Effizienter interaktiver Entwurf von Klassifikationssystemen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Tim-Oliver Müller

aus Dortmund

Tag der mündlichen Prüfung: 21. Dezember 2005

Erster Gutachter: Prof. Dr. R. Dillmann

Zweiter Gutachter: Prof. Dr. H. Gemmeke

Zusammenfassung

Bei dem Entwurf eines Klassifikationssystems muß eine Prozeßkette bestehend aus Vorverarbeitung, Merkmalsextraktion, Merkmalsselektion und Klassifikation aufgebaut werden. Im Allgemeinen wird die Vorverarbeitung und Merkmalsextraktion mit einem anderen Werkzeug als dem für die Merkmalsselektion und Klassifikation durchgeführt. Bei diesem Vorgehen kombiniert man die jeweils zur Verfügung stehenden Algorithmen, bis man mit dem Ergebnis zufrieden und der Generalisierungsfehler möglichst niedrig ist.

Um das beste Klassifikationssystem mit dem niedrigsten Generalisierungsfehler zu erhalten, müssen theoretisch alle möglichen Algorithmenkombinationen für Merkmalsselektion und Klassifikation getestet werden. Durch das iterative Vorgehen und den Wechsel zwischen den Werkzeugen ist der Entwurf zeitaufwendig und umständlich. Ein auf Klassifikationssysteme spezialisiertes Entwicklungswerkzeug erleichtert den Entwurf von Klassifikationssystemen, wenn es nicht nur entsprechende Algorithmen für den vollständigen Aufbau einer Prozeßkette in sich vereint, sondern den Entwickler auch durch Vorschläge bei der Auswahl geeigneter Algorithmen für die Merkmalsselektion und Klassifikation unterstützt. Für eine solche Unterstützung muß das Entwicklungswerkzeug über eine Meta-Ebene verfügen, mit deren Hilfe eine automatische Rückkopplung von Ergebnissen der Prozeßkette in den Entwurf möglich ist. Bislang kann mit keinem der bekannten Softwaretools der Aufbau von Klassifikationssystemen aktiv unterstützt werden, da keines die notwendige Meta-Ebene anbietet.

Im Rahmen dieser Arbeit wurde eine Komponentensoftware speziell für den effizienten, interaktiven Entwurf von Klassifikationssystemen entwickelt, die unter anderem auch eine Meta-Ebene für die Rückkopplung von Ergebnissen der Prozeßkette und der Steuerung des Entwurfsprozesses bereit stellt. Das Prinzip der Komponentensoftware ist ideal geeignet für den schnellen Aufbau von Prozeßketten. Dadurch wird der zur Zeit übliche manuelle Entwurf von Klassifikationssystemen erheblich vereinfacht. Zudem sind in dieser Komponentensoftware Methoden implementiert, die unter Zuhilfenahme der Meta-Ebene die Auswahl geeigneter Merkmalsselektoren und Klassifikatoren unterstützen. Hierfür wurde ein effizienter „Bubble-Search“ Algorithmus entwickelt, der das NP-vollständige Problem der automatischen Auswahl auf ein zweidimensionales Suchproblem beschränkt. Mit Hilfe dieser Methode wurde eine Automatisierung des Entwurfsprozesses erreicht.

Eingesetzt wurde die neue Bubble-Search Methode bisher in der Brustkrebsfrüherkennung, der Lymphozytendetektion und der Lunkerdetektion in Aluminiumspritzguß-Werkstücken. Mit der entwickelten Komponentensoftware wurden 15 verschiedene Klassifikationsprobleme untersucht, für die der „beste“ Generalisierungsfehler bekannt war. Die automatisch erstellten Klassifikationssysteme erreichten Generalisierungsfehler mit in den meisten Fällen besserer, aber keinesfalls schlechterer Qualität. Je nach Komplexität des jeweiligen Problems erforderte der automatische Aufbau der Klassifikationssysteme zwischen wenigen Minuten und einigen Stunden. Für einen vergleichbaren manuellen Entwurf ohne Zuhilfenahme des Entwicklungswerkzeuges wurden Tage oder Wochen benötigt. Das Entwicklungswerkzeug mit den integrierten Methoden für die automatische Algorithmenauswahl stellt eine bedeutsame Neuerung für den Entwurf von Klassifikationssystemen dar.

Aufgrund ihres universellen Charakters wurde die neue Komponentensoftware auch in anderen Projekten erfolgreich eingesetzt, beispielsweise im Bereich der Mikrosystemtechnik für die Steuerung elektronischer Nasen. Mit Hilfe der graphischen Benutzeroberfläche konnten in kürzester Zeit umfangreiche Prozeßketten für die Bildverarbeitung entworfen und angewendet werden. Für die Medizintechnik wurden mit der Komponentensoftware neue Algorithmen entwickelt und erprobt. Mit ihrer Flexibilität, der effizienten graphischen Benutzeroberfläche und der einfachen Integration neuer Algorithmen ist die entwickelte Komponentensoftware ein wertvolles Entwicklungswerkzeug nicht nur für die Klassifikation.

Karlsruhe, den 7. November 2006

Danksagung

Das Anfertigen einer Promotion ist ohne Unterstützung aus dem Umfeld eines Promovierenden undenkbar. So haben auch mich viele Menschen im Laufe meiner Promotion auf vielfältig verschiedene Weise unterstützt. An dieser Stelle möchte ich all diesen Menschen meinen tief empfundenen Dank aussprechen.

Ganz besonders bedanken möchte ich mich bei meiner Frau, Antje Müller, die mich in dieser langen Zeit unermüdlich bestärkt und immer wieder liebevoll aufgerichtet hat. Während dieser Zeit hat sie sicherlich viele Einschränkungen in Kauf nehmen müssen.

Ebenfalls ganz speziell danken möchte ich meinen Eltern, die mir meine Ausbildung und insbesondere mein Studium ermöglicht haben. Ich stünde heute nicht an dieser Stelle, wenn sie mir nicht in vielen Belangen, natürlich vor allem in finanzieller Hinsicht, umfassend unter die Arme gegriffen hätten.

Schließlich wird mir Rainer Stotzka ganz besonders im Gedächtnis bleiben, der mich in seiner umfassenden Fachkenntnis angeleitet und tatkräftig betreut hat. Auch ist er mir während dieser Zeit ein guter Freund geworden.

Es bleiben viele liebe Kollegen und Studenten, von denen ich jedem einzelnen einen eigenen Abschnitt widmen könnte. Stellvertretend seien hier nur einige genannt: Volker Hartmann, Nicole Ruiter und Michael Beller als gute Kollegen und anregende Diskussionspartner, Michael Sutter und Hansjörg Neiber als Diplomanden mit vielen guten Ideen, die Mitarbeiter des Instituts für Prozeßdatenverarbeitung und Elektronik sowie nicht zuletzt mein Institutsleiter Prof. Hartmut Gemmeke, der sich sehr für mich eingesetzt hat.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Als Klassifikation bezeichnet man die eindeutige Zuordnung eines Musters zu einer bestimmten Klasse. Ein Klassifikationssystem ist eine abgeschlossene Einheit mit definierten Schnittstellen zu dem Zweck der Klassifikation. Über diese Schnittstellen wird als Reaktion auf ein als Eingabe vorgelegtes Muster eine Klassenzugehörigkeit ausgegeben. Klassifikationssysteme stellen insbesondere den Kern eines jeden computergestützten Diagnosesystems dar. Der Bedarf an diesen Systemen steigt ständig an, da sie im Gegensatz zu menschlichen Experten im allgemeinen reproduzierbare und objektive Entscheidungen treffen [Giger, 2000] [Dehning, 1995] [Comaniciu, 2000].

Es existiert kein universell zu verwendendes Klassifikationssystem [Jain, 2000]. Für jedes vorgelegte Klassifikationsproblem muß ein individuelles System entworfen werden. Abbildung 1 zeigt graphisch, wie bei Aufbau und Anwendung vorgegangen wird.

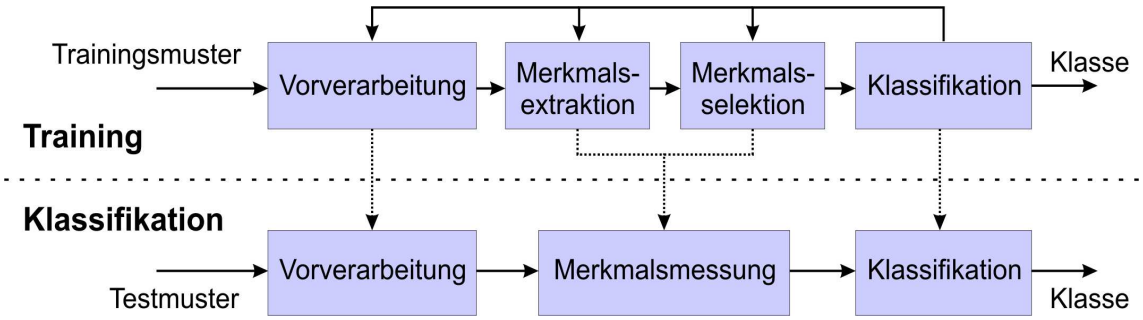


Abbildung 1. Überblick über Entwurf und Anwendung von Klassifikationssystemen nach [Jain, 2000]. ***Training:*** *mit Hilfe von vorklassifizierten Beispielen (Trainingsmustern) wird eine Prozeßkette von Vorverarbeitung bis Klassifikation aufgebaut. Die Prozeßkette wird solange optimiert, bis der erwartete Generalisierungsfehler minimal ist.* ***Klassifikation:*** *nach dem Entwurf werden Merkmalsextraktion und Merkmalsselektion zusammengefaßt. Mit der erstellten Prozeßkette können unklassifizierte Muster (Testmuster) einer Klasse zugeordnet werden.*

Beim Aufbau (Training) wird aus verschiedenen Algorithmen eine Prozeßkette erstellt. Ausgegangen wird hierbei von meist vorklassifizierten Beobachtungen, beispielsweise Bildern, die als Trainingsmuster bezeichnet werden. Die Bilder werden zunächst vorverarbeitet. Dies kann beispielsweise durch eine Segmentierung geschehen [Jähne, 1997]. Die Merkmalsextraktion ermittelt Eigenschaften einer Beobachtung ungeachtet deren Wichtigkeit [Lehmann, 1997]. Dahingegen filtert die Merkmalsselektion diejenigen Merkmale heraus, die eine Beobachtung besonders gut beschreiben. Bei der Klassifikation lernt ein Klassifikator, die Beobachtungen gemäß den Vorgaben richtig zu klassifizieren. Die

Güte einer solchen Prozeßkette wird durch ihren Generalisierungsfehler beschrieben [Witten, 2000]. Man variiert die Algorithmen der Prozeßkette solange, bis man mit dem Ergebnis zufrieden und der Generalisierungsfehler möglichst niedrig ist. Die Anwendung des resultierenden Klassifikationssystems besteht in der Klassifikation, d.h. in der Zuordnung von Testmustern unbekannter Klassenzugehörigkeit zu Klassen.

Das beschriebene Entwurfsschema ist in der Praxis unbefriedigend. Wird beispielsweise für die computergestützte Diagnose ein neuartiger Segmentierungsalgorithmus entwickelt, muß ein vollständig neues Klassifikationssystem aufgebaut werden. Zu der Implementierung und dem Test des Segmentierungsalgorithmus wird das bevorzugte Werkzeug des jeweiligen Entwicklers eingesetzt, beispielsweise Khoros [Alef, 2004]. Nach erfolgter Merkmalsextraktion wird die Entwicklung des Klassifikationssystems im allgemeinen mit Hilfe eines anderen Werkzeugs fortgesetzt, beispielsweise SAS [SAS, 2004]. Der Entwurf des Klassifikationssystems wird aufgrund des iterativen Vorgehens und des Wechsels zwischen den Werkzeugen zeitaufwendig und umständlich. Insbesondere stehen für die Merkmalsselektion und Klassifikation zahlreiche Algorithmen zur Verfügung, deren Kombination zu unterschiedlich guten Generalisierungsfehlern führt. Die Suche nach der besten Kombination ist ein NP-vollständiges Problem. In der Praxis werden meist einige Kombinationen ausprobiert und solange getestet, bis ein akzeptabler Generalisierungsfehler erreicht wurde. Einen niedrigen Generalisierungsfehler garantiert dieses Verfahren jedoch nicht.

Das bisherige Vorgehen bei dem Entwurf von Klassifikationssystemen ist nicht geeignet, um gute Ergebnisse in akzeptabler Zeit zu erzielen.

1.2 Lösungsansatz

Die Verbesserungsidee besteht darin, den bisher manuellen und konventionell programmierten Entwurfsprozeß gezielt zu unterstützen und zu automatisieren. Als Rahmen hierzu soll ein effizientes Entwicklungswerkzeug für die Klassifikation geschaffen werden, das einerseits Methoden für die Automatisierung bereit stellt und andererseits einen interaktiven manuellen Entwurf auf graphischer Basis ermöglicht.

Für die Automatisierung sind neuartige Methoden erforderlich, die effizient geeignete Merkmalsselektoren und Klassifikatoren mit guten Generalisierungsfehlern auswählen. Theoretisch kann jede

Kombination von Merkmalsselektoren und Klassifikatoren die optimale mit dem geringsten Generalisierungsfehler sein. Ein Ausprobieren aller Kombinationsmöglichkeiten ist aufgrund der enormen Komplexität für die praktische Anwendung ungeeignet. Die Automatisierung des Entwurfs kann daher nur durch Einschränkungen erreicht werden, mit denen die Komplexität begrenzt wird. Eine solche mögliche Einschränkung ist die Reduktion auf ein zweidimensionales Optimierungsproblem aus genau einem Merkmalsselektor und einem Klassifikator. Herkömmliche Methoden sind zu der Lösung dieses zweidimensionalen Optimierungsproblems ungeeignet. Aus diesem Grund müssen neue Lösungsansätze entwickelt werden.

Zu der Vereinfachung des manuellen Entwurfs muß das Entwicklungswerkzeug für die Klassifikation über eine effiziente, graphische Benutzerschnittstelle verfügen, mit der die benötigten Klassifikationsalgorithmen wie mit einem Baukastensystem schnell und interaktiv zu einer Prozeßkette zusammengesetzt werden können. Ideal geeignet hierfür ist eine Komponentensoftware, in der alle notwendigen Algorithmen als Komponenten integriert sind. In einer Komponentensoftware können Komponenten sehr schnell und interaktiv zusammengefügt werden.

Um die Methoden für die Automatisierung des Entwurfsprozesses zu verwirklichen, muß das Entwicklungswerkzeug zwingend über Möglichkeiten verfügen, mit deren Hilfe Algorithmen der Prozeßkette automatisch instantiiert, zusammengefügt und kontrolliert werden können. Solche Möglichkeiten können auf Basis einer Meta-Ebene realisiert werden. Für eine Komponentensoftware ist das *self-guided assembly* [Szyperski, 1999], das es Komponenten ermöglicht, ihrerseits andere Komponenten zu erzeugen und zu kontrollieren, besonders gut geeignet. Methoden für die Automatisierung des Entwurfs können dadurch als einfache Komponenten realisiert werden.

1.3 Gliederung der Arbeit

Es wäre vorteilhaft, wenn ein vorhandenes System in ein Entwicklungswerkzeug für Klassifikationssysteme umgewandelt werden könnte. Dazu werden in Kapitel 2 „Stand der Technik“ die konkreten Anforderungen an ein derartiges Entwicklungswerkzeug zusammengefaßt und verfügbare Systeme und Konzepte daraufhin untersucht. Es stellt sich leider heraus, daß keines die konkreten Anforderungen erfüllt, jedoch Teile dieser Systeme und Konzepte für die Implementierung eines solchen Entwicklungswerkzeuges genutzt werden können.

Zur Unterstützung des Entwurfsprozesses von Klassifikationssystemen ist es unumgänglich, sich näher mit den Algorithmen für Merkmalsselektion und Klassifikation zu befassen. Kapitel 3 „Grundlagen der Klassifikation“ stellt gängige Klassifikationsverfahren und deren Anwendung vor. Alle diese Verfahren wurden in dem implementierten Entwicklungswerkzeug integriert. Das Kapitel endet mit der Betrachtung verschiedener Schätzmethoden des Generalisierungsfehlers, da diese eine zentrale Rolle in der Rückkopplungsschleife des automatischen Entwurfs spielen.

Die Automatisierung des Entwurfsprozesses wird in Kapitel 4 „Automatische Synthese von Klassifikationssystemen“ beschrieben. In diesem Kapitel werden Methoden entwickelt, mit denen das nach der Beschränkung verbleibende Optimierungsproblem der Auswahl von Merkmalsselektoren und Klassifikatoren gelöst werden kann. Insbesondere wird auf die Entwicklung eines effizienten „Bubble-Search“ Algorithmus eingegangen, der auf zweidimensionale Suchprobleme beschränkt ist und mit dem in akzeptabler Zeit gute Klassifikationssysteme ermittelt werden können. Im Anschluß werden die verschiedenen Methoden bezüglich ihrer Leistungsfähigkeit beurteilt.

Die Implementierung der „Komponentensoftware für die Klassifikation“ ist in Kapitel 5 beschrieben. Dieses neue Java-Werkzeug konnte inzwischen in einer Reihe von ganz anderen Softwareanwendungen nicht nur für die Klassifikation erfolgreich genutzt werden. Mit der kurzen Vorstellung dieser Anwendungen und der Zusammenfassung endet das Kapitel.

Die entwickelte Komponentensoftware für die Klassifikation wurde im praktischen Einsatz getestet. Kapitel 6 „Ergebnisse automatisch erstellter Klassifikationssysteme“ präsentiert im ersten Abschnitt die systematische Untersuchung des Bubble-Search Algorithmus mittels künstlicher Klassifikationsprobleme. Im zweiten Abschnitt werden automatisch synthetisierte Klassifikationssysteme vorgestellt, die das Resultat realer Klassifikationsaufgaben sind.

Kapitel 7 „Diskussion und Ausblick“ diskutiert einerseits die entwickelten Methoden für die automatische Synthese von Klassifikationssystemen. Die erzielten Ergebnisse werden zusammengefaßt und beurteilt. Andererseits wird die neuartige Komponentensoftware für die Klassifikation beurteilt.

Im Anhang befinden sich Literatur- und Publikationsverzeichnis, Lebenslauf, eine Übersicht über implementierten Komponenten und Datentypen, eine Beschreibung der untersuchten Klassifikationsprobleme sowie wichtige Pseudo-Codes der entwickelten Methoden.

2 Stand der Technik

Der Aufbau von Klassifikationssystemen kann durch ein spezielles Entwicklungswerkzeug für die Klassifikation erheblich erleichtert und vereinfacht werden, wenn damit Prozeßketten gemäß Abbildung 1 schnell erstellt werden können und zudem Methoden für eine automatische Auswahl von Merkmalsselektoren und Klassifikatoren integriert sind. Es wäre besonders vorteilhaft, wenn ein vorhandenes System mit geringem Aufwand in ein solches Entwicklungswerkzeug umgewandelt werden könnte, so daß lediglich Methoden für eine automatische Auswahl entwickelt und implementiert werden müssen. In den folgenden Abschnitten werden zunächst die Anforderungen an ein derartiges Entwicklungswerkzeug zusammengetragen und im Anschluß daran die zur Zeit verfügbaren Systeme und Realisationsformen verglichen.

2.1 Anforderungen

Zunächst werden die zwingend notwendigen Anforderungen an ein Entwicklungswerkzeug für die Klassifikation vorgestellt. Im Anschluß daran werden einige wünschenswerte Eigenschaften aufgezählt, die zwar nicht notwendig sind, für eine Implementierung aber von Vorteil wären.

2.1.1 Notwendige Eigenschaften

Komponentensoftware: Als Komponentensoftware bezeichnet man ein Rahmenwerk, das Regeln und Schnittstellen für die Verwaltung von Komponenten zur Verfügung stellt und ihr Zusammensetzen ermöglicht [Szyperski, 1999]. Algorithmen können wie mit einem Baukastensystem rasch miteinander kombiniert werden. Eine Komponentensoftware mit einer graphischen Oberfläche für ein manuelles Eingreifen und die Ergebnisdarstellung ist daher ideal geeignet, schnell und interaktiv Prozeßketten wie in Abbildung 1 zu erstellen. Ein Entwicklungswerkzeug für die Klassifikation sollte daher eine Komponentensoftware sein.

Klassifikationsalgorithmen: Aus der Literatur sind zahlreiche Klassifikationsalgorithmen bekannt, die jeweils für bestimmte Klassifikationsaufgaben besonders gut geeignet sind. Eine eingeschränkte Anzahl von Klassifikationsalgorithmen schließt daher von vornherein potentielle Lösungen aus. Ein Entwicklungswerkzeug für die Klassifikation muß aus diesem Grund eine möglichst große Anzahl von Klassifikationsalgorithmen anbieten bzw. leicht um solche Algorithmen erweitert werden können.

Meta-Ebene: Der Entwurfsprozeß soll durch Methoden für die automatische Auswahl von Klassifikationsalgorithmen unterstützt werden. Hierzu ist eine Rückkopplung der Klassifikationsergebnisse in den Entwurfsprozeß erforderlich. Um eine Implementierung solcher automatisierenden Methoden zu ermöglichen, muß ein Entwicklungswerkzeug für die Klassifikation Kontrollfunktionen beispielsweise für die Rückkopplung auf einer übergeordneten Ebene zur Verfügung stellen.

Eine solche Meta-Ebene kann auf verschiedene Arten realisiert werden. Bereits eine Schnittstelle zu den internen Methoden eines Entwicklungswerkzeuges kann als Meta-Ebene bezeichnet werden, wenn sie die Möglichkeiten für Kontrolle und Rückkopplung des Entwurfsprozesses bietet. Weitere Möglichkeiten bestehen beispielsweise in einer integrierten Skript-Sprache oder der Möglichkeit von Makroaufzeichnungen. Beide Methoden kapseln den Zugriff auf die internen Methoden eines Entwicklungswerkzeuges, anstelle sie direkt zur Verfügung zu stellen. Ermöglicht man Komponenten innerhalb einer Komponentensoftware ihrerseits wieder andere Komponenten über definierte Schnittstellen zu erzeugen und zu kontrollieren, bezeichnet man dies als *self-guided assembly* [Szyperski, 1999]. Diese Form einer Meta-Ebene ist besonders gut geeignet, weil Methoden für die Entwurfsunterstützung dadurch als Komponenten implementiert werden können. Als Komponenten implementierte Methoden sind leichter zu erweitern oder auszutauschen als statisch in das Entwicklungswerkzeug integrierte.

2.1.2 Wünschenswerte Eigenschaften

Erweiterbarkeit: Zum Aufbau von Prozeßketten gemäß Abbildung 1 sollten viele verschiedene Algorithmen für die einzelnen Teilschritte zur Verfügung stehen. Die Integration bereits vorhandener Bibliotheken beispielsweise für die Bildverarbeitung sollte möglichst einfach sein, um den Implementierungsaufwand zu minimieren. Ein Entwicklungswerkzeug für die Klassifikation sollte sich daher besonders einfach erweitern lassen.

Plattformunabhängigkeit: Die Anwendung und Entwicklung von Klassifikationssystemen findet unter vielen verschiedenen Betriebs- und Rechensystemen statt. Das Entwicklungswerkzeug kann auf diesen verschiedenen Systemen leichter eingesetzt werden, wenn es plattformunabhängig ist. Eine Implementierung in Java oder einer vergleichbaren Programmiersprache ist daher wünschenswert.

Diese gewünschten Eigenschaften sind nicht zwingend notwendig für ein Entwicklungswerkzeug für die Klassifikation, jedoch wird die Handhabung und Umsetzung durch die Erfüllung dieser Anforderungen wesentlich vereinfacht.

2.2 Komponentensoftware

Zur Umsetzung eines Entwicklungswerkzeuges für die Klassifikation ist eine Komponentensoftware mit einer großen Anzahl von Klassifikationsalgorithmen und einer Meta-Ebene für die Automatisierung des Entwurfsprozesses erforderlich. Im diesem Abschnitt werden bekannte Implementierungen und Konzepte von Komponentensoftware vorgestellt und dahingehend bewertet, ob sie zu einem solchen Entwicklungswerkzeug erweitert werden können.

2.2.1 Implementierungen von Komponentensoftware

Es existieren zahlreiche konkrete Umsetzungen von Komponentensoftware. Im folgenden werden repräsentative Beispiele von Komponentensoftware vorgestellt, die einen Überblick über die Allgemeinheit geben.

Khoros [Alef, 2004] [Argiro, 2001], ist die wohl bekannteste Komponentensoftware vor allem für die Bildverarbeitung. Sie besteht aus einer graphischen Benutzeroberfläche und Entwicklungstools, mit der eigene Komponenten erstellt werden können. Anwender können mit Hilfe der Benutzeroberfläche Komponenten interaktiv instanziieren und diese mit Datenkanälen graphisch verbinden. Khoros stellt zu diesem Zeitpunkt etwa fünf verschiedene Algorithmen für die Klassifikation wie beispielsweise k-nearest-Neighbour zur Verfügung. Es beinhaltet keine Meta-Ebene und bietet daher keine Möglichkeit für ein self-guided assembly. Der Entwurf eigener Komponenten ist trotz der bereit gestellten Hilfsmittel komplex, weshalb eigene Algorithmen oder vorhandene Bibliotheken schwierig zu integrieren sind. Khoros wird mittlerweile nur noch als kommerzielles Produkt vertrieben. Der Quelltext steht nicht länger frei zur Verfügung und eine Modifikation der Quellen zu eigenen Zwecken ist daher ausgeschlossen.

LabView [LabView, 2004] ist eine weit verbreitete graphische Benutzerumgebung der Firma National Instruments. Es wird kommerziell vertrieben und dient zur Ansteuerung von Hardwarekomponenten für die Signalerfassung Es ist mittlerweile als ein de facto Standard für Slow-Control-Umgebungen anzusehen. Wie bei Khoros lassen sich Komponenten visuell miteinander verknüpfen. Es bietet

umfangreiche Hilfsmittel für den Entwurf eigener Applikationen. Die Bildverarbeitung und Klassifikation werden nur wenig unterstützt, da LabView auf die Ansteuerung und Auswertung von Hardware spezialisiert ist. Eine Meta-Ebene ist nicht vorhanden.

Eine große Zahl von Komponentensoftware ähnelt Khoros oder LabView, ist jedoch zunehmend als Speziallösung für bestimmte Aufgaben konzipiert. ***QuickCog*** [König, 1999] [König, 2003], ***MontiVision*** [MontiVision, 2003] und ***AdOculos*** [AdOculos, 2003] sind kommerzielle Lösungen vorwiegend für die industrielle Bildverarbeitung. Sie bieten einige eingeschränkte Klassifikationsmöglichkeiten und verfügen über keine Meta-Ebene, mit der eine übergeordnete Ablaufsteuerung möglich wäre.

BlackBox [Blackbox, 1997] [Oberon, 1994] und ***OpenDoc*** [OpenDoc, 1994] sind Komponenten-Frameworks auf der Basis von CORBA [Szyperski, 1999]. Beide sind vor allem für visuelle Komponenten konzipiert und definieren sehr komplexe Schnittstellen. ***Portos*** [Portos, 1997] ist eine Erweiterung von BlackBox, die auf Echtzeitanwendungen spezialisiert ist. ***MoMo*** [Heppner, 1997] ist ein verteiltes, objektorientiertes Framework für Realzeitanwendungen, entwickelt am Forschungszentrums Karlsruhe. ***SOFIA*** [Wenz, 2003] ist eine echtzeitfähige Komponentensoftware für mechatronische Systeme und wurde an der Universität Karlsruhe entwickelt. In diesen spezialisierten Systemen sind weder Komponenten für die Bildverarbeitung oder Klassifikation verfügbar, noch steht eine Meta-Ebene für eine automatische Rückkopplung bereit.

Generell verfügt keine der zur Zeit implementierten Komponentensoftware über eine verwendbare Meta-Ebene. Klassifikationsalgorithmen sind in diesen Systemen meist in nur geringer Anzahl integriert. MoMo ist als einziges (weitgehend) plattformunabhängig. In manchen Systemen lassen sich keine eigenen Komponenten integrieren und fast alle Systeme sind kommerziell. Programmtechnische Erweiterungen in dem Sinne, daß diese Systeme die Anforderungen erfüllen könnten, sind aufgrund des geschlossenen Quelltextes nicht möglich.

2.2.2 Realisationsformen von Komponentensoftware

Die Implementierung einer Komponentensoftware für die Klassifikation ist notwendig, wenn keine der verfügbaren Implementierungen von Komponentensoftware im Sinne der Anforderungen erweitert werden kann. In diesem Fall muß eines der folgenden Konzepte als Basis einer eigenen Implementierung verwendet werden.

SUN stellt mit den ***JavaBeans*** [Ullenboom, 2001] ein plattformunabhängiges Konzept für Softwarekomponenten in der Programmiersprache Java zur Verfügung. Solche Softwarekomponenten können mit graphischen Entwicklungswerkzeugen zu Applikationen zusammengesetzt werden [Flanagan, 1998]. Die Definition von JavaBeans erfolgt vor allem durch *Design Patterns.* Klassen können zur Laufzeit untersucht und ausgewertet werden, da dies in der Sprachspezifikation von Java integriert ist [Szyperski, 1999]. JavaBeans beschränken sich nicht nur auf visuelle Komponenten. Sie unterscheiden sich vor allem durch ihre feine Granularität von den ***Enterprise JavaBeans*** (***EJB***), die über eine wesentlich gröbere Granularität verfügen [Roman, 1999].

Die Konzepte ***OLE*** (Object Linking and Embedding), ***COM*** (Component Object Model) und ***DCOM*** (Distributed COM) wurden von der Firma Microsoft entwickelt. Sie definieren keinerlei Zugriffsvorschriften und beschreiben lediglich Schnittstellen [Szyperski, 1999]. Ebenfalls von Microsoft wurden auf Basis der Programmiersprache Visual Basic die Konzepte VBX (Visual Basic Extension), OCX (OLE Control Extension) und ***ActiveX*** entwickelt. ActiveX-Komponenten sind immer an eine graphische Oberfläche gebunden [Maurer, 2000]. Seit 2000 bietet Microsoft das Framework ***.NET*** an, mit dem komponentenorientierte Anwendungsentwicklungen unter Windows möglich ist. Bewährte Konzepte wie Just-In-Time Compiler, Garbage Collection oder Introspection wurden von Java übernommen. Die .NET Spezifikation ist allerdings viel umfangreicher als ein normales Komponentenkonzept, da es COM/DCOM ablösen und zur internen Neustrukturierung von Microsofts Betriebssystemen und Anwendungen eingesetzt werden soll [Schwichtenberg, 2001].

Das wohl am besten standardisierte Konzept ist die *Common Object Request Broker Architecture* (***CORBA***) [Szyperski, 1999]. Es wurde für die offene Kommunikation zwischen unzähligen Plattformen und Implementierungen entwickelt. Aus CORBA ist das *CORBA Component Model* (CCM) hervorgegangen, das zur Zeit das mächtigste und auch komplexeste Komponentenmodell darstellt [Staudacher, 2001], aber damit auch für eine effiziente Umsetzung zu komplex und zu laufzeitaufwendig ist.

2.3 Systeme für die Klassifikation

Möglicherweise kann ein verfügbares Hilfsmittel für die Klassifikation derart verändert werden, daß es die gestellten Anforderungen erfüllt. Die nachfolgenden Systeme werden häufig für den Entwurf von Klassifikationssystemen verwendet.

Das *Waikato Environment for Knowledge Analysis* (***Weka***) [Witten, 2000] ist eine Klassifikationsbibliothek mit graphischer Oberfläche, die über einige einfache Methoden für die graphische Programmierung verfügt. Sie ist jedoch keine Komponentensoftware im eigentlichen Sinn. Es basiert auf der Programmiersprache Java, wodurch es automatisch plattformunabhängig ist. Es ist frei erhältlich und hervorragend modular aufgebaut. Weka stellt von allen vorgestellten Systemen bei weitem die meisten und vielfältigsten Algorithmen für die Merkmalsselektion (ungefähr 100 durch Kombination verschiedener Verfahren) und Klassifikation (ungefähr 60) zur Verfügung. Eine Integration von Methoden zur Vorverarbeitung oder Merkmalsextraktion ist in Weka nicht möglich. Es stellt keine Meta-Ebene bereit, mit der eine automatische Rückkopplung des Klassifikationsentwurfes möglich wäre.

Yet Another Learning Environment (***YALE***) ist eine freie Entwicklungsumgebung für Maschinenlernen und Data Mining [Ritthoff, 2001] [YALE, 2003]. Seine Stärke liegt in der Merkmalsextraktion und Merkmalsselektion, die durch „Vorverarbeitungsketten“ in der Beschreibungssprache XML definiert werden können. Es basiert auf dem Weka-System und bindet dessen Algorithmen ein. YALE ist vollständig Java implementiert und plattformunabhängig. Es verfügt über keine Meta-Ebene.

SAS (Statistical Analysis Software) [SAS, 2004] ist ein kommerzielles Klassifikationswerkzeug der Firma SAS. Es stellt einige wenige Algorithmen für die Merkmalsselektion und Klassifikation bereit, ist jedoch weit verbreitet. Die Integration eigener Algorithmen ist nicht möglich. Der integrierte Script-Interpreter könnte unter Umständen als Meta-Ebene verwendet werden, die Rückkopplung des Entwurfsprozesses durch einen Script-Interpreter ist jedoch umständlich. Desweiteren können solche kommerziellen Produkte nicht an eigene Anforderungen angepaßt werden, da der Quelltext nicht verfügbar ist. ***Statistica*** ist ein Produkt der Firma StatSoft [Statistica, 2004] und stellte wie SAS keine geeignete Meta-Ebene bereit. Eigene Algorithmen können auch hier nicht integriert werden. ***S*** ist ein kommerzielles Paket für statistische Untersuchungen [S-System, 2004]. ***R*** ist eine freie Implementierung mit den Funktionalitäten von S [R-Project, 2004]. Beide sind für die Klassifikation nur bedingt geeignet, da ihr Schwerpunkt auf der statistischen Analyse liegt.

Einzig in ***Visual Apprentice*** [Jaimes, 2001] wird die dynamische Auswahl von Merkmalen und Klassifikatoren vorgeschlagen. Visual Apprentice ist ein Framework, daß den automatischen Entwurf von Klassifikatoren für Szenen/Objekten aus Bildern oder Videos erlaubt. Die Merkmalsselektion wird durch einen einzigen Algorithmus vorgenommen und es werden lediglich vier verschiedene Klassifikatoren verwendet. Die Auswahl des am besten geeigneten Klassifikationssystems wird durch simples Ausprobieren aller möglichen Kombinationen (in diesem Fall vier) ermittelt. Diese Methode ist für

eine höhere Anzahl aufgrund des rasch ansteigenden Aufwandes ungeeignet. Das Framework ist nur kommerziell erhältlich und der Quelltext ist nicht verfügbar.

2.4 Zusammenfassung

Es existiert eine große Anzahl unterschiedlicher Komponentensoftware. Keines dieser Systeme verfügt über eine nutzbare Meta-Ebene, mit der eine automatische Rückkopplung des Entwurfsprozesses möglich wäre. Die meisten Systeme verfügen über nur sehr wenige Klassifikationsalgorithmen. Die Quelltexte der kommerziellen Produkte sind nicht zugänglich und eine Erweiterung im Sinne der Anforderungen ist in diesem Falle unmöglich.

Die vorhandenen Werkzeuge für die Klassifikation können nicht für eine Vorverarbeitung und Merkmalsextraktion erweitert werden. Ebenso wie bei der Komponentensoftware ist in keinem der Systeme eine Meta-Ebene implementiert. Bei den kommerziellen Produkten ist eine Modifikation des Systems zur Erfüllung der Anforderungen ausgeschlossen. Einzig das System Weka stellt eine große Anzahl von Klassifikationsalgorithmen zur Verfügung, die im Rahmen dieser Arbeit auch genutzt wurden.

Keines der vorgestellten Systeme kann zu einem Entwicklungswerkzeug für die Klassifikation umgewandelt werden. Als Konsequenz ergibt sich die Notwendigkeit eines eigenen, neuartigen Ansatzes. Dieser Ansatz, basierend auf dem System Weka und dem Konzept JavaBeans, wird in Kapitel 5 „Komponentensoftware für die Klassifikation“ behandelt.

3 Grundlagen der Klassifikation

> ***Klassifikation:*** *„[lateinisch] die, Einteilung von Dingen oder Begriffen nach gemeinsamen Merkmalen, z.B. Dezimalklassifikation, Schiffsklassifikation; Klassifikation der Lebewesen, Systematik.*
>
> ***Bibliographisches Institut & F. A. Brockhaus AG, 2001***

Als ***Klassifikation*** bezeichnet man die Zuordnung einer ***Beobachtung*** zu einer Gruppe mit gemeinsamen Eigenschaften. Die Gruppe wird als ***Klasse*** bezeichnet [Brockhaus, 2004]. Beobachtungen können sowohl konkrete als auch abstrakte Ausprägungen annehmen und gehören immer nur genau einer Klasse an. Dadurch sind eindeutige Entscheidungen über eine Klassenzugehörigkeit möglich. Ein System, welches Beobachtungen bestimmten Klassen zuordnet, nennt man ***Klassifikationssystem***. Ein Klassifikationssystem besteht aus einer Sequenz von Algorithmen. Der zentrale Algorithmus, der die Zuordnung von Beobachtungen zu Klassen vornimmt, ist der ***Klassifikator***.

Bevor ein Klassifikationssystem Beobachtungen bestimmten Klassen zuordnen kann, sind geeignete Vorbereitungen durchzuführen. Zum einen müssen die möglichen Klassen bekannt sein, die für eine Zuordnung zur Verfügung stehen. Andererseits sind charakteristische Informationen notwendig, wodurch sich die Einordnung einer Beobachtung in eine bestimmte Klasse von der in eine andere Klasse unterscheidet. Menschen lernen diese Information anhand von Beispielen. Ebenso müssen Klassifikationssysteme zur Initialisierung mit repräsentativen Beispielen des Klassifikationsproblems trainiert werden, wobei ein Beispiel aus einer Beobachtung mit zugeordneter Klasse besteht. Diese Beispiele bezeichnet man als ***Muster***. Die Mustermenge ist eine Teilmenge aller möglichen Ausprägungen und wird als ***Stichprobe*** bezeichnet. Je besser und größer die Stichprobe ist, desto erfolgversprechender ist das Training. Je repräsentativer die Stichprobe ist, desto generischer ist die spätere Klassifikation von unbekannten Beobachtungen.

3.1 Überwachtes und unüberwachtes Lernen

Bei dem Training eines Klassifikators unterscheidet man zwei grundsätzlich verschiedene Trainingsmethoden:

- Im ersten Fall verfügt man über keinerlei Vorwissen über die Klassenzugehörigkeit der Beobachtungen. Das Ziel ist es, Anhäufungen der Daten im beobachteten Raum zu identifizieren und die Beobachtungen sogenannten ***Ballungen,*** beziehungsweise ***Clustern,*** zuzuordnen. Man bezeichnet dieses Vorgehen als ***unüberwachtes Lernen, Ballungsanalyse*** oder ***Clustering*** [Michie, 1994]. Anschließend können weitere Beobachtungen den festgelegten Clustern zugeordnet werden.

- Im zweiten Fall ist die Klassenzugehörigkeit der Beobachtungen, die als Lernbeispiele dienen, bekannt. Die Aufgabenstellung besteht darin, folgende Beobachtungen in eine der explizit gegebenen Klassen einzuteilen [Noltemeier, 1976]. Die Klassen müssen derart zugeordnet werden, daß ein möglichst geringer Klassifikationsfehler entsteht. Die Zuordnungsvorschrift wird während des Trainings aus den Mustern bestimmt. Man spricht bei Beobachtungen mit vorgegebenen Klassen von ***überwachtem Lernen***. Im Allgemeinen wird in der Literatur die Klassifikation mit überwachtem Lernen als ***Diskriminanzanalyse*** bezeichnet [Michie, 1994].

Im folgenden wird immer von einer Klassifikation mit überwachtem Lernen ausgegangen. Hierzu werden Beobachtungen mit bekannter Klassenzugehörigkeit vorausgesetzt. Die Vorausklassifikation geschieht beispielsweise durch einen Experten für das jeweilige Problem, der mit seinem Fachwissen alle Beobachtungen ihrer jeweiligen Klasse zuordnet. Die entwickelten Methoden zur Automatisierung des Klassifikationsentwurfs lassen sich aber ohne jede Einschränkung auf Klassifikation mit unüberwachtem Lernen übertragen, indem die Klassen zunächst mittels Ballungsanalyse bestimmt werden. Die Daten werden danach behandelt, als wären sie von einem Experten vorklassifiziert worden.

Die Vorausklassifikation wirft die Frage auf, wozu eine maschinelle Klassifikation benötigt wird, wenn doch jemand grundsätzlich in der Lage ist, die Zuordnung vorzunehmen. Für eine maschinelle Klassifikation gibt es jedoch gute Gründe:

- Höhere Geschwindigkeit, um schnelle Entscheidungen zu fällen:
  Ein Mensch ist unter Umständen zu langsam, um aus Meßwerten schnell notwendige Handlungen abzuleiten. Eine Maschine kann entsprechende Daten um ein Vielfaches schneller verarbeiten.

- Unvoreingenommenheit und Reproduzierbarkeit:
  Ein wichtiges Kriterium für eine Entscheidung ist die Unabhängigkeit der entscheidenden Instanz von den Eingangsdaten. Menschen können sich aber von unwichtigen Informationen oder persönlichen Einschätzungen zu falschen Beurteilungen verleiten lassen.

- Breite Verfügbarkeit:
  Klassifikationssysteme können kostengünstig dupliziert und verteilt werden. Die Ausbildung eines Experten hingegen ist zeitaufwendig und teuer.

3.2 Merkmale und Merkmalsextraktion

Eine Beobachtung setzt sich aus vielen einzelnen, gemessenen oder berechneten Werten zusammen. Diese Werte bezeichnet man als ***Merkmale***. Die Merkmale werden zu einem ***Merkmalsvektor*** zusammengefaßt. Eine Anzahl *m* extrahierter Merkmale spannt einen Vektorraum auf, den man als ***Merkmalsraum*** bezeichnet. Eine Beobachtung nimmt genau einen Punkt im Merkmalsraum ein. Merkmale treten in zwei verschiedenen Ausprägungen auf: numerisch oder nominal. Numerische Merkmale verfügen über einen kontinuierlichen Wertebereich, während für einen diskreten Wertebereich mit einer geringen Anzahl an Ausprägungen nominale Merkmale verwendet werden. In der Praxis bildet man häufig nominale Werte auf numerische Werte ab, um die Verarbeitung zu erleichtern. Ein naheliegendes, bijektives Abbildungsverfahren ist die Aufzählung aller nominalen Werte. Häufig ist die Klassenzugehörigkeit von Beobachtungen in Form eines nominalen Merkmals gegeben und wird dem Merkmalsvektor hinzugefügt.

Als Eingangsdaten für eine Klassifikation dienen Beobachtungen aufgrund zumeist physikalisch gemessener Werte. Für eine medizinische Anwendung erfolgt die Messung beispielsweise durch ein bildgebendes Verfahren. Das aufgenommene Bild enthält die Informationen, die für eine Diagnose verwendet werden können. Für ein Klassifikationssystem muß die Information in eine verarbeitbare Form transformiert werden. Diesen Vorgang bezeichnet man als ***Merkmalsextraktion***. Die Schwierigkeit der Merkmalsextraktion liegt darin, die semantische Bedeutung der Eingangsdaten als erfaßbare Skalen abzubilden. Prinzipiell bestehen zwei mögliche Vorgehensweisen:

- Ein Experte auf dem entsprechenden Gebiet weiß ganz genau, durch welche spezifische Information sich das gestellte Problem lösen läßt. Hierzu muß die intuitive, visuelle Auffassung des

Experten in einen ebenso leistungsfähigen Algorithmus umgesetzt werden. Der Experte muß seine Analysemethoden sehr gut beschreiben können. Allgemeine Merkmale wie beispielsweise der Kontrast eines Bildes lassen sich einfach mit Standardalgorithmen erfassen.

- In manchen Fällen kann der Experte seine Intuition und sein Expertenwissen nicht in Form von eindeutigen Merkmalen beschreiben oder es ist sogar kein Experte für dieses Klassifikationsproblem verfügbar. In diesem Fall extrahiert man alle berechenbaren und irgendwie sinnvoll erscheinenden Merkmale. Dabei hofft man, daß in den Merkmalen genügend Information enthalten ist, um das Problem zu lösen. Die derart erstellten Muster enthalten oftmals Redundanz oder für die Klassifikation überflüssige Daten. Dies wirkt sich negativ auf die Qualität des Klassifikationssystems aus. Es empfiehlt sich grundsätzlich immer, die wirklich brauchbaren Merkmale vor der Klassifikation durch Algorithmen zu selektieren und das Ergebnis gemäß Abbildung 1 in die Merkmalsextraktion zurückzuführen. Das Vorgehen bei einer Merkmalsselektion wird in Kapitel 3.5 vorgestellt.

Als Muster wird im folgenden immer ein Merkmalsvektor mit bekannter Einordnung in eine Klasse bezeichnet. Transponiert man die Merkmalsvektoren aller Muster und ordnet sie untereinander an, erhält man eine Tabelle. Jede Zeile nimmt genau ein Muster auf und jede Spalte enthält die Werte eines Merkmals für die verschiedenen Muster. Ist für eine Beobachtung die Klasse unbekannt, bleibt das Feld in der Klassenspalte leer. Die Darstellung in Form einer Tabelle ist naheliegend und daher weit verbreitet.

3.3 Klassifikatoren

Im Idealfall gruppieren sich alle Beobachtungen einer Klasse zu einer Punktwolke und die Punktwolken verschiedener Klassen überschneiden sich nicht. In diesem Fall können alle Klassen eindeutig voneinander getrennt werden. Intuitiv wird man eine Beobachtung mit unbekannter Klasse derjenigen Klasse zuordnen, deren Punktwolke ihr am nächsten liegt. Abbildung 2 zeigt eine zweidimensionale Darstellung der Punktwolken eines realen Klassifikationsproblems. Es ist das Iris-Klassifikationsproblem und wird weitläufig zu Testzwecken verwendet [Fisher, 1936]. Die Aufgabe besteht darin, verschiedene Iris-Arten anhand der Blütenblattlänge und -breite sowie der Kelchblattlänge und -breite zu unterscheiden. Die Muster setzen sich aus den vier Merkmalen und der zugehörigen Klasse zusammen. Zum besseren Verständnis wurde der Merkmalsraum von vier auf zwei Dimensionen projiziert. Die Farbe eines Punktes gibt die Klassenzugehörigkeit der entsprechenden Beobachtung wieder.

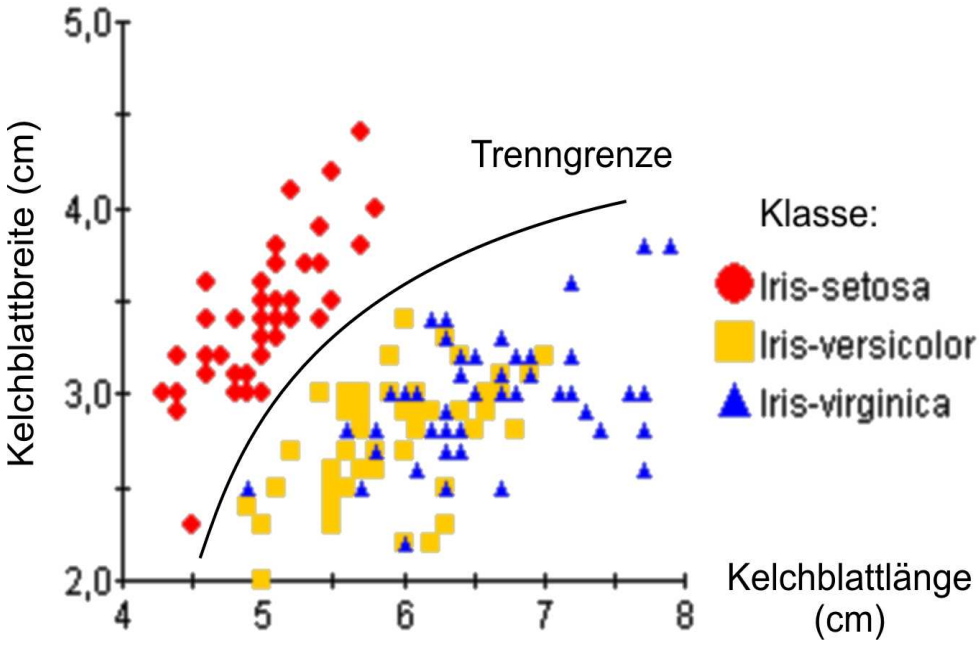


Abbildung 2. Visualisierung des Iris-Klassifikationsproblems. Das Problem besteht aus 150 Mustern. Jedes Muster gehört genau zu einer der drei möglichen Klassen und wird als entsprechend eingefärbter Punkt in einer Ebene dargestellt. Idealerweise gruppieren sich die Muster zu eindeutig separierten Punktwolken. Im Beispiel kann die Klasse Iris-setosa gut von den anderen Klassen getrennt werden. Die anderen beiden Klassen lassen sich in der Projektion nicht durch eine Gerade voneinander trennen.

Die Klasse Iris-setosa (rote Kreise) bildet eine gut von den anderen Klassen abgegrenzte Punktwolke. Sie läßt sich intuitiv durch eine einzige Trenngrenze von den anderen Punktwolken separieren. Beobachtungen mit unbekannter Klasse, die oberhalb der Trenngrenze liegen, würden der Klasse Iris-setosa zugeordnet werden. Die beiden übrigen Klassen lassen sich aufgrund der Überlappung nicht durch eine einzige Trenngrenze voneinander trennen.

Ein Klassifikator bildet den (m)-dimensionalen Merkmalsraum in Form eines Merkmalsvektors X durch eine interne Funktion f auf einen eindimensionalen Klassenraum ab:

$$f(X) \rightarrow c \qquad c \in \mathbb{N} \ \wedge \ X \in \mathbb{R}^m \qquad \textbf{Gl. 1}$$

wobei c den Wertebereich einer Aufzählung aller Klassen annimmt. Bei dem Training des Klassifikators wird implizit die Abbildungsfunktion f in Gleichung 1 bestimmt. Im folgenden wird die Funktionsweise einiger wichtiger Klassifikatortypen beschrieben.

Fisher-Diskriminanzanalyse

Wie in Abbildung 2 angedeutet, können zwei Klassen in einem zweidimensionalen Raum durch eine eindimensionale lineare Funktion getrennt werden. Wenn sich die Punktwolken überlappen, werden einige Muster falsch klassifiziert. Allgemein benötigt man für einen m-dimensionalen Merkmalsraum eine (m-1)-dimensionale Hyperebene. Eine empirische Methode, solche Hyperebenen zu bestimmen, ist die ***Fisher-Diskriminanzanalyse*** [Fisher, 1936]. Sie wurde 1936 von R. A. Fisher beschrieben und beruht ausschließlich auf der Auswertung gemessener Merkmale. Die allgemeine Form einer linearen Diskriminante für einen Merkmalsvektor X ist durch die Koeffizienten $a_0, a_1, \cdots, a_m$ gegeben [Nilsson, 1990]:

Gl. 2
$$f(X) = a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_m \cdot x_m - a_0$$

Anschaulich entspricht dies einer gewichteten Summe aller Merkmale. Jeder einzelnen Klasse wird somit für jede Beobachtung ein Wert zugeordnet. Man klassifiziert eine Beobachtung zugunsten der Klasse mit dem höchsten Wert.

Bayes-Klassifikator

In der Praxis treten von n verschiedenen Klassen eines Klassifikationsproblems selten alle Klassen gleichermaßen häufig auf. Die Verteilung einer Klasse ω_i ohne Berücksichtigung eventueller Merkmale wird durch die ***a-priori*** Wahrscheinlichkeit $P(\omega_i)$ beschrieben:

Gl. 3
$$P(\omega_i) = \frac{Anzahl\ Muster\ mit\ Klasse\ \boldsymbol{i}}{Anzahl\ aller\ Muster}$$

Es ist möglich, eine Klassifikation nur aufgrund der a-priori Wahrscheinlichkeit durchzuführen. Jede Beobachtung wird genau der Klasse zugeordnet, deren $P(\omega_i)$ am größten ist. Die Methode bezeichnet man als ***no-Data*** Verfahren. Trotz einer möglicherweise extrem niedrigen Fehlerrate ist das Verfahren zumeist völlig ungeeignet, da keine Gewichtung der Klassen vorgenommen wird.

Liegt eine Beobachtung mit Merkmalsvektor X vor, ist eine Entscheidung einzig aufgrund der a-priori Wahrscheinlichkeit keine gute Lösung. Die gemessenen Merkmale müssen in die Entscheidungsfindung mit einbezogen werden. Die Wahrscheinlichkeit für das Auftreten einer Klasse ω_i unter der Voraussetzung, daß der Merkmalsvektor X gemessen wurde, bezeichnet man als a-posteriori Wahr-

scheinlichkeit $P(\omega_i \mid X)$. Unter Berücksichtigung des Merkmalsvektors *X* ordnet man einer Beobachtung diejenige Klasse ω_i zu, welche über die größte a-posteriori Wahrscheinlichkeit verfügt [Fukunaga, 1990]:

$$P(\omega_i \mid X) > P(\omega_j \mid X) \quad \forall\, j \neq i \qquad \textbf{Gl. 4}$$

Gleichung 4 ist als Bayes-Regel für den minimalen Fehler bekannt und führt per Definition zur besten Klassifikation mit dem geringsten Fehler. Die Wahrscheinlichkeiten können mit Kostenfaktoren gewichtet werden, wenn beispielsweise der Fehlklassifikation einer bestimmten Klasse besondere Bedeutung zufällt. Die Schwierigkeit dieses Verfahrens besteht in der Bestimmung der a-posteriori Wahrscheinlichkeiten. Steht eine große Menge von Mustern zur Verfügung, bestimmt man die Wahrscheinlichkeiten am einfachsten durch eine look-up-table, in der alle möglichen Kombinationen aus Beobachtungen und zugehörigen Klassen vermerkt sind. Im Allgemeinen sprengt die dabei zu verwaltende Datenmenge aller möglichen Kombinationen allerdings den Rahmen eines jeden Rechensystems [Weiss, 1991]. Die a-posteriori Wahrscheinlichkeiten können aber durch die a-priori und die bedingten Wahrscheinlichkeiten bestimmt werden. Nach dem Satz von Bayes über die bedingten Wahrscheinlichkeiten gilt [Witten, 2000]:

$$P(\omega_i \mid X) = \frac{P(X \mid \omega_i) \cdot P(\omega_i)}{P(X)} \qquad \textbf{Gl. 5}$$

Gleichung 4 läßt sich mittels Gleichung 5 in folgenden Ausdruck überführen:

$$P(X \mid \omega_i) \cdot P(\omega_i) > P(X \mid \omega_j) \cdot P(\omega_j) \quad \forall\, j \neq i \qquad \textbf{Gl. 6}$$

Zur Berechnung der Bayes-Regel für den minimalen Fehler und somit dem bestmöglichen Klassifikator müssen die a-priori Wahrscheinlichkeiten $P(\omega_i)$ und die klassenbedingten Wahrscheinlichkeitsdichten $P(X \mid \omega_i)$ geschätzt werden. Die a-priori Wahrscheinlichkeiten lassen sich leicht aus dem Anteil einer Klasse an der Gesamtmenge aller Muster schätzen. In der Praxis ist die Schätzung der klassenbedingten Wahrscheinlichkeitsdichten schwierig. Zur Vereinfachung wird häufig die Unabhängigkeit der Merkmale vorausgesetzt, obwohl dies in den meisten Fällen nicht gegeben ist. Dennoch führt die Annahme häufig zu sehr guten Ergebnissen [Witten, 2000].

Ein weit verbreiteter Ansatz, die klassenbedingten Wahrscheinlichkeitsdichten $P(X \mid \omega_i)$ in Gleichung 6 zu bestimmen, liegt in der Annahme bestimmter Wahrscheinlichkeitsdichten und der anschließenden Schätzung ihrer Parameter aus der Stichprobe. Meist wird eine Normalverteilung der Klassen angenommen. Eine multivariate Normalverteilung wird durch ihren Mittelwertsvektor *E* und ihre Kovarianzmatrix Σ bestimmt:

Gl. 7
$$N(X) = \frac{1}{\sqrt{(2\pi)^m \,|\Sigma|}}\, e^{-\frac{1}{2}(X-E)^T \Sigma^{-1}(X-E)} \qquad X \in \mathbb{R}^m$$

Unter der oben genannten Annahme ergibt sich für die bedingte Wahrscheinlichkeit einer Klasse ω_i:

Gl. 8
$$P(X \mid \omega_i) = \frac{1}{\sqrt{(2\pi)^m \,|\Sigma_i|}}\, e^{-\frac{1}{2}(X-E_i)^T \Sigma_i^{-1}(X-E_i)}$$

Mit Hilfe von Gleichung 8 wird ein Muster derjenigen Klasse ω_i zugeordnet, für die gilt:

Gl. 9
$$\frac{P(\omega_i)}{\sqrt{|\Sigma_i|}}\, e^{-\frac{1}{2}(X-E_i)^T \Sigma_i^{-1}(X-E_i)} > \frac{P(\omega_j)}{\sqrt{|\Sigma_j|}}\, e^{-\frac{1}{2}(X-E_j)^T \Sigma_j^{-1}(X-E_j)} \qquad \forall\, j \neq i$$

Für eine Klassifikation nach Gleichung 9 müssen die Kovarianzmatrizen und Mittelwertsvektoren aus der Stichprobe bestimmt werden. Da für diese Methode die Parameter der Wahrscheinlichkeitsdichten geschätzt werden, bezeichnet man sie als ***parametrische Klassifikation*** [Fukunaga, 1990].

Aus Gleichung 9 ergibt sich nach Gleichsetzen und Auflösen ein Polynom zweiten Grades als optimale Trennfunktion zwischen zwei Klassen. Sind die Kovarianzmatrizen beider Normalverteilungen gleich, reduziert sich die Trennfunktion auf eine lineare Funktion. Abbildung 3 zeigt graphisch die optimalen Trenngrenzen normalverteilter Klassen. Für eine lineare Trennfunktion müssen $O(m)$ Parameter pro Klasse bestimmt und gespeichert werden. Für eine quadratische Trennfunktion steigt der Aufwand auf $O(m^2)$. Entsprechend mehr Beispiele müssen zur Verfügung stehen, um die quadratische Trennfunktion ausreichend gut bestimmen zu können.

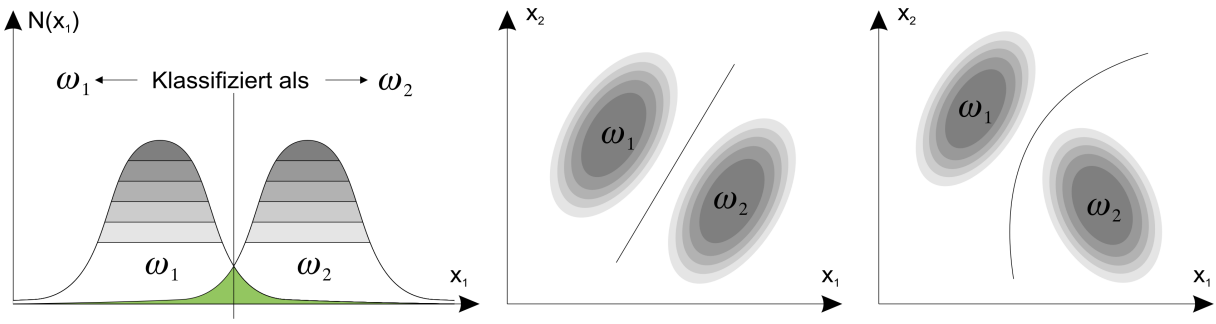


Abbildung 3. Links: Die eindimensionalen Wahrscheinlichkeitsdichten zweier normalverteilter Klassen mit gleicher Varianz der Klassen ω_1 und ω_2. Die gemeinsam überdeckte Fläche (grün) entspricht dem Bayes-Fehler. Der geringste Fehler wird erreicht, wenn die Trenngrenze in den Schnittpunkt beider Kurven gelegt wird. Mitte: Zweidimensional normalverteilte Wahrscheinlichkeitsdichten mit gleicher Varianz der Klassen in Höhenliniendarstellung. Die Trenngrenze wird durch eine Gerade gebildet. Rechts: Zweidimensional normalverteilte Klassendichten mit unterschiedlicher Varianz der Klassen. Die Trenngrenze wird durch ein Polynom zweiten Grades bestimmt.

Nicht quadratisch separierbare Probleme können durch komplexere Funktionen getrennt werden. Der Einsatz von Polynomen mit beliebigem Grad als Separatoren ist in [Schürmann, 1977] beschrieben. Eine andere häufig angewandte Technik ist die Approximation nichtlinearer Trennfunktionen durch stückweise lineare Teilfunktionen. Klassifikationsprobleme mit mehr als zwei Klassen werden häufig gelöst, indem man eine Klasse abspaltet und die restlichen Klassen zu einer einzigen zusammenfaßt. Das konstruierte Zwei-Klassen-Problem wird wie beschrieben gelöst. Von den zusammengefaßten Klassen wird wiederum eine abgespalten. Dies wird wiederholt, bis nur noch eine Klasse übrig ist. Die endgültige Lösung des Problems besteht in der seriellen Kombination der Trennfunktionen.

K-nearest-Neighbour

Betrachtet man die Punktwolken der Muster aus Abbildung 4 und fügt eine neue Beobachtung ein, würde man ihr intuitiv die Klasse der Punktwolke zuordnen, der sie am nächsten liegt [Lehmann, 1997]. Wenn eine Beobachtung zwischen zwei Punktwolken liegt oder gleichzeitig in zwei Wolken, ist die Vorschrift „am nächsten“ nicht präzise genug. Eine Beobachtung ist einer Punktwolke genau dann nah, wenn viele Mitglieder der Punktwolke in ihrer Umgebung liegen. Einer neuen Beobachtung wird daher genau die Klasse zugeordnet, der die meisten ihrer *k* nächsten Nachbarn angehören. Abbildung 4 zeigt graphisch das Einfügen einer Beobachtung in das Iris-Klassifikationsproblem unter Berücksichtigung der drei nächsten Nachbarn. Klassifikatoren, die Klassenzuordnungen aufgrund von Nachbarschaftsbeziehungen treffen, bezeichnet man als k-nearest-Neighbour Klassifikatoren. Zur

Bestimmung der k nächsten Nachbarn ist ein Abstandsmaß nötig. Am häufigsten werden entweder die euklidische Distanz

Gl. 10
$$D_E^2(X,Y) = (X-Y)^2 = \sum_{i=1}^{m} (x_i - y_i)^2$$

oder die Mahalanobis-Distanz verwendet:

Gl. 11
$$D_M^2(X,Y) = (X-Y)^T \cdot \Sigma^{-1} \cdot (X-Y) = \sum_{i=1}^{m}\sum_{j=1}^{m} (x_i - y_i) \cdot \sigma_{ij} \cdot (x_j - y_j)$$

X und Y bezeichnen die Merkmalsvektoren zweier Muster, Σ bezeichnet die Kovarianzmatrix der Merkmale und m ist die Anzahl der Merkmale. Die Mahalanobis-Distanz entspricht einer mit der Kovarianzmatrix normierten Euklidnorm. Für normierte, unkorrelierte Merkmale ist die Kovarianzmatrix die Einheitsmatrix und die Mahalanobis-Distanz geht in die Euklidnorm über [Micheli-Tzanakou, 2000]. Der Term auf der rechten Seite findet sich im Exponenten aus Gleichung 9 wieder und der dazugehörige Klassifikator wird daher auch als Mahalanobis-Distanz-Klassifikator bezeichnet.

Wird eine neue Beobachtung eingefügt, muß zu allen vorhandenen Mustern der Abstand berechnet werden. Bei einer naiven Implementierung verursacht die Berechnung des Abstandes einen Aufwand $O(m)$ linear zur Anzahl m der Muster. Die k geringsten Abstände werden durch die Sortierung bestimmt, die zusätzlich für jedes Muster einen Aufwand $O(\log(k))$ verursacht. Der Gesamtaufwand einer Klassifikation beläuft sich somit auf $O(m \cdot \log(k))$. In realen Anwendungen mit festgelegtem k ist der Aufwand jedoch nahezu linear. Die Berechnung des Abstandes zusammen mit dem Speicheraufwand sorgt für einen hohen konstanten Aufwand, der in die Abschätzung der Komplexität nicht eingeht. Es existieren verschiedene spezialisierte Implementierungen, welche den benötigten Aufwand reduzieren können [Arya, 1994]. Vergleichbare sind in der eingesetzten Klassifikationsbibliothek Weka bislang nicht implementiert.

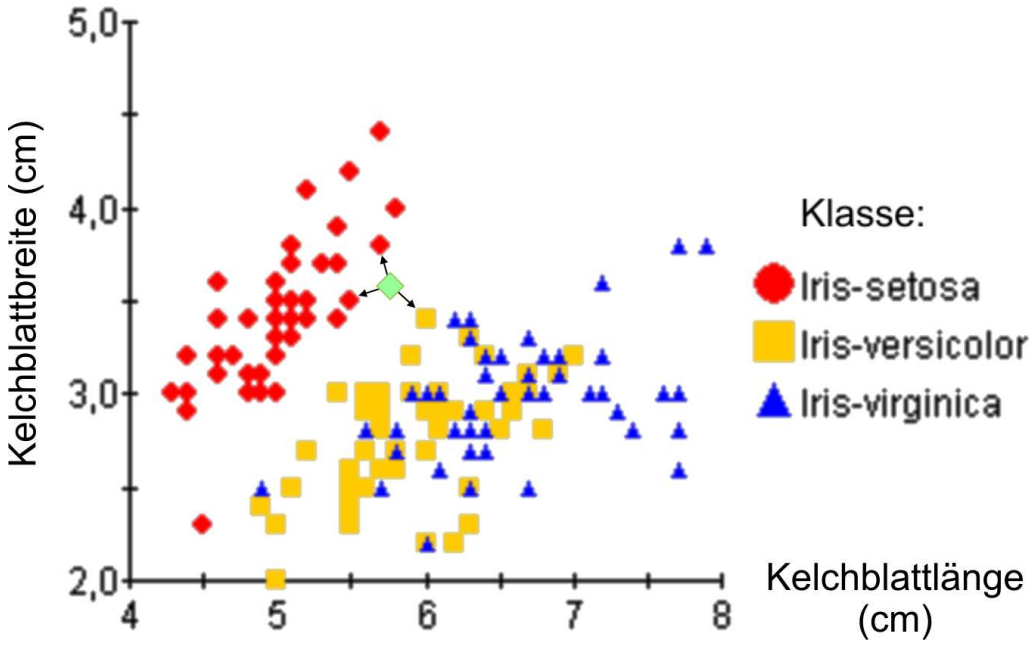


Abbildung 4. Bestimmung der Klassenzugehörigkeit durch einen 3-nearest-Neighbour Klassifikator. Für die neue Beobachtung (grüne Raute) finden sich zwei Nachbarn mit der Klasse Iris-setosa und ein Nachbar mit der Klasse Iris-versicolor. Der Beobachtung wird daher die Klasse Iris-setosa zugeordnet.

Weitere Klassifikationsalgorithmen

Es existieren zahlreiche weitere Klassifikationsalgorithmen, die in der Komponentensoftware für die Klassifikation implementiert wurden wie beispielsweise Perzeptrons [McCulloch, 1943] [Rosenblatt, 1962], Neuronale Netze [Weiss, 1991], Support Vector Machines [Witten, 2000] [Ziegenmeyer, 2003], Regelbasiert [Witten, 2000] wie PRISM oder Entscheidungsbäume [Michie, 1994] wie C4.5 und ID3. An dieser Stelle sei auf die angegebene Literatur verwiesen, da eine detaillierte Erläuterung der Algorithmen den Rahmen dieser Arbeit sprengen würde.

3.4 Fehlerabschätzung von Klassifikationssystemen

Die verschiedenen Klassifikatoren verwenden unterschiedliche Methoden für die Klassifikation und können deshalb auch unterschiedliche Ergebnisse liefern. Nach dem Aufbau eines Klassifikationssystems stellt sich daher die Frage nach dessen Qualität. Selbstverständlich strebt man das bestmögliche System an. Manchmal genügen allerdings auch annähernd optimale Ergebnisse, wenn man im Gegenzug dafür eine Reduktion an Rechenzeit und Aufwand gewinnt. Für eine Beurteilung des synthetisierten Klassifikationssystems soll im folgenden nur die Trennfähigkeit der Klassifikatoren beurteilt wer-

den. Eigenschaften wie Zeitaufwand für Training und Klassifikation oder Speicherbedarf sollen unberücksichtigt bleiben. Erstellte Klassifikationssysteme können somit eingeordnet und gegeneinander abgeschätzt werden. Für die Beurteilung eines Klassifikationssystems sollen zunächst zwei weitverbreitete Mißverständnisse richtig gestellt werden:

- Die Qualität eines Klassifikators mißt sich keineswegs alleine darin, wie gut er die zum Training verwendeten Muster zu trennen vermag. Im einfachsten Fall wird bei der Synthese eine Tabelle zum Nachschlagen (look-up-table) angelegt, in welcher die Ergebnisse gespeichert bleiben. Dies entspricht einem Auswendiglernen der Stichprobe. Damit lassen sich alle Muster in schnellstmöglicher Zeit korrekt klassifizieren. Mit einem Klassifikator sollen jedoch neue, bislang unbekannte Beobachtungen einer Klasse zugeordnet werden. Ein Gütekriterium für einen Klassifikator muß vorwiegend das Verhalten auf unbekannten Daten berücksichtigen.

- Die Güte eines Klassifikators ist immer von dem aktuell gestellten Problem abhängig. Es gibt keine absolute Reihenfolge in der Qualität von Klassifikatoren, welche einmalig berechnet werden kann. Für jede neue Stichprobe muß die Qualität neu ermittelt werden, um den besten Klassifikator für das gestellte Klassifikationsproblem zu bestimmen.

Damit ein Klassifikator mit unbekannten Daten getestet werden kann, müssen zwangsläufig zwei Datensätze verwendet werden. Mit dem ersten Datensatz wird der Klassifikator trainiert und mit dem zweiten Datensatz getestet. Dementsprechend spricht man von Trainingsdaten und Testdaten. Die beiden Datensätze werden möglichst geschickt aus der Stichprobe konstruiert, um einen guten Klassifikator durch viele Trainingsmuster und trotzdem eine möglichst gute Fehlerabschätzung durch viele Testmuster zu erhalten.

Definition des Klassifikationsfehlers

Ein naheliegendes Maß für die Güte eines Klassifikators ist die Fehlerrate. Ein Fehler ist die Zuordnung einer falschen Klasse zu einer Beobachtung. Zur Berechnung der Güte wird jedes einzelne Muster der Stichprobe klassifiziert und die zugeordnete Klasse mit der bekannten Klasse des Musters verglichen[1]. Die Anzahl der fehlerhaften Klassifikationen wird auf die Anzahl der präsentierten Muster normiert:

1 *Es werden hier nur Klassifikatoren mit dieser Vorgabe betrachtet (überwachtes Lernen!)*

$$Fehlerrate = \frac{Anzahl\ falsch\ klassifizierter\ Muster}{Anzahl\ klassifizierter\ Muster} \qquad \textbf{Gl. 12}$$

Man erhält einen Wert zwischen null und eins, welcher die Güte des Klassifikators auf der präsentierten Auswahl von Mustern widerspiegelt. Ist der Wert null, wurden alle Muster richtig eingeteilt. Ist der Wert eins, wurden alle Muster falsch klassifiziert. Dieser Wert ist der ***Klassifikationsfehler*** und wird in den meisten Fällen in prozentualer Form angegeben. Bei einer genügend großen Anzahl von Mustern nähert sich diese Abschätzung asymptotisch dem wahren Klassifikationsfehler des Klassifikators [Weiss, 1991]. Der Klassifikationsfehler gewichtet alle Fehler gleichmäßig. Bei vielen Klassifikationsproblemen spielt jedoch die Art des Fehlers bei der Beurteilung der Güte eine große Rolle. Ist eine unterschiedliche Gewichtung notwendig, kann diese durch eine ***confusion matrix*** (Fehlermatrix) in die Berechnung des Klassifikationsfehlers eingebracht werden [Weiss, 1991].

In der Literatur wird der Klassifikationsfehler für die Trainingsmuster als ***Trainingsfehler*** e_t bezeichnet. Eines der häufigsten Probleme des Aufbaus von Klassifikationssystemen ist die zu geringe Anzahl von Trainingsmustern. Klassifikatoren tendieren bei geringen Datenmengen leicht zu einer Überspezialisierung. Die Trainingsmuster werden lediglich auswendig gelernt. Der Trainingsfehler schätzt einzig das Klassifikationsverhalten auf den bekannten Mustern ab und ist deshalb generell zu optimistisch. Das Verhalten wird als ***biased*** (voreingenommen) bezeichnet. Demgegenüber steht der Fehler auf den Testmustern, welcher als ***Testfehler*** oder ***Generalisierungsfehler*** e_g bezeichnet wird. Der Generalisierungsfehler kann nur geschätzt werden, da die Größe der Stichprobe beschränkt ist. Trotzdem hat der geschätzte Generalisierungsfehler eine hohe Aussagekraft, da er sich mit zunehmender Anzahl von Testmustern dem tatsächlichen Generalisierungsfehler annähert. Zur Angabe der Klassifikationsgüte wird fast ausschließlich der geschätzte Generalisierungsfehler verwendet.

Im folgenden wird als „Generalisierungsfehler“ immer der „geschätzte Generalisierungsfehler“ bezeichnet. Ausnahmen hiervon werden explizit angegeben. In den nächsten Abschnitten werden Methoden vorgestellt, mit welchen die Stichprobe „geschickt“ in Testmuster und Trainingsmuster unterteilt werden können. Die Methoden dienen nur einer möglichst genauen Fehlerabschätzung. Zum Aufbau des endgültigen Klassifikators wird immer die gesamte Stichprobe verwendet.

Trennen in Test- und Trainingsdaten (Holdout)

Die korrekte Fehlerabschätzung basiert darauf, den Klassifikator mit unbekannten Daten zu testen. Es ist daher naheliegend, nicht alle vorhandenen Muster für das Training zu verwenden und die Stichprobe aufzuteilen. Die erste Hälfte verwendet man, um den Klassifikator zu trainieren. Die zweite Hälfte wird dazu verwendet, den Generalisierungsfehler zu schätzen. Dieses Verfahren bezeichnet man als ***Holdout***, da bei dem Training des Klassifikators ein Teil der Muster zurückgehalten wird. In der Praxis hat die Methode einige schwerwiegende Nachteile. Im Allgemeinen stehen nicht einmal genug Muster für ein ausreichendes Training des Klassifikators zur Verfügung. Halbiert man die Stichprobe, verliert man für das Training wertvolle Beispiele. Eine Variante dieses Verfahrens besteht daher darin, die Anzahl der Testmuster zu Gunsten der Trainingsmuster zu verschieben. Aber selbst wenn man zwei Drittel oder gar neun Zehntel der Stichprobe zum Training einsetzt, vergeudet man Daten, durch welche sich der Klassifikator entscheidend verbessern könnte. Daher eignet sich ein Holdout nur dann, wenn nahezu unbegrenzt Muster zur Verfügung stehen. In diesem Fall liefert das Verfahren die beste Abschätzung des Generalisierungsfehlers mit dem geringsten Aufwand.

Zufällig wiederholtes Mischen (Random Resampling)

Bei einem Holdout kann es passieren, daß durch ungeschickte Trennung der Muster zu viele Beispiele einer bestimmten Klasse in einen der beiden Datensätze aufgenommen werden. Im ungünstigsten Fall ist eine Klasse überhaupt nicht in einem der beiden Datensätze enthalten. Das Holdout-Verfahren läßt sich verbessern, indem die Muster derart aufgeteilt werden, daß die Verhältnisse der Klassenanteile in beiden Unterdatensätzen gewahrt bleiben. Dies wird als ***Stratify*** bezeichnet. Eine solchermaßen gesteuerte Auswahl ist die einzig zulässige Operation auf der Stichprobe, da bei anderen Eingriffen die statistische Unabhängigkeit der Muster nicht mehr gewährleistet ist. Die Idee des Stratifying führt dazu, das Holdout-Verfahren nicht nur einmal durchzuführen, indem die Daten an einem Punkt in der vorgegebenen Reihenfolge aufgetrennt werden, sondern sie vorher zufällig zu durchmischen und das Verfahren beliebig oft zu wiederholen. Der Generalisierungsfehler ergibt sich aus dem gemittelten Generalisierungsfehler aller Durchläufe. Im Allgemeinen wählt man für die Anzahl b der Wiederholungen willkürlich einen Wert erheblich kleiner als die Anzahl o der Muster. Es ist für einen Menschen mit Schwierigkeiten verbunden, eine zufällige Mischung der Muster zu erreichen. Für eine Maschine sind jedoch sehr gute Ergebnisse in Bezug auf die statistische Unabhängigkeit der gemischten Muster zu erwarten.

Methoden der Kreuzvalidierung

Wirkliche Schwierigkeiten bei der Schätzung des Generalisierungsfehlers treten erst dann auf, wenn nur eine sehr begrenzte Anzahl von Mustern zur Verfügung steht. Dies ist insbesondere bei medizinischen Klassifikationsproblemen der Fall, wenn nur wenige Testpersonen untersucht werden konnten. Einige Dutzend Muster für ein Random Resampling oder Holdout sind nicht ausreichend. Die Rollen von Trainingsdatensatz und Testdatensatz können nach erfolgter Fehlerabschätzung allerdings vertauscht werden. Dadurch kann der Klassifikator mit den vorherigen Testmustern erneut trainiert und die Fehlerabschätzung mit den vorherigen Trainingsmustern wiederholt werden. Die Generalisierungsfehler beider Durchgänge werden gemittelt. Insgesamt werden dadurch alle Daten für Training und Test verwendet. Die Muster in zwei gleich große Gruppen einzuteilen und nacheinander jeweils die eine Gruppe für das Training und die andere für den Test zu verwenden, bezeichnet man als ***two-fold-crossvalidation*** oder ***zweifache Kreuzvalidierung***. Verschiebt man das Größenverhältnis der Gruppen zugunsten der Trainingsdaten, bis die Testdaten nur noch ein Drittel aller Muster enthalten, verbessert sich der Klassifikator entsprechend. In diesem Fall erstellt man drei Gruppen und verwendet jeweils eine für den Test und die beiden anderen für das Training. Entsprechend nennt man das Verfahren ***three-fold-crossvalidation*** oder ***dreifache Kreuzvalidierung***. Eine Aufteilung in beliebig viele Gruppen bis zur Anzahl o der Muster kann die Qualität der Fehlerabschätzung erheblich steigern. Allgemein wird ein Klassifikator jeweils mit den Mustern der entfernten Gruppe getestet. Die Generalisierungsfehler werden aufsummiert und auf die Anzahl der Gruppen normiert.

Das statistisch genaueste Verfahren dieser Art ist die ***vollständige Kreuzvalidierung***, die auch als ***leave-one-out*** bezeichnet wird. Dabei wird sukzessive genau ein Muster aus der Stichprobe entfernt und der Klassifikator mit den restlichen Mustern trainiert. Auf diese Weise wird der bestmögliche Klassifikator erstellt. Bemerkenswert an der leave-one-out Methode ist, daß die Fehlerabschätzung systematisch weder zu gut noch zu schlecht ist [Weiss, 1991]. Man bezeichnet sie daher als ***unbiased***. Ein schwerer Nachteil dieser Methode liegt in dem verhältnismäßig hohem Berechnungsaufwand. Dieser steigt zwar nur linear mit der Anzahl der Muster, in jedem Schritt müssen jedoch die Kosten für das Training des Klassifikators hinzugefügt werden. Aus diesem Grund wird bei genügend großer Stichprobe oft ein Mittelweg zwischen Zuverlässigkeit und Laufzeitverhalten gewählt. Als guter Kompromiß wird in der Praxis die ***ten-fold-crossvalidation*** oder ***zehnfache Kreuzvalidierung*** betrachtet. Für die Überlegenheit der ten-fold-crossvalidation gibt es sowohl empirische Beobachtungen als auch theoretische Hinweise [Witten, 2000].

Das leave-one-out Verfahren ist zur Fehlerabschätzung für große Mustermengen sehr genau. Da es jedoch über eine hohe Varianz verfügt, kann es für kleine Mustermengen (kleiner 100 Mustern) zu starken Differenzen zwischen Schätzung und tatsächlichem Generalisierungsfehler kommen.

Bootstrap Verfahren

Bootstrap Verfahren liefern insbesondere für kleine Stichproben gute Schätzungen des Generalisierungsfehlers [Fukunaga, 1990].

Die bisher vorgestellten Methoden wählen Muster aus der Stichprobe aus und teilen diese eindeutig in Test- und Trainingsdaten auf. Jedes Muster wird genau einer Gruppe zugeordnet. Das Verfahren zur Aufteilung der Muster bezeichnet man als „zufällige Auswahl ohne Zurücklegen“. Bei dem Bootstrap Verfahren verbleiben zufällig ausgewählte Muster in der Stichprobe. Die Trainingsdaten werden erstellt, indem man aus den o Mustern der Stichprobe (o)-mal zufällig ein Muster aussucht und in den Trainingsdatensatz kopiert. Einige Muster können dadurch mehrfach im Trainingsdatensatz vertreten sein, andere überhaupt nicht. Unter dem Oberbegriff Bootstrap existieren mehrere Verfahren, welche den Generalisierungsfehler auf unterschiedliche Weise aus den Trainingsdaten schätzen. Am weitesten verbreitet sind ***e0-bootstrap*** und ***0.632-bootstrap***. Diese beiden Methoden werden im folgenden vorgestellt.

Zunächst stellt sich die Frage, wieviele Muster der Stichprobe mit o Elementen im Trainingsdatensatz nach o Schritten enthalten sind. Die Wahrscheinlichkeit, daß ein bestimmtes Muster in die Trainingsdaten kopiert wird, beträgt bei Gleichverteilung der Muster in jedem Auswahlschritt $1/o$. In jedem Schritt beträgt die Wahrscheinlichkeit für das Muster, nicht ausgewählt zu werden, $1-1/o$. Nach o Schritten beträgt die Wahrscheinlichkeit p, daß ein Muster nicht in die Trainingsdaten kopiert wurde:

Gl. 13
$$p = \left(1-\frac{1}{o}\right)^{o} \approx e^{-1} = 0{,}368 \qquad o \gg 1$$

Umgekehrt bedeutet dies, daß ein Muster mit der Wahrscheinlichkeit von etwa 0,632 in den Trainingsdatensatz kopiert wird. Der Testdatensatz besteht aus den $m \approx 0{,}368 \cdot o$ Mustern, die nicht in die Trainingsdaten kopiert wurden.

Bei dem e0-bootstrap wird der Fehler auf dem Testdatensatz als Schätzung des Generalisierungsfehlers verwendet. Diese Methode verfügt auch für kleine Stichproben über eine geringe Varianz und liefert daher eine stabile Abschätzung des Generalisierungsfehlers. Der Preis hierfür ist eine generell zu pessimistische Abschätzung [Michie, 1994]. Das systematisch zu schlechte Verhalten des e0-bootstrap begründet sich aus der Tatsache, daß für den Aufbau des Klassifikators im Mittel nur etwa 63 Prozent der Muster verwendet werden.

Die 0.632-bootstrap Methode kompensiert die zu pessimistische Schätzung durch die Testdaten mit Hilfe der generell zu optimistischen Schätzung durch die Trainingsdaten. Der Generalisierungsfehlers e_g wird durch Linearkombination von Trainingsfehler e_{tr} und Testfehler e_{te} geschätzt:

$$e_g = 0{,}632 \cdot e_{te} + 0{,}368 \cdot e_{tr} \qquad \textbf{Gl. 14}$$

Andere Linearkombinationen sind ebenfalls gebräuchlich [The Globus Alliance, 2005] [Weiss, 1991]. In der Praxis wiederholt man das Bootstrap Verfahren etwa 200 mal, um durch die Mittelung eine genauere Abschätzung zu erreichen [Weiss, 1991].

Komplexität der verschiedenen Fehlerschätzverfahren

Die beschriebenen Methoden zur Schätzung des Generalisierungsfehlers bieten verschiedene Vor- und Nachteile. Die Wahl der richtigen Methode ist abhängig von der Größe der Stichprobe, der gewünschten Genauigkeit der Schätzung und der zur Berechnung verfügbaren Zeit. Tabelle 1 gibt Aufschluß über den Aufwand der verschiedenen Methoden in Form benötigter Iterationen und als *O*-Notation [Manber, 1989]. Die Methoden mit gleicher Komplexität unterscheiden sich durch konstante Faktoren, welche bei der *O*-Notation nicht wiedergegeben werden. In der Praxis fallen diese Konstanten stark ins Gewicht. Je nach Beschaffenheit des Datensatzes können sich zweifache Kreuzvalidierung und Bootstrap Verfahren um mehrere Größenordnungen an Rechenzeit unterscheiden. Liegt eine „unbegrenzte“ Stichprobe vor, brauchen die vorgestellten Methoden nicht angewandt zu werden. Test- und Trainingsdatensatz können direkt mit der gewünschten Anzahl an Mustern erstellt werden.

		Anzahl				
		Datensatz	Trainingsdaten	Testdaten	Iterationen	Komplexität
Methode	Holdout	o	$o - m$	m	1	$O(1)$
	Resampling	o	$o - m$	m	$b \ll o$	$O(1)$
	Two-fold	o	$0{,}5 \cdot o$	$0{,}5 \cdot o$	2	$O(1)$
	Ten-fold	o	$0{,}9 \cdot o$	$0{,}1 \cdot o$	10	$O(1)$
	Leave-one-out	o	$o - 1$	1	o	$O(o)$
	Bootstrap	o	o	$o - m$	~ 200	$O(1)$

Tabelle 1. Laufzeitverhalten von Algorithmen zur Schätzung des Generalisierungsfehlers. Eine Zeile enthält jeweils die Information über eine bestimmte Schätzmethode. Bei der Abschätzung durch die O-Notation ist die Komplexität aller Methoden bis auf leave-one-out identisch. In der Praxis kleiner Datensätze unterscheiden sich die Methoden jedoch erheblich durch konstante Faktoren, weshalb die notwendigen Iterationen eine bessere Abschätzung der zu erwartenden Laufzeit erlauben.

3.5 Verbesserung der Klassifikation durch Merkmalsselektion

Wird ein Klassifikator bei gleichbleibender Menge von Mustern mit einer Untermenge von Merkmalen trainiert, geht man intuitiv davon aus, daß sich das Klassifikationsergebnis verschlechtern wird. Überraschenderweise beobachtet man in der Praxis oftmals das Gegenteil. Mit einer Untermenge von Merkmalen kann bei gleichbleibender Anzahl von Mustern eine Verbesserung des Generalisierungsfehlers eintreten. Der Trainingsfehler nimmt dabei normalerweise zu. Als Faustregel für die Größe einer brauchbaren Stichprobe gilt ein Verhältnis von mindestens zehnmal mehr Mustern als Merkmalen. Diese Regel wird in der Praxis häufig eingesetzt, obwohl sie in vielen Fällen falsch ist [Fukunaga, 1990]. Je komplexer ein Klassifikator ist, desto mehr Parameter müssen bestimmt werden und desto mehr Beispiele sind hierzu notwendig. Ein einfacher Klassifikator generalisiert bei einer geringen Anzahl von Mustern deshalb häufig besser als ein komplexer Klassifikator.

Ein Klassifikator kann niemals besser werden als die Beispiele, mit denen er trainiert wird. In der Praxis ist die Qualität einer Stichprobe meist weit von dem Ideal entfernt. Sie kann durch starke Korrelation der Merkmale nahezu unbrauchbar sein und häufig fehlen seltene, aber für die Klassifikation kritische Werte von Merkmalen. Es sind jedoch nur wenige nachträgliche Eingriffe in eine vorliegende Stichprobe erlaubt, die ihre statistische Unabhängigkeit nicht gefährden:

- Muster mit fehlenden Merkmalen könnten entfernt werden. Dadurch reduziert sich aber die Größe der Stichprobe. Aus diesem Grund wird häufig versucht, fehlende Merkmale eines Mus-

ters zu interpolieren. Fehlt ein Merkmal in sehr vielen Mustern, wird es aus allen Mustern der Stichprobe entfernt.

- Mehrfach korrelierte Merkmale werden zusammengefaßt. Merkmale, die nichts oder nur wenig zur Klassifikation beitragen, werden im allgemeinen ganz aus der Stichprobe entfernt.

Dieses Vorgehen bezeichnet man als ***Merkmalsselektion***. Man sucht nach der optimalen Untermenge in der Menge aller Merkmale, mit der sich der geringste Generalisierungsfehler ergibt. Bei P Merkmalen muß die Potenzmenge mit insgesamt 2^P Elementen vergleichend getestet werden. Für diesen Test müssen alle möglichen Klassifikatoren mit allen möglichen Merkmalskombinationen trainiert und alle Generalisierungsfehler miteinander verglichen werden. Die Komplexität dieser Methode ist derart hoch, daß sie in der Praxis schon bei relativ kleinen Merkmalsmengen versagt. Aus diesem Grund haben sich folgende Verfahren für die Merkmalsselektion als praktikabel erwiesen [Witten, 2000] [Weiss, 1991]:

- Die einfachste und schnellste Methode besteht in der Annahme von Unabhängigkeit der ausgewählten Merkmale. Es werden alle Merkmale ausgewählt, die aufgrund eines einfachen statistischen Tests wichtig erscheinen. Dies kann beispielsweise die Varianz der Merkmale sein. Merkmale ohne Varianz tragen keine Unterscheidungsinformation und werden entfernt. Merkmale mit hoher Varianz ermöglichen oft eine gute Unterscheidung zwischen den Klassen. Solche Merkmale werden daher bevorzugt ausgewählt. In realen Klassifikationsproblemen ist die Unabhängigkeit von Merkmalen jedoch häufig nicht gegeben.

- Eine schrittweise Auswahl des jeweils besten Merkmals (***stepwise-forward-selection***) aus der verbleibenden Menge berücksichtigt teilweise die Korrelation von Merkmalen. Aus der selektierten Menge wird allerdings niemals ein Merkmal zurückgelegt, weshalb Korrelationen nicht vollständig berücksichtigt werden. Die Ergebnisse sind zumeist besser als die unter der oftmals nicht zutreffenden Annahme der Unabhängigkeit von Merkmalen ermittelten.

- Das schrittweise Entfernen des schlechtesten Merkmals (***stepwise-backward-elemination***) aus der ursprünglichen Merkmalsmenge führt zu ähnlich guten Ergebnissen wie eine stepwise-forward-selection. Korrelationen werden ebenfalls nicht vollständig berücksichtigt.

- Abwechselndes Auswählen und Entfernen von Merkmalen bietet die Möglichkeit, Korrelationen zwischen Merkmalen noch besser zu erfassen als durch die Anwendung von jeweils nur einem dieser Verfahren. Beide Verfahren werden solange alterniert, bis die Qualität der Merkmalsmenge einem vorgegebenen Kriterium genügt.

Hauptkomponentenanalyse

Ein weit verbreitetes Verfahren zur Reduktion hochdimensionaler Merkmalsräume ist die Hauptkomponentenanalyse (***Principal Component Analysis***, ***PCA***) [Pearson, 1901] [Hotelling, 1933]. Da der Aufwand einer manuellen Berechnung extrem hoch ist, ließ sich das Verfahren zunächst nur für wenige Merkmale einsetzen. Erst mit der Einführung maschineller Rechenmaschinen gewann die PCA an Bedeutung [Manly, 1994].

Die Idee der Hauptkomponentenanalyse besteht darin, die gegebenen Merkmale $x_1, x_2, \cdots, x_m$ auf Zusammenhänge zu untersuchen und diese zu neuen, künstlichen Merkmalen $z_1, z_2, \cdots, z_m$ zusammenzufassen, welche unkorreliert sind. Das Verfahren besteht in einer linearen Transformation des Merkmalsraumes *X* in den Raum *Z* und wird als Hauptachsentransformation oder Karhunen-Loève-Transformation bezeichnet [Lehmann, 1997]. Ein Merkmal z_i besteht aus einer Linearkombination der ursprünglichen Merkmale:

Gl. 15
$$z_i = a_{i1} \cdot x_1 + a_{i2} \cdot x_2 + \cdots + a_{im} \cdot x_m$$

Die z_i werden Hauptkomponenten genannt. Sie sind im Gegensatz zu den x_i unkorreliert und repräsentieren linear unabhängige Merkmale im Merkmalsraum. Während der Transformation versucht man, die Varianz der z_i zu maximieren. Zusätzlich sortiert man sie derart um, daß z_1 über die größte Varianz verfügt, z_2 über die zweitgrößte Varianz, u.s.w. Es ergibt sich die Reihenfolge:

$$var(z_1) \geqslant var(z_2) \geqslant \cdots \geqslant var(z_m)$$

wobei $var(z_i)$ die Varianz des Merkmals z_i beschreibt. Um die Varianz einer Hauptkomponente zu maximieren, führt man zu der Transformation zusätzlich folgende Bedingung ein:

Gl. 16
$$a_{i1}^2 + a_{i2}^2 + \cdots + a_{im}^2 = 1$$

da die Varianz ansonsten durch Erhöhen einer der Konstanten a_{ij} beliebig vergrößert werden kann. Die Transformation läßt sich mit Hilfe der Kovarianzmatrix Σ der Merkmale bestimmen, indem man die Eigenwerte von Σ berechnet. Eine Kovarianzmatrix ist symmetrisch und verfügt ausschließlich über positive Eigenwerte. Werden diese ihrer Größe nach geordnet $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_m$, dann entspricht der i-te Eigenwert λ_i der Varianz der i-ten Hauptkomponente $var(z_i)$ und die Konstanten a_{i1}, a_{i2}, $\cdots$, a_{im} der Hauptkomponente z_i entsprechen dem skalierten Eigenvektor zu λ_i [Manly, 1994].

Um eine gerechte Bewertung aller Merkmale zu erreichen, werden diese vor der Transformation normiert. Die normierten Merkmale haben den Mittelwert *0* und verfügen über eine Varianz von eins. Die verwendete Kovarianzmatrix erhält daher die Form

$$\Sigma = \begin{bmatrix} 1 & \sigma_{12} & \cdots & \sigma_{1m} \\ \sigma_{21} & 1 & \cdots & \sigma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m1} & \sigma_{m2} & \cdots & 1 \end{bmatrix} \qquad \textbf{Gl. 17}$$

wobei $\sigma_{ii}=1$ die Varianz des Merkmals x_i ist und $\sigma_{ij}=\sigma_{ji}$ die Kovarianz zwischen Merkmal x_i und x_j angibt. Die derart normierte Form der Kovarianzmatrix wird als Korrelationsmatrix bezeichnet.

Die Summe der Eigenwerte einer Matrix entspricht der Summe ihrer Spur. Die Spur einer Kovarianzmatrix besteht aus der Summe der Varianzen aller Merkmale:

$$\sum_{i=1}^{m} \lambda_i = \sum_{i=1}^{m} \sigma_{ii} = m \qquad \textbf{Gl. 18}$$

Aus Gleichung 18 ist ersichtlich, daß eine Hauptkomponentenanalyse bei der Basistransformation die Gesamtvarianz aller Merkmale erhält, diese jedoch umverteilt. Redundant gemessene Merkmale werden durch die Transformation zu einer Hauptkomponente zusammengefaßt. Da die Dimensionen der Räume *X* und *Z* gleich bleiben, ergeben sich bei redundanten Merkmalen zwangsläufig Eigenwerte und somit Varianzen mit dem Wert null. Da man die Varianz, wie im vorhergehenden Abschnitt bereits erwähnt, als ein Maß für den Informationsgehalt eines Merkmals verwenden kann, können im Anschluß an die Transformation Hauptkomponenten mit einer Varianz nahe *0* vernachlässigt werden [Jähne, 1997]. Die Schwelle variiert jedoch mit jedem neuen Problem und muß individuell betrachtet werden. Eine extreme Reduktion läßt sich beispielsweise durch ausschließliche Verwendung der

ersten beiden Hauptkomponenten erzwingen. In der Praxis wird dadurch ein zweidimensionales Schaubild mit „maximalem“ Informationsgehalt ermöglicht.

Die Varianz ist allerdings nicht immer das beste Maß für den Nutzen eines Merkmals. Fällt die Varianz eines Merkmals klassenweise gering aus und haben die Klassen einen geringen Abstand, lassen sie sich mit kleinem Generalisierungsfehler trennen. Ist die Varianz eines zweiten Merkmals klassenweise groß und überlappen sich die Klassen, lassen sich die Klassen nur mit großem Fehler voneinander trennen. Das erste Merkmal ist für eine Klassifikation wesentlich besser geeignet als das zweite. Eine PCA mit Schwellwertverfahren würde allerdings das zweite Merkmal als wichtiger einstufen und das erste Merkmal verwerfen.

In der Praxis sind Merkmale sehr oft voneinander abhängig. Durch eine PCA erfolgt jedoch nicht zwangsläufig eine Merkmalsreduktion. Sind alle Merkmale unkorreliert, findet lediglich eine Basistransformation und keine Reduktion statt. Gegen alle Vorteile einer Hauptkomponentenanalyse steht ein schwerwiegender Nachteil: Die künstlich erzeugten Merkmale beschreiben Zusammenhänge im Ursprungsraum, wobei sie die gemessenen Merkmale vereinen und deren Information „komprimieren“. Ein Mensch kann sich nur sehr schwer eine Vorstellung über die semantische Bedeutung der vereinten Merkmale machen.

3.6 Zusammenfassung

Aus der Literatur sind zahlreiche Algorithmen für die Klassifikation und Merkmalsselektion bekannt, die für unterschiedliche Klassifikationsaufgaben jeweils verschieden gut geeignet sind. Die meisten dieser Algorithmen stehen bereits als kommerzielle, oftmals auch als freie Implementierungen zur Verfügung. Die Problematik bei dem Entwurf von Klassifikationssystemen besteht nicht länger in dem Entwurf der Klassifikationsalgorithmen und ihrer Implementierung, sondern in der gezielten Auswahl der verfügbaren Algorithmen, die für das jeweils gestellte Klassifikationsproblem optimal geeignet sind. Diese Aufgabe wird im allgemeinen manuell und „aus dem Bauch heraus“ gelöst. Ein Werkzeug, das diese Aufgabe automatisch löst, ist bislang unbekannt.

Die Qualität einer Algorithmenkombination kann durch Schätzung des zu erwartenden Generalisierungsfehlers bestimmt werden. Ein primitiver, mechanischer Ansatz besteht in der Aufzählung und

dem Test aller denkbaren Algorithmenkombinationen. Aufgrund der hierzu erforderlichen Laufzeiten ist dieser Ansatz in der Praxis jedoch undurchführbar.

Benötigt werden deswegen effiziente Methoden, die dem Entwickler von Klassifikationssystemen das Problem der Auswahl geeigneter Klassifikationsalgorithmen abnehmen und in kurzer Zeit gute Klassifikationssysteme erstellen. Solche Synthesemethoden werden im folgenden Kapitel hergeleitet und bewertet.

4 Automatische Synthese von Klassifikationssystemen

Synthese: *[griechisch] „Zusammenfügung, Verknüpfung einzelner Teile zu einem höheren Ganzen“*

[Duden - Das Fremdwörterbuch, 2001]

Ein Entwicklungswerkzeug erleichtert den Entwurf von Klassifikationssystemen besonders dann, wenn es mit Hilfe integrierter Methoden automatisch eine geeignete Auswahl von Merkmalsselektoren und Klassifikatoren trifft. Diese Methoden sollen für eine gestellte Klassifikationsaufgabe innerhalb akzeptabler Zeit ein Klassifikationssystem mit niedrigem Generalisierungsfehler bestimmen. Im folgenden werden derartige Methoden für die automatische Synthese von Klassifikationssystemen entwickelt und beschrieben. In Kapitel 6 werden diese Methoden an verschiedenen Klassifikationsaufgaben vergleichend getestet.

4.1 Synthesemethoden

Der Entwurf von Klassifikationssystemen ist ein NP-vollständiges Problem. Die optimale Lösung für eine gestellte Klassifikationsaufgabe besteht aus einer beliebigen Kombination von Merkmalsselektoren und Klassifikatoren. Diese optimale Kombination zu bestimmen erfordert den Aufbau und Test aller Kombinationsmöglichkeiten von Klassifikationsalgorithmen. Ein solch stupides „Durchprobieren“ ist aus Komplexitätsgründen nicht praktikabel.

4.1.1 Ansätze für den Entwurf von Synthesemethoden

Das Ziel eines automatischen Entwurfs liegt darin, in möglichst ***kurzer*** Zeit ein Klassifikationssystem mit ***niedrigem***, idealerweise dem optimalen Generalisierungsfehler zu erstellen. Beides widerspricht sich im allgemeinen, da das Klassifikationssystem mit dem niedrigsten Generalisierungsfehler nur durch Test aller Kombinationen aus Merkmalsselektoren und Klassifikatoren ermittelt werden kann.

Wovon werden Generalisierungsfehler und Laufzeitverhalten beeinflußt? Der erreichbare Generalisierungsfehler ist lediglich abhängig von den getesteten Kombinationsmöglichkeiten (das „optimale“

Klassifikationssystem sei dabei das beste aus den verfügbaren Algorithmen erstellbare). Die genaueste Schätzung des Generalisierungsfehlers wird im allgemeinen durch eine vollständige Kreuzvalidierung erreicht. Andere Schätzmethoden sind zum Teil erheblich schneller, allerdings auch wesentlich ungenauer.

Die benötigte Laufzeit für den Test aller Kombinationsmöglichkeiten skaliert mit mehreren unterschiedlichen Faktoren:

- der Anzahl der Klassifikatoren k und der Merkmalsselektoren s
- der Anzahl der Merkmale m und der Muster o
- der Anzahl der verfügbaren Recheneinheiten
- der verwendeten Schätzmethode des Generalisierungsfehlers

Betrachtet man zunächst nur einzelne Recheneinheiten und wird einer der anderen Faktoren reduziert, verringert sich die Laufzeit. Werden jedoch nicht alle diese Faktoren ausgenutzt, kann nicht mehr garantiert das optimale Klassifikationssystem bestimmt werden. An dieser Stelle muß ein Kompromiß zwischen akzeptablem Generalisierungsfehler und akzeptabler Laufzeit gefunden werden.

Ein automatischer Entwurf ist zunächst einmal nur dann in akzeptabler Zeit möglich, wenn die unbeschränkte Anzahl der Kombinationsmöglichkeiten begrenzt wird. Die folgenden beiden Einschränkungen reduzieren die Suche nach dem besten Klassifikationssystem auf ein zweidimensionales Optimierungsproblem und beschränken die maximal benötigte Laufzeit. Sie bilden die Grundlage für die entwickelten Synthesemethoden:

- Komplexe Kombinationen von Klassifikationsalgorithmen verbessern die Qualität einfacher Kombinationen oftmals nur geringfügig, wobei der zeitliche Aufwand zur Bestimmung der komplexen Kombination unverhältnismäßig anwächst. Beispielsweise nimmt die Laufzeit durch hierarchische Schachtelungen um Größenordnungen zu, wohingegen der Generalisierungsfehler zumeist nur um wenige Prozent abnimmt. Aus diesem Grund ist es ohne großen Qualitätsverlust zulässig, den Aufwand durch Vernachlässigung komplexer Kombinationsmöglichkeiten zu

beschränken. Das gesuchte Klassifikationssystem soll sich daher aus genau einem Merkmalsselektor und genau einem Klassifikator zusammensetzen.

- Manche Klassifikatoren können individuell parametrisiert werden. Eine Variation dieser Parameter kann die Qualität der Klassifikation verbessern, erfordert jedoch ebenfalls einen unverhältnismäßig hohen Zeitaufwand. Da der Gewinn im allgemeinen nicht groß genug ist, um den höheren Aufwand zu rechtfertigen, wird im folgenden keine Parameteroptimierung durchgeführt. Es werden die jeweiligen Standardwerte der implementierten Algorithmen verwendet.

Um in dem verbleibenden, zweidimensionalen Suchraum die beste Kombination zu bestimmen, ist es naheliegend, sämtliche Möglichkeiten aufzuzählen und ihre jeweiligen Generalisierungsfehler zu schätzen. Die Komplexität $O(s \cdot k \cdot o)$ aus dem Produkt der Anzahl von Merkmalsselektoren s, Klassifikatoren k und Muster o (bei vollständiger Kreuzvalidierung) ist für den praktischen Einsatz jedoch immer noch zu hoch. Auf zu diesem Zeitpunkt gängigen Rechensystemen kann eine Synthese durch Aufzählen aller verbliebenen Kombinationsmöglichkeiten leicht mehrere Wochen benötigen (siehe Kapitel 6).

Herkömmliche Optimierungsmethoden wie beispielsweise *Gradientenabstieg* können zu der Lösung dieses Optimierungsproblems nicht herangezogen werden. Von dem Generalisierungsfehler einer bestimmten Kombination kann nämlich nicht auf die Generalisierungsfehler benachbarter Kombinationen geschlossen werden. Weder zwischen den Algorithmen der Merkmalsselektion noch zwischen denen der Klassifikation besteht eine Beziehung bezüglich des Generalisierungsfehlers. Als Schlußfolgerung müssen eigene Methoden für die Lösung des Optimierungsproblems entwickelt werden, die an den laufzeitbestimmenden Faktoren ansetzen.

4.1.2 Reduktion von Klassifikatoren und Selektoren

Die folgenden Methoden reduzieren die benötigte Laufzeit, indem sie nur einen Teil der möglichen Kombinationen von Klassifikatoren und Merkmalsselektoren verwenden. Die Idee besteht im wesentlichen darin, solange Kombinationen zu testen wie Rechenzeit zur Verfügung steht. Der Vollständigkeit halber werden auch extrem einfache Methoden vorgestellt.

Zufällige Klassifikatorwahl (mit Merkmalsselektion)

Das einfachste Klassifikationssystem besteht trivialerweise aus genau einem Klassifikator ohne vorangehende Merkmalsselektion (Null-Selektor). Zufällig einen beliebigen Klassifikator auszuwählen ist die mit Abstand schnellste Möglichkeit, ein Klassifikationssystem zu synthetisieren. Der Gesamtaufwand beträgt $O(o)$ (o ist die Anzahl der Muster) bei Schätzung des Generalisierungsfehlers durch vollständige Kreuzvalidierung. Der Generalisierungsfehler des resultierenden Systems kann zwar berechnet, jedoch nicht gegenüber anderen Klassifikationssystemen eingeordnet werden (die ja nicht erstellt werden). Im ungünstigsten Fall erstellt diese Methode sogar das schlechteste Klassifikationssystem.

Ein Klassifikationssystem ohne Merkmalsselektion wird in den meisten Fällen verbessert, wenn die Merkmale zuvor von einem beliebigen Merkmalsselektor bearbeitet werden [Neiber, 2001]. Anstelle nur eines Klassifikators kann zusätzlich ein beliebiger Merkmalsselektor zufällig ausgewählt werden. Die Komplexität beträgt weiterhin $O(o)$.

Wiederholte zufällige Klassifikatorwahl (mit Merkmalsselektion)

Die zufällige Auswahl eines Klassifikators kann so oft wiederholt werden, bis alle Klassifikatoren getestet wurden oder ein vorgegebenes Abbruchkriterium erfüllt ist. Als Ergebnis wird das Klassifikationssysteme mit dem niedrigsten Generalisierungsfehler verwendet. Mögliche Abbruchkriterien werden in Abschnitt 4.1.5 vorgestellt und sind grundsätzlich für alle im folgenden vorgestellten Synthesemethoden gültig. Der Aufwand beträgt $O(k \cdot o)$, bedingt durch die iterative Auswahl aus k Klassifikatoren. Verständlicherweise werden mit dieser Methode im allgemeinen bessere Klassifikationssysteme synthetisiert als durch zufällige Auswahl eines einzigen Klassifikators.

Auch diese Synthesemethode läßt sich durch eine Merkmalsselektion verbessern. Zuerst wird zufällig ein Merkmalsselektor ausgewählt und die Merkmalsselektion durchgeführt. Danach wird wiederholt zufällig ein Klassifikator ausgewählt. Die Komplexität dieser Methode beträgt ebenfalls $O(k \cdot o)$. Eine Variante besteht in der Vorgabe eines besonders schnellen Merkmalsselektors anstelle eines zufälligen, da der Generalisierungsfehler allgemein mit jedem beliebigen Merkmalsselektor sinkt. Ein hierfür gut geeigneter Merkmalsselektor ist die in Abschnitt 3.5 stepwise-forward-selection, die gemeinhin einen guten Kompromiß zwischen Aufwand und Leistung bietet.

Eine weitere Verbesserung der Synthesemethode ist möglich, indem nicht irgendein beliebiger Merkmalsselektor ausgewählt wird, sondern der vermeintlich „beste“. Alle Merkmalsselektoren werden mit einem bestimmten Klassifikator kombiniert. Darauf wird eine Merkmalsselektion mit dem Merkmalsselektor durchgeführt, mit dem der niedrigste Generalisierungsfehler erreicht wurde. Mit diesen Merkmalen werden alle Klassifikatoren getestet. Die zweimalige, sequentielle Suche führt nicht zwangsläufig zur besten Kombination, der Aufwand fällt mit $O((s+k)\cdot o)$ allerdings wesentlich niedriger aus als der für die Untersuchung aller möglichen Kombinationen.

Schließlich können alle Merkmalsselektoren mit allen Klassifikatoren kombiniert und getestet werden. Der Vorteil dieser Methode liegt auf der Hand: Das optimale Klassifikationssystem (im Sinne der Einschränkungen) wird sicher bestimmt. Die Komplexität beträgt $O(s\cdot k\cdot o)$ und kann in der Praxis zu wochenlangen Laufzeiten führen, wie in Kapitel 6 dokumentiert ist. Diese Synthesemethode ist daher für große Klassifikationsprobleme mit vielen tausend Mustern ungeeignet.

Kombination von Klassifikatoren mit Merkmalsselektoren bei Erfolgsaussicht

Die Hinzunahme eines Merkmalsselektors zu einem Klassifikator kann den Generalisierungsfehler entscheidend reduzieren [Witten, 2000]. Die Kombinationen aller Klassifikatoren mit allen möglichen Merkmalsselektoren ist allerdings sehr zeitaufwendig. Aus diesem Grund sollten nur „erfolgversprechende“ Klassifikatoren mit einem Merkmalsselektor kombiniert werden.

Wie definiert sich „erfolgversprechend“? Liegt der Generalisierungsfehler e_g eines Klassifikators unter einer als Parameter vorgegebenen, zu erreichenden Fehlerschwelle *S,* kann der Test weiterer Klassifikatoren abgebrochen und die Synthese vorzeitig beendet werden. Liegt der Generalisierungsfehler auch nur minimal oberhalb dieser Schwelle, wird der Klassifikator verworfen. In diesem Fall könnte die Fehlerschwelle durch eine vorhergehende Merkmalsselektion möglicherweise erreicht werden. Ein Klassifikator wird als erfolgversprechend definiert, wenn gilt:

$$e_g \leq S+\varepsilon \qquad \textbf{Gl. 19}$$

wobei ε entweder als absoluter Wert vorgegeben wird oder aus S berechnet werden kann. Ist die Bedingung aus Gleichung 19 erfüllt, werden zufällig gewählte Merkmalsselektoren mit dem Klassifikator kombiniert. Wird dabei die Schwelle unterschritten, endet die Synthese, andernfalls wird mit dem nächsten Klassifikator fortgesetzt. Die Komplexität dieser Methode beträgt $O(s\cdot k\cdot o)$. Wird die

Schwelle S jedoch günstig gewählt, kann diese Synthesemethode in der Praxis schnell zu einem Ergebnis führen.

Liste mit Erfolgsaussicht bestimmter Kombinationen

Manche Klassifikatoren oder Merkmalsselektoren führen erfahrungsgemäß häufiger zu guten Ergebnissen als andere. Klassifikatoren und Merkmalsselektoren werden deshalb nicht zufällig, sondern in einer vorgegebenen Reihenfolge verwendet. Dieses Vorgehen entspricht in etwa dem eines Experten mit grundlegenden Kenntnissen über Klassifikationssysteme. Beispielsweise liefert ein Naive-Bayes-Klassifikator generell (jedoch nicht immer) gute Ergebnisse [Witten, 2000]. Bei komplexeren Verteilungen der Merkmale liefert ein k-nearest-Neighbour Algorithmus oftmals gute Ergebnisse. Beide Klassifikatoren sollten aus diesem Grund zuerst getestet werden. Für die Merkmalsselektion kann beispielsweise die *stepwise-forward-selection* und die *stepwise-backward-elemination* verwendet werden, da diese beiden Methoden mit minimalem Berechnungsaufwand gute Ergebnisse liefern [Witten, 2000]. Bei einer Zeitbegrenzung werden so die erfolgversprechendsten Algorithmen zuerst verwendet. Eine Aufzählung von Klassifikatoren, beginnend mit den aussichtsreichsten, ist beispielsweise: Naive-Bayes, k-nearest-Neighbour, Support-Vector-Machines, Entscheidungsbäume, Entscheidungstabellen, Fisher-Diskriminanzanalyse, Neuronale Netze, Single-Feature und no-Data. Naive-Bayes beziehungsweise k-nearest-Neighbour als aussichtsreichste Klassifikatoren werden in der Literatur beschrieben [Witten, 2000], der Rest der Liste folgt aus eigenen Beobachtungen und kann variieren. Ein Nachteil besteht in der „festen Verdrahtung“ der Reihenfolge. Neu hinzugekommene Klassifikationsalgorithmen müssen manuell in die Liste eingetragen werden. Ist keine Einschätzung der Klassifikatoren möglich, werden sie ans Ende der Liste gestellt. Die Einordnung bislang unbekannter Klassifikatoren ans Ende der Liste kann auch automatisch erfolgen. Die Liste kann automatisch aktualisiert werden, wenn der Erfolg verwendeter Klassifikatoren kontinuierlich protokolliert wird. Die Komplexität entspricht einer vollständigen Kombination aller möglichen Klassifikatoren und Selektoren $O(s \cdot k \cdot v)$. Diese Methode kommt aber mit den verschiedenen Abbruchkriterien aus Abschnitt 4.1.5 schneller zu einem guten Ergebnis.

Liste mit Komplexität bestimmter Kombinationen

Eine Reihenfolge der zu synthetisierenden Systeme kann auch anhand der jeweiligen Komplexität vorgegeben werden. Vorausgesetzt wird, daß die generierten Klassifikationssysteme untereinander und von dem entsprechenden Klassifikationsproblem unabhängig sind[1]. Im Mittel ist der Zeitaufwand

1 *Synthetisierte Klassifikationssysteme sind in der Praxis meist in irgendeiner Form von der jeweiligen Klassifikationsaufgabe abhängig*

für die Synthese genau dann am geringsten, wenn die Klassifikationssysteme in aufsteigender Reihenfolge der für ihre Synthese benötigten Zeit synthetisiert werden. Die Komplexität unbekannter Klassifikatoren und Merkmalsselektoren zur Laufzeit zu ermitteln ist nahezu unmöglich. Aus diesem Grund müssen die Algorithmen mit einer Meta-Information über ihre jeweiligen Komplexität ausgestattet werden. Die Reihenfolge kann mit den jeweiligen Anforderungen variieren. Eine mögliche Aufzählung ist beispielsweise: no-Data, k-nearest-Neighbour, Fisher-Diskriminanzanalyse, Naive-Bayes, Entscheidungstabellen, Entscheidungsbäume, Support-Vector-Machines und Neuronale Netze [Witten, 2000]. Algorithmen ohne Meta-Information werden grundsätzlich zuletzt verwendet. In die Liste können weitere Kriterien wie beispielsweise der Speicheraufwand einfließen. Die Komplexität der Methode beträgt $O(s \cdot k \cdot o)$.

4.1.3 Reduktion von Merkmalen und Mustern

Reduktion der Merkmalsanzahl

Eine Merkmalsselektion wird vor allem deshalb durchgeführt, um redundante oder irrelevante Merkmale zu entfernen. Gleichzeitig reduziert sich dadurch auch der Aufwand für das Training des Klassifikators. Generell sollte mit einer möglichst geringen, aber ausreichenden Anzahl von Merkmalen eine Synthese durchgeführt werden. Ein geeignetes Verfahren, den Merkmalsraum dahingehend zu transformieren, wurde in Abschnitt 3.5 vorgestellt: die Hauptkomponentenanalyse (PCA). Die transformierten Merkmale sind entsprechend ihrer Nützlichkeit (Varianz) geordnet. Über einen simplen Schwellwert kann die Zahl der Merkmale im Vorfeld eingeschränkt werden. Bei der folgenden Synthese werden Kombinationen aus Merkmalsselektoren und Klassifikatoren nur auf die transformierten Merkmale angewendet. Der Aufwand für die Synthese kann somit problemgebunden reduziert werden.

Reduktion der Musteranzahl

Die Anzahl der Muster ist für die Komplexität der Synthese bedeutend, da sie einerseits linear in eine vollständige Kreuzvalidierung eingeht und andererseits zumindest linear in die Synthese jedes einzelnen Klassifikationssystems. Dadurch verursacht sie wenigstens einen quadratischen Aufwand. Prinzipiell ist in allen Mustern ein Teil der Gesamtinformation enthalten, weshalb alle Muster für die Synthese eines optimalen Klassifikationssystems verwendet werden müssen. Die Idee zur Verminderung des Aufwandes besteht darin, anhand einer reduzierten Mustermenge das beste Klassifikationssystem zu ***schätzen***. Anschließend wird das Klassifikationssystem mit allen zur Verfügung stehenden Mustern trainiert. Ein solches Vorgehen ist genau dann sinnvoll, wenn die folgenden beiden Punkte erfüllt sind:

- der Aufwand für die Schätzung muß geringer ausfallen als der Aufwand für das Training aller möglichen Systeme mit allen Mustern

- die Schätzung muß mit hoher Zuverlässigkeit das tatsächlich beste System bestimmen

Ein konkreter Algorithmus auf Basis dieser Methode ist die Blasensynthese (Bubble Synthesis). Aufgrund ihrer Bedeutung wird der Blasensynthese ein eigener Abschnitt 4.2 gewidmet.

4.1.4 Wahl der Methode für die Fehlerabschätzung

Ein kritischer Punkt bei der Synthese von Klassifikationssystemen ist die Abschätzung des Generalisierungsfehlers, da diese für jedes synthetisierte System separat durchgeführt werden muß. In der Praxis stehen für ein einfaches Hold-Out-Verfahren selten ausreichend Muster zur Verfügung. Deshalb müssen rechenintensive Algorithmen wie Kreuzvalidierung oder Bootstrap-Verfahren eingesetzt werden. Für kleine Klassifikationsprobleme mit nur wenigen Dutzend Mustern werden die genauesten Testfehlerschätzungen durch die Bootstrap-Verfahren oder eine vollständige Kreuzvalidierung erreicht. Der Aufwand für eine vollständige Kreuzvalidierung ist linear abhängig von der Anzahl der Muster $O(o)$. Zunehmend ungenauere Abschätzungen liefern zehnfache oder dreifache Kreuzvalidierung. Der Gewinn dieser Verfahren liegt in ihrem konstanten Aufwand $O(1)$. Für Bootstrap-Verfahren werden in der Praxis 200 Wiederholungen empfohlen, weshalb ihre Komplexität ebenfalls als $O(1)$ einzustufen ist [Weiss, 1991]. In der Praxis sind diese Verfahren bei Klassifikationsproblemen mit wenigen dutzend Mustern bedingt durch ihre rechenintensiven mathematischen Operationen jedoch erheblich zeitaufwendiger als eine vollständige Kreuzvalidierung.

Ist eine möglichst genaue Schätzung des Generalisierungsfehlers notwendig, muß der Aufwand für eine vollständige Kreuzvalidierung oder ein Bootstrap-Verfahren in Kauf genommen werden. Bei großen Klassifikationsproblemen ab mehreren hundert Mustern kann die Synthese von Klassifikationssystemen durch zehnfache Kreuzvalidierung jedoch um Größenordnungen beschleunigt werden. Welches Verfahren zur Testfehlerschätzung am günstigsten ist, hängt von der Größe der jeweiligen Klassifikationsaufgabe ab.

4.1.5 Kriterien für vorzeitigen Abbruch der Synthese

Für die Synthesemethoden kann eine Reihe von Abbruchkriterien definiert werden. Das Klassifikationssystem mit dem bis zu diesem Zeitpunkt niedrigsten Generalisierungsfehler wird als Ergebnis verwendet. Die folgenden wurden betrachtet:

- Die Synthese endet, wenn alle Kombinationen aus Merkmalsselektoren und Klassifikatoren verwendet wurden.
- Die Synthese endet, wenn sie explizit durch einen interaktiven Eingriff von außen abgebrochen wird.
- Die Synthese endet, nachdem eine maximale Zeitspanne verstrichen ist.
- Die Synthese endet, wenn der Generalisierungsfehler eine Schwelle *S* unterschreitet.

Diese Abbruchkriterien haben keinen Einfluß auf die Komplexität einer Synthesemethode.

Abbruchkriterium Wahrscheinlichkeitsschwelle

Viele Kombinationen aus Merkmalsselektoren und Klassifikatoren liefern für das gleiche Klassifikationsproblem ähnliche Generalisierungsfehler. Wird ein Klassifikationssystem mit einem Generalisierungsfehler nahe dem besten Generalisierungsfehler synthetisiert, lohnt sich der Aufwand für die Suche nach dem besten Klassifikationssystem oftmals nicht mehr und die Synthese kann vorzeitig beendet werden.

Der beste erreichbare Generalisierungsfehler ist unbekannt, solange nicht alle möglichen Kombinationen aus Merkmalsselektoren und Klassifikatoren getestet wurden. Die Distanz des Generalisierungsfehlers einer bestimmten Kombination zu der besten kann während der Synthese nicht ermittelt werden. Man kann jedoch die Wahrscheinlichkeit $P(X)$ berechnen, mit der ein synthetisiertes Klassifikationssystem zu der Gruppe der *r* besten Klassifikationssysteme gehört, wenn jede zufällig ausgewählte Kombination mit der annähernd gleichen Wahrscheinlichkeit die beste sein kann. *N b*ezeichne die Anzahl der zu einem bestimmten Zeitpunkt bereits synthetisierten Klassifikationssysteme. In der Gesamtmenge der möglichen Klassifikationssysteme befinden sich *R* mögliche Kombinationen. Sor-

tiert man alle Klassifikationssysteme aufsteigend nach ihrem Generalisierungsfehler, bezeichnet man die ersten *r* Systeme als die Gruppe der bestmöglichen Klassifikationssysteme. Dieser Wert *r* wird als Parameter vorgegeben. Die Wahrscheinlichkeit $P(X)$, daß sich *x* Elemente von *r* möglichen unter *N* aus *R* zufällig gezogenen Elementen befinden, folgt einer hypergeometrischen Verteilung. Diese hat die Wahrscheinlichkeitsfunktion:

Gl. 20

$$P(X=x) = \frac{\binom{r}{x}\cdot\binom{R-r}{N-x}}{\binom{R}{N}}$$

Die Synthese soll enden, wenn sich nach *N* synthetisierten Klassifikationssystemen mindestens ein Klassifikationssystem $(x \geq 1)$ mit einer als Parameter vorgegebenen Wahrscheinlichkeit *W* unter den besten *r* Klassifikationssystemen befindet:

Gl. 21

$$W \leq P(x \geq 1)$$

mit

Gl. 22

$$P(x \geq 1) = 1 - P(0)$$

Aus Gleichung 20, Gleichung 21 und Gleichung 22 erhält man

Gl. 23

$$W \leq 1 - \prod_{i=1}^{r}\left(1 - \frac{N}{R-i+1}\right)$$

Die Synthese wird beendet, wenn die Wahrscheinlichkeitsschwelle *W* überschritten wird. Schränkt man die Menge der besten Klassifikationssysteme auf genau das eine optimale Klassifikationssystem ein, ist $r=1$ und Gleichung 20 degeneriert zu einer Gleichverteilung. Die Wahrscheinlichkeit *W* ist in diesem Fall linear abhängig von der Anzahl *N* der bereits synthetisierten Klassifikationssysteme:

Gl. 24

$$W \leq \frac{N}{R}$$

Die Wahrscheinlichkeitsschwelle kann in allen Synthesemethoden eingesetzt werden, die zufällig und gleichverteilt Klassifikationssysteme synthetisieren. Die Komplexität der Synthesemethode ändert

sich dadurch nicht. Im Gegensatz zu der absoluten Beschränkung der Synthesezeit wird die Wahrscheinlichkeitsschwelle relativ zu der Anzahl der bisher synthetisierten Klassifikationssysteme ausgewertet.

4.1.6 Verteilte Synthese

Einen völlig anderen Ansatz stellt die massive verteilte Synthese von Klassifikationssystemen dar. Dies ist möglich, da die Kombinationen verschiedener Klassifikatoren und Merkmalsselektoren unabhängig voneinander sind. Die verteilte Synthese kann als eigenständige Methode eingesetzt werden. Stehen genügend Verarbeitungseinheiten zur Verfügung, kann parallel jedes mögliche Klassifikationssystem erstellt und dessen Generalisierungsfehler geschätzt werden. Auch können die meisten vorgestellten Synthesemethoden von einer verteilten Verarbeitung profitieren. In real existierenden lokalen Netzwerken bestehend aus 20 bis 200 Rechnern kann dadurch ein Zeitgewinn linear zu der Anzahl der Recheneinheiten um zwei oder mehr Größenordnungen erreicht werden. Seit kurzem wird in der Informatik auch an der Verknüpfung globaler Rechnernetzwerke zu einem sogenannten Grid mit zehntausenden von Recheneinheiten geforscht [The Global Grid Forum, 2005]. Der Zugriff erfolgt über transparente Schnittstellen in Form von sogenannten Webservices [Cerami, 2002]. Allein im Forschungszentrum Karlsruhe werden bis zum Jahre 2007 für physikalische Hochenergieexperimente über 4000 Pentium PCs von 1 und 3 GHz sowie 2 Petabyte an Festplattenplatz zur Verfügung gestellt. Von diesen sind zu Beginn 2005 bereits 1000 PCs installiert [Alef, 2004]. Ein solches Grid würde sich hervorragend eignen, um damit verteilt Klassifikationssysteme zu synthetisieren. Notwendig für eine Synthese auf dem Grid wäre die Implementierung eines entsprechenden Webservices. Die Verwaltung und Kommunikation werden bereits von einer sogenannten Middleware zur Verfügung gestellt. Verschiedenartige Middleware wird gerade entwickelt, so beispielsweise das Globus Toolkit [The Globus Alliance, 2005]. Aufgrund der sich noch in der Entwicklung befindlichen Middleware und dem zur Zeit auf Hochenergieexperimente beschränkten Zugang konnten leider noch keine Versuche in Richtung Grid unternommen werden.

Bei der Umsetzung von verteilten Synthesemethoden entsteht ein Verwaltungsoverhead durch die Koordination der Arbeitspakete und der Kommunikation zwischen den Verarbeitungseinheiten. Die Komplexität $O(o)$ wird durch die Schätzung des Generalisierungsfehlers verursacht. Für das Zusammenführen der verteilten Arbeitspakete ist in der Praxis der Aufwand $O(1)$ anzusetzen, da hierzu nur die Generalisierungsfehler miteinander verglichen werden müssen. Die Synthesezeit wird durch den maximalen Zeitaufwand für die Synthese der lokalen Klassifikationssysteme bestimmt.

4.2 Blasensynthese (Bubble Synthesis)

Die Blasensynthese stellt eine konkrete Umsetzung der im vorigen Abschnitt beschriebenen Synthesemethode durch Reduktion der Mustermenge dar. Sie ist nur eine von vielen Umsetzungsmöglichkeiten des zugrunde liegenden Prinzips. Der Name ruht von der Tatsache her, daß innerhalb einer Liste aller möglichen Klassifikationssysteme das beste Klassifikationssystem in jedem Iterationsschritt wie eine Luftblase immer weiter „aufsteigt“, bis es im letzten Schritt ganz oben an der Spitze steht. Empirische Beobachtungen haben gezeigt, daß sich mit der Blasensynthese in drei Iterationsschritten gute Ergebnisse erreichen lassen. Es sind jedoch Verfahren mit einer anderen Anzahl von Iterationsschritten denkbar.

4.2.1 Algorithmus der Blasensynthese

Die Blasensynthese wird im folgenden anhand von drei einander ähnelnden Iterationsschritten erklärt. Im ersten Schritt werden alle n_1 möglichen Kombinationen aus Klassifikatoren und Merkmalsselektoren mit Hilfe einer „geringen“ Untermenge von Mustern synthetisiert und diese anschließend entsprechend ihres Generalisierungsfehlers aufsteigend sortiert. Aufgrund der stark eingeschränkten Menge an Mustern ist die Schätzung des Generalisierungsfehlers ungenau, weshalb das oberste Klassifikationssystem in der Liste nicht zwangsläufig das tatsächlich beste System ist. Der Liste ist jedoch die „Tendenz“ zu entnehmen, welche Kombinationen besonders vielversprechend erscheinen. Im nächsten Schritt wird diese Tendenz verwendet, indem aus der erstellten Liste die besten n_2 Klassifikationssysteme ausgewählt werden. Diese besten n_2 Klassifikationssysteme werden mit einer größeren Untermenge an Mustern trainiert und in einer zweiten Liste aufsteigend nach Generalisierungsfehler sortiert. Die zweite Liste ist aufgrund der größeren Anzahl verwendeter Muster zuverlässiger als die erste und verspricht bessere Prognosen hinsichtlich der Tendenz. Dafür müssen im zweiten Schritt nicht mehr alle möglichen Klassifikationssysteme getestet werden, sondern nur noch die ersten n_2 Kombinationen der ersten Liste. Im dritten und letzten Schritt werden alle Muster für die Synthese verwendet, wobei nur noch die besten n_3 Systeme der zweiten Liste verwendet werden.

Wie findet sich für diese Methode eine geeignete Untermenge an Mustern? Muster müssen zufällig aus der Gesamtmenge ausgewählt werden. Fließt eine zu geringe Anzahl von Mustern in die Synthese ein, wird die Tendenz nur unzureichend geschätzt. Bei einer zu großen Untermenge von Mustern ist die Verringerung des Aufwandes gegenüber einer systematischen Synthese aller möglichen Klassifikationssysteme nicht ausreichend. Zudem ist die Anzahl der Muster, welche für eine „statistische Signi-

fikanz“ notwendig ist, von der Anzahl der verwendeten Merkmale abhängig. Daher sollte die allgemeine Faustregel eines Verhältnisses von zehn zu eins der Anzahl an Mustern gegenüber Merkmalen beachtet werden, obwohl sie nicht allgemein gültig ist [Fukunaga, 1990]. Desweiteren wird die Anzahl der Muster mit *o,* die Anzahl der Merkmale insgesamt mit *m* und die Anzahl der verwendeten Muster in Schritt *i* mit o_i bezeichnet. Unter den verschiedenen Voraussetzungen haben sich für die Blasensynthese die folgenden Parameterwerte experimentell bewährt:

o ~ Muster
o_i ~ Submuster
m ~ Merkmale
n_i ~ Kombinationen

a.) *m* klein und *o* klein, falls $o > 10 \cdot m$
$\Rightarrow \quad o_1 = m, \quad o_2 = 10 \cdot m, \quad o_3 = o$

b.) *m* groß und *o* klein, falls $o > m$ aber $o < 10 \cdot m$
$\Rightarrow \quad o_1 = m, \quad o_2 = o$

c.) *m* klein und *o* groß, falls $o > 100 \cdot m$
$\Rightarrow \quad o_1 = 10 \cdot m, \quad o_2 = 100 \cdot m, \quad o_3 = o$

d.) *m* groß und *o* groß, falls $o > 10 \cdot m$
wie a)

e.) *o* extrem klein, falls $o < m$
$\Rightarrow \quad o_1 = o$, keine Reduktion

Die Werte für n_2 und n_3 wurden empirisch ermittelt. Geeignete Werte für n_2 liegen zwischen *10* und *30*, Werte für n_3 zwischen $\frac{n_2}{3}$ und $\frac{n_2}{10}$ mit $1 < n_3 < 10$. Besondere Behandlung erfordern einige Spezialfälle wie beispielsweise eine höhere Anzahl von Klassen als Merkmale. In diesen Fällen wird die Anzahl der Submuster geeignet angepaßt.

Die Komplexität der Blasensynthese für den Fall a), c) oder d) beläuft sich auf $O(s \cdot k \cdot m)$ im ersten Schritt bei vollständiger Kreuzvalidierung. Der zweite Schritt besteht im wesentlichen aus dem Training einer konstanten Anzahl von Klassifikationssystemen mit zugehöriger Kreuzvalidierung und hat eine Komplexität von $O(m)$. Der letzte Schritt trainiert eine konstante Anzahl von Systemen mit allen Mustern für eine Kreuzvalidierung und hat einen Aufwand linear zur Anzahl der Muster $O(o)$. Die

Synthesealgorithmus	Komplexität
Zufällige Klassifikatorwahl	$O(o)$
Wiederholte zufällige Klassifikatorwahl	$O(k \cdot o)$
Zufällige Klassifikatorwahl mit Selektor	$O(o)$
Wiederholte zufällige Klassifikatorwahl mit Selektor	$O(k \cdot o)$
Wiederholte zufällige Klassifikatorwahl mit wiederholter Selektion	$O((s+k) \cdot o)$
Vollständige zufällige Klassifikatorwahl mit Selektor	$O(s \cdot k \cdot o)$
Wiederholte zufällige Klassifikatorwahl mit Selektor bei Testfehlernähe	$O(s \cdot k \cdot o)$
Simple Parallelverarbeitung aller möglichen Kombinationen	$O(o)$
Zufallssuche von Selektoren bei Vorgabe stabiler Klassifikatoren	$O(s \cdot o)$
Zufallssuche von Klassifikatoren bei Vorgabe stabiler Selektoren	$O(k \cdot o)$
Vollständige Liste von Klassifikatoren nach Erfolgsaussicht	$O(s \cdot k \cdot o)$
Vollständige Liste von Klassifikatoren nach Komplexität	$O(s \cdot k \cdot o)$
Reduktion der Merkmalsanzahl	$O(s \cdot k \cdot o)$
Reduktion der Musteranzahl durch Blasensynthese	$O(s \cdot k \cdot m + o)$

Tabelle 2. Komplexitätsübersicht verschiedener Synthesealgorithmen bei vollständiger Crossvalidierung. Die Anzahl der Klassifikatoren wird mit k bezeichnet, die Anzahl der Selektoren mit s, die Anzahl der Muster mit o und die Anzahl der Merkmale mit m.

Gesamtkomplexität des Algorithmus beträgt $O(s \cdot k \cdot m + o)$. Der Vorteil dieser Methode wirkt sich besonders dann aus, wenn m erheblich kleiner als o ist. Dies ist für viele Klassifikationsprobleme der Fall. Tabelle 2 stellt die Komplexität der Blasensynthese denen der anderen Synthesemethoden gegenüber.

Vernachlässigt man die simple Parallelverarbeitung aller Kombinationen, dann hat die Blasensynthese die niedrigste Komplexität aller Verfahren, die sämtliche Kombinationsmöglichkeiten von Klassifikatoren und Merkmalsselektoren berücksichtigen. Sie sollte somit die effektivste Synthesemethode für Klassifikationssysteme sein. Es stellt sich jedoch die Frage, ob die Blasensynthese tatsächlich in jedem Fall (oder zumindest in den meisten Fällen) ein gutes Klassifikationssystem erstellt.

4.2.2 Konvergenz zum besten Klassifikationssystem

Die Korrektheit der Blasensynthese läßt sich zunächst intuitiv nachvollziehen: Erstellt man alle möglichen Klassifikationssysteme mit allen zur Verfügung stehenden Mustern und sortiert diese aufsteigend in einer Liste nach ihrem Generalisierungsfehler, befindet sich das beste System definitionsgemäß an der Spitze. Entfernt man ***ein*** Muster aus der ursprünglichen Menge und wiederholt das Verfahren, behalten die Klassifikationssysteme innerhalb der Liste sehr wahrscheinlich ihre Positionen bei. Das beste Klassifikationssystem wird höchstwahrscheinlich immer noch an erster Stelle stehen. Entfernt man weitere Muster, wird diese Wahrscheinlichkeit abnehmen. Das beste Klassifikationssystem kann schließlich seinen ersten Platz in der Liste verlieren. Es stellt sich die Frage, wie viele

Muster auf diese Weise entfernt werden können, bis das beste Klassifikationssystem nicht mehr unter den *n* ersten Systemen der Liste liegt.

Eine allgemeine Formel anzugeben, mit welcher Wahrscheinlichkeit die Blasensynthese das beste Klassifikationssystem für beliebige Klassifikationsprobleme synthetisiert, ist unmöglich. Die ursprüngliche Aufgabe wäre gelöst und die Synthese von Klassifikationssystemen kein NP-vollständiges Problem. Aus einer Untermenge an Mustern kann nicht konkret das beste Klassifikationssystem für die Menge aller Muster bestimmt werden. Die Kernaussage der Blasensynthese besteht darin, aus einer Untermenge die Zusammensetzung des besten Klassifikationssystems für die Gesamtmenge zu *schätzen*. Der durch die Schätzung auftretende Unsicherheitsfaktor wird kompensiert, indem schrittweise die Stichprobengröße verbessert wird. Dazu wird „sicherheitshalber“ in jedem Schritt anstelle des momentan besten Klassifikationssystems die Menge der *n* besten Systeme weiterverarbeitet.

Für ein spezielles Klassifikationsproblem kann die Wahrscheinlichkeit, daß die Blasensynthese das beste Klassifikationssystem synthetisiert, bestimmt werden. Hierzu muß man zunächst das beste Klassifikationssystem durch Kombination aller Selektoren und Klassifikatoren als Referenz ermitteln. Für eine Blasensynthese aus drei Schritten werden danach alle möglichen Untermengenkombinationen mit o_1, o_2 und o_3 aufgezählt. Für alle Untermengen werden jeweils Kombinationen aus Selektoren und Klassifikatoren gebildet und alle Möglichkeiten, die während der Blasensynthese auftreten können, systematisch durchsucht. Dadurch läßt sich für ein konkretes Klassifikationsproblem genau bestimmen, mit welcher Wahrscheinlichkeit bestimmte Klassifikationssysteme synthetisiert werden und wie nahe deren Generalisierungsfehler der Referenz kommen. Durch die vielen Kombinationsmöglichkeiten beläuft sich die Komplexität des Ansatzes auf:

$$O\left(s \cdot k \cdot o \cdot \binom{o}{o_1} \cdot \binom{o}{o_2}\right) \qquad \textbf{Gl. 25}$$

Die Komplexität aus Gleichung 25 übersteigt bei weitem die Komplexität einer vollständigen Suche nach dem besten Klassifikationssystem. Mit diesem Verfahren läßt sich auch immer nur die Wahrscheinlichkeit für ein bestimmtes Klassifikationsproblem bestimmen. Aus diesem Grund wurde ein anderer Weg beschritten, um die Konvergenz der Blasensynthese zu dem besten Klassifikationssystem nachzuweisen.

Nachweis mit Hilfe künstlicher Klassifikationsprobleme

Verwendet man ein reales Klassifikationsproblem, sind sowohl der Generalisierungsfehler als auch die Zusammensetzung des besten Klassifikationssystems in den allermeisten Fällen nicht bekannt. Ein durch die Blasensynthese synthetisiertes Klassifikationssystem kann nicht beurteilt werden, da kein bestes Klassifikationssystem als Referenz bereitsteht. Die einzige Möglichkeit, ein „Referenz-Klassifikationssystem“ zu erstellen, besteht in der systematischen Kombination aller Klassifikatoren und Selektoren. Das entspricht aber genau der ursprünglichen Problemstellung, die aus Komplexitätsgründen undurchführbar ist.

Die Idee einer empirischen Verifikation besteht darin, künstliche Klassifikationsprobleme mit genau definierten Verteilungen der Merkmale und vorgegebenem Klassifikationsfehler zu generieren. Die Kenntnis der Merkmalsverteilungen gibt Aufschluß über potentiell geeignete Klassifikatoren und ermöglicht somit eine Beurteilung eines synthetisierten Systems gegenüber einem Referenzsystem.

In den Entwurfsprozeß künstlicher Klassifikationsprobleme gehen vier Parameter ein: die Anzahl der Muster o, die Anzahl der Merkmale m, die Anzahl der Klassen c und der angestrebte Generalisierungsfehler e_g. Eine Vereinfachung besteht in der Forderung nach Unabhängigkeit der Merkmale. Der Gesamtfehler wird gleichmäßig auf alle Merkmale verteilt. Weiterhin werden o.B.d.A. gleiche, normierte Verteilungen der Merkmale je Klasse und eine Gleichverteilung der Muster auf alle Klassen angenommen. Um solche künstlichen Klassifikationsprobleme zu generieren, wurden zwei Verfahren entwickelt und implementiert, die im folgenden vorgestellt werden.

Klassenweise gleichverteilte Merkmale

Klassenweise gleichverteilte Merkmale sind verhältnismäßig leicht zu generieren, weshalb sie zur systematischen Untersuchung eingesetzt wurden. Zur Vereinfachung wird angenommen, daß jedes Merkmal für jede Klasse in einem Intervall der Länge eins gleichverteilt ist. Alle Merkmale spannen für jeweils eine Klasse einen Hyperwürfel mit der Kantenlänge eins auf. Im Merkmalsraum befinden sich somit c Hyperwürfel, welche sich auch überlappen können. In diesen Bereichen ist die Klassenzugehörigkeiten für alle überlappenden Klassen gleich wahrscheinlich und eine eindeutige Zuordnung eines Musters zu einer Klasse ist unmöglich. Wird in jedem dieser Bereiche generell zugunsten einer der beteiligten Klassen entschieden, werden die Muster der anderen Klassen falsch zugeordnet und führen zu dem Klassifikationsfehler. Dieser Fehler entspricht dem Volumen der Überlappung und der

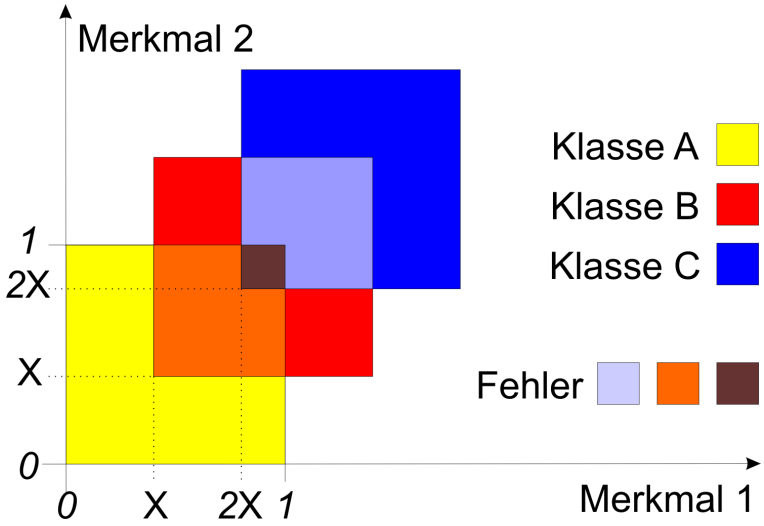


Abbildung 5. Klassifikationsproblem mit zwei gleichverteilten Merkmalen zu drei Klassen. Die klassenweisen Verteilungen sind je um ein Offset x verschoben. Der minimal mögliche Fehler eines solchen Klassifikationsproblems ist die Summe der orangenen Fläche, der violetten Fläche und zwei mal der braunen Fläche.

Gesamtfehler entspricht dem Volumen aller Überlappungen. Abbildung 5 zeigt dies beispielhaft im zweidimensionalen Fall für zwei Merkmale und drei Klassen.

Um das Klassifikationsproblem zu erstellen, werden Merkmale zunächst in einem Intervall von *0* bis *1* gleichverteilt generiert. Der Intervallbeginn der nächsten Klasse wird jeweils um ein Offset verschoben. Das Offset bestimmt man aus dem gewünschten Fehler des Klassifikationsproblems, der durch das Inhaltsverhältnis von Hyperwürfel-Überlappungen $I_{Ü}$ zu allen Hyperwürfeln I_H definiert ist:

$$e_g = \frac{I_{Ü}}{I_H} \qquad \textbf{Gl. 26}$$

Der Inhalt aller Hyperwürfel entspricht aufgrund einer Kantenlänge von eins

$$I_H = 1^m \cdot c \qquad \textbf{Gl. 27}$$

Der Inhalt der Hyperwürfel-Überlappung entspricht der Summe der Überlappung von je zwei aufeinanderfolgenden Klassen. Mit einer Kantenlänge der Hyperwürfel-Überlappung von *1-x* ergibt sich

$$I_{Ü} = (c-1)\cdot(1-x)^m \qquad \textbf{Gl. 28}$$

Aus Gleichung 26, Gleichung 27 und Gleichung 28 ergibt sich zur Berechnung des Offsets zwischen den Merkmalsverteilungen

Gl. 29

$$x = 1 - \sqrt[m]{\frac{c \cdot e_g}{(c-1)}}$$

Mit Hilfe von Gleichung 29 ergibt sich der Algorithmus zur Synthese künstlicher Klassifikationsprobleme mit klassenweise gleichverteilten Merkmalen. Der Pseudo-Code ist im Anhang abgedruckt.

Klassenweise normalverteilte Merkmale

Ereignisse natürlicher Prozesse sind häufig normalverteilt. Aus diesem Grund wurden auch künstliche Klassifikationsprobleme mit klassenweise normalverteilten Merkmalen erstellt, da diese reale Stichproben besser repräsentieren. Bei klassenweise normalverteilten Merkmalen ist zusätzlich zum Generalisierungsfehler auch bekannt, daß der Bayes-Klassifikator der bestmögliche Klassifikator ist, denn die verwendete Implementierung des Bayes-Klassifikator setzt für die klassenweisen Wahrscheinlichkeitsdichten Normalverteilungen voraus.

Zum besseren Verständnis des Verfahrens zeigt Abbildung 6 ein eindimensionales Klassifikationsproblem mit drei Klassen und einem klassenweise normalverteilten Merkmal. Die Verteilungsfunktionen der drei Klassen überlappen sich. Bei der Klassifizierung eines Musters entscheidet man sich für die Klasse, für die an der Stelle des gemessenen Merkmalswertes der Funktionswert der Verteilungsfunktion am größten ist. Die Muster in den überlappenden Bereichen werden dadurch zum Teil falsch klassifiziert. Der Anteil falsch klassifizierter Muster entspricht der Summe $I_{Ü}$ der Flächeninhalte unterhalb der Verteilungsfunktionen, die durch Verteilungsfunktionen mit höheren Funktionswerten „verdeckt“ werden. Der minimal mögliche Generalisierungsfehler e_g bei den gegebenen Verteilungen aus Abbildung 6 ist definiert durch das Verhältnis aus der Summe $I_{Ü}$ zu der Summe I_H der Flächen aller Verteilungsfunktionen:

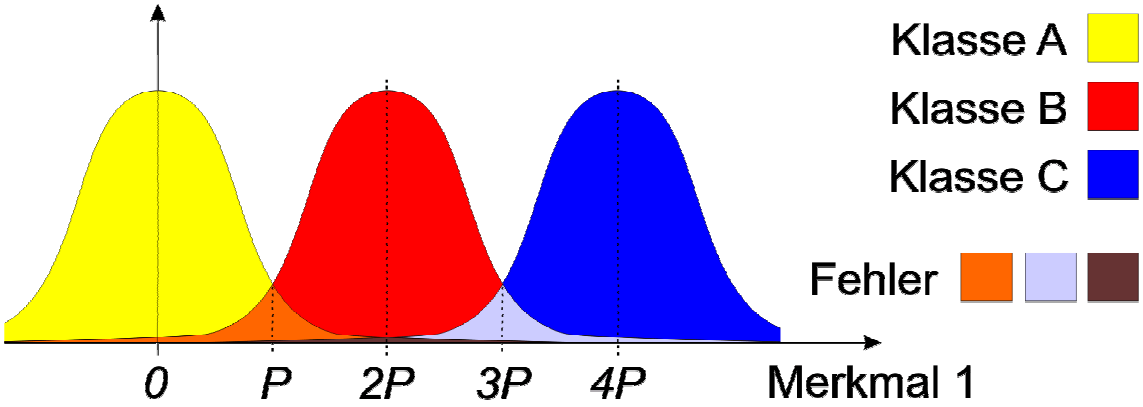


Abbildung 6. Klassifikationsproblem mit einem normalverteilten Merkmal und drei Klassen. Die klassenweisen Verteilungen sind je um ein Offset p verschoben. Der minimal mögliche Fehler eines solchen Klassifikationsproblems ist die Summe der orangenen Fläche plus der violetten Fläche plus zwei mal der braunen Fläche.

$$e_g = \frac{I_{Ü}}{I_H} \qquad \textbf{Gl. 30}$$

Der Flächeninhalt einer Verteilungsfunktion ist eins. Der Flächeninhalt I_H setzt sich aus c Flächen mit dem Inhalt eins zusammen. Zur Vereinfachung nimmt man an, daß sich der Generalisierungsfehler gleichmäßig verteilt. Bei c Klassen setzt sich der Flächeninhalt $I_{Ü}$ aus der Summe von c-1 überlappenden Flächen mit dem Inhalt I zusammen. Aus Gleichung 30 ergibt sich:

$$e_g = \frac{(c-1)\cdot I}{c} \qquad \textbf{Gl. 31}$$

Der Flächeninhalt I ergibt sich aus zwei symmetrischen Teilflächen, die durch das Integral der Normalverteilung bestimmt werden. Ausschlaggebend ist der Schnittpunkt P zwischen den beiden Normalverteilungen in Abbildung 6. Für die Normalverteilung der Klasse A mit Erwartungswert 0 gilt:

$$I = 2 \cdot \int_P^{\infty} \frac{1}{\sqrt{2\pi}\cdot\sigma} e^{-\frac{x^2}{2\cdot\sigma^2}} dx \qquad \textbf{Gl. 32}$$

Die Normalverteilungen sind um das Offset $\bar{x}=2\cdot P$ verschoben. Durch Änderung des Offsets kann der Generalisierungsfehler verändert werden. Bei Vorgabe eines Generalisierungsfehlers ist allerdings das Offset $\bar{x}$ gesucht, das zu genau diesem Generalisierungsfehler führt. Hierzu muß Gleichung 32 numerisch invertiert werden, da es keine explizite inverse Funktion gibt [Beasley, 1977]. Zur Herstellung mehrdimensionaler normalverteilter Klassifikationsprobleme wird wie im Falle der gleichverteil-

ten Merkmale eine Anordnung der Punktwolken gewählt, die sich auf den eindimensionalen Fall zurückführen läßt und bei der gleichzeitig keine Merkmale durch eine Merkmalsselektion entfernt werden. Eine solche Anordnung ist die Plazierung rotationssymmetrischer Punktwolken auf der Hauptdiagonalen mit gleichmäßigem Offset *x*. Die Punktwolken können durch Projektion auf die erste Merkmalsachse in den eindimensionalen Fall überführt werden. Durch die Projektion rücken die Zentren der Punktwolken näher zusammen, was zu einem für den mehrdimensionalen Fall „falschen" Offset $\overline{x}$ führt. Das Offset $\overline{x}$ kann durch einen Korrekturfaktor, welcher sich durch die Projektion aus der Anzahl der Merkmale ergibt, in das tatsächlich notwendige Offset *x* überführt werden:

Gl. 33

$$\overline{x} = 2 \cdot P = \sqrt{\underbrace{x^2 + \ldots + x^2}_{m}} = x \cdot \sqrt{m}$$

Der Pseudo-Code zur Herstellung mehrdimensionaler Klassifikationsprobleme mit klassenweise normalverteilten Merkmalen befindet sich im Anhang.

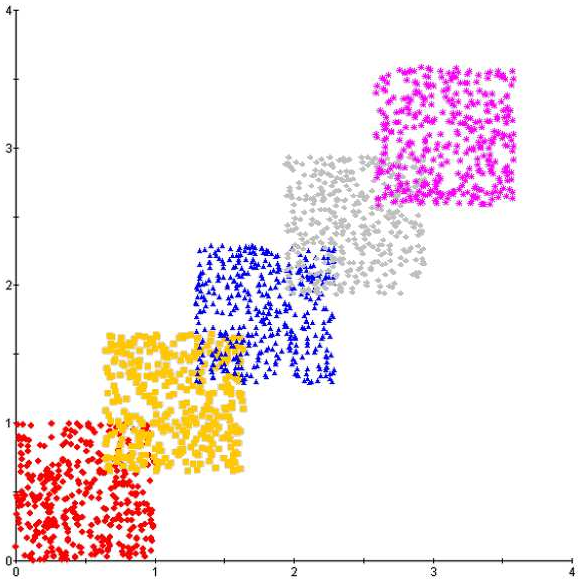


Abbildung 7. Darstellung eines künstlichen Klassifikationsproblems mit zwei klassenweise gleichverteilten Merkmalen und fünf Klassen. Der Testfehler durch überlappende Flächen beträgt zehn Prozent.

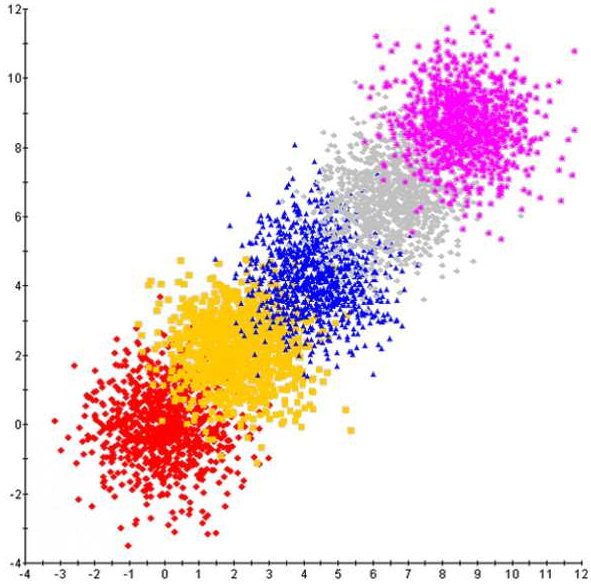


Abbildung 8. Darstellung eines künstlichen Klassifikationsproblems mit zwei klassenweise normalverteilten Merkmalen und fünf Klassen. Der Testfehler durch überlappende Flächen beträgt zehn Prozent.

Beide Algorithmen wurden als Komponenten der entwickelten Komponentensoftware realisiert, wodurch die künstlichen Klassifikationsprobleme in der gleichen Testumgebung wie die realen Klassifikationsprobleme untersucht werden konnten. Die Abbildungen 7 und 8 zeigen ein gleichverteiltes und ein normalverteiltes Klassifikationsproblem, die in der Komponentensoftware erstellt und visualisiert wurden. Mit den Algorithmen wurden sechs repräsentative Klassifikationsprobleme generiert

und mit der Blasensynthese verarbeitet. Die Ergebnisse der Untersuchungen sind in Kapitel 6.1 beschrieben.

4.3 Zusammenfassung

Das optimale Klassifikationssystem für eine gestellte Klassifikationsaufgabe besteht aus einer hierarchischen Kombination von Merkmalsselektoren und Klassifikatoren. Diese optimale Kombination zu bestimmen ist ein NP-vollständiges Problem. Der Entwurf von Klassifikationssystemen kann aus Komplexitätsgründen nicht durch simples Aufzählen und Testen aller Kombinationsmöglichkeiten automatisiert werden.

Eine automatische Synthese von Klassifikationssystemen ist möglich, wenn zwei Einschränkungen zur Begrenzung der Komplexität getroffen werden: Zum einen wird keine Parameteroptimierung der eingesetzten Algorithmen durchgeführt. Zum anderen werden die Kombinationsmöglichkeiten auf genau einen Merkmalsselektor und einen Klassifikator beschränkt. Die Synthese eines Klassifikationssystems wird dadurch auf ein zweidimensionales Optimierungsproblem reduziert.

Auf das resultierende Optimierungsproblem sind herkömmliche Lösungsmethoden nicht anwendbar, da zwischen den Generalisierungsfehlern der Kombinationen keine Nachbarschaftsbeziehungen existieren. Zur Lösung des Optimierungsproblems wurden neue Verfahren entwickelt, die als ***Synthesemethoden*** bezeichnet werden. Die verschiedenen Synthesemethoden unterscheiden sich insbesondere darin, daß sie für eine bestimmte Klassifikationsaufgabe mit zunehmender Komplexität und Laufzeit zunehmend bessere Klassifikationssysteme mit niedrigerem Generalisierungsfehler synthetisieren.

Von allen Synthesemethoden, die sämtliche Kombinationen berücksichtigen, hat die Blasensynthese die niedrigste Komplexität. Sie basiert auf der schrittweisen Schätzung des besten Klassifikationssystems aus Teilmengen von Mustern, wobei sich die Schätzung mit wachsender Anzahl eingesetzter Muster verbessert. Das gesuchte Klassifikationssystem steigt gleich einer Luftblase aus allen möglichen Kombinationen auf.

Für die Erprobung der entwickelten Synthesemethoden müssen diese in einer geeigneten Umgebung mit vielen Klassifikationsalgorithmen und Möglichkeiten für die Rückkopplung implementiert werden. Keines der zur Zeit verfügbaren Systeme ist hierfür geeignet. Aus diesem Grund wurde auf Basis

verfügbarer Implementierungen und Konzepte eine Komponentensoftware für die Klassifikation entwickelt, mit der die Synthesemethode getestet wurden. Diese Komponentensoftware wird im folgenden Kapitel vorgestellt.

5 Komponentensoftware für die Klassifikation

> ***Software components:*** *are binary units of independent production, acquisition and deployment that interact to form a functioning system.*
>
> ***[Szyperski, 1999]***

Keines der zur Zeit verfügbaren Systeme bietet sowohl befriedigende Möglichkeiten für die Klassifikation als auch eine Meta-Ebene, die für eine Automatisierung des Entwurfs von Klassifikationssystemen unerläßlich ist. Auch kann keines der bekannten Systeme derart erweitert werden, daß es die Anforderungen an ein Entwicklungswerkzeug für Klassifikationssysteme erfüllt.

Um dennoch den Entwurf von Klassifikationssystemen in der gewünschten Form zu unterstützen, bleibt nur die Entwicklung eines neuartigen Entwicklungswerkzeuges für die Klassifikation. Verständlicherweise werden dazu vorhandene Implementierungen und Systeme soweit wie möglich genutzt, um den benötigten Aufwand zu minimieren. Unter diesen Vorraussetzungen liegt das Ziel darin, eine quelloffene[1] Komponentensoftware (nicht nur) für die Klassifikation bereit zu stellen, die leicht erweiterbar ist und die deshalb auch in vielen anderen Bereichen eingesetzt werden kann. Dieses Kapitel behandelt die Konzepte und die Implementierung der entwickelten Komponentensoftware.

5.1 Umsetzung der Anforderungen

In Anlehnung an Entwicklungswerkzeuge für konventionelle Programmierung, die als Integrated Development Environment (IDE) bezeichnet werden, wird die entwickelte Komponentensoftware als ***Integrated Component Environment*** oder im folgenden abkürzend mit ***ICE*** bezeichnet.

5.1.1 Verwendete Systeme und Konzepte

Die Komponentensoftware für die Klassifikation wurde auf ***Weka*** aufgebaut. Weka verfügt über die umfangreichste Implementierung von Klassifikationsalgorithmen. Für die Umsetzung als Komponentensoftware wurde die Programmiersprache Java und das Konzept der ***JavaBeans*** gewählt. Für diese

1 *In Form von OpenSource und beispielsweise der GNU-Lizenz*

Wahl spricht insbesondere, daß Weka in Java implementiert ist. Die Integration der Klassifikationsalgorithmen als Komponenten vereinfacht sich durch den Einsatz von Java und JavaBeans erheblich. Java ist plattformunabhängig und erfüllt dadurch implizit eine der gewünschten Anforderungen aus Kapitel 2. Einige Synthesemethoden aus Kapitel 4 erfordern eine objektorientierte Implementierung, die mit Java erfüllt werden kann. Die graphische Benutzeroberfläche wurde nach dem Vorbild von ***Khoros*** gestaltet, da sich dessen Prinzip der visuellen Programmierung bewährt hat. Zudem ist Khoros weit verbreitet und ein „Wiedererkennungseffekt“ der Kontrollstrukturen durchaus erwünscht. Konkrete Implementierungen aus Khoros konnten allerdings nicht verwendet werden.

Bei der Implementierung eines eigenen Konzeptes kann besonders einfach eine ***Meta-Ebene*** bereit gestellt werden, die für die Umsetzung der Synthesemethoden aus Kapitel 4 notwendig ist. Kontrollmöglichkeiten wie das ***self-guided assembly*** für die Manipulation von Komponenten durch sich selbst sind bei der Entwicklung berücksichtigt worden.

5.1.2 Einfache Integration bereits vorhandener Bibliotheken

Ein großer Teil der Merkmalsselektoren und Klassifikatoren aus Weka wurde als Komponenten in die Entwicklungsumgebung integriert. Aus der Bildverarbeitung standen zudem eigene Algorithmen für die Vorverarbeitung und Merkmalsextraktion zur Verfügung, die wiederverwendet werden sollten. Dementsprechend wurde großen Wert auf die einfache Integrationsmöglichkeiten vorhandener Software gelegt, obwohl dies zunächst nur als wünschenswerte Eigenschaft angeführt wurde.

Die Objektorientiertheit erlaubt die Implementierung der für die Interaktion zwischen Komponenten und Komponentensoftware notwendigen Kontrollstrukturen in einer abstrakten Basisklasse. Sämtliche Komponenten werden von dieser Basisklasse abgeleitet und erben dadurch die Kontrollstrukturen. Für eine neue Komponente sind lediglich die Datenstrukturen und die Funktionalität zu implementieren. Die Funktionalität kann durch Überschreiben eines einzigen Methodenaufrufes eingebunden werden. Die Integration vorhandener Bibliotheken sowie der Neuentwurf von Komponenten wird durch diesen Ansatz vereinfacht.

5.1.3 Implizite Strukturierung durch objektorientierten Ansatz

Eine Synthesemethode soll aus einer großen Menge von Komponenten selbständig Merkmalsselektoren und Klassifikatoren bestimmen. Ein automatisches Erkennen und Zuordnen erfordert „Markierun-

gen“ der Komponenten. Bei einer konventionellen Umsetzung muß hierzu eine Vielzahl von Meta-Information mitgeführt werden. Hierbei trägt der Entwickler explizit die Verantwortung für die Korrektheit der Meta-Information. Bei dem objektorientierten Ansatz hingegen wird die Meta-Information implizit durch Vererbung gesetzt. Voraussetzung hierfür ist lediglich, daß alle Merkmalsselektoren von einem gemeinsamen Basis-Selektor und alle Klassifikatoren von einem gemeinsamen Basis-Klassifikator abgeleitet sind. In Java sind Methoden integriert, mit welchen die Elternklassen einer Klasse ermittelt werden können. Die Zugehörigkeit einer Komponente beispielsweise zu der Gruppe der Merkmalsselektoren kann durch diese Methoden festgestellt werden. Dieses Verfahren ist weniger fehleranfällig und erheblich transparenter als der Einsatz von Meta-Information.

Die Kompatibilität unterschiedlicher Komponenten kann durch einen objektorientierten Ansatz und der Untersuchung der verwendeten Datentypen ebenfalls ermittelt werden. Dies ist beispielsweise für Synthesemethoden wie aus Abschnitt c erforderlich. Solche Methoden können nur in objektorientierten Umgebungen eingesetzt werden. Bekannte Komponentensoftware beruht meist nicht auf einem objektorientierten Ansatz.

5.1.4 Interaktion durch graphische Benutzeroberfläche

Ein Klassifikationssystem stellt eine Kombination modularer Algorithmen dar. Keine Art von Software ist daher für eine schnelle Entwicklung von Klassifikationssystemen besser geeignet als eine Komponentensoftware. Ein interaktiver Entwurf von Klassifikationssystemen erfordert eine graphische Benutzeroberfläche, mit der Komponenten ausgewählt und schnell zu einer Verarbeitungskette verbunden werden können. Bei einem automatischen Entwurf durch Synthesemethoden ist ein visuelles Feedback der Struktur des synthetisierten Klassifikationssystems äußerst hilfreich. Ein synthetisiertes Klassifikationssystem kann dadurch schnell verifiziert oder angepaßt werden. Eine geeignete visuelle Darstellung erlaubt außerdem den Vergleich von manuell erstellten Klassifikationssystemen mit automatisch synthetisierten.

Die Programmiersprache Java stellt umfangreiche Bibliotheken für die graphische Oberflächenprogrammierung bereit [Horstmann, 2000]. Mit diesen Bibliotheken wurde eine Benutzeroberfläche implementiert, mit der ein schnelles, interaktives Zusammenfügen von Komponenten möglich ist.

5.1.5 Unterstützung von Synthesemethoden durch eine Meta-Ebene

Synthesemethoden müssen aus der Menge aller verfügbaren Komponenten die Merkmalsselektoren und Klassifikatoren identifizieren. Diese sollen ausgewählt, algorithmisch auf der graphischen Oberfläche plaziert und über Datenkanäle miteinander verbunden werden können. Eine Synthesemethode benötigt daher zumindest dieselbe Kontrolle wie die graphische Oberfläche. Dieselben Aktionen, mit welchen von einem Benutzer interaktiv ein Klassifikationssystem synthetisiert wird, müssen von einer Synthesemethode ausgeführt werden können. Die Meta-Ebene stellt im einfachsten Fall eine Menge von Schnittstellen dar, die sowohl von der graphischen Benutzeroberfläche als auch von den Synthesemethoden genutzt wird. Die Integration von Synthesemethoden könnte dann beispielsweise als Modul innerhalb der Entwicklungsumgebung erfolgen. Alternativ hierzu kann beispielsweise ein Script-Interpreter oder eine Makroaufzeichnung eingesetzt werden, die auf der Menge der Schnittstellen aufsetzen. Wesentlich eleganter und flexibler ist allerdings die Implementierung von Synthesemethoden in Form eigenständiger Komponenten, die sich nach dem Prinzip des ***self-guided assembly*** selbst dynamisch verändern können. Abbildung 9 zeigt schematisch die grundlegende Struktur der Komponentensoftware.

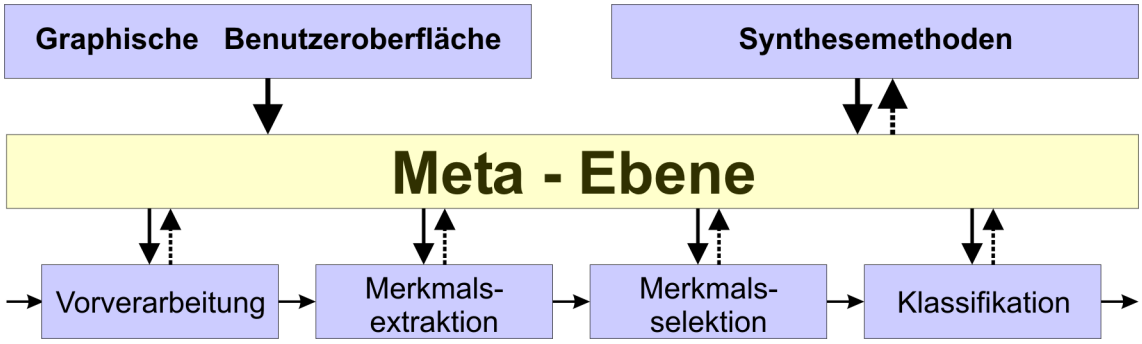


Abbildung 9. Integration einer Meta-Ebene in der entwickelten Komponentensoftware: Mit Hilfe der graphischen Benutzeroberfläche lassen sich Algorithmen interaktiv zu einem Klassifikationssystem zusammensetzen. Die integrierte Meta-Ebene stellt die hierzu notwendige Schnittstelle bereit. Dieselbe Schnittstelle kann von Synthesemethoden zur Automatisierung des Entwurfs verwendet werden.

5.2 Wesentliche Konzepte der Komponentensoftware

Für die Implementierung der Komponentensoftware müssen drei grundlegende Konzepte umgesetzt werden. Diese sind Softwarekomponenten, Datenflußsteuerung und abstrakte globale Datentypen.

5.2.1 Softwarekomponenten

Gekapselte Algorithmen mit definierter Schnittstelle und Funktionalität bezeichnet man als ***Softwarekomponenten*** oder kurz ***Komponenten***. Die Bezeichnung Softwarekomponente wird in der Literatur meist sehr großzügig verwendet. Im Rahmen der entwickelten Komponentensoftware wurden die Softwarekomponenten gemäß [Szyperski, 1999] als voneinander unabhängige Einheiten betrachtet, die derart miteinander agieren, daß sie ein vollständiges System bilden.

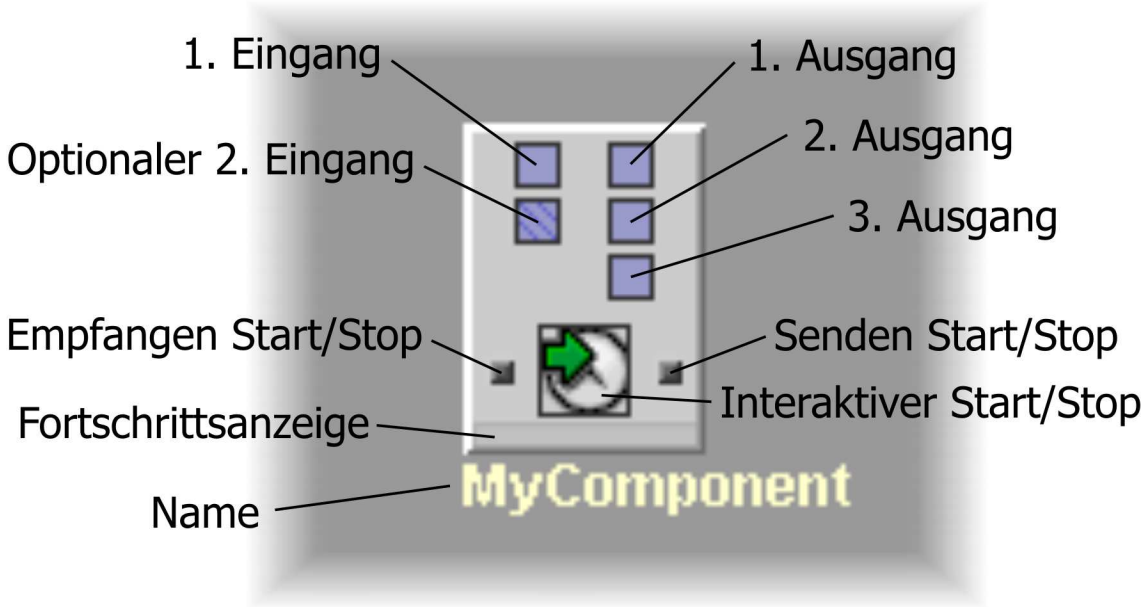


Abbildung 10: Die graphische Benutzerschnittstelle einer Komponente. Oben links und rechts sind die Anschlüsse für Eingangs- und Ausgangsdaten zu sehen. In der Mitte befindet sich der Start/Stop Schalter, mit dem der Benutzer die Komponente ausführen oder anhalten kann. Links und rechts davon befinden sich die Synchronisationsanschlüsse. Über Synchronisationsleitungen kann die Komponente von anderen Komponenten gestartet werden oder selbst andere Komponenten starten. Unter dem Start/Stop Schalter liegt die Fortschrittsanzeige, die den aktuellen Stand einer Verarbeitung widerspiegelt. Zur Identifikation wird unter der Komponente ihr benutzerdefinierter Name eingeblendet.

Durch die Kapselung ist die Umsetzung der Funktionalität einer Komponente transparent. Jede Komponente verfügt über eine Menge von Eingangsdaten, Ausgangsdaten und Parameter. Diese Mengen dürfen auch leer sein. Jede Komponente kann zusätzlich über eine Meta-Beschreibung verfügen. Beliebige Komponenten können genau dann miteinander kombiniert werden, wenn ihre Schnittstellen

es zulassen. Je generischer die Schnittstelle ist, desto mehr Algorithmen lassen sich miteinander verbinden.

Eine Komponente stellt aus Benutzersicht einen verschlossenen Behälter dar, dem Daten zugeführt werden. Entsprechend seiner ausgeschriebenen Funktionalität werden diese Daten verarbeitet und das Ergebnis zur Verfügung gestellt. Eine Anzahl von Komponenten kann wie Lego-Steine zusammengesteckt werden. Aus einfachen Bausteinen können dadurch schnell komplexe Verarbeitungsstrukturen erstellt werden. Zur Gruppierung von Komponenten wurden sogenannte ***Container-Komponenten*** eingeführt, in denen andere Komponenten abgelegt werden können. Werden solche Container ausgeführt, wird die Verarbeitungsstruktur der in ihrem inneren enthaltenen Komponenten ausgeführt.

Komponenten müssen dem Benutzer eine Schnittstelle zur Verfügung stellen, damit dieser mit ihnen arbeiten kann. Abbildung 10 zeigt die graphische Benutzerschnittstelle von Komponenten, wie sie in der Komponentensoftware ICE umgesetzt wurde. Die graphische Darstellung einer Komponente besteht aus einem rechteckigen Behälter. Darunter befinde sich der Name der Komponente. Die Eingangsdaten werden über Datenkanäle an eine Komponente herangeführt. Die Anschlüsse für diese Kanäle befinden sich in der oberen Hälfte in Form von blauen oder grünen Quadraten, gefärbt entsprechend dem Status eines Anschlusses. Links befinden sich die Dateneingänge und rechts die Datenausgänge. Die Anzahl der Ein- und Ausgänge ist für jede Komponente individuell gegeben und abhängig von ihrer Funktionalität. In der Mitte ist die Schaltfläche für einen manuellen Start der Komponente zu sehen. Wird sie vom Benutzer betätigt, startet die Verarbeitung. Bei einer weiteren Betätigung wird der Prozeß abgebrochen. Komponenten können nur dann gestartet werden, wenn diese entweder über keine Eingänge verfügen oder an allen erforderlichen Eingängen gültige Daten anliegen. Eingänge können als optional gekennzeichnet werden und müssen dann für eine Ausführung nicht mit gültigen Daten belegt sein. Links und rechts neben dem Start/Stop Schalter befinden sich punktförmige Synchronisationanschlüsse, durch die andere Komponenten den Start dieser Komponente auslösen können oder durch die diese Komponente andere Komponenten starten kann. Im unteren Abschnitt der Komponente befindet sich ein Fortschrittsdialog, der visuell den aktuellen Verarbeitungsstand der Komponente anzeigt.

Zustände einer Komponente

Abbildung 11: Die möglichen Zustände einer Komponente erklärt von links nach rechts: die Fortschrittsanzeige einer untätigen Komponente ist leer. Werden Daten verarbeitet, füllt sich die Fortschrittsanzeige prozentual zum Fortschritt mit blauer Farbe. Das Icon des Start/Stop Schalters ist verändert. Wurde die Komponente abgebrochen, erscheint die Anzeige gelb. Im Fehlerfall wird sie rot gefärbt. Wurde die Verarbeitung ordnungsgemäß beendet, färbt sich die Fortschrittsanzeige grün. Stehen an den Ausgängen der Komponente Ergebnissen bereit, erscheinen die Anschlüsse grün, ansonsten blau.

Abbildung 11 erläutert von links nach rechts die verschiedenen Zustände einer Komponente. Solange keine Verarbeitung ausgelöst wurde, ist die Komponente untätig. Die Ausgänge enthalten keine Daten bzw. sind noch ungültig. Die Verarbeitung kann erst gestartet werden, wenn an allen erforderlichen Eingangskanälen Daten anliegen. Dies wird durch eine Grünfärbung des Eingangs angezeigt. Der zweite Eingang der Komponente in Abbildung 11 ist optional und somit nicht für die Verarbeitung erforderlich. Er wird schraffiert dargestellt. Wird die Komponente gestartet, wechselt die Darstellung des Start/Stop Schalters. Solange eine Verarbeitung stattfindet, wird anstelle des grünen Pfeils ein rotes Kreuz angezeigt. Im Laufe der Verarbeitung füllt sich der Fortschrittsdialog mit einem blauen Balken von links nach rechts. Initial ist der Balken leer. Wird die Komponente durch nochmaliges Betätigen des Start/Stop Schalters vorzeitig abgebrochen, färbt sich die Fortschrittsanzeige gelb. Tritt während der Verarbeitung ein Fehler auf, wird der Balken rot eingefärbt. Nur wenn die Verarbeitung ordnungsgemäß abgeschlossen wurde, erscheint die Fortschrittsanzeige grün und die Ausgänge werden ebenfalls grün eingefärbt. Eine Verarbeitung kann beliebig oft zurückgesetzt und neu gestartet werden.

In der Entwicklungsumgebung ist eine Basis-Komponente definiert, von der alle verwendeten Komponenten abgeleitet sein müssen (siehe Anhang). Die Basis-Komponente enthält die notwendige Infrastruktur für die einfache Integration neu erstellter Komponenten: eine Plugin Schnittstelle für die graphische Darstellung der Komponente, die Kommunikation mit anderen Komponenten, die Verwaltung

von Parametern und die Umsetzung der gewünschten Funktionalität. Beispielsweise wird die Steuerung von Fortschrittsdialog oder Systemausgaben über Schnittstellen der Basis-Komponente abgewickelt. Aufgrund des objektorientierten Ansatzes erbt jede Komponente automatisch die nötigen Schnittstellen von der Basis-Komponente.

Ablaufphasen einer Komponente

Die Ausführung einer Komponente findet jeweils in einem eigenen Thread statt. Dies erlaubt ein gewisses Maß an paralleler Verarbeitung und erhöht die Flexibilität. Der Thread wird bei jedem Start einer Komponente neu angelegt und durchläuft drei essentielle Phasen:

- Initialisierung: In diesem Abschnitt wird geprüft, ob ein Start der Komponente überhaupt zulässig ist. Die Eingangsdaten werden auf ihre Existenz und Gültigkeit überprüft und die Verarbeitung gegebenenfalls abgebrochen. Bei Gültigkeit werden unter anderem verschiedene Variablen initialisiert sowie die Ausgänge und der Fortschrittsdialog zurückgesetzt.

- Funktionalität: Das eigentliche Herz einer Komponente befindet sich in einer einzigen Methode: componentRun(). In dieser Methode wird die gesamte Funktionalität abgewickelt. Die Eingangsdaten stehen als Variablen zur Verfügung. Diese können ausgelesen und verarbeitet werden, wobei dem Programmierer das Setzen der Fortschrittsanzeige und Reaktionen auf Abbruchanforderung überlassen bleiben. Die erstellten Ausgangsdaten müssen abschließend in die Variablen der Ausgänge übertragen werden. Der boolsche Rückgabewert der Methode ist entweder „True“, wenn die Verarbeitung erfolgreich war, oder „False“ im Falle eines Fehlers.

- Abschluß: Durch diesen Abschnitt wird die Komponente in jedem Fall beendet. Zunächst werden Variablen zurückgesetzt. Beispielsweise wird der Fortschrittsdialog dem Status entsprechend eingefärbt. War die Verarbeitung erfolgreich, werden die errechneten Daten an nachfolgende Komponenten verteilt und auf Synchronisation wartende Komponenten benachrichtigt. Auf das Prinzip der Kommunikation wird im folgenden Abschnitt 5.2.2 eingegangen.

Aufgrund der visuellen Darstellung der jeweiligen Komponente durch vererbte Methoden liegt die Kontrolle zum großen Teil bei der Basis-Komponente. Die graphische Darstellung einer Komponente wird von „Plugins“ übernommen. Das Erscheinungsbild einer Komponente kann dadurch auf einfache Weise geändert werden. Die erzwungene strikte Trennung von Funktionalität und Darstellung bietet

viele Vorteile. Das Plugin zur Visualisierung kann beispielsweise auch leer sein. Abläufe aus Komponenten können dadurch als eigenständige Applikationen ohne Verbindung mit einer graphischen Oberfläche ausgeführt werden.

5.2.2 Datenflusssteuerung und Synchronisation

Abbildung 12 zeigt die möglichen Zustände einer typischen Datenverbindung zwischen zwei Komponenten beginnend von Daten ungültig bis zu externer Synchronisation.

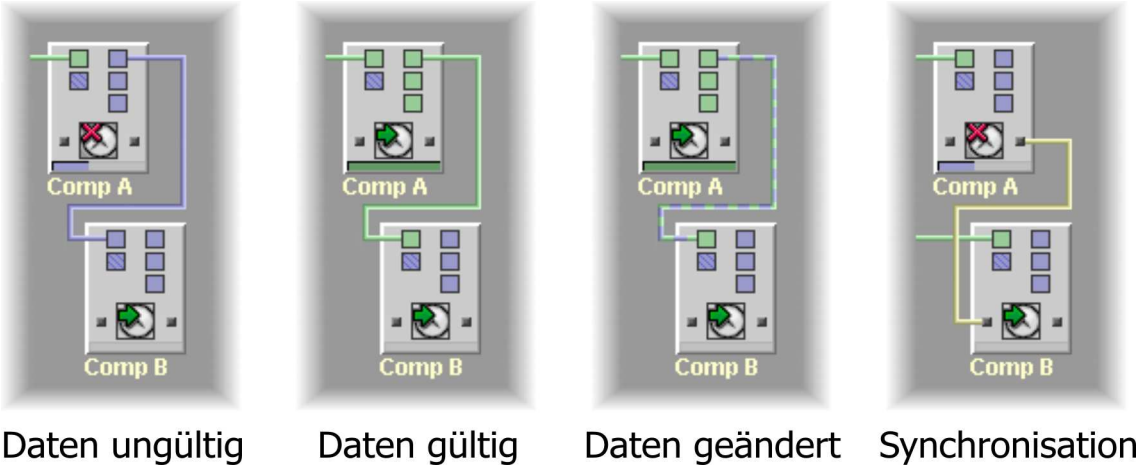


Abbildung 12. Kanäle zwischen Komponenten erklärt von links nach rechts: Ausgang eins der Komponente CompA ist mit Eingang eins der Komponente CompB verbunden. CompA stellt zunächst keine Daten zur Verfügung. Der Datenkanal ist leer und die Verbindung erscheint blau. Nachdem CompA die Berechnung abgeschlossen hat und Daten zur Verfügung stellt, erscheint die Verbindung grün. Ändern sich nach der Berechnung die Parameter von CompA, sind die berechneten Daten veraltet und bleiben aber vorerst im Kanal erhalten. Dies wird durch eine gestrichelte Verbindung symbolisiert. Die letze Darstellung zeigt einen Synchronisationskanal. Beendet CompA ordnungsgemäß die Verarbeitung, wird CompB auch ohne Verbindung über einen Datenkanal zu CompA über den Synchronisationskanal gestartet.

Die graphische Darstellung aller Verbindungen wird von einem Verbindungs-Manager übernommen. Mit Hilfe des Verbindungs-Managers können Komponenten einerseits interaktiv durch den Benutzer verbunden werden. Andererseits stehen dieselben Schnittstellen zum Verbindungsaufbau den Softwarekomponenten zur Verfügung. Komponenten selbst können daher auch automatisch Verbindungen aufbauen. Während des Verbindungsaufbaus fügt eine Daten erzeugende Komponente (Producer) den Daten verbrauchenden Partner (Consumer) ihrer internen Liste hinzu. Der Consumer geht daraufhin in einen Ruhezustand über und wartet auf gültige Daten an seinen Eingängen. Schließt ein Producer erfolgreich seine Verarbeitung ab, benachrichtigt er jeden Consumer seiner Liste. Konkret ruft er dazu eine Methode des Consumers auf, welche die Referenzen der Ausgangsdaten in die Eingänge des Consumers kopiert. Unter bestimmten Bedingungen wird die Verarbeitung durch den nachfolgenden Consumer automatisch angestoßen. Ist dies nicht der Fall, muß der Consumer interaktiv durch einen

Benutzer gestartet werden. In den Datenkanälen existiert kein kontinuierlicher Fluß von Daten, sondern es werden nur kurzzeitig Nachrichten übertragen. Bei Bedarf können Verbindungen auch wieder abgebaut werden. Dazu muß der Consumer lediglich aus der Liste des Producers entfernt werden. Das beschriebene Konzept der Nachrichtenübertragung wird als Event-Modell bezeichnet.

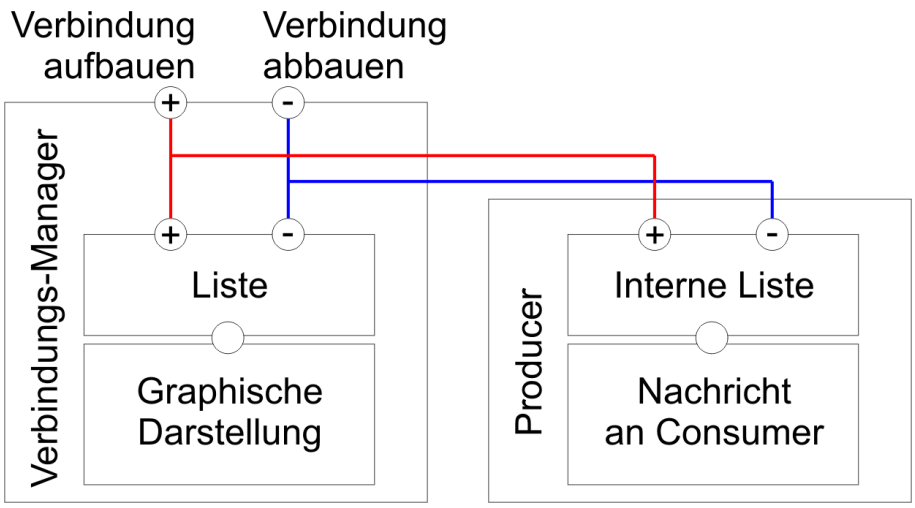


Abbildung 13. Der Verbindungsmanager übernimmt Darstellung und Management von Komponenten. Die Komponenten tauschen Daten ohne Hilfe des Verbindungs-Managers aus.

Komponenten können auf drei verschiedene Arten gestartet werden: erstens explizit durch einen Mausklick des Benutzers. Zweitens durch einen boolschen Parameter eines Datenflusses, der bei Benachrichtigung ausgewertet wird. Die dritte Möglichkeit besteht in der Übertragung einer Nachricht zum Start einer Komponente. Oft ist es notwendig, mit einer Komponente eine andere zu starten, auch wenn beide nicht durch einen Datenkanal miteinander verbunden sind. Da Komponenten auf diese Weise synchronisiert werden können, bezeichnet man eine solche Verbindung als Synchronisationskanal. Über einen Synchronisationskanal können außer des Startsignals keine weiteren Daten übertragen werden. In dem Bild ganz rechts in Abbildung 12 ist die obere Komponente über einen gelb dargestellten Synchronisationskanal mit der unteren Komponente verbunden. Die beiden Komponenten sind nicht durch einen Datenkanal miteinander verbunden. Muß die untere Komponente nach erfolgreicher Beendigung der oberen Komponente gestartet werden, ist dies nicht über den Datenkanal möglich. Zu diesem Zweck existieren Synchronisationskanäle.

Das Prinzip der Synchronisation entspricht dem des Datenflusses. Während des Verbindungsaufbaus trägt der Producer den Consumer in seine interne Liste ein. Der Producer startet nach erfolgreicher Verarbeitung den Thread des Consumers. Dies ist identisch mit einer Betätigung des Start/Stop Schal-

ters. An einen Synchronisationsausgang können beliebig viele Consumer angeschlossen werden. Synchronisationseingänge dagegen werden von höchstens einem Producer versorgt. Synchronisationsverbindungen können bei Bedarf ebenso wieder abgebaut werden. Wie die Verwaltung der Datenverbindungen wird die der Synchronisationsverbindungen von dem Verbindungs-Manager übernommen.

5.2.3 Abstrakte globale Datentypen

Die Idee einer Komponentensoftware beruht auf transparenten Komponenten verschiedenen Ursprungs mit unterschiedlichen Funktionen, die sich universell verbinden lassen. Idealerweise ist bis auf die Bezeichnung der Funktionalität keinerlei Wissen über eine Komponente notwendig. Werden in der Praxis allerdings zwei Algorithmen von verschiedenen Entwicklern miteinander verbunden, müssen die ausgetauschten Datenstrukturen genau beschrieben sein. Dies bedeutet sehr wohl konkretes Wissen um die Bedeutung der Datenstrukturen und steht somit in direktem Widerspruch zu der angestrebten Transparenz, d.h. ein kritischer Faktor bei dem Entwurf von Komponentensoftware liegt in einer sauberen Definition der Schnittstelle zwischen Komponenten. Einige existierende Komponentensysteme sind in der Lage, die Schnittstellenstruktur ihrer Bausteine zu untersuchen. Durch diese Untersuchung wird eine Pseudo-Transparenz gewonnen. Die Syntax und somit Kompatibilität einer Schnittstelle läßt sich überprüfen, jedoch nicht ihre Semantik interpretieren. Das folgende Beispiel verdeutlicht dies: Zwei verschiedene Entwickler bieten eine Komponente für die Bildfilterung an. Beide Komponenten verfügen über einen Eingang. Die erwartete Datenstruktur beider Schnittstellen ist ein einfaches Bytearray. Somit können beide Komponenten an den Ausgang einer erzeugenden Komponente angeschlossen werden, wenn diese als Ausgangsstruktur ein Bytearray zur Verfügung stellt. Die erste Komponente interpretiert das Bytearray als acht Bit Graustufenbild. Die zweite Komponente interpretiert das Bytearray jedoch als 16 Bit Graustufenbild. Ohne zusätzliche Information ist die Schnittstelle „wertfrei“. Ihre Syntax ist zwar erfüllt, nicht jedoch ihre semantische Bedeutung festgelegt. Die Trennung von Semantik und Syntax für Komponentenschnittstellen ist von zentraler Bedeutung [Berg, 1997].

Herkömmliche Komponentensysteme wie Khoros definieren lediglich eine Syntax. Die Semantik kann durch den folgenden, idealistischen Ansatz in Schnittstellen eingebracht werden: Für bestimmte Anwendungen sind grundsätzlich ähnliche Datenstrukturen notwendig. Aus jedem Arbeitsgebiet können Vorschläge gesammelt und zu ***abstrakten globalen Datentypen*** zusammengefaßt werden, ähnlich den aus der Informatik bekannten ***abstrakten Datentypen*** [Manber, 1989]. Im obigen Beispiel der Bildfilter wäre eine mögliche Lösung die Definition eines Datentyps „Graustufenbild“, der global

definiert und allen Entwicklern auf der Welt bekannt ist. Die Semantik wird aus Schnittstellen herausgenommen, indem gemeinsame Datentypen im voraus als Klassen einer objektorientierten Programmiersprache definiert werden. Die gesammelten Datentypen werden in einer Klassenhierarchie zusammengefaßt, die in einer Basisklasse wurzelt. Die Basisklasse implementiert alle notwendigen Methoden für die Integration in die Komponentensoftware. Die Klassenhierarchie wird global zur Verfügung gestellt. Die enthaltenen Datentypen sind die einzig zugelassenen Strukturen für den Austausch in Komponentenschnittstellen. Neuentwicklungen können somit nicht nur auf Komponenten zurückgreifen, sondern auch auf bereits implementierte Datentypen.

Globale Datentypen ermöglichen eine grobe Kompatibilitätsprüfung von Ablaufstrukturen. Der Datentyp einer produzierenden Komponente muß dem der konsumierenden Komponente entsprechen oder von diesem zumindest abgeleitet sein, um eine Verbindung zu erlauben. Die Kompatibilitätsprüfung ist eine notwendige Voraussetzung für die Synthesemethode durch Klassenhierarchie. Abstrakte globale Datentypen sind optimal geeignet für Komponentenschnittstellen. Ihre Schwäche liegt in der naiven Annahme, global einheitliche Datentypen für alle nur denkbaren Eventualitäten zu schaffen. Allerdings steht mit der entwickelten Klassenhierarchie (siehe Anhang) eine Basis für zukünftige Entwicklungen bereit. Über viele Iterationen können im Laufe der Zeit Datentypen aus unterschiedlichen Einsatzgebieten zusammenwachsen und schließlich die angestrebte globale Klassenhierarchie bilden.

Eine Visualisierung von Datentypen beispielsweise erfordert in herkömmlicher Komponentensoftware wie Khoros, daß eine visualisierende Komponente über Struktur und Semantik eines jeden Datentyps informiert ist. Dies steht im Widerspruch zur angestrebten Transparenz einer Komponente. Nach objektorientiertem Ansatz bleibt die Visualisierung eines Datentyps ausschließlich diesem selbst überlassen, da nur er über seine Struktur und Semantik orientiert ist. Für eine hohe Transparenz ist es notwendig, möglichst viel Funktionalität von Komponenten in Datentypen zu verlagern. Dies geschieht durch Definition abstrakter[1] Schnittstellen in der Datentypen-Basisklasse, von der sämtliche Datentypen abgeleitet sind. Alle abgeleiteten Klassen verfügen dadurch über diese Schnittstellen. Der Implementierungsaufwand ist grundsätzlich unabhängig davon, ob eine Funktionalität in einer Komponente oder einem Datentypen integriert wird. Der Austausch von Semantik wird durch Definition abstrakter Schnittstellen jedoch überflüssig. Bestimmte Komponenten sind auf diese Weise auf jeden Datentyp anwendbar. Beispielsweise definiert die Basisklasse Schnittstellen zur textuellen bzw. graphischen

1 *Definiert, aber nicht implementiert, daß heißt ohne Funktionalität.*

Darstellung eines Datentyps. Für eine graphische Visualisierung existiert die abstrakte Methode „visualize2D“ in der Basisklasse, die von jedem abgeleiteten Datentypen explizit überschrieben werden kann. Ein Datentyp implementiert dadurch seine individuelle graphische Darstellung. Die generell anwendbare Komponente „Visualize“ ruft lediglich die Methode visualize2D des übergebenen Datentypen auf.

5.3 Implementierung der Konzepte

Im folgenden wird die graphische Benutzeroberfläche vorgestellt, welche der interaktiven Steuerung sowohl des manuellen als auch des durch Synthesemethoden unterstützten Entwurfs von Klassifikationssystemen dient. Im Anschluß wird die Integration der Klassifikations- und Synthesealgorithmen beschrieben.

5.3.1 Graphische Benutzeroberfläche

Abbildung 14 zeigt die graphische Benutzeroberfläche von ICE. Auf der rechten Seite ist der aktive „Desktop“ zu sehen. Auf dem Desktop werden Komponenten plaziert und miteinander verbunden. Desktops sind immer mit einer Container-Komponente assoziiert, die wiederum andere Komponenten enthalten kann. Wird eine Container-Komponente „geöffnet“, wird die aktuelle Ansicht mit dem zugeordneten Desktop vertauscht. Komponenten können ausgewählt, verschoben, gelöscht oder in Container-Komponenten abgelegt und so gruppiert werden. Parameter werden geändert, indem eine Komponente ausgewählt wird. Nach der Auswahl erscheinen links oben die Properties der Komponente. Als Properties werden extern sichtbare Variablen von JavaBeans bezeichnet. Diese Properties entsprechen den Laufzeitparametern einer Komponente, die im Prinzip eine erweiterte JavaBean ist. Abgeleitete Komponenten erben die Properties ihrer Vorgänger und können eigene hinzufügen. Unter der Anzeige der Properties ist die hierarchische Struktur des gesamten Ablaufes zu sehen. Geschachtelte Komponenten erlauben baumartige Strukturen. Diese sind ähnlich organisiert wie ein Dateisystem. Container-Komponenten entsprechen in dieser Analogie Verzeichnissen und die übrigen Komponenten den Dateien. Über den Desktop kann immer nur ein Knoten angezeigt werden. Der Strukturbaum erlaubt einen schnellen Überblick und raschen Zugriff auf alle Komponenten. In der obersten Zeile der Oberfläche befindet sich eine Menuleiste, welche Zugriffe auf Dateiverwaltung, Optionen, Hilfsmittel und Komponenten ermöglicht. Komponenten werden einem Desktop durch Auswahl aus dem Komponentenmenu hinzugefügt. Unter der Menuleiste befindet sich eine Schnellstartleiste mit den gebräuchlichsten Befehlen wie beispielsweise „Kopieren“, Ausschneiden“ oder „Einfügen“.

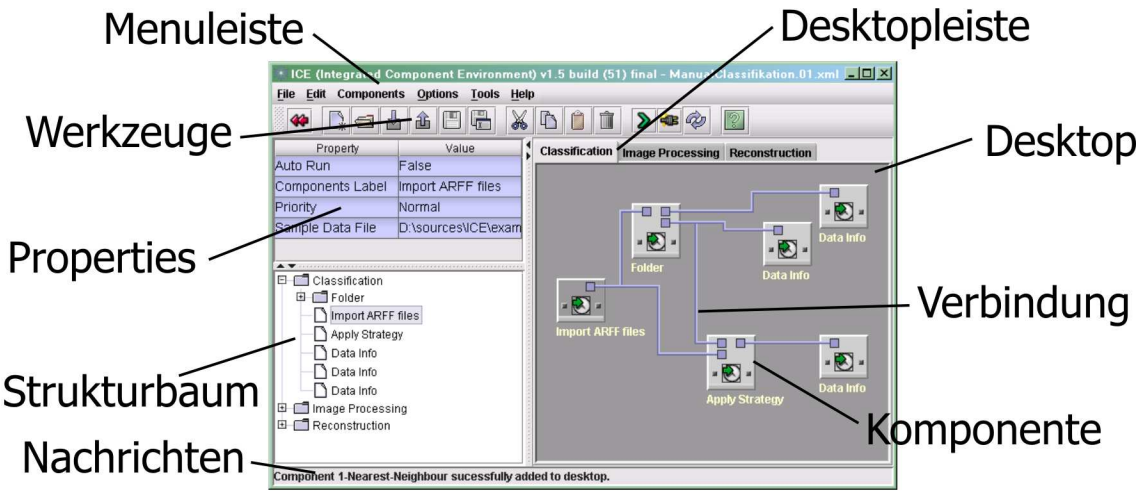


Abbildung 14. Das Integrated Component Environment (ICE) ist eine Komponentensoftware und wurde in erster Linie als Entwicklungsumgebung für Klassifikationssystemen erstellt. In dem Fenster rechts darunter befindet sich der Desktop, auf dem Komponenten und ihre Verarbeitungsreihenfolge visualisiert werden. Links oben ist das Property-Fenster mit den Parametern der aktuell ausgewählten Komponente zu sehen. Darunter befindet sich der Strukturbaum, der hierarchische Schachtelungen der Komponenten ähnlich einem Dateisystem visualisiert. Die Menu- und Werkzeugleiste befinden sich ganz oben.

Die hierarchische Struktur eines erstellten Ablaufes kann persistent abgelegt werden. Die Komponenten und ihre Verbindungen durch Datenkanäle werden als strukturierte XML-Dateien mit zugehöriger DTD (Document Type Definition) gespeichert [Großwendt, 2000][Myers, 2000]. Von Vorteil ist neben der Plattformunabhängigkeit aufgrund der textuellen Persistenz die Möglichkeit zur Darstellung und Weiterverarbeitung von XML-Dateien durch eine Vielzahl externer Softwarehilfsmittel.

5.3.2 Klassifikatoren und Merkmalsselektoren

Die Klassifikatoren und Merkmalsselektoren von ICE werden hauptsächlich vom Weka-System gestellt. Die Weka eigenen Klassen werden von Komponenten gekapselt und sind somit über die Schnittstellen der Komponentensoftware zu erreichen. Im folgenden werden die fünf zur Synthese eines Klassifikationssystems notwendigen Komponenten grob skizziert. Ein Desktop für eine beispielhafte, manuelle Klassifikation ist im Anhang abgebildet.

Schnittstelle zu Klassifikationsproblemen

Alle untersuchten Klassifikationsprobleme sind als Textdateien in Tabellenform im ARFF-Format gespeichert, das durch die Weka-Bibliothek vorgegeben ist. Vor einer Verarbeitung werden die Daten durch eine Import-Komponente eingelesen und im Datentyp Features abgelegt, welcher die Grundlage

für eine Klassifikation darstellt. Die einzige Property der Import-Komponente ist der Dateiname des zu importierenden Klassifikationsproblems. Die eingelesenen Daten werden über einen Ausgang vom Typ Features nachfolgenden Komponenten zur Verfügung gestellt.

Merkmalsselektoren

Das Grundprinzip einer Merkmalsselektion ist immer ähnlich. Alle Merkmalsselektoren sind aus diesem Grund von der Komponente *AbstractSelector* abgeleitet (siehe Anhang). In dieser Klasse wird die Methode componentRun() bereits implementiert. Lediglich eine Aufzählungsmethode und eine Bewertungsfunktion muß in jedem abgeleiteten Merkmalsselektor definiert werden. Insgesamt wurden 23 verschiedene Merkmalsselektoren in der Komponentensoftware implementiert. Jeder Merkmalsselektor verfügt über einen Eingang vom Typ *Features*. Der erste Ausgang ist ebenfalls vom Typ *Features*, wobei die Daten auf die selektierte Untermenge der Merkmale eingeschränkt sind. Der zweite Ausgang ist vom Typ SelectorData und enthält die Information, welche Merkmale der ursprünglichen Merkmalsmenge selektiert wurden. Selektoren besitzen keine individuellen Properties.

Klassifikatoren

Wie bei den Merkmalsselektoren ist das Schema für eine Klassifikation im Prinzip immer gleich. Aus diesem Grund werden alle Klassifikatoren von der Komponente *AbstractClassifier* abgeleitet (siehe Anhang). In diesem Basis-Klassifikator werden alle Methoden und Definitionen bereitgestellt, die für eine Klassifikation notwendig sind. Alle Klassifikatoren verfügen dadurch automatisch über das gleiche Erscheinungsbild. Die Methode componentRun() ist für alle Klassifikatoren identisch und im Basis-Klassifikator implementiert. Für einen abgeleiteten Klassifikator muß lediglich eine Datenstruktur mit dem gewünschten Klassifikatortyp definiert werden. Klassifikatoren verfügen über einen Eingang vom Typ Features. Der erste Ausgang ist vom Typ *ClassificationError*. Er enthält Informationen über Test- und Trainingsfehler sowie eine Fehlermatrix. Der zweite Ausgang ist vom Typ *AbstractClassifierData*. In diesem Typ ist der trainierte Kern des Klassifikators enthalten. Der Kern enthält alle Informationen, die der Klassifikator durch das Training ermittelt hat. Mit diesem Kern ist die Klassifikation weiterer Daten möglich. Klassifikatoren besitzen eine einzige individuelle Property, welche im Basis-Klassifikator definiert wird: die Methode der Fehlerschätzung. Zur Abschätzung des Generalisierungsfehlers wurden vier Varianten einer Kreuzvalidierung und zwei Varianten eines Bootstrap-Verfahrens implementiert. Die Verfahren unterscheiden sich beispielsweise durch höhere Genauigkeit der Fehlerschätzung für im Gegenzug längere Laufzeit. Insgesamt wurden 14 verschiedene Klassifikatoren in die Komponentensoftware integriert. Dadurch ergeben sich für eine gestellte Klas-

sifikationsaufgabe 322 Kombinationsmöglichkeiten, deren Güte durch Schätzung des Generalisierungsfehlers beurteilt werden muß.

Zusammenfassen eines Klassifikationssystems

Ein synthetisiertes Klassifikationssystem soll zur Klassifikation von Folgedaten eingesetzt werden. Hierzu ist die Information über die selektierte Untermenge der Merkmale und der trainierte Kern des Klassifikators notwendig. Diese Daten sind sehr stark miteinander verknüpft, resultieren aber aus unterschiedlichen Quellen. Aus diesem Grund werden sie zu einem gemeinsamen Datentyp *StrategyData* zusammengefaßt. Dies geschieht in der Komponente *SynthesizeStrategy*. Diese besitzt keine individuellen Properties. Der erste Eingang ist vom Typ *AbstractClassifierData* und empfängt den von einer Klassifikator-Komponente trainierten Kern. Der zweite Eingang ist vom Typ *SelectorData* und empfängt die von einem Merkmalsselektor generierte Information über die Untermenge der Merkmale. Der Ausgang von *SynthesizeStrategy* ist vom Typ *StrategyData*.

Anwenden eines Klassifikationssystems

Um ein synthetisiertes Klassifikationssystem schließlich auf neue, unbekannte Daten anzuwenden, dient die Komponente *ApplyStrategy*. Diese besitzt keine eigenen Properties. Der erste Eingang ist vom Typ *StrategyData* und ist das Resultat der Komponente *SynthesizeStrategy*. Der zweite Eingang ist vom Typ *Features* und enthält zu klassifizierende Daten gleicher Struktur, auf welcher der Merkmalsselektor eine Untermenge der Merkmale selektiert. *ApplyStrategy* wendet die durch *StrategyData* definierte Lösungsstrategie auf die Daten an. *Features* wird kopiert und die Klassen der einzelnen Beobachtungen entsprechend der durchgeführten Klassifikation geändert. Das Ergebnis wird am Ausgang als Daten vom Typ *Features* zur Verfügung gestellt.

5.3.3 Synthesekomponenten und Synthesekoordination

Neben den fünf vorgestellten Komponentenarten, mit denen auf manuellem Weg Klassifikationssysteme erstellt werden können, sind für eine automatische Synthese zwei weitere Komponentenarten erforderlich. Nachfolgend werden diese beiden Komponentenarten vorgestellt.

Synthesekomponenten

Dies sind als Komponenten implementierte Verfahren, die aus Komponenten für die Klassifikation automatisch Klassifikationssysteme synthetisieren. Alle Synthesekomponenten arbeiten nach dem gleichen Schema. Zudem besitzen sie ähnliche Laufzeitparameter. Die Ähnlichkeit motiviert bei einer objektorientierten Implementierung die gemeinsame Superklasse *AbstractStrategy*, von der alle Syn-

thesekomponenten abgeleitet sind. Eine Synthesekomponente muß Komponenten zur Klassifikation auswählen und in die graphische Oberfläche einfügen. Dies wurde gelöst, indem die gemeinsame Superklasse *AbstractStrategy* von der Container-Komponente abgeleitet wurde. Eine Synthesekomponente ist nicht nur Komponente, sondern zugleich ein Container, der andere Komponenten beinhaltet. Die Komponente *AbstractStrategy* verfügt zudem über die Möglichkeit, die in ihr enthaltenen Komponenten algorithmisch zu manipulieren. Eine Synthesekomponente besitzt einen Eingang vom Typ *Features*. Der erste Ausgang ist vom Typ *ClassificationError* und enthält die Fehlerinformation des besten synthetisierten Systems. Der zweite Ausgang ist vom Typ *StrategyData*. Er enthält das synthetisierte Klassifikationssystem. Wird die Synthesekomponente wie eine Container-Komponente geöffnet, gibt sie graphisch die aktuelle Lösung wieder. Während der Synthese läßt sich „online“ verfolgen, welche Merkmalsselektoren und Klassifikatoren gerade ausgewählt, verbunden und getestet werden. Das Verhalten der Synthesekomponente läßt sich durch ihre Properties beeinflussen. Zum einen kann die Synthesezeit beschränkt werden. Ist die eingestellte Frist abgelaufen, wird die Synthese abgebrochen. Über eine weitere Property wird eine Fehlerschwelle angegeben. Die Synthese endet, sobald der Generalisierungsfehler eines generierten Klassifikationssystems diese Schwelle unterschreitet. Die dritte Property stellt die Methode zur Fehlerschätzung ein. Diese wird direkt an die Property des verwendeten Klassifikators weitergegeben. Die letzte Property erlaubt, zwischen Synthese und Klassifikation umzuschalten. Da eine Synthesekomponente ein Container ist, kann der enthaltene Ablauf wie in einer normalen Container-Komponente ausgeführt werden. Steht der Schalter auf „Synthese“, wird zunächst ein Klassifikationssystem erstellt. Nach Umschalten auf „Klassifikation“ können mit dem synthetisierten Klassifikationssystem Eingangsdaten klassifiziert werden. Die Synthesemethoden mit Zufallsstrategie verfügen über eine fünfte und sechste Property, mit denen sich die beschriebene Wahrscheinlichkeitsschwelle aus Abschnitt 4.1.5 kontrollieren läßt.

Von den in Kapitel 4 vorgeschlagenen Synthesemethoden wurden die folgenden implementiert und eingesetzt: sieben Zufallsmethoden, vollständige Liste nach Komplexität, vollständige Liste nach Erfolgsaussicht und Blasensynthese. Die Blasensynthese wurde in vier Varianten implementiert, die miteinander verglichen wurden: ohne Merkmalsselektor, vorgegebener Merkmalsselektor, Merkmalsselektor nach bestem Klassifikator und vollständige Kombination von Klassifikatoren mit Merkmalsselektoren. Insgesamt wurden 13 Synthesekomponenten in der Komponentensoftware integriert. Für die Beschleunigung der Zufallsmethoden wurden eine Zeitschwelle, eine Wahrscheinlichkeitsschwelle und eine Fehlerschwelle implementiert.

Koordination von Synthesekomponenten

Für einen Vergleich der verschiedenen Synthesemethoden ist eine große Menge an Klassifikationssystemen zu synthetisieren und zu bewerten. Die Synthese kann sich über Tage oder gar Wochen hinziehen. Um die Ergebnisse korrekt zu protokollieren, wurde eine eigene Komponente entworfen: der *StrategyCoordinator*. Der *StrategyCoordinator* ist von der Container-Komponente abgeleitet. Seine Aufgabe besteht darin, eine große Anzahl von Klassifikationssystemen iterativ von einer Synthesekomponente synthetisieren zu lassen. Über eine Property kann eine Synthesekomponente zur Untersuchung ausgewählt werden. Für jede Iteration werden Merkmalsselektor, Klassifikator, Synthesezeit etc. des synthetisierten Klassifikationssystems in einer Datei protokolliert. Die Komponente *StrategyCoordinator* verfügt über die gleichen Properties wie eine Synthesekomponente. Diese werden an die zu untersuchende Synthesekomponente durchgereicht. Zusätzlich besitzt der *StrategyCoordinator* eine Property für die Angabe der Protokolldatei, in der die Ergebnisse protokolliert werden sollen, und eine Property für die Anzahl der gewünschten Iterationen. Die Komponente *StrategyCoordinator* verfügt lediglich über einen Dateneingang vom Typ *Features*. Da die Ergebnisse als Protokolldatei gespeichert werden, besitzt die Komponente keine Datenausgänge

5.3.4 Werkzeuge der Entwicklungsumgebung

Bei dem Einsatz der Komponentensoftware ICE steht der schnelle Entwurf von Klassifikationssystemen im Vordergrund. Der konventionelle Programmieraufwand für eine Anwendung soll sich auf ein Minimum beschränken. In die Komponentensoftware ICE wurde daher eine Reihe von Werkzeugen integriert, die den Entwurf eigener Komponenten und Datentypen erheblich vereinfachen. Alle Werkzeuge sind vollständig in Java implementiert und als Module in die graphische Oberfläche der Komponentensoftware integriert.

Entwurf von Komponenten

Für die Entwicklung eigener Komponenten in der Komponentensoftware könnten bestehende Komponenten einfach kopiert und modifiziert werden. Allerdings ist dazu eine genaue Kenntnis über die interne Struktur von Komponenten erforderlich. Dies soll für eine Entwicklung eigener Komponenten aber nicht vorausgesetzt werden. Mit dem Werkzeug *CreateComponent* lassen sich schnell und einfach neue Komponenten erstellen. Beispielsweise wird die Klassenhierarchie bereits implementierter Komponenten erstellt und visualisiert. Die Visualisierung erleichtert das gezielte Ableiten neuer Komponenten von bereits bestehenden erheblich. Durch eine Eingabemaske werden die Rahmenbedingungen für eine neue Komponente festgelegt. Aus den Rahmenbedingungen wird automatisch der Quell-

text erstellt. Im Quelltext muß lediglich die Methode componentRun() mittels konventioneller Programmierung modifiziert werden. Die festgelegten Parameter der Komponente können auch nachträglich im generierten Quelltext verändert werden. Zum Einbinden neuer Komponenten sind keine besonderen Aktionen notwendig. Neue Komponenten werden bei einem Start automatisch eingebunden. Der Aufwand für die Entwicklung neuer Komponenten wird damit auf ein Minimum reduziert.

Entwurf von Datentypen

Das Werkzeug *CreateDatatype* dient für den Entwurf eigener Datentypen. Analog zum Entwurf von Komponenten werden über eine Eingabemaske Parameter eingestellt, die für den Datentypen notwendig sind. Ein neuer Datentyp kann von einem verfügbaren abgeleitet werden, indem dieser aus einer automatisch erstellten Liste ausgewählt wird. Verfügbare Datentypen können automatisch erkannt werden, da sie durch den objektorientierten Ansatz eine gemeinsame Basisklasse besitzen. Die Definition eigener Datentypen ist nicht unbedingt konform mit der Idee abstrakter globaler Datentypen. Für den praktischen Gebrauch und die prototypische Implementierung eines Datentyps ist dieses Werkzeug allerdings unumgänglich.

Erstellen eigenständiger Applikationen aus Verarbeitungsketten

Nachdem eine Verarbeitungskette mit Hilfe der Komponentensoftware ICE erstellt wurde, ist die Entwicklungsphase abgeschlossen. Der Overhead durch die Entwicklungsumgebung ist für eine eigenständige Anwendung enorm. Idealerweise soll die Anwendung von unnötigem Ballast getrennt werden. Dazu wird mit dem Werkzeug *Supercompiler* aus dem Ablauf Java-Quelltext erstellt. Der Quelltext enthält die algorithmische Umsetzung der visuellen Ablaufstruktur, jedoch ohne die graphischen Anteile. Zum Aufbau von Verbindungen werden dabei die gleichen Methoden und Schnittstellen verwendet, wie sie auch von Synthesemethoden zur Synthese von Klassifikationssystemen eingesetzt werden. Der resultierende Quelltext spiegelt den Ablauf als selbständiges Java Programm wieder und kann auch ohne die graphische Benutzeroberfläche angewendet werden.

5.4 Anwendungsgebiete außerhalb der Klassifikation

Aufgrund ihres universellen Charakters kann die entwickelte Komponentensoftware auch vorzüglich in anderen Anwendungsgebieten als der Klassifikation eingesetzt werden. Die Komponentensoftware wurde als Entwicklungsplattform für die Medizintechnik verwendet, beispielsweise für die Brustkrebsdiagnose zur Detektion von Mikroverkalkungen [Müller, 2000] [Müller, 2001a] und der Klassifi-

kation von Bildausschnitten durch einen Case-Based-Reasoner [Korf, 2000]. Ebenso wurden mit Hilfe der Komponentensoftware neuartige Methoden für die Registrierung von Kernspin-Mammogrammen mit Röntgenmammogrammen entwickelt [Ruiter, 2001]. ICE wurde auch für die Segmentierung und Detektion von Lymphozyten [Beller, 2003] [Beller, 2004] und Lunkern (Fehlern in Spritzguß-Werkstücken) erfolgreich eingesetzt. In der Mikrosystemtechnik wurde mit ICE als Entwicklungsplattform die Ansteuerung elektronischer Nasen realisiert [Goschnick, 2003]. Für die Auswertung von Funkdaten eines Wetterexperiments wurde ICE als Entwicklungsplattform für die Konzipierung und Umsetzung eines TCP/IP-Servers [da Silva, 2001] verwendet. Die Komponentensoftware konnte zudem für zahlreiche weitere Aufgaben für die prototypische Entwicklung genutzt werden, da die effiziente graphische Benutzeroberfläche eine schnelle Montage von Prozeßketten beispielsweise für die Bildverarbeitung ermöglicht und bereits eine große Anzahl nützlicher Komponenten implementiert wurde.

5.5 Zusammenfassung

Für die Implementierung des ***Integrated Component Environment (ICE)*** wurden drei grundlegende Konzepte umgesetzt: ***Softwarekomponenten***, ***Datenflußsteuerung*** und ***abstrakte globale Datentypen***. Alle Softwarekomponenten werden von einer Basiskomponente abgeleitet, durch die sie automatisch Methoden für die Kommunikation und Steuerung durch die Komponentensoftware ICE erben. Zur Integration neuer Softwarekomponenten muß lediglich die Funktionalität implementiert werden. Die Koordination von Softwarekomponenten erfolgt durch Datenflußsteuerung. Zwischen Softwarekomponenten können Datenkanäle errichtet werden, so daß von einer Erzeugerkomponente Daten zu einer Verbraucherkomponente übertragen werden können.

Für die Klassifikation wurden 23 Merkmalsselektoren und 14 Klassifikatoren als Softwarekomponenten implementiert und untersucht. Mit Hilfe der graphischen Benutzeroberfläche können mit diesen Softwarekomponenten interaktiv Klassifikationssysteme erstellt werden. Für die Automatisierung des Entwurfs wurden 13 Synthesemethoden als Softwarekomponenten implementiert, welche die Meta-Ebene des ***self-guided assembly*** nutzen. Aufgrund ihrer Flexibilität wurde die Komponentensoftware erfolgreich in vielen verschiedenen Projekten nicht nur für die Klassifikation eingesetzt: der Steuerung elektronischer Nasen in der Mikrosystemtechnik, der Segmentierung und Detektion von Fehlern in Spritzguß-Werkstücken, der Entwicklung und Realisierung eines TCP/IP-Servers für ein Wetterdatenexperiment, der Brustkrebsdiagnose und Lymphozytendetektion in der Medizintechnik sowie nicht zuletzt für eine große Anzahl von Bildverarbeitungsaufgaben.

Besonders hervorzuheben an der neuen Komponentensoftware sind die vollständig objektorientierte Implementierung in Java, die äußerst einfache Integration neuer Algorithmen und die effiziente Benutzerschnittstelle für die schnelle visuelle Montage von Prozeßketten.

6 Ergebnisse automatisch erstellter Klassifikationssysteme

Mit Hilfe der Komponentensoftware für die Klassifikation konnten die entwickelten Synthesemethoden implementiert und getestet werden. Der erste Teil dieses Kapitels beschäftigt sich mit der Leistungsfähigkeit der Blasensynthese und ihrer Verifikation. Der Algorithmus wird mit künstlichen Klassifikationsproblemen bekannter Wahrscheinlichkeitsdichte und bekanntem Generalisierungsfehler untersucht. Im zweiten Teil werden die Syntheseergebnisse realer Klassifikationsaufgaben vorgestellt, die mit der Komponentensoftware für die Klassifikation ermittelt wurden. Die verschiedenen Synthesemethoden werden miteinander verglichen und ihre Leistungsfähigkeit beurteilt.

Voraussetzungen zur Synthese

Insgesamt wurden 13 Synthesemethoden aus Kapitel 4 in der Komponentensoftware für die Klassifikation implementiert. Von diesen Synthesemethoden wurden die folgenden eingesetzt:

- Simple Single Random: Ein Klassifikator wird zufällig ausgewählt. Es wird keine Merkmalsselektion durchgeführt. Dies ist die Implementierung der zufälligen Klassifikatorwahl.

- Single Random: Ein Merkmalsselektor und ein Klassifikator werden zufällig ausgewählt. Dies ist die Implementierung der zufälligen Klassifikatorwahl mit Merkmalsselektor.

- Zero Selector: Ein Klassifikator wird wiederholt und zufällig ausgewählt. Es wird keine Merkmalsselektion vorgenommen. Dies ist die Implementierung der wiederholten zufälligen Klassifikatorwahl.

- One Selector: Ein Klassifikator wird wiederholt und zufällig ausgewählt, wobei zuvor eine Merkmalsselektion mit einem bestimmten Merkmalsselektor durchgeführt wird. Der *Best-First-Selektor*, der auf dem Prinzip der stepwise-forward-selection basiert, liefert gute Ergebnisse bei geringer Laufzeit [Witten, 2000]. Dieser wird für die Merkmalssektion vorgegeben. Diese Implementierung ist eine Variante der wiederholten zufälligen Klassifikatorwahl mit Merkmalsselektor.

- Full Random: Merkmalsselektor und Klassifikator werden wiederholt zufällig ausgewählt. Dies ist die Implementierung der vollständigen zufälligen Klassifikatorwahl mit Merkmalsselektor.

- Promising List: Anhand einer vorgegebenen Liste werden Merkmalsselektoren und Klassifikatoren kombiniert. Die Liste ist nach beobachteter Erfolgsaussicht geordnet. Dies ist die Implementierung der vollständigen Liste von Klassifikatoren nach Erfolgsaussicht.

- Blasensynthese: Mit einer Teilmenge der Muster wird schrittweise die beste Kombination von Merkmalsselektor und Klassifikator für die gesamte Stichprobe geschätzt. Dies ist eine mögliche Umsetzung der Reduktion der Musteranzahl.

- Blasensynthese BF: Bei dieser Variante der Blasensynthese wird der *Best-First-Selektor* als Merkmalsselektor vorgegeben und es werden lediglich die Klassifikatoren variiert.

Der Generalisierungsfehler wird am besten durch eine vollständige Kreuzvalidierung abgeschätzt. Da die Abschätzung aufwendig ist und für große Klassifikationsprobleme (beginnend mit Musteranzahl größer 1000) zu inakzeptablen Laufzeiten führt, wurde stattdessen meist eine zehnfache Kreuzvalidierung verwendet. Diese verfügt über eine höhere Varianz als eine vollständige Kreuzvalidierung [Witten, 2000] und kann in Einzelfällen zu fehlerhaften Beurteilungen führen. Aus diesem Grund wurde die Synthese je nach Größe des Klassifikationsproblems zwischen fünf und 100 mal wiederholt.

Bei der Untersuchung der Synthesealgorithmen werden nur die ausgewählten Klassifikatoren der erstellten Klassifikationssysteme betrachtet. Einige der bei der Synthese verwendeten Klassifikatoren wurden in den Grundlagen lediglich erwähnt. Dies betrifft die Implementierung von Entscheidungsbäumen (J48), regelbasierten Klassifikatoren (PART) und Support-Vector-Machines (SMO) der Weka-Bibliothek. Die Funktionsweise und algorithmische Umsetzung dieser Klassifikatoren wird in [Witten, 2000] beschrieben.

Um die Untersuchungen zu beschleunigen, wurden auf bis zu fünf verschiedenen PCs gleichzeitig Klassifikationssysteme synthetisiert. Zur eindeutigen Vergleichbarkeit der Synthesemethoden wurde jedes Klassifikationsproblem jedoch immer auf demselben PC untersucht. Die PCs waren mit 450 bis 1000 MHz getakteten Pentium Prozessoren ausgestattet. Als Betriebssysteme wurden Linux mit

Kernel 2.x und Microsofts Win95, WindowsNT und Windows2000 eingesetzt. Als virtuelle Java-Maschinen wurden Java 2 kompatible Installationen von Version 1.2.x bis 1.3.x verwendet.

6.1 Verifikation der Blasensynthese

Relevant für eine Bewertung sind die Laufzeit in der praktischen Anwendung und die Güte der synthetisierten Klassifikationssysteme in Form des geschätzten Generalisierungsfehlers. Je geringer Laufzeit und Generalisierungsfehler sind, desto besser ist die Qualität einer Synthesemethode. Eine allgemeine und quantitative Bewertung kann empirisch geführt werden, wenn viele Klassifikationsprobleme untersucht und die Ergebnisse von Laufzeit und Generalisierungsfehler zusammengefaßt werden. Für eine Sortierung nach Güte ist es notwendig, Laufzeit und Generalisierungsfehler durch eine Funktion zusammenzufassen. Eine solche Funktion kann nur subjektiv gewählt werden. Für die gestellten Klassifikationsaufgaben werden Laufzeit und Generalisierungsfehler daher separat diskutiert.

Die Laufzeit ist absolut und kann einfach gemessen werden. Die Beurteilung des Generalisierungsfehlers ist relativ. Die Güte definiert sich dadurch, wie nahe der geschätzte Generalisierungsfehler einem theoretisch erreichbaren minimalen Fehler kommt. Dazu muß das erreichbare Minimum bekannt sein, was im allgemeinen nicht der Fall ist. Es bestehen zwei Möglichkeiten, dieses Minimum zu bestimmen:

- Der minimal erreichbare Fehler kann durch Test aller Kombinationen von Merkmalsselektoren und Klassifikatoren ermittelt werden. Das unter den gegebenen Einschränkungen aus Kapitel 4 beste Klassifikationssystem durch Test aller Kombinationen aufzubauen, ist aus Komplexitätsgründen nahezu unmöglich (Dies ist ja die eigentliche Motivation für diese Arbeit). Diese Methode könnte bestenfalls für kleine Klassifikationsprobleme mit wenigen Mustern und Merkmalen eingesetzt werden. Die klassenweise Verteilung der Merkmale und damit der theoretisch optimale Klassifikator kann nur geschätzt werden.

- Generiert man künstliche Klassifikationsprobleme mit den Algorithmen aus Abschnitt 4.2, kann man den zu generierenden Generalisierungsfehler explizit vorgeben. Im direkten Vergleich kann ermittelt werden, wie nah der geschätzte Generalisierungsfehler eines synthetisierten Klassifikationssystems diesem theoretisch erreichbaren Fehler kommt. Durch die Vorgabe der klas-

senweisen Verteilungen der Merkmale ist sogar der theoretisch am besten geeignete Klassifikator bekannt.

Künstliche Klassifikationsprobleme

Zur Verifikation der Blasensynthese wurden mehrere unterschiedliche Klassifikationsprobleme untersucht. Als Parameter künstlich generierter Klassifikationsprobleme müssen die klassenweisen Verteilungen der Merkmale, der Generalisierungsfehler und die Anzahl der Klassen, Merkmale und Muster vorgegeben werden. Um eine robuste Abschätzung der Leistungsfähigkeit der Blasensynthese zu ermöglichen, wurden mehrere verschiedene Parametersätze verwendet. Die Parameter wurden lediglich durch die folgenden Überlegungen eingeschränkt, ansonsten aber willkürlich gewählt:

- Es sollen drei Klassifikationsprobleme mit einer mittleren, großen und sehr großen Anzahl von Mustern generiert werden. Festgelegt wurden 1000, 5000 und 10000 Muster. Derart große Klassifikationsprobleme lassen sich aus Komplexitätsgründen nur noch mit der Blasensynthese bearbeiten, weshalb sie besonders für deren Untersuchung geeignet sind.

- Die Anzahl der Merkmale soll unterschiedlich und im Rahmen realer Probleme sein. Das zehn-zu-eins Verhältnis von Mustern gegenüber Merkmalen soll übertroffen werden, um statistisch aussagekräftige Stichproben zu erhalten. Für die drei Klassifikationsprobleme wurden in obiger Reihenfolge 3, 20 und 2 Merkmale vorgegeben.

- Die Anzahl der Klassen soll gering sein, um bei der vorgegebenen Anzahl von Mustern und Merkmalen statistisch aussagekräftige Stichproben zu generieren. Für die Klassifikationsprobleme wurden in obiger Reihenfolge 5, 2 und 2 Klassen festgelegt. Zwei-Klassen Probleme sind besonders deshalb interessant, weil sich jedes Multi-Klassen Klassifikationsproblem in eine Anzahl sequentieller Zwei-Klassen Probleme zerlegen läßt.

- Die Generalisierungsfehler wurden im Rahmen realer Klassifikationsprobleme gewählt. Für die drei Klassifikationsprobleme wurden Generalisierungsfehler mit zehn, zwölf und fünf Prozent vorgegeben.

- Es sollen Klassifikationsprobleme mit gleichen Parametern, aber mit klassenweise unterschiedlicher Verteilung der Merkmale generiert werden. Es wurden jeweils drei Klassifikations-

probleme mit klassenweise normalverteilten und mit klassenweise gleichverteilten Merkmalen erstellt.

Insgesamt wurden mit den Algorithmen aus Abschnitt 4.2 sechs Klassifikationsprobleme generiert und anschließend mit Hilfe der Blasensynthese bearbeitet. Die Implementierung des Bayes-Klassifikators in der Komponentensoftware nimmt als Model klassenweise normalverteilte Merkmale an. Man erwartet von der Blasensynthese, daß sie für die klassenweise normalverteilten Klassifikationsprobleme den Bayes-Klassifikator auswählt, der für solche Probleme definitionsgemäß optimal ist. Für die klassenweise gleichverteilten Klassifikationsprobleme erwartet man vorwiegend Klassifikatoren wie Entscheidungsbäume, welche die scharfen Übergänge von Gleichverteilungen gut repräsentieren können.

6.1.1 Klassenweise gleichverteilte Merkmale

Klassifikationsproblem mit 1 000 Mustern

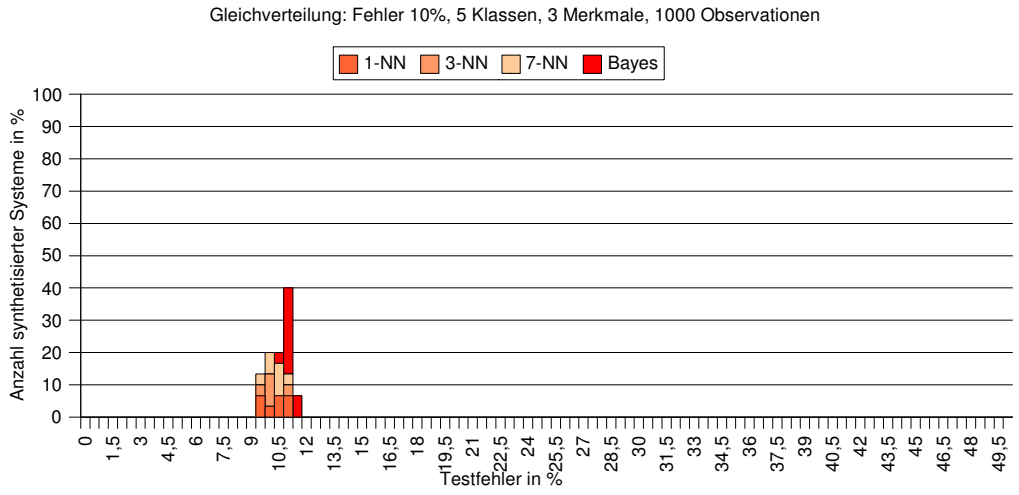


Abbildung 15. Klassifikationsproblem: 1000 Muster, 3 gleichverteilte Merkmale, 5 Klassen, vorgegebener Generalisierungsfehler 10%. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Der Datensatz wurde 30 mal mit der Blasensynthese bearbeitet. 12 (40%) der insgesamt 30 synthetisierten Klassifikationssysteme wiesen einen Generalisierungsfehler von 11% aus. An 8 dieser 12 Systeme war ein Bayes-Klassifikator beteiligt, an einem ein 7-nearest-Neighbour, an einem ein 3-nearest-Neighbour und an zweien ein 1-nearest-Neighbour. Die Blasensynthese erstellte in allen Fällen ein „gutes“ Klassifikationssystem, da sich sämtliche Generalisierungsfehler im Bereich der erwarteten 10% bewegen.

Für dieses Klassifikationsproblem wurden mit Hilfe der Blasensynthese 30 Klassifikationssysteme synthetisiert. Abbildung 15 zeigt die Ergebnisse in Form eines Diagramms. Die x-Achse gibt die

geschätzten Generalisierungsfehler der synthetisierten Systeme an. In Abbildung 15 bewegt sich der Generalisierungsfehler zwischen 9,5 und 11,5 Prozent, wobei die Werte auf 0,5 Prozent gerundet wurden. Die y-Achse gibt die Anzahl der Klassifikationssysteme an, welche über den entsprechenden Generalisierungsfehler der x-Achse verfügen. Die farbliche Kennung der Balken zeigt, wie oft ein bestimmter Klassifikator an dem jeweiligen Generalisierungsfehler beteiligt ist. Der Balken ganz links in Abbildung 15 besagt beispielsweise, daß von 30 synthetisierten Klassifikationssystemen 13,3 Prozent über einen gerundeten Generalisierungsfehler von 9,5 Prozent verfügen. Zwei der vier Systeme basieren auf einem 1-nearest-Neighbour Klassifikator, eines auf einem 3-nearest-Neighbour Klassifikator und eines auf einem 7-nearest-Neighbour Klassifikator.

Teilweise erreichten die Klassifikationssysteme Generalisierungsfehler von 9.5 Prozent. Dies ist niedriger als der in dieser Klassifikationsaufgabe als Minimum vorgegebene Generalisierungsfehler und wird durch verschiedene Effekte hervorgerufen: Zum einen verfügt die generierte Stichprobe möglicherweise über einen tatsächlich niedrigeren Generalisierungsfehler als 10 Prozent, da sie von der „Zufallsauswahl“ der Muster abhängig ist. Andererseits kann auch die zehnfache Kreuzvalidierung aufgrund ihrer Varianz (siehe Anhang C) zu Schätzfehlern führen.

Aufgrund der „scharfen“ Kanten der Gleichverteilungen wurden zunächst Klassifikatoren wie beispielsweise Entscheidungsbäume besonders häufig erwartet. Von der Blasensynthese wurden jedoch ausschließlich der Bayes-Klassifikator und k-nearest-Neighbour Klassifikatoren ausgewählt. Nachfolgende Tests verschiedener Klassifikatoren mit dieser Klassifikationsaufgabe bei zehnfacher Kreuzvalidierung haben gezeigt, daß Entscheidungsbäume oder regelbasierte Klassifikatoren im Mittel tatsächlich zu schlechteren Ergebnissen führen als der Bayes-Klassifikator oder k-nearest-Neighbour Klassifikator und die Blasensynthese somit zu korrekten Ergebnissen kam.

Obwohl Entscheidungsbäume in einem Teil der Ergebnisse aufgrund der Streuung des Generalisierungsfehlers ebenso gute Ergebnisse liefern wie k-nearest-Neighbour Klassifikatoren, werden sie von der Blasensynthese nicht für das letztendliche Klassifikationssystem ausgewählt. Dieser Effekt wird durch die konkrete Implementierung der Blasensynthese verursacht: Die Reihenfolge der zu untersuchenden Klassifikatoren ist alphabetisch vorgegeben. Bei der Sortierung der Klassifikatoren nach ihrem Generalisierungsfehler in den Zwischenschritten der Blasensynthese findet keine Vertauschung der Reihenfolge statt. Daher werden bei gleich guten Ergebnissen immer dieselben Klassifikatoren bevorzugt (beispielsweise „3NN“ vor „PART“).

Um die Leistungsfähigkeit der Blasensynthese besser abzuschätzen, wurde das Klassifikationsproblem mit anderen Synthesemethoden untersucht und die Ergebnisse wurden verglichen. Der Generalisierungsfehler wurde durch eine zehnfache Kreuzvalidierung geschätzt. Die Blasensynthese benötigte für die Synthese eines Klassifikationssystems zwischen 71 bis 162 Sekunden. Die Schwankungen der Laufzeit werden beispielsweise durch unterschiedlich gewählte Klassifikatoren verursacht. Zum Vergleich der Laufzeiten wurden 100 Klassifikationssysteme mit der Synthesemethode One Selector synthetisiert. Die Synthesezeit lag zwischen 14826 und 20883 Sekunden pro System bei einem Generalisierungsfehler von 10,2 bis 11,3 Prozent. Die Synthesemethode One Selector wählt in den meisten Fällen Klassifikatoren auf Basis von Entscheidungsbäumen aus. Für einen weiteren Vergleich wurde ein Klassifikationssystem mit der Synthesemethode Promising List synthetisiert. Die Synthesezeit betrug 641354 Sekunden (etwa 7,5 Tage). Der Generalisierungsfehler lag bei 10,3 Prozent und wurde mit einer zehnfachen Kreuzvalidierung geschätzt. Eine vollständige Kreuzvalidierung würde bei 1000 Mustern näherungsweise zu hundertfachem Zeitaufwand führen. Im Falle der Promising List würde die Synthese eines einzigen Klassifikationssystems mehr als zwei Jahre dauern. Die Blasensynthese beschleunigt die Synthese um einige Größenordnungen bei vergleichbarem Generalisierungsfehler.

Das Klassifikationsproblem wurde zusätzlich mit einer Variante der Blasensynthese untersucht. Bei der Blasensynthese BF wird der *Best-First-Selektor* vorgegeben. Die verschiedenen Klassifikatoren werden mit diesem Merkmalsselektor kombiniert. Mit der Blasensynthese BF konnten 100 Systeme synthetisiert werden, da die Variante erheblich weniger Laufzeit benötigt. Dabei wurden fast ausschließlich Bayes-Klassifikatoren und k-nearest-Neighbour Klassifikatoren als beste Klassifikatoren bestimmt. Die geschätzten Generalisierungsfehler gruppieren sich alle in der Nähe des vorgegebenen Generalisierungsfehlers von zehn Prozent. Die Synthesezeiten liegen zwischen 14 und 33 Sekunden. Sie sind noch einmal erheblich geringer als die der gewöhnlichen Blasensynthese. Die Ergebnisse sind allerdings unter dem Aspekt zu betrachten, daß der tatsächliche Generalisierungsfehler gleichmäßig über alle Merkmale verteilt ist. Eine Merkmalsselektion kann nicht zu einer Verbesserung führen und ist tatsächlich überflüssig. Für reale Klassifikationsprobleme trifft dies nur in Ausnahmefällen zu.

Klassifikationsproblem mit 5 000 Mustern

In Abbildung 16 ist das Ergebnis von 19 synthetisierten Klassifikationssystemen zu sehen. Die Synthesezeit betrug zwischen 887 bis 2767 Sekunden pro System. Auffallend ist zunächst die Trennung in

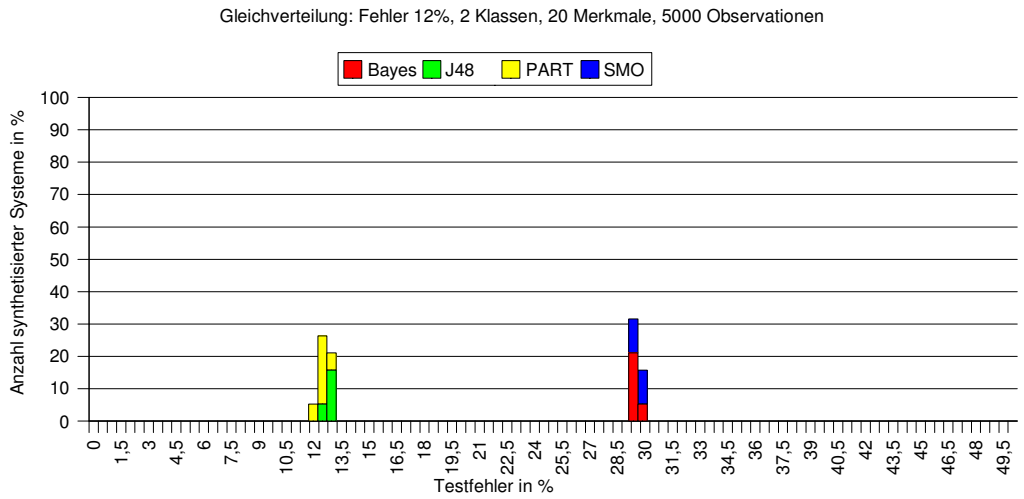


Abbildung 16. Klassifikationsproblem: 5000 Muster, 20 gleichverteilte Merkmale, 2 Klassen, 12% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Der Klassifikator J48 basiert auf Entscheidungsbäumen, der Klassifikator PART ist regelbasiert und SMO ist die Implementierung einer Support-Vector-Machine. Auffallend ist die Spaltung in zwei Gruppen mit deutlichem Unterschied zwischen den Generalisierungsfehlern. Diese deutliche Abweichung trat nur bei dieser einen Klassifikationsaufgabe auf.

zwei Gruppen mit großer Differenz der Generalisierungsfehler, die bei keinem anderen Klassifikationsproblem auftrat. In etwa der Hälfte aller Fälle wurden Entscheidungsbäume ausgewählt. In den restlichen Fällen wurde in den Zwischenschritten der Blasensynthese jedoch ein anderer Klassifikator gewählt. Diese sind für das vollständige Klassifikationsproblem aber ungeeignet und führen zu extrem schlechten Generalisierungsfehlern. Bei der Tendenzbestimmung liegen die Klassifikatoren derart nahe beieinander, daß beispielsweise geringfügige Fehler bei der Bestimmung des Generalisierungsfehlers zur Auswahl eines weniger gut geeigneten Klassifikators führen. Eine vollständige Kreuzvalidierung könnte die Synthese möglicherweise verbessern. Die Schwankungen der Synthesezeit werden durch unterschiedlich lange Trainingszeiten der ausgewählten Klassifikatoren verursacht. Im Gegensatz zu den 1000 Sekunden Synthesezeit eines Bayes-Klassifikators benötigt ein regelbasierter Klassifikator etwa 2500 Sekunden.

Klassifikationsproblem mit 10 000 Mustern

Das Verhältnis von Klassen und Merkmalen zu Mustern ist bei diesem Klassifikationsproblem sehr gut. Abbildung 17 zeigt das Ergebnis von neun Syntheseversuchen. Aufgrund der hohen statistischen Aussagekraft der Stichprobe liegen die Generalisierungsfehler dem vorgegebenen Fehler des Klassifikationsproblems sehr nahe. Die Synthesezeit pro System betrug zwischen 213 und 1971 Sekunden. Die starken Schwankungen werden wiederum durch unterschiedliche Klassifikatorwahl verursacht. Besonders viel Zeit mit etwa 1900 Sekunden pro Synthese muß bei den k-nearest-Neighbour Klassifikatoren aufgewendet werden.

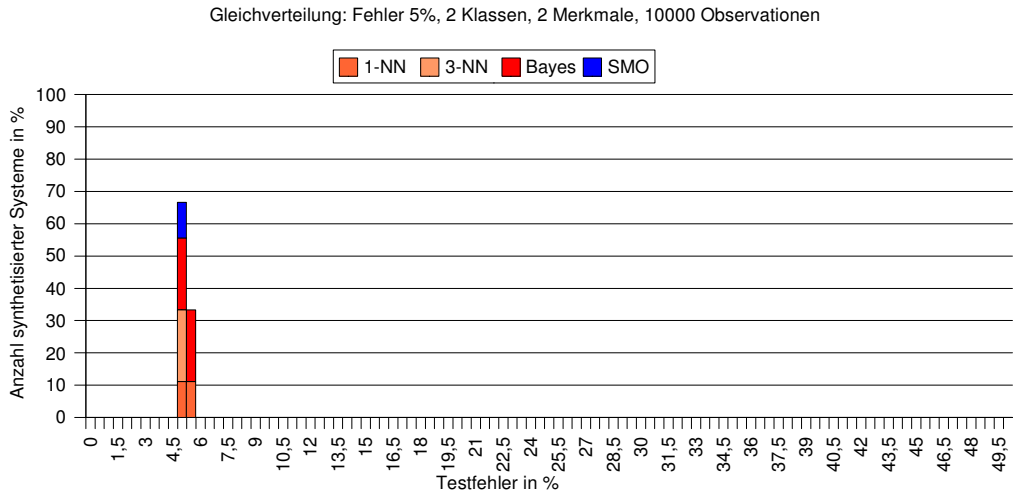


Abbildung 17. Klassifikationsproblem: 10000 Muster, 2 gleichverteilte Merkmale, 2 Klassen, 5% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Für alle synthetisierten Klassifikationssysteme liegt der Generalisierungsfehler nahe bei 5%.

6.1.2 Klassenweise normalverteilte Merkmale

Klassifikationsproblem mit 1 000 Mustern

Abbildung 18 zeigt die Zusammensetzung der synthetisierten Klassifikationssysteme für das Klassifikationsproblem mit den 1000 Mustern. Etwa ein Drittel der Systeme basiert auf einem Bayes-Klassifikator. Dieser liefert mit zehn Prozent exakt den erwarteten Generalisierungsfehler. Die übrigen Systeme setzen sich wiederum aus k-nearest-Neighbour Klassifikatoren zusammen. Überraschend schlecht schneidet der 1-nearest-Neighbour Klassifikator (15 Prozent aller durch die Blasensynthese erstellten Systeme) mit relativ hohem Fehler ab. Dieser Effekt kann vermieden werden, wenn für die Tendenzbestimmung in den Zwischenschritten der Blasensynthese eine vollständige Kreuzvalidierung zur Schätzung des Generalisierungsfehlers verwendet wird. Die Synthese der 100 Klassifikationssys-

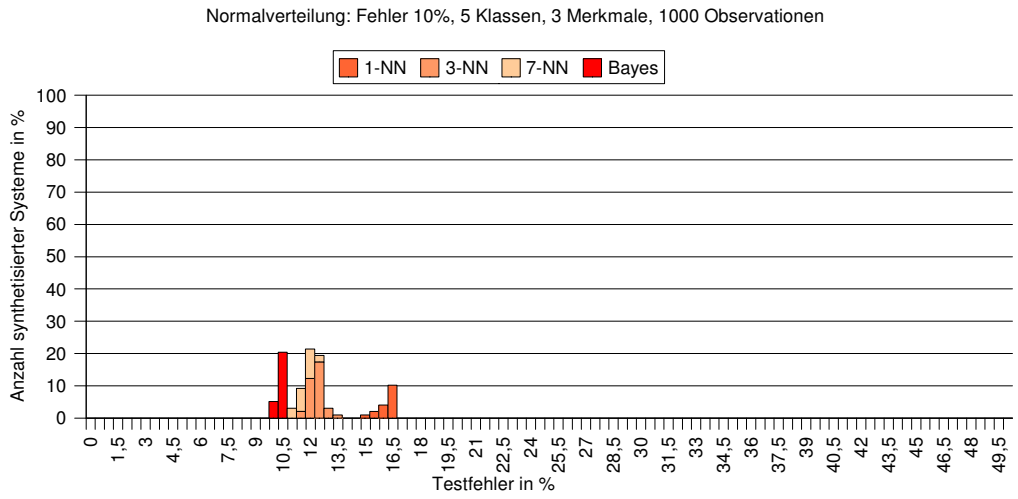


Abbildung 18. Klassifikationsproblem: 1000 Muster, 3 normalverteilte Merkmale, 5 Klassen, 10% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Erwartungsgemäß ist der Bayes-Klassifikator für diesen Datensatz am besten geeignet. Bei etwa einem sechstel der synthetisierten Klassifikationssysteme ist der Generalisierungsfehler größer als 14%. Ein Erklärung hierfür wird im Text gegeben.

teme benötigte jeweils zwischen 171 und 254 Sekunden. Keine andere Synthesemethode erreicht in derart kurzer Zeit derartig gute Klassifikationsergebnisse. Manche Synthesemethoden können aufgrund ihrer Komplexität überhaupt nicht auf dieses Klassifikationsproblem angewendet werden.

Die Vermutung, daß die Synthese bei Einsatz einer vollständigen Kreuzvalidierung verbessert wird, konnte im Anschluß an diese Ergebnisse verifiziert werden. Bei Einsatz von vollständiger Kreuzvalidierung verfügten von den 99 synthetisierten Klassifikationssystemen 91 über einen Generalisierungsfehler von 12,5 Prozent oder besser. Daher sind nur noch acht Prozent der erstellten Systeme als ungenügend einzustufen. Für diese Untersuchungen wurden auf einem 2,8 GHz Prozessor etwa 3,5 Tage benötigt.

Ein mit Abbildung 18 nahezu identisches Diagramm ergibt sich bei Verwendung der Blasensynthese BF. Allerdings ist die zur Synthese benötigte Zeit deutlich geringer. Die 100 durch synthetisierten Systeme wurden in nur 14 bis 35 Sekunden synthetisiert. Dies bedeutet gegenüber der gewöhnlichen Blasensynthese noch einmal einen Zeitgewinn von etwa einer Größenordnung.

Klassifikationsproblem mit 5 000 Mustern

Das Ergebnis der 16 Synthesen entspricht auf den ersten Blick den Erwartungen. Etwa die Hälfte aller Systeme basiert auf einem Bayes-Klassifikator. Die Generalisierungsfehler befinden sich in der Nähe

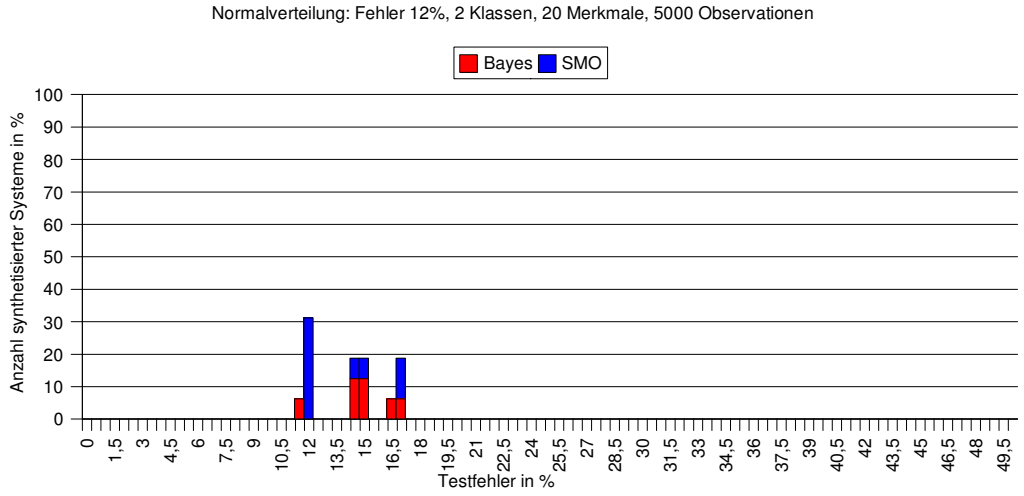


Abbildung 19. Klassifikationsproblem: 5000 Muster, 20 normalverteilte Merkmale, 2 Klassen, 12% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Auffallend ist die häufige Auswahl des Bayes-Klassifikators, wie er allerdings für diesen Datensatz auch erwartet wurde. Die Generalisierungsfehler weichen leicht von den vorgegebenen 12% ab, bewegen sich aber in der Nähe dieses Wertes.

der vorgegebenen zwölf Prozent des Klassifikationsproblems. Abbildung 19 zeigt die Anteile der Klassifikatoren an den Generalisierungsfehlern. Die Fehler ganz rechts im Bereich von 16 Prozent sollten für den implementierten Bayes-Klassifikator bei normalverteilten Klassifikationsproblemen niedriger sein. Für gleiche Klassifikatoren sollte grundsätzlich der gleiche Generalisierungsfehler bestimmt werden. Eine mögliche Erklärung liegt in der speziellen Kombination von Merkmalsanzahl, Musteranzahl und einer zehnfachen Kreuzvalidierung für dieses Klassifikationsproblem, wodurch die Varianz bei der Abschätzung des Generalisierungsfehler steigt. Im Mittel nähern sich die Generalisierungsfehler der synthetisierten Systeme zwar den erwarteten zwölf Prozent an, in Einzelfällen kommt es aber durch die stärkere Streuung vermehrt zu Ausreißern, bedingt durch unglückliche Unterteilung der Stichprobe. Dies kann nur durch eine höhere Anzahl von Syntheseversuchen geklärt werden.

Nachträglich konnte der Einfluß einer vollständigen Kreuzvalidierung untersucht werden. Abbildung 20 zeigt für dieselbe Klassifikationsaufgabe das Ergebnis bei Verwendung einer vollständigen Kreuzvalidierung. Die Ergebnisse der Synthese sind deutlich besser. Dies beweist, daß die zehnfache Kreuzvalidierung zu schlechteren Generalisierungsfehlern führen kann. Möglicherweise könnte eine adaptive Schätzmethode die Qualität der Blasensynthese verbessern. Der Nachteil der vollständigen Kreuzvalidierung liegt in der eindeutig höheren Laufzeit (etwa 3,5 Tage für 40 Klassifikationssysteme gegenüber wenigen Stunden).

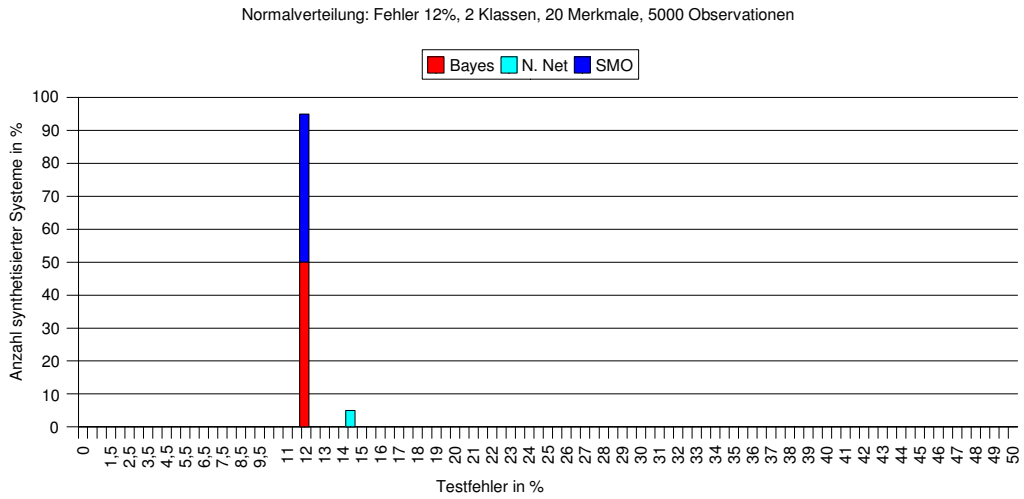


Abbildung 20. Klassifikationsproblem: 5000 Muster, 20 normalverteilte Merkmale, 2 Klassen, 12% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler, wobei eine vollständige Crossvalidierung für die Fehlerschätzung gewählt wurde. Der mittlere Generalisierungsfehler wird dadurch deutlich verbessert.

Zum Vergleich von Qualität und Aufwand wurden die Synthesemethoden Single Random und One Selector mit dem Klassifikationsproblem untersucht. One Selector benötigte für 24 Systeme jeweils zwischen 17264 und 42788 Sekunden bei Generalisierungsfehlern um zwölf Prozent. Single Random benötigte für 100 Systeme jeweils zwischen 39 und 214266 Sekunden (etwa 2,5 Tage) bei Generalisierungsfehlern zwischen zwölf und 50 Prozent. Die lange Laufzeit wird hauptsächlich durch die Komplexität der Merkmalsselektion verursacht, die mit der Anzahl der Merkmale und Muster stark ansteigt. Die Blasensynthese benötigt hingegen nur zwischen 1207 und 14431 Sekunden. Die starken Schwankungen der Blasensynthese wird durch die Auswahl der beiden unterschiedlichen Klassifikatoren verursacht. Der Klassifikator SMO[1] benötigt für das Training erheblich mehr Zeit als der Bayes-Klassifikator. Der Zeitgewinn der Blasensynthese ist jedoch offensichtlich.

Wiederum ergibt sich bei Verwendung der Blasensynthese BF ein mit der gewöhnlichen Blasensynthese nahezu identisches Diagramm. Es wurden 100 Klassifikationssysteme synthetisiert. Für je ein System wurde zwischen zwölf und 66 Sekunden benötigt. Auffallend ist allerdings das Fehlen von Bayes-Klassifikatoren und der große Anteil an k-nearest-Neighbour Klassifikatoren. Die Einschränkung auf eine bestimmte Merkmalsselektion wirkt sich in den Zwischenschritten der Blasensynthese BF offensichtlich negativ aus.

1 *Der Klassifikator ist die Implementierung einer Support-Vector-Machine [Witten, 2000].*

Klassifikationsproblem mit 10 000 Mustern

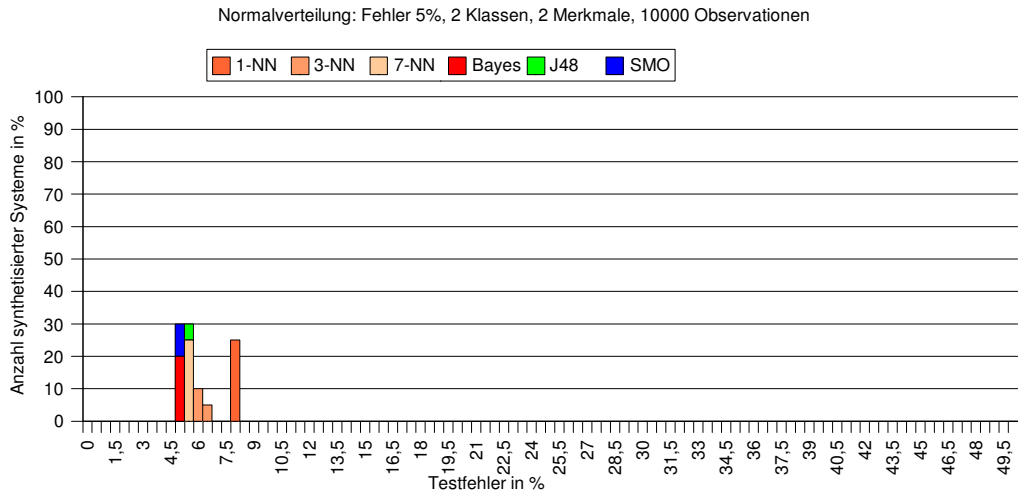


Abbildung 21. Klassifikationsproblem: 10000 Muster, 2 normalverteilte Merkmale, 2 Klassen, 5% Fehler. Das Diagramm zeigt die Verteilung der durch die Blasensynthese ausgewählten Klassifikatoren über den Generalisierungsfehler. Bei etwa einem Viertel der synthetisierten Systeme wird der 1-nearest-Neighbour Klassifikator verwendet. Er ist offensichtlich nicht besonders gut für das Klassifikationsproblem geeignet Die Generalisierungsfehler der anderen Systeme bewegen sich alle im erwarteten Bereich um 5%.

Abbildung 21 zeigt die Verteilung der Klassifikatoren an den Generalisierungsfehlern für dieses Klassifikationsproblem. Auffallend ist zunächst der große Anteil von k-nearest-Neighbour Klassifikatoren, obwohl Bayes-Klassifikatoren dominieren sollten. Während der Tendenzbestimmung in den Zwischenschritten der Blasensynthese werden k-nearest-Neighbour Klassifikatoren bei dieser Klassifikationsaufgabe offensichtlich bevorzugt. Die Generalisierungsfehler liegen im erwarteten Rahmen. Lediglich die Ausreißer bedingt durch den 1-nearest-Neighbour Klassifikator verursachen einen Fehler von acht Prozent. Stuft man diese als fehlerhafte Synthese ein, liegt der Anteil korrekt synthetisierter Klassifikationssysteme bei etwa 75 Prozent. Für eine einzelne Synthese wurden zwischen 174 bis 2127 Sekunden benötigt. Das Klassifikationsproblem wurde zum Vergleich der Laufzeit mit der Promising List Synthesemethode bearbeitet. Der Generalisierungsfehler von acht synthetisierten Klassifikationssystemen lag zwischen 5,2 und 5,4 Prozent bei einer Laufzeit von jeweils etwa 58000 Sekunden (16 Stunden). Auch mit dieser Synthesemethode wurde nur in zwei Fällen der Bayes-Klassifikator als optimaler Klassifikator bestimmt. In sechs Fällen wurde ein anderer Klassifikator gewählt. Dies könnte entweder durch die zehnfache Kreuzvalidierung bedingt sein. Der Zeitgewinn durch Blasensynthese beträgt für dieses Klassifikationsproblem zwischen ein und zwei Größenordnungen gegenüber anderen Synthesemethoden.

6.2 Anwendung auf reale Klassifikationsaufgaben

Als konkrete Anwendung wurden Klassifikationsaufgaben aus dem Gebiet der Medizintechnik mit der Komponentensoftware ICE bearbeitet. Diese Aufgaben werden zusammen mit den Ergebnissen der Synthese beschrieben. Der Nutzen einer Komponentensoftware für die Klassifikation konnte anhand dieser Klassifikationsaufgaben bestätigt werden. Die Generalisierungsfehler wurden in den meisten Fällen durch zehnfache Kreuzvalidierung geschätzt. Einige Klassifikationsprobleme wie beispielsweise die aus Coocurrence-Matrizen gewonnenen Daten für die Brustkrebsdiagnose sind von solcher Größe, daß auch mit einer zehnfachen Kreuzvalidierung keine vollständige Analyse mehr durchgeführt werden konnte. Bei diesem Klassifikationsproblem mußten einige Synthesemethoden nach mehreren Wochen Laufzeit ergebnislos abgebrochen werden. Nur Synthesemethoden mit geringer Komplexität konnten hierfür noch eingesetzt werden. Die Werte von Wiederholungen und verwendeten Fehlerabschätzungen werden bei den jeweiligen Klassifikationsaufgaben mit angegeben.

Die Klassifikationsaufgaben im folgenden Abschnitt befassen sich mit der Brustkrebsdiagnose und entstammen teils eigenen Arbeiten sowie teils der Weka-Bibliothek. Im darauf folgenden Abschnitt werden die Ergebnisse weiterer Klassifikationsaufgaben zusammengefaßt, die im Anhang mit ausführlichen Tabellen und Kommentaren vorgestellt werden.

6.2.1 Klassifikation von Mammogrammausschnitten

Die Merkmale dieser Klassifikationsaufgabe wurden mit Hilfe von Coocurrence-Matrizen aus Mammogrammausschnitten wie in Abbildung 22 extrahiert [Müller, 2001a] [Yang, 2000]. Der Datensatz enthält die beiden Klassen „enthält Mikroverkalkungen“ und „enthält keine Mikroverkalkungen“. Für jede Klasse wurden 128 Muster in Form von Bildausschnitten erstellt. Aus den Bildausschnitten wurden durch summierte Coocurrence-Matrizen (64 mal 64) 4096 Merkmale extrahiert. Das manuell erstellte Klassifikationssystem für dieses Klassifikationsproblem ergab einen Generalisierungsfehler von 38 Prozent. Aufgrund der hohen Anzahl von Merkmalen und der geringen Anzahl von Mustern entspricht der Generalisierungsfehler den Erwartungen. Einzelne gute Merkmale können durch eine Merkmalsselektion selektiert werden, jedoch lassen sich aufgrund der geringen Größe der Stichprobe eventuelle Zusammenhänge zwischen den Merkmalen nicht erkennen.

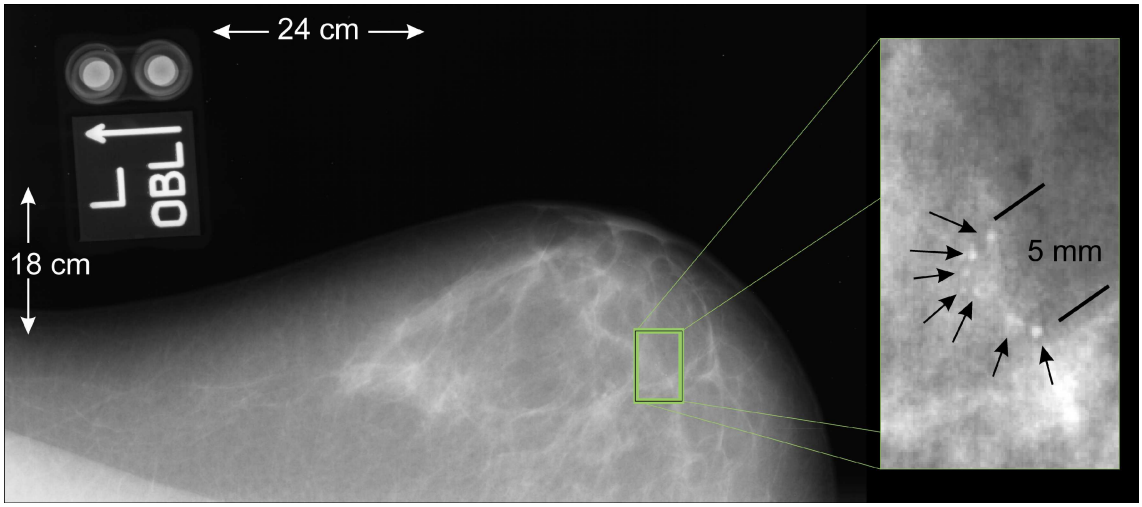


Abbildung 22. Röntgenbild einer weiblichen Brust (Mammogramm). Mammogramme werden für die Detektion von Brustkrebs verwendet. Indizien für Brustkrebs sind Mikroverkalkungen. Die Ausschnittsvergrößerung (Region of Interest) zeigt eine Gruppe besonders deutlich sichtbarer Mikroverkalkungen.

Tabelle 3 zeigt die Ergebnisse des Zero Selectors, des One Selectors, der Promising List und der Blasensynthese für die Klassifikationsaufgabe. Falls die Synthese mehrmals wiederholt wurde, ist als Fehler der mittlere Fehler zusammen mit der Varianz angegeben. Die Blasensynthese liefert das beste Ergebnis. Eigentlich sollte die Promising List bei vollständiger Abarbeitung der Liste das beste Ergebnis liefern, da alle Kombinationen von Klassifikatoren und Selektoren getestet werden. Tatsächlich ist der Generalisierungsfehler um mehr als vier Prozent schlechter als bei der Blasensynthese. Dieser große Unterschied kann nur durch die zehnfache Kreuzvalidierung verursacht werden, da Promising List und Blasensynthese zu gleichen Ergebnissen kommen müssen (und dies bei anderen Klassifikationsaufgaben auch tun). Eine vollständige Kreuzvalidierung würde bei geschätzter linearer Zunahme des Zeitaufwandes 200 Tage benötigen. Der deutliche Zeitverlust der Promising List gegenüber der Blasensynthese ist durch das systematische Abarbeiten der Liste bedingt, da ganze Kombinationen getestet und dadurch Merkmalsselektoren mehrfach berechnet werden. Aufgrund der großen Anzahl von Merkmalen macht sich der Effekt der Merkmalsselektion in diesem Fall deutlich bemerkbar. In Bezug auf Klassifikationsfehler und Zeitaufwand ist der One Selector für dieses Klassifikationsproblem am effektivsten.

Eigenschaften: ca. 38% Fehler, 4096 Merkmale, 2 Klassen und 256 Muster.

Synthesemethode	Test	Zeit in s	GF in % (σ)
Zero Selector	10-fold	3576 – 5182	40,37 (2,36)
One Selector	10-fold	77 – 88	40,13 (0,08)
Promising List	10-fold	648687	39,71
Blasensynthese	10-fold	60556	35,35

Tabelle 3. Generalisierungsfehler und Laufzeiten verschiedener Synthesemethoden. Die Merkmale wurden mit Hilfe von Coocurrence-Matrizen aus Röntgen-Mammogrammen extrahiert.

Eigenschaften: ca. 21% Fehler, 85 Merkmale, 2 Klassen und 1004 Muster.

Synthesemethode	Test	Zeit in s	GF in % (σ)
Zero Selector	10-fold	1531 – 1589	22,17 (4,98)
One Selector	10-fold	1231 – 2332	27,74 (0,49)
Promising List	10-fold	20944	6,47 (0,00)
Blasensynthese	10-fold	7428 – 8227	6,45 (0,33)

Tabelle 4. Generalisierungsfehler und Laufzeiten verschiedener Synthesemethoden. Die Merkmale wurden mit Hilfe von hybriden neuronalen Netzen aus Röntgen-Mammogrammen extrahiert.

Aus denselben Bildausschnitten wurden anschließend neue Merkmale mit Hilfe hybrider neuronaler Netze extrahiert [Stotzka, 2000]. Um die Größe der Stichprobe zu erhöhen und um eine Rotationsinvarianz zu erreichen, wurden die Bildausschnitte vor der automatischen Merkmalsextraktion dreimal um je 90 Grad rotiert. Die Anzahl der zur Verfügung stehenden Muster wurde dadurch vervierfacht. Die Muster wurden anschließend mit einem Case-Based-Reasoning System (CBR) klassifiziert [Korf, 2000]. Der CBR lieferte einen Generalisierungsfehler von 21 Prozent. Für die Umsetzung des CBR wurden näherungsweise drei Wochen benötigt. Tabelle 4 zeigt die Ergebnisse einiger Synthesemethoden. Die Blasensynthese erstellte in einer Zeit von etwa zwei Stunden ein Klassifikationssystem mit einem Generalisierungsfehler von nur noch sechs Prozent. Selbst die vollständige Untersuchung durch die Promising List benötigt weniger als sechs Stunden für die Synthese eines optimalen Klassifikationssystems. Der Suche nach einem geeigneten Klassifikationssystem wurde somit erheblich beschleunigt, wobei gleichzeitig eine enorme Verbesserung des Generalisierungsfehler erreicht werden konnte.

6.2.2 Synthese von Klassifikationssystemen für verschiedene Aufgabenbereiche

Sämtliche mit der Komponentensoftware ICE bearbeiteten Klassifikationsaufgaben deuten darauf hin, daß der Entwurf von Klassifikationssystemen durch die Automatisierung enorm erleichtert wird. Insbesondere die Blasensynthese liefert innerhalb kürzester Zeit Klassifikationssysteme mit guten Generalisierungsfehlern [Müller, 2004]. Der Übersichtlichkeit halber werden die Ergebnisse tabellarisch zusammengefaßt. Der direkte Vergleich der Ergebnisse der verschiedenen Synthesemethoden und deren Interpretation befindet sich in Tabellenform im Anhang. Das Problem der Lymphozytenklassifikation stammt aus [Berting, 2000], die anderen Klassifikationsprobleme sowie deren bislang beste Generalisierungsfehler stammen aus der Weka-Bibliothek [Witten, 2000] und werden im Anhang beschrieben. Für jedes Klassifikationsproblem wird die benötigte Zeit für die Synthese, der mit Hilfe der Blasensynthese erreichte Generalisierungsfehler und der bislang beste Generalisierungsfehler aufgelistet. Die folgenden Klassifikationsprobleme wurden untersucht:

Klassifikationsproblem	Zeit in Min	GF in % (σ)	GF aus Lit. in %
Kontaktlinsen	1	16,67 (0,00)	17,0
Tiergattungen	2	1,95 (0,05)	2,0
Iris-Klassifikationsproblem	3	3,5 (0,29)	3,3
Wetterdaten	3	17,0 (2,45)	15,0
Herzerkrankungen	3	14,05 (1,08)	21,0
Lymphozytenklassifikation	6	1,58 (0,00)	2,5
Lymphographie	6	13,25 (0,72)	15,0
Diabetes	6	21,66 (2,50)	24,0
Hepatitis	11	12,17 (0,31)	17,0
Gehaltsverhandlungen	15	3,51 (0,00)	3,5
Pferde-Kolik	40	13,25 (0,80)	13,0
Soja-Bohnen	400	7,64 (0,24)	3,0

Tabelle 5. Die Tabelle zeigt die benötigten Laufzeiten und Generalisierungsfehler (inklusive Standardabweichung) der Blasensynthese, die wiederholt auf verschiedene Klassifikationsaufgaben angewandt wurde. Für die verschiedenen Klassifikationsaufgaben ist in der letzten Spalte der bislang beste aus der Literatur bekannte Generalisierungsfehler angegeben.

Für alle gestellten Klassifikationsaufgaben wurde in kürzester Zeit ein Klassifikationssystem erstellt, das dem besten bisher bekannten Klassifikationssystem zumindest ebenbürtig war. In vielen Fällen wurden sogar niedrigere Generalisierungsfehler erreicht. Der Datensatz für die Lymphozytenklassifikation wurde beispielsweise im Rahmen einer Diplomarbeit erstellt. Für den manuellen Entwurf des Klassifikationssystems wurden zwei Wochen benötigt. Das erstellte Klassifikationssystem erreichte einen Generalisierungsfehler von 2,5 Prozent. Im Gegensatz dazu liefert die Blasensynthese nach etwa sechs Minuten ein Klassifikationssystem mit einem Generalisierungsfehler von 1,6 Prozent. Die Blasensynthese führte nicht nur zu einem beachtlichen Zeitgewinn, sondern sogar zu einem besseren Generalisierungsfehler.

6.3 Zusammenfassung

Mit Hilfe künstlicher Klassifikationsprobleme von bekannter Verteilungsdichte und bekanntem Generalisierungsfehler konnte die Blasensynthese verifiziert werden. Für sechs verschiedene Klassifikationsaufgaben wurden Klassifikationssysteme in Serie erstellt und ihre Generalisierungsfehler ermittelt. Die Ergebnisse zeigen, daß die Blasensynthese mit hoher Zuverlässigkeit sehr gute Klassifikationssysteme in der Nähe des tatsächlichen Generalisierungsfehlers erzeugt. Je nach Größe der gestellten Klassifikationsaufgabe wird für die Synthese zwischen wenigen Minuten und einigen Stunden benötigt.

Mit der Komponentensoftware für die Klassifikation wurden reale Klassifikationsaufgaben bearbeitet, deren bislang „bester“ Generalisierungsfehler aus der Literatur bekannt war. Die Klassifikationsaufgaben wurden mit verschiedenen Synthesemethoden behandelt, die entsprechend ihrer Komplexität zu unterschiedlich guten Generalisierungsfehlern führten. Die Blasensynthese lieferte die besten Ergebnisse in Bezug auf benötigter Zeit und erzieltem Generalisierungsfehler. In allen Fällen wurde der bislang beste Generalisierungsfehler erreicht, oftmals sogar unterboten. Hierzu wurden je nach Größe der gestellten Klassifikationsaufgabe wenige Minuten bis einige Stunden benötigt. Für vergleichbare manuelle Entwicklungen von Klassifikationssystemen ohne die Komponentensoftware ICE mußten beispielsweise 14 Tage oder mehr aufgewandt werden. Damit wurde das Ziel der effizienten automatischen Synthese von Klassifikationssystemen erreicht.

7 Diskussion und Ausblick

Zu Beginn dieser Arbeit stand die Anforderung nach einem effektiven Entwicklungswerkzeug für die Klassifikation, welches den umständlichen und langwierigen manuellen Entwicklungsprozeß von Klassifikationssystemen vereinfacht und beschleunigt. Als Ergebnis dieser Arbeit steht nun ein solches Entwicklungswerkzeug in Form einer Komponentensoftware bereit, welches den Entwurf von Klassifikationssystemen mit Hilfe von speziellen Synthesemethoden automatisiert. Sowohl die Komponentensoftware für die Klassifikation als auch die Synthesemethoden wurden im Rahmen der Arbeit entwickelt und implementiert. Die Komponentensoftware für die Klassifikation ist zur Zeit das einzige bekannte Entwicklungswerkzeug, das den Entwurf von Klassifikationssystemen komfortabel unterstützt und automatisiert.

7.1 Die Synthesemethoden für den automatischen Entwurf

Der automatische Entwurf von Klassifikationssystemen stellt ein NP-vollständiges Problem dar. Um in der Praxis dennoch eine Unterstützung des Entwurfsprozesses zu ermöglichen, mußten vereinfachende Annahmen beispielsweise durch die Beschränkung der Kombinationsmöglichkeiten von Klassifikationsalgorithmen getroffen werden. Das hieraus resultierende zweidimensionale Optimierungsproblem kann mit herkömmlichen Optimierungsmethoden nicht gelöst werden. Zur effizienten Lösung dieses Problems wurden daher eigene Ansätze verfolgt, die als ***Synthesemethoden*** bezeichnet werden.

Die entwickelten Synthesemethoden wurden mit realen und künstlich konstruierten Klassifikationsaufgaben untersucht. Einfache Synthesemethoden erstellen in kurzer Zeit (wenigen Sekunden) Klassifikationssysteme, führen aber oftmals zu unzureichenden Generalisierungsfehlern im Vergleich zum möglichen besten Generalisierungsfehler. Komplexere Synthesemethoden produzieren Klassifikationssysteme mit gutem Generalisierungsfehler, benötigen dafür aber entsprechend mehr Zeit. Die benötigte Zeitspanne variiert je nach Größe der Klassifikationsaufgabe und Schätzmethode des Generalisierungsfehlers zwischen wenigen Sekunden und mehreren Tagen.

Als besonders leistungsfähig hat sich das Konzept der iterativen Schätzung des besten Klassifikationssystems erwiesen. Ein implementierter Vertreter dieses allgemeinen Konzepts ist die ***Blasensynthese***.

Die Untersuchungen von insgesamt 15 realen Klassifikationsaufgaben haben gezeigt, daß gerade diese Synthesemethode in akzeptabler Zeit Klassifikationssysteme mit gutem Generalisierungsfehler erstellt. Für alle getesteten Klassifikationsaufgaben wurden Generalisierungsfehler erreicht, die zumeist besser waren als die bislang bekannten Generalisierungsfehler, jedoch niemals schlechter. Hierfür wurde auf herkömmlichen PCs je nach Größe der Klassifikationsaufgabe zwischen wenigen Minuten und einigen Stunden benötigt. Die manuelle Bearbeitung einer konkreten Klassifikationsaufgabe aus der Medizintechnik erforderte zwei Wochen. Für dieselbe Klassifikationsaufgabe erstellte die Blasensynthese ein Klassifikationssystem in nur fünf Minuten, das sogar über einen besseren Generalisierungsfehler verfügte. Bei einer weiteren Klassifikationsaufgabe aus der Medizintechnik konnte der bislang beste Generalisierungsfehler von 21 auf sechs Prozent reduziert werden. Anstelle von drei Wochen für den manuellen Entwurf wurden für den automatischen lediglich zwei Stunden benötigt.

Die Blasensynthese ist nicht gleichermaßen gut für alle Klassifikationsprobleme geeignet. Jedoch wurden nur bei einer einzigen Klassifikationsaufgabe zum Teil Klassifikationssysteme erstellt, die über einen deutlich schlechteren Generalisierungsfehler verfügten. Die konkrete Implementierung der Blasensynthese ist besonders leistungsstark, sobald das Verhältnis von Mustern zu Merkmalen zehn zu eins übersteigt. Je mehr sich dieses Verhältnis eins zu eins nähert, desto mehr degeneriert die Blasensynthese zu einer systematischen Aufzählung aller möglichen Kombinationen des zweidimensionalen Optimierungsproblems. Allerdings ist in der Praxis bei solch ungünstigen Verhältnissen die Stichprobe meist entsprechend „klein“, wodurch die für eine systematische Aufzählung benötigte Zeit oftmals auch geringer ist.

Im Rahmen dieser Arbeit wurden einige Synthesemethoden entwickelt, die jedoch nicht implementiert wurden. Diese Methoden werden im Anhang „*Komplexe Synthesemethoden*“ ausführlich erörtert. Zu ihnen gehören beispielsweise Case-Based-Reasoning, Meta-Klassifikation, genetische Algorithmen und andere. Eine Implementierung solcher Methoden ist sehr komplex, weshalb bislang darauf verzichtet wurde. Es ist fraglich, ob diese Synthesemethoden schneller zu besseren Klassifikationssystemen führen als die bisher implementierten Methoden und ob der benötigte Mehraufwand durch den möglichen Gewinn gerechtfertigt wird. Diese Untersuchungen bleiben zukünftigen Arbeiten überlassen.

Steht eine entsprechend große Zahl von Rechnern oder gar ein GRID zur Verfügung, könnte durch eine verteilte Synthese nochmals eine enorme Leistungssteigerung von zwei bis drei Größenordnun-

gen erreicht werden. Die Komponentensoftware ICE könnte leicht an eine parallele Verarbeitung angepaßt werden. Im wesentlichen ist hierzu die Spezifikation eines Client-Server Protokolls notwendig. Die Programmiersprache Java bietet für eine Implementierung beispielsweise unter Zuhilfenahme von TCP/IP alle notwendigen Voraussetzungen, um schnell zu einer guten Umsetzung zu gelangen. Auf eine Implementierung wurde bislang verzichtet, da keine ausreichend große Zahl von Rechnern verfügbar war. In Zukunft wird am Forschungszentrum Karlsruhe jedoch Grid Computing eingeführt [GRID, 2004] [Alef, 2004]. In einem Grid stehen Tausende von Recheneinheiten in einer heterogenen Umgebung zur Verfügung. Damit ist bereits in naher Zukunft eine verteilte Synthese möglich.

Abschließend stellt sich die Frage, inwieweit die für die Synthese getroffenen Einschränkungen (keine Parameteroptimierung der Klassifikatoren, Beschränkung auf Kombination aus genau einem Merkmalsselektor und einem Klassifikator) gelockert werden könnten. Ohne diese Einschränkungen kann der Entwurf von Klassifikationssystemen bislang nicht automatisiert werden. Die Ergebnisse aus Kapitel 6 zeigen aber deutlich, daß trotz der Einschränkungen Klassifikationssysteme von exzellenter Qualität erzeugt werden.

7.2 ICE – Die Komponentensoftware für die Klassifikation

Kapitel 2 macht deutlich, daß kein verfügbares System als Ausgangsbasis für ein Entwicklungswerkzeug für die Klassifikation verwendet werden kann. Aus diesem Grund war zwangsläufig der Entwurf einer geeigneten Umgebung notwendig, um die Synthesemethoden aus Kapitel 4 zu implementieren und zu testen.

Aufgrund der Struktur von Klassifikationssystemen wurde für die konkrete Umsetzung das Konzept einer Komponentensoftware gewählt. Um den softwaretechnischen Aufwand zu minimieren, wurde soweit wie möglich auf bereits verfügbare Implementierungen zurückgegriffen. Als Basis der Klassifikationsalgorithmen wurde das Weka System [Witten, 2000] gewählt, das eine große Anzahl solcher Algorithmen bereit stellt. Die Umsetzung der Komponentensoftware erfolgte in Java und als Komponentenkonzept wurde JavaBeans [Flanagan, 1998] verwendet. Die graphische Oberfläche wurde nach dem Vorbild des Bildverarbeitungssystems Khoros aufgebaut. Neben den Klassifikationsalgorithmen stellt die entwickelte Komponentensoftware insbesondere die notwendige Meta-Ebene zur Verfügung, die eine Rückkopplung der Ergebnisse während eines automatischen Entwurfs überhaupt ermöglicht. Sie erlaubt Komponenten ein ***self-guided assembly***, wodurch eine Komponente ihre Funktionalität

dynamisch aus anderen Komponenten aufbauen kann. Eine solche Meta-Ebene ist zwingend erforderlich für die Implementierung der Synthesemethoden aus Kapitel 4.

Das Konzept der ***abstrakten globalen Datentypen*** mag zunächst idealistisch erscheinen. Obwohl der Erweiterungsprozeß dieser Datentypen langwierig ist, stellen sie bei konsequenter Umsetzung den einzigen Weg für den transparenten Einsatz von Komponenten dar. Von existierender Komponentensoftware wird diese Problematik meist ignoriert. Aufgrund der Bindung an bestimmte Datentypen bilden sich isolierte Gruppen von Komponenten, die nicht mehr mit anderen Gruppen interagieren können. Der eigentliche Sinn modularer Komponenten geht dadurch verloren.

Durch die Wahl der Programmiersprache Java kann die Komponentensoftware ICE plattformübergreifend und ohne Änderungen eingesetzt werden. Die Interaktion mit dem Benutzer und das visuelle Feedback einer Synthese konnte komfortabel gestaltet werden, da der Entwurf und die Wartung einer graphischen Benutzeroberfläche in Java sehr einfach ist. Die Integration neuer Klassifikatoren und Selektoren wird durch das objektorientierte Java sehr erleichtert. Datentypen und Komponenten können dadurch automatisch analysiert und in funktionelle Gruppen eingeteilt werden, ohne die Komponentensoftware hierfür zu konfigurieren. Ein Nachteil der Programmiersprache ist allerdings ihr ressourcenintensives Verhalten. Bei einer kontinuierlichen Visualisierung der automatischen Synthese eines Klassifikationssystems wird der Vorgang merklich verlangsamt. Allerdings lag der Schwerpunkt dieser Arbeit nicht in der Optimierung der Implementierung. Solche Verbesserungen bleiben nachfolgenden Arbeiten überlassen.

ICE ist deshalb so erfolgreich, weil es über eine effiziente graphische Benutzeroberfläche bedient wird und sehr leicht neue Algorithmen integriert werden können. Zur Zeit stehen weit über 300 Komponenten zur Verfügung. Von den in Kapitel 4 entwickelten Synthesemethoden wurden bislang 13 als Komponenten implementiert. Für die Klassifikation stehen mittlerweile beinahe 100 Merkmalsselektoren und 55 Klassifikatoren bereit, die auf Weka basieren. Für die Bildverarbeitung und Merkmalsextraktion wurde ebenfalls eine große Anzahl von Algorithmen eingebunden. Aufgrund der universellen Fähigkeiten der Komponentensoftware wird diese auch in zahlreichen anderen Projekten erfolgreich eingesetzt. Beispielhafte Anwendung fand die Komponentensoftware bislang in der Entwicklung von Methoden für die Brustkrebsfrüherkennung [Müller, 2001a] [Müller, 2001b] [Müller, 2004], der Segmentierung und Detektion von Lymphozyten [Beller, 2003] [Beller, 2004], von Lunkern in Aluminiumspritzguß-Werkstücken sowie als Rahmenwerk für ein allgemeines Segmentierungswerkzeug [Beller, 2005a] [Beller, 2005b]. Die Registrierung von Kernspin- mit Röntgenmammogrammen

[Ruiter, 2001] erfordert keine Klassifikation. ICE wird in diesem Falle als Entwicklungswerkzeug für die rasche Montage von Prozeßketten verwendet. Ebenso wurde ICE als Entwicklungsplattform im Bereich der Mikrosystemtechnik sowohl für die Steuerung elektronischer Nasen [Goschnick, 2003] als auch für den Entwurf und die Realisierung eines TCP/IP-Servers für den Datentransfer eines Wetterexperimentes [da Silva, 2001] eingesetzt.

ICE wird in Zukunft weiter gepflegt und in vielen Projekten verwendet werden. Wie die Klassifikationsbibliothek Weka soll die Komponentensoftware ICE in Zukunft als OpenSource-Software zur Verfügung stehen. Auf diese Art kann das Entwicklungswerkzeug einer großen Anwenderschaft zur Verfügung gestellt werden, die sich an der weiterführenden Entwicklung beteiligen können. Das Ziel sollte sein, einen offenen Standard für Komponentensoftware zu schaffen, deren Anwendungsgebiet nicht auf die Klassifikation beschränkt ist.

7.3 Schlußfolgerung

Die in dieser Arbeit entwickelte Komponentensoftware für die Klassifikation ermöglicht einen schnellen und interaktiven Entwurf von Klassifikationssystemen und vereinfacht das zur Zeit übliche manuelle Vorgehen erheblich. Bislang war für den Entwurf von Klassifikationssystemen ein Expertenwissen unumgänglich. In dieser Komponentensoftware wurden zudem neuartige Methoden integriert, die erstmals eine Automatisierung des Entwurfs von Klassifikationssystemen ermöglichen. Der Entwurfsprozess wird durch diese Methoden enorm erleichtert. Automatisch erstellte Klassifikationssysteme erreichten auf bekannten Problemen Generalisierungsfehler mit in den meisten Fällen besserer, aber keinesfalls schlechterer Qualität. Ein Expertenwissen ist für den Entwurf nicht mehr notwendig. Bei vorhandenem Expertenwissen können die erstellten Klassifikationssysteme nach eigenen Vorstellungen interaktiv verändert werden. Der automatische Aufbau eines Klassifikationssystems benötigt je nach Komplexität der jeweiligen Klassifikationsaufgabe zwischen wenigen Minuten und einigen Stunden, im Gegensatz zu Tagen oder Wochen eines manuellen Vorgehens. Aus diesen Gründen stellt die entwickelte Komponentensoftware für die Klassifikation eine wichtige Neuerung für den Entwurf von Klassifikationssystemen dar.

Lebenslauf

24. Juli 1968	Geboren in Dortmund
1975 bis 1979	Grundschule Emmendingen
1979 bis 1988	Abitur am Goethe Gymnasium Emmendingen Hochschulreife
1988 bis September 1989	Grundwehrdienst
Oktober 1989 bis Februar 1998	Studium an der Universität Karlsruhe Freiberuflich tätig als Berater und Systementwickler Abschluß als Diplom-Informatiker
seit Juni 1998	Wissenschaftlicher Mitarbeiter, Promotion am Forschungszentrum Karlsruhe
Juni 1998 bis Mai 2002	Computergestützte Diagnose
Juni 2002 bis April 2004	3D Ultraschall Computertomographie
Mai 2004 bis Juli 2006	Grid Computing

Publikationsliste

[1] Ruiter, N.V., Zapf, M., Stotzka, R., Müller, T.O., Schlote-Holubek, K., Göbel, G., and Gemmeke, H.. *First Images with a 3D-Prototype for Ultrasound Computer Tomography*. IEEE International Ultrasonics Symposium, 2005

[2] Ruiter, N.V., Stotzka, R., Müller, T.O., Liu, R., and Gemmeke, H.. *State of the Art and Challenges in Ultrasound Computer Tomography*. EMBEC - European Medical & Biological Engineering Conference, 2005

[3] Schlote-Holubek, K., Stotzka, R., Ruiter, N.V., Müller, T.O., Göbel, G., and Gemmeke, H.. *Development of Ultrasound Sensor Arrays for 3D Ultrasound Computer Tomography*. COMS - Commercialization of Micro and Nano Systems, 2005

[4] Müller, T.O., Hartmann, V., and Gemmeke, H.. *Wiki - Gruppenkommunikation der Zukunft*. Bericht der Frühjahrstagung der SEI, Forschungszentrum Rossendorf, Nummer 603, 2005

[5] Beller, M., Stotzka, R., Müller, T.O., and Gemmeke, H.. *An example-based system to support the segmentation of stellate lesions*. BVM - Bildverarbeitung für die Medizin, 2005

[6] Müller, T.O., Ruiter, N.V., Stotzka, R., Beller, M., Eppler, W., and Gemmeke, H.. *Ultrasound Computertomography, Distributed Volume Reconstruction and GRID-Computing*. BVM - Bildverarbeitung für die Medizin, 2005

[7] Ruiter, N.V., Müller, T.O., Stotzka, R., and Gemmeke, H.. *Evaluation of Different Approaches of Transmission Tomography in Ultrasound Computer Tomography*. BVM - Bildverarbeitung für die Medizin, 2005

[8] Stotzka, R., Ruiter, N.V., Müller, T.O., Rong, L., Schlote-Holubek, K., Göbel, G., and Gemmeke, H.. *Entfaltung von Ultraschallsignalen für verbesserte Bildqualität in der Ultraschall Computertomographie*. BVM - Bildverarbeitung für die Medizin, 2005

[9] Beller, M., Gemmeke, H., Müller, T.O., and Stotzka, R.. *Semi-Supervised Segmentation of Images*. MIVP - Iranian Conference on Machine Vision and Image Processing, 2005

[10] Stotzka, R., Ruiter, N.V., Müller, T.O., Rong, L., and Gemmeke, H.. *High resolution image reconstruction in ultrasound computer tomography using RF signal deconvolution*. SPIE Medical Imaging, 2005

[11] Ruiter, N.V., Stotzka, R., Müller, T.O., Gemmeke, H., Reichenbach, J.R., and Kaiser, W.A.. *Model-based Registration of X-ray Mammograms and MR Images of the Female Breast*. MIC - IEEE Medical Imaging Conference, 2004

[12] Müller, T.O., Stotzka, R., Ruiter, N.V., Schlote-Holubek, K., and Gemmeke, H.. *3D Ultrasound-Computertomography: Data Acquisition Hardware*. MIC - IEEE Medical Imaging Conference, 2004

[13] Stotzka, R., Ruiter, N.V., Müller, T.O., and Hartlieb, J.. *3D images with 10x higher res*, European Hospital, Nummer 4, 2004

[14] Eppler, W., Müller, T.O., Hartmann, V., Tchilingarian, S., Augenstein, A., Naji, M., and Gemmeke, H.. *Grid-Computing in Echtzeit*. Forschungszentrum Karlsruhe Nachrichten, Forschungszentrum Karlsruhe GmbH, Nummer 36, 2004

[15] Beller, M., Stotzka, R., and Müller, T.O.. *Application of an Interactive Feature-Driven Segmentation*. BMT – Biomedizinische Technik, 2004

[16] Stotzka, R., Müller, T.O., Ruiter, N.V., Schlote-Holubek, R., Liu, R., Göbel, G., and Gemmeke, H.. *A new 3D Ultrasound Computer Tomography Demonstrator System*. BMT – Biomedizinische Technik, 2004

[17] Stotzka, R., Müller, T.O., Gemmeke, H., and Hartlieb, J.M.A.. *Krebsvorsorge mit Tiefblick. 3D Ultraschall Tomographie aus Karlsruhe*, Health Technologies, Nummer 2, 2004

[18] Ruiter, N.V., Müller, T.O., Stotzka, R., Gemmeke, H., Reichenbach, J.R., and Kaiser, W.A.. *Registration of X-ray Mammograms and MR-Volumes of the Female Breast based on Simulated Mammographic Deformation*. IWDM - International Workshop on Digital Mammography, 2004

[19] Pfleiderer, S.O.R., Stotzka, R., Müller, T.O., Gemmeke, H., and Kaiser, W.A.. *Ultrasound computed tomography: A new method supplying standardized, reproducible cross section images*. ECR - European Congress of Radiology, 2004

[20] Ruiter, N.V., Müller, T.O., Stotzka, R., Gemmeke, H., Reichenbach, J.R., and Kaiser, W.A.. *Erste Evaluierung eines modellbasierten Registrierungsalgorithmus für Röntgen-mammogramme und MR-Volumina der Brust*. BVM - Bildverarbeitung für die Medizin, 2004

[21] Stotzka, R., Müller, T.O., Schlote-Holubek, K., and Göbel, G.. *Ultraschallwandler-Array-Systeme für die 3D Ultraschall Computertomographie*. BVM - Bildverarbeitung für die Medizin, 2004

[22] Müller, T.O., Stotzka, R., Beller, M., Ruiter, N.V., and Hartmann, V.. *Schneller Aufbau medizinischer Diagnosesysteme mit der Komponentensoftware ICE*. BVM - Bildverarbeitung für die Medizin, 2004

[23] Stotzka, R., Widmann, H., Müller, T.O., and Schlote-Holubek, K.. *Prototype of a new 3D ultrasound computer tomography system: transducer design and data recording*. SPIE Medical Imaging, 2004

[24] Stotzka, R., Müller, T.O., Schlote-Holubek, K., Göbel, G., and Gemmeke, H.. *Neue Ultraschallsensoren in der Computertomographie*. , Forschungszentrum Karlsruhe GmbH, Nummer , 2004

[25] Müller, T.O., Deck, T.M., Stotzka, R., Gemmeke, H., Höpfel, D., and Li, M.. *Ultrasound computertomography: image reconstruction using local absorption and sound speed profiles*. ESEM - European Society for Engineering and Medicine, 2003

[26] Stotzka, R., Müller, T.O., Schlote-Holubek, K., Deck, T.M., Vaziri Elahi, S., Göbel, G., and Gemmeke, H.. *Aufbau eines Ultraschall-Computertomographen für die Brustkrebsdiagnostik*. BVM - Bildverarbeitung für die Medizin, 2003

[27] Deck, T.M., Müller, T.O., Stotzka, R., and Gemmeke, H.. *Rekonstruktion von Geschwindigkeits- und Absorptionsbildern eines Ultraschall-Computertomographen*. BVM - Bildverarbeitung für die Medizin, 2003

[28] Ruiter, N.V., Müller, T.O., Stotzka, R., Gemmeke, H., Reichenbach, J.R., and Kaiser, W.A.. *Finite Element Simulation of the Breast's Deformation during Mammography to Generate a Deformation Model for Registration*. BVM - Bildverarbeitung für die Medizin, 2003

[29] Ruiter, N.V., Stotzka, R., Müller, T.O., Reichenbach, J.R., Wurdinger, S., Gemmeke, H., and Kaiser, W.A.. *Model Based Fusion of X-ray Mammograms and MR Volumes of the Female Breast*. MRM - Congress on MR-Mammography, 2003

[30] Stotzka, R., Müller, T.O., Schlote-Holubek, K., and Gemmeke, H.. *Ultrasound computertomography for breast cancer diagnosis*. ESEM - European Society for Engineering and Medicine, 2003

[31] Ruiter, N.V., Müller, T.O., Stotzka, R., Gemmeke, H., Reichenbach, J.R., and Kaiser, W.A.. *Automatic Image Matching for Breast Cancer Diagnostics by a 3D Deformation Model of the Mamma*. BMT - Biomedizinische Technik, 2002

[32] Stotzka, R., Würfel, J., Müller, T.O., and Gemmeke, H.. *Medical Imaging by Ultrasound-Computertomography.* SPIE Medical Imaging, 2002

[33] Müller, T.O., and Stotzka, R.. *ICE: Komponentensoftware für die computergestützte Diagnose*. BMT - Biomedizinische Technik, 2001

[34] Stotzka, R., Würfel, J., Müller, T.O., and Gemmeke, H.. *Medizintechnik in HPE: Ultraschall-Computertomographie*. Frühjahrstagung der SEI, Forschungszentrum Rossendorf, 2001

[35] Müller, T.O., Stotzka, R., Höpfel, D., and Yang, H.. *Texturanalyse zur Detektion gruppierter Mikroverkalkungen bei der Brustkrebsfrüherkennung*. BVM - Bildverarbeitung für die Medizin, 2001

[36] Ruiter, N.V., Müller, T.O., and Stotzka, R.. *Elastisches Matching von Röntgenmammogrammen und dreidimensionalen Magnetresonanzdaten*. BVM - Bildverarbeitung für die Medizin, 2001

[37] Ruiter, N.V., Müller, T.O., Stotzka, R., and Gemmeke, H.. *Elastic registration of X-Ray mammograms and three-dimensional MRI data*. SCAR - Symposium for Computer Applications in Radiology, 2001

[38] Müller, T.O.. *3D Diagnostik in der Mammographie*. Bericht der Herbsttagung der SEI, Forschungszentrum Rossendorf, Nummer 305, 2000

[39] Müller, T.O., Stotzka, R., and McCoy, D.. *3D Reconstruction of Clustered Microcalcifications: Software Components*. IWDM - International Workshop on Digital Mammography, 2000

[40] Müller, T.O., Ruiter, N.V., Stotzka, R., and Kaiser, W.A.. *Automatic Matching of MR Volume Data and X-ray Mammograms*. ECR - European Congress of Radiology, 2000

[41] Müller, T.O., Stotzka, R., Neiber, H., Eppler, W., and Gemmeke, H.. *Bildgebende Verfahren in der Diagnostik des Mammakarzinoms*. Forschungszentrum Karlsruhe Nachrichten, Forschungszentrum Karlsruhe GmbH, Nummer 32, 2000

[42] Neiber, H., Müller, T.O., and Stotzka, R.. *Local Contrast Enhancement for the Detection of Microcalcifications*. IWDM - International Workshop on Digital Mammography, 2000

[43] Stotzka, R., Haase, J., and Müller, T.O.. *3D Reconstruction of clustered microcalcifications from two mammograms: information preservation*. SPIE Medical Imaging, 1999

[44] Müller, T.O., Stotzka, R., Eppler, W., and Gemmeke, H.. *Three-dimensional Reconstruction of Clustered Microcalcifications*. CAR - Computer Assisted Radiology and Surgery, 1998

[45] Stotzka, R., Müller, T.O., Eppler, W., and Gemmeke, H.. *A Low Cost Computer Assisted Mammography Workstation*. IWDM - International Workshop on Digital Mammography, 1998

[46] Müller, T.O., Stotzka, R., Hochmuth, A., Eppler, W., and Gemmeke, H.. *Volume Reconstruction of Clustered Microcalcifications in Mammograms*. IWDM - International Workshop on Digital Mammography, 1998

[47] Stotzka, R., Müller, T.O., Eppler, W., and Gemmeke, H.. *Three-dimensional Reconstruction of Clustered Microcalcifications from two Digitized Mammograms*. SPIE Medical Imaging, 1998

[48] Müller, T.O.. *Volumen-Rekonstruktion gruppierter Mikroverkalkungen*. Diplomarbeit, Universität Karlsruhe, 1998

Literaturverzeichnis

[AdOculos, 2003] Ad Oculos Homepage. , http://www.theimagingsource.com/prod/soft/adoculos/adoculos_textbook_deu.htm, 2003

[Alef, 2004] . *Aufbau des großen GridKA-Clusters am Forschungszentrum Karlsruhe*. Forschungszentrum Karlsruhe Nachrichten, Forschungszentrum Karlsruhe GmbH, Nummer 36, 2004

[Althoff, 1992] K.-D. Althoff, S. Weß, B. Bartsch-Spörl et al. *Case-Based Reasoning in Expert Systems: Which Role Do Cases Play for Knowledge-Based Systems?*. Fachzeitschrift Künstliche Intelligenz, Nummer 4, 14-21, 1992

[Argiro, 2000] D. Argiro, S. Kubica, M. Young and S. Jorgensen. *Khoros: An Integrated Development Environment for Scientific Computating and Visualization*. Enabling Technologies for Computational Science: Frameworks, Middleware and Environments, pp. 147-157, Kluwer Academic Publishers, 2000

[Argiro, 2001] D. Argiro, K. Farrar and S. Kubica. *Cantata: The Visual Programming Environment for the Khorus System*. Visualization, Imaging and Image Processing Conference Proceedings, 2001

[Arya, 1994] . *An optimal algorithm for approximate nearest neighbor searching*. , Nummer , 1994

[Beasley, 1977] J. D. Beasley and S. G. Springer. *Algorithm AS 111: The Percentage Points of the Normal Distribution*. Applied Statistics, pp. 118-121, unknown Publisher, 1977

[Beller, 2003] . *Merkmalsgesteuerte Segmentierung in der medizinischen Mustererkennung*. BVM - Bildverarbeitung für die Medizin, 2003

[Beller, 2004] Beller, M., Stotzka, R., and Müller, T.O.. *Application of an Interactive Feature-Driven Segmentation*. BMT - Biomedizinische Technik, 2004

[Beller, 2005a] Beller, M., Stotzka, R., Müller, T.O., Gemmeke, H., and Gemmeke, H.. *Semi-Supervised Segmentation of Images*. MIVP - Iranian Conference on Machine Vision and Image Processing, 2005

[Beller, 2005b] Beller, M., Stotzka, R., Müller, T.O., and Gemmeke, H.. *An example-based system to support the segmentation of stellate lesions*. BVM - Bildverarbeitung für die Medizin, 2005

[Berg, 1997] K. Berg. *Component-Based development: No Silver Bullet*. Object Magazine, Nummer 3, 54-57, 1997

[Berting, 2000] A. Berting. *Automatische Lymphozytenidentifikation*. Diplomarbeit, Forschungszentrum Karlsruhe - Hauptabteilung Prozessdatenverarbeitung und Elektronik, 2000

[Blackbox, 1997] BlackBox Developer and BlackBox Component Framework. Oberon Microsystems, Inc., http://www.oberon.ch/, 1997

[Brockhaus, 2004] . *Der Brookhaus multimedial DVD*. F.A. Brockhaus, 2004, ISBN 3-411-06515-X

[Cendrowska, 1987] J. Cendrowska. *PRISM: An algorithm for inducing modular rules*. International Journal of Man-Machine Studies, Nummer 27, 1987

[Cerami, 2002] . *Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. O'Reilly, 2002, ISBN 0-596-00224-6

[Comaniciu, 2000] D. Comaniciu, P. Meer and D. Foran. *Image Guided Decision Support System for Pathology*. Machine Vision and Applications, Nummer 4, , 2000

[da Silva, 2001] . *Konzipierung eines TCP/IP-Servers für die Funkdaten aus einem Mipas-Ballon*. Diplomarbeit, Forschungszentrum Karlsruhe - Hauptabteilung Prozessdatenverarbeitung und Elektronik, 2001

[Dehning, 1995] O. Dehning. *Wissensbasierte Inspektion industrieller Objekte*. Institut für Nachrichtentechnik und Informationsverarbeitung, Universität Hannover, 1995

[Duden - Das Fremdwörterbuch, 2001] . *Duden - Das Fremdwörterbuch*. Bibliographisches Institut, 1982, ISBN 3-411-20905-4

[Fisher, 1936] R. A. Fisher. *The use of multiple measurements in taxonomic problems*. Annals of Eugenics, Nummer 7, 1936

[Flanagan, 1998] D. Flanagan. *Java in a Nutshell*. O'Reilly, 1998, ISBN 3-89721-100-9

[Fukunaga, 1990] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990, ISBN 0-12-269-851-7

[Giger, 2000] M. L. Giger. *Computer-aided diagnosis of breast lesions in medical images*. Computing in Medicine, Nummer 5, 39-45, 2000

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989, ISBN 0201157675

[Goschnick, 2003] . *ELMINA. Elektronische Mikronasen für Überwachungs- und Regelaufgaben in Gebäuden und Produktionsanlagen. Abschlußbericht des HGF-S.* , Forschungszentrum Karlsruhe GmbH, Nummer FZKA-6893, 2003

[GRID, 2004] GridKa. Forschungszentrum Karlsruhe GmbH, http://savannah.fzk.de/, 2004

[Großwendt, 2000] V. Großwendt. *Ausgezeichnet XML - Eine Einführung*. Java Magazin, Nummer 1, 79-84, 2000

[Heppner, 1997] S. Heppner und R. Oberle. *MoMo - eine verteilte, objektorientierte Realzeitarchitektur.* FZK Nachrichten, Forschungszentrum Karlsruhe GmbH, Nummer 6029, 1997

[Horstmann, 2000] C. S. Horstmann and G. Cornell. *Core Java 2*. SUN Microsystems Press, 2000, ISBN 0-13-081934-4

[Hotelling, 1933] H. Hotelling. *Analysis of a complex of statistical variables into principal components*. Journal of Educational Psychology, Nummer 24, 1933

[Jähne, 1997] B. Jähne. *Digitale Bildverarbeitung*. Springer Verlag, 1997, ISBN 3-540-61379-X

[Jaimes, 2001] . *Automatic Selection of Visual Features and Classifiers*. Storage and Retrival for Image and Video Databases VIII, 2001

[Jain, 2000] A. K. Jain, R. P. W. Duin and J. Mao. *Statistical Pattern Recognition: A Review.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Nummer 1, 4-36, 2000

[König, 1999] A. König, M. Eberhardt and R. Wenzel. *QuickCog Self-Learning Recognition System - Exploiting machine learning techniques*. Image Processing Europe, Nummer 9, 10-19, 1999

[König, 2003] QuickCog - Cognitive Systems Design Environment. , http://www.quickcog.de/, 2003

[Korf, 2000] R. Korf. *Automatische Erkennung von Mammatumoren durch eine Kombination neuronaler Netze und Case-Based-Reasoning*. Diplomarbeit, Forschungszentrum Karlsruhe - Hauptabteilung Prozessdatenverarbeitung und Elektronik, 2000

[LabView, 2004] LabView - The Software That Powers Virtual Instrumentation. National Instruments, http://www.ni.com/labview/, 2004

[Lehmann, 1997] T. Lehmann, W. Oberschelp, E. Pelikan und R. Repges. *Klassifikation und Mustererkennung*. Bildverarbeitung für die Medizin, pp. 395-429, Springer Verlag, 1997

[Manber, 1989] U. Manber. *Introduction To Algorithms - A Creative Approach*. Addison-Wesley, 1989, ISBN 0-201-12037-2

[Manly, 1994] B. F. J. Manly. *Multivariate Statistical Methods*. Chapman & Hall, 1994, ISBN 0-412-60300-4

[Maurer, 2000] P. M. Maurer. *Components: What if they gave a revolution and nobody came?*. Computer, Nummer 6, 28-34, 2000

[McCulloch, 1943] W. McCulloch and W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity.* Bulletin of Mathematical Biophysics, Nummer 1, 1943

[Micheli-Tzanakou, 2000] E. Micheli-Tzanakou. *Supervised and Unsupervised Pattern Recognition*. CRC Press, 2000, ISBN 0-8493-2278-2

[Michie, 1994] D. Michie, D. J. Spiegelhalter und C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994, ISBN 0-13-106360-X

[MontiVision, 2003] MontiVision Development Kit. , http://www.montivision.com, 2003

[Müller, 2000] Müller, T.O., Stotzka, R., and McCoy, D.. *3D Reconstruction of Clustered Microcalcifications: Software Components*. IWDM - International Workshop on Digital Mammography, 2000

[Müller, 2001a] T. O. Müller, R. Stotzka, D. Höpfel und H. Yang. *Texturanalyse zur Detektion gruppierter Mikroverkalkungen bei der Brustkrebsfrüherkennung*. Bildverarbeitung für die Medizin BVM, 2001

[Müller, 2001b] T. O. Müller und R. Stotzka. *ICE: Komponentensoftware für die computergestützte Diagnose*. Biomedizinische Technik BMT, 2001

[Müller, 2004] T.O. Müller, R. Stotzka, M. Beller, N.V. Ruiter und V. Hartmann. *Schneller Aufbau medizinischer Diagnosesysteme mit ICE*. Bildverarbeitung für die Medizin BVM, 2004

[Myers, 2000] T. Myers und A. Nakhimovsky. *Java XML.* MITP Verlag, 2000, ISBN 3-8266-0661-2

[Neiber, 2001] H. Neiber. *Strategieplanung zum automatischen Aufbau von Klassifikationssystemen*. Diplomarbeit, Forschungszentrum Karlsruhe - Hauptabteilung Prozessdatenverarbeitung und Elektronik, 2001

[Niemann, 1993] Niemann, H.. *Klassifikation von Mustern*. Springer Verlag, 1993, ISBN

[Nilsson, 1990] N. J. Nilsson. *The Mathematical Foundations of Learning Machines*. Morgan Kaufmann Publishers, 1990, ISBN 1-55860-123-6

[Noltemeier, 1976] H. Noltemeier. *Computergestützte Planungssysteme*. Physica-Verlag, 1976, ISBN 3-7908-0170-4

[Oberon, 1994] Oberon/F Users Guide. Oberon Microsystems, Inc., http://www.oberon.ch/, 1994

[OpenDoc, 1994] IBM Corporation, The System Object Model and the Component Object Model: a comparison of technologies from a developer's perspective, 1994

[Pearson, 1901] K. Pearson. *On lines and planes of closest fit to a system of points in space*. Philosophical Magazine, Nummer 2, 1901

[Portos, 1997] Portos Realtime Operating System and Denia Development Environment for Portos. Oberon Microsystems, Inc., http://www.oberon.ch/, 1997

[R-Project, 2004] The R Projekt for Statistical Computing. R-Projekt, http://www.r-project.org/, 2004

[Rechenberg, 1997] P. Rechenberg und G. Pomberger. *Informatik-Handbuch*. Hanser Verlag, 1997, ISBN 3-446-18691-3

[Ritthoff, 2001] . *YALE: Yet Another Learning Environment*. , Universität Dortmund, Nummer 763, 2001

[Roman, 1999] E. Roman. *Mastering Enterprise JavaBeans*. John Wiley & Sons, 1999, ISBN 0-471-33229-1

[Rosenblatt, 1962] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962, ISBN

[Ruiter, 2001] Ruiter, N.V., Müller, T.O., and Stotzka, R.. *Elastisches Matching von Röntgenmammogrammen und dreidimensionalen Magnetresonanzdaten*. BVM - Bildverarbeitung für die Medizin, 2001

[S-System, 2004] The S System. S, http://stat.bell-labs.com/S/, 2004

[SAS, 2004] SAS: Statistical Analysis System. SAS, http://www.sas.com/offices/europe/germany/index.html, 2004

[Schalkoff, 1995] Schalkoff, R.J.. *Pattern Recognition*. John Wiley & Sons, 1995, ISBN

[Schürmann, 1977] J. Schürmann. *Polynomklassifikatoren für die Zeichenerkennung*. R. Oldenburg Verlag, 1977, ISBN 3-486-21371-7

[Schwichtenberg, 2001] H. Schwichtenberg. *Runderneuerung*. iX - Magazin für professionelle Informationstechnik, Nummer 9, 102-108, 2001

[Statistica, 2004] Statistica - Data Mining, Data Analysis, Quality Control and Web Analytics Software. StatSoft, http://www.statsoft.com/, 2004

[Staudacher, 2001] B. Staudacher und R. Pichler. *CORBA 3.0 - Komponentenmodell: Anwendung und Besonderheiten*. OBJEKTspektrum, Nummer 1, 32-39, 2001

[Stotzka, 1995] R. Stotzka. *Automatisches Tumorklassifikationssystem für mikroskopische Gewebeschnitte*. Shaker Verlag, 1995

[Stotzka, 2000] R. Stotzka. *Hybrid Neural Network and Statistical Classification Algorithms in Computer Assisted Diagnosis*. SPIE Medical Imaging 2000, 2000

[Szyperski, 1999] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. ACM Press Books, 1999, ISBN 0-201-17888-5

[The Global Grid Forum, 2005] The Global Grid Forum. , http://www.ggf.org/, 2005

[The Globus Alliance, 2005] The Globus Alliance. , http://www.globus.org/, 2005

[Twomey, 1998] J. M. Twomey. *Bias and Variance of Validation Methods for Function Approximation Neural Networks Under Conditions of Sparse Data*. IEEE Transactions on Systems, Man, and Cybernetics, Nummer 3, 404-416, 1998

[Ullenboom, 2001] C. Ullenboom. *Java ist auch eine Insel*. Galileo Computing, 2001, ISBN 3-89842-174-0

[von Fournier, 1992] D. von Fournier, H.-W. Anton, H. Junkermann und G. Bastert. *Breast cancer screening*. Cancer Diagnosis, pp. 78-87, Springer Verlag, 1992

[Weiss, 1991] S. M. Weiss und C. A. Kulikowski. *Computer Systems that Learn*. Morgan Kaufmann Publishers, 1991, ISBN 1-55860-065-5

[Wenz, 2003] . *Echtzeitfähige Komponentensoftware für die Entwicklung rekonfigurierbarer mechatronischer Systeme*. Intelligente mechatronische Systeme, 2003

[Witten, 2000] I. H. Witten und E. Frank. *Data Mining*. Morgan Kaufmann Publishers, 2000, ISBN 1-55860-552-5

[YALE, 2003] YALE - Yet Another Learning Environment. Universität Dortmund, http://yale.cs.uni-dortmund.de/, 2003

[Yang, 2000] H. Yang. *Design and Implementation of a Microcalcification Cluster Classifier.* Diplomarbeit, Forschungszentrum Karlsruhe - Hauptabteilung Prozessdatenverarbeitung und Elektronik, 2000

[Zell, 1991] . *The SNNS Neural Network Simulator.* Informatik-Fachberichte, 1991

[Ziegenmeyer, 2003] . *Optimierung und Anpassung der Support-Vector-Klassifikation motiviert durch reale Diagnoseanwendungen*. Diplomarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, 2003

Anhang A: Begriffsdefinitionen

Begriff	*Bedeutung*	*Bezeichnung*
Merkmal	Ein gemessener oder berechneter Wert.	x oder y
Merkmalsvektor	Ein Vektor aus Merkmalen. Das m-te Merkmal bezeichnet in vielen Fällen die Klassenzugehörigkeit.	X oder Y m Elemente
Beobachtung	Ein Merkmalsvektor, welcher einen bestimmten Zustand repräsentiert. Die Klassenzugehörigkeit ist nicht bestimmt.	
Muster	Eine Beobachtung mit bekannter Klassenzugehörigkeit.	
Stichprobe	Eine Menge von Mustern.	O Anzahl o
Klasse	Eine Menge von Beobachtung mit gleichen Eigenschaften.	Anzahl c
Klassifikation	Eindeutige Zuordnung einer Beobachtung zu einer Klasse.	
Klassifikator	Ein Algorithmus, der eine Klassifikation durchführt.	K Anzahl k
Selektor	Ein Algorithmus, der aus einem Merkmalsvektor eine Untermenge für eine bessere Klassifikation auswählt.	S Anzahl s
Klassifikationssystem	Kombination von Klassifikatoren und Selektoren.	
Synthesemethode	Ein Algorithmus, der automatisch Klassifikationssysteme synthetisiert.	
Testdaten	Eine Untermenge von Mustern der Stichprobe, die für den Test des Klassifikators verwendet werden.	
Trainingsdaten	Eine Untermenge von Mustern der Stichprobe, welche für das Training des Klassifikators verwendet werden.	
Klassifikationsfehler	Die Anzahl falsch klassifizierter Muster normiert auf die Anzahl aller Muster.	
Trainingsfehler	Der Klassifikationsfehler auf den Trainingsdaten.	e_t
Testfehler	Der Klassifikationsfehler auf den Testdaten.	e_g
Generalisierungsfehler	Der Klassifikationsfehler auf einer unendlich großen Stichprobe. Er wird mit Hilfe des Testfehlers geschätzt.	
Güte eines Klassifikationssystems	Der Generalisierungsfehler eines Klassifikationssystems für ein gegebenes Klassifikationsproblem.	
Güte einer Synthesemethode	Die Verbindung der Güte eines synthetisierten Klassifikationssystems mit der zur Synthese benötigten Zeit.	

Anhang B: Klassifikationsprobleme

B.1 Mammographie-Klassifikationsprobleme

a) Klassifikationsproblem erstellt durch Coocurrence-Matrizen

Eigenschaften: ca. 38% Fehler, 4096 Merkmale, 2 Klassen und 256 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	723 – 80134	36,0 – 45,6
Simple Single Random	10-fold	9 – 179	40,5 – 58,0
Single Random	10-fold	42 – 2664	36,5 – 52,0
Zero Selector	10-fold	3576 – 5182	36,1 – 41,7
One Selector	10-fold	77 – 88	39,7 – 40,5
Promising List	10-fold	648687	39,7
Blasensynthese	10-fold	60556	35,3

Zwei der verwendeten Klassifikationsprobleme für die Mammadiagnostik wurden mit Hilfe der Mammogramme klinischer Partner selbst erstellt. Im wesentlichen wurden aus den Mammogrammen 256 Regions of Interest (ROIs) ausgeschnitten und diese in die Klassen „enthält Mikroverkalkungen“ und „enthält keine Mikroverkalkungen“ eingeteilt. Aus den Bildausschnitten wurden durch Coocurrence-Matrizen automatisch 4096 Merkmale extrahiert [Yang, 2000]. Das Verhältnis von Merkmalen zu Mustern ist in diesem Klassifikationsproblem extrem ungünstig.

b) Klassifikationsproblem erstellt durch hybride Neuronale Netze

Eigenschaften: ca. 21% Fehler, 85 Merkmale, 2 Klassen und 1004 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	6 – 463	18,9 – 43,2
Simple Single Random	Leave-one-out	107 – 68594	27,3 – 49,8
Single Random	10-fold	8 – 133	17,7 – 33,0
Zero Selector	10-fold	1531 – 1589	18,3 – 29,8
One Selector	10-fold	1231 – 2332	26,8 – 28,6
Promising List	10-fold	20944	6,5
Blasensynthese	10-fold	7428 – 8227	6,0 – 6,9

Das zweite Klassifikationsproblem wurde ebenfalls mit den Bildausschnitten erstellt. Zunächst wurden die Ausschnitte dreimal um je 90 Grad rotiert, wodurch sich die Anzahl der ROIs auf 1004 erhöht und eine Rotationsinvarianz erreicht wurde. Aus angelernten hybriden neuronalen Netzen wurden daraufhin automatisch 85 Merkmale extrahiert [Korf, 2000]. Das Verhältnis von Merkmalen zu Mustern und die Qualität der Merkmale ist im zweiten Klassifikationsproblem erheblich besser, weshalb mit diesem bei der Detektion von Mikroverkalkungen wesentlich bessere Ergebnisse erreicht wurden.

c) Klassifikationsproblem Breast Cancer „breast-cancer.arff"

Dieses Klassifikationsproblem entstammt der Weka-Bibliothek. Es beschreibt das Wiederauftreten von Brustkrebsfällen nach erfolgter Behandlung und wurde bereits mehrfach in der Literatur verwendet. Drei der vier bekannten Literaturstellen beinhalten einen Generalisierungsfehler zwischen 34 und 28 Prozent, der mit den Synthesemethoden auf 24 Prozent verbessert werden konnte. Der in der vierten Literaturstelle beschriebene Fehler von 22 Prozent konnte mit den verfügbaren Klassifikationsalgorithmen nicht erreicht werden.

Eigenschaften: ca. 22-34% Fehler, 9 Merkmale, 2 Klassen und 286 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	15 – 280	24,5 – 33,2
Single Random	Leave-one-out	16 – 286	24,5 – 34,3
Zero Selector	Leave-one-out	1107 – 1418	24,5
One Selector	Leave-one-out	868 – 900	26,6
Promising List	Leave-one-out	20325 – 22981	24,1
Blasensynthese BF	Leave-one-out	40 – 212	26,6 – 32,9

Origin: Matjaz Zwitter & Milan Soklic, University Medical Center, Ljubljana, Yugoslavia
Ming Tan and Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)

B.2 Allgemeine Klassifikationsprobleme

Die nachfolgenden Klassifikationsprobleme wurden für die Untersuchung der Synthesemethoden verwendet. Sie sind zusammen mit der Weka-Bibliothek gebündelt oder stammen aus Projekten des Forschungszentrums Karlsruhe. Es werden nur die notwendigsten Referenzen erwähnt. Detaillierte Informationen zu Autoren, Herkunft und bisherige Verwendung lassen sich dem jeweiligen Kommentar im Kopf der Dateien entnehmen [Witten, 2000]. Den Klassifikationsproblemen sind die detaillierten Ergebnisse der verschiedenen Synthesemethoden als Tabellen zugeordnet. Im Kopf jeder Tabelle befindet sich die Anzahl der Merkmale, Muster und Klassen, sowie der bislang beste in der Literatur erwähnte Generalisierungsfehler.

a) Lymphozytenklassifikation

Eigenschaften: ca. 2,5% Fehler, 45 Merkmale, 2 Klassen und 189 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	8 – 360	2,6 – 7,9
Single Random	Leave-one-out	10 – 170	2,1 – 7,9
Zero Selector	Leave-one-out	556 – 1055	3,2
Blasensynthese BF	Leave-one-out	480 – 512	4,5
Promising List	10-fold	605 – 622	1,6 – 2,1
Blasensynthese BF	10-fold	15 – 19	2,5 – 3,5
Blasensynthese	10-fold	370 – 495	1,6

Das Problem der Lymphozytenklassifikation entstammt einem Projekt des Forschungszentrums Karlsruhe mit dem Schwerpunkt der Bildverarbeitung. Angefärbte Blutzellen werden als „Lymphozyt“ oder „Nicht-Lymphozyt“ eingestuft. Die Aufgabe wurde bereits im Rahmen einer Diplomarbeit näherungsweise gelöst [Berting, 2000]. Aufgrund der guten Vergleichbarkeit wurde die Problemstellung bei der Entwicklung von Synthesemethoden wieder aufgegriffen [Neiber, 2001]. Das Klassifikationsproblem zeichnet sich durch einen relativ geringen Fehler aus. Bemerkenswert ist vor allem, daß die Synthesemethoden innerhalb weniger Minuten Klassifikationssysteme synthetisierten, die besser waren als diejenigen, welche im Rahmen der Diplomarbeit erstellt wurden. Im Gegensatz zur automatischen Synthese mit etwa 7 Minuten wurden für den manuellen Entwurf des Klassifikationssystems schätzungsweise 14 Tage benötigt. Der Fehler konnte geringfügig von 2,5% auf 1,6% verbessert werden. Es wurden keine Informationen über die Beschaffenheit der Daten verwendet.

Wie in der Praxis häufig zu beobachten, besitzt auch dieses Klassifikationsproblem ein ungünstiges Verhältnis von Merkmalen zu Mustern. 189 Muster sind bei 45 Merkmalen zu wenig, um zuverlässige statistische Aussagen zu treffen. Der Zero Selector beweist, daß der Einsatz eines Selektors keine Verbesserung des Generalisierungsfehlers bewirkt. Die Blasensynthese ist bei diesem Klassifikationsproblem einer vollständigen Aufzählung aller möglichen Kombinationen nicht überlegen.

b) Iris-Klassifikationsproblem „iris.arff"

Origin: Creator: R.A. Fisher

Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

Eigenschaften: ca. 4% Fehler, 4 Merkmale, 3 Klassen und 150 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	4 – 22	3,3 – 66,6
Single Random	Leave-one-out	5 – 17	3,3 – 66,6
Zero Selector	Leave-one-out	55 – 96	3,3
One Selector	Leave-one-out	92 – 96	3,3
Full Random	Leave-one-out	1979 – 2163	3,3
Promising List	Leave-one-out	1987 – 2001	3,3
Blasensynthese BF	Leave-one-out	19 – 26	3,3 – 4,0
Blasensynthese	Leave-one-out	187 – 218	3,3 – 4,0

Das Iris-Klassifikationsproblem nimmt eine besondere Stellung ein. Er ist weit verbreitet und wird häufig als erstes Klassifikationsproblem bei der Untersuchung von Klassifikatoren eingesetzt. Die drei unterschiedlichen Klassen sind nicht linear separierbar. Aufgrund der relativ geringen Anzahl von 150 Mustern und vier Merkmalen sind Klassifikationsergebnisse intuitiv nachvollziehbar und die Synthesezeiten gering. Das Klassifikationsproblem ist deshalb sehr gut geeignet, um wiederholt Klassifikationssysteme zu synthetisieren und zu vergleichen.

Die Promising List benötigt etwa 2000 Sekunden für die Synthese eines Klassifikationssystems. Aufgrund der geringen Größe kann das Klassifikationsproblem auch von der Full Random Strategie mehrmals bei vollständiger Kreuzvalidierung durchlaufen werden. Das bestmögliche System wird auch von den Synthesemethoden Zero Selector und One Selector synthetisiert. Die Blasensynthese BF generiert häufig ein System mit einem der Promising List vergleichbaren Generalisierungsfehler, ist der Promising List in Bezug auf die Laufzeit aber deutlich überlegen. Der Zeitgewinn beträgt beinahe zwei Größenordnungen. Etwa drei Viertel aller durch die Blasensynthese erstellten Klassifikationssysteme verfügten über einen Generalisierungsfehler von 3,3 Prozent. Der Generalisierungsfehler der restlichen Klassifikationssysteme betrug 4,0 Prozent.

c) Klassifikationsproblem Lymphographie „lympf.arff"

Origin: Matjaz Zwitter & Milan Soklic, University Medical Center, Ljubljana, Yugoslavia

Igor Kononenko, University E.Kardelj and Bojan Cestnik, Jozef Stefan Institute

Eigenschaften: ca. 15% Fehler, 18 Merkmale, 4 Klassen und 148 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	2 – 8	16,6 – 45,3
Single Random	10-fold	2 – 8	15,7 – 35,8
Zero Selector	10-fold	23 – 26	15,5 – 17,5
One Selector	10-fold	12 – 13	13,5 – 16,3
Promising List	10-fold	302 – 306	12,7 – 14,1
Blasensynthese	10-fold	332 – 362	12,2 – 14,1

Das Klassifikationsproblem „Lymphographie“ stammt ebenso wie das Klassifikationsproblem „Breast Cancer“ aus dem Institute of Oncology, Jugoslawien. Bei der Untersuchung von Lymphknoten wird nach auffälligen Veränderungen gesucht. Das Klassifikationsproblem repräsentiert Muster mit einer Reihe von Merkmalen, welche durch Lymphographie gewonnen wurden. Der im Klassifikationsproblem angegebene Fehler von 15 Prozent konnte um ein bis drei Prozent gesenkt werden. Über den Zeitaufwand der manuell erstellten Lösungen sind keine Angaben verfügbar. Der Zeitaufwand der automatischen Synthese von etwa fünf Minuten dürfte jedoch erheblich geringer ausfallen. Aufgrund des Verhältnisses von 18 Merkmalen zu nur 148 Mustern liegt der Zeitaufwand der Blasensynthese innerhalb der gleichen Größenordnung wie derjenige der Promising List. Die Blasensynthese verspricht für derartige Klassifikationsprobleme keinen Gewinn.

d) Klassifikationsproblem Herzerkrankungen „heart-c.arff“

Eigenschaften: ca. 21% Fehler, 13 Merkmale, 5 Klassen und 302 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	2 – 9	21,8 – 45,5
Single Random	10-fold	3 – 127	16,1 – 45,6
Zero Selector	10-fold	60 – 418	16,5 – 18,1
One Selector	10-fold	20 – 22	14,9 – 19,2
Promising List	10-fold	510 – 516	14,8 – 15,2
Blasensynthese	10-fold	200 – 252	14,8 – 15,2

Origin: Origin: Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
V.A. Medical Center, Long Beach and Cleveland Clinic: Robert Detrano, M.D., Ph.D.

Dieses Klassifikationsproblem aus der Weka-Bibliothek besteht hauptsächlich aus Daten der Cleveland Clinic Foundation sowie drei weiterer Kliniken zur Diagnose von Herzerkrankungen. Von den ursprünglichen 75 Merkmalen werden im allgemeinen nur 13 verwendet. Die Muster verteilen sich auf fünf Klassen: Das Fehlen einer Herzerkrankung sowie vier zunehmend kritische Stadien einer Erkrankung. Selbst mit einer einfachen Synthesemethode wie Zero Selector kann der Klassifikationsfehler von bisher 21 Prozent verwendeter Systeme im Mittel auf 17 Prozent reduziert werden. Die Synthesemethoden Promising List und Blasensynthese reduzieren den Fehler in weniger als neun beziehungsweise vier Minuten auf etwa 15 Prozent.

e) Klassifikationsproblem Diabetes „diabetes.arff"

Origin: National Institute of Diabetes and Digestive and Kidney Diseases
Vincent Sigillito (vgs@aplcen.apl.jhu.edu), The Johns Hopkins University

Eigenschaften: ca. 24% Fehler, 8 Merkmale, 2 Klassen und 768 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	2 – 191	23,1 – 34,9
Zero Selector	10-fold	562 – 1630	22,3 – 25,9
One Selector	10-fold	62 – 73	23,4 – 25,8
Promising List	10-fold	168634	22,5
Blasensynthese	10-fold	394 – 490	18,8 – 23,7

Das Klassifikationsproblem repräsentiert ein Klassifikationsproblem zur Diagnose von Diabetes und ist der Weka-Bibliothek entnommen. Das Ziel ist, mit den Merkmalen des Klassifikationsproblems festzustellen, ob bei einem untersuchten Patienten Anzeichen einer Zuckererkrankung vorliegen oder nicht. Die relativ große Varianz des Klassifikationsfehlers bei der Blasensynthese läßt sich auf die zehnfache Kreuzvalidierung zurückführen. Im Mittel liegt der Fehler der Promising List erwartungsgemäß nicht über dem der Blasensynthese. Allerdings macht sich bei der hohen Anzahl von Mustern das Konzept der Blasensynthese sehr effizient bemerkbar. Sie ist im Mittel um einen Faktor von 400 schneller. Selbst wenn Laufzeitmessungen durch Multitasking-Betriebssysteme fehlerbehaftet sind, ist die Tendenz deutlich zu entnehmen. Ein derartig strukturiertes Klassifikationsproblem ist ein Paradebeispiel für die Blasensynthese. Ebenfalls auffällig ist der Geschwindigkeitsvorteil von One Selector gegenüber Zero Selector. Dies läßt sich unter anderem auf eine erhebliche Reduktion von acht auf drei Merkmale während der Merkmalsselektion zurückführen. Das Training eines Klassifikators und die zehnfache Kreuzvalidierung werden dadurch generell beschleunigt.

f) Klassifikationsproblem Kontaktlinsen „contact-lenses.arff"

Origin: Cendrowska, J. "PRISM: An algorithm for inducing modular rules", International Journal of Man-Machine Studies, 1987, 27, 349-370

Eigenschaften: ca. 17% Fehler, 4 Merkmale, 3 Klassen und 24 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	1 – 3	16,7 – 37,5
Single Random	Leave-one-out	1 – 5	16,7 – 37,5
Zero Selector	Leave-one-out	12 – 13	16,7
One Selector	Leave-one-out	11 – 20	29,1
Promising List	Leave-one-out	85 – 96	16,7
Blasensynthese BF	Leave-one-out	9 – 38	29,1
Blasensynthese	Leave-one-out	66 – 81	16,7

Dieses Klassifikationsproblem dient dazu, Patienten mit einem Sehfehler geeignete Kontaktlinsen zu empfehlen [Cerami, 2002]. Entsprechend verteilen sich die Muster auf die drei Klassen „harte", „weiche" oder „keine" empfohlene Kontaktlinsen. Das Klassifikationsproblem ist der Weka-Bibliothek entnommen und besteht aus nur 24 Mustern zu je 4 Merkmalen. Entsprechend der geringen Anzahl von Mustern ist die statistische Aussagekraft in Frage gestellt, ermöglicht jedoch eine vollständige Untersuchung aller Kombinationen von Klassifikatoren mit Selektoren. Die Promising List und die Blasensynthese benötigten für die Synthese eines Klassifikationssystems etwa 80 Sekunden. Beide Synthesemethoden liefern mit 16,6 Prozent auch die geringsten Klassifikationsfehler. Die Blasensynthese BF sowie One Selector generieren reproduzierbar Klassifikationssysteme mit einem Fehler von 29 Prozent. Der vorgegebene Selektor dieser Synthesemethode ist für das Klassifikationsproblem ungeeignet. Eine einfache Synthesemethode ohne Merkmalsselektion wie der Zero Selector liefert Klassifikationssysteme mit vergleichbarem Generalisierungsfehler. Für vergleichbare Klassifikationsprobleme zahlt sich eine Reduktion der Muster daher nicht aus.

g) Klassifikationsproblem Gehaltsverhandlungen „labor.arff"

Origin: Stan Matwin, Computer Science Dept, University of Ottawa, (mailto:stan@uotcsi2.bitnet)

Eigenschaften: ca. 3,5% Fehler, 16 Merkmale, 2 Klassen und 57 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	6 – 24	7,0 – 35,0
Single Random	Leave-one-out	2 – 64	7,0 – 35,0
Zero Selector	Leave-one-out	71 – 75	7
One Selector	Leave-one-out	19 – 76	3,5
Promising List	Leave-one-out	1366 – 1845	3,5
Blasensynthese BF	Leave-one-out	30 – 77	3,5 – 10,5
Blasensynthese	Leave-one-out	890 – 986	3,5

Dieses Klassifikationsproblem enthält die Muster über Tarifabschlüsse von Firmen in Kanada und ist in der Weka-Bibliothek enthalten. Die Muster werden in die beiden Klassen „guter“ und „schlechter“ Abschluß eingeteilt. Das Klassifikationsproblem verfügt über eine geringe Anzahl von 57 Mustern mit je 16 Merkmalen. Aufgrund der geringen Größe kann dieses Klassifikationsproblem mehrfach bei vollständiger Kreuzvalidierung untersucht werden. Der Vorteil der Blasensynthese kommt aber wegen der geringen Anzahl an Mustern nicht zum Tragen. Ausgehend von den Synthesemethoden mit Zufallsstrategie bis zu den komplexeren Methoden ist eine starke Verbesserung des mittleren Klassifikationsfehlers zu beobachten. Die Blasensynthese und die Promising List liegen in bezug auf Klassifikationsfehler und Laufzeit gleichauf. Die Blasensynthese BF kann wie die Synthesemethode One Selector erhebliche Geschwindigkeitsvorteile bei annähernd gleichem Klassifikationsfehler vorweisen. Creators: Collective Barganing Review, monthly publication, Labour Canada, Industrial Relations Information Service, Ottawa, Ontario, K1A 0J2, Canada, (819) 997-3117

h) Klassifikationsproblem Soja-Bohnen „soybean.arff“

Origin: R.S. Michalski and R.L. Chilausky,
International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

Eigenschaften: ca. 3% Fehler, 35 Merkmale, 19 Klassen und 683 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	44 – 5384	6,9 – 72,0
Zero Selector	10-fold	288 – 299	6,7 – 8,8
One Selector	10-fold	177 – 182	7,2 – 9,1
Promising List	10-fold	91621	7,0
Blasensynthese BF	10-fold	103 – 107	7,3 – 7,6
Blasensynthese	10-fold	24147	7,1

Mit diesem Klassifikationsproblem kann für erkrankte Soja-Bohnen Pflanzen die Art ihrer Erkrankungen beurteilt werden. Die 683 Muster verteilen sich auf 15 verschiedene Klassen beziehungsweise mögliche Erkrankungen. Das beste bisher bekannte System aus gewichteten Netzwerken liefert einen Klassifikationsfehler von drei Prozent. Dieser Wert wird von keinem der synthetisierten Systeme erreicht. Selbst das durch die Promising List erstellte Klassifikationssystem erreicht einen Generalisierungsfehler von sieben Prozent. Anscheinend ist keine Kombination aus den in der Komponentensoftware implementierten Klassifikatoren und Selektoren für dieses Klassifikationsproblem geeignet. Die Promising List benötigt etwa 25 Stunden für die Synthese. Die Blasensynthese liefert vergleichbare Ergebnisse in einem Drittel der Zeit. Einen ähnlichen Generalisierungsfehler liefert auch die Blasensynthese BF. In weniger als zwei Minuten wird ein akzeptables Klassifikationssystem erstellt. Die Ergebnisse des One Selector entsprechen einer Untermenge der Blasensynthese BF. Dementsprechend müßten auch die Klassifikationsfehler der durch den One Selector erstellten Klassifikationssysteme in denen der durch Blasensynthese BF erstellten enthalten sein. Dies trifft bei diesem Klassifikationsproblem nicht zu. Eine mögliche Ursache hierfür kann die zehnfache Kreuzvalidierung sein.

i) Klassifikationsproblem Hepatitis „hepatitis.arff"

Origin: G. Gong (Carnegie-Mellon University) via Bojan Cestnik
Jozef Stefan Institute, Ljubljana, Yugoslavia

Eigenschaften: ca. 17% Fehler, 19 Merkmale, 2 Klassen und 155 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	4 – 81	14,8 – 20,8
Single Random	10-fold	2 – 259	12,4 – 20,6
Zero Selector	10-fold	22 – 24	13,5 – 17,3
One Selector	10-fold	16 – 18	12,9 – 13,5
Promising List	10-fold	533 – 536	11,4 – 12,8
Blasensynthese	10-fold	653 – 683	11,5 – 12,7

Dieses Klassifikationsproblem kann zur Vorhersage des Ausganges einer Hepatitis Erkrankung eingesetzt werden. Die beiden möglichen Klassenzuweisungen beschreiben Tod oder Überleben eines Patienten. Bisherige Systeme haben einen minimalen Klassifikationsfehler von 17 Prozent erreicht. Dieser Wert wurde von allen Synthesemethoden deutlich unterboten. Mit der Promising List und der Blasensynthese konnten in etwa zehn Minuten Klassifikationssysteme erstellt werden, welche den Fehler sogar von 17 auf zwölf Prozent reduzierten. Wiederum liefern Promising List und Blasensynthese vergleichbare Klassifikationsfehler, wenn auch die Blasensynthese für dieses Klassifikationsproblem keinen Geschwindigkeitsvorteil bietet.

j) Klassifikationsproblem Pferde-Kolik „colic.arff"

Origin: Mary McLeish & Matt Cecile, Department of Computer Science, University of Guelph
Will Taylor (taylor@pluto.arc.nasa.gov)

Eigenschaften: ca. 13% Fehler, 23 Merkmale, 2 Klassen und 368 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	2 – 19	15,7 – 37,0
Single Random	10-fold	2 – 9	18,4 – 37,0
Zero Selector	10-fold	72 – 88	14,4 – 15,0
One Selector	10-fold	23 – 27	18,4 – 18,5
Promising List	10-fold	4564	13
Blasensynthese BF	10-fold	15 – 17	18,4 – 18,5
Blasensynthese	10-fold	2350 – 3133	12,2 – 14,1

Dieses Klassifikationsproblem entstammt der Tiermedizin. Es repräsentiert die Merkmale unterschiedlicher Fälle von Koliken bei Pferden. Mit den Daten soll über die Notwendigkeit eines chirurgischen Eingriffes im Falle einer Erkrankung entschieden werden. Die beiden Klassen teilen die Muster in Operation „notwendig" oder „nicht notwendig" ein. Für das Klassifikationsproblem sind keine früheren Klassifikationen in der Literatur bekannt. Die Planung mit Hilfe der Promising List benötigt für das mittelgroße Klassifikationsproblem etwa 75 Minuten bei einer zehnfachen Kreuzvalidierung. Der Klassifikationsfehler liegt bei etwa 13 Prozent. Die Blasensynthese benötigt für vergleichbare Klassifikationsergebnisse im Mittel 45 Minuten. Die Synthese durch One Selector oder Blasensynthese BF liefert innerhalb weniger Sekunden ein Ergebnis. Der fest vorgegebene Selektor der beiden Synthesemethoden ist für das Klassifikationsproblem offensichtlich ungeeignet, da sich der Klassifikationsfehler von 13 auf 18 Prozent verschlechtert. Allerdings wird innerhalb weniger Sekunden ein Klassifikationssystem erstellt, welches interaktiv optimiert werden kann.

k) Klassifikationsproblem Zoo „zoo.arff"

Origin: Richard S. Forsyth, 8 Grosvenor Avenue, Mapperley Park, Nottingham NG3 5DX

Eigenschaften: ca. 2% Fehler, 17 Merkmale, 7 Klassen und 101 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	10-fold	2 – 4	4,0 – 59,2
Single Random	10-fold	2 – 4	5,9 – 59,4
Zero Selector	10-fold	4 – 19	4,0 – 6,0
One Selector	10-fold	19 – 20	4,0 – 6,0
Promising List	10-fold	219 – 236	1,9 – 2,0
Blasensynthese	10-fold	138 – 163	1,9 – 2,0

Mit dem Klassifikationsproblem „Zoo“ soll Aufschluß über die Zugehörigkeit von Tieren zu unterschiedlichen Gattungen gewonnen werden. Mit den Merkmalen sollen die Muster einer der sieben Klassen zugeteilt werden. In der Literatur werden keine Klassifikationsergebnisse anderer Systeme erwähnt. Der minimale Klassifikationsfehler von zwei Prozent wird erwartungsgemäß durch die Promising List und die Blasensynthese erreicht. Die Blasensynthese kann leichte Geschwindigkeitsvorteile verbuchen. Für dieses Klassifikationsproblem werden schon mit sehr einfachen Planern wie dem One Selector sehr gute Ergebnisse erreicht. Mit nur wenig mehr Zeitaufwand wird innerhalb von etwa drei Minuten ein optimales Klassifikationssystem durch Promising List oder Blasensynthese synthetisiert.

l) Klassifikationsproblem Wetter „weather.arff“

Origin: Fiktives Beispiel aus I. Witten, E. Frank, Data Mining [Witten, 2000]

Eigenschaften: ca. 15% Fehler, 4 Merkmale, 2 Klassen und 14 Muster.

Synthesemethode	Test	Zeit in s	Fehler in %
Simple Single Random	Leave-one-out	1 – 6	21,4 – 85,7
Single Random	Leave-one-out	1 – 38	21,4 – 71,4
Zero Selector	10-fold	13 – 21	15,0 – 30,0
One Selector	10-fold	16 – 38	35,7
Promising List	10-fold	210	15,0
Blasensynthese	10-fold	185 – 213	15,0

Dieses Klassifikationsproblem ist in der Weka-Bibliothek enthalten. In dem fiktiven Beispiel sollen Entscheidungen darüber getroffen werden, ob unter bestimmten Wetterbedingungen ein Fußballspiel ausgetragen werden soll oder nicht. Die Klassifikationssysteme mit minimalem Klassifikationsfehler wurden von der Blasensynthese und der Promising List synthetisiert. Auf dem relativ kleinen Klassifikationsproblem lassen sich Vorhersagen mit einer Genauigkeit von 15 Prozent treffen. Der Geschwin-

digkeitsvorteil der Blasensynthese greift nicht aufgrund des Verhältnisses von Merkmalen zu Mustern. Eine festverdrahtete Selektion wie bei dem One Selector führt offensichtlich zu schlechteren Ergebnissen. Der Zero Selector ohne Merkmalsreduktion liefert für diesen Fall in kürzerer Zeit wesentlich bessere Klassifikationsergebnisse.

Anhang C: Varianzen bei zehnfacher Kreuzvalidierung

Zu der Untersuchung des Verhaltens verschiedener Klassifikatoren bei zehnfacher Kreuzvalidierung wurde für die folgenden beiden Klassifikationsaufgaben mehrmals hintereinander ein Klassifikator trainiert und sein Generalisierungsfehler geschätzt. Die Aufgabe verfügte über einen vorgegebenen Generalisierungsfehler von zehn Prozent, fünf Klassen, drei Merkmale und 1000 Observationen.

C.1 Klassenweise gleichverteilte Merkmale

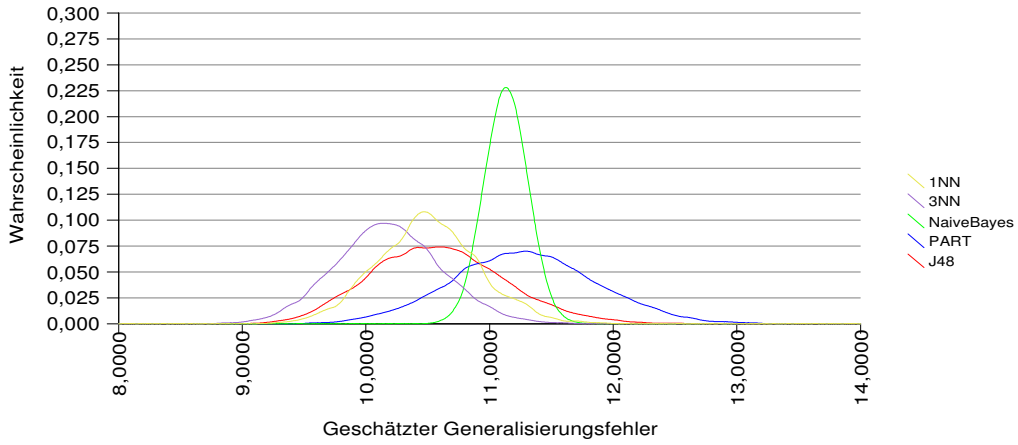


Abbildung 23. Deutlich zu sehen ist die gegenüber dem Bayes-Klassifikator hohe Varianz der anderen Klassifikatoren. Der Generalisierungsfehler eines Klassifikationssystems kann mit zehnfacher Kreuzvalidierung offensichtlich nicht sehr genau geschätzt werden.

C.2 Klassenweise normalverteilten Merkmale

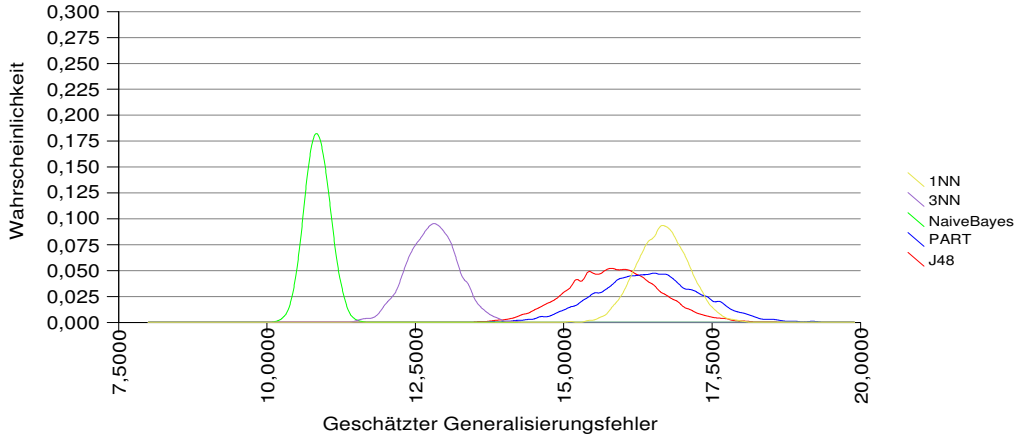


Abbildung 24. Der Bayes-Klassifikator ist der für Klassifikationsaufgaben mit normalverteilten Merkmalen am besten geeignete Klassifikator. Die Klassifikatoren PART und J48 zeichnen sich in diesem Fall durch eine besonders hohe Streuung bei zehnfacher Kreuzvalidierung aus.

Anhang D: Komponentensoftware

D.1 Klassifikatoren

Klassifikator	*Beschreibung des Klassifikators*
A-Priori	Komponente ZeroR: Nutzung der a-Priori Wahrscheinlichkeiten von Klassen ohne Verwendung von Merkmalen
Single Feature	Komponente OneR: Klassifikation mit Hilfe des einzigen, als bestes eingestuften Merkmals
k-nearest-Neighbour	Komponente kNN: Schätzung der lokalen Wahrscheinlichkeitsdichten durch Nachbarschaftsbeziehungen
Naive-Bayes	Komponente Naive Bayes: Annahme von Unabhängigkeit der Merkmale und Klassifikation mittels kleinstem Bayes-Fehler
Lineare Regression	Komponente Linear Regression: Klassifikation aufgrund linearer Trenngrenzen zwischen den Klassen
Support-Vector-Machine	Komponente SMO: Sequential Minimal Optimization – Nichtlineare Transformation der Merkmale mit anschließender linearer Diskriminanzanalyse
Gewichtete Regression	Komponente LWR: Locally Weighted Regression – Lineare Regression mit Gewichtung der Muster durch den Abstand
Entscheidungsbaum	Komponente J48: Entscheidungsbaum basierend auf Informationsgewinn bei dem Split an einzelnen Merkmalen
Regelbasiert	Komponente PART: Erstellen von Entscheidungsregeln aufgrund von „pruned“ Entscheidungsbäumen
Neuronales Netz	Komponente NeuralNet: Neuronales Netz mit Backpropagation
Modellbaum	Komponente M5Prime: Numerische Vorhersage durch Modellbaum (Entscheidungsbaum mit Modell in den Blättern)
Decision Stump	Komponente Decision Stump: Reduzierter Entscheidungsbaum mit nur einem Level, oft bei Boosting oder Bagging verwendet
Entscheidungstabelle	Komponente Decision Table: Entscheidungstabelle basierend auf durch Best-First-Selektor eingeschränkter Merkmalsmenge

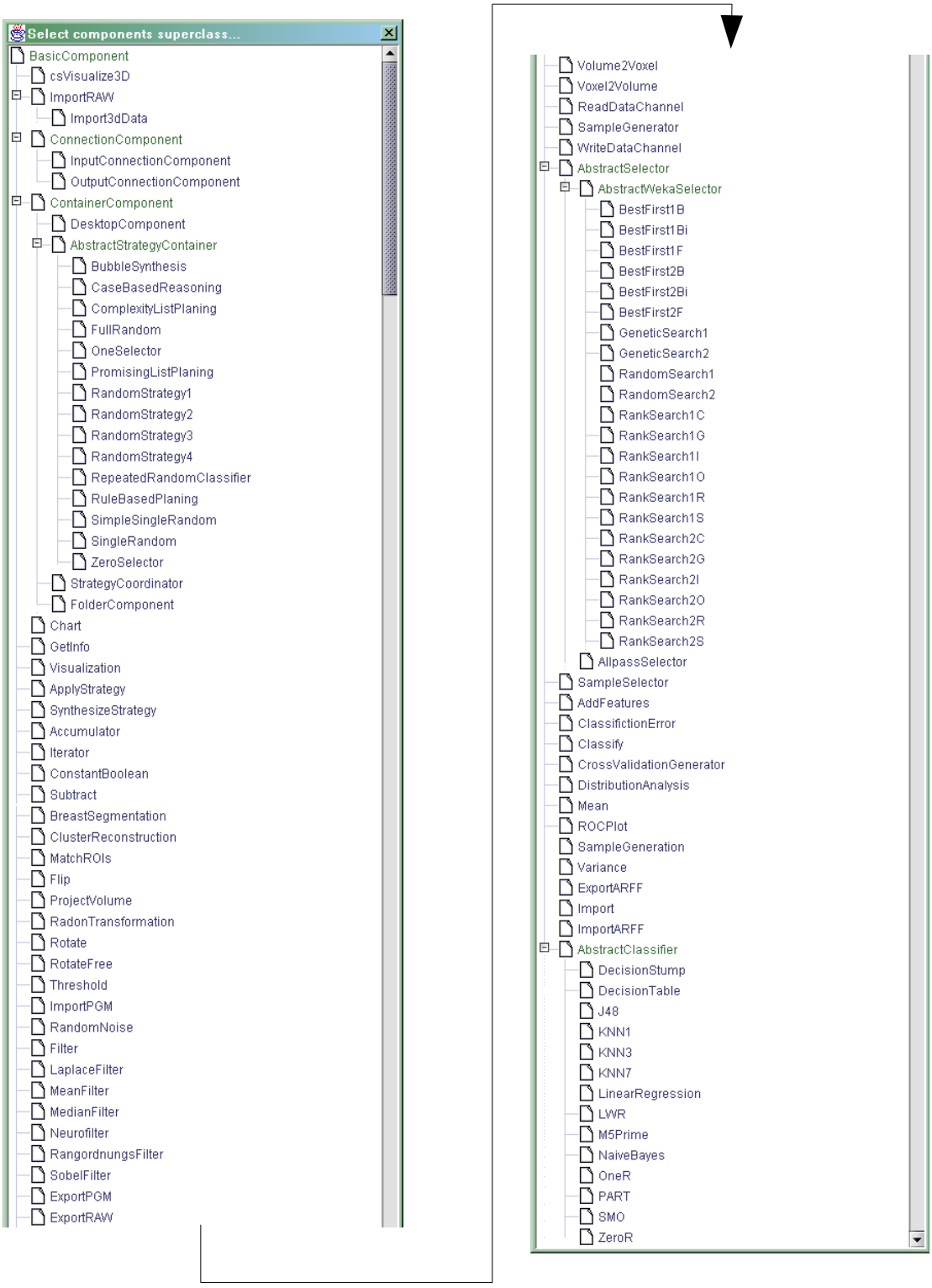
Tabelle 6. Übersicht und kurze Beschreibung über die in der Komponentensoftware integrierten Klassifikatoren. Die Algorithmen entstammen der freien Klassifikationsbibliothek Weka [Witten, 2000].

D.2 Selektoren

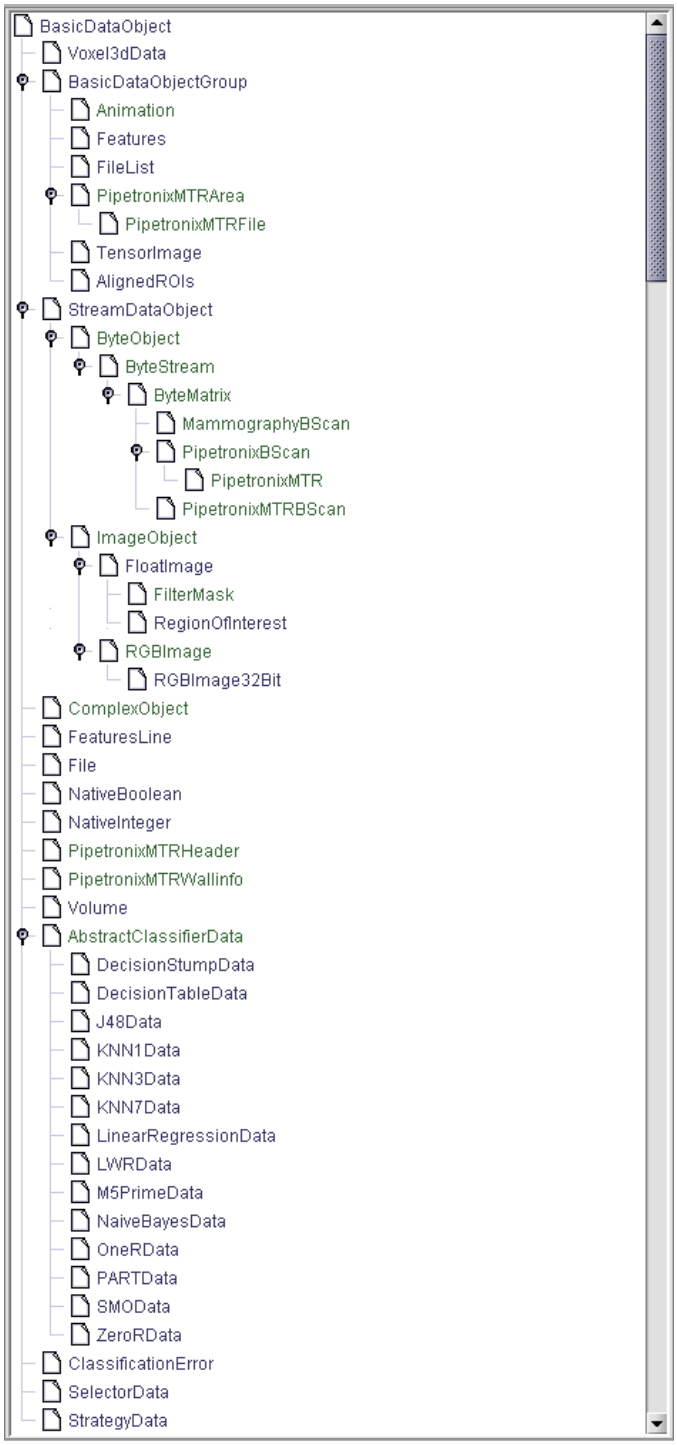
Selektor	*Beschreibung des Selektors*
Allpaß: ein Selektor	Dummy-Selektor läßt alle Merkmale passieren
Best-First: sechs Selektoren	Schema Forward, Backward und Bidirektionale Suche: - jeweils mit einer korrelationsbasierten Bewertung der Auswahl - jeweils mit einer statistischen Bewertung der Auswahl
Zufallssuche: zwei Selektoren	Zufällige Auswahl von Selektoren: - mit einer korrelationsbasierten Bewertung der Auswahl - mit einer statistischen Bewertung der Auswahl
Genetische Algorithmen: zwei Selektoren	Genetische Algorithmen zur Auswahl von Merkmalen: - mit einer korrelationsbasierten Bewertung der Auswahl - mit einer statistischen Bewertung der Auswahl
Rangsuche: zwölf Selektoren	Auswahl der Merkmale nach vorgegebener Ordnung: - Bewertung durch Chi-Squared Test - Bewertung durch Gain Ratio - Bewertung durch Information Gain - Bewertung durch bestes Merkmal - Bewertung durch Nachbarschaften - Bewertung durch Wahrscheinlichkeit Jeweils mit Bewertung der Auswahl: - mit einer korrelationsbasierten Bewertung der Auswahl - mit einer statistischen Bewertung der Auswahl

Tabelle 7. Übersicht und kurze Beschreibung über die in der Komponentensoftware integrierten Selektoren. Die Algorithmen entstammen der freien Klassifikationsbibliothek Weka [Witten, 2000]. Detaillierte Erläuterungen und tiefergehende Literaturhinweise zu den Verfahren finden sich in der Dokumentation der Bibliotheken.

D.3 Hierarchie der Softwarekomponenten



D.4 Abstrakte globale Datentypen



D.5 Manuelle Klassifikation

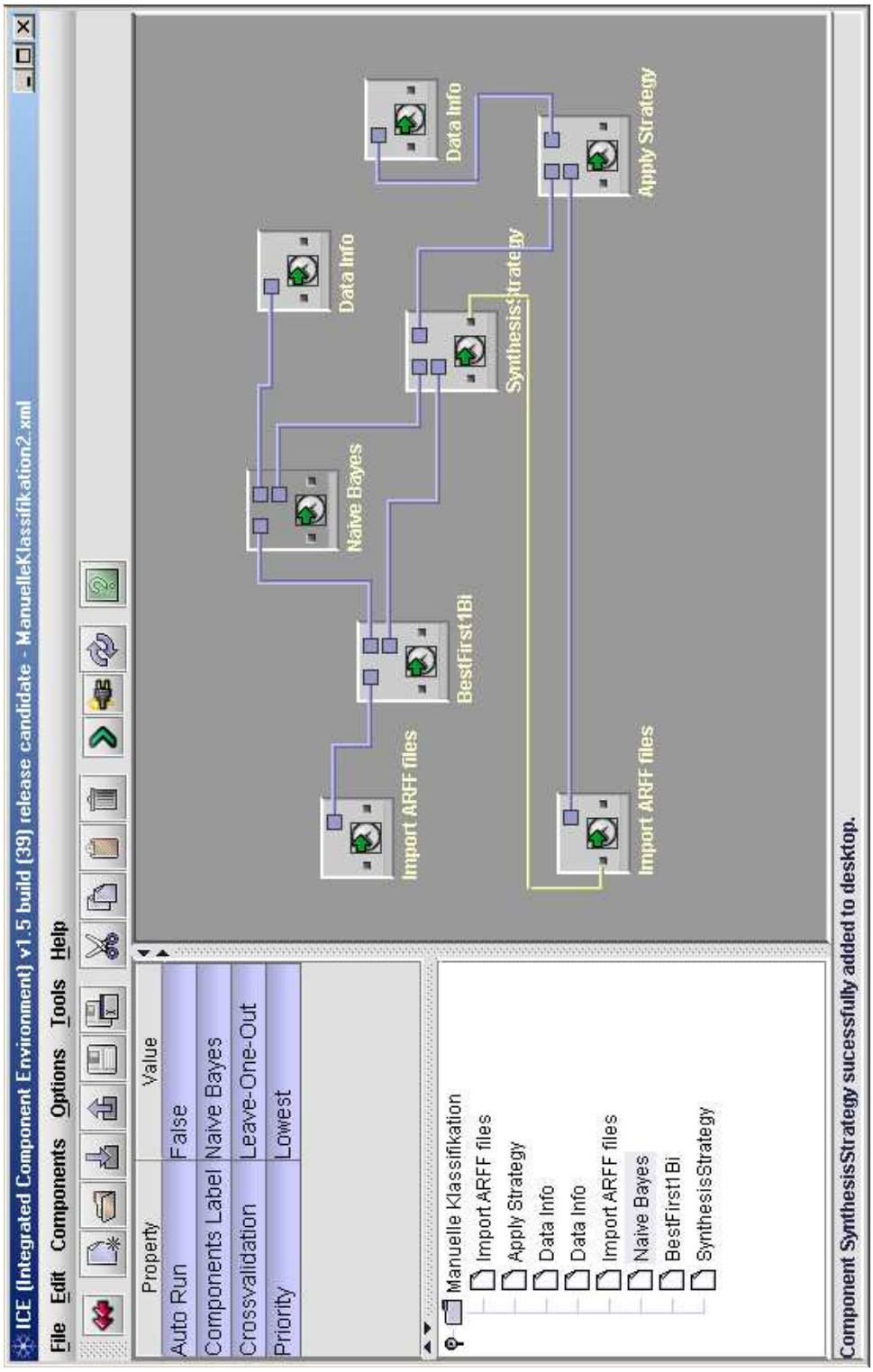


Abbildung 25. Beispiel einer manuell erstellten Ablaufstruktur zur Klassifikation.

Anhang E: Pseudo-Code

E.1 Synthesemethoden

a) Zufällige Klassifikatorwahl

```
Simple Single Random SSR
    s = Selector(AllpassSelector)
    c = getRandomClassifier()
    addConnection(s,c)
    s.start()
    c.start()
    return Strategy(s,c)
```

b) Wiederholte zufällige Klassifikatorwahl

```
Zero Selector ZS
    try {
        s = Selector(AllpassSelector)
        s.start()
        while (moreClassifiers()) {
            c = getRandomClassifier()
            addConnection(s,c)
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

c) Zufällige Klassifikatorwahl mit Selektor

```
Simple Random SR
    s = getRandomSelector();
    c = getRandomClassifier();
    addConnection(s,c)
    s.start()
    c.start()
    return Strategy(s,c);
```

d) Wiederholte zufällige Klassifikatorwahl mit Selektor

```
Repeated Random Classifier with Selector RRCS
    try {
        s = getRandomSelector();
        s.start();
        while (moreClassifiers()) {
            c = getRandomClassifier()
            addConnection(s,c)
            c.start()
            if (c.getError() < bestStrategy.getError()) {
```

```
                bestStrategy = Strategy(s,c)
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

e) Wiederholte zufällige Klassifikatorwahl mit wiederholter Selektion

```
Repeated Random Classifier with Repeated Selector RRCRS
    try {
        c = Classifier(NaiveBayes)
        while (moreSelectors()) {
            s = getRandomSelector()
            addConnection(s,c)
            s.start()
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
        while (moreClassifiers()) {
            c = getRandomClassifier()
            addConnection(bestSelector,c)
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

f) Vollständige zufällige Klassifikatorwahl mit Selektor

```
Full Random FR
    try {
        while (moreSystems()) {
            s = getRandomSystemSelector()
            c = getRandomSystemClassifier()
            addConnection(s,c)
            s.start()
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

g) Wiederholte zufällige Klassifikatorwahl mit Selektor bei Testfehlernähe

```
Repeated Random Classifer with Conditioned Selector RRCCS
    try {
        while (moreClassifiers()) {
            s = Selector(AllpassSelector)
            c = getRandomClassifier()
            addConnection(s,c)
            s.start()
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
            if (c.getError() - 5 < maxTolerableError) {
                while (moreSelectors()) {
                    s = getRandomSelector()
        addConnection(s,c)
                    s.start()
                    c.start()
                    if (c.getError() < bestStrategy.getError()) {
                        bestStrategy = Strategy(s,c)
                    }
                }
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return Strategy(bestSelector,bestClassifier)
```

h) Parallelverarbeitung

```
Parallel Synthesis PS
    try {
        while (moreSystems()) {
            s = getRandomSystemSelector()
            c = getRandomSystemClassifier()
            p = getFreeRemoteProzessor()
            forkRemoteSynthesis(p,s,c)
        }
        while (prozessorsPending()) {                                   strategy =
getNextProzessorResult()
            if (strategy.getError() < bestStrategy.getError()) {
                bestStrategy = strategy
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

i) Zufallssuche nach Test stabiler Klassifikatoren

```
Best Classifiers First BCF
    try {
        c = Classifier(NaiveBayes)
        while (moreSelectors()) {
```

```
            s = getRandomSelector()
            addConnection(s,c)
            s.start()
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
        c = Classifier(kNN)
        while (moreSelectors()) {
            s = getRandomSelector()
            addConnection(s,c)
            s.start()
            c.start()
            if (c.getError() < bestStrategy.getError()) {
                bestStrategy = Strategy(s,c)
            }
        }
        bestStrategy = RRCCS(bestStrategy,pendingClassifiers)
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    return bestStrategy
```

j) Vollständige Liste von Klassifikatoren nach Erfolgsaussicht

```
Promising List PL
    try {
        while (morePromisingSystems()) {
            c = getNextPromisingClassifier()
            while (morePromisingSelectors()) {
                s = getNextPromisingSelector()
                addConnection(s,c)
                s.start()
                c.start()
                if (c.getError() < bestStrategy.getError()) {
                    bestStrategy = Strategy(s,c)
                }
            }
        }
    } catch (TimeOutException)
    } catch (ExceedMaxErrorException)
    } catch (ExceedBestSystemProbabilty)
    } catch (InterruptedException)
    updatePromisingLists()
    return bestStrategy
```

k) Blasensynthese

```
Bubble Synthesis BS
    try {
        bestStrategies.init()
        observations = getRandomObservations(m1)
        while (moreSystems()) {
            strategy = getNextBestStrategy()
            strategy.start()
        }
        bestStrategies.sort()
```

```
        bestStrategies.threshold(n)
        observations = getRandomObservations(m2)
        while (moreBestStrategies()) {
            strategy = getNextBestStrategy()
            strategy.start()
        }
        bestStrategies.sort()
        bestStrategies.threshold(l)
        observations = getRandomObservations(m3)
        while (moreBestStrategies()) {
            strategy = getNextBestStrategy()
            strategy.start()
        }
    } catch (TimeOutException)
    } catch (InterruptedException)
    bestStrategies.sort()
    return bestStrategies.getFirstElement()
```

E.2 Künstliche Klassifikationsprobleme

a) Klassenweise gleichverteilte Merkmale

```
Equally Distributed Set EDS
    data.init(o, m, k);
    for (i=0; i<o; i++) {
        class = getRandomInteger(0, k-1)
        for (j=0; j<m; j++) {
            x = calculateEqualDistributionOffset(m, k, eg)
            value = getRandomDouble(0, 1) + class*x
            data.setFeature(i, j, value)
        }
        data.setClass(i, class)
    }
```

b) Klassenweise normalverteilte Merkmale

```
Normally Distributed Set NDS
    data.init(o, m, k);
    for (i=0; i<o; i++) {
        class = getRandomInteger(0, k-1)
        for (j=0; j<m; j++) {
            x = calculateNormalDistributionOffset(m, k, eg)
            value = getRandomGaussian() + class*x
            data.setFeature(i, j, value)
        }
        data.setClass(i, class)
    }
```

Anhang F: Komplexe Synthesemethoden

Leistungsfähigere Synthesemethoden sind aufwendiger zu implementieren. Die im folgenden vorgestellten Synthesemethoden wurden aus diesem Grund noch nicht in der Komponentensoftware für die Klassifikation implementiert. Ihre Beschreibungen können jedoch als Grundlage für eine effektivere Synthese von Klassifikationssystemen verwendet werden.

F.1 Synthese mit Interpretation der Eingangsdaten

Eine Interpretation der Eingangsdaten setzt beinahe immer problemspezifisches Wissen voraus. Bekannte Lösungen werden beispielsweise in einer Datenbank abgelegt. Eine neue Lösung für ein unbekanntes Klassifikationsproblem wird dann aus den bekannten Lösungen der Datenbank konstruiert.

a) Synthese durch Klassifikation

Analog der Klassifikation von Mustern liegt es nahe, ein Klassifikationssystem aufgrund bereits gelöster Klassifikationsprobleme aufzubauen. Eine neue Klassifikationsaufgabe wird anhand von Meta-Merkmalen einer Meta-Klasse von Problemen zuordnet, für die eine bestimmte Lösungsstrategie optimal ist. Dies entspricht einer Klassifikation von Klassifikationsaufgaben.

- Die größte Schwierigkeit liegt darin, die Lösungen bekannter Klassifikationsprobleme durch geeignete Merkmale in einem Merkmalsraum zu repräsentieren. Dieses Problem zieht sich konsequent durch alle Synthesemethoden, die nach bereits bekannten Lösungen suchen und diese mit den aktuellen Anforderungen vergleichen. Eine numerische Meta-Beschreibung eines Klassifikationsproblems ist zumeist nur schwer möglich. Eine Klassifikationsaufgabe läßt sich wesentlich einfacher wie in den folgenden Abschnitten erläutert nominell beschreiben, oftmals sogar nur durch boolsche Aussagen. Mit solchen Merkmalen sind Vergleiche möglich. Aus diesem Grund läßt sich eine Klassifikation von Klassifikationsproblemen gut durch Entscheidungsbäume realisieren.

- Für unbekannte Klassifikationsprobleme bietet sich diejenige Lösung als Grundlage an, welche für ein ähnliches Problem bereits verwendet wurde. Ähnlichkeit definiert sich in diesem Zusammenhang durch Nachbarschaftsbeziehungen von Klassifikationsaufgaben. Zur Auswahl einer benachbarten Lösung kann analog zur herkömmlichen Klassifikation der k-nearest-Neighbour Algorithmus eingesetzt werden. Bei dem Einsatz von k-nearest-Neighbour stellt sich zusätzlich zur Meta-Beschreibung ein weiteres Problem. Um eine bereits bekannte Lösung mit dem Algorithmus zu finden, ist ein geeignetes Abstandsmaß für den aus den Meta-Beschreibungen aufgespannten Merkmalsraum notwendig. Da die Merkmale hierfür meist nicht numerisch sind, gestaltet sich die Definition eines Abstandsmaßes schwierig. Ein mögliches Maß ist beispielsweise die Anzahl von Übereinstimmungen nomineller Merkmale.

Theoretisch läßt sich für eine Klassifikation jeder beliebige Klassifikationsalgorithmus einsetzen. Entscheidend für die Qualität ist wie bei jedem Klassifikationssystem vor allem eine geeignete Wahl der Merkmale und eine umfangreiche Stichprobe.

b) Synthese durch Case Based Reasoning

Die Datenbank mit den Lösungen zu den bereits bekannten Klassifikationssystemen kann in Form von „Expertenwissen“ realisiert werden. Eine gängige Methode, Expertenwissen zu verwalten und anzuwenden, ist fallbasiertes Schließen (***Case Based Reasoning, CBR***) [Althoff, 1992]. Case Based Reasoning modelliert den Prozeß der menschlichen Entscheidungsfindung und simuliert softwaretechnisch das Entscheidungsverhalten eines Experten. Ein CBR-System setzt sich aus vier Teilen zusammen: Retrieve, Reuse, Revise und Retain [Korf, 2000]. Der Schritt Retrieve ist für das Wiederauffinden eines bereits bekannten Falles zuständig. Reuse paßt die vorhandene Lösung eines Falles an das aktuelle Problem an. Revise prüft, ob die angepaßte Lösung erfolgreich ist und fordert gegebenenfalls eine Korrektur. Retain schließlich sorgt dafür, daß neue Lösungen in die Falldatenbank aufgenommen und ungültig gewordene daraus entfernt werden. Die Schwierigkeit eines solchen Systems liegt vor allem in der notwendigerweise expliziten Repräsentation von Expertenwissen im Abschnitt Retrieve. Dieses Wissen muß durch einen Experten eingebracht werden, wobei oftmals keine konkreten Regeln zur Verfügung stehen, sondern jahrelange Erfahrung ein „Gefühl“ für die jeweilige Situation vermittelt. Auch der Schritt Reuse erfordert detaillierte Beschreibungen, auf welche Weise eine bereits vorhandene Lösung an ein gegebenes Problem angepaßt werden kann. Der Transfer des Expertenwissens ist in der Praxis mit erheblichen Komplikationen verbunden. Wie bei der Synthese durch Klassifikation liegt eine zusätzliche Schwierigkeit in der Meta-Beschreibung eines vorliegenden Klassifikationsproblems. Ohne eine geeignete Meta-Beschreibung kann im Schritt Retrieve keine Verbindung zu bekannten Lösungen hergestellt werden. Case Based Reasoning ist ein Modell zur Umsetzung

von Expertensystemen. In der Praxis ist diese Umsetzung jedoch mit großen Schwierigkeiten verbunden.

c) Synthese durch Klassenhierarchie

Betrachtet man einen Ablauf aus aufeinanderfolgenden Algorithmen abstrakt, bestehen zwischen den einzelnen Algorithmen notwendigerweise genau definierte Schnittstellen. Das Ergebnis eines Algorithmus wird als Parameter für einen anderen Algorithmus verwendet. Dessen Ergebnis wiederum kann als Parameter weiterer Algorithmen verwendet werden. Ein Algorithmus mit berechnetem Ergebnis wird als Provider, ein Algorithmus mit offenem Ergebnis als Consumer definiert. Das Ergebnis des Producers liegt an dessen Ausgang bereit und wird an den Eingang des Consumer angelegt. Producer und Consumer lassen sich genau dann miteinander verbinden, wenn die Datentypen ihrer Schnittstelle übereinstimmen. Zur Synthese eines Ablaufs werden zunächst statische Producer verwendet, d.h. Producer welche nicht von anderen Algorithmen abhängen. Eventuell notwendige Parameter werden beispielsweise durch einen Benutzer übergeben. Diese Initialisierung verankert einen Ablauf. Im weiteren wird für jeden Producer anhand des Datentypen seines Ergebnisses eine Menge von potentiellen Consumern erstellt. Aus dieser Menge wird jeweils ein Consumer ausgewählt und mit dem Ergebnis des Producers parametrisiert. Nachdem der Consumers seine Arbeit beendet hat, wird er als Producer behandelt und sein Ergebnis anderen Consumern zur Verfügung gestellt. Nach und nach entsteht ein Strukturbaum, der durch einen Algorithmus ohne Berechnung neuer Daten, wie beispielsweise einer Visualisierung, terminiert wird. Die Synthese sollte durch zusätzliche Randbedingungen kontrolliert werden, um die Terminierung zu garantieren. Ist das Ergebnis unbefriedigend, weil beispielsweise der Generalisierungsfehler eines synthetisierten Klassifikationssystems zu hoch ist, wird der Ablauf mittels Back-Tracking zurück gesetzt.

Ein Vergleich auf genau übereinstimmende Datentypen schränkt die Synthese sehr ein. Das Konzept läßt sich durch eine objektorientierte Programmiersprache noch verbessern. Anstelle von Datentypen verwendet man Klassen. Eine objektorientierte Programmiersprache verfügt über eine Klassenhierarchie. Die Schnittstelle zwischen Producer und Consumer wird genau dann als gleich definiert, wenn die Ausgangsklasse des Producers gleich der Eingangsklasse des Consumers ist, oder wenn die Eingangsklasse des Consumers von der Ausgangsklasse des Producers abgeleitet ist. Dies eröffnet mehr Kombinationsmöglichkeiten und erlaubt spezialisiertere Algorithmen.

Diese Synthesemethode ist datenflußgesteuert und eher für die Synthese komplexerer Ablaufstrukturen als die eines Klassifikationssystems geeignet. Der Nachteil einer Synthese durch Klassenhierarchie liegt in der Komplexität und dem Garantieren einer sinnvollen Terminierung. Der große Vorteil

besteht darin, daß implizite Informationen wie die Klassenhierarchie verwendet werden und daher keine Meta-Information oder Interaktion durch einen Benutzer notwendig ist.

d) Synthese durch Meta-Beschreibung

Die Notwendigkeit einer Meta-Beschreibung von Klassifikationsproblemen zieht sich durch nahezu alle intelligenteren Synthesemethoden. Eine numerische Beschreibung der Probleme läßt sich meist nicht formulieren. Entscheidend ist die Assoziation zwischen beiden Problemen, die durch eine Meta-Beschreibung mit nominellen Merkmalen in den meisten Fällen einfacher durchzuführen ist. Oftmals lassen sich die Meta-Beschreibungen zu boolschen Ausdrücken vereinfachen. Die folgenden Beispiele nomineller, boolscher und numerischer Merkmale könnten für eine medizinische Klassifikationsaufgabe als Meta-Beschreibungen verwendet werden: Medizinisch, Röntgenbild, Knochen, Ultraschall, Kontrast, Helligkeit, Hochpaß-Filter, zweidimensional, Tumor, LaPlace-Filter, Auflösung, Entropie, Alter, Geschlecht und beliebig viele mehr. Die Liste läßt sich nahezu endlos fortsetzen, ist aber für jede Klassifikationsaufgabe individuell. Bei einem Vergleich werden Merkmale, die nicht von beiden Problembeschreibungen verwendet werden, ignoriert. Um eine möglichst hohe Übereinstimmung zu erreichen, sollten alle zur Meta-Beschreibung verwendeten Merkmale in einer globalen Liste zu Verfügung stehen.

F.2 Synthese ohne Interpretation der Eingangsdaten

Die folgenden Synthesemethoden erfordern keine Kenntnisse über die Struktur oder Herkunft des jeweiligen Klassifikationsproblems. Die Methoden verknüpfen Komponenten auf verschiedene Arten und bewerten die Kombinationen. Sie eignen sich aufgrund ihrer Struktur besonders für Kombinationen bestehend aus mehreren Dutzend Komponenten. Bei einfachen Kombinationen, bestehend aus nur einem Merkmalsselektor und einem Klassifikator, sind die Methoden aufgrund des notwendigen Verwaltungsaufwands weniger gut geeignet.

a) Synthese durch genetische Algorithmen

Genetische Algorithmen lösen Probleme, indem sie mögliche Lösungen in einer Menge zusammenfassen und auf dieser evolutionäre Abläufe simulieren. Jedes Individuum der Menge stellt jeweils genau eine vollständige, mehr oder weniger erfolgreiche, Lösung des Problems dar. Die Lösung wird durch die Gensequenz des Individuums kodiert [Goldberg, 1989]. Auf der Population sind drei Operatoren definiert: Selektion, Progression und Mutation. Bei der Selektion werden untaugliche Individuen aus der Menge aussortiert. Die Tauglichkeit wird über eine „Fitneß-Funktion“ ermittelt. Durch Progression werden aus den verbliebenen Individuen Nachkommen gebildet. Dies geschieht beispiels-

weise durch Crossover oder Rekombination. Durch Mutation der Gene wird ein weiteres, zufälliges Element in die Fortpflanzung eingeführt. Hierzu wird ein Gen nach der Progression mit einer geringen Wahrscheinlichkeit modifiziert. Nicht immer ist die gesamte Menge an der Progression beteiligt. Zuvor kann eine Selektion geeigneter Eltern durchgeführt werden. Der gesamte Vorgang wird iterativ wiederholt, bis das Problem durch die Gensequenz eines Individuums optimal gelöst wird.

Das Problem bei der Umsetzung genetischer Algorithmen liegt normalerweise in der Codierung der Gensequenz. Die Synthese von Klassifikationssystemen ist mit Abbildung 9 jedoch leicht in eine Codierung umzusetzen. Die einfachste Lösung eines Klassifikationsproblems besteht aus einer Gensequenz in der Reihenfolge „Merkmalsselektor – Klassifikator“. Die Fitness-Funktion eines Klassifikationssystems gegenüber anderen Klassifikationssystemen ist durch den Generalisierungsfehler gegeben. Hierzu muß aus der Gensequenz eines Individuums ein Klassifikationssystem aufgebaut werden, um den Generalisierungsfehler zu schätzen.

Problematisch ist der Zeitpunkt des Abbruchs. Wann wurde die optimale Lösung gefunden? Theoretisch müßten hierzu alle möglichen Gensequenzen ausprobiert werden. In der Praxis beendet man die Synthese, wenn der Generalisierungsfehler eine tolerierbare Schwelle unterschreitet oder keine signifikante Änderung des Generalisierungsfehlers mehr stattfindet. Zur unbedingten Terminierung wird eine maximale Anzahl von Iterationsschritten vorgegeben.

Genetische Algorithmen sind sehr leistungsfähig und werden vor allem bei NP-vollständigen Problemen eingesetzt, für welche sie durch Näherungslösungen in akzeptabler Zeit gute Ergebnisse liefern. Sie sind anfällig für Oszillation und lokale Minima, weshalb Parameter wie die Mutationsrate kritischen Einfluß auf die Terminierung nehmen. Da der genaue Zeitpunkt der Terminierung nicht vorhersehbar ist, kann keine direkte Aussage über die Komplexität getroffen werden. Sie ist zumindest linear abhängig von der Anzahl der Muster und damit $\Omega(o)$, bedingt durch die Schätzung des Generalisierungsfehlers für die erstellten Klassifikationssysteme.

b) Synthese durch Grammatiken

Die Synthese durch Klassenhierarchie, bei der die Verknüpfungen von Daten erzeugenden Komponenten mit Daten verarbeitenden Komponenten implizit durch Klassenhierarchie gesteuert wird, führt zu der Idee einer expliziten Vorgabe aller zulässigen Verknüpfungen. Eine Verknüpfung gibt beispielsweise vor, daß auf die Komponente *A* entweder die Komponente *B* oder die Komponenten-Kombination *CD* verwendet werden muß. Die Kombination *CD* bedeutet die Ausführung der Komponente *C* mit anschließender Ausführung der Komponente *D*, wobei *C* als Datenerzeuger für *D* fungiert. Der-

artige Verknüpfungsvorschriften bezeichnet man als Grammatiken. Eine Grammatik besteht aus einem Quadrupel mit Terminalsymbolen, Nonterminalsymbolen, Produktionsregeln und einem Satzsymbol [Rechenberg, 1997]. Das Satzsymbol wird für die Initialisierung eines Satzes verwendet. In jedem Schritt wird zufällig eine Produktionsregel ausgewählt und auf den aktuellen Satz angewendet, wodurch ein neuer Satz entsteht. Die Terminalsymbole der Grammatik entsprechen den Komponenten. Ein Beispiel für eine Grammatik zur Synthese einfacher Klassifikationssysteme ist:

- Terminalsymbole = $\{s_1, s_2, s_3, k_1, k_2\}$
- Nonterminalsymbole = $\{S, K\}$
- Produktionsregeln = $\{O \rightarrow S, S \rightarrow s_1 K, S \rightarrow s_2 k_2, S \rightarrow s_3 k_1, K \rightarrow k_1, K \rightarrow k_2\}$
- Satzsymbol = $\{O\}$

Drei mögliche Selektoren werden durch die Symbole s_1, s_2 und s_3 repräsentiert, zwei mögliche Klassifikatoren durch k_1 und k_2. Die Produktionsregeln besagen, daß Selektor s_1 mit allen Klassifikatoren kombiniert werden darf, Selektor s_2 nur mit Klassifikator k_2 und Selektor s_3 nur mit Klassifikator k_1. Die Grammatik läßt folgende Systeme zu: $s_1 k_1$, $s_1 k_2$, $s_2 k_2$ und $s_3 k_1$. Im Gegensatz zur Synthese durch Klassenhierarchie wird stärker Einfluß auf mögliche Verknüpfungen genommen. Beispielsweise fließt in der obigen Grammatik das Expertenwissen ein, daß die Klassifikationssysteme $s_2 k_1$ und $s_3 k_2$ grundsätzlich sinnlos sind und daher bei der Synthese nicht untersucht werden müssen.

Zur Implementierung der Synthese ist eine Abbildung vorhandener Merkmalsselektoren und Klassifikatoren in ein Alphabet aus Terminalsymbolen notwendig. Das Aufstellen der Nonterminalsymbole und der Produktionsregeln erfordert Kenntnisse über das Zusammenspiel von Merkmalsselektoren und Klassifikatoren. Hierzu wird Expertenwissen benötigt. Sind beispielhafte Klassifikationsprobleme gegeben, können durch Synthesemethoden mit Zufallsstrategie Klassifikationssysteme synthetisiert und dabei Verknüpfungsvorschriften extrahiert werden. Gegebenenfalls erfordern diese eine interaktive Korrektur. Die Komplexität ist von den Produktionsregeln und der Anzahl der Symbole abhängig. Die Terminierung ist gewährleistet, wenn die Grammatik nicht rekursiv ist, d.h. wenn kein Nonterminalsymbol auf beiden Seiten der Produktionsregeln vorkommt.

c) Synthese durch statistische Verfahren

Bei der Synthese durch Grammatiken wird aus den Produktionsregeln zufällig eine Regel ausgewählt. Manche Produktionsregeln führen aber häufiger zum Ziel als andere. Diese sollten bevorzugt ausgewählt werden. Die Produktionsregeln werden hierzu nicht als gleichverteilt angesehen, sondern mit Häufigkeitsverteilungen gewichtet und mit der entsprechenden Gewichtung zufällig ausgewählt. Wie aber werden die Häufigkeitsverteilungen ermittelt?

Stehen ausreichend Beispiele von Abläufen zur Verfügung, kann ein zweidimensionales Histogramm über Kombinationen von Producern und Consumern erstellt werden. Beispielsweise könnte aufgrund empirischer Beobachtung ermittelt werden, daß auf den Merkmalsselektor A zehnmal häufiger der Consumer B als der Consumer C folgt. Das Histogramm ist identisch mit den Häufigkeitsverteilungen. Sind keine Beispiele für Klassifikationssysteme gegeben, wird das Histogramm während der Synthese durch andere Synthesemethode aus erfolgreichen Kombinationen von Merkmalsselektor und Klassifikator erstellt. Im Laufe der Zeit wird für die Produktionsregeln eine immer bessere Schätzung der Häufigkeitsverteilungen ermittelt.