristallisiert, die allerdings keinen direkten Einfluss auf die vorliegende Arbeit hat. Sie soll an dieser Stelle der Vollständigkeit halber kurz erläutert werden. Es hat sich gezeigt, dass für die manuelle Bestimmung der Werte mit Hilfe der Feature Base, eine Verfeinerung des Knotentyps 'value' nötig ist. Es wird zusätzlich noch zwischen auswählbaren und nicht auswählbaren Werten unterschieden. Nicht auswählbar ist ein Wert, wenn er, trotzdem er in der Hierarchie unterhalb eines Feature-Knotens liegt, nicht zum Wertebereich gehört. Solche Knoten dienen dazu, die Feature Base übersichtlicher zu gestalten und zu systematisieren. Ein Spezialfall von nicht auswählbaren Knoten sind solche, die keine Geschwister haben. Sie bekommen keine Nummer, sondern ihr ν ist das Symbol '⊥'. Da sie keine Geschwister haben, muss der Nutzer auf dieser Ebene der Hierarchie auch keine Entscheidung treffen und kann direkt beim Kindknoten mit der Bestimmung fortfahren.

In der Realität wird jedem Knoten auch noch ein Bild zugeordnet, das die manuelle Bestimmung erleichtert. Auf diese Funktion kann im vorliegenden Modell verzichtet werden. Es soll aber eine letzte Funktion eingeführt werden, die den Umgang mit der Feature Base erleichtert: die Pfad-Zuweisung.

Der Pfad

Der Pfad dient als zusätzlicher, eindeutiger Bezeichner für einen Knoten: $PATH : V \rightarrow P$. P ist eine Menge von Pfadausdrücken, die folgendermaßen aufgebaut sind:

Berechnet werden soll der Pfadausdruck $PATH(x)$ für einen Knoten x .

1. Sei w der Wurzelknoten des Baumes. An erster Stelle steht $\nu(y)$ gefolgt von einem Punkt, wobei gilt $w \mapsto y$ und $y \mapsto \mapsto x$ gilt.
2. Falls $y=x$ ist, wird an dieser Stelle abgebrochen. Ansonsten folgt darauf $\nu(z)$, mit $y \mapsto z$ und $z \mapsto \mapsto x$, gefolgt von einem Punkt.
3. Dann wird $y:=z$ gesetzt und der Algorithmus bei 2. fortgesetzt.

Im Folgenden wird auch anstelle vom ‘Knoten x , mit $PATH(x) = 13.1.$ ’, kurz vom ‘Knoten 13.1.’ gesprochen, oder von dem Knoten mit dem Code 13.1.

5.2.3 Distanzfunktion

Das Ziel des Schreibererkennungssystems ist es, zwei Handschriften zu vergleichen, indem die sie beschreibenden Feature-Vektoren γ_a und γ_b verglichen werden. Gesucht ist dafür eine Distanzfunktion der Form

$$d_\Gamma : \Gamma \times \Gamma \rightarrow [0..1]$$

Dies kann zum Beispiel die normalisierte Hamming-Distanz (vgl. 2.2.1) sein. Danach gilt für $\gamma_a = (v_1^a, \dots, v_n^a)^T$ und $\gamma_b = (v_1^b, \dots, v_n^b)^T$, dass $d_\Gamma = \frac{d_{f_1} + \dots + d_{f_n}}{n}$, wobei d_{f_i} die Differenz (oder anders: Distanz) von v_i^a und v_i^b ist.

5.2.4 Die Feature-Distanzfunktion

Die Funktion d_{f_i} dient dazu, die einzelnen Attribute jeweils paarweise zu vergleichen, zum Beispiel den G-Schlüssel-Wert des einen Vektors mit dem des anderen, den Rastral-Wert der beiden Vektoren, usw. Es wird also zusätzlich eine Distanzfunktion für jedes einzelne Feature f_i benötigt:

$$d_{f_i} : W_i \times W_i \rightarrow [0..1]$$

Für d_{f_i} lassen sich einige notwendigen Eigenschaften festlegen:

$$1) \forall x, y \in W_i : d_{f_i}(x, y) = 0 \Leftrightarrow x = y$$

2) $\forall x, y \in W_i \wedge x$ und y haben die gleiche Tiefe (bzw. gleiche Pfadlänge) von der Wurzel aus:

$$\forall x' \in \Lambda_x : d_{f_i}(x', y) > d_{f_i}(x, y)$$

$$\forall y' \in \Lambda_y : d_{f_i}(x, y') > d_{f_i}(x, y)$$

Die erste Formel besagt, dass die Distanz zweier identischer Werte Null ist. Sie impliziert ebenfalls, dass unterschiedliche Werte nicht die Distanz Null haben können. Die zweite Formel gibt das Vorgehen beim Vergleich von Knoten verschiedener Ebenen im Feature-Base-Baum vor. Zwei Knoten x' und y , die aus unterschiedlichen Ebenen des Feature-Base-Baumes stammen, haben immer eine größere Distanz zueinander als die Knoten x und y , wobei x ein Vorfahr

von x' ist, der auf der gleichen Ebene wie y liegt. Der Grund dafür wurde schon in Kapitel 2.1.2 beschrieben: Knoten, die nahe den Blättern des Baumes liegen, beschreiben komplexere Symbole als jene, die näher an der Wurzel liegen. Analog gilt die zweite Eigenschaft von Punkt 2). Weitere Eigenschaften der Feature-Distanzfunktion werden im folgenden Abschnitt 5.2.5 erklärt. Die Umsetzung der Eigenschaften wird in Abschnitt 6.2.1 diskutiert.

Damit sind die allgemeinen Eigenschaften der Feature-Distanzfunktion bekannt. Es stellt sich aber noch immer die Frage, wie die Ähnlichkeit² gemessen, interpretiert und dargestellt werden kann. Es gibt dafür drei Möglichkeiten, die sich in Aufwand und Komplexität unterscheiden:

1. Der einfache Wertevergleich,
2. der Wertevergleich mit Wertanalyse und
3. der Wertevergleich basierend auf Wertanalyse und zusätzlichen Informationen.

Diese drei Varianten sollen nun genau erläutert werden.

Wertevergleich

Die einfachste Lösung ist eine Boolesche Logik, bei der nur unterschieden wird, ob die Werte gleich oder verschieden sind:

$$d_{f_i}^{Bool}(x, y) = \begin{cases} 0, & \text{falls } x = y \\ 1 & \text{sonst} \end{cases}$$

In den meisten Fällen wird dieses einfache Abstandsmaß jedoch nicht ausreichen, denn es impliziert, dass die kleinste Änderung im Schriftbild eines Kopisten genauso wie ein maximaler Unterschied behandelt wird. Es wird daher ein Maß benötigt, welches eine Abbildung auf den gesamten Bereich $[0, 1]$ erlaubt. Dazu müssen die Werte genauer analysiert werden.

Wertevergleich mit Wertanalyse

Die Idee dabei ist, die Struktur des Feature-Base-Baumes zur Bestimmung der Distanzen auszunutzen. Dazu seien die Codes $x_P = PATH(x)$ und $y_P = PATH(y)$ von zwei Werten x und y gegeben. Unter einer Wertanalyse ist das Zerlegen von x_P und y_P in (x_P^1, \dots, x_P^n) und (y_P^1, \dots, y_P^n) zu verstehen. Die Werte x_P^i und y_P^i geben jeweils den Knoten an, der in der i . Ebene gewählt wurde und entsprechen jeweils der Nummer ν des Knotens auf der i . Ebene, der auf dem Pfad zu x (bzw. y) liegt. Damit ist die Wertanalyse die Umkehrung des Algorithmus, der in Abschnitt 5.2.2 zur Definition des Pfades genutzt wurde. Die Feature-Distanzfunktion wäre dann:

²In diesem Fall ist es anschaulicher, von Ähnlichkeit zu sprechen und die Distanz als dazu indirekt proportionale Größe im Hinterkopf zu behalten. In den Formeln wird aber weiterhin die Distanz d_{f_i} benutzt.

$$d_{f_i}^{Ana}(x, y) = \frac{\sum_{j=1}^n d_{f_i}^{Bool}(x_P^j, y_P^j)}{n}$$

Zwei Werte, deren Pfade sich nur in einer Ebene unterscheiden, wären damit sehr ähnlich. Diese Distanzfunktion erreicht die Genauigkeit $\frac{1}{n}$, wobei n die Tiefe des Wertebereichbaumes ist. Außerdem basiert sie immer noch auf einer Distanzfunktion mit Boolescher Logik. Darum werden Ähnlichkeiten von Werten einer Ebene nicht erkannt. Dazu werden zusätzliche Informationen benötigt.

Wertevergleich mit Zusätzlichen Informationen

Bei vielen Merkmalen kann die Distanz nicht aus der Baumstruktur abgeleitet werden. In solchen Fällen kann eine Distanzmatrix darüber Auskunft geben, welche Distanz jeweils zwei Werte haben:

$$dis_{f(i)} = \begin{pmatrix} d_{x_1, y_1} & \dots & d_{x_n, y_1} \\ \dots & \dots & \dots \\ d_{x_1, y_n} & \dots & d_{x_n, y_n} \end{pmatrix}, \forall j : x_j, y_j \in W_i,$$

Damit gilt für die Distanzfunktion:

$$d_{f_i}^{Inf}(x, y) = d_{x, y}, \text{ wobei } d_{x, y} \text{ aus } dis_{f(i)} \text{ an der Stelle } (x, y) \text{ stammt.}$$

Im praktischen Einsatz gilt nun, dass $d_{f_i}^{Bool}$ den beiden anderen Funktionen aufgrund ihrer geringen Komplexität vorzuziehen ist, sofern sie Ergebnisse mit ausreichend guter Qualität liefert. Das Gleiche gilt für $d_{f_i}^{Ana}$ gegenüber $d_{f_i}^{Inf}$. Was in diesem Fall ausreichend gute Qualität bedeutet, muss im Folgenden noch untersucht werden. Mit dieser Frage beschäftigt sich Kapitel 5.2.4.

In diesem Abschnitt wurden Differenzen, bzw. Ähnlichkeiten, zwischen zwei Feature-Werten ausführlich diskutiert. Im folgenden Abschnitt soll die Frage beantwortet werden, welche Bedeutung große Ähnlichkeiten und damit kleine Distanzen haben und welche Eigenschaften sie erfüllen.

5.2.5 Ähnlichkeiten

Zwei Schreibercharakteristiken sollen eine kleine Distanz haben, wenn sie von dem gleichen Schreiber stammen, und eine große, wenn sie nicht von dem gleichen Schreiber sind. Doch was bedeutet das für die Ähnlichkeiten auf Feature-Ebene?

Die Ähnlichkeit zweier Feature-Werte entspricht der Wahrscheinlichkeit, dass für den Schreiber mit dem typischen Feature-Wert x der Wert y vorliegt.

Anhand des Beispiels in Abbildung 5.1 soll das verdeutlicht werden. Die Abbildung zeigt die Knoten mit den Codes (PATH) 1.3.1.4.1.2., 1.3.1.4.2. und 1.3.1.4.1.1. Es handelt sich dabei um verschiedene Möglichkeiten für das Hauptelement des F-Schlüssels. Angenommen, über einen Schreiber ist bekannt, dass er normalerweise das Symbol a) schreibt, eine symmetrische, nach links geöffnetes Form mit einem Kringel am unteren Ende. Nun kann es leicht einmal

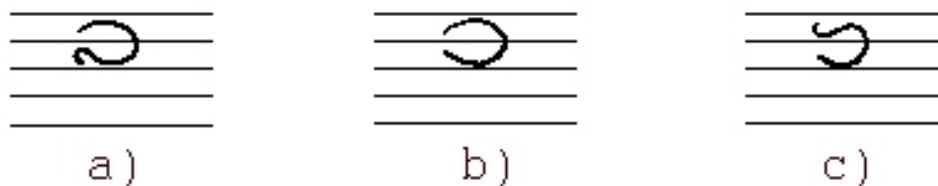


Abbildung 5.1: Beispiel für ähnliche Feature-Werte

passieren, dass dieser Schreiber den Kringel in der Eile nicht schreibt oder ihn nur andeutet. In diesem Fall würde dies dem Wert b) in Abbildung 5.1 zugeordnet werden. Die Distanz zwischen diesen Werten sollte also sehr klein sein, da die Wahrscheinlichkeit groß ist, dass für den Schreiber mit dem typischen Feature-Wert 1.3.1.4.1.2. (a) der Wert 1.3.1.4.2. (b) vorliegt. An diesem Beispiel lässt sich auch gut eine weitere Eigenschaft der Feature-Distanzfunktion zeigen. Sie ist nämlich nicht symmetrisch. Die Wahrscheinlichkeit, dass der Schreiber, der normalerweise keinen Kringel schreibt, plötzlich doch eine Einrollung zeichnet, ist viel geringer als im umgekehrten Fall.

Damit gilt nur eine Eigenschaften einer Metrik nach [35]:

$$\begin{aligned} d(i, j) &\geq 0 \quad \forall i, j \\ d(i, j) &= 0 \quad \text{only if } i = j \end{aligned}$$

Die folgenden beiden aber nicht:

$$d(i, j) = d(j, i) \quad \forall i, j$$

$$d(i, j) \leq d(i, k) + d(k, j) \quad \forall i, j, k$$

Die letztgenannte Bedingung wird Dreiecksungleichung genannt. Dass sie nicht gilt, kann auch mit Hilfe der Werte in Abbildung 5.1 illustriert werden. Das die Distanz von Wert a) zu b) gering ist, wurde schon erläutert. Auch zwischen b) und c) existiert noch eine Ähnlichkeit. Allerdings ist die Distanz von a) nach c) nicht kleiner oder gleich der Summe der beiden anderen Ähnlichkeiten, denn dass ein Schreiber, der normalerweise einen Kringel unten macht, diesen plötzlich weglässt und dafür eine Einrollung oben macht, ist so gut wie ausgeschlossen.

Aufgrund der beschriebenen drei Eigenschaften wird für das Schreibererkennungssystem, wie in Abschnitt 5.1.1 schon erwähnt, ein eigenes Ähnlichkeitsmaß benötigt.

5.2.6 Features mit komplexen Werten

Wie in Kapitel 4.2.4 bereits erläutert wurde, gibt es Merkmale für die mehrere Werte existieren können. Wenn ein Schreiber eine Note auf zwei verschiedene Weisen schreibt, so dass der Nutzer sich nicht eindeutig für eine entscheiden

kann, muss er die Möglichkeit haben, beide Werte anzugeben. Bisher war ein Feature-Vektor:

$$\gamma = (v_1, \dots, v_n)^T, \forall i \in \{0..n\} : v_i \in W_i$$

Im allgemeinen Fall, in dem mehrere Werte für ein Feature erlaubt sind, ist ein Feature-Vektor:

$$\gamma^K = (V_1, \dots, V_n)^T, \forall i \in \{0..n\} : V_i \subseteq W_i$$

Die einzelnen Mengen V_i werden komplexe Werte des komplexen Feature-Vektors genannt. Da es nun Feature-Vektoren mit komplexen Werten gibt, muss auch eine angepasste komplexe Feature-Distanzfunktion definiert werden:

$$d_{f_i}^K : \mathfrak{P}(W_i) \times \mathfrak{P}(W_i) \rightarrow [0..1] \quad (\mathfrak{P}(W_i) : \text{Potenzmenge von } W_i)$$

mit der Eigenschaft:

$$\min_{v^1 \in V_1, v^2 \in V_2} (d_{f_i}(v^1, v^2)) \leq d_{f_i}^K(V_1, V_2) \leq \frac{\sum_{v^1 \in V_1, v^2 \in V_2} d_{f_i}(v^1, v^2)}{|V_1| \times |V_2|}$$

Es stellt sich die Frage, wie die Distanz zwischen zwei komplexen Werten berechnet werden soll. In Abschnitt 8.6 werden verschiedene Varianten untersucht. Die oben genannte Formel beschreibt allerdings die Eigenschaft, die $d_{f_i}^K(V_1, V_2)$ auf jeden Fall erfüllen muss. Dazu werden die Distanzen d_{f_i} aller Kombinationen von Elementen aus V_1 und V_2 betrachtet. $d_{f_i}^K$ darf nicht kleiner sein, als die kleinste Distanz d_{f_i} . Die Distanz soll maximal den durchschnittlichen Wert aller Distanzen d_{f_i} annehmen, denn sonst würde ein komplexer Wert allein aufgrund seines erhöhten Informationsgehaltes schlechter bewertet als ein normaler Wert.

5.2.7 Nullwerte

In Abschnitt 4.2.3 wurde bereits erwähnt, dass Nullwerten eine besondere Bedeutung zukommt. Es werden in der Datenbanktheorie ([27], [34], [52], [44], [55], [58]) verschiedene Arten von Nullwerten unterschieden. Im Fall des Schreibererkennungssystems existieren Nullwerte mit zwei unterschiedlichen Bedeutungen:

Non-Information-Null (?) bedeutet, dass über diesen Wert keine Informationen vorliegen. Das kann zum Beispiel der Fall sein, wenn der Schreiber in dem vorliegenden Werk keinen C-Schlüssel benutzt. In diesem Fall würde das C-Schlüssel-Feature mit dem Nullwert '?' belegt.

No-Applicable-Null (\top) bedeutet, dass nie ein Wert existieren wird. Das kann beispielsweise beim C-Schlüssel auftreten. Der C-Schlüssel besteht aus mehreren Elementen, die jeweils durch ein Attribut beschrieben werden. Es ist typisch für einen Schreiber, welche dieser Elemente er schreibt und welche nicht. Wenn er kein linkes Element schreibt, bekommt das Feature '1.2.1. Linkes Element' den Nullwert ' \top '. Im Gegensatz zu dem

ersten Nullwert (?), bei dem nicht bekannt ist, wie ein Kopist ein Symbol schreibt, ist bei 'T' klar, dass er es nicht schreibt. Es ist eine Hinweis gebende Eigenschaft des Schreibers. Die Bedeutung 'es wird nie ein Wert existieren' bezieht sich im übrigen nur auf den einen Feature-Vektor. In einem anderen Feature-Vektor des gleichen Kopisten kann theoretisch durchaus ein Wert für das Attribut existieren, was aber unwahrscheinlich ist und zu einer hohen Distanz führen sollte.

Die Umsetzung der Nullwerte wird in Kapitel 6.2.3 diskutiert. In diesem Kapitel wurden die Wahl eines instanzbasierten Klassifikationsverfahren begründet, die Problemstellung formal eingeführt und die daraus resultierenden Anforderungen und Eigenschaften der benötigten Strukturen und Methoden definiert. Im folgenden Kapitel wird dargestellt, wie die Anforderungen an das System in Form von Strukturen, Methoden und Algorithmen umgesetzt werden.

Kapitel 6

Umsetzung

Im vorherigen Kapitel wurden die Anforderungen an das Schreibererkennungssystem formuliert und allgemein gültige Lösungsansätze aufgezeigt. Im Gegensatz zu dieser theoretischen Betrachtungsweise werden in diesem Kapitel Möglichkeiten diskutiert, das System unter Berücksichtigung der in Kapitel 5 beschriebenen Eigenschaften und Anforderungen praktisch umzusetzen. Dafür wird zuerst die Systemarchitektur erläutert. In Abschnitt 6.2 werden die Verfahren, Methoden und Strukturen des Systems beschrieben. Das Kapitel endet mit der Einführung einiger Bewertungsmaße, mit deren Hilfe das entwickelte Schreibererkennungssystem bewertet werden kann.

6.1 Systemarchitektur

In Kapitel 4.1 wurden die allgemeinen Einsatzszenarios bereits beschrieben. Daraus ergibt sich im Detail die Systemarchitektur, die in Abbildung 6.1 dargestellt ist. Ausgangspunkt sind eine Menge von Notenhandschriften. Dabei wird zwischen zwei Fällen unterschieden. Die Notenhandschriften können entweder in die Datenbank eingefügt oder für eine Anfrage nach dem zugehörigen Schreiber genutzt werden. Darüber hinaus liegt eine auf der Feature Base basierende Handschrift-Merkmal-Taxonomie vor. Bevor eine Notenhandschrift weiter verarbeitet werden kann, müssen aus ihr die entscheidenden Merkmale extrahiert werden. Dafür wird für jedes vorhandene Merkmal manuell ein passender Wert in der Feature Base bestimmt. Die Menge dieser Werte bildet einen Feature-Vektor. Dieser Vorgang muss in beiden Fällen durchgeführt werden. Wie nach der Merkmals-Extraktion weiter verfahren wird, hängt davon ab, welche Operation der Nutzer ausführen möchte. Handelt es sich um eine Einfüge-Operation wird der Weg des dicken Pfeils nach unten verfolgt und der Feature-Vektor in die Datenbank eingefügt. Im Falle einer Anfrage wird der Feature-Vektor mit dem k -Nearest-Neighbor-Verfahren klassifiziert. Das Klassifikationsergebnis wird an den Nutzer gesendet.

Bevor die Klassifikation allerdings zum ersten Mal ausgeführt werden kann, müssen die Klassifikationsparameter bestimmt und optimiert werden. Die unteren dicken Pfeile zeigen diesen Vorgang. Dazu muss bereits eine Testmenge von Feature-Vektoren in der Datenbank enthalten sein. Es wird dann eine in-

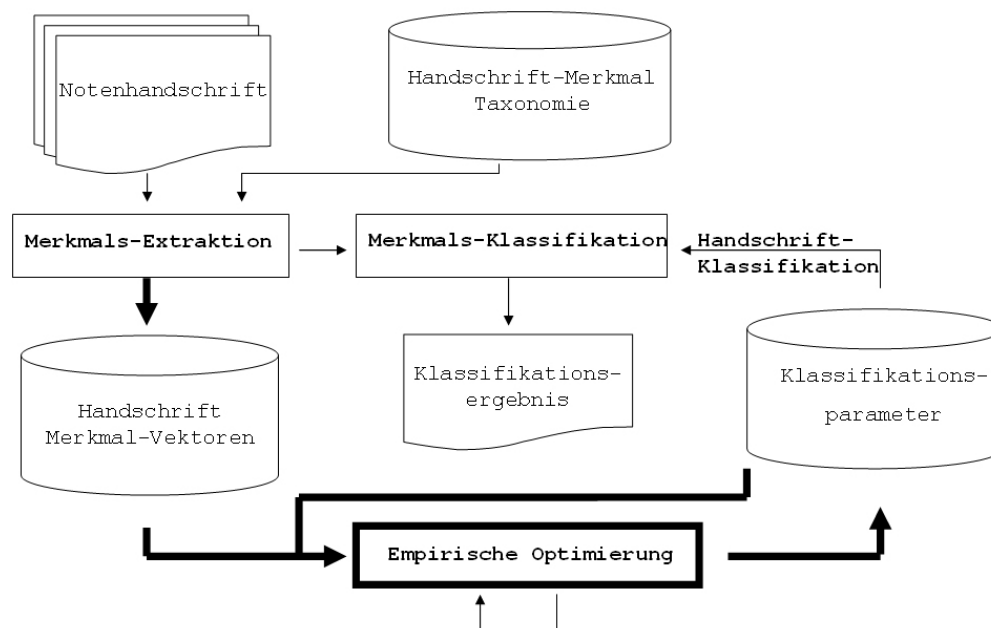


Abbildung 6.1: Systemarchitektur

itiale Konfiguration für das Klassifikationsmodell festgelegt. Die Belegung der Parameter basiert auf dem vorhandenen musikwissenschaftlichen Fachwissen. Die Optimierung der Klassifikationsparameter anhand der Testmenge wird in Kapitel 8 beschrieben.

6.2 Verfahren, Methoden und Strukturen

In Kapitel 5.1 wurde bereits begründet und erläutert, dass und warum ein instanzbasiertes DM-Verfahren zur Schreibererkennung eingesetzt werden soll. Abbildung 6.2 veranschaulicht den Datenfluss des Schreibererkennungssystems. Der Ablauf ist wie folgt. Ein Anwender stellt eine Anfrage, in Form eines Feature-Vektors (γ_q). Mit Hilfe der Distanzfunktion d_Γ werden die k nächsten Feature-Vektoren von γ_q berechnet. Dabei ist k keine Konstante, sondern wird durch einen Schwellwert (t) bestimmt, der nur solche Feature-Vektoren erlaubt, die eine bestimmte Distanz t zu γ_q nicht überschreiten. So ergibt sich eine Antwortmenge von Feature-Vektoren A_{FV} :

$$A_{FV} = \{\gamma \in \Gamma \mid d_\Gamma(\gamma, \gamma_q) \leq t\}$$

Die Wahl eines Schwellwertes anstelle einer konstanten Anzahl von Elementen wurde getroffen, damit die Antwortmenge dynamisch mit der Größe des Datenbestandes wachsen kann. Die Konstante k wäre sonst für kleine Anzahlen von Feature-Vektoren in der Datenbank gleich Eins gewesen und hätte mit dem Wachsen der Datenbank angepasst werden müssen. Außerdem kann so die Entscheidung getroffen werden, dass ein Schreiber noch nicht in der Datenbank

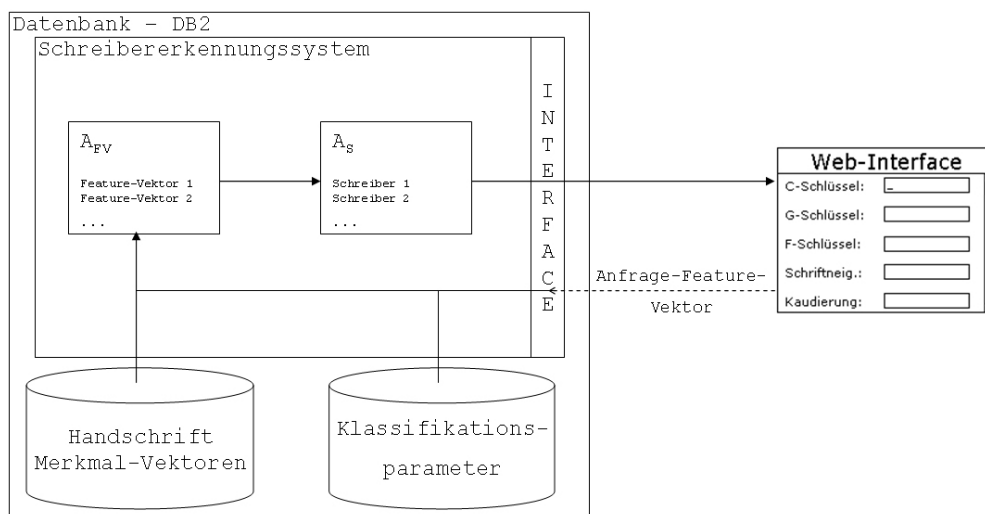


Abbildung 6.2: Detaillierter Ablauf einer Anfrage

vorhanden ist. Dies ist der Fall, wenn A_{FV} die leere Menge ist.

A_{FV} kann dem Nutzer zwar schon als Antwort dienen, im Allgemeinen wird er sich aber nur für die Schreiber interessieren und nicht für die Schreibercharakteristiken selbst. Darum muss mit Hilfe der Menge von Feature-Vektoren A_{FV} eine Antwortmenge A_S berechnet werden, die die zugehörigen Schreiber enthält. Im besten Fall enthält diese Menge maximal einen Schreiber. Das ist entweder der Schreiber, von dem die Notenhandschrift, repräsentiert durch γ_q , stammt, oder die leere Menge, wenn sich der Schreiber noch nicht in der Datenbank befindet. Praktisch wird es sich bei A_S aber um eine Liste mit möglichen Schreibern handeln, wobei jedem eine Bewertung (Ranking) zugeordnet wird. Der Schreiber der Notenhandschrift sollte im besten Fall der mit der besten Bewertung sein. Es ist nicht Ziel dieser Diplomarbeit, eine optimale Ranking-Funktion zu finden. Es werden aber eine grundlegende Ranking-Funktion implementiert und zusätzlich Bewertungsmaße berechnet und angeboten, die zur Bestimmung des Rankings herangezogen werden können. Abschnitt 6.3 stellt diese vor.

6.2.1 Ähnlichkeiten

In Kapitel 5.2.5 wurde über das Ähnlichkeitsmaß folgendes gesagt:

Die Ähnlichkeit zweier Feature-Werte entspricht der Wahrscheinlichkeit, dass für den Schreiber mit dem typischen Feature-Wert x der Wert y vorliegt.

Das ist aber nur in der Theorie der Fall. Bei der praktischen Umsetzung muss ein weiterer Einflussfaktor beachtet werden. Dabei handelt es sich um den Fehler, der bei manueller Bestimmung der Feature-Werte auftreten kann. Es wurde bereits erläutert, dass die Ähnlichkeit von b) zu a) in Abbildung 5.1 nicht sehr groß ist, weil ein Schreiber, der normalerweise keinen Kringel macht,

plötzlich nicht doch einen schreiben wird. Es kann aber geschehen, dass er die Feder nicht rechtzeitig absetzt und so der Ansatz eines Kringels entsteht, oder dass ein Schreiber eine Einrollung in einigen Fällen nicht sehr deutlich schreibt. In solchen Fällen kann es passieren, dass bei der manuellen Bestimmung ein Wert gewählt wird, der dem anderen möglichen Wert optisch sehr ähnlich ist.

Damit existieren zwei Ähnlichkeitsmaße, die im besten Fall durch zwei verschiedene Distanzfunktionen beschrieben werden und dann zu einer verbunden werden. Das würde bedeuten:

$$d_{f_i} = g(d_{f_i}^a, d_{f_i}^b)$$

Die Feature-Distanzfunktion wäre damit eine Funktion in Abhängigkeit der beiden speziellen Distanzfunktionen. Für die Umsetzung in dieser Diplomarbeit wird der Einfachheit halber von solcher Unterteilung abstrahiert und nur eine einfache Feature-Distanzfunktion d_{f_i} genutzt. Bei der Festlegung der Feature-Distanzfunktionen wird aber darauf geachtet, dass sie beide Aspekte berücksichtigt, sowohl die Wahrscheinlichkeit, dass für den Schreiber mit dem typischen Feature-Wert x der Wert y vorliegt, als auch den Fehler beim Bestimmen der Werte, der aufgrund optischer Ähnlichkeiten auftritt.

Eine zweite Vereinfachung, die für die Entwicklung eines Prototypen gemacht wird, ist der Verzicht auf die Berücksichtigung der Eigenschaft, die in Abschnitt 5.2.5 beschrieben wird. Dort wurde gesagt, es gilt nicht:

$$d(i, j) = d(j, i) \quad \forall i, j$$

Für den Prototypen wird unterstellt, dass eben diese Eigenschaft doch gilt, dass die Distanzfunktion also symmetrisch ist. Damit kann insbesondere in den Fällen wo Distanzmatrizen benötigt werden, die Hälfte der Arbeit gespart werden. Auf das theoretisch korrekte Vorgehen kommt Kapitel 9 wieder zurück. Es müssen nur alle $d(i, j)$ mit $i < j$ bestimmt werden und $d(j, i) = d(i, j)$ kann dann berechnet werden. Im folgenden Abschnitt soll die Frage beantwortet werden, in wie vielen Fällen, solche Distanzmatrizen benötigt werden.

6.2.2 Distanzfunktionen

In Kapitel 5.2.4 wurde zwischen drei Arten von Feature-Distanzfunktionen unterschieden. Es gibt solche mit

1. einfachem Wertevergleich,
2. dem Wertevergleich mit Wertanalyse und
3. dem Wertevergleich basierend auf Wertanalyse und zusätzlichen Informationen.

Es hat sich gezeigt, dass die meisten Features in die dritte Kategorie gehören. Es existieren aber auch einige in der ersten Gruppe.

Feature-Code	Beschreibung
6.1.	Ligatur mit Schlüssel
6.2.	Tonartvorzeichen ligiert
7.1.	Rastral - Größe
7.2.	Rastral - Linienabstände im System
7.3.	Rastral - Anordnung der Systeme
11.1.	Schreibgewohnheiten - Kustoden
11.2.	Schreibgewohnheiten - Vorhandensein von Schlüsseln
11.3.	Schreibgewohnheiten - Vorhandensein von Tonartvorzeichen
11.4.	Schreibgewohnheiten - Format
11.5.	Schreibgewohnheiten - b oder # als Auflösungszeichen
12.	Laufweite

Tabelle 6.1: Features mit Boolescher Feature-Distanzfunktion

Einfacher Wertevergleich

Bei den in Tabelle 6.1 aufgeführten Features reicht ein einfacher Wertevergleich aus, um die Distanz zwischen je zwei Werten x und y zu bestimmen. Sie beträgt:

$$d_{f_i}^{Bool}(x, y) = \begin{cases} 0, & \text{falls } x = y \\ 1 & \text{sonst} \end{cases}$$

Es existiert keine Abstufung der Ähnlichkeiten. Zwei Werte haben entweder die minimale oder maximale Distanz. Bei den Features 7.2., 7.3., 12. und 11.1. bis 11.5. ist der Grund dafür, dass sie nur zwei gegensätzliche Werte annehmen können. Im Fall der Schreibgewohnheiten sind dies meist Ja/Nein-Entscheidungen. Entweder sind Kustoden mehrheitlich vorhanden oder nicht. Das Format kann Hochformat oder Querformat sein.

Für die restlichen Features existieren zwar mehr als zwei Werte, diese haben aber keine Ähnlichkeiten aufzuweisen. Ein Sechs-Millimeter-Rastral gehört nicht mit größerer Wahrscheinlichkeit als ein Neun-Millimeter-Rastral zu einem Schreiber, der normalerweise einen Fünf-Millimeter-Rastral benutzt. Werden zwei verschiedene Rastrale verglichen, kann die Distanz nur eins sein.

Wertevergleich mit Wertanalyse

Streng genommen gehört kein einziges Feature in diese Kategorie. Der Grund dafür ist, dass beim ebenenweisen Vergleich der Feature-Werte, wie er in Kapitel 5.2.4 beschrieben wird, ein boolescher Vergleich nicht ausreicht. Es sind dafür Distanzmatrizen nötig. Da diese aber nur die Werte einer Ebene der Feature-Base beschreiben, sind sie bedeutend kleiner als eine Distanzmatrix für den gesamten Wertebereich. Die Gesamtdistanz zweier Werte wird dann mittels der Distanzen der einzelnen Ebenen der Werte berechnet.

In diese etwas weiter als ursprünglich geplant gefasste Kategorie gehören die Merkmale aus Tabelle 6.2. Das Prinzip soll mit Hilfe des Features 13.1. veranschaulicht werden. Die Werte der Viertelpause haben drei Ebenen. Eine beschreibt das obere Teilstück, eine das mittlere und eine das untere. Abbildung

Feature-Code	Beschreibung
13.1.	Viertelpause
6.5.	# - Vorzeichen
1.1.1.	einzigiges Grundelement des G-Schlüssels

Tabelle 6.2: Features mit Wertevergleich durch Wertanalyse



Abbildung 6.3: Zwei mögliche Werte für das Viertelpause-Attribut

6.3 zeigt zwei mögliche Werte, bei denen sich das obere und untere Element unterscheiden. Diese Unterschiede spiegeln sich in der zweiten und dritten Ebene wieder¹. Zuerst werden die Distanzen für jede Ebene bestimmt. Aus den Distanzmatrizen kommt die Information, dass die Distanz auf der zweiten Ebene 0,2 beträgt. Dabei handelt es sich um die Ähnlichkeit zwischen den oberen Strichen. Die Distanz zwischen den unteren Elementen beträgt 0,1, da sie leicht verwechselt werden können. Die Feature-Distanzfunktion $d_{Viertelpause}$ addiert die Distanzen aller Ebenen und kommt damit auf eine Gesamtdistanz von 0,3. Bei dem Merkmal 1.1.1. (C-Schlüssel, vgl. 2.1.1) sieht die Funktion viel komplizierter aus. Sie umfasst 18 Ebenen und basiert nicht nur auf einfacher Addition der einzelnen Distanzen. Es existieren darüber hinaus zusätzlich noch Ähnlichkeiten zwischen den verschiedenen Ebenen. So haben zum Beispiel Knoten der elften Ebene eine Mindestdistanz von 0,1 zu Knoten der zehnten Ebene und von 0,4 zu Knoten der zwölften Ebene. Diese Ebenenverwandtschaft wird in einer weiteren Distanzmatrix dargestellt.

Wertevergleich mit Wertanalyse und zusätzlichen Informationen

Die meisten Attribute gehören in die dritte Kategorie. Für die nicht in den ersten beiden Kategorien genannten Features ist die einzige Möglichkeit die Feature-Distanzfunktion umzusetzen, jeweils eine Distanzmatrix zu erstellen, die die Distanzen aller Werte des Wertebereichs enthält. Diese Matrizen wurden von Musikwissenschaftlern basierend auf ihrem Fachwissen erstellt und im Excel-Format abgespeichert. Es ist Aufgabe einer Komponente des Prototypen, diese Dateien einzulesen und in ein internes Format umzuformen. Aus Performance-Gründen werden alle möglichen Werte der Feature-Distanzfunktionen aus den ersten beiden Kategorien einmalig berechnet und intern als Distanzmatrix abgespeichert. So müssen die Distanzen nur noch aus den Matrizen abgelesen werden und es sind keine Berechnungen zur Laufzeit nötig.

¹In der vierten und fünften Ebene, wenn das Suffix '13.1.', das für das Feature steht, mitgezählt wird.

6.2.3 Nullwerte

In Abschnitt 5.2.7 wurden zwei verschiedene Nullwerte beschrieben:

- Non-Information-Null (?)
- No-Applicable-Null (\top)

Ein Non-Information-Null soll das Klassifikationsergebnis nicht beeinflussen - weder positiv noch negativ. Darum wird eine Feature-Distanzfunktion d_{f_i} nur dann zur Berechnung herangezogen, wenn ihre beiden Argumente ungleich '??' sind. Eine Feature-Distanzfunktion sei **anwendbar**, wenn keines ihrer beiden Argumente mit einem Non-Information-Null belegt ist:

$$d_{f_i}(v_1, v_2) \text{ ist anwendbar} \Leftrightarrow v_1, v_2 \in W_i \wedge v_1 \neq ? \wedge v_2 \neq ?$$

Die Distanzfunktion für zwei Feature-Vektoren $\gamma_a = (v_1^a, \dots, v_n^a)^T$ und $\gamma_b = (v_1^b, \dots, v_n^b)^T$, basierend auf der gewichteten, normierten Hamming-Distanz, die bisher

$$d_\Gamma = \frac{\sum_{i=1}^n w_i * d_{f_i}}{\sum_{i=1}^n w_i}$$

war, lautet dann:

$$d_\Gamma = \frac{\sum_{\forall i: d_{f_i} \text{ anwendbar}} w_i * d_{f_i}}{\sum_{\forall i: d_{f_i} \text{ anwendbar}} w_i}$$

Analog sehen andere Distanzfunktionen aus. Besonders wichtig ist, dass die Formel das in Abschnitt 4.1 beschriebene Anfrage-Szenario unterstützt. Typisch für eine Anfrage ist ja, dass sie extrem viele Nullwerte (?) enthält.

Die Behandlung des No-Applicable-Null wurde während der Entwicklungszeit dieser Diplomarbeit immer wieder diskutiert. Lange Zeit war der Ansatz, die Entscheidung, ob es sich bei einem Nullwert um ein '?' oder ' \top ' handelt, vom Feature abhängig zu machen. Das bedeutet, dass es eine kleine Gruppe von Merkmalen gibt, bei denen ein fehlender Wert typisch für den Schreiber ist und eine andere Gruppe, wo das nicht der Fall ist. Fehlt zum Beispiel das linke Element beim C-Schlüssel, heißt das nicht, dass diese Information nicht bekannt ist, sondern dass es sich um eine typische Eigenschaft des Schreibers handelt. Der beschriebene Ansatz ist sehr anwenderfreundlich, da der Nutzer nicht den Unterschied zwischen den Nullwerten kennen muss und die Unterscheidung automatisch gemacht werden kann. Genau das ist aber auch der Nachteil dieser Methode, denn manchmal kann diese Entscheidung nicht automatisch getroffen werden und der Nutzer muss selbst entscheiden, ob der Wert nur nicht bekannt ist oder ob es eine typische Eigenschaft des Schreibers ist.

Deshalb wird im Schreibererkennungssystem explizit zwischen zwei Nullwerten unterschieden. Ein Non-Information-Null wird weiterhin durch einen fehlenden Wert des Features beschrieben. So können die beschriebenen Verfahren weiterhin benutzt werden. Für das No-Applicable-Null wird ein neuer Feature-Wert eingeführt, der zu allen anderen Werten des Wertebereichs die Distanz

Wert	NW	Bedeutung	Erklärung
”	?	Non-Information-Null	Es liegen keine Informationen vor, wie der Schreiber das Symbol schreibt.
'1.2.1.0.'	⊤	No-Applicable-Null	Fehlen des Symbols ist typisch für den Schreiber.

Tabelle 6.3: Nullwerte am Beispiel des linken C-Schlüssel-Elementes (1.2.1.)

eins hat. Dieser Wert liegt in der Feature Base direkt unter dem Feature-Knoten und hat die bisher noch unbelegte Nummer (ν) null (0). Der Nutzer kann so explizit zwei verschiedene Nullwerte angeben. Tabelle 6.3 zeigt dies am Beispiel des linken Elementes des C-Schlüssels (1.2.1.). Wird im Folgenden von einem Nullwert gesprochen, ist immer das Non-Information-Null (?) gemeint, da das No-Applicable-Null (⊤) wie ein normaler Wert des Wertebereichs behandelt wird.

6.2.4 Features mit komplexen Werten

Die oben beschriebenen Auswirkungen von Nullwerten auf die Distanzfunktion lassen sich auch auf Features mit komplexen Werten übertragen (vgl. Abschnitt 5.2.6). Eine komplexe Feature-Distanzfunktion heißt anwendbar, wenn keines ihrer Argumente die leere Menge ist:

$$d_{f_i}^K(V_1, V_2) \text{ ist anwendbar} \Leftrightarrow V_1, V_2 \subseteq W_i \wedge V_1 \neq \emptyset \wedge V_2 \neq \emptyset$$

Die komplexe Distanzfunktion lautet dann:

$$d_{\Gamma}^K = \frac{\sum_{\forall i: d_{f_i}^K \text{ anwendbar}} w_i * d_{f_i}^K}{\sum_{\forall i: d_{f_i}^K \text{ anwendbar}} w_i}$$

Die Implementierung dieser Distanzfunktion wird in Abschnitt 7.3.2 erläutert.

6.3 Bewertungsmaße

Im Laufe der Entwicklung eines Software-Systems ist es nötig, das Modell und die Leistung des Systems zu bewerten. Damit kann das Modell validiert und gegebenenfalls optimiert werden. Im folgenden Abschnitt werden einige Maße vorgestellt, die der Bewertung des DM-Modells und der Ausgabe des Schreibererkennungssystems dienen.

6.3.1 Bewertung der Partition

Das erste Maß dient der Bewertung der Distanzfunktion und ihrer Parameter. Dazu soll eine abgewandelte Form der Bewertungsfunktion genutzt werden, die in Abschnitt 5.1.2 für Clustering-Verfahren eingeführt wurde. Es handelt

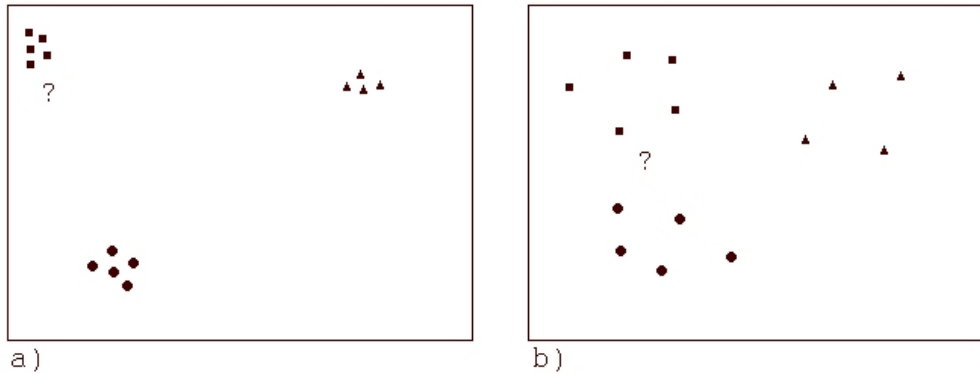


Abbildung 6.4: Eine gute (a) und eine schlechte (b) Partition

sich dabei um eine Bewertungsfunktion für instanzbasierte Cluster-Verfahren, die auf das vorliegende, instanzbasierte Klassifikationsmodell angewandt wird. Dazu wird eine **Funktion (SF, Scoring Function)** gewählt, welche die Anforderung an eine Partition von Clustern direkt umsetzt: Instanzen eines Clusters müssen eine kleine Distanz haben, Instanzen verschiedener Cluster eine große.

$$SF = \frac{\sum_{\forall(\gamma_1, \gamma_2) \in \Gamma_{\neq}^2} d_{\Gamma}(\gamma_1, \gamma_2) / |\Gamma_{\neq}^2|}{\sum_{\forall(\gamma_1, \gamma_2) \in \Gamma_{=}^2} d_{\Gamma}(\gamma_1, \gamma_2) / |\Gamma_{=}^2|}, \text{ mit}$$

$$\Gamma_{=}^2 = \{(\gamma_1, \gamma_2) | \gamma_1 \neq \gamma_2, \gamma_1 \text{ und } \gamma_2 \text{ sind Instanzen der gleichen Klasse}\}$$

$$\Gamma_{\neq}^2 = \{(\gamma_1, \gamma_2) | \gamma_1 \text{ und } \gamma_2 \text{ sind Instanzen verschiedener Klassen}\}$$

Im Nenner von SF wird der durchschnittliche Abstand aller Instanzen gleicher Klassen bestimmt. Das ist der Hauptunterschied zur ursprünglich vorgestellten Funktion, denn dort wurden jeweils die Abstände der Instanzen zum Cluster-Mittelpunkt berechnet. Beide Varianten sind aber ein Maß für die Dichte/Enge ('tightness', [35]) eines Clusters. Es wird der Vergleich einer Instanz mit sich selbst außen vor gelassen, da diese Distanzen immer Null sind, und keine Aussage über die Qualität des Data-Mining-Modells erlauben. Im Zähler werden dementsprechend die Distanzen von Feature-Vektoren verschiedener Schreiber berechnet. In einer optimalen Partition ist der Nenner möglichst klein und der Zähler möglichst groß. Das Ziel ist die Maximierung von SF.

Im Falle des Schreibererkennungssystems muss zur Berechnung von SF zuvor kein Clustering-Verfahren ausgeführt werden, da dessen Ergebnis schon bekannt ist. In der Testmenge ist die Information, welche Feature-Vektoren zu welchen Schreibern gehören, bereits vorhanden. Es wird also angenommen, dass ein Clustering-Algorithmus alle Cluster korrekt erzeugt hat und seine Ausgabe genau die Testmenge ist. Damit lässt SF, angewandt auf die Testmenge, Rückschlüsse auf die Qualität der Distanzfunktion und ihrer Parameter zu. Das Prinzip wird in Abbildung 6.4 veranschaulicht. Sie zeigt zwei Partitionen derselben Menge von Instanzen in Cluster, die sich nur durch die Wahl der

Distanzfunktion unterscheiden. Abbildung 6.4 a) würde eine bessere Bewertung als Abbildung 6.4 b) bekommen, da die Cluster selbst kleiner sind und untereinander besser abgrenzbar. Im Schaubild ist auch zu erkennen, dass eine neue Instanz (symbolisiert durch das Fragezeichen) in Szenario a) viel besser einer Klasse zugeordnet werden kann, als in der zweiten Variante. Es ist deshalb das Ziel, SF bei allen Tests zu maximieren.

6.3.2 Bewertung der Antwortmenge A_{FV}

Dieses Bewertungsmaß allein reicht allerdings noch nicht aus. Letztendlich soll der Nutzer in der Lage sein, zu einer Schreibercharakteristik (γ_q) eine Menge ähnlicher Schreibercharakteristiken (A_{FV}) zu erhalten, die ihn bei der Entscheidung unterstützen, den zugehörigen Schreiber zu bestimmen. Das genaue Vorgehen wurde bereits in Abschnitt 6.2 beschrieben. Es wird ein Maß benötigt, dass die Qualität der Antwortmenge A_{FV} widerspiegelt. Solche Bewertungen sind aus dem Bereich des Information Retrieval bekannt.

In [29] heißt es: *Beim Information Retrieval (IR) geht es darum, Nutzende mit Informationen zu versorgen. Das IR System ist dabei ein Werkzeug mit dem Informationen ausgewählt werden können. Zur Interaktion mit diesem Werkzeug muss zum einen der Informationsbedarf der Nutzenden dem System in geeigneter Form übermittelt werden; zum anderen müssen die gefundenen Informationen bzw. eine geeignete Darstellung dieser Informationen den Nutzenden präsentiert werden.* Typisch für diesen Bereich ist eine inhaltsorientierte Suche auf Textdokumenten. Beim Vektorraum-Retrieval werden Dokumente und Anfragen als Punkte in einem hochdimensionalen, metrischen Vektorraum repräsentiert, zwischen denen eine Distanz definiert ist. Bis auf den Fakt, dass unser Modell auf einer eigenen Metrik basiert, da die herkömmliche Metrik hier nicht anwendbar ist (vgl. Abschnitt 5.1), kann das IR-Modell auf das Schreibererkennungssystem übertragen werden. Es handelt sich dann nicht mehr um die Suche nach ähnlichen Textdokumenten, sondern nach ähnlichen Feature-Vektoren.

Damit können die aus dem IR-Bereich bekannten Maße zur Evaluation der Ergebnisse genutzt werden. [54] nennt die drei gebräuchlichsten Maße:

$$precision = \frac{|A \cap B|}{|B|}$$

$$recall = \frac{|A \cap B|}{|A|}$$

$$fallout = \frac{|\bar{A} \cap B|}{|\bar{A}|}$$

A - Menge aller relevanten Feature-Vektoren

B - Menge aller (in der Antwort) erhaltenen Feature-Vektoren

\bar{A} - Menge aller nicht relevanten Feature-Vektoren

\bar{B} - Menge aller nicht erhaltenen Feature-Vektoren

Übertragen auf das Schreibererkennungssystem bedeutet das folgendes. Es wird eine Anfrage (γ_q) nach dem Schreiber gestellt, zu dem ein bestimmter

Feature-Vektor gehört. Die erwartete Antwortmenge (A) enthält alle Feature-Vektoren des gleichen Schreibers. Die tatsächlich erhaltene Antwortmenge ($B=A_{FV}$) umfasst die k nächsten Nachbarn des Anfrage-Feature-Vektors, was im besten Falle mit der erwarteten Menge übereinstimmt. Die erste Formel ‘precision’ gibt den Anteil der Feature-Vektoren in der erhaltenen Antwortmenge an, die auch wirklich relevant sind. Sie ist ein Maß für die Qualität des Ergebnisses. Der zweite Wert ‘recall’ bestimmt den Anteil der erwarteten Feature-Vektoren, die dann auch tatsächlich in der Antwortmenge enthalten sind und ist damit ein Maß für die Quantität des Ergebnisses. Das Maß ‘fallout’ ist nicht ganz so gebräuchlich. Es setzt die Zahl der nicht-relevanten Vektoren in der Antwort ins Verhältnis zu allen nicht-relevanten Vektoren. Precision und Recall sind in der Praxis von einander abhängig. Das eine kann nur auf Kosten des anderen verbessert werden.

6.3.3 Bewertung der Antwortmenge A_S

Es wurde bereits in Abschnitt 6.2 erläutert, dass es in dieser Arbeit nicht um das Finden eines optimalen Rankings geht. Darum soll an dieser Stelle ein einfaches Maß definiert werden, das genutzt werden kann, um die Antwortmenge A_S zu bewerten. Für eine Menge von Anfragen sei:

$$K = \frac{X}{N}, \text{ wobei}$$

X = Anzahl aller Antwortmengen, in denen der erwartete Schreiber das beste Ranking hat.

N = Anzahl aller Anfragen.

Damit wird die IR-Anfrage wieder auf die automatische Klassifikation zurückgeführt. Entweder die richtige Klasse wird erkannt oder nicht. K gibt somit den Anteil aller Anfragen an, in denen die Klasse korrekt bestimmt wurde. Für den Prototypen wurde eine einfache Variante implementiert, um den Schreiber zu bestimmen. Sie wählt den Schreiber aus, zu dem in A_{FV} die meisten Handschriften gehören. Trifft das auf mehrere zu, wird der mit der geringsten Distanz zu γ_q gewählt. Dieser Funktion genügen schon sehr kleine Antwortmengen, was mit besseren Werten bei der Precision, jedoch auf Kosten des Recalls, einher geht. Es sei hier noch einmal darauf hingewiesen, dass es sich bei dieser Umsetzung nur um eine einfache Möglichkeit handelt. Eine Verbesserung wäre es zum Beispiel, den Einfluss der einzelnen Feature-Vektoren zu gewichten und zu normalisieren. Dies ist allerdings nicht Thema der vorliegenden Arbeit und wird deshalb im Ausblick (Abschnitt 9.2) wieder aufgegriffen. K wird bei der Entwicklung einer geeigneten Ranking-Funktion eine größere Rolle spielen, in den in Kapitel 8 beschriebenen Tests wird K allerdings bereits verwendet.

6.3.4 Einfluss von Nullwerten

Eine der entscheidenden Anforderungen an das Schreibererkennungssystem (vgl. Abschnitt 4.2.3) ist die angemessene Behandlung von Nullwerten (‘?’),

vgl. Abschnitt 6.2.3). Sie sollen die Distanz zwischen zwei Feature-Vektoren weder positiv noch negativ beeinflussen. Es ist eine typische Eigenschaft der Feature-Vektoren, dass sie sehr viele Nullwerte enthalten. Wenn ein Schreiber auf einem Notenblatt ein bestimmtes Symbol nicht benutzt, kann dieses auch nicht klassifiziert werden und wird mit einem Nullwert belegt. Der Anteil der Nullwerte beträgt um die 50 Prozent. Sie haben dadurch insofern einen Einfluss auf die Berechnung der Distanz, als es sein kann, dass weniger gemeinsame Features für den Vergleich übrig bleiben, weil für die Berechnung der anderen Feature-Distanzfunktionen mindestens ein Wert nicht bekannt ist. Es wurden zwei Maße eingeführt, um die Bedeutung von Nullwerten für die Distanzbestimmung zu testen:

$N_F : \Gamma \rightarrow [0..1]$, wobei

$$N_F(\gamma) = \frac{\sum_{v_i: v_i = \text{null}} w_i}{\sum_{v_i} w_i}$$

$N_D : D \rightarrow [0..1]$, wobei D die Menge aller Distanzfunktionen ist und

$$N_D(d(\gamma_a, \gamma_b)) = \frac{\sum_{v_i: v_i^a = \text{null} \vee v_i^b = \text{null}} w_i}{\sum_{v_i} w_i}$$

Das erste Maß gibt den gewichteten Anteil der Features in einem Feature-Vektor an, die mit einem Nullwert belegt sind. Das zweite Maß gibt über den Anteil der Merkmale Auskunft, die nicht zur Berechnung der Distanz herangezogen werden konnten, da mindestens einer der beiden zu vergleichenden Werte ein Nullwert ist. N_F und N_D sollen in den Tests, die in Kapitel 8 beschrieben werden, genutzt werden, um die Ergebnisse nach dem Vorkommen von Nullwerten zu beurteilen.

In diesem Kapitel wurde dargestellt, wie die in Kapitel 6 aufgestellten Anforderungen an das System in Form von Strukturen, Methoden und Algorithmen umgesetzt werden können. Darüber hinaus wurden Bewertungsmaße beschrieben, die genutzt werden können, um das zu entwickelnde Schreibererkennungssystem zu testen und zu bewerten. Das folgende Kapitel illustriert die prototypische Implementierung der entwickelten Strukturen und Verfahren.

Kapitel 7

Implementierung

7.1 Nutzen vorhandener Tools

In diesem Kapitel soll untersucht werden, inwieweit bereits existierende Data-Mining-Tools für die auf den Strukturen und Methoden basierende Umsetzung (vgl. Kapitel 6) des Schreibererkennungssystems genutzt werden können. Die hier untersuchten Programme wurden schon in Abschnitt 3.2 eingeführt. Sie müssen einige notwendigen Eigenschaften erfüllen, die in der Reihenfolge ihrer Wichtigkeit aufgeführt werden. Folgendes muss möglich sein:

1. Unterstützung eines instanzbasierten Klassifikationsverfahrens (k-Nearest-Neighbor)
2. Wahl der Distanzfunktion
3. Anpassen der Gewichte der Distanzfunktion
4. Angabe einer Distanzmatrix für jedes Attribut
5. Unterstützung von Nullwerten

Die Motivation ist es im Allgemeinen, ein bereits existierendes Tool zu nutzen, um den Programmieraufwand zu verringern und auf schon getestete Software zu setzen, und damit viele Fehlerquellen auszuschließen. Diese beiden Gründe sind zwar auch in einem gewissen Maß auf das Schreibererkennungssystem zu übertragen, allerdings fallen sie nicht sehr stark ins Gewicht, da sowohl die Hauptarbeit, als auch die Hauptfehlerquelle im DM-Modell selbst liegt und nicht in der Implementierung des Algorithmus. Darum sollte ein Tool zusätzliche Funktionalität bieten, die sich in Zukunft als nützlich erweisen kann, wie zum Beispiel die Möglichkeit zur Visualisierung des DM-Modells.

Die meisten Tools erfüllen schon die erste und wichtigste Anforderung nicht, denn sie unterstützen keine instanzbasierten Klassifikationsverfahren. Das betrifft im Einzelnen:

- Clementine
- Enterprise Miner

Tool	Darwin	Weka	MLC++
Wahl der Distanzfunktion	±	±	±
Wahl der Gewichte	+	–	+
Distanzmatrizen	–	+	–
Unterstützung der Nullwerte	±	±	±
Unterstützung komplexer Werte	–	–	–

Tabelle 7.1: Übersicht über die Anforderungen an die Tools (+: erfüllt –: nicht erfüllt, ±: nicht im benötigten Maße erfüllt)

- Intelligent Miner
- Model 1

Damit bleiben nur noch die Programme Darwin, MLC++ und Weka übrig, die im folgenden genauer betrachtet werden. Einen Überblick über die Ergebnisse gibt Tabelle 7.1.

Darwin

Darwin führt ein instanzbasiertes Klassifikationsverfahren unter dem Namen 'Match Model'. Das 'Match Model' basiert auf dem k-Nearest-Neighbor-Verfahren. Standardmäßig setzt Darwin $k = 2$ und bestimmt alle weiteren Parameter mit Hilfe einer Trainingsmenge. Der Nutzer kann allerdings k und die Gewichte der Distanzfunktion selbst angeben und auf den Trainingslauf verzichten. Zur Behandlung von Nullwerten bietet Darwin zwar Funktionalität zur Vorverarbeitung der Daten. Diese reicht aber nicht für die in Abschnitt 6.2.3 formulierte Anforderung aus. Die Möglichkeiten zur Visualisierung sind nach [38] bei Darwin begrenzt.

Für Darwin spricht seine einfache Bedienung. Allerdings erfüllt es die in Tabelle 7.1 genannten Anforderungen nur teilweise und kann als externes Tool nicht direkt in die DB2-Datenbankumgebung integriert werden.

Weka

Das Weka-Tool bietet zwei Klassen basierend auf dem k-Nearest-Neighbor-Algorithmus: IB1¹ und IBk. Der Nutzer kann zwar k festlegen, hat aber keinen Einfluss auf die Wahl der Distanzfunktion. Die Gewichte werden anhand einer Trainingsmenge bestimmt und können nicht manuell beeinflusst werden. Es ist aber möglich, eine Distanzmatrix anzugeben.

Die ursprüngliche Klasse IBk erfüllt somit auch nur wenige der an sie gestellten Anforderungen. Da der Quellcode frei zugänglich ist, könnte er allerdings an das Schreibererkennungssystem angepasst werden. Der Aufwand käme jedoch dem einer vollständigen Neuentwicklung gleich.

¹kurz für: InstanzBasiert mit $k = 1$

MLC++

MLC++ definiert für Klassifikationsverfahren eine abstrakte Klasse namens *Inducers*. Eine Unterklasse davon ist *IB*, wobei es sich um eine Implementierung von [20] handelt. Nach [41] ist bei der Funktion die Wahl der Gewichte möglich.

Für MLC++ gilt das Gleiche wie für das Weka-Tool. Der Aufwand zur Anpassung der Klassen an die Anforderungen des Schreibererkennungssystems käme dem einer Neuentwicklung gleich. Darum wird im Prototyp keines der genannten Tools eingesetzt. Es kann aber durchaus von Nutzen sein, zu einem späteren Zeitpunkt noch einmal auf diese Programme zurückzukommen, wenn es um die Frage der Visualisierung der Ergebnisse geht. Das ist sinnvoll, falls Abstriche bei den genannten Anforderungen gemacht werden können, um dafür die Möglichkeiten eines Tools zur Visualisierung zu nutzen. Das Weka-Tool wird im Rahmen dieser Arbeit trotzdem Anwendung finden, nämlich wenn es in Abschnitt 8.1 um die Bestimmung der Prioritäten der Attribute mit Hilfe von Klassifikationsverfahren geht.

Nachdem in diesem Abschnitt existierende Programme diskutiert wurden, soll in den folgenden beiden Abschnitten gezeigt werden, wie das Schreibererkennungssystem praktisch umgesetzt werden kann.

7.2 Integration in die Datenbank

Ein entscheidender Aspekt der Umsetzung ist die Integration in die DB2-Umgebung. Eine vernünftige Integration hat mehrere bedeutende Vorteile:

Transparenz für den Nutzer In diesem Fall muss der Nutzer (hier auch Client genannt) nicht der Endnutzer sein, sondern es kann sich dabei auch um eine Anwendung handeln, welche die Funktionalität des Schreibererkennungssystems nutzen will. Transparenz bedeutet, dass der Client nicht mehr über den internen Aufbau des Systems wissen muss, als unbedingt nötig. Das hat einerseits den Vorteil, dass die Anfragen so einfach wie möglich gestaltet und andererseits, dass Änderungen an der internen Struktur vorgenommen werden können, ohne dass der Nutzer dadurch beeinträchtigt wird.

Performance-Steigerung Für die Performance des Systems ist es von Vorteil, wenn die Kommunikationswege zwischen Schreibererkennungssystem und Datenbank so kurz wie möglich gehalten werden und der Datenaustausch auf ein Minimum beschränkt wird. Im schlechtesten Fall würden sämtliche Feature-Vektoren an den Client geschickt, um dort die nächsten Nachbarn zu berechnen, im besten Fall nur die endgültigen Ergebnisse.

Einmalige Installation Ein weiteres Ziel ist es, eine einmalige Installation des Schreibererkennungssystems auf einem Server zu ermöglichen. Eine mehrmalige Client-seitige Installation soll vermieden werden. Unter anderem trägt dieser Ansatz auch zu einer Performance-Steigerung des Systems bei.

Im vorherigen Abschnitt wurde gezeigt, dass die Ausnutzung der Funktionalität der Intelligent Miner zur Integration des Systems in die Datenbank leider nicht möglich ist. Im Folgenden sollen weitere Ansätze vorgestellt werden, welche die Integration dennoch gewährleisten.

Die Umsetzung der Transparenz wurde in den Architekturbeschreibungen des Systems in den Abbildungen 4.1 (Seite 39), 4.2 (Seite 40) und 6.2 (Seite 57) schon vorweggenommen. Dort ist zu erkennen, dass der Nutzer nie direkt auf die Tabellen der Datenbank zugreift, sondern immer über eine Schnittstelle (Interface). Dafür bietet DB2 nutzerdefinierte Funktionen (User-Defined Functions, UDFs) und Stored Procedures an. Diese werden weiter unten ausführlich erläutert und diskutiert. Im weiteren Verlauf des Projektes können auch neue Datentypen (User-Defined Datatypes, UDTs) definiert werden, die typische Datentypen des Schreibererkennungssystems und Operationen darauf unterstützen. Zum Beispiel wäre ein Datentyp für einen Knoten der Feature Base möglich oder auch nur für den Code des Knoten. Operationen darauf könnten dann beispielsweise Verwandtschaften zwischen zwei Knoten feststellen. UDTs werden in dieser Arbeit aber nicht weiter betrachtet.

User-Defined Functions

DB2 bietet die Möglichkeit, die Funktionalität einer Anwendung zu verbessern, indem Programme auf dem Server laufen können. Es bietet die oben genannten Stored Procedures und UDFs an. UDFs erlauben es, die Datenbankanfragesprache SQL zu erweitern. Sie werden nach ihrem Rückgabedatentyp unterschieden:

Eine Scalar Function gibt bei jedem Aufruf einen einzelnen Wert zurück. Ein Beispiel dafür ist die Funktion, die für eine gegebene Zeichenkette die Anzahl der enthaltenen Zeichen bestimmt.

Eine Column Function benötigt eine Menge von Werten (column) und gibt einen Wert zurück. Sie ist auch unter dem Namen Aggregatfunktion bekannt. Mit ihrer Hilfe kann zum Beispiel der Durchschnittswert einer Spalte berechnet werden.

Eine Table Function gibt eine ganze Tabelle zurück. Sie kann in der FROM-Klausel einer Datenbankanfrage benutzt werden.

Das Schreibererkennungssystem soll die nächsten Nachbarn eines gegebenen Feature-Vektors berechnen. Der Rückgabewert ist die Menge A_{FV} , bzw. A_S , die bereits aus Abschnitt 6.2 bekannt ist. A_{FV} und A_S lassen sich gut in Tabellenform darstellen, daher wäre eine Table Function am geeignetsten. Die Anfrage

```
SELECT * FROM ENOTE.TABLE_FUNCTION(FEATUE_VEKTOR)
```

könnte dann die möglichen Schreiber des gegebenen Feature-Vektors liefern. Eine zweite Spalte der Tabelle könnte das Ranking der Schreiber beinhalten. Leider können aus Table Functions heraus keine SQL-Anfragen gestellt werden,

<pre>public static void A_VF (String featureValue01, . . String featureValue78, int[] errorCode, String[] errorMsg, ResultSet[] rs)</pre>	<pre>public static void A_S (String featureValue01, . . String featureValue78, int[] errorCode, String[] errorMsg, ResultSet[] rs)</pre>
--	---

Tabelle 7.2: Die Definition der Stored Procedures in Java

so dass nicht auf die Feature-Vektoren in der Datenbank zugegriffen werden kann. Damit sind UDFs für diesen Zweck nicht einsetzbar und es müssen Stored Procedures genutzt werden.

7.2.1 Stored Procedures

Stored Procedures sind Prozeduren, die auf dem DB2-Server liegen. Damit werden die oben genannten drei Ziele der Integration in eine Datenbank umgesetzt. Stored Procedures erlauben Eingabe- (IN) und Ausgabeparameter (OUT) und außerdem solche, die beides (IN/OUT) sind. Es besteht die Möglichkeit, ein `ResultSet`² an die aufrufende Methode zurückzugeben. Für das Schreibereerkennungssystem bedeutet das Folgendes: Es werden zwei Stored Procedures benötigt, welche die in 6.2 beschriebene Funktionalität umsetzen. In Abbildung 7.2 werden diese beiden Methoden `A_VF` und `A_S` gezeigt. Beide benötigen einen Feature-Vektor als Eingabe. Die Werte für die einzelnen Merkmale werden einzeln an die Prozeduren übergeben. Diese haben damit 78 IN-Parameter. Dazu kommen zwei OUT-Parameter, die einen eventuell auftretenden Fehler-Code und die dazugehörige Fehler-Meldung enthalten. Der letzte Parameter ist für das `ResultSet` bestimmt. Auf die Implementierung der Schnittstellen wird in Abschnitt 7.3.2 genauer eingegangen.

Damit die Prozedur aus der Datenbankumgebung heraus aufgerufen werden kann, muss sie in die Datenbank eingebunden werden. Dafür dient das ‘CREATE PROCEDURE’-Kommando [14], das in Abbildung 7.3 gezeigt wird. In der Abbildung wird in den Zeilen zwei bis sechs deutlich, wie die IN- und OUT-Parameter deklariert werden. Als nächstes wird angegeben, dass genau ein `ResultSet` zurückgegeben wird. Die Zeilen acht und neun bedeuten, dass der Datenbank-Manager die Stored Procedures als Methoden einer Java-Klasse aufruft, und dass die Parameterübergabe den Konventionen der Java-Spezifikation entspricht. Die Spezifikation beinhaltet zum Beispiel, dass IN/OUT- und OUT-Parameter als ein-elementige Arrays behandelt werden. Auf Grund dieser Einstellung können auch keine internen Datenbankinformationen an die Methode übergeben werden (Zeile zehn). Die Stored Procedure nutzt einen eigenen Speicherbereich (FENCED) und kann sowohl SQL-Anfragen als auch -Updates ausführen (MODIFIES SQL DATA). Die dreizehnte Zeile besagt nur, dass die

²Ein `ResultSet` ist ein Zeiger (Cursor), mit dem sequenziell auf die einzelnen Tupel einer DB-Tabelle zugegriffen werden kann.

Zeile	Anweisung
1	CREATE PROCEDURE A_VF
2	(
3	IN FEATURE01 VARCHAR(100), ...
4	OUT sqlCode INTEGER,
5	OUT errorMsg VARCHAR(255)
6)
7	DYNAMIC RESULT SETS 1
8	LANGUAGE JAVA
9	PARAMETER STYLE JAVA
10	NO DBINFO
11	FENCED
12	MODIFIES SQL DATA
13	PROGRAM TYPE SUB
14	EXTERNAL NAME 'enh:de.enotehistory.SP.A_VF'

Tabelle 7.3: Einbindung der Stored Procedure in die Datenbank

Prozedurargumente einzeln übergeben werden. Als letztes werden das JAR-File, das Java-Package und die Java-Methode bestimmt, auf die sich die Stored Procedure bezieht.

Damit steht fest, wie der Nutzer auf die Daten zugreifen kann. Der folgende Abschnitt zeigt, wie die Daten intern als Tabellen (Relationen) in der Datenbank realisiert werden.

7.2.2 Relationenschema des Schreibererkennungssystem

Das Relationenschema für die Daten des eNoteHistory-Projektes ist in Anhang A.7 komplett dargestellt. Es handelt sich dabei um eine Momentaufnahme, da das Schema noch häufigen Änderungen und Anpassungen unterworfen ist. Es ermöglicht die Speicherung der Feature Base, der Schreiber, Werke, Schreibercharakteristiken, etc. Hier soll nur auf die Relationen eingegangen werden, die direkt für diese Arbeit von Interesse sind: die Feature Base, die Feature-Vektoren und die Distanzmatrizen.

Feature Base

Die Knoten des Feature-Base-Baumes werden in der Tabelle 'NODES' gespeichert. Dort existiert für jeden Knoten ein Eintrag. Dieser verweist zum Beispiel auf den Vaterknoten und gibt den Knotentyp (τ , siehe 5.2.2) an. Der Primärschlüssel ist zur Zeit noch der Knoten-Code (PATH, siehe 5.2.2), wird in Zukunft aber eine generierte ID sein, damit eine Änderung des Codes einfach möglich ist. Darüber hinaus enthält die Relation 'PICTOGRAM' Piktogramme und einen Verweis (Fremdschlüssel) auf den zugehörigen Knoten der Feature Base. Damit ist es möglich, Bilder für die Merkmale oder Merkmalswerte zu speichern.

Feature-Vektoren

In der Tabelle ‘FEATUREVECTORS’ werden allgemeine Informationen über die Schreibercharakteristiken gespeichert, zum Beispiel ein Verweis auf den Schreiber und das Werk aus dem sie stammen. Die eigentlichen Merkmalswerte werden in einer separaten Relation abgelegt. Damit wird dem Umstand Rechnung getragen, dass Merkmale mehrere Werte aufweisen können. Die Tabelle ‘FVVALUES’ enthält neben dem Verweis auf den zugehörigen Feature-Vektor das Merkmal und seinen Wert. Hier gilt das Gleiche wie für die Feature Base: zur Identifizierung des Merkmals und der Werte dient noch der Code des zugehörigen Knoten, in Zukunft wird dies eine generierte ID sein.

Distanzmatrizen

In Kapitel 6.2.2 wurde begründet, warum für alle Feature eine interne Distanzmatrix erzeugt werden soll. Für viele Attribute ist sie sowieso nötig. In allen anderen Fällen verbessert es die Performance des Systems, wenn die Distanzen einmalig vorberechnet und dann nur noch abgelesen werden müssen. Die Distanzmatrizen werden in der Tabelle ‘DISTANCES’ gespeichert. Die Tabelle besteht aus vier Spalten, die für je ein Wertepaar die zugehörige Distanz enthalten. Eine der Spalten gibt das Merkmal an, zu dem die beiden Werte gehören. Die Distanz zweier Werte x und y ist auf diese Weise einfach und effizient durch die SQL-Anfrage

```
SELECT DISTANCE FROM DISTANCES WHERE VALUE1=x AND VALUE2=y
```

zu bestimmen. Ein Index über den Attributen ‘VALUE1’ und ‘VALUE2’ verbessert dabei die Performance. In die Relation werden allerdings nur die Tupel eingetragen, für welche die Distanz kleiner als die Maximaldistanz (1, 00) ist. Damit kann der Speicherbedarf um ein Vielfaches verringert werden, weil nur eine geringe Anzahl von Wertepaaren eine Distanz kleiner als Eins aufweist.

In diesem Kapitel wurde gezeigt, wie das Schreibererkennungssystem in die Datenbank integriert wird. Eine gute Einführung in das Gebiet der Datenbanken geben [36] und [37]. Ausführliche Literatur zu IBMs Datenbank DB2 ist beispielsweise mit [14], [11], [10], [12] und [13] reichlich vorhanden.

7.3 Prototypische Implementierung

In diesem Kapitel wird erläutert, wie die in den vorherigen Kapiteln beschriebenen Algorithmen und Strukturen prototypisch in Java 1.4 umgesetzt werden. Die Java-Klassen werden sich in einer JAR-Datei namens ‘enh’ (für eNoteHistory) in dem Package ‘de.enotehistory’ befinden. Zur Kommunikation mit dem Datenbanksystem wird die JDBC-Schnittstelle eingesetzt.

Das Kapitel unterteilt sich in zwei Abschnitte. Im ersten Teil werden Java-Tools vorgestellt, welche die benötigten Daten in die Datenbank einfügen und

im zweiten Teil wird die Implementierung der Stored Procedures zur Schreibererkennung diskutiert.

7.3.1 Import Tools

Bevor das Schreibererkennungssystem eingesetzt werden kann, müssen die in Abschnitt 7.2.2 beschriebenen Relationen mit Daten gefüllt werden. Es müssen

- die Feature Base,
- die Distanzmatrizen und
- eine Testmenge von Feature-Vektoren

in die Datenbank eingefügt werden. Diese Anwendungen befinden sich in dem Package ‘de.enotehistory.import’ und werden im Folgenden genauer vorgestellt.

Import der Feature Base

Die Feature Base wurde von den Musikwissenschaftlern in Gestalt von HTML-Seiten formuliert. Für alle Kinder eines Knoten existiert jeweils eine Seite. Jedes dieser Geschwister verlinkt eine neue Seite, die wiederum dessen Kindknoten enthält. Blattknoten sind nicht weiter verlinkt. Falls vorhanden, wird zu den einzelnen Knoten zusätzlich ein Piktogramm angezeigt. Die Link-Struktur dieser Menge von HTML-Seiten entspricht genau den Pfaden des Feature-Base-Baum. Das Kommandozeilen-Tool ‘populateFeatureBaseTable’ führt einen Tiefendurchlauf auf dieser Struktur durch und fügt dabei die Knoten, die es antrifft, in die Datenbank ein. Es benutzt den HTML-Parser aus der Java-Standard-Bibliothek. Beim Besuch einer HTML-Seite entscheidet es zuerst, ob die Knoten durch Piktogramme beschrieben werden oder nicht. Für beide Fälle gibt es eine separate Methode, da das Vorgehen verschieden ist. Enthält die Seite Bilder, werden diese als Separator der einzelnen Knotenbeschreibungen genutzt. Das bedeutet, dass alle Informationen ab dem ersten Bild zum ersten Knoten gehören. Die Beschreibungen zwischen dem zweiten und dem dritten Bild werden dem zweiten Knoten zugewiesen, usw. Sind keine Bilder vorhanden, wird jedes HTML-Element (im Sinne des HTML Parsers) als ein Knoten angesehen. Die HTML-Seiten enthalten allerdings auch Querverweise zu anderen HTML-Seiten, oder Links, die in der Hierarchie wieder hinauf führen (der bekannte Link ‘zurück’). Würde auch diesen Links nach oben beschriebenen Verfahren gefolgt werden, entartet der Baum zu einem zyklischen Graphen. Darum müssen solche Verweise ignoriert werden. Der Algorithmus folgt deshalb nur solchen Links, die zu direkten Nachfahren des Knotens in der Hierarchie führen. Die Nachfahren können anhand ihres Knoten-Codes erkannt werden, da sie den Pfad des Vorfahren als Suffix enthalten.

Die Feature Base enthält auch Knoten, denen keine Nummer (ν), sondern das Symbol ‘_’ zugeordnet wird. Dabei handelt es sich um Werte-Knoten, die nicht auswählbar sind und keine Geschwister haben (vgl. Abschnitt 5.2.2). Sie müssen darum vom Nutzer nicht explizit ausgewählt werden, sondern es kann an

der Stelle gleich bei Ihrem Kindknoten mit der Auswahl fortgefahren werden. Deshalb existiert für solche Knoten keine eigene HTML-Datei. Wird so ein Knoten angetroffen, dann wird ein Eintrag für ihn in der Datenbank erzeugt, und mit seinem Kindknoten fortgefahren.

Das Programm protokolliert zusätzlich die bereits besuchten Dateien, die noch nicht besuchten Dateien und die Zahl der gefundenen Knoten. Ein zweites Tool namens 'importImages' liest alle Bilder des HTML-Verzeichnisses und seiner Unterverzeichnisse und fügt sie in die Datenbanktabelle 'PICTOGRAM' ein. Der Dateiname der Bilder entspricht jeweils dem Code des zugehörigen Knotens.

Import der Distanzmatrizen

Im Kapitel 6.2.2 wurde die Umsetzung der Distanzen ausführlich diskutiert. Es sollen für alle Wertebereiche Distanzmatrizen erstellt werden und wie in Abschnitt 7.2.2 beschrieben in die Datenbank eingefügt werden. Soweit Distanzmatrizen nötig sind, liegen diese in Excel-Dateien vor, die von den Musikwissenschaftlern erstellt wurden. Das Programm unterstützt fünf Möglichkeiten der Beschreibung von Distanzen:

1. Boolesche Funktion
2. Wertanalyse
3. Wertanalyse mit ebenenweisen Distanzmatrizen
4. eine Distanzmatrix
5. mehrere Distanzmatrizen

Der erste Fall entspricht der aus Abschnitt 6.2.2 als 'Einfacher Wertevergleich' bekannten Gruppe von Features. Sie müssen nicht in die Datenbank eingefügt werden, da ein einfacher Wertevergleich ausreicht, um zu bestimmen, ob die Distanz eins oder null ist.

Die nächsten beiden Fälle fallen in die zweite Gruppe 'Wertevergleich mit Wertanalyse'. Es existieren einerseits Feature mit kleinen Wertebereichen, bei denen einfache Gesetzmäßigkeiten für die Ähnlichkeit der Feature gelten. Bei Merkmal '3.1.1.', der Kaudierung des Notenhalses bei der aufwärts ausgerichteten Viertelnote, gilt die Regel, dass benachbarte Knoten die Distanz 0,7 haben und alle anderen die Distanz eins. Solche einfachen Regeln werden im Import-Tool hart-kodiert. Andererseits existieren auch Feature, bei denen die Regeln zur Beschreibung der Ähnlichkeiten zusätzlich Distanzmatrizen für jede Ebene der Wertebereichshierarchie benötigen. Das Prinzip wurde bereits in Abschnitt 6.2.2 am Beispiel der Viertelpause (Abbildung 6.3) erläutert. Das Programm liest die ebenenweise formulierten Distanzmatrizen aus den Excel-Dateien ein und wendet die den Merkmalen entsprechenden Regeln an. Da für jedes Feature eine eigene Regel existiert, wurde auch für jedes Merkmal eine separate Java-Methode erstellt.

Der vierte und fünfte Fall in der oben genannten Liste entsprechen der dritten Gruppe aus Abschnitt 6.2.2: dem ‘Wertevergleich mit Wertanalyse und zusätzlichen Informationen’. Es handelt sich dabei um die Fälle, wo eine reine Distanzmatrix vorliegt, die aus einer Excel-Datei eingelesen und in die Datenbank eingefügt wird (vierter Fall). Da Excel nur 255 Spalten erlaubt, liegen für einige Feature mehrere Dateien vor, die intern erst zu einer Distanzmatrix zusammengesetzt werden müssen (fünfter Fall).

Import einer Testmenge von Feature-Vektoren

Für den Test des Prototypen liegt eine Menge von Feature-Vektoren vor. Diese wurde von den Musikwissenschaftlern in einer Excel-Datei festgehalten. In dieser Datei steht jeweils ein Feature-Vektor mit seinen Werten in einer Zeile. Komplexe Feature-Werte werden mit Hilfe eines Schrägstriches dargestellt. Hat beispielsweise das Merkmal ‘2.3.2. Achtel abwärts’ den Eintrag ‘2.3.2.1. / 2.3.2.2. / 2.3.2.3.’ heißt das, dass die drei genannten Werte gleichberechtigt auftreten.

Das Kommandozeilen-Tool ‘importFeatureVectors’ aus dem Java-Package ‘de.enotehistory.import’ liest die Excel-Datei ein, erkennt komplexe Werte und fügt die Daten in die Datenbanktabellen ‘FEATUREVECTORS’ und ‘FVVALUES’ ein.

Nachdem die Programme beschrieben wurden, welche die benötigten Daten in die Datenbank einfügen, soll im Folgenden die eigentliche Klassifikationskomponente erläutert werden.

7.3.2 Klassifikationskomponente

Der Prototyp implementiert die beiden in Abschnitt 7.2.1 vorgestellten Stored Procedures. Dazu werden zunächst einige hilfreiche Datentypen eingeführt. Abbildung 7.1 zeigt das Package ‘de.enotehistory.datatypes’. Dort werden drei wichtige Interfaces und deren Implementierungen definiert.

Interfaces

Das erste Interface namens ‘Feature’ erlaubt die Beschreibung eines Merkmals. Für den Prototypen werden der Feature-Code, dessen Beschreibung, seine Position in der Prioritätenliste und das Gewicht in der Distanzfunktion gespeichert. Das Interface ‘FeatureVector’ bietet die Funktionalität zur Verwaltung eines Feature-Vektors. Es existieren Methoden zur Speicherung und Rückgabe des zugehörigen Schreibers, von Bewertungsmaßen, die in Kapitel 6.3 vorgestellt wurden und der eigentlichen Feature-Werte. Die wichtigste Methode ist

```
double distance(FeatureVector otherVector)
```

Sie berechnet die Distanz zu einem anderen Feature-Vektor. Es existiert ebenfalls eine Methode, welche die Werte des Feature-Vektors als Array von Objekten des Typs ‘FeatureValue’ zurückgibt. ‘FeatureValue’ ist eine Interface, das genau einen Merkmalswert verwaltet. Es werden drei Methoden definiert:

Interface Summary	
<u>Feature</u>	The Feature interface stores and handles information about a feature.
<u>FeatureValue</u>	The FeatureValue interface provides methodes for storing and working with values of features.
<u>FeatureVector</u>	The FeatureVector interface provides methodes for storing and working with feature vectors.
<u>TreeNode</u>	The TreeNode interface combines the characteristics of a FeatureBaseNode and a FeatureValue.
Class Summary	
<u>DBFeatureValue</u>	DBFeatureValue is a ddatabase based implementation of the FeatureValue interface.
<u>MemFeature</u>	MemFeature is an implementation of the Feature interface that loads the feature data once and keeps it in the memory.
<u>MemFeatureVector</u>	MemFeatureVector is an implementation of the FeatureVector interface that loads the feature data once and keeps it in the memory.

Abbildung 7.1: Das ‘datatypes’-Package in Java

Feature `getFeature()` gibt das Feature zurück, zu dem dieser Wert gehört.

`java.lang.String[]` `getValue()` liefert den eigentlichen Wert des Merkmals.

In den meisten Fällen wird das Array die Größe eins haben. In den Fällen, in denen echte komplexe Feature-Werte auftreten, entspricht die Array-Größe der Anzahl der Werte des Merkmals.

double `distance(FeatureValue otherValue)` berechnet die Distanz zweier Werte. Diese Methode entspricht der Feature-Distanzfunktion aus den Abschnitten 5.2.4 und 6.2.2.

Implementierungen

Das Package ‘de.enotehistory.datatypes’ enthält auch Implementierungen der einzelnen Interfaces. Eine Implementierung von ‘Feature’ ist die Klasse ‘MemFeature’. Sie holt sich die Informationen über das Merkmal einmalig aus der Datenbank, legt sie im Hauptspeicher ab und greift dann auf diese Daten zu. Eine Alternative dazu wäre es zum Beispiel, die Daten nicht vorher zu laden, sondern jeweils bei Bedarf mittels einer Datenbankanfrage umzusetzen.

Analog funktioniert die Klasse ‘MemFeatureValue’, die das Interface ‘FeatureValue’ implementiert. Sie lädt die Distanzmatrix einmalig in ein Array und liest die Distanzen dann nur noch dort ab. Im Gegensatz dazu stellt die Klasse ‘DBFeatureValue’ bei jedem Aufruf von ‘distance(otherValue)’ an die Datenbank Anfragen der Form:

```
SELECT DISTANCE FROM DISTANCES WHERE VALUE1=x AND VALUE2=y,
```

wobei x, y zwei Feature-Werte sind. Die Tabelle ‘DISTANCES’ ist bereits aus Abschnitt 7.2.2 bekannt. Bevor die Methode auf die Datenbank zugreift, testet sie zusätzlich, ob die beiden Werte identisch sind. Wenn dies der Fall ist,

Zeile	Java-Anweisungen
1	public double distance(FeatureVector othVector)
2	{
3	FeatureValue[] v1 = this.getVector();
4	FeatureValue[] v2 = othVector.getVector();
5	if (v1 == null v2 == null) return ERROR;
6	if (v1.length != v2.length) return ERROR;
7	
8	double sum = 0.0d;
9	double weights = 0.0d;
10	
11	for (int i = 0 ; i < v1.length ; i++)
12	{
13	if (v1[i] == null v2[i] == null) continue;
14	double weight = v1[i].getFeature().getWeight();
15	double dis = v1[i].distance(v2[i]);
16	if (dis < -0.1) continue;
17	sum += weight * dis;
18	weights += weight;
19	}
20	if (weights < 0.01) return 0.0d;
21	return sum / weights;
22	}

Tabelle 7.4: Die Umsetzung der Distanzfunktion in Java

ist die Distanz null und muss nicht extra aus der Datenbank gelesen werden. Damit ist es auch nicht zwingend nötig, dass solche Distanzen in der Tabelle ‘DISTANCES’ stehen.

Die eigentliche Distanzfunktion wird in Klassen umgesetzt, die das Interface ‘FeatureVector’ implementieren, wie zum Beispiel ‘MemFeatureVector’. Die Methode ‘distance’ in Abbildung 7.4 setzt die Hamming-Distanz um. Sie berechnet die Distanz zwischen zwei Feature-Vektoren (‘this’ und ‘othVector’). Zuerst holt sich die Methode die beiden Feature-Wert-Arrays v1 und v2 und testet diese auf Korrektheit (Zeile drei bis sechs). Auf jedes Wertepaar wird die Feature-Distanzfunktion angewandt (Zeile 15). Außerdem wird in Zeile 14 das Gewicht des aktuellen Features bestimmt. Stellt die Feature-Distanzfunktion fest, dass einer der beiden Werte ein Nullwert ist, so gibt sie den Wert -1.0 zurück. In diesem Fall kann das Merkmal ignoriert werden (Zeile 16). In den Zeilen 17 und 18 wird die Summe über den Distanzen und Gewichten gebildet, wie sie für die Hamming-Distanz-Funktion aus Abschnitt 6.2.3 benötigt werden. Der Quotient dieser beiden Werte wird in Zeile 21 berechnet und zurückgegeben. Für die Tests im nächsten Kapitel wurde die ‘distance’-Methode auch mit der Euklidischen-Distanz umgesetzt. Eine der beiden Varianten wurde für die Tests jeweils auskommentiert. In Abbildung 7.4 wird deutlich, dass in der Implementierung entsprechend der formalen Herleitung Fließkommazahlen für die Distanzen und Gewichte genutzt werden. Für einen Prototypen ist dies ausreichend. Im weiteren Optimierungsprozess des Schreibererkennungssystem sollte dafür allerdings der Integer-Datentyp genutzt werden, da Operationen auf diesem bedeutend schneller ablaufen. Die jetzigen Distanzen könnten beispielsweise auf den Bereich [0,100] abgebildet werden.

Die Stored Procedures können einfach auf die Distanz-Methode des Feature-Vektor-Interfaces zugreifen. Sie sind in Abbildung 7.5 in Pseudo-Code dargestellt. Der Aufbau von A.FV und A.S ist fast identisch. Darum wird das Vorgehen am Beispiel von A.S erklärt und gegebenenfalls auf die Unterschiede zu A.FV hingewiesen. Die Methoden müssen zuerst ein Feature-Vektor-Objekt (fv) für den Anfrage-Feature-Vektor initialisieren (Zeile 12). In Zeile 13 werden die Feature-Vektoren aus der Datenbank in ein Array geladen. Aus Performance-

Zeile	Anweisungen
0	public static void A_S (
1	String featureValue01,
2	.
3	.
4	String featureValue78,
5	int[] errorCode,
6	String[] errorMsg,
7	ResultSet[] rs)
8	{
9	FeatureVector fv;
10	FeatureVector[] database;
11	
12	Initialisiere fv mit featureValue01 bis featureValue78
13	Lade database aus der Datenbank
14	
15	Berechne die k nächsten Nachbarn von fv in database
16	Berechne aus den Nachbarn die Schreiber
17	
18	Gebe das Ergebnis mittels rs zurück
19	}

Tabelle 7.5: Die Stored Procedures in Pseudo-Code

Gründen kann dieser Schritt auch einmalig stattfinden und in eine andere Methode ausgelagert werden. Die Berechnung der k nächsten Nachbarn kann durch eine einfache Schleife realisiert werden. In dieser werden die Distanzen durch `fv.distance(database[i])` berechnet und die Feature-Vektoren, deren Distanz zu `fv` unter dem Schwellwert liegt, in eine geordnete Liste aufgenommen. Die Funktion `A_FV` ist damit bereits am Ende angelangt. `A_S` muss aus dieser Menge, wie in Abschnitt 6.2 beschrieben, noch die Menge der Schreiber berechnen (Zeile 16). Die beiden Stored Procedures legen dann eine temporäre Datenbanktabelle an, in die sie die Ergebnisse schreiben, öffnen ein `ResultSet` auf diese Tabelle und geben es an den Nutzer zurück.

Der Quell-Code des Packages 'de.enotehistory' ist zu umfangreich, um ihn in ausgedruckter Form beizulegen. Der interessierte Leser kann sich diesbezüglich an den Autor dieser Arbeit oder an ein Mitglied des eNoteHistory-Projektes über die Internetseite www.enotehistory.de wenden. Nachdem dieses Kapitel gezeigt hat, wie der Prototyp implementiert wird, beschreibt das nächste Kapitel, wie das System getestet wurde.

Kapitel 8

Test und Optimierung des Systems

Das Data-Mining-Modell und die initiale Systemkonfiguration beruhen auf dem Fachwissen von Musikwissenschaftlern. Bevor das Schreibererkennungssystem eingesetzt werden kann, müssen das Modell und seine Parameter mit realen Daten überprüft und gegebenenfalls optimiert werden. Für diesen Zweck liegt ein Datensatz mit circa 80 Schreibercharakteristiken von 50 verschiedenen Schreibern vor. Aus Data-Mining-Sicht ist das sehr wenig. Es muss darum besonders auf die Gefahr des Overfitting geachtet werden.

Das Modell soll auf folgende Punkte hin untersucht werden:

- Priorität der Features,
- Wahl der Distanzfunktion,
- Gewichte in der Distanzfunktion,
- Optimierung der Distanzmatrizen
- Bewertung der Antwortmengen in Zusammenhang mit Nullwerten und
- Betrachtung von Features mit mehreren Werten

Die einzelnen Punkte werden in diesem Kapitel genau beschrieben. In den meisten Tests wird die Bewertung K (vgl. 6.3.3) angegeben, die in besonderem Maße von der Wahl des Schwellwertes t abhängt. Dieser Zusammenhang wird in Abschnitt 8.5 genauer untersucht. Bis dahin wird jeweils die beste Bewertung K angegeben, die mit der entsprechenden Konfiguration des Systems möglich ist. Das Gleiche gilt für die Precision und den Recall-Wert. Es werden jeweils die Werte angegeben, die in dem Szenario mit der besten Bewertung K erreicht werden.

8.1 Priorität der Features

Das Data-Mining-Modell basiert auf einer Liste von Features, in der diese nach ihrer Relevanz für die Schreiberhanderkennung geordnet sind. Die Ordnung

Priorität	Feature
über Durchschnitt	9.7.1., 1.3.2.3., 11.3., 6.2., 13.1., 5.1., 3.1.2., 9.5.2., 1.3.1., 9.5.1., 3.2.2.
unter Durchschnitt	9.3.1., 3.3.2., 5.3., 12., 9.1.1.2., 2.3.2., 2.3.1., 2.2.1., 2.2.2., 7.2., 6.5., 11.4., 1.1.1.

Tabelle 8.1: Zusammenfassung der Testergebnisse für Prioritäten der Attribute

basiert auf musikwissenschaftlichem Fachwissen und kann in Anhang A.1 eingesehen werden. Ziel der Optimierungsphase ist es, diese Prioritäten mit DM-Techniken zu überprüfen und zu verbessern. Das geschieht über zwei Ansätze, die bereits in Abschnitt 2.2.4 vorgestellt wurde. Im ersten Ansatz werden alle Untermengen der Feature-Menge (F) betrachtet, die genau ein Element weniger als F haben und die auf den entsprechenden Distanzfunktionen basierenden Ergebnisse bewertet. Der zweite Ansatz besteht darin, die Bedeutung der Attribute aus den Ausgaben induktiver Klassifikationsverfahren abzuleiten. Dieser Ansatz wird später in diesem Abschnitt vorgestellt, denn zunächst soll der erste genauer beschrieben werden.

Bisher war die Distanzfunktion die Abbildung:

$$d_{\Gamma} : \Gamma \times \Gamma \rightarrow [0..1]$$

Für diesen Test werden die folgenden Distanzfunktionen genutzt:

$$d_{\Gamma/v_i} : \Gamma/v_i \times \Gamma/v_i \rightarrow [0..1], \text{ wobei}$$

$$\Gamma/v_i = \{\gamma/v_i \mid \gamma/v_i = (v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n)^T, \forall j : v_j \in W_j\}.$$

Für alle d_{Γ/v_i} mit $i \in \{1, \dots, |F|\}$ wird die Bewertung SF berechnet. Ein Feature, durch dessen Fehlen in der Berechnung sich die Bewertung verschlechtert, wird als wichtiger erachtet, als ein solches, bei dem sich die Bewertung verbessert. Dadurch kann eine Priorität für jedes Feature bestimmt werden, die angibt, wie wichtig das Feature in der Berechnung für die Testmenge ist. Genau da liegt aber auch ein Problem. Die Testmenge enthält nur sehr wenige Feature-Vektoren, so dass es leicht zu Overfitting kommen kann. Eine Schlussfolgerung kann deshalb nicht sein, dass ein Attribut überflüssig in der Berechnung der Distanzen ist, sondern nur, dass es einen geringeren Einfluss auf die Berechnungen haben sollte. Darüber hinausgehende Entscheidungen können erst mit einer viel größeren Testmenge getroffen werden. Aus diesem Grund werden auch nur $(n-1)$ -elementige Teilmengen betrachtet. Die Information über die Bewertung beim Weglassen einer Kombination von Features ist zwar interessant, kann aber nur zur Schlussfolgerung führen, die beteiligten Features ganz aus der Berechnung heraus zu lassen. Da dies nicht geschehen soll, ist die Information an dieser Stelle nicht von Interesse.

Die Ergebnisse des Tests sind in Anhang A.2 zu finden. Die Bewertung des Modells, in dem alle Features vorhanden sind, ist zum Vergleich auch in der

Tabelle enthalten und fett gedruckt. Die Features sind nach ihrer Bewertung sortiert. Das Feature, durch dessen Fehlen die schlechteste Bewertung erreicht wird und das damit als wichtigstes für die Testmenge angesehen wird, steht an erster Stelle. Die Features, deren Bewertungen mehr als ein Prozent von der Ausgangsbewertung nach oben oder unten abweichen, werden durch jeweils eine Leerzeile von den nicht so stark abweichenden getrennt. Sie werden in Tabelle 8.1 zusammengefasst.

Die meisten der Features mit einer geringen Priorität sind auch in dieser Kategorie zu erwarten gewesen. Sie haben Wertebereiche, die nur zwei bis fünf Werte umfassen und sind damit nicht besonders gut geeignet, die Klasse vorauszusagen. Bei dem Attribut 11.4. (Format) kann schon mit einem Blick auf die Testmenge erkannt werden, dass es am Ende der Prioritätenliste stehen würde. Die Wahl zwischen Hoch- und Querformat lässt keinen Rückschluss auf den Schreiber zu und erscheint eher zufällig belegt. Interessant ist allerdings, dass das Feature 1.1.1. (G-Schlüssel, einzügiges Grundelement) in der auf dem Test basierenden Liste an letzter Stelle steht. Zumal es aus musikwissenschaftlicher Sicht eine sehr hohe Priorität hat. Die Ergebnisse zeigen, dass das Modell ohne das Feature 1.1.1. minimale Werte sowohl für die Distanzen zwischen Instanzen einer Klasse (SF-Nenner), als auch verschiedener Klassen (SF-Zähler) hat. Daraus folgt, dass Attribut 1.1.1. die Distanzen im Durchschnitt erhöht. Ein Blick auf die Testmenge legt folgende Begründung nahe. Der G-Schlüssel hat von allen Merkmalen den größten und komplexesten Wertebereich. Viele Werte unterscheiden sich nur durch eine Kleinigkeit, so dass es der Regelfall ist, dass in der Testmenge verschiedene Feature-Vektoren eines Schreibers verschiedene Werte für den G-Schlüssel haben, die aber eine große Ähnlichkeit haben. Diese Distanz beträgt aber eben nicht Null, wie bei vielen anderen Merkmalen, bei denen die manuelle Klassifizierung, auf Grund der kleineren Wertebereiche, eine bessere Vorarbeit leistet. Es ist allerdings zu erwarten, dass die Bedeutung des G-Schlüssels mit steigender Anzahl der Feature-Vektoren in der Datenbank zunimmt.

In der Liste der Features mit einer hohen Priorität fallen einige Features auf, die dort nicht zu erwarten waren: 9.7.1., 11.3. und 6.2. Sie haben alle sehr kleine Wertebereiche und sollten deshalb nicht gut für die Vorhersage des Schreibers geeignet sein. In Anhang A.2 wird deutlich, worauf ihre gute Bewertung basiert. Sie tragen dazu bei, dass die durchschnittlichen Distanzen innerhalb der Cluster (SF-Nenner) kleiner werden, während der SF-Zähler nahezu unverändert bleibt. Das liegt daran, dass in der Testmenge Instanzen eines Schreibers auch immer genau den gleichen Wert für diese Attribute haben.

Tests mit dem Weka-Tool (vgl. 3.2 und 7.1), in denen induktive Klassifikationsverfahren auf die Testmenge angewendet werden, bestätigen die genannten Ergebnisse. Das 1-Rule-Verfahren (vgl. Abschnitt 2.2.2) bestimmt das Feature '13.1. Viertelpause' als jenes mit dem kleinsten totalen Fehler (47/89). Es wurde bereits in Kapitel 2.1.1 erläutert, dass die Viertelpause eine große Individualität in Bezug auf die grafische Gestalt zulässt und damit für die Klassifikation gut geeignet ist. Darüber hinaus wurde ein Entscheidungsbaum basierend auf der Testmenge berechnet. In einem Entscheidungsbaum stehen solche Attribute nahe der Wurzel, die am besten für die Klassifikation geeignet sind (vgl. Abschnitt

Distanz- funktion	Fälle mit besserem Score	SF- Durchschnitt	Fälle mit besserem K	K- Durchschnitt
Hamming	36	0,261	34	0,836
Euklidische	0	0,439	2	0,784

Tabelle 8.2: Zusammenfassung der Testergebnisse für die Distanzfunktion

2.2.2). Im vorliegenden Fall sind das die Feature ‘13.1. Viertelpause’, ‘5.2. Verlauf der Bebalkung’ und ‘1.3.1. Hauptelement des C-Schlüssels’. Es handelt sich bei allen Attributen um solche, die auch in den im vorherigen Absatz beschriebenen Tests eine große Bedeutung hatten und von denen dies im Vorhinein erwartet wurde beziehungsweise mit der kleinen Testmenge begründbar ist.

In diesem Abschnitt ist deutlich geworden wie groß die Gefahr des Overfittings ist. In der Tabelle in Anhang A.3, auf die im folgenden Abschnitt näher eingegangen wird, ist ersichtlich, dass ein Modell, dass auf den berechneten Prioritäten basiert, eine viel bessere Bewertung bekommt. Sie beträgt im Schnitt 3,09 im Vergleich zu 2,69 - der Bewertung des musikwissenschaftlichen Modells. Im Prototyp soll eine Mischung aus musikwissenschaftlichen und den aus dem Tests resultierenden Prioritäten eingesetzt werden, indem jedes Attribut die über allen Listen gemittelte Priorität bekommt. In den weiteren Tests wird jeweils untersucht werden, wie die Bewertungen basierend auf den musikwissenschaftlichen Prioritäten und den Testprioritäten aussehen.

8.2 Wahl der Distanzfunktion

Die Wahl der richtigen Distanzfunktion spielt eine entscheidende Rolle. Die Möglichkeiten für zwei Feature-Vektoren $\gamma_a = (v_1^a, \dots, v_n^a)^T$ und $\gamma_b = (v_1^b, \dots, v_n^b)^T$ sind:

$$d_\Gamma = \frac{w_1 \cdot d_{f_1} + \dots + w_n \cdot d_{f_n}}{\sum_{i=1}^n w_i} \text{ (normierte, gewichtete Hamming-Distanz)}$$

$$d_\Gamma = \sqrt{\frac{(w_1 \cdot d_{f_1})^2 + \dots + (w_n \cdot d_{f_n})^2}{\sum_{i=1}^n w_i^2}} \text{ (normierte, gewichtete Euklidische Distanz)}$$

$$d_\Gamma = \sqrt[k]{\frac{(w_1 \cdot d_{f_1})^k + \dots + (w_n \cdot d_{f_n})^k}{\sum_{i=1}^n w_i^k}} \text{ (normierte, gewichtete Distanz mit beliebigem Exponenten)}$$

Im Großen und Ganzen unterscheiden sich diese Varianten nur in der Höhe der Potenz der einzelnen Summanden. Hohe Potenzen erhöhen den Einfluss von großen Differenzen (vgl. [57]). Abweichungen bei einzelnen Merkmalen fallen also stärker ins Gewicht. Es ist darum zu erwarten, dass mit steigender Potenz gerade der Nenner von SF ansteigt, der die durchschnittliche Distanz von Feature-Vektoren des gleichen Schreibers angibt.

Die Vermutung wird durch die Testergebnisse bestätigt, die in Anhang A.3 ausführlich dargestellt sind. Tabelle 8.2 fasst die Ergebnisse zusammen. Die beiden Distanzfunktionen wurden in den verschiedensten Konfigurationen ge-

testet. Die Parameter, die dafür zusätzlich variiert wurden, sind in Anhang A.3 der Vollständigkeit halber aufgeführt, werden aber erst in den folgenden Abschnitten eingeführt und genauer beschrieben. Es zeigt sich deutlich, dass die Hamming-Distanz in allen Fällen zu einer besseren Bewertung SF führt als die Euklidische. Die Partitionen, die auf der Hamming-Distanz basieren, haben im Schnitt eine um 0,178 Punkte bessere Bewertung als solche, die mit Hilfe der Euklidischen Distanz bestimmt wurden. Außerdem fällt auch die Beurteilung aufgrund der Bewertung K eindeutig für die Hamming-Distanz aus. Aus diesem Grund wird in dem zu entwickelnden Schreibererkennungssystem die Hamming-Distanz eingesetzt und alle folgenden Tests werden sich nur noch auf diese beziehen, denn es wurde auch schon gezeigt, dass die Euklidische Distanz bei keiner Konfiguration zu besseren Werten führt. Deshalb wird darauf verzichtet Distanzfunktionen mit noch höheren Potenzen zu testen, da sich die beschriebenen Nachteile nur noch verstärken würden.

8.3 Gewichte in der Distanzfunktion

Im Abschnitt über die Prioritäten der Features wurde bereits eine Liste aufgestellt, in der die Schreibermerkmale nach ihrer Bedeutung für die Schreibererkennung geordnet sind. Es stellt sich die Frage, wie diese Prioritäten als Gewichte der Distanzfunktion umgesetzt werden können. Es geht also darum, die Gewichte w_1 bis w_n in folgender Formel zu bestimmen:

$$d_{\Gamma} = \frac{w_1 \cdot d_{f_1} + \dots + w_n \cdot d_{f_n}}{\sum_{i=1}^n w_i}$$

Dazu sollen folgende Fälle untersucht werden:

Konstant $\forall i \in \{1..n\} : w_i = 1$

Alle Features haben das gleiche Gewicht. Prioritäten werden so noch nicht umgesetzt.

Linear Gruppier $\forall i \in \{1..n\} : w_i = \lceil i/26 \rceil$

Die Merkmale wurden ursprünglich aus musikwissenschaftlichen Gründen in drei Gruppen eingeteilt. Diese Gewichtung trägt dem Rechnung. Die ersten 26 Feature bekommen alle das Gewicht '1', die folgenden 26 Merkmale das Gewicht '2' und die letzten 26 das Gewicht '3'.

Linear $\forall i \in \{1..n\} : w_i = i$

Die drei Gruppen sind aufgehoben. Alle Merkmale haben unterschiedliche Gewichte, die sich linear erhöhen.

Quadratisch Gruppier $\forall i \in \{1..n\} : w_i = \lceil i/26 \rceil^2$

Diese Gewichtung ist analog zu der linear gruppierten. An dieser Stelle werden die einzelnen Gewichte aber zusätzlich quadriert.

Quadratisch $\forall i \in \{1..n\} : w_i = i^2$

Entsprechend gibt es auch eine quadratische Gewichtung, welche die Gruppierung aufhebt und den Merkmalen Gewichte zuordnet die quadratisch ansteigen.

Distanzfunktion	SF-Durchschnitt	K-Durchschnitt
konstant	3,266	0,852
linear gruppiert	3,942	0,856
linear	3,963	0,856
quadratisch gruppiert	4,600	0,824
quadratisch	5,226	0,807

Tabelle 8.3: Zusammenfassung der Testergebnisse für die Gewichte in der Distanzfunktion

In den genannten Formeln steht das unwichtigste Feature mit dem kleinsten Gewicht jeweils an erster Stelle ($i=1$). Dies dient der Vereinfachung der Formeln. Um konsequent in der Darstellung zu sein, dass die wichtigsten Attribute weiter vorne stehen, müsste das i auf den rechten Seiten der Formeln jeweils durch $(79 - i)$ ersetzt werden.

Im vorherigen Abschnitt wurde erklärt, dass unwichtige Attribute nicht ganz vernachlässigt werden können, sondern nur ans Ende der Prioritätenliste verschoben werden, um mit einem kleinen Gewicht versehen zu werden. Aus diesem Grund ist zu erwarten, dass eine quadratische Gewichtung der Features die besten Ergebnisse erzielt, da bei dieser der Einfluss der wichtigen Merkmale gegenüber den unwichtigen am größten ist.

In Tabelle 8.3 sind die ausführlichen Testergebnisse aus Anhang A.4 zusammengefasst. Es zeigt sich, dass die linearen Varianten in allen Fällen, unabhängig von der restlichen Konfiguration des Systems, mit der SF-Bewertung besser abschneiden als die konstante Gewichtung. Mit den quadratischen Gewichten wird aber eine noch bessere SF-Bewertung erreicht und damit eine bessere Partition der Instanzenmenge in Klassen. Die Unterscheidung zwischen gruppierten und nicht gruppierten Gewichten macht keinen großen Unterschied in der Bewertung aus. Ein anderes Bild ergibt sich bei der Betrachtung der K-Bewertung. Sie stuft die linearen Varianten besser ein als die quadratischen. Wie kommt es dazu? Das Modell mit quadratischen Gewichten erreicht extrem gute Distanzwerte für solche Feature-Vektoren-Vergleiche, bei denen die Attribute mit hoher Priorität ähnlich sind. Es hat sich gezeigt, dass für zwei Schreibercharakteristiken γ_1 und γ_2 des selben Schreibers $d_\Gamma(\gamma_1, \gamma_2) < 10^{-5}$ werden kann. Es gibt in der Testmenge aber auch vereinzelte Fälle, bei denen sich zwei Instanzen eines Schreibers in genau solchen Attributen unterscheiden, die eine hohe Priorität haben, wodurch d_Γ übermäßig groß ausfällt. Auf die Bewertung SF wirkt sich so ein Ausreißer nicht so stark aus, da seine große Distanz durch die extrem kleinen Distanzen der anderen Instanzen seiner Klasse kompensiert wird. Auf die Bewertung K wirkt sich dieser Ausreißer allerdings negativ aus. Für K ist es wichtig, dass alle Distanzen der Feature-Vektoren eines Schreibers unter einem bestimmten Schwellwert liegen. Darum wird hier ein Modell bevorzugt, bei dem die Werte im Durchschnitt zwar nicht extrem klein sind, aber die maximale Anzahl unter diesem Schwellwert liegt. So erreichen die lineare und sogar die konstante Gewichtung bessere K-Bewertungen, denn hier wirkt sich ein irreführendes Attribute mit hoher Priorität nicht so negativ aus, da es kein extrem hohes Gewicht hat.

Art	Neuer Wert
Translation	Alter Wert - 0,2
	Alter Wert - 0,1
	Alter Wert + 0,1
	Alter Wert + 0,2
Skalierung	Alter Wert / 4
	Alter Wert / 2
	Alter Wert * 2
Quadratisch / Kubisch	$(AlterWert)^3$
	$(AlterWert)^2$
	$\sqrt{AlterWert}$
	$\sqrt[3]{AlterWert}$

Tabelle 8.4: Testszenario für die Ähnlichkeitswerte

Die Bewertung der verschieden gewichteten Attribute und damit die Wahl einer der vorgestellten Varianten, hängt von der Testmenge und insbesondere ihrer Größe ab. Es ist folgendes zu erwarten: Je mehr Feature sich in der Menge und damit im n-dimensionalen Feature-Raum befinden, desto größer wird die Bedeutung von besonders kleinen Klassen sein. Somit sollten sich die Bewertungen von SF und K mit wachsenden Datenmengen angleichen. Ob sich diese Vermutung bestätigt, kann nur in Tests mit bedeutend mehr Schreibercharakteristiken überprüft werden. Im Prototypen wird die lineare Gewichtung als Kompromiss zwischen SF- und K-Bewertung eingesetzt.

8.4 Optimierung der Distanzmatrizen

Alle Ähnlichkeitswerte, die nicht Null oder Eins sind, wurden manuell festgelegt. Das soll in diesem Fall auch solche Ähnlichkeiten einschließen, für die eine Berechnungsvorschrift vorliegt, denn auch diese wurde von Menschen und damit nicht automatisch erstellt. Bei diesen Werten handelt sich um Fachwissen, dass nicht durch induktive DM-Verfahren anhand von Trainingsmengen gewonnen werden kann (vgl. 5.2.5, 6.2.1 und 7.2.2). Anlass für das Überprüfen dieser Werte ist die Vermutung, dass es für Menschen nicht trivial ist, Ähnlichkeiten zwischen zwei Objekten eindeutig mit einem Wert zwischen Null und Eins zu belegen. Die Werte für ähnliche Symbole liegen zwischen 0,1 und 0,95. Gerade 0,95 ist dem Maximalwert schon so nahe, dass der Verdacht besteht, die Werte müssten etwas tiefer ausfallen. Darum sollen die Ähnlichkeitswerte wie in Tabelle 8.4 variiert werden, um zu sehen, wie sich das auf die Bewertung SF auswirkt. Sollten die Werte durch die Berechnung über die Intervallgrenzen von [0..1] hinaus gehen, werden sie auf den jeweiligen Randwert gesetzt.

Die ausführlichen Testergebnisse sind in Anhang A.5 aufgeführt und in Tabelle 8.5 zusammengefasst. In der detaillierten Auflistung wird deutlich, dass sich die Bewertung nur leicht verbessert oder verschlechtert, wenn die Distanzwerte variiert werden. Die Vermutung hat sich allerdings bestätigt, dass die Werte im allgemeinen etwas zu hoch sind. Tabelle 8.5 zeigt, dass genau die

Bewertung	Veränderung des Distanzmaßes
Verbesserung	Translation (-0,2), Translation (-0,1) Skalierung (/4), Skalierung (/2) Kubisch, Quadratisch
Verschlechterung	Translation (+0,1), Translation (+0,2) Skalierung (*2) Wurzel, Dritte Wurzel

Tabelle 8.5: Zusammenfassung der Testergebnisse für die Distanzwerte

Veränderungen zu einer leicht besseren Bewertung führen, die die Distanzwerte herabsetzen. Das Erhöhen der Werte führt durchgängig zu einer schlechteren Bewertung. Insgesamt hat dieser Test gezeigt, dass die ursprünglichen Distanzwerte gut gewählt sind und so beibehalten werden können.

8.5 Bewertung der Antwortmengen in Zusammenhang mit Nullwerten

Die Bewertungen der beiden Antwortmengen A_{FV} und A_S hängt im hohen Maße von der Wahl des Schwellwertes t ab. Im Anhang A.6 sind Precision und Recall für A_{FV} und die Bewertung K für A_S bei der Variation des Schwellwertes angegeben. Die anderen Modellparameter sind die in den letzten Abschnitten genannten.

8.5.1 Precision und Recall

Die Tabelle in Anhang A.6 zeigt zuerst einen trivialen Zusammenhang. Mit fallendem Schwellwert fällt auch der Recall-Wert, während der Precision-Wert steigt. Im Falle der besten Bewertung K erreichen Precision und Recall Werte von circa 85 bzw. 70 Prozent. In Abschnitt 6.3.3 wurde bereits erläutert, dass K Kombinationen bevorzugt, die eine höhere Precision haben. In der Tabelle wird auch deutlich, dass K kein globales, sondern mehrere lokale Maxima hat. In den Tests hat sich gezeigt, dass es entweder ein oder zwei nah beieinander liegende Maxima gibt. In diesem Fall wird der Schwellwert gewählt, bei dem die Werte von Precision und Recall besser sind, was aber auch nicht immer eindeutig ist. Insgesamt ist K im Vergleich zu den anderen Bewertungen eher kleinen Schwankungen unterlegen. Die Werte liegen zwischen 0,77 und 0,88. Wie in Abschnitt 8.3 bereits begründet wurde, ist bei einer größeren Testmenge auch mit größeren Schwankungen der Bewertung K zu rechnen.

8.5.2 Nullwerte

In allen Tests, in denen K bestimmt wurde, wurde auch N_D bestimmt und zwar jeweils zweifach. Als erstes für alle positiven Fälle aus K , in denen das System die Klassenzugehörigkeit der Instanz korrekt bestimmt hat. Der Wert in der Tabelle (A.6) ist der Mittelwert all dieser N_D . Analog wird der Anteil

der Nullwerte für die Fälle bestimmt, in denen eine Klassenzugehörigkeit nicht richtig erkannt wurde. Der Grund für dieses Vorgehen ist die Vermutung, dass gerade irreführende Distanzen zwischen zwei Feature-Vektoren dadurch entstehen, dass die beiden Feature-Vektoren viele Nullwerte enthalten und nur wenige Features zur Berechnung der Distanz herangezogen werden können.

Dieser Verdacht wurde in sämtlichen Tests ausnahmslos bestätigt. Die Tabelle in Anhang A.6 veranschaulicht das. Das Maß N_D hat bei korrekt klassifizierten Schreibern den Durchschnittswert 0,405 und bei falsch bestimmen 0,341. Das bedeutet, dass im ersten Fall über 40 Prozent der Features verglichen werden konnten und im zweiten nur 34 Prozent. Bei anderen Konfigurationen fällt dieser Unterschied noch größer aus. Die Schlussfolgerung daraus kann nur sein, für jedes Element der Antwortmengen anzugeben, auf welchem Wert für N_D es beruht. Je größer N_D ist, desto verlässlicher ist auch die berechnete Distanz. Die Angabe von N_D kann explizit geschehen, oder indem N_D in ein Bewertungsmaß einfließt, zum Beispiel in eine Ranking-Funktion.

8.6 Features mit mehreren Werten

In Kapitel 5.2.6 wurden Merkmale mit komplexen Werten bereits theoretisch eingeführt. Es wurde erläutert, dass die Distanz zwischen zwei Attributen zwischen dem Minimum und dem Durchschnittswert der einzelnen Distanzen liegen soll:

$$\min_{v^1 \in V_1, v^2 \in V_2} (d_{f_i}(v^1, v^2)) \leq d_{f_i}^K(V_1, V_2) \leq \frac{\sum_{v^1 \in V_1, v^2 \in V_2} d_{f_i}(v^1, v^2)}{|V_1| \times |V_2|}$$

Wo genau die Distanz in diesem Intervall liegen soll, blieb bisher offen und soll in einem Test mit dem bereits definierten Modell überprüft werden. Dabei werden folgende Formeln zur Berechnung von $d_{f_i}^K(\gamma_1^K, \gamma_2^K)$ eingesetzt:

1) **Minimum** $\min_{v^1 \in V_1, v^2 \in V_2} (d_{f_i}(v^1, v^2))$

Das Minimum ist die untere Grenze des möglichen Bereichs, in dem die Distanz liegen soll.

2) **Maximum** $\max_{v^1 \in V_1, v^2 \in V_2} (d_{f_i}(v^1, v^2))$

Das Maximum wurde in Abschnitt 5.2.6 ausgeschlossen, deshalb soll hier gezeigt werden, dass diese Überlegung richtig war.

3) **Arithmetisches Mittel** $\frac{\sum_{v^1 \in V_1, v^2 \in V_2} d_{f_i}(v^1, v^2)}{|V_1| \times |V_2|}$

Das Arithmetische Mittel ist die obere Grenze des möglichen Bereichs, in dem die Distanz liegen soll.

4) **Tendenz zum Maximum** $\sqrt[k]{\frac{\sum_{v^1 \in V_1, v^2 \in V_2} d_{f_i}(v^1, v^2)^k}{|V_1| \times |V_2|}}$

Durch das Potenzieren der einzelnen Distanzen bekommen höhere Distanzen ein größeres Gewicht in der Summe und der Gesamtwert tendiert mehr in Richtung des Maximums. Je höher die Potenz, desto stärker wirkt sich dieser Effekt aus. Im Test wird $k = 2$ und $k = 3$ getestet.

Formel	SF-Nenner	SF-Zähler	SF	K	PR	RE
1)	0,110	0,387	3,529	0,865	0,873	0,647
2)	0,211	0,488	2,317	0,820	0,853	0,679
3)	0,162	0,439	2,713	0,865	0,846	0,744
4) mit k=2	0,180	0,457	2,535	0,854	0,858	0,697
4) mit k=3	0,188	0,465	2,471	0,831	0,853	0,697
5) mit k=2	0,139	0,419	3,002	0,888	0,861	0,739
5) mit k=3	0,127	0,408	3,200	0,888	0,873	0,690

Tabelle 8.6: Zusammenfassung der Testergebnisse für Attribute mit komplexen Werten

5) Tendenz zum Minimum $\left(\frac{\sum_{v^1 \in V_1, v^2 \in V_2} \sqrt[k]{d_{f_i}(v^1, v^2)}}{|V_1| \times |V_2|}\right)^k$

Hierbei handelt es sich um einen Sonderfall der vorhergehenden Formel mit $0 > k > 1$. Dadurch wird allerdings der genau gegenteilige Effekt erreicht, die Gesamtdistanz nähert sich dem Minimum. Auch hier wird $k = 2$ und $k = 3$ getestet.

Tabelle 8.6 zeigt die Ergebnisse des Tests. Die Maximumvariante hat sich wie erwartet als die schlechteste erwiesen. Sowohl SF als auch K haben dort den schlechtesten Wert. Alle Varianten, die nahe am Minimum liegen, erhalten gute Bewertung von SF und K, wobei SF die Minimumvariante präferiert und K die Formel mit Tendenz zum Minimum. Da letztgenannte aber viel komplizierter zu berechnen ist und somit eine schlechte System-Performance bedingen würde, wird die erste Formel für das Schreibererkennungssystem gewählt.

Die Tests in diesem Kapitel haben gezeigt, dass der Prototyp in circa 90 Prozent aller Fälle den richtigen Schreiber bestimmen kann. Damit bewegt sich schon dieser erste Ansatz in der Größenordnung verwandter Systeme. Die beiden in Abschnitt 3.1 beschriebene Modelle erreichen eine Erfolgsquote von 94 bis 98 bzw. 93 Prozent. Dafür, dass diese Werte nicht ganz erreicht werden, gibt es zwei Ursachen. Einerseits ist der vorliegende Ansatz noch immer Änderungen im Modell unterlegen und wird so auch in Zukunft noch verbessert werden können. Andererseits muss das Schreibererkennungssystem im Gegensatz zu den anderen Systemen, nicht nur erkennen, um welchen Schreiber aus einer Menge von Schreibern es sich handelt. Darüber hinaus muss es zusätzlich bestimmen, ob der Kopist überhaupt schon in der Menge der Schreiber vorhanden ist. Dadurch ergibt sich automatisch eine höhere Fehlerwahrscheinlichkeit. Wie das Modell noch weiter verbessert werden kann wird im folgenden Kapitel in Abschnitt 9.2 aufgezeigt.

Kapitel 9

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein System entwickelt, das mit Hilfe des k-Nearest-Neighbor-Verfahrens Ähnlichkeiten zwischen verschiedenen Notenschreiberhänden erkennt und diese Funktionalität in eine Datenbank integriert. In diesem Kapitel soll zusammengefasst werden, wie die in Abschnitt 4.2 aufgestellten Forderungen erfüllt wurden und im letzten Abschnitt ein Ausblick auf die Weiterentwicklung des Schreibererkennungssystems gegeben werden.

9.1 Umsetzung der Anforderungen

In Abschnitt 4.2 wurden fünf Anforderungen an das System gestellt. Im Folgenden soll gezeigt werden, inwieweit diese erfüllt wurden.

Ausnutzung geeigneter Data-Mining-Verfahren Für die Umsetzung des Systems wird ein leicht abgewandeltes k-Nearest-Neighbor-Verfahren genutzt. Andere Verfahren haben sich als ungeeignet erwiesen (vgl. Abschnitt 5.1). Zudem hat sich das k-Nearest-Neighbor-Verfahren schon in anderen Systemen zur Handschrifterkennung bewiesen, wie in Abschnitt 3.1 erläutert wurde.

Datenbankintegration Die Integration des Systems in die Datenbank DB2 wird in Kapitel 7.2 diskutiert. Der Einsatz der IBM Intelligent Miner ist nicht sinnvoll, da sie keine instanzbasierten Lernverfahren unterstützen (vgl. Abschnitt 7.1). Die relevanten Daten des Schreibererkennungssystems werden in Datenbanktabellen gespeichert, wodurch ein effizienter Zugriff auf diese Informationen durch das Datenbanksystem gewährleistet ist. Darüber hinaus werden dem Nutzer Stored Procedures als Schnittstellen für die Funktionalität des Systems angeboten.

Unterstützung von Nullwerten Die Bedeutung und Behandlung von Nullwerten war und ist viel diskutiert. Das Schreibererkennungssystem unterscheidet zwischen zwei Arten von Nullwerten (vgl. Abschnitt 6.2.3).

Das ‘No-Applicable-Null’ (T) wird wie ein normales Element des Wertebereichs behandelt, dass zu allen anderen Werten die Distanz eins hat. Das Non-Information-Null (?) wird ebenfalls berücksichtigt und wie in Abschnitt 6.2.3 beschrieben gehandhabt. In Kapitel 8.5.2 wurde darüber hinaus in Tests festgestellt, dass die Anzahl der Nullwerte (?) einen Einfluss auf die Qualität der Ergebnisse hat. Je mehr Nullwerte ein Feature-Vektor enthält, desto weniger aussagekräftig ist seine Distanz zu einem anderen Feature-Vektor.

Mehrere Werte für ein Feature Eine weitere Anforderung an das System ist die Unterstützung von mehreren Werten für ein Feature. Die Umsetzung dieser so genannten komplexen Feature-Werte wurde in Abschnitt 5.2.6 beschrieben. Die optimale Implementierung des Modells für komplexe Feature-Werte konnte erst in Tests (vgl. Abschnitt 8.6) bestimmt werden. Das Ergebnis sollte aber in Zukunft durch weitere Tests mit größeren Testmengen validiert und, wenn nötig, angepasst werden.

Fehlertoleranz Die Fehlertoleranz wurde durch ein spezielles Ähnlichkeitsmaß realisiert. Das Ähnlichkeitsmaß wurde in Abschnitt 5.2.5 formal eingeführt. Kapitel 6.2.1 beschreibt die Umsetzung der Ähnlichkeiten im Prototypen. Es wird für jedes Merkmal eine symmetrische Distanzmatrix erzeugt und in der Datenbank gespeichert. Dies ist eine Abstraktion vom theoretischen Fall, in dem zwei nicht symmetrische Distanzmatrizen pro Merkmal existieren (vgl. Abschnitt 6.2.1).

9.2 Ausblick

Die vorliegende Arbeit ist eine der ersten im Rahmen des eNoteHistory-Projektes, die sich mit der Entwicklung eines Schreibererkennungssystems auf Basis einer Handschrift-Merkmal-Taxonomie beschäftigt. Das Thema war und ist es, Klassifikations- und Clustering-Verfahren und Möglichkeiten der Integration des Systems in eine Datenbankumgebung zu untersuchen und einen Prototypen zu entwickeln. Dabei haben sich einerseits neue Anforderungen aller Beteiligten des Projektes an das Schreibererkennungssystem ergeben. Andererseits wurden durch die Analysen und Diskussionen dieser Arbeit neue Wege und Ansätze aufgezeigt, die in Zukunft weiter verfolgt werden können oder sollten. Solche weiterführenden Ansätze und Optimierungsstrategien wurden jeweils direkt in den betroffenen Kapiteln erläutert. Sie sollen im Folgenden trotzdem noch einmal zusammenfassend aufgeführt werden.

Erweiterung des Modells

In Abschnitt 5.2.2 wurde bereits erklärt, dass die ursprünglich geplante Menge der Knotentypen mit $Type = \{\text{‘prefix’}, \text{‘feature’}, \text{‘value’}\}$ nicht mehr ausreicht, da noch zwischen auswählbaren und nicht auswählbaren Werten unterschieden werden muss. Diese Änderung wurde in der Datenbanktabelle ‘NODE_TYPES’ (vgl. Anhang A.7) bereits umgesetzt, muss aber noch auf das formale Modell übertragen werden.

Darüber hinaus soll es in Zukunft möglich sein, Feature zu gruppieren und eine Distanzfunktion auf einer Gruppe von Merkmalen zu definieren. Ein Beispiel dafür ist der C-Schlüssel, der durch mehrere Feature beschrieben wird:

- 1.2.1. Linkes Element
- 1.2.2.1. Markierung der C-Linie durch ein Element
- 1.2.2.2.1. Markierung der C-Linie durch zwei Elemente (oberes Element)
- 1.2.2.2.2. Markierung der C-Linie durch zwei Elemente (unteres Element)
- 1.2.3. Rechtes Element

Eine Distanzfunktion soll die Ähnlichkeit zweier C-Schlüssel basierend auf den Ähnlichkeiten ihrer Elemente berechnen. Das erfordert eine neue Terminologie, da das Prinzip der Distanzfunktion, die mit Hilfe von Feature-Distanzfunktionen berechnet wird, so nicht mehr funktioniert.

Eine weitere Möglichkeit zur Verbesserung des Systems besteht darin, Optimierungsstrategien einzusetzen, die sich im Data-Mining-Bereich für instanzbasierte Verfahren bewährt haben und in [57] ab Seite 193 beschrieben werden. Es kann zum Beispiel versucht werden, die Anzahl der Instanzen, die für das k-Nearest-Neighbor-Verfahren genutzt werden, zu minimieren. Kandidaten, die dafür geeignet sind, bei der Klassifikation einer neuen Instanz nicht berücksichtigt zu werden, sind beispielsweise Instanzen mit vielen Nullwerten oder solche, die in der Vergangenheit zu schlechten Ergebnissen bei der Schreiberbestimmung geführt haben. Eine andere Variante ist, die Gewichte der Distanzfunktion dynamisch anzupassen oder sogar für verschiedene Klassen verschiedene Gewichte zu benutzen. Dadurch kann ausgedrückt werden, dass ein bestimmtes Merkmal für eine spezielle Klasse besonders charakteristisch ist.

Die Aufgabe des Prototypen ist es, zu zeigen, dass das theoretische Modell auch praktisch umsetzbar ist. Im weiteren Verlauf des Projektes kann die Implementierung des Schreibererkennungssystems auf eine bessere Performance hin optimiert werden. Eine Möglichkeit, das zu erreichen, wurde in Abschnitt 7.3.2 angeregt. Die Idee ist es, die Distanzen nicht mehr durch Fließkommazahlen zwischen null und eins sondern durch ganze Zahlen zwischen null und hundert darzustellen. Die Genauigkeit der Werte in den Distanzmatrizen würde darunter nicht leiden, die Rechenzeit aber verbessert werden.

Eine Frage, die in dieser Arbeit nicht ausführlich diskutiert wurde, ist das Verhalten des Systems bei Updates. Aufgrund des instanzbasierten Lernansatzes sollten Änderungen an der Feature Base, den Distanzmatrizen oder ein Einfügen von neuen Instanzen in die Datenbank jederzeit problemlos möglich sein. Es ist aber sinnvoll, dieses Thema genauer zu diskutieren und Methoden zu analysieren, die während eines Updates aufgerufen werden können. Dabei kann es sich zum Beispiel um die oben genannte Methode handeln, die beim Einfügen eines neuen Feature-Vektors überprüft, inwieweit dieser für künftige Klassifikationen neuer Instanzen geeignet ist.

Erweiterung der Datenbankintegration

In Kapitel 7.2 wurden Wege aufgezeigt, welche die Transparenz des Schreibererkennungssystem für den Nutzer erhöhen. Beispiele dafür sind User-Defined Types zur Darstellung der Knoten des Feature-Base-Baumes und damit spezielle Datentypen für Merkmale und ihre Werte. Eine weitere Möglichkeit besteht darin, generierte IDs als Primärschlüssel für die Tupel (Zeilen) der Tabellen zu benutzen und somit den Knoten-Code als Schlüssel abzulösen. Dadurch können Änderungen an der Feature Base (speziell des Feature-Code) transparent und einfach unter Einhaltung der Integritätsbedingungen vollzogen werden.

Eine andere Erweiterung der Datenbankintegration des Systems wurde in Abschnitt 7.1 gezeigt. In diesem Abschnitt wurde begründet, dass für den Prototypen der Einsatz von Data-Mining-Tools nicht nötig. Die Visualisierung des DM-Modells und der Ergebnisse kann im weiteren Verlauf des Projektes an Bedeutung gewinnen. In diesem Fall ist es sinnvoll, den Einsatz der Tools, die ihre Stärke in der Visualisierung haben, nochmals zu überprüfen. Abschnitt 7.1 hat auch gezeigt, dass dann Abstriche bei der praktischen Umsetzung des in Kapitel 5.2.1 eingeführten Modells gemacht werden müssen, weil kein Tool alle Anforderungen erfüllt.

Analyse weitere Verfahren und Technologien

Ein interessanter Weg zur Verbesserung eines Systems zur Handschrifterkennung wurde in [23] aufgezeigt. Dieser Ansatz wurde in Abschnitt 3.1.2 erklärt. Er beruht darauf, ein funktionierendes Handschrifterkennungssystem basierend auf dem k-Nearest-Neighbor-Verfahren unter Information-Retrieval-Gesichtspunkten zu optimieren. Die Schlussfolgerung in [23] lautete, dass die IR-Methode ähnliche Ergebnisse erzielt, wie die zuvor entwickelten Ansätze, sich aber durch seine lineare Zeitkomplexität hervorhebt. Darüber hinaus eröffnet das aus dem IR-Bereich bekannte Verfahren zur Bestimmung der Gewichte (vgl. Abschnitt 3.1.2) neue Möglichkeiten für den Vergleich zweier Notenhandschriften. Ein anderer Aspekt ist die Berechnung von einer mit einem Ranking versehenen Antwortmenge A_S wie sie in Abschnitt 6.3.3 angedacht wurde.

Ein weiterer Punkt, der im Verlauf des eNoteHistory-Projektes von Interesse sein kann, wurde in Abschnitt 6.2.1 erläutert. Der Prototyp verwendet ein vereinfachtes Ähnlichkeitsmaß. Es kann darum untersucht werden, inwieweit eine völlig der formalen Spezifikation aus Abschnitt 5.2.5 folgende Umsetzung des Ähnlichkeitsmaßes die Ergebnisse des Systems verbessert.

Waren alle oben genannten Punkte nur Vorschläge, wie das Schreibererkennungssystem in Zukunft weiter optimiert werden kann, ist der nun folgende doch stark zu empfehlen. Für die vorliegende Arbeit lag nur eine sehr kleine Testmenge mit Feature-Vektoren vor. Es werden aber laufend neue Schreibercharakteristiken bestimmt. Darum sollten die in Abschnitt 8 vorgestellten Tests zu einem späteren Zeitpunkt mit einer größeren Testmenge noch einmal wiederholt werden, um so die Ergebnisse zu bestätigen oder an die neuen Gegebenheiten anzupassen.

Anhang A

Tabellen

A.1 Priorität der Features aus musikwissenschaftlicher Sicht

Priorität	Feature-Path-Code	Erklärung
1	1.1.1.	einziges Grundelement
2	1.1.2.	mehrzügige Schlüssel / Zusatzzeichen
3	1.2.1.	linkes Element
4	1.2.2.1.	durch ein Element
5	1.2.2.2.1.	oberes Element
6	1.2.2.2.2.	unteres Element
7	1.2.3.	rechtes Element
8	1.3.1.	Hauptelement
9	1.3.2.1.	Zusatzelement links
10	1.3.2.2.	Zusatzelement (Hauptelement schneidend)
11	1.3.2.3.	erstes Zusatzelement rechts
12	1.3.2.4.	zweites Zusatzelement rechts
13	9.1.1.1.1.	Bügel
14	9.1.1.1.2.	frei hinzugefügte Zeichen
15	9.1.1.2.	unterer Bügel
16	9.1.2.1.	Bügel
17	9.1.2.2.	Basisstrich
18	9.1.2.3.	Mittelmarke
19	9.1.3.1.1.	oberer Abschluß
20	9.1.3.1.2.	unterer Abschluß
21	9.1.3.2.	Schnittpunkt, Lage weitere Marken
22	9.2.	Eins (1)
23	9.3.1.	Bügel
24	9.3.2.	Basisstrich
25	9.3.3.	Mittelmarke
26	9.4.1.	Zahl
27	9.4.2.	Zusatzstrich
28	9.5.1.	Winkel
29	9.5.2.	Senkrechte

30	9.5.3.	Sonderformen
31	9.6.	Acht (8)
32	9.7.1.	Bruchstrich
33	9.7.2.	Ligatur von Zähler und Nenner
34	13.1.	Viertelpause
35	2.1.1.	Halbe aufwärts
36	2.1.2.	Halbe abwärts
37	2.2.1.	Viertel aufwärts
38	2.2.2.	Viertel abwärts
39	2.3.1.	Achtel aufwärts
40	2.3.2.	Achtel abwärts
41	2.4.1.	Sechzehntel aufwärts
42	2.4.2.	Sechzehntel abwärts
43	3.1.1.	Halbe aufwärts
44	3.1.2.	Halbe abwärts
45	3.2.1.	Viertel aufwärts
46	3.2.2.	Viertel abwärts
47	3.3.1.	Achtel aufwärts
48	3.3.2.	Achtel abwärts
49	3.4.1.	Sechzehntel aufwärts
50	3.4.2.	Sechzehntel abwärts
51	4.1.1.	Achtel aufwärts
52	4.1.2.	Achtel abwärts
53	4.2.1.	Sechzehntel aufwärts
54	4.2.2.	Sechzehntel abwärts
55	11.1.	Kustoden
56	11.2.	Vorhandensein von Schlüsseln
57	11.3.	Vorhandensein von Tonartvorzeichen
58	11.5.	b / # als Auflösungszeichen
59	11.4.	Format
60	12.	Laufweite (in Stimmen)
61	5.1.	Relation Balken - Stiele
62	5.2.	Verlauf
63	5.3.	Abstand bei Mehrfachbebalkung
64	6.1.	Ligatur m. und Schlüssel ?
65	6.2.	Tonartvorzeichen ligiert ?
66	6.3.	b-Vorzeichen
67	6.4.	Auflösungszeichen
68	6.5.	# - Vorzeichen
69	8.1.	schwarz
70	8.2.1.1.	aufwärts
71	8.2.1.2.	abwärts
72	8.2.2.1.	aufwärts
73	8.2.2.2.	abwärts
74	10.1.	erstes Zeichen
75	10.2.	weiteres Zeichen
76	7.1.	Größe

77	7.2.	Linienabstände im System
78	7.3.	Anordnung der Systeme

A.2 Testergebnis: Priorität der Features

Feature	SF-Nenner	SF-Zähler	SF	Feature	SF-Nenner	SF-Zähler	SF
9.7.1.	0,132	0,411	3,122	8.2.1.1.	0,128	0,412	3,205
1.3.2.3.	0,130	0,407	3,125	9.1.1.1.2.	0,128	0,411	3,205
11.3.	0,131	0,415	3,156	4.2.1.	0,128	0,411	3,205
6.2.	0,131	0,414	3,163	1.2.3.	0,128	0,411	3,206
13.1.	0,128	0,405	3,164	9.6.	0,128	0,410	3,206
5.1.	0,128	0,405	3,168	1.3.2.2.	0,128	0,411	3,207
3.1.2.	0,130	0,411	3,170	9.1.2.1.	0,128	0,411	3,207
9.5.2.	0,129	0,410	3,171	normal	0,128	0,411	3,207
1.3.1.	0,127	0,403	3,171	1.1.2.	0,128	0,411	3,207
9.5.1.	0,130	0,412	3,174	1.3.2.1.	0,128	0,411	3,207
3.2.2.	0,128	0,408	3,175	9.1.2.3.	0,128	0,411	3,207
				9.2.	0,128	0,411	3,207
3.4.2.	0,128	0,408	3,177	9.3.3.	0,128	0,411	3,207
9.4.1.	0,129	0,410	3,179	9.4.2.	0,128	0,411	3,207
9.1.1.1.1.	0,128	0,407	3,179	9.5.3.	0,128	0,411	3,207
11.2.	0,131	0,417	3,181	9.7.2.	0,128	0,411	3,207
11.1.	0,129	0,411	3,181	7.1.	0,128	0,411	3,207
1.2.2.2.2.	0,129	0,409	3,182	9.1.2.2.	0,128	0,411	3,207
1.2.2.2.1.	0,129	0,409	3,183	9.3.2.	0,128	0,411	3,207
5.2.	0,129	0,413	3,187	8.2.2.2.	0,128	0,411	3,210
3.2.1.	0,133	0,423	3,190	6.3.	0,129	0,414	3,210
4.1.2.	0,130	0,414	3,190	1.2.1.	0,129	0,413	3,215
8.1.	0,133	0,423	3,190	4.1.1.	0,129	0,414	3,219
6.4.	0,128	0,410	3,191	8.2.1.2.	0,127	0,410	3,220
1.3.2.4.	0,129	0,413	3,193	2.1.1.	0,127	0,409	3,224
3.1.1.	0,130	0,416	3,194	2.1.2.	0,127	0,411	3,225
1.2.2.1.	0,128	0,410	3,196				
6.1.	0,129	0,411	3,197	9.3.1.	0,127	0,410	3,227
11.5.	0,129	0,411	3,197	3.3.2.	0,126	0,408	3,230
9.1.3.1.2.	0,129	0,411	3,198	5.3.	0,126	0,408	3,235
2.4.2.	0,129	0,411	3,198	12.	0,127	0,410	3,238
3.4.1.	0,130	0,417	3,199	9.1.1.2.	0,128	0,414	3,242
2.4.1.	0,128	0,410	3,199	2.3.2.	0,126	0,410	3,250
8.2.2.1.	0,129	0,411	3,200	2.3.1.	0,125	0,408	3,259
3.3.1.	0,131	0,420	3,201	2.2.1.	0,125	0,409	3,277
10.1.	0,128	0,411	3,201	2.2.2.	0,125	0,410	3,291
10.2.	0,128	0,411	3,202	7.2.	0,123	0,408	3,309
9.1.3.1.1.	0,129	0,412	3,202	6.5.	0,123	0,407	3,320
9.1.3.2.	0,129	0,412	3,203	11.4.	0,122	0,410	3,344
7.3.	0,128	0,410	3,203	1.1.1.	0,119	0,401	3,373
4.2.2.	0,128	0,410	3,204				

A.3 Testergebnis: Distanzfunktion

SF-Nenner	SF-Zähler	SF	K	PR	RC	Distanzfunktion	Gewichte	Ähnlichkeiten	Prioritäten
0,128	0,411	3,207	0,865	0,880	0,686	Hamming	konstant	Standard	Musikwissenschaftlich
0,300	0,594	1,981	0,876	0,880	0,686	Euklidische	konstant	Standard	Musikwissenschaftlich
0,132	0,432	3,283	0,876	0,861	0,705	Hamming	linear	Standard	Musikwissenschaftlich
0,302	0,608	2,011	0,843	0,886	0,649	Euklidische	linear	Standard	Musikwissenschaftlich
0,151	0,491	3,251	0,831	0,815	0,650	Hamming	quadratisch	Standard	Musikwissenschaftlich
0,313	0,651	2,081	0,820	0,806	0,570	Euklidische	quadratisch	Standard	Musikwissenschaftlich
0,109	0,360	3,297	0,843	0,889	0,671	Hamming	konstant	Translation (-0,2)	Musikwissenschaftlich
0,272	0,548	2,010	0,775	0,861	0,642	Euklidische	konstant	Translation (-0,2)	Musikwissenschaftlich
0,111	0,377	3,398	0,843	0,870	0,649	Hamming	linear	Translation (-0,2)	Musikwissenschaftlich
0,273	0,561	2,053	0,787	0,833	0,610	Euklidische	linear	Translation (-0,2)	Musikwissenschaftlich
0,128	0,433	3,379	0,820	0,824	0,561	Hamming	quadratisch	Translation (-0,2)	Musikwissenschaftlich
0,282	0,606	2,152	0,775	0,772	0,496	Euklidische	quadratisch	Translation (-0,2)	Musikwissenschaftlich
0,101	0,334	3,295	0,831	0,840	0,686	Hamming	konstant	Skalierung (/2)	Musikwissenschaftlich
0,259	0,522	2,021	0,719	0,795	0,743	Euklidische	konstant	Skalierung (/2)	Musikwissenschaftlich
0,103	0,351	3,416	0,865	0,843	0,669	Hamming	linear	Skalierung (/2)	Musikwissenschaftlich
0,258	0,535	2,073	0,742	0,850	0,618	Euklidische	linear	Skalierung (/2)	Musikwissenschaftlich
0,120	0,409	3,406	0,798	0,778	0,543	Hamming	quadratisch	Skalierung (/2)	Musikwissenschaftlich
0,266	0,582	2,190	0,730	0,809	0,444	Euklidische	quadratisch	Skalierung (/2)	Musikwissenschaftlich
0,128	0,411	3,207	0,865	0,880	0,686	Hamming	konstant	Standard	Nach Trainingsmenge
0,300	0,594	1,981	0,876	0,880	0,686	Euklidische	konstant	Standard	Nach Trainingsmenge
0,090	0,398	4,397	0,865	0,901	0,738	Hamming	linear	Standard	Nach Trainingsmenge
0,246	0,586	2,382	0,831	0,856	0,732	Euklidische	linear	Standard	Nach Trainingsmenge
0,059	0,400	6,741	0,820	0,882	0,721	Hamming	quadratisch	Standard	Nach Trainingsmenge
0,163	0,591	3,613	0,798	0,851	0,659	Euklidische	quadratisch	Standard	Nach Trainingsmenge
0,109	0,360	3,297	0,843	0,889	0,671	Hamming	konstant	Translation (-0,2)	Nach Trainingsmenge
0,272	0,548	2,010	0,775	0,861	0,642	Euklidische	konstant	Translation (-0,2)	Nach Trainingsmenge
0,076	0,351	4,642	0,854	0,914	0,656	Hamming	linear	Translation (-0,2)	Nach Trainingsmenge
0,223	0,545	2,441	0,809	0,880	0,560	Euklidische	linear	Translation (-0,2)	Nach Trainingsmenge
0,049	0,358	7,350	0,798	0,851	0,652	Hamming	quadratisch	Translation (-0,2)	Nach Trainingsmenge
0,146	0,554	3,796	0,742	0,820	0,527	Euklidische	quadratisch	Translation (-0,2)	Nach Trainingsmenge
0,101	0,334	3,295	0,831	0,831	0,686	Hamming	konstant	Skalierung (/2)	Nach Trainingsmenge
0,259	0,522	2,021	0,742	0,830	0,648	Euklidische	konstant	Skalierung (/2)	Nach Trainingsmenge
0,071	0,329	4,642	0,831	0,935	0,547	Hamming	linear	Skalierung (/2)	Nach Trainingsmenge
0,212	0,522	2,465	0,742	0,840	0,565	Euklidische	linear	Skalierung (/2)	Nach Trainingsmenge
0,047	0,339	7,232	0,775	0,837	0,645	Hamming	quadratisch	Skalierung (/2)	Nach Trainingsmenge
0,139	0,534	3,851	0,742	0,781	0,460	Euklidische	quadratisch	Skalierung (/2)	Nach Trainingsmenge

A.4 Testergebnis: Gewichte der Features

SF-Nenner	SF-Zähler	SF	K	Gewichte	Ähnlichkeiten	Prioritäten
0,128	0,411	3,207	0,865	konstant	Standard	Musikwissenschaftlich
0,130	0,429	3,288	0,876	linear gruppiert	Standard	Musikwissenschaftlich
0,132	0,432	3,283	0,876	linear	Standard	Musikwissenschaftlich
0,136	0,450	3,322	0,854	quadratisch gruppiert	Standard	Musikwissenschaftlich
0,151	0,491	3,251	0,831	quadratisch	Standard	Musikwissenschaftlich
0,109	0,360	3,297	0,843	konstant	Translation (-0,2)	Musikwissenschaftlich
0,110	0,374	3,413	0,854	linear gruppiert	Translation (-0,2)	Musikwissenschaftlich
0,111	0,377	3,398	0,843	linear	Translation (-0,2)	Musikwissenschaftlich
0,113	0,393	3,473	0,820	quadratisch gruppiert	Translation (-0,2)	Musikwissenschaftlich
0,128	0,433	3,379	0,820	quadratisch	Translation (-0,2)	Musikwissenschaftlich
0,101	0,334	3,295	0,865	konstant	Skalierung (/2)	Musikwissenschaftlich
0,101	0,348	3,442	0,854	linear gruppiert	Skalierung (/2)	Musikwissenschaftlich
0,103	0,351	3,416	0,865	linear	Skalierung (/2)	Musikwissenschaftlich
0,105	0,369	3,519	0,798	quadratisch gruppiert	Skalierung (/2)	Musikwissenschaftlich
0,120	0,409	3,406	0,798	quadratisch	Skalierung (/2)	Musikwissenschaftlich
0,128	0,411	3,207	0,865	konstant	Standard	Nach Trainingsmenge
0,091	0,395	4,354	0,876	linear gruppiert	Standard	Nach Trainingsmenge
0,090	0,398	4,397	0,865	linear	Standard	Nach Trainingsmenge
0,070	0,388	5,515	0,843	quadratisch gruppiert	Standard	Nach Trainingsmenge
0,059	0,400	6,741	0,820	quadratisch	Standard	Nach Trainingsmenge
0,109	0,360	3,297	0,843	konstant	Translation (-0,2)	Nach Trainingsmenge
0,076	0,350	4,580	0,854	linear gruppiert	Translation (-0,2)	Nach Trainingsmenge
0,076	0,351	4,642	0,854	linear	Translation (-0,2)	Nach Trainingsmenge
0,059	0,346	5,904	0,820	quadratisch gruppiert	Translation (-0,2)	Nach Trainingsmenge
0,049	0,358	7,350	0,798	quadratisch	Translation (-0,2)	Nach Trainingsmenge
0,101	0,334	3,295	0,831	konstant	Skalierung (/2)	Nach Trainingsmenge
0,072	0,328	4,575	0,820	linear gruppiert	Skalierung (/2)	Nach Trainingsmenge
0,071	0,329	4,642	0,831	linear	Skalierung (/2)	Nach Trainingsmenge
0,056	0,327	5,868	0,809	quadratisch gruppiert	Skalierung (/2)	Nach Trainingsmenge
0,047	0,339	7,232	0,775	quadratisch	Skalierung (/2)	Nach Trainingsmenge

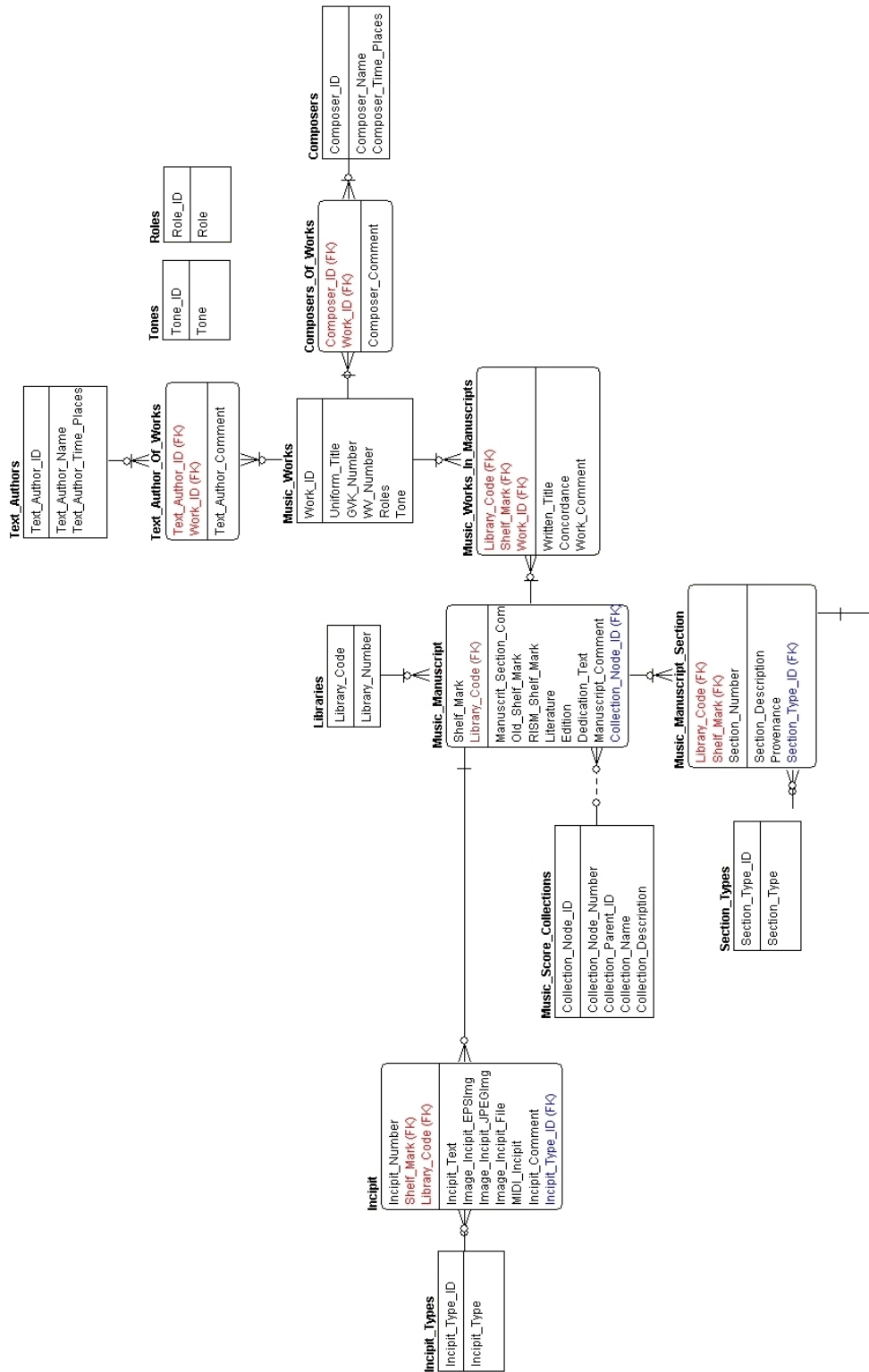
A.5 Testergebnis: Ähnlichkeitswerte

SF-Nenner	SF-Zähler	SF	Ähnlichkeiten	Prioritäten
0,151	0,491	0,308	Standard	Musikwissenschaftlich
0,128	0,433	0,296	Translation (-0,2)	Musikwissenschaftlich
0,139	0,462	0,302	Translation (-0,1)	Musikwissenschaftlich
0,163	0,521	0,313	Translation (+0,1)	Musikwissenschaftlich
0,175	0,549	0,318	Translation (+0,2)	Musikwissenschaftlich
0,105	0,368	0,284	Skalierung (/4)	Musikwissenschaftlich
0,120	0,409	0,294	Skalierung (/2)	Musikwissenschaftlich
0,185	0,573	0,323	Skalierung (*2)	Musikwissenschaftlich
0,115	0,403	0,286	Kubisch	Musikwissenschaftlich
0,128	0,435	0,294	Quadratisch	Musikwissenschaftlich
0,172	0,541	0,318	Wurzel	Musikwissenschaftlich
0,182	0,564	0,322	Dritte Wurzel	Musikwissenschaftlich
0,059	0,400	0,148	Standard	Nach Trainingsmenge
0,049	0,358	0,136	Translation (-0,2)	Nach Trainingsmenge
0,054	0,379	0,142	Translation (-0,1)	Nach Trainingsmenge
0,065	0,422	0,154	Translation (+0,1)	Nach Trainingsmenge
0,070	0,441	0,160	Translation (+0,2)	Nach Trainingsmenge
0,041	0,308	0,132	Skalierung (/4)	Nach Trainingsmenge
0,047	0,339	0,138	Skalierung (/2)	Nach Trainingsmenge
0,076	0,462	0,164	Skalierung (*2)	Nach Trainingsmenge
0,043	0,336	0,129	Kubisch	Nach Trainingsmenge
0,049	0,359	0,135	Quadratisch	Nach Trainingsmenge
0,070	0,435	0,161	Wurzel	Nach Trainingsmenge
0,075	0,451	0,167	Dritte Wurzel	Nach Trainingsmenge

A.6 Testergebnis: Schwellwert, Precision, Recall, K und Nullwerte

Schwellwert t	Precision PR	Recall RE	Bewertung K	N_D der richtig klas. Instanzen	N_D der falsch klas. Instanzen
0,44	0,231	0,918	0,382	0,392	0,357
0,43	0,234	0,918	0,382	0,392	0,357
0,42	0,234	0,918	0,382	0,392	0,357
0,41	0,234	0,918	0,382	0,392	0,357
0,40	0,256	0,918	0,382	0,392	0,357
0,39	0,257	0,918	0,382	0,392	0,357
0,38	0,260	0,918	0,382	0,392	0,357
0,37	0,263	0,918	0,382	0,392	0,357
0,36	0,263	0,915	0,382	0,392	0,357
0,35	0,265	0,915	0,382	0,392	0,357
0,34	0,274	0,915	0,382	0,392	0,357
0,33	0,279	0,915	0,404	0,391	0,349
0,32	0,290	0,915	0,404	0,391	0,349
0,31	0,304	0,915	0,404	0,391	0,349
0,30	0,335	0,911	0,427	0,388	0,349
0,29	0,353	0,911	0,449	0,388	0,357
0,28	0,378	0,896	0,472	0,388	0,357
0,27	0,430	0,896	0,483	0,391	0,352
0,26	0,453	0,873	0,483	0,390	0,344
0,25	0,494	0,873	0,494	0,396	0,335
0,24	0,525	0,873	0,528	0,399	0,332
0,23	0,552	0,853	0,562	0,405	0,321
0,22	0,616	0,810	0,607	0,403	0,327
0,21	0,659	0,797	0,663	0,403	0,327
0,20	0,668	0,766	0,674	0,403	0,332
0,19	0,694	0,747	0,753	0,410	0,337
0,18	0,748	0,732	0,787	0,410	0,250
0,17	0,759	0,732	0,820	0,410	0,304
0,16	0,827	0,724	0,854	0,410	0,215
0,15	0,843	0,712	0,876	0,410	
0,14	0,843	0,705	0,865	0,408	
0,13	0,861	0,705	0,876	0,408	
0,12	0,867	0,668	0,876	0,405	
0,11	0,864	0,600	0,831	0,407	
0,10	0,883	0,550	0,787	0,403	
0,09	0,892	0,529	0,775	0,404	
0,08	0,892	0,501	0,775	0,404	
0,07	0,892	0,438	0,730	0,423	
0,06	0,892	0,355	0,685	0,438	
0,05	0,898	0,306	0,652	0,447	
0,04	0,898	0,213	0,573	0,448	
0,03	0,898	0,213	0,573	0,448	
0,02	0,898	0,167	0,517	0,458	
0,01	0,898	0,093	0,461	0,481	

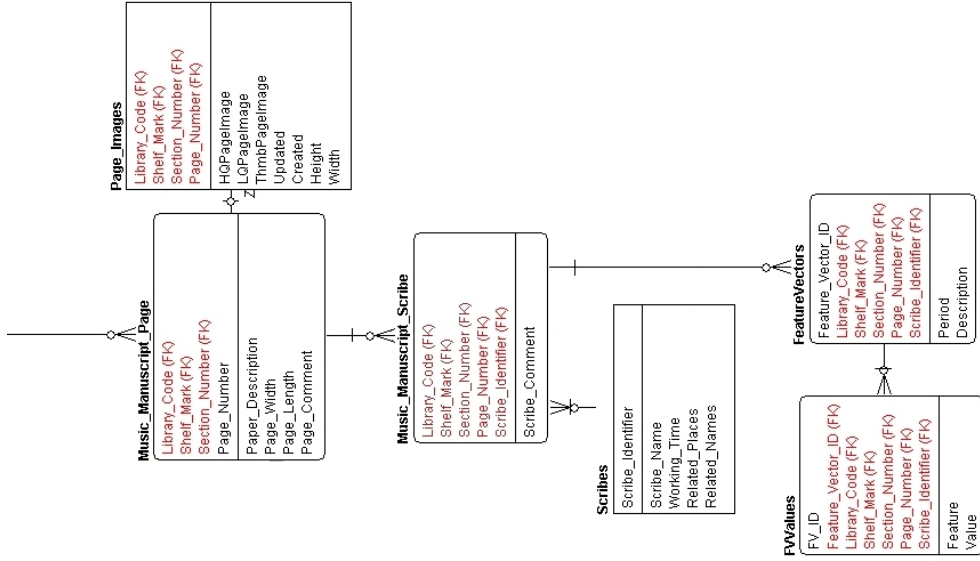
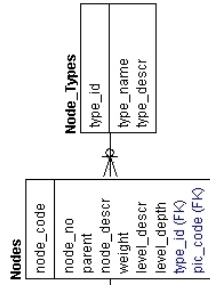
A.7 eNoteHistory Modell



Feature Base

Pictogram
pic_code
pic

Distances
VALUE1
VALUE2
FEATURE
DISTANCE



A.8 Abkürzungs- und Symbolverzeichnis

	Bedeutung	Eingeführt in Abschnitt
A_{FV}	Antwortmenge der Feature-Vektoren	6.2
A_S	Antwortmenge der Schreiber	6.2
d_Γ	Distanzfunktion	5.2.3
d_{f_i}	Feature-Distanzfunktion	5.2.4
$d_{f_i}^{Bool}$	Boolsche Feature-Distanzfunktion	5.2.4
$d_{f_i}^{Ana}$	Feature-Distanzfunktion mit Wertanalyse	5.2.4
$d_{f_i}^{Inf}$	Feature-Distanzfunktion mit Distanzmatrix	5.2.4
$d_{f_i}^K$	Komplexe Feature-Distanzfunktion	5.2.6
D	Menge aller Distanzfunktionen	6.3.4
DM	Data Mining	2.2
f_i	Feature	5.2.1
F	Menge aller Feature	5.2.1
FB	Feature Base	5.2.2
IR	Information Retrieval	3.1.2, 6.3.2
K	Bewertungsmaß	6.3.3
N_D	Bewertungsmaß für Nullwerte	6.3.4
N_F	Bewertungsmaß für Nullwerte	6.3.4
SF	Bewertungsmaß	6.3.1
SP	Stored Procedure	7.2.1
UDF	User-Defined Function	7.2
UDT	User-Defined Type	7.2
W_i	Wertebereich von f_i	5.2.1
γ	Feature-Vektor	5.2.1
γ^K	Feature-Vektor mit komplexen Werten	5.2.6
Γ	Menge aller Feature-Vektoren	5.2.1
Λ_x	Menge aller Nachkommen von x	5.2.2
μ	Bezeichnung eines Knotens	5.2.2
ν	Nummer eines Knotens	5.2.2
τ	Typ eines Knotens des FB	5.2.2
$x \mapsto y$	y ist Kindknoten von x	5.2.2
$x \mapsto \mapsto y$	y ist Nachkomme von x	5.2.2
?	Non-Information-Null	5.2.7
\top	No-Applicable-Null	5.2.7

Literaturverzeichnis

- [1] Sas-homepage. <http://www.sas.com>.
- [2] Spss-homepage. <http://www.spss.com>.
- [3] *dtv-Atlas zur Musik. Historischer Teil*. Deutscher Taschenbuch Verlag, München, 1989.
- [4] *dtv-Atlas zur Musik. Systematischer Teil*. Deutscher Taschenbuch Verlag, München, 1989.
- [5] *Darwin Reference. Release 3.0.1.*, 1998.
- [6] *Die Musik in Geschichte und Gegenwart. Allgemeine Enzyklopädie der Musik. Band 1-9*. Bärenreiter Verlag, Kassel, 1998.
- [7] *Using Darwin. Release 3.0.1.*, 1998.
- [8] *Darwin Installation and Administration. Release 3.7*, 2000.
- [9] *Darwin New Features. Release 3.7*, 2000.
- [10] *IBM DB2 Universal Database Version 7. Application Building Guide*, 2000.
- [11] *IBM DB2 Universal Database Version 7. Application Development Guide*, 2000.
- [12] *IBM DB2 Universal Database Version 7. Call Level Interface Guide and Reference*, 2000.
- [13] *IBM DB2 Universal Database Version 7. SQL First Steps*, 2000.
- [14] *IBM DB2 Universal Database Version 7. SQL Reference*, 2000.
- [15] *DB2 Intelligent Miner for Data. Application Programming Interface and Utility Reference*, first edition, 2002.
- [16] *DB2 Intelligent Miner Modeling*, first edition, 2002.
- [17] *DB2 Intelligent Miner Scoring Administration and Programming for DB2*, first edition, 2002.
- [18] *DB2 Intelligent Miner Visualization*, first edition, 2002.
- [19] *Using DB2 Intelligent Miner for Data*, first edition, 2002.
- [20] D.W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies* 36(1), 1992.
- [21] D.W. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning, vol.6*, 1991.
- [22] A. Bensefia, A. Nosary, T. Paquet, and L. Heutte. Writer identification by writer's invariants. *International Workshop on Frontiers in Handwriting Recognition*, 2001.
- [23] A. Bensefia, T. Paquet, and L. Heutte. Information retrieval based writer identification. *5th International Conference on Enterprise Information Systems*, 2003.
- [24] P. Berkhin. *Survey of Clustering Data Mining Techniques*. Accrue Software, San Jose, CA.
- [25] B. Boser, I. Guyon, and V. Vapnik. An training algorithm for optimal margin classifiers. *Fifth Annual Workshop on Computational Learning Theory:144-152*, 1992.
- [26] I. Bruder, A. Finger, A. Heuer, and T. Ignatova. Towards a digital document archive for historical handwritten music scores. In *6th International Conference of Asian Digital Libraries ICADL*, Kuala Lumpur, Malaysia, 2003.

- [27] E. F. Codd. Extending the database relational model to capture more meaning, *acm trans. database systems*, vol. 4, no. 4. 1979.
- [28] J. T. Favata and G. Srikantan. A multiple feature/resolution approach to handprinted digit and character recognition. *International Journal of Imaging Systems and Technology* 7, 1996.
- [29] R. Ferber. *Information Retrieval. Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. dpunkt.verlag, Heidelberg, 2003.
- [30] D. H. Fisher. in *Machine Learning*, 2. 1987.
- [31] Center for Automated Learning and Discovery. *Lab Software*. Carnegie Mellon University, Pittsburgh, PA.
- [32] M. Gluck and J. Corter. in *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum, Irvine, CA., 1985.
- [33] M. Goebel and L. Gruenwald. *A Survey of Data Mining and Knowledge Discovery Software Tools*.
- [34] J. Grant. Null values in a relational data base. *information processing letters*, vol. 6, no. 5. 1977.
- [35] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.
- [36] A. Heuer and G. Saake. *Datenbanken: Konzepte und Sprachen*. International Thomson Publishing, Bonn, 1997.
- [37] A. Heuer and G. Saake. *Datenbanken: Implementierungstechniken*. MITP-Verlag, Bonn, 1999.
- [38] J. F. Elder IV and D. W. Abbott. *A Comparison of Leading Data Mining Tools*. Fourth International Conference on Knowledge Discovery and Data Mining, New York, NY, 1998.
- [39] R. Kohavi. *MLC++ Tutorial*. Stanford, CA, 1995.
- [40] R. Kohavi, G. John, R. Long, D. Manley, and K. Pfleger. *Mlc++: A machine learning library in c++*. 1994.
- [41] R. Kohavi and D. Sommerfield. *MLC++ Machine Learning Library in C++*, 1996.
- [42] E. Krüger. Die musikaliensammlungen des erbprinzen friedrich ludwig von württemberg stuttgart und der herzogin luise friederike von mecklenburg-schwerin in der universitätsbibliothek rostock. dissertation., 2003.
- [43] E. Krüger and T. Schwinger. <http://www.phf.uni-rostock.de//fkw/imu/-enh/index.htm>. 2003.
- [44] Y. E. Lien. Multivalued dependencies with null values in relational databases. *proc. 5th int. conf. on very large data bases*. 1979.
- [45] R. Long. *MLC++ Library Classes*, 1994.
- [46] J. MacQueen. Some methods for classification and analysis of multivariate observations. In M. Lucien, L. Cam, and J. Neyman, editors, *Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I, Statistics*. University of California Press, 1967.
- [47] N. Miskov. Survey of tools for data mining.
- [48] A. Nosary. Reconnaissance automatique des textes manuscrits par adaptation au scribeur. *Thèse de Doctorat, Université de Rouen*, 2003.
- [49] A. Nosary, L. Heutte, T. Paquet, and Y. Lecourtier. Writer identification by writer's invariants. *International Workshop on Frontiers in Handwriting Recognition*, 1999.
- [50] J. R. Quinlan. Induction of decision trees. *Machine Learning* 1 (1):81-106, 1986.
- [51] J. R. Quinlan. *C4.5: Programs for machine learning*. Academic Press, Morgan Kaufmann, San Francisco, CA., 1993.

- [52] R. Reiter. Data bases, a logical perspective. proc. workshop on data abstraction databases and conceptual modeling. 1980.
- [53] S. N. Srihari, S.-H. Cha, H. Arora, and S. Lee. Individuality of handwriting. *Journal of Forensic Sciences* 47(4), 2002.
- [54] C. J. van Rijsbergen. *Information Retrieval*. 1979.
- [55] Y. Vassiliou. Null values in data base management: A denotational semantics approach. *proc. acm sigmod int. conf. management of data*. 1979.
- [56] A. C. Vinci. *Die Notenschrift. Grundlagen der traditionellen Musiknotation*. Bärenreiter Verlag, Kassell, 1988.
- [57] Ian H. Witten and Eibe Frank. *Data Mining - Practical Machine Learning Tools and Techniques with Java Implementations*. Academic Press, Morgan Kaufmann, San Francisco, CA., 2000.
- [58] C. Zaniolo. Database relations with null values. *journal of computer system sciences*. 1982.
- [59] B. Zhang, S. N. Srihari, and S. Lee. Individuality of handwritten characters. *7th International Conference on Document Analysis and Recognition*, 2003.

Tabellenverzeichnis

2.1	Konstruiertes Beispiel mit Schriftmerkmalen von zwei Schreibern	18
2.2	Untersuchung der Attribute des Schreiberbeispiels	22
2.3	Distanzen im ersten (a) und zweiten (b) Durchlauf des K-Means-Verfahrens	26
2.4	Abstandstabelle nach dem ersten (a) und dritten (b) Durchlauf des Single-Linkage-Verfahrens	27
6.1	Features mit Boolescher Feature-Distanzfunktion	59
6.2	Features mit Wertevergleich durch Wertanalyse	60
6.3	Nullwerte am Beispiel des linken C-Schlüssel-Elementes (1.2.1.)	62
7.1	Übersicht über die Anforderungen an die Tools	68
7.2	Die Definition der Stored Procedures in Java	71
7.3	Einbindung der Stored Procedure in die Datenbank	72
7.4	Die Umsetzung der Distanzfunktion in Java	78
7.5	Die Stored Procedures in Pseudo-Code	79
8.1	Zusammenfassung der Testergebnisse für Prioritäten der Attribute	81
8.2	Zusammenfassung der Testergebnisse für die Distanzfunktion	83
8.3	Zusammenfassung der Testergebnisse für die Gewichte in der Distanzfunktion	85
8.4	Testszenario für die Ähnlichkeitswerte	86
8.5	Zusammenfassung der Testergebnisse für die Distanzwerte	87
8.6	Zusammenfassung der Testergebnisse für Attribute mit komplexen Werten	89

Abbildungsverzeichnis

2.1	Verschiedene Noten und Pausen	13
2.2	G-Schlüssel, C-Schlüssel und F-Schlüssel	14
2.3	Die Merkmalsgruppe der Fähnchen.	16
2.4	Die Werte des Merkmales 4.1.1.	17
2.5	Ein (nicht optimaler) Entscheidungsbaum	21
2.6	Mögliche Wurzeln des Entscheidungsbaums	23
2.7	Dendrogramm für das Schreiberbeispiel	27
4.1	Einfügen einer neuen Schreibercharakteristik	39
4.2	Anfrage nach einem Schreiber	40
5.1	Beispiel für ähnliche Feature-Werte	52
6.1	Systemarchitektur	56
6.2	Detaillierter Ablauf einer Anfrage	57
6.3	Zwei mögliche Werte für das Viertelpause-Attribut	60
6.4	Eine gute (a) und eine schlechte (b) Partition	63
7.1	Das ‘datatypes’-Package in Java	77

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 11. März 2004

Thesen

Jeder Schreiber hat eine für ihn typische, eindeutige Schreibercharakteristik, die auf einer Menge von Schriftmerkmalen basiert.

Das musikwissenschaftliche Fachwissen bedingt ein eigenes Ähnlichkeitsmaß.

Das Klassifikationsverfahren k-Nearest-Neighbor ist zur Notenschreiberidentifikation gut geeignet und hat sich bereits in Verfahren der Schreiberidentifikation bewährt.

Vorhandene Data-Mining-Tools sind für die Notenschreiberidentifikation nicht besonders geeignet, weil ihre Funktionalität für diese spezielle Anwendung zu allgemein ist.

Das vorgestellte Konzept einer Feature Base ist für alle Anforderungen an die Notenschreiberidentifikation im Rahmen des eNoteHistory-Projektes geeignet.

Die prototypische, Java-basierte und damit plattformunabhängige Umsetzung des entwickelten Modells demonstriert den möglichen Einsatz eines Notenschreibererkennungssystems.

Verschiedene Arten von Nullwerten sowie komplexe Werte haben einen signifikanten Einfluss auf die Ergebnisse der Notenschreiberidentifikation.