MASTERARBEIT

im Rahmen des Universitätslehrganges
„Geographical Information Science and Systems (UNIGIS MSc)“
am Zentrum für GeoInformatik der
Paris-Lodron-Universität Salzburg

zum Thema

Objektklassifizierung mit Support Vector Machines

vorgelegt von

Roland J. Graf

U1436 / UNIGIS MSc Jahrgang 2009

zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems)“

Gutachter:
Ao. Univ. Prof. Dr. Josef Strobl

Betreuer:
Univ. Doz. Mag. Dr. Stefan Wegenkittl

19.Dezember 2011

Diese Arbeit ist meiner Tochter Sarah Maria gewidmet,
die in den letzten Jahren aufgrund meiner Studien
viel auf mich verzichten hat müssen.

Eidesstattliche Erklärung

Hiermit versichere ich, Roland J. Graf, geboren am 6.Mai 1964, diese Masterarbeit ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit die wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet.

Salzburg, 19.Dezember 2011 Roland Graf

Informationen

Vor- und Zuname:	DI(FH) DI Roland J. Graf
Institution:	Universität Salzburg
Universitätslehrgang:	Geographical Information Science & Systems (UNIGIS MSc)
Titel der Masterarbeit:	Objektklassifizierung mit Support Vector Machines
Gutachter:	Ao. Univ. Prof. Dr. Josef Strobl
Betreuer:	Univ. Doz. Mag. Dr. Stefan Wegenkittl

Schlagwörter

1. Schlagwort:	Object Classification
2. Schlagwort:	Support Vector Machines
3. Schlagwort:	Object Based Image Analysis

Kurzfassung

Diese Masterarbeit erklärt die theoretischen Grundlagen von Support Vector Machines (SVMs) und zeigt deren Anwendung in objektbasierten Landnutzungs- und -bedeckungsklassifizierungen.

Die Kapitel am Beginn der Master Thesis beschreiben die Grundlagen der Bildverarbeitung, sowie die zugrundeliegenden Konzepte der automatischen Klassifizierung mittels maschinellen Lernens und definieren die Notation von Merkmalsvektoren als Repräsentanten der zu klassifizierenden Daten. Nach einer Einführung in die Methoden numerischer Klassifizierung auf Grundlage von Diskriminantenfunktionen wird das mathematische Konzept von Linear-SVMs für linear trennbare Muster (Hard-Margin SVMs) und nicht-linear trennbare Muster (Soft-Margin SVMs) zur Separierung der Merkmalsvektoren mittels Hyperebenen erklärt. In einem weiteren Schritt wird das SVM-Modell mit Kernel-Methoden erweitert, um bei nicht linearen Klassifizierungsproblemen die Merkmalsvektoren in einem höherdimensionalen Merkmalsraums zu separieren.

Für eine Reihe von Versuchen wurde ein Definiens eCognition Plug-In für SVMs und mehrere MATLAB-Skriptdateien entwickelt. Der zweite Teil der Arbeit dokumentiert einige Implementierungsdetails und widmet sich dann gänzlich der Anwendung von SVMs in objektbasierten Bildklassifizierungssystemen. Ziel ist die Formulierung konkreter Empfehlungen zur Optimierung sowohl hinsichtlich der Daten als auch der Parametrisierung von SVMs, um eine Feinabstimmung des Klassifikators sowie eine Vereinfachung und Beschleunigung des Klassifizierungsprozesses zu erreichen.

Die Klassifikationsergebnisse und die Auswirkungen unterschiedlicher Parametersätze für SVMs werden anhand klassifizierter Daten mittels numerischen und grafischen, vergleichenden Methoden bewertet. Dabei werden sowohl Konturdiagramme zur Darstellung der Cross Validation Accuracies bei unterschiedlicher Parametrisierung und Kernel als auch Konfusionsmatrizen und Tabellen analysiert, um die Diskriminierungsfähigkeit von Kernel-SVMs und deren praktische Verwendbarkeit zu bewerten. Zuletzt werden einige Optimierungsmöglichkeiten – wie beispielsweise eine Merkmalstransformation, -selektion und -reduktion und eine Reduktion der Trainingsdaten – anhand von konkreten Beispielen gezeigt und deren Auswirkungen auf das Klassifikationsergebnis diskutiert.

Abstract

This Master Thesis introduces the theoretical background of Support Vector Machines (SVMs) and discusses their usage in object-based land use and land cover classification systems.

The chapters at the beginning of the Master Thesis describe the foundation of image processing, the underlying concepts of supervised learning methods and define the notation of feature vectors for data representation in classification problems. After outlining the numerical classification process and basics of linear discriminant function analysis, the thesis reviews the mathematical concept of Linear Support Vector Machines and Soft Margin methods to find separating hyperplanes for feature vectors. In a subsequent step, the model will be extended for nonlinear classification problems. Kernel methods will be introduced to map the data from the original input space into a kernel feature space of higher dimensionality.

For a series of experiments with SVMs, a Definiens eCognition SVM Plug-In and multiple MATLAB script files have been implemented. The second part documents some implementation details und evaluates the applicability of Support Vector Machines in object-based image classification systems. These chapters also describe the determination of preconditions and parameter settings for a successful SVM usage. Another goal is the formulation of recommendations for optimizing both, the data and the SVM model, to fine tune the classification results and to simplify and speed-up the classification process.

The quality of classification and the effect of different parameter settings are evaluated with pre-classified data by graphical and numerical comparisons. Contour plots of cross validation accuracies for selected parameters settings and kernels as well as confusion matrices and processing time comparisons are analyzed to evaluate the discriminatory power of Kernel-SVMs and their practical usability. Finally, optional optimization efforts (feature transformation, feature selection and training data reduction) are discussed and some examples show whether the changes affect the results.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Notation

$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{C}$	Menge der komplexen Zahlen
$\mathbb{N}$	Menge der natürlichen Zahlen
$\|X\|$	Anzahl der Elemente der endlichen Menge X (Kardinalität)
$\mathbf{A}^\top$	Transponierte Matrix $\mathbf{A}$
$\|\cdot\|$	Euklidische Norm, $\|\mathbf{x}\| := \sqrt{\mathbf{x}^\top \mathbf{x}}$
D	Menge der Daten $D = (X, Y) = T \cup E$
T	Menge der Trainingsdaten $T \subset D$
E	Menge der Testdaten $E \subset D, T \cap E = \{\}$
M	Anzahl der Trainingsdatensätze $M = \|T\|$
m	Anzahl der Eingangsvariablen, entspricht der Dimensionalität des Merkmalsraums $m = \dim(\mathcal{X})$
$\mathcal{X}$	Eingangsraum (Merkmalsraum) $\mathcal{X} = \mathbb{R}^m$ mit Dimension $m \in \mathbb{N}$
X	Menge der Eingangs(Roh-)daten $X \subset \mathcal{X}, X = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
$\mathbf{x}_i$	i-ter Vektor der Trainingsdaten $\mathbf{x}_i = (x_{i1}, \ldots, x_{im})$ $\mathbf{x} \in \mathcal{X}$ und $1 \leq i \leq M$
Ω_j	Klasse mit dem Index j
n	Anzahl der Klassen $n \in \mathbb{N}, n \geq 2$
K	Menge der Klassen $K = \{\Omega_1, \ldots, \Omega_n\}$
$\mathcal{Y}$	Ergebnisraum $\mathcal{Y} = \mathbb{R}^n$
Y	Menge der Klassenlabels $Y = \{y_1, \ldots, y_M\}$
y_i	Klassenlabel $y \in \mathcal{Y}$ assoziiert mit dem Eingangsvektor $\mathbf{x}_i$ $y_i \in \{1, \ldots, n\}$ oder $y_i \in \{-1, +1\}$ oder $y_i \in \mathbb{R}, y_i = (y_{i1}, \ldots, y_{in}), y_{ij} \geq 0, j = \{1, \ldots, n\}, \sum y_{ij} = 1$
S	Menge der Support Vector Indizes
B	Menge der begrenzten Support Vector Indizes
U	Menge der unbegrenzten Support Vector Indizes

$\mathbf{H}$	Hyperebene
$\mathbf{w}$	Normalvektor zur Hyperebene
$\mathbf{w}^\top\mathbf{x}$	Skalarprodukt der Vektoren $\mathbf{w}$ und $\mathbf{x}$
α_i, β_i	Lagrangemultiplikatoren assoziiert mit $\mathbf{x}_i$
ξ_i	Schlupfvariable assoziiert mit $\mathbf{x}_i$
C	Fehlergewicht (Größe des Margins)
b	Bias
$\mathcal{F}$	hochdim. SVM-Merkmalsraum, $dim(\mathcal{F}) > m$
$\phi(\mathbf{x}) : \mathcal{X} \to \mathcal{F}$	Funktion zur Transformation von $\mathbf{x} \in \mathcal{X}$ in den Merkmalsraum $\mathcal{F}$
$\mathcal{K}(\mathbf{x}, \mathbf{z})$	Kernelfunktion äquivalent zu $\phi(\mathbf{x})^\top\phi(\mathbf{z})$
γ	Parameter des RBF-Kernels
κ	Parameter des Sigmoid-Kernels

1 Einführung

Satellitengestützte Aufnahmeverfahren zeichnen heute schon Bilder der Erdoberfläche in einer Detailgenauigkeit auf, die vor einigen Jahren noch kaum denkbar gewesen wäre. Geosensoren liefern rund um die Uhr hochpräzise Daten aus den entlegensten Winkeln der Erde. Aktuelle Bildprodukte der Fernerkundung dringen mittlerweile mehr und mehr in den klassischen Bereich der Photogrammetrie vor (Jacobson, 2006), doch in Zukunft werden sowohl der stetige technische Fortschritt, von kommerziellen, militärischen und wissenschaftlichen Ambitionen getrieben als auch eine Absenkung der Satellitenbahnen noch höhere Auflösungen und Bildfrequenzen möglich machen. Diese Entwicklung führt unweigerlich zur Notwendigkeit hochgradig automatisierter Bildvberarbeitungsmethoden.

1.1 Motivation

Die Ermittlung der Landnutzungs- und -bedeckungsklassifizierung zählt zu den wesentlichen Aufgaben der Fernerkundung, doch schon die nächste Generation der Fernerkundungssatelliten lässt neue Dimensionen in den Fernerkundungsdaten erahnen (Möller, 2011), aber auch – ohne eine weitere Verbesserung der Bildverarbeitungsmethoden – etwaige Engpässe in der Prozessierung der hochwertigen Daten befürchten.

Eine visuelle Bildinterpretation[1] (engl. *visual image interpretation*) der Fernerkundungsdaten (Satellitenbilder, Luftbilder) ist durchaus noch gängig, trotzdem zwingt das ständig steigende Datenvolumen und der Anspruch, die Bilddaten zeitnah und im Ergebnis reproduzierbar zu interpretieren, vermehrt zur einer effizienten und zumindest teilautomatisierten Verarbeitung der Bilder (Blaschke, 2010). Demzufolge verlangen auch die Benutzer zunehmend nach speziellen, automatisierbaren Algorithmen und Verfahren, um aus den qualitativ hochwertigen und umfangreichen Rohbilddaten die für die jeweiligen Fragestellungen wertvollen Informationen extrahieren und ableiten zu können. Die vorliegende Arbeit beschäftigt sich eingehend mit einem Teilproblem der Bildverarbeitung und -analyse – der automatischen und effizienten Klassifikation von Bildobjekten.

[1] Unter einer visuellen Bildinterpretation versteht man die Bildinterpretation durch einen menschlichen Beobachter, die sowohl das Erkennen von Objekten als auch die Interpretation der Zusammenhänge und der daraus abgeleiteten Schlussfolgerungen einschließt (Baldenhofer, 2011).

1.2 Problemstellung

Die in vielen Bereichen herausfordernde Aufgabe einer Klassifizierung besteht darin, die einzelnen Bildpunkte oder Objekte eines segmentierten Bildes auf Basis ihrer charakteristischen Eigenschaften (Merkmale) bestimmten Klassen zuzuordnen, selbst wenn mit den im Bild enthaltenen Spektralinformationen nicht automatisch zweifelsfrei eine Klassenzuordnung möglich ist. Eine Erhöhung des Informationsgehalts durch beispielsweise die Ableitung zusätzlicher, voneinander möglichst unabhängiger Merkmale aus den Bildsegmenten und der Relationen der Bildsegmente zueinander macht eine Unterscheidung der Segmente oft überhaupt erst möglich, erhöht aber auch deutlich den algorithmischen Aufwand und damit die Komplexität der Klassifikatoren.

Erschwerend kommt hinzu, dass die für Klassifikatoren wesentlichen Zusammenhänge einzelner Merkmale oder Merkmalskombinationen mit der Klassifikation selten linearer Natur sind. Oft ergeben sich durchaus komplexe Abhängigkeiten, die in mathematischen Modellen oder umfangreichen Regelwerken nachgebildet werden müssen. Zudem nimmt bei vielen Klassifikatoren auch die Extraktion und die Anzahl der Merkmale eine zentrale Bedeutung ein (Guyon et al., 2006).

Die Problemstellung aus dem jeweiligen Anwendungsbereich bestimmen die Anforderungen an den Klassifikator. Wesentliche Anforderungskriterien – speziell aus der Sicht der Fernerkundung – sind (a) ein stabiler Prozessablauf, (b) eine hohe Anpassungsfähigkeit des Klassifikators an (c) stark variierende und komplexe Merkmalsverteilungen, eine möglicherweise (d) hohe Anzahl unterschiedlicher Klassen und eine mögliche (e) Unterrepräsentation bestimmter Klassen in den Lernmustern, die (f) Nutzung optional verfügbaren Kontext-Wissens, (g) die Parametrisierung des Klassifikators und (h) die Prozessierungsdauer (Duda et al., 2000; Lucht et al., 2009; Niemann, 2003).

In Abhängigkeit der Klassen- und Merkmalsanzahl und der Merkmalstypen decken sowohl stochastische als auch deterministische Klassifikatoren die Anforderungen mehr oder weniger erfolgreich ab. Welche Methoden auch immer zum Einsatz kommen, alle haben sie gemeinsam, dass obige Anforderungen schnell zu komplexen, in praktischer Anwendung schwer bis nicht handhabbaren Lösungsstrategien führen, die kaum noch zu automatisieren sind.

1.3 Lösungsansatz

Eine *Support Vector Machine* (SVM) ist ein mathematisches Verfahren der Mustererkennung. Sie gehört zu den lernenden Klassifikatoren, mit denen unbekannte Daten auf Basis von zuvor aus Trainingsdaten gelernter Regeln klassifiziert werden können.

Mit den Algorithmen der SVMs, die sich aus der *Statistischen Lerntheorie* (Vapnik & Chervonenkis, 1974; Vapnik, 1998) herleiten, können sowohl linear trennbare, als auch - durch eine Erweiterung der SVMs mit Kernel-Methoden - nicht-linear trennbare Daten im mehrdimensionalen Merkmalsraum separiert werden. Ohne statistische Annahmen bezüglich der Verteilung der Daten versucht die SVM eine lineare Trennebene (*Hyperebene*) im mehrdimensionalen Merkmalsraum so durch die Datenpunkte der Trainingsdaten zu legen, dass die Daten der einzelnen Klassen weitgehend fehlerfrei voneinander getrennt bzw. unklassifizierte Daten korrekt Klassen zugeordnet werden können.

Mit steigender Anzahl der verwendeten Merkmale nimmt auch die Dimension der Merkmalsmatrix zu. Segmentierte Fernerkundungsbilder führen so zu hochdimensionalen Merkmalsräumen und schwer trennbaren Daten bei vergleichsweise geringem Trainingsdatenaufkommen. Gerade solche Bedingungen bereiten konkurrierenden Klassifikatoren zunehmend Probleme, wohingegen Support Vector Machines im Allgemeinen gut für derartige Problemstellungen geeignet sind (vgl. Kapitel 5.4 bis 5.6).

Ein Vorteil von SVMs besteht darin, dass eine vergleichsweise geringe Anzahl von aus den Datenpunkten abgeleiteten Stützvektoren (engl. *Support Vectors*) reicht, um die Lage der Hyperebene im mehrdimensionalen Raum festzulegen. Umgekehrt verhindert der hochoptimierte Algorithmus zur Suche der Hyperebene, dass eine zu hohe Anzahl von Trainingspunkten zum gefürchteten Übertraining des Klassifikators führt, bei dem die Generalisierung soweit abnimmt, dass der Algorithmus die Trainingsdaten exakt nachbildet und später bei zu klassifizierenden und von den Trainingsdaten abweichenden Daten versagt.

Lassen sich hochdimensionale, komplexe Eingangsdaten im Eingangsdatenraum nicht linear trennen, so wird die Trennebene in einem höherdimensionalen Merkmalsraum gesucht. Damit eine aufwendige Transformation der Daten in diesem höherdimensionalen Raum ausbleiben kann, bedient man sich eines sog. *Kernel-Tricks* (Burges, 1998). Dabei erfolgt die Hin- und Rücktransformation implizit mit einer sogenannten Kernelfunktion, die als Repräsentant der Hyperebene im höherdimensionalen Merkmalsraum betrachtet werden kann (Heinert, 2010a). Dieser mathematische Kunstgriff steigert nicht nur die Generalisierungsleistung der SVM, sondern auch die Klassifizierungsgeschwindigkeit.

Obwohl SVMs aufgrund ihrer Eigenschaften vorweg als prädestiniert für Fernerkundungsklassifizierungen gelten mögen, bleiben trotzdem eine Reihe von Fragen offen, deren Beantwortung die Voraussetzung der erfolgreichen praktischen Nutzung von SVMs und Ziel dieser Arbeit ist.

1.4 Zielsetzungen

Support Vector Machines werden im Vergleich zu anderen Algorithmen in der Fernerkundung noch vergleichsweise selten eingesetzt (Heinert, 2010a). Spezielle Computerprogramme, wie beispielsweise das in dieser Arbeit verwendete *eCognition*[2], bieten zwar eine Reihe von Methoden zur Klassifikation von Bildobjekten, SMVs sehen sie aber oft nicht oder nur rudimentär vor. Die grundsätzliche Vorgehensweise bleibt dabei meist dem Benutzer überlassen. Viele Benutzer folgen einer heuristischen Trial-and-Error-Methode und nutzen die Erfahrungen aus vorangegangenen Klassifizierungen. „Vor allem die Auswahl geeigneter Merkmale, eine nachvollziehbare Gewichtung der Parameter [...] sowie die Behandlung eventuell widersprüchlicher Merkmalsausprägungen stellen noch ungelöste Probleme dar“, wie De Lange (2006) festhält.

Um eine Empfehlung für oder gegen SVMs zur Landnutzungs- und Landbedeckungsklassifikationen abgeben zu können und um zu klären, ob und unter welchen Bedingungen sie in der Praxis den Anforderungen genügen, sind folgende Fragen zu beantworten:

- **Merkmalsauswahl**: Wie muss die Auswahl der Merkmale für SVMs erfolgen, um eine optimale Klassifikationsleistung zu erreichen?
- **Kernel-Auswahl**: Wie erfolgt die Auswahl des Kernels bei nicht linear trennbaren Mustern?
- **Training**: Was muss sowohl beim Training selbst als auch bei der Auswahl der Trainingsdaten beachtet werden? Wie kann eine Generalisierung gewährleistet werden? Wann kommt es zu einem Übertraining (engl. *overfitting*) oder auch zu einem Untertraining (engl. *underfitting*) und was sind die Auswirkungen davon?
- **Parametrisierung**: Wie werden Support Vector Machines parametrisiert, um eine optimale Klassifikationsleistung zu erreichen?
- **Automatisierung**: Ist Spezialwissen notwendig, um SVMs sinnvoll als Klassifikationswerkzeug einsetzen zu können? Könnte die Parametrisierung oder das Training automatisiert werden?
- **Evaluierung**: Wie kann auf Basis der Trainingsdaten eine Bewertung oder Schätzung der Klassifizierungsleistung erfolgen?

Das Ziel aller Überlegungen ist einerseits eine performante Klassifizierungsmethode auf Basis von Support Vector Machines zu entwickeln, andererseits eine gute Klassifizierungsleistung unter der Prämisse einer einfachen Benutzbarkeit zu erreichen. Denn unabhängig von den verwendeten Algorithmen wird ein Klassifikator nur dann eine entsprechende

[2] eCognition Software: http://www.ecognition.com/ (Zugriff: 2011-06-01)

Akzeptanz beim Benutzer finden, wenn er einfach zu verwenden ist und reproduzierbar akzeptable und stabile Ergebnisse liefert.

Ausdrückliches Nicht-Ziel der vorliegenden Arbeit ist die Abhandlung einer allgemeinen Vorgangsweise zur Klassifikation von Fernerkundungsbildern. Der Fokus liegt uneingeschränkt bei Support Vector Machines, deren Eigenschaften und deren Anwendung.

Als Zielgruppe der Lösungen und Antworten auf obige Fragen werden GeowissenschafterInnen angenommen, die nicht zwingend alle mathematischen Details von Support Vector Machines kennen und verstehen müssen, um sie trotzdem erfolgreich anzuwenden.

1.5 Roadmap und Struktur der Thesis

Abbildung 1.1 visualisiert die Roadmap der Arbeit.

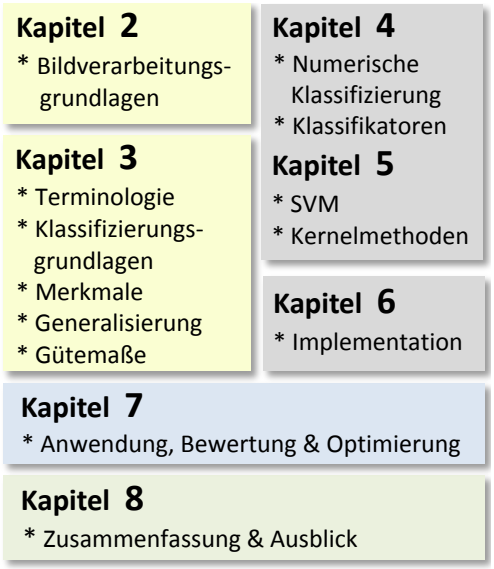


Abbildung 1.1: Roadmap der Thesis

Nach der Einleitung in Kapitel 1 führt Kapitel 2 in die allgemeinen Grundlagen der Bildverarbeitung und die objektbasierte Bildanalyse ein.

Kapitel 3 erklärt die automatische Klassifizierung mittels maschinellen Lernens, diskutiert die Darstellung, Selektion und Extraktion von Merkmalen und beschreibt die Bewertung der Klassifikationsgüte. Leser mit entsprechendem Grundlagenwissen können diese beiden Kapitel auch überspringen.

Die beiden folgenden Kapitel bilden den mathematisch dominierten Teil dieser Arbeit. Kapitel 4 definiert die Notation, führt in die Grundlagen numerischer Klassifizierung ein und gibt einen kurzen Überblick über gängige Klassifizierungsmethoden. Kapitel 5 setzt sich ausschließlich mit Support Vector Machines auseinander und zeigt die mathematischen Zusammenhänge von Linear-SVMs bis zu Kernel-SVMs für nicht-linear separierbare Muster.

Kapitel 6 skizziert die Implementierungen und die Testumgebung. Kapitel 7 zeigt abschließend die praktische Anwendung von Support Vector Machines und diskutiert die Klassifikationsergebnisse. Der zweite Teils dieses Kapitels legt den Schwerpunkt auf die Optimierung des Klassifizierungsprozesses mit SVM-basierten Klassifikatoren.

Kapitel 8 schließt die Arbeit mit einer Zusammenfassung der Erkenntnisse und einem Ausblick auf eine mögliche Fortführung dieser Arbeit ab.

2 Grundlagen der Bildverarbeitung

Im folgenden Kapitel wird der Leser in die allgemeinen Grundlagen der digitalen Bildverarbeitung und der Bildanalyse mit Blick auf Anwendungen in der Fernerkundung eingeführt. Diese Einführung erhebt nicht den Anspruch, alle Aspekte der digitalen Bildverarbeitung und Fernerkundung zu erläutern, sondern soll die Terminologie und den Sachverhalt nur soweit klären, wie es dem allgemeinen Verständnis der später diskutierten Bildanalyse und Klassifikation mittels Support Vector Machines dienlich ist.

Der Fokus der folgenden Einführung liegt auf 2D-Bildern, für die zur Evaluierung der SVM im Kapitel 7 auch die Testdaten vorliegen. Dies stellt keinerlei Einschränkungen für die folgenden Betrachtungen dar, denn Höheninformationen und daraus abgeleitete Informationen stellen nur einige von vielen zur Klassifizierung verfügbaren Merkmalen dar, wie sich im Kapitel 3.3 zeigt. Die in den Kapiteln 2 und 3 eingeführten Mustererkennungsprinzipien gelten sowohl für 2D- als auch für 3D-Bilder.

2.1 Digitale Bildverarbeitung

Unter digitaler Bildverarbeitung (engl. *digital image processing*, manchmal *computer vision*) versteht man im Allgemeinen die rechnergestützte Verarbeitung digitaler Bilder.

Typische Anwendungsgebiete der digitalen Bildverarbeitung finden sich beispielsweise in bildgebenden Verfahren der Medizin, Biologie und Mikroskopie, bildbasierten Überwachungs- und Kontrollsystemen, Texterkennungs- und Robotiksystemen sowie in Fernerkundungssystemen (Hermes, 2005; Gevantmakher & Meinel, 2004).

Im Kontrast zur Bildbearbeitung verfolgt die Bildverarbeitung das vorrangige Ziel, Bilder auszuwerten, visuelle Informationen aus den digitalen Abbildungen zu gewinnen und Bildinhalte zu interpretieren (Hermes, 2005). Um Informationen aus den digitalen Bildern zu extrahieren, werden die Bilder gezielt manipuliert und vermessen, nach unterschiedlichsten Kriterien untersucht oder in andere Abbildungsformate transformiert.

Wie der folgende Abschnitt erläutert, entscheiden die den Analysen zugrundeliegenden Daten und die erwartete Qualität der Bildinterpretationsergebnisse, ob Kontextwissen zur Verarbeitung und Interpretation notwendig ist oder nicht.

2.2 Bildverarbeitungsstufen

Die Literatur unterscheidet zwei Bildverarbeitungsstufen: eine Verarbeitung auf niedriger und eine Verarbeitung auf höherer Stufe. Diese werden wie folgt beschrieben und voneinander abgegrenzt (Hermes, 2005; Uhl, 2005):

- **Low-level Image Processing** oder *Low-level Vision* bezeichnet einfache und lokale Methoden, die keinerlei Kenntnisse über den Inhalt des Bildes notwendig machen. Typischerweise handelt es sich dabei um Algorithmen, welche die Bilder ausschließlich auf Basis der den Bildpunkten zugeordneten Werten in einzelne Strukturen unterteilen (Segmentierung) und diese möglichst sinnvoll nach ausgewählten Merkmalen mit Hilfe von Klassifikationsverfahren auf Klassen verteilen. Die Kanten- und Regionenfindung können hierbei als Beispiele dienen.
- **High-level Image Processing** oder *High-level Vision* bezeichnet Methoden, die ein Zusatzwissen voraussetzen, um ein Bild korrekt interpretieren zu können. Oft werden diese Methoden mit künstlicher Intelligenz in Zusammenhang gebracht. Grundlage der Bildinterpretation ist die Zuordnung von Bildbestandteilen zu realen Objekteinheiten auf Basis von Kontextwissen.

Die in dieser Arbeit diskutierten Algorithmen und Klassifikatoren werden vorweg der *Low-level Vision* zugeordnet. Die Segmentierung und Klassifikation erfolgt dabei größtenteils auf Basis der Bilddaten und es wird kein oder über bestimmte Merkmale nur begrenzt Kontextwissen miteinbezogen. Eine Klassifikation mit Hilfe von Kontextwissen würde andere als die verwendeten Bildmerkmale und besprochenen Algorithmen erfordern, wenngleich die Aufbereitung und Auswahl der Trainingsdaten und deren Unterscheidungsmerkmale durchaus unter Einflussnahme von Kontextwissen erfolgen können, wie in den späteren Kapiteln noch deutlich wird (vgl. Kapitel 3.3).

2.3 Bildverarbeitungsablauf

Der prinzipielle Verarbeitungsaufbau in einem Low-level Bildverarbeitungs- und Analysesystem ist stets ähnlich und der Bildverarbeitungsprozess durchläuft stets ähnliche Verarbeitungsoperationen. Grob kann der Ablauf, wie er auch in Abbildung 2.1 skizziert ist, in fünf Prozessstufen unterteilt werden (De Lange, 2006; Baldenhofer, 2011; Handels, 2000; Hermes, 2005; Jähne, 2005):

1. **Bildgewinnung**: Digitalisierung bzw. Erfassung und Speicherung digitaler 2D-/3D-Bildaufnahmen mittels Digitalkameras, Sensoren und Scanner. In der Geoinformatik handelt es sich vorwiegend um eine Primärdatenaufnahme durch die Vermessung

der von der Erdoberfläche ausgehenden elektromagnetischen Strahlung mittels Sensoren. Sekundäre Aufnahmesysteme, mit denen analoge Luftbilder mit Scannern digitalisiert werden, verlieren zunehmend an Bedeutung.

2. **Bildvorverarbeitung**: Normierung, Restaurierung und Rekonstruktion des digitalen Bildes, Anwendung von Filtern zur Rauschunterdrückung, Hervorhebung bestimmter Muster und Merkmale, Entzerrung, Kontrastspreizung, Farbsättigung und Transformation des Farbraums, Kantenschärfung und -detektion, Kalibrierung und Überlagerung mehrerer Bilder (Registrierung).

3. **Segmentierung**: Unterteilung des Bildes in inhaltlich zusammenhängende Regionen durch die Identifikation und Zusammenfassung benachbarter und homogener Bildpunkte; beispielsweise gleiche oder ähnliche Farbe oder Textur (siehe Seite 11 ff, Abschnitt 2.6)

4. **Merkmalsextraktion**: Extraktion einfacher Strukturen und wesentlicher Merkmale, wie beispielsweise der Form, Geometrie und Morphologie, zur späteren Unterscheidung der Segmente (siehe Seite 24, Abschnitt 3.3.6). Die aus den signifikanten Merkmalen gebildeten Merkmalsvektoren bilden die Eingangsdaten für die nachfolgende Bildklassifikation.

5. **Bildklassifikation**: Einordnung von Segmenten und Bildobjekten in Klassen auf Basis bekannter oder berechneter Merkmale (siehe Seite 30, Abschnitt 3.4). Voraussetzung ist die Extraktion der wesentlichen Merkmale zur Unterscheidung der einzelnen Klassen.

Das Diagramm in Abbildung 2.1 auf Seite 9 visualisiert den prinzipiellen Prozessablauf in einem Low-Lewel Bildverarbeitungs- und Analysesystem. Am Beginn des Prozesses steht die Gewinnung der Bilddaten. In der Abbildung werden die Bilddaten - unabhängig des Bildformats bzw. Prozessierungsstandes - über alle Prozessstufen jeweils mit Ellipsen symbolisiert. Mittels Operationen werden die Bilddaten aufgewertet oder in eine weitere Abbildungsebene transformiert. Die einzelnen Operationen sind optional und werden in der Abbildung jeweils in Form von Rechtecken dargestellt. Die Pfeile weisen auf die Ablauffolge der einzelnen Operationen hin. Die punktierten Linien symbolisieren optionale Iterationsschritte innerhalb des Ablaufs. Am Ende des Prozessablaufs steht eine pixel- oder wie in dieser Arbeit eine objektbezogene Klassifizierung, auf die auch der Fokus der folgenden Ausführungen gerichtet ist.

Quellen wie Gevantmakher & Meinel (2004); Handels (2000) und Lehmann et al. (2005) zählen auch die Bilddarstellung und Bildspeicherung zur Bildverarbeitung. Diese beiden Prozessschritte bleiben in obiger Aufzählung und in der schematischen Darstellung des prinzipiellen Ablaufs der Abbildung 2.1 unberücksichtigt und werden in dieser Arbeit auch nicht ausdrücklich behandelt.

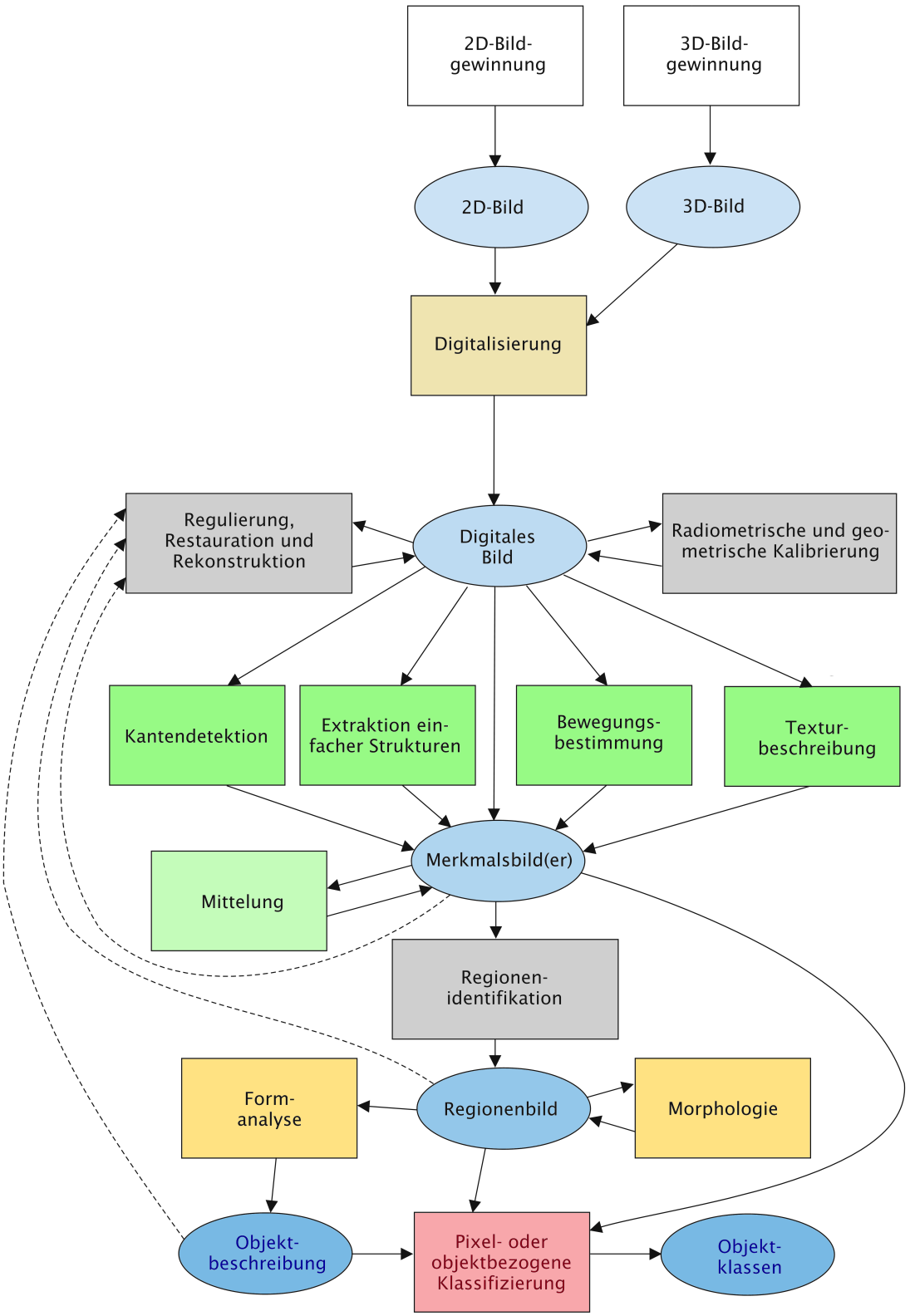


Abbildung 2.1: Prinzipieller Ablauf in einem Low-Level Bildverarbeitungs- und Analysesystem. Adaptiert aus JÄHNE (2005)

2.4 Bildgewinnung

In der Fernerkundung (engl. *remote sensing*) werden die Bildinformationen ausschließlich über eine berührungsfreie Vermessung und Interpretation der von der Erdoberfläche reflektierten oder emittierten elektromagnetischen Strahlung gewonnen (DIN18716-3, 1997). Die Ausgangssituation für eine spätere Verarbeitung und Bildklassifikation ist ein zur Erde gesendetes, multispektrales, hochauflösendes und geokodiertes Bild, bestehend aus einzelnen Bildpunkten (*Pixel*).

Generiert wird das Bildmaterial aus den Messdaten von *Sensorsegmenten* (eine Plattform mit in Kameras, Scannern oder Radars verbauten Fernerkundungssensoren), die von Flugzeugen oder Satelliten, manchmal auch Ballons oder Drohnen über die zu vermessenden Landstriche geflogen werden. Die Bedeutung und der Einsatz gängiger Fernerkundungsinstrumente richtet sich nach deren technischer Leistungsfähigkeit. Die in den Sensorsegmenten eingebauten Aufnahmesysteme werden nach vier Kriterien beurteilt (De Lange, 2006; Baldenhofer, 2011):

- Die *räumliche Auflösung* ist ein Maß, das die kleinste identifizierbare Fläche auf einer Fernerkundungsaufnahme angibt. Häufig wird sie mit den geometrischen Eigenschaften des Aufnahmesystems, die letztlich die Größe eines Bildpunktes ergeben, gleichgesetzt.

- Die *spektrale Auflösung* wird durch die Anzahl, die Breite und die Lage der Wellenlängenbereiche (Kanäle) bestimmt, mit der ein Aufnahmesystem das elektromagnetische Spektrum abtastet. Panchromatische Bilder werden in einem bestimmten Bereich meist breitbandig gemessen. Multichromatische Aufnahmesysteme zeichnen die Strahlung der Erdoberfläche gleichzeitig in mehreren Spektralbereichen auf.

- Die *radiometrische Auflösung* wird durch die Abstufung der einzelnen Signale bestimmt. Eine Abstufung von beispielsweise 256 Grauwerten erfordert eine radiometrische Auflösung von 8 Bit ($2^8 = 256$). Üblich sind 8 oder 12 Bit.

- Die *temporale Auflösung* beschreibt die zeitlichen Abstände von einer Aufnahme bis zur nächsten Aufnahme eines bestimmten Gebiets.

Obwohl ein Großteil der auf die Erde einfallenden elektromagnetischen Strahlung absorbiert und nur der kleinere Teil der Strahlung reflektiert wird, wird in erster Linie die reflektierte, zudem aber auch die emittierte Strahlung gemessen. Die Messung der Reflexionen und Emissionen mit einem *passiven Sensor*, vor allem mit der Sonne als Strahlungsquelle, gestaltet sich technisch vergleichsweise einfach. Messungen außerhalb des Spektrums der Sonne sind – auch durch den Einsatz von *aktiven Sensoren* – aufwendiger. Sie benötigen eigene Strahlungsquellen (Liu & Mason, 2009).

2.5 Bildvorverarbeitung

Auch wenn das Bild in hochauflösender Form vorliegt, so sind noch eine Reihe weiterer Verarbeitungsschritte notwendig, um aus den digitalen Bilddaten für die jeweilige Applikation relevante Informationen zu gewinnen. Die ersten Schritte nach der Bildaufzeichnung sind oft Bildvorverarbeitungsoperationen zur Bildkorrektur oder -verbesserung.

Die Ausführung von Operationen zur Bildkorrektur sind Bildmodifikationsschritte mit dem Ziel, das Bild für spätere Prozessschritte optimal vorzubereiten, ohne den Bildinformationsgehalt selbst zu ändern (Hermes, 2005). Im Wesentlichen kommen bei einer Bildvorverarbeitung (engl. *image preprocessing*) Algorithmen aus der Signalverarbeitung zur Anwendung. Eingesetzt werden Verfahren zur Verstärkung oder Schwächung der Signalwerte, Filter zur Rauschunterdrückung, zum Entfernen von Bildstörungen oder zur Entzerrung der Bildinformationen, sowie Algorithmen zur geometrischen Entzerrung, zur Normierung oder zur Farb- und Grauwertkorrektur. Manchmal wird auch eine Transformation des Farbraums, eine Kantenschärfung und -detektion, eine Kalibrierung und eine Überlagerung mehrerer Bilder durchgeführt, um die Qualität des Bildes weiter zu verbessern und um eine Normierung, Restaurierung und Rekonstruktion des digitalen Bildes zu erreichen (Jähne, 2005; Niemann, 2003). Letztlich sollen bestimmte Muster und Merkmale hervorgehoben werden, um die darauffolgende Segmentierung und Klassifizierung zu erleichtern.

2.6 Segmentierung

Gerade der Fortschritt in der Bildaufnahme verkompliziert weitere Analysen, da das Auffinden homogener Bildelemente durch die steigende Auflösung erschwert wird und die Datenmenge zudem beträchtlich steigt. Gebäude und Waldflächen werden nicht mehr als zusammenhängende Flächen erkannt, sondern als einzelne, separate Bildelemente abgebildet (De Lange, 2006). Bestimmte Verfahren stützen sich deshalb nicht auf die einzelnen Pixel der Abbildung, sondern auf Regionen (in diesem Zusammenhang oft auch als Segmente oder Objekte[1] bezeichnet), die durch eine Gruppierung benachbarter Bildpunkte mit gleichen oder ähnlichen Merkmalen gebildet werden.

[1] Genau genommen ist zu diesem Zeitpunkt die Verwendung des Ausdrucks *Objekt* noch nicht korrekt, denn ein Bildelement ist erst dann ein Objekt, wenn es explizit als solches klassifiziert wurde (Jähne, 2005). Aufgrund des allgemeinen Sprachgebrauchs wurde in dieser Arbeit auch schon zuvor der Begriff *Objekt* verwendet, ohne auf diese exakte Unterscheidung hinzuweisen. Zudem kann in der Fernerkundung in der Regel jedes Segment klassifiziert werden, da es keinen klassischen Bildhintergrund gibt, sondern jedes Bildsegment immer ein Abbild einer konkreten Einheit auf der Erde darstellt, vergleiche auch auch *Geoobjekt* (engl. *spatial object*) in Rostock (2011).

Eine als *Segmentierung* bezeichnete Zusammenfassung löst sich von der pixelbasierten Sichtweise. Sie stellt eine Art Klassifikation auf Pixelebene dar, bei der die zu untersuchenden Bildelemente auf Pixelbasis über geeignete Merkmale zu inhaltlich zusammenhängenden, überdeckungsfreien, homogenen Bildregionen vereint werden (Hermes, 2005; Tönnies, 2005). Grundlage dieser Zusammenfassung ist die Annahme, dass ein betrachtetes Pixel zur gleichen Klasse wie sein Nachbarpixel gehört (Blaschke, 2000).

Bei der Segmentierung von Bildern kommen pixel-, kanten-, bereichs- oder modellbasierte Segmentierungsstrategien oder die Kombination mehrerer Verfahren zum Einsatz, die von Handels (2000), Hermes (2005), Jähne (2005), Lehman et al. (1997) und Navulur (2006) beschrieben und wie folgt zusammengefasst werden können:

- **Pixelbasierte Verfahren** werden vor allem bei multispektralen Bilddaten eingesetzt. Sie verwenden die Farbe oder die Grauwerte der einzelnen Pixel. Lokale Nachbarschaften werden vorerst nicht berücksichtigt. Zur Bildung eines Segments werden die einzelnen Merkmale (Grau- oder Farbwerte) der Bildpunkte mit einem oder mehreren zuvor festgelegten Schwellwerten verglichen. Die Zuordnung der einzelnen Bildpunkte erfolgt auf aufgrund eines Über- oder Unterschreitens der Merkmalswerte. Problematisch ist dabei, wenn das pixelbasierte Bild keine homogenen Flächen aufweist.

- **Kantenbasierte Verfahren** detektieren starke lokale Veränderungen ausgesuchter Bildmerkmale (Farbwerte, Merkmalswerte). Bei multispektralen Bildern folgen sie den Kanten, die sich durch die Farb- oder Grauwertunterschiede benachbarter Bildpunkte ergeben. Wird eine Kante gefunden, so wird mit einem Konturverfolgungsalgorithmus die Kontur solange verfolgt, bis ein Ende der Kante oder wieder der Ausgangspunkt erreicht ist. Im Gegensatz zu pixelbasierten Verfahren ist kein Schwellwert notwendig. Das Verfahren macht keine Kalibrierung des Bildes erforderlich und funktioniert selbst dann, wenn das Bild insgesamt in seiner Helligkeit variiert. Lediglich der Farb- oder Grauwertunterschied (Kontrast) wird zur Erkennung der Kante herangezogen.

- **Bereichsbasierte Verfahren** analysieren die Homogenität bestimmter Merkmale zusammenhängender Bildbereiche. Wie beim pixelbasierten Verfahren werden dabei die Bildelemente auf Basis der Farb- oder Grauwerte segmentiert. Einander angrenzende Elemente mit ähnlichen Merkmalen werden zu einem Segment vereint.

- **Modellbasierte Verfahren** stützen sich nicht nur auf lokale Nachbarschaftsverhältnisse, sondern auf spezifisches Vorwissen. Ist beispielsweise die geometrische Form eines Segments vorab bekannt oder wurden typische Formeigenschaften aus bereits segmentierten Bildobjekten erhoben, so kann die Form auch als Orientierungshilfe bei der Kantensuche und -verfolgung dienlich sein, selbst wenn Störungen

auftreten und eine Kante unterbrochen wäre. Bei einer atlasbasierten Segmentierung, einer Erweiterung des modellbasierten Verfahrens, werden Bildobjekte mit bestimmten Formen an bestimmten Positionen erwartet und diese mit korrespondierenden Datensätzen verknüpft (Handels, 2000). Verfahren, die zusätzliche Informationen zur Generierung der erwarteten Segmente einbringen, sind zielgerichteter, jedoch auch anwendungs- und kontextabhängiger als andere Verfahren (Tönnies, 2005).

Abbildung 2.2 zeigt das Beispiel einer Segmentierung, bei der ein multispektrales, pixelbasiertes Bild (links) segmentiert wurde. Im segmentierten Bild (rechts) sind die homogenen Flächen bereits zu Regionen zusammengefasst. Die Bestimmung der Schwellenwerte beeinflusst im Wesentlichen die *Granularität* des segmentierten Bildes. Die Werte wurden im Beispiel so gewählt, dass große Segmente entstehen konnten, obwohl die Flächen oft eine deutlich sichtbare Inhomogenität in Form von Helligkeitsunterschieden aufweisen.

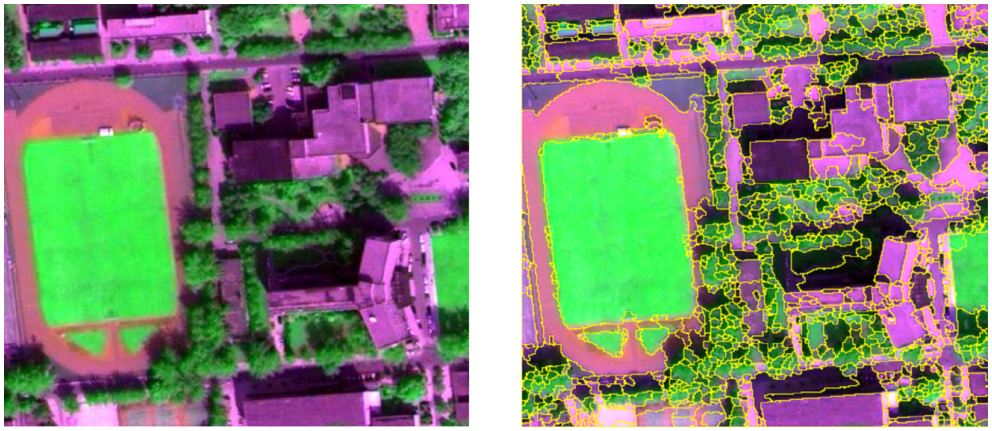


Abbildung 2.2: Falschfarbenbild (links) und das daraus erzeugte segmentierte Bild (rechts). Übernommen aus Peijun Li & Xiao (2008).

Mit der Segmentierung des pixelbasierten Bildes werden komplexe Bildstrukturen durch die Bildung homogener Regionen stark vereinfacht. Sie hat sich sowohl in der geoinformatischen und medizinischen Bildverarbeitung als auch im Bereich der Videoverarbeitung und -übertragung etabliert. In vielen Fällen ermöglicht sie eine schnelle Verarbeitung der Bilder und eine effiziente Kompression der Datenströme (Navulur, 2006).

Sowohl bei der Segmentierung als auch bei der Klassifizierung nehmen Merkmale eine besondere Bedeutung ein (vgl. Kapitel 3.3). Die zur Segmentierung verwendeten Merkmale können, müssen aber nicht gleich den zur Klassifizierung verwendeten Merkmalen sein. Eine Segmentierung ist eine Form von Informationstransformation, bei der die Anzahl der Bildelemente zwar verringert, die Anzahl der die Elemente beschreibenden Merkmale jedoch erhöht wird. Sowohl Pixelmerkmale als auch zusätzliche Merkmale, wie deren Formen oder deren Beziehungen zueinander, liefern die Eingangsdaten der unmittelbar auf die Segmentierung folgende objektbasierten Bildanalyse (Jähne, 2005).

2.7 Objektbasierte Bildanalyse

Ein zentrales Anliegen der Bildverarbeitung ist die am Ende eines Bildverarbeitungsprozesses stattfindende Bildanalyse (engl. *image analysis*). Sie ist ein Teilgebiet der Mustererkennung (engl. *pattern recognition*) und bedeutet in der digitalen Bildverarbeitung vielfach einen nicht unerheblichen algorithmischen Aufwand. Häufig wird aus der Anwendung abgeleitet der Begriff Bildanalyse mit der Klassifizierung gleichgesetzt, oft ist eine Klassifizierung aber auch die Voraussetzung einer erweiterten Analyse. Pixel und/oder Bildobjekte und deren Merkmale bilden die Datengrundlage dazu.

Die Literatur (Jähne, 2005; Navulur, 2006) unterscheidet zwei grundsätzliche Typen von Bildanalyseverfahren: die *pixelbasierten* und die *objektbasierten Bildanalysen*:

1. **Pixelbasierte Bildanalysen**: Bildanalysen auf Basis von Pixel unabhängig ihrer räumlichen Lage sind nicht immer zielführend, da die einzelnen Bildpunkte nicht oder kaum in Relation zueinander gesetzt werden und vergleichsweise wenige Merkmale (Farbe, Helligkeit, usw.) zur Unterscheidung bieten (Blaschke, 2000). Zu vermehrten Problemen führen vor allem sogenannte Mischpixel, die vor allem in den Übergangsbereichen verschiedener Oberflächentypen entstehen und aufgrund ihrer spektralen Merkmale keine eindeutige Zuordnung zulassen (De Lange, 2006).

   Gerade in der medizinischen Bildverarbeitung und in der Fernerkundung mit den multispektralen, hochauflösenden Bildern hat sich gezeigt, dass eine Klassifikation auf Basis der individuellen, hochauflösenden Pixel im mehrdimensionalen Spektralraum nicht immer zu befriedigenden Ergebnissen führt. Die Sichtweise auf die Bildpunkte ist zu stark lokal begrenzt. Deshalb sind alternative Analyseansätze gefordert (Blaschke & Strobl, 2001), zu denen auch die objektbasierte Bildanalyse gezählt wird.

2. **Objektbasierte Bildanalysen**: Bildanalysen auf Basis von Objekten (Bildsegmenten) liefern oft bessere Analyseergebnisse als pixelbasierte Verfahren (Blaschke, 2010; Huang & Ni, 2008; Lehman et al., 1997). In den letzten Jahren hat sich deshalb bei der Verarbeitung geoinformatischer und auch medizinischer Bilddaten die objektbasierte Bildanalyse (engl. *object-based image analysis* – OBIA[2]) als Spezialisierung innerhalb der Bilderkennung immer mehr durchgesetzt (Blaschke, 2010; Handels, 2000).

   Bei dieser an die Segmentierung anschließenden Bildanalyse wird das Bild nicht im Pixelraum, sondern im Objektraum analysiert. Dabei werden nicht mehr die Pixel

[2]OBIA wird von einigen Autoren auch mit *object-oriented image analysis* bezeichnet (Navulur, 2006)

und deren Primärmerkmale als kleinste Einheit, sondern die Objekte als Primitive[3] und als Repräsentation homogener Regionen und deren Merkmale (Attribute) als Grundlage zur weiteren Bildklassifizierung herangezogen (Navulur, 2006). Mit Hilfe mathematischer Modelle und regelbasierter Systeme werden die Objekte und deren Merkmale untersucht und in Beziehung zueinander gesetzt. Dabei ist die Auswahl geeigneter Merkmale mitunter aufwendig und durchaus nicht trivial, denn die Merkmalszusammensetzung ist eine der wesentlichen Einflussgrößen auf den Erfolg oder Misserfolg einer Bildanalyse oder Klassifikation (siehe Kapitel 3.3, *Merkmale*). Die Perspektivenänderung vom Pixel zum Objekt und die Betrachtung einer Abbildung auf einer höheren Abstraktionsebene bieten eine Reihe von Vorteilen.

Der Vorteil der OBIA besteht einerseits darin, dass zusammengehörende und beieinanderliegende Pixel in einem Segmentierungsprozess zu Objekten vereint werden und dadurch die Anzahl der Primitive drastisch reduziert wird. Anderseits werden diese Objekte aber verglichen mit den Pixeln des Ausgangsbildes mit einer Reihe zusätzlicher, von den Segmenten abgeleiteten, nicht-spektraler Merkmale (*Sekundärmerkmale*) angereichert. Neben den spektralen Werten und deren abgeleiteten Größen wie beispielsweise dem Mittelwert, der Standardabweichung und der Varianz über die Grauwerte, kann deren Form, Textur, Morphologie und die Relation zu angrenzenden Objekten, die Orientierung oder der Grenzlinienverlauf für eine spätere Analyse verwendet werden. Die Dimension des Merkmalsraums wird damit drastisch erhöht. Mitunter können dadurch bessere Analyse- und Klassifikationsergebnisse erzielt werden, da die Anzahl der auswertbaren Unterscheidungsmerkmale zunimmt (Baatz et al., 2008; Handels, 2000).

Die objektbasierte Bildanalyse findet in der Materialforschung, Mikroskopie und gerade in der Fernerkundung und medizinischen Bildverarbeitung mittlerweile eine beachtliche Verbreitung, da in diesen Bereichen die Vorteile stark überwiegen. Neben den angeführten Quellen sei der Leser für weitere Literatur auf eine Sammlung wissenschaftlicher Artikel zu OBIA in Blaschke et al. (2008) verwiesen. Zudem zeichnet der Autor Blaschke (2010) in *Object based analysis for remote sensing* die Entwicklung und den aktuellen Stand der Forschung aus der Sicht der Fernerkundung und gibt einen umfangreichen Überblick über weitere einschlägige Literatur zu diesem Thema.

Die bisherigen Ausführungen haben in die Grundlagen der Bildverarbeitung, der Bildgewinnung, der Vorverarbeitung und als Vorbereitung in eine objektbasierte Bildanalyse in die Segmentierung eingeführt. Für das folgende Kapitel 3 sei als Ausgangssituation ein segmentiertes Bild angenommen, dessen Objekte auf Basis der Merkmale bestimmten Klassen zugeordnet werden sollen.

[3] Der Begriff *Primitiv* wird häufig zur Bezeichnung von elementaren geometrischen Figuren verwendet, aus denen sich dann komplexere Formen zusammensetzen lassen (Bungartz, 2002).

3 Einführung in die Klassifizierung

Am Beginn dieses Kapitels stehen die Einführung allgemeiner im Umfeld der Klassifikation verwendeter Fachbegriffe und eine Erläuterung der im Folgenden verwendeten Notation. In weiterer Folge wird die Problematik der Merkmalsauswahl diskutiert, bevor die automatische Bildklassifikation eingeführt und der grundsätzliche Klassifikationsablauf mit einer linearen Diskriminantenfunktion gezeigt wird.

Zu den in dieser Arbeit mehrfach referenzierten Standardwerken dieser Themen zählt das Buch *Pattern Classification* von Duda et al. (Duda et al., 2000). Neben Niemann (2003) sollte vor allem Guyon et al. (2006) zum Thema Merkmalsextraktion nicht unerwähnt bleiben. Weitere Quellen werden an entsprechenden Stellen angeführt.

3.1 Terminologie

Eine *Klassifikation* kann als eine systematische Ordnung von Objekten[1] in hierarchischen Systemen interpretiert werden, deren Objekte in den jeweiligen Klassen sich nach meist formalen Kriterien unterscheiden (Ferber, 2003). Eine Klassifikation ist das Ergebnis einer *Klassifizierung*[2], bei der die Objekte kategorisiert und zu *Klassen* zusammengefasst werden.

Die Basis der Klassenzuteilung bildet der Vergleich zwischen Klasseneigenschaften und den den Objekten zugeschriebenen oder berechneten *Merkmalen* (Attribute). Die Zuteilung von Objekten in bestehende Klassen (Objektklassen) übernimmt ein *Klassifikator*, oft auch als *Klassifizierer* (engl. *classifier*) bezeichnet. Dieser Prozess wird als *Klassifizierung* bezeichnet[3]. Welches *Klassifizierungsverfahren* (engl. *classification model*) dabei zur Anwendung kommt, hängt einerseits von den Daten, andererseits von der jeweiligen Anwendung und den daraus abgeleiteten Anforderungen an den Klassifikator ab (Handels, 2000; Hermes, 2005; Tönnies, 2005).

[1] In dieser Arbeit wird aus dem Kontext heraus mehrheitlich von Objekten und Segmenten geschrieben, wenngleich es sich in anderen Zusammenhängen auch um beliebige Datensätze handeln kann, die einer Klassifikation zugeführt werden.

[2] Das Studium einschlägiger, deutschsprachiger Fachliteratur belegt, dass zwischen den Begriffen *Klassifikation* und *Klassifizierung* oft nicht unterschieden wird. So werden auch in den in dieser Arbeit verwendeten und referenzierten Quellen die beiden Begriffe abwechselnd und gleichbedeutend verwendet. Dementsprechend wurde auch in vorangegangenen Teilen der Arbeit die Verwendung dieser Begrifflichkeiten aufgrund der besseren Lesbarkeit nicht immer exakt eingehalten.

[3] In einigen Fachgebieten wird die Klassifizierung auch als *Klassierung* bezeichnet.

3.2 Erläuterung der Notation

Der Schwerpunkt dieser Arbeit liegt mehr auf der Vermittlung der zugrunde liegenden Konzepte als auf mathematischer Strenge und Vollständigkeit. Trotzdem sind einige Definitionen und Notationsfestlegungen notwendig, um mathematische Zusammenhänge im Folgenden formal erklären zu können. Die Notation erfolgt in den meisten Fällen in Anlehnung an die jeweils referenzierte Literatur. Zum schnellen Nachschlagen sei auf die Kurzreferenz *Notation*, Seite xiii verwiesen. Abweichungen von den Festlegungen werden an entsprechender Stelle explizit vermerkt.

Wir notieren die Menge der natürlichen, reellen und komplexen Zahlen mit $\mathbb{N}$, $\mathbb{R}$ und $\mathbb{C}$. Alle anderen Mengen werden grundsätzlich mit kursiven lateinischen Großbuchstaben wie beispielsweise in $X = \{\mathbf{x}_1 \ldots, \mathbf{x}_m\}$ bezeichnet. Räume wie beispielsweise der Eingangsraum $\mathcal{X}$ werden mit kaligraphischen Buchstaben bezeichnet. Mehrdimensionale Räume wie bei $\mathcal{X} \subset \mathbb{R}^m$ werden mit einem hochgestellten Ganzzahl größer gleich 2 oder einer Variablen wie im Beispiel m ($m > 2$) notiert. Die Anzahl der Elemente einer endlichen Menge X wird mit $|X|$ beschrieben.

Vektoren $\mathbf{x}$ werden mit fett geschriebenen lateinischen Kleinbuchstaben bezeichnet und werden grundsätzlich als Spaltenvektor angenommen. Matrizen $\mathbf{M}$ werden mit fett geschriebenen lateinischen Großbuchstaben geschrieben. Ein hochgestelltes $^\top$ bezeichnet die Transponierte einer Matrix $\mathbf{M}^\top$ oder eines Vektors $\mathbf{x}^\top$. Indizes werden mit tiefgestellten ganzen Zahlen oder tiefgestellten Variablen geschrieben. Die Notation $(x_1, \ldots, x_m)$ bezeichnet einen Zeilenvektor mit m Elementen. Der korrespondierende Spaltenvektor wird mit einem angefügten $^\top$ gekennzeichnet, wie beispielsweise $(x_1, \ldots, x_m)^\top$. Die Notation $\mathbf{x}_i$ für $i = 1, \ldots, M$ bezeichnet den i-ten Spaltenvektor, wobei $i \in \mathbb{N}$ aufsteigend von 1 bis M läuft.

$\mathbf{w}^\top\mathbf{x}$ wird als das Skalarprodukt bzw. innere Produkt der beiden Vektoren $\mathbf{w}$ und $\mathbf{x}$ gelesen. Die euklidische Norm des Vektors $\mathbf{x}$ ist definiert als $\|\mathbf{x}\| := \sqrt{\mathbf{x}^\top\mathbf{x}}$. Funktionen $f(x)$ werden mit dem Funktionsnamen und in Klammer den Funktionsargumenten notiert. Die Funktionsnamen können beliebige Zeichen wie $\phi(x)$ oder Zeichenfolgen wie beispielsweise $\mathrm{sign}(e)$ sein. Die Transformation des Eingangsraums $\mathcal{X}$ in den Merkmalsraum $\mathcal{F}$ wird mit einem Pfeil wie beispielsweise in $\phi : \mathcal{X} \to \mathcal{F}$ symbolisiert.

3.3 Merkmale

In der Bildanalyse und -klassifikation erfolgt sowohl die Segmentierung, als auch die Unterscheidung der Objekte anhand von Objektmerkmalen. Sie stellen die eigentlichen zu klassifizierenden Daten dar. Merkmale können einzeln oder in Kombination (Bishop, 2006)

als Eingangsdaten für eine Klassifizierung verwendet werden. Jedes einzelne Merkmal eines Objekts kann zur Unterscheidung beitragen, in der Regel ist aber nicht jedes Merkmal für sich signifikant[4] genug und im gleichen Maße geeignet, Objekte klassenabhängig voneinander zu unterscheiden (Hermes, 2005).

3.3.1 Merkmalstypisierung

Eine Grundvoraussetzung zur automatischen Analyse und Klassifikation digitaler Bilder ist das Vorhandensein charakteristischer Merkmale zur Unterscheidung der Bildobjekte. Merkmale werden in Primär- und Sekundärmerkmale unterteilt (Handels, 2000).

- **Primärmerkmale** eines pixelbasierten Digitalbildes werden während der Bildaufnahme erfasst und entsprechen den Kanälen des Bildes.
- **Sekundärmerkmale** sind von den Primärmerkmalen abgeleitet und lassen sich beispielsweise über nicht-lineare Merkmalstransformationen errechnen.

Bildpunkte haben eine Reihe charakteristischer Primärmerkmale, wie beispielsweise die Farbe, Helligkeit und Geometrie. Wie in den Kapiteln 2.6 und 2.7 diskutiert, sind aber trotz multispektraler Aufnahmen in hoher Auflösung und nachgeschalteter Merkmalstransformationen die Analysemöglichkeiten anhand der spektralen Pixelmerkmale, im Besonderen ohne die Berücksichtigung der Lage der Bildpunkte und deren räumlicher Abhängigkeiten stark eingeschränkt. Durch heuristische Verfahren oder die Segmentierung eines Bildes kann aber mit geringem Aufwand eine größere Anzahl zusätzlicher Sekundärmerkmale erzeugt werden.

Vergleichsweise einfach zu berechnen sind beispielsweise geometrische, densitometrische, Textur- sowie Farbmerkmale. Aufwendiger in der Ermittlung sind sogenannte Kontextmerkmale, welche auch die Topologie des Bildes berücksichtigen. Eine Reihe weiterer durch Bildsegmente gewonnener Merkmale sind zum Beispiel Texturen, etwaige Nachbarschaftsbeziehungen, räumliche Distanzen zu anderen Segmenten, die Richtung der Hauptachsen oder eine Grenzlinie. Tabelle 3.1 zeigt eine mögliche Unterteilung der aus dem Bild und den Objekten extrahierten Merkmale.

3.3.2 Merkmalsdaten und Skalenniveaus

Die Rohdaten einer numerischen Klassifizierung sind in der Regel Primärmerkmale, die das Ergebnis einer Messung sind. Wie gut ein Merkmal gemessen und das Messergebnis

[4] Als *signifikant* (bezeichnend) bezeichnet man in der Statistik jene Größen, deren Wahrscheinlichkeit eines zufälligen Zustandekommens gering ist.

Unterteilung	Beschreibung	Beispiele
Geometrische Merkmale	Informationen über die geometrische Form bzw. Kontur eines Objekts	Länge, Breite, Umfang, Fläche, Orientierung, Krümmung, Schwerpunkt, Rundheit, Kompaktheit
Densitometrische Merkmale	Merkmale spektraler Art auf Basis der Grauwertverteilung; Reflexionsverhalten oder Absorptionsvermögen	Mittelwert, Streuung und Schiefe des Grauwerthistogramms
Statistische Merkmale	Informationen über die Eigenschaft einer statistischen Einheit	Anzahl der Pixel, Momente, Autokorrelation, Mittelwert, Standardabweichung, Varianz, Schiefe, Exzess, Korrelation, Kovarianz
Textur-basierte Merkmale	Information über die Eigenschaft der Oberflächenstruktur	Kontrast, farbliche Homogenität, Entropie, Energie
Farb-basierte Merkmale	Farbinformationen aus Basis unterschiedlicher Farbmodelle (z.B. RGB, CMY, HSV, u.a.)	Farbe, Sättigung, Helligkeit, Reinheit
Kontext-basierte Merkmale	Berücksichtigung der Topologie des Bildes	Nachbarschaftliche Beziehungen; Relationen der Objekte untereinander

Tabelle 3.1: Unterteilung extrahierter Merkmale

numerisch ausgedrückt werden kann, hängt von der jeweiligen Eigenschaft ab. So lässt sich beispielsweise die Fläche eines Grundstücks gut in Zahlen ausdrücken, wohingegen die Motivation ein Grundstück zu kaufen schwerer allgemeingültig in Zahlen auszudrücken ist. Die Art und Weise, wie eine Objekteigenschaft beschrieben wird, bestimmt das sogenannte Skalenniveau (Backhaus et al., 2011).

Im Allgemeinen unterscheidet man zwischen zwei grundsätzlichen Typen von Merkmalsausprägungen (Skalenniveaus): **metrische Merkmale**, die mittels reeller Zahlen beschrieben werden, und **nicht-metrische Merkmale**, die mittels Symbolen beschrieben werden (Backhaus et al., 2011; Niemann, 2003). Das Skalenniveau bestimmt, wie die Ausprägungen des Merkmals zu interpretieren sind, welche mathematischen Operationen möglich sind und ob und welche Transformationen mit den skalierten Merkmalswerten möglich sind. Tabelle 3.2 stellt die unterschiedlichen Skalen gegenüber.

Kommen numerische Klassifikatoren zur Anwendung, dann ist eine der wesentlichsten Voraussetzungen, dass die Merkmale in Zahlen ausgedrückt werden können. In dieser Arbeit werden vorwiegend reellwertige metrische Merkmale $(x_1, \ldots, x_m)$ behandelt, da diese die Datengrundlagen der in den späteren Kapiteln betrachteten numerischen Klassifikatoren bilden.

Skala		Definition	math.Op.	Beispiele
Metrisch	Ratio	Merkmalsausprägungen in Form von reellen Zahlen samt Dimension und Bezugspunkt	$=, \neq, >, <, +, -, \times, \div$, Summe	Fläche, Umfang, Helligkeit, Preis
	Interval	Merkmalsausprägungen in Form von reellen Zahl samt Dimension **ohne** Bezugspunkt	$=, \neq, >, <, +, -$, Mittelwert	Preisdifferenz, Zeitskala
Nicht-Metrisch	Ordinal	Qualitative Merkmalsausprägungen mit natürlicher Ordnung; Aufstellung einer Rangordnung möglich	$=, \neq, >, <$, Median, Quantile	*groß*/*mittel*/*klein*; 1/2/3/4/5
	Nominal	Qualitative Merkmalsausprägungen **ohne** natürliche Ordnung; Werte sind wie Namen zu verstehen	$=, \neq$, Häufigkeiten	Antworten vom Typ *ja*/*nein*; Nationalität; Enumeratoren

Tabelle 3.2: Typen (Skalenniveaus) von Merkmalen

Liegen nicht-metrische Merkmale vor, so können aus der Menge der Merkmalssymbole $(s_1, \ldots, s_m)$ sogenannte *Symbolketten* gebildet werden. Die Kodierung eines Merkmals mit einer Symbolkette ist der eines Merkmalsvektors ähnlich. Auch sie kann durch ihr Muster eine Klasse oder einen Datenpunkt eindeutig beschreiben. Ein wesentlicher Nachteil dieser Kodierung besteht jedoch darin, dass damit nicht oder nur eingeschränkt gerechnet werden kann und damit numerische Klassifikatoren für derartige Daten ungeeignet wären.

Symbolketten können in reellwertige Merkmalsvektoren umgeformt werden, indem die einzelnen Elemente der Symbolkette durch reelle Zahlen kodiert werden. Eine einfache und oft angewendete Möglichkeit besteht darin, einen binären Vektor aus 0 und 1 zu bilden, bei dem für jede Merkmalsausprägung eine eigene Spalte eingeführt wird, bei denen eine 1 als „Ausprägung vorhanden“ und eine 0 als „Ausprägung nicht vorhanden“ interpretiert wird. Eine weitere Möglichkeit besteht in einer numerischen Kodierung der Ausprägungen in Abhängigkeit der Häufigkeit in Bezug auf die Zielklasse. Beide Methoden haben den Nachteil, dass keine Abstände zwischen den Merkmalsausprägungen berechnet werden können. Obwohl dadurch auch numerische Klassifikatoren, welche reellwertige Merkmalsvektoren verlangen, verwendet werden können, ist diese Umformung in jedem Fall nur dann zielführend, wenn die Kompaktheit der Merkmale innerhalb einer Klasse gewährleistet wird (Niemann, 2003, S.20: Postulat 3). Die Vektoren gleicher Klassen sollen also auch nach dieser Umformung in einer eng begrenzten Region im Merkmalsraum abgebildet werden.

3.3.3 Merkmalsvektoren und Merkmalsräume

Als Ausgangspunkt der folgenden Überlegungen und Definitionen sei eine segmentierte Karte zur Darstellung der Landnutzung - wie in Abbildung 3.1 dargestellt - angenommen. Jedes Segment (Objekt) wird durch Merkmale beschrieben. Die farbig dargestellten Segmente der linken Abbildung sind jeweils einer bestimmten Klasse (Landnutzung) zugeordnet. Die weißen Flächen werden durch jene Segmente gebildet, die in diesem Beispiel unklassifiziert geblieben sind.

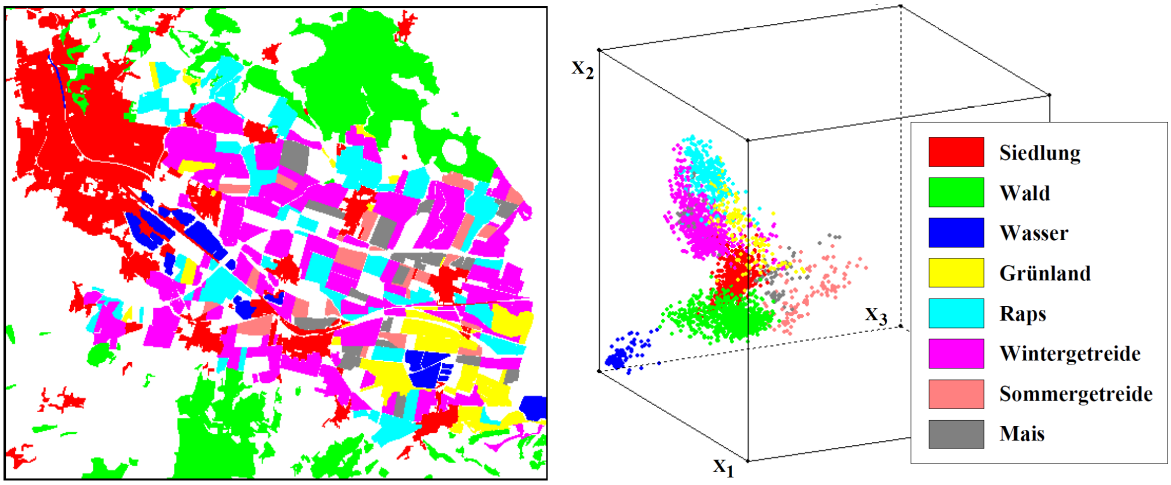


Abbildung 3.1: Links: Ausschnitt einer segmentierten Karte mit acht Landnutzungsklassen. Rechts: Verteilung der Merkmalsvektoren der Segmente im 3-dimensionalen Merkmalsraum (Lucht et al., 2009).

Die verallgemeinerte und mathematische Formulierung der Zusammenhänge der in Abbildung 3.1 dargestellten Karte lautet:
Sei $\mathcal{X}$ der Eingangsraum (Merkmalsraum) und $\mathcal{Y}$ der Ergebnisraum. Sei jedes der M Objekte durch m reellwertige Merkmale $x_1, \dots x_m$ beschrieben, so lässt sich jedes Objekt als Merkmalsvektor $\mathbf{x} = (x_1, \dots x_m)^\top$ für $\mathbf{x} \in \mathcal{X}$ interpretieren. Bei Vorliegen m-dimensionaler Daten gilt $\mathcal{X} \subset \mathbb{R}^m$. Sei zudem jedem Objekt bzw. jedem Vektor $\mathbf{x}$ ein skalarer Klassenlabel $y \in \{1, \dots, n\}$ für $y \in \mathcal{Y}$ und $\mathcal{Y} \subset \mathbb{R}$ für n mögliche Klassen zugewiesen, so wird das i-te Objekt für $i = 1, \dots, M$ durch das Datenpaar $(\mathbf{x}_i, y_i)$ definiert. Die Definition der Objektmenge D als Datengrundlage für die folgenden Betrachtungen lautet

$$D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_M, y_M)\} \subset (\mathcal{X} \times \mathcal{Y})^M \tag{3.1}$$

Die m reellwertigen Merkmale spannen einen m-dimensionalen euklidischen *Merkmalsraum* (engl. *feature space*) $\mathcal{X}$ auf. Damit lässt sich - wie in der rechten Abbildung 3.1

veranschaulicht - jedes Objekt durch seinen Merkmalsvektor $\mathbf{x}_i$ für $i = 1, \ldots, M$ an einer bestimmten Position im Merkmalsraum abbilden (Tönnies, 2005).

Wie durch die Einfärbung der Punkte in der rechten Abbildung 3.1 symbolisiert, bilden die Punkte gleicher Klassen im Merkmalsraum $\mathcal{X}$ im Idealfall kompakte oder mathematisch beschreibbare Punktwolken. Aus diesen Wolken können beispielsweise verallgemeinerte Klassenvektoren oder die Klassen separierende Grenzflächen zwischen den Punktwolken abgeleitet werden.

Werden unklassifizierte Vektoren im Merkmalsraum $\mathcal{X}$ abgebildet, so wird bei einer Klassifizierung die Klassenzugehörigkeit beispielsweise auf Basis der relativen Nähe der Vektoren zu den jeweiligen Punktwolken oder stellvertretend zum Klassenvektor, oder aus den Grenzflächen zwischen den Klassen ermittelt (Hermes, 2005; Jähne, 2005). Einen wesentlichen Einfluss auf die Komplexität einer Klassifizierung hat dabei die Dimensionalität des Merkmalsraums.

3.3.4 Dimensionalität des Merkmalsraums

Die Anzahl der zur Unterscheidung verwendeten Merkmale bestimmt die Dimensionalität $m = \dim(\mathcal{X})$ des Merkmalsraums $\mathcal{X}$. Eine hohe Anzahl von Merkmalen erleichtert die Unterscheidung der Objekte, da in der Regel der Abstand zwischen den Klassen mit steigender Dimension größer wird. Je höher die Dimensionalität eines Merkmalsraums aber ist, desto aufwendiger ist auch die Trennung der Objekte in diesem mehrdimensionalen Raum und desto komplexer ist die Parametrisierung des Klassifikators.

Es stellt sich also die Frage, wie viele und welche Merkmale konkret aus der gegebenen Merkmalsmenge ausgewählt werden sollen. Niemann (2003) fasst die sich widersprechenden Forderungen wie folgt zusammen:

> Eine „beste“ Untermenge von Merkmalen hat die Eigenschaft, dass es keine andere mit höchstens genau so vielen Merkmalen gibt, wobei die Merkmale dieser anderen Untermenge eine Klassifikation mit geringerer Fehlerwahrscheinlichkeit erlauben (Niemann, 2003, S. 246).

Wie ist diese Regel zu erklären? Ohne hierbei auf die konkrete Berechnung einzugehen - dafür sei auf weiterführende Literatur wie Guyon et al. (2006) und Niemann (2003) verwiesen - möchte man vorweg annehmen, dass die Verwendung möglichst vieler Merkmale die besten Unterscheidungsergebnisse und demnach geringste Fehlerwahrscheinlichkeit zur Folge hat (Jähne, 2005). Diese Annahme trifft jedoch nicht zwingend zu. Vielmehr zeigt sich, dass die sukzessive Hinzunahme weiterer Merkmale das Klassifikationsergebnis mitunter wieder verschlechtert. Abbildung 3.2 zeigt dieses Phänomen.

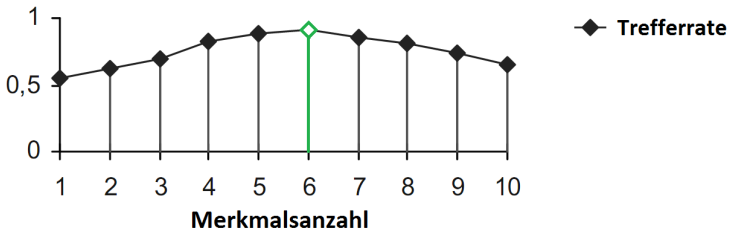


Abbildung 3.2: Peaking-Phänomen bei sukzessiv zunehmender Merkmalsanzahl. Adaptiert aus Handels (2000).

Bei vielen Klassifikatoren tritt bei der sukzessiven Hinzunahme von Merkmalen das sogenannte *Peaking-Phänomen* auf. Im Beispiel der Abbildung 3.2 kann mit 1 Merkmal eine Trefferrate von 55% erreicht werden und mit jedem Hinzufügen eines weiteren Merkmals nimmt die Trefferrate des Erkennungssystems zu. Nach 6 Merkmalen wird ein Maximalwert (Peak) von 0.9 erreicht, dies entspricht einer Trefferrate von 90%. Ab diesem Zeitpunkt nimmt die Trefferrate aber wieder sukzessive ab, wenn ein zusätzliches Merkmal dem Erkennungssystem hinzugefügt wird. Handels (2000) begründet dieses Phänomen damit, dass durch die Zunahme der Merkmale die Anzahl der zu bestimmenden Systemparameter ebenfalls zunimmt. Ist die Menge der Trainingsdaten jedoch konstant oder gar beschränkt, so unterliegen die im Training geschätzten Parameter einer größeren Schwankungsbreite[5].

Die Nachteile bezüglich des erhöhten Aufwands bei steigender Dimensionalität kann auch die hohe Rechenleistung moderner Klassifizierungssysteme nicht wettmachen, da die Menge der Daten zum Trainieren generalisierender automatischer Klassifikatoren bei gleichbleibender Klassifizierungsleistung mit der Merkmalsanzahl exponentiell steigt[6] (Tönnies, 2005) und die Menge der Trainingsdaten in der Regel nicht unbeschränkt ist (Handels, 2000). Siehe Kapitel 3.4.1 und Kapitel 3.5. Die Anzahl der Merkmale sollte demnach möglichst gering sein, die geforderte Erkennungsleistung sollte aber trotzdem stabil sein.

[5] Jähne (2005) begründet die Unnotwendigkeit vieler Merkmale zur Separation in Klassen zudem damit, dass nur 10 Merkmale, die einzeln nur 2 Klassen voneinander trennen, schon $2^{10} = 1024$ Objektklassen separieren könnten, eine große Anzahl von Merkmalen also oft gar nicht notwendig ist. Zu Jähnes Feststellung sei angemerkt, dass dies nur im idealisierten Fall zutreffen kann, bei dem sich die Klassen im Merkmalsraum sehr gleichmäßig verteilen.

[6] Bishop (2006) bezeichnet diesen exponentiellen Zusammenhang als *Curse of Dimensionality*, übersetzt als „Fluch der Dimensionalität“ (Bishop, 2006, Seite 33-37) und verweist dabei auf Bellman (1961), der diesen Begriff eingeführt hat, um den Anstieg der Komplexität und des Datenvolumens beim Hinzufügen weiterer Dimensionen in den mathematischen Raum zu beschreiben.

3.3.5 Dimensionsreduktion

Gründe für eine Merkmalsreduktion sind sowohl die besseren Analyseergebnisse, als auch eine Performanzsteigerungen in der Merkmalsaufbereitung und der Klassifikation (Guyon et al., 2006). Das Ziel einer Dimensionsreduktion ist die Beschränkung des Merkmalsraums auf ein Mindestmaß unkorrelierter Merkmale bzw. eine Lösung „mit der kleinsten Anzahl von Merkmalen im Hinblick auf die Effizienz des Mustererkennungssystems“ (Handels, 2000, S. 257). Erreicht werden kann dies unabhängig der Klassifikationsstrategie (Beckmann et al., 2007) durch eine gezielte

- **Merkmalsextraktion** – lineare oder nicht-lineare Kombination bestehender Merkmale oder Transformation bestehender Merkmale in einen kleiner dimensionierten Merkmalsraum
- **Merkmalsselektion** – Auswahl der bestmöglichen Teilmenge bestehender Eingangsmerkmale

Die Extraktion und die Selektion der Merkmale sind ausschlaggebend für ein gutes Analyse- bzw. ein gutes Klassifikationsergebnis (Jähne, 2005). Beide Strategien bestimmen wesentlich die „Güte“[7] des gesamten Analysesystems. So widmet sich auch die einschlägige Literatur, etwa die für diese Arbeit verwendeten Quellen wie Handels (2000), Hermes (2005), Jähne (2005), Liu & Mason (2009), Niemann (2003) und sehr ausführlich vor allem Guyon et al. (2006) diesem Thema.

3.3.6 Merkmalsextraktion

Ein wesentlicher Beitrag zur Merkmalsextraktion (engl. *feature extraction*) leistet die Segmentierung der Bilder, denn Bildsegmente erlauben gegenüber Pixeln die Ableitung weiterer, auch nicht-spektraler Merkmale (siehe Kapitel 2.7). Bei einer Segmentierung wird die Anzahl der Bildobjekte durch eine Regionenbildung zwar deutlich reduziert, die Anzahl der Merkmale jedes Bildobjekts hingegen deutlich vergrößert. Auf Basis dieser zusätzlichen Merkmale können die Objekte einfacher bestimmten Klassen (Kategorien) zugeordnet werden bzw. Klassen überhaupt erst gebildet werden.

Die Art der extrahierten Merkmale und deren Interpretation kann höchst unterschiedlich sein. Bei der Interpretation beispielsweise der Farbe, der Größe und der Ausrichtung eines Objekts, also vorwiegend visueller und deskriptiver Merkmale, stößt man kaum auf Probleme, wohingegen die Deutung abgeleiteter und transformierter Merkmalswerte, wie beispielsweise der Fourierkoeffizienten schwieriger und aufwendiger ist. Welche

[7] Der Begriff Güte wird hierbei mit der Verarbeitungsgeschwindigkeit, der Zahl der verwendeten Merkmale, den Kosten des Systems und vor allem mit der Fehlerwahrscheinlichkeit bei der Klassifikation oder einem mathematisch einfacher zu behandelnden Äquivalent gleichgesetzt (Niemann, 2003).

Merkmale letztlich extrahiert werden und den Anforderungen einer späteren Analyse gerecht werden, hängt von den jeweiligen Bilddaten und der jeweiligen Anwendungs- und Klassifikationsanforderungen ab (Hermes, 2005).

Geeignete Merkmale lassen sowohl quantitative als auch qualitative Aussagen bzgl. eines Objekts zu. Die Extraktion von Bildmerkmalen entspricht einer Wiedergabe der zugrunde liegenden Bildinformationen in stark komprimierter Form. Gute Klassifikationsergebnisse können erreicht werden, wenn sich die Objekte eindeutig vom Hintergrund abheben, sich untereinander in ihren Merkmalen deutlich unterscheiden und sich nicht in ihren Merkmalsausprägungen überlappen (Hermes, 2005). Durch Merkmalsextraktion werden die bei einer Klassifizierung zu detektierenden Muster in den sog. Merkmalsraum transformiert.

3.3.7 Merkmalsselektion

Eine Selektion von Merkmalen hat vorrangig das Ziel, sowohl die für eine Analyse und Klassifikation charakteristischen Merkmale als auch jene Merkmale mit dem größten Informationsgehalt aus der Menge aller zur Verfügung stehenden Merkmale auszuwählen (Handels, 2000). Von einer guten Merkmalsauswahl (engl. *feature selection*) kann dann gesprochen werden, wenn die Merkmalsvektoren der Objekte im Merkmalsraum in einer eng begrenzten Region abgebildet werden und die Regionen der einzelnen Klassen gut voneinander separiert werden können (Tönnies, 2005). Voraussetzung dafür ist, dass sich die Objekte entsprechend ihrer Klassenzugehörigkeit in einem oder mehreren Merkmalen unterscheiden (Handels, 2000). Wie viele Merkmale notwendig sind und wie sie bewertet werden können, zeigen folgende Betrachtungen.

Sei eine erhebliche Anzahl m aus den Komponenten der Merkmalsvektoren gebildeten Merkmalsausprägungen $F_1, \ldots, F_m$ angenommen, die eine Merkmalsmenge $F = \{F_1, \ldots, F_m\}$ bilden, so erkennt man aus den nachfolgenden Gleichungen (3.2) und (3.3) schnell, dass ein primitiver *Brute-Force*-Algorithmus, bei dem alle möglichen Merkmalskombination über ein berechnetes Gütemaß miteinander verglichen werden, um jene Merkmalsteilmenge F' zu bestimmen, deren Gütemaß am besten ist, praktisch kaum durchführbar ist. Für eine unbekannte Anzahl m der Merkmale errechnet sich die Zahl der möglichen Merkmalsteilmengen $F(n)$ mit (Handels, 2000)

$$F(n) = \sum_{m=1}^{n} \binom{n}{m} = 2^n - 1 \tag{3.2}$$

Dies entspricht der Mächtigkeit der Potenzmenge minus eins, da zumindest ein Merkmal benötigt wird und die leere Menge daher nicht gezählt wird. Die Merkmalsteilmenge wächst mit der Anzahl der selektierten Merkmale n exponentiell. Selbst wenn a priori die

maximale Anzahl der Merkmale auf m beschränkt würde, wäre die Zahl der möglichen Merkmalsteilmengen $T(n, m)$ mit (Handels, 2000)

$$T(n,m) = \sum_{i=1}^{m} \binom{n}{i} \tag{3.3}$$

beträchtlich und aufgrund der Laufzeiten heutiger Computer eine exakte Lösung über ein Probierverfahren kaum sinnvoll errechenbar. Aufgrund des Aufwands werden in der Praxis mittels iterativer Verfahren, wie im Ablaufdiagramm in Abbildung 3.3 dargestellt, errechnete approximative Lösungen angestrebt.

Diese Verfahren liefern auf Basis von Gütemaßzahlen $G_S(F')$ (Kapitel 3.6) Merkmalsteilmengen F'_{best}, die nicht zwingend gleich der optimalen Merkmalsteilmenge F'_{opt} sind. Zur Vorauswahl optionaler Merkmalsteilmengen F' kommen beispielsweise sowohl heuristische Verfahren als auch Greedy- und genetische Algorithmen zum Einsatz (Handels, 2000). Für weitere Details sei auf Duda et al. (2000), Guyon et al. (2006), Handels (2000) und Niemann (2003) verwiesen.

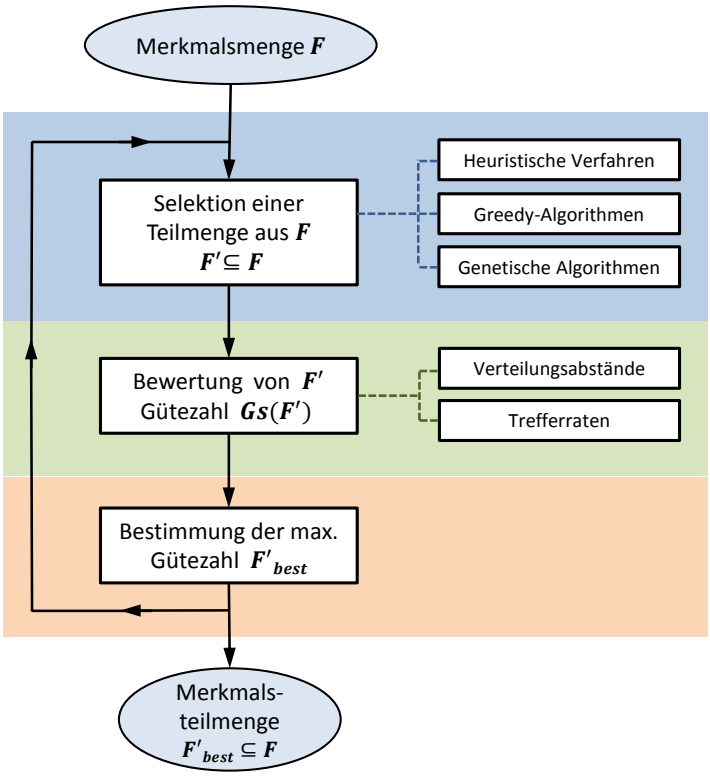


Abbildung 3.3: Iterativer Ablauf einer Merkmalsauswahl mittels Vorauswahl und einer Einzelbewertung mittels Gütezahlen. Adaptiert aus Handels (2000).

Eine besonders ausführliche Behandlung des Problems der Merkmalsauswahl und Merkmalsreduktion erfährt der interessierte Leser in Guyon et al. (2006), welcher sich ausschließlich diesem Thema widmet.

3.3.8 Diskriminierungsfähigkeit der Merkmale

Bisher wurde angenommen, dass Objekte über bestimmte Merkmale mitunter besser, über andere jedoch schlechter voneinander unterschieden werden können. Die unbeantwortete Frage lautet, welche Merkmale oder Merkmalskombinationen sich nun am besten zur Klassifikation eignen.

Wie sehr bestimmte Merkmale für ein Klassifizierungssystem taugen, hängt davon ab, wie die Merkmalswerte über die jeweiligen Klassen verteilt sind. Setzt man voraus, dass die Merkmale voneinander statistisch unabhängig sind, so können aus der Verteilungsdichtefunktion jedes Merkmals die Bewertung der Trennfähigkeit, die sogenannte *Diskriminierungsfähigkeit* und die Berechnung von Gütekriterien (siehe Kapitel 3.6) erfolgen. Das folgende Beispiel soll dies verdeutlichen.

Abbildung 3.4 zeigt die geschätzten Dichtefunktionen $f_\Omega(x)$ des Merkmals x zweier Klassen Ω_1 und Ω_2. Der Mittelwert der Klassen ist mit μ_i markiert, die Breite der Verteilungskurven wird durch die Streuung des Merkmals für die jeweilige Klasse bestimmt. Die Zuteilung eines Merkmalswerts $\underline{x}$ zu einer der beiden Klassen erfolgt über die aus den Dichtefunktionen abgeleiteten Regionen R_1 und R_2 mit der Klassifizierungsregel

$$Class(\underline{x}) = \begin{cases} \Omega_1 & \text{falls} \quad \underline{x} \in R_1 \\ \Omega_2 & \text{falls} \quad \underline{x} \in R_2 \end{cases} \tag{3.4}$$

Fällt $\underline{x}$ in R_1, gehört aber zu Ω_2 oder fällt $\underline{x}$ in R_2, gehört aber zu Ω_1, so kommt es zu Fehlklassifikationen.

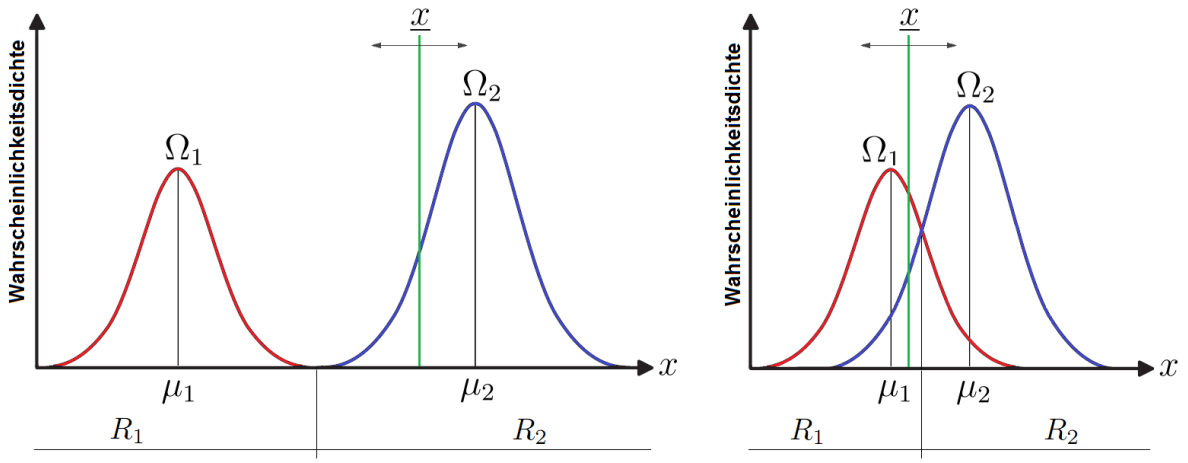


Abbildung 3.4: Beispiel einer möglichen Merkmalsverteilung für die Klassen Ω_1 und Ω_2. Links: Eindeutige Trennung der beiden Klassen über das Merkmal x. Rechts: Keine eindeutige Trennung über das Merkmal x.

Wie aus der linken Abbildung 3.4 deutlich wird, eignet sich das Merkmal x hinreichend gut, die zwei Klassen Ω_1 und Ω_2 zu trennen, da sowohl der relative Abstand zwischen den Mittelwerten μ_1 und μ_2 groß genug als auch die Streuung klein genug sind, so dass sich die Verteilungskurven nicht hochgradig überlappen. $\underline{x}$ würde problemlos der Klasse Ω_2 zugerechnet werden können.

Liegen bei gleicher Streuung die beiden Mittelwerte zu dicht beieinander, so ist eine Entscheidung für eine der beiden Klassen in den Bereichen der Überlappung der beiden Dichtefunktionen $f_{\Omega_1}(x)$ und $f_{\Omega_2}(x)$ deutlich schwieriger. Die Grenze zwischen R_1 und R_2 wird im Beispiel der rechten Abbildung 3.4 in den Schnittpunkt der Dichtefunktionen f_{Ω_1} und f_{Ω_2} gelegt. Das Merkmal $\underline{x}$ würde aufgrund der Dichtefunktionen der Klasse Ω_1 zugeteilt, es verbleibt jedoch eine im Vergleich zum linken Beispiel deutlich höhere Restwahrscheinlichkeit einer Fehlklassifikation, da aufgrund der Dichtefunktion f_{Ω_2} das Merkmal $\underline{x}$ auch der Klasse Ω_2 zugerechnet werden könnte. Aus den Flächen der Verteilungsfunktionen und den überlappenden Bereiche können sowohl die Wahrscheinlichkeiten für einen Fehler als auch die einer korrekten Klassifikation errechnet werden. Wie groß diese Wahrscheinlichkeiten letztlich sind, hängt von den Dichtefunktionen und von Festlegung der Grenze zwischen R_1 und R_2 ab.

Aus dem Mittelwert μ und der Streuung σ^2 der Klassen Ω_1 und Ω_2 kann ein einfaches Gütemaß q berechnet werden (Hermes, 2005):

$$q = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{3.5}$$

Die Voraussetzung für die Verwendung obiger Formeln ist eine im Wesentlichen an die Normalverteilung angenäherte Verteilungsdichtefunktion, aus der letztlich der Abstand q berechnet wird. Von der Verteilungsfunktion unabhängiger ist ein Gütemaß, das den mittleren quadratischen Abstand MQD der Wahrscheinlichkeitsdichteverteilungen zur Bewertung heranzieht. Sei $f_{\Omega_1}(t)$ die geschätzte Dichtefunktion der Daten der Klasse Ω_1 und $f_{\Omega_2}(t)$ die geschätzte Dichtefunktion der Daten der Klasse Ω_2 (vgl. Abbildung 3.4), dann lässt sich MQD wie folgt berechnen:

$$MQD = \int_{t=-\infty}^{+\infty} \left[f_{\Omega_1}(t) - f_{\Omega_2}(t)\right]^2 dt \tag{3.6}$$

Der Wertebereich von MQD ist $0 \leq MQD \leq 2$ und bezieht sich wie obige Gleichung auch auf einen Zweiklassenfall.

Aus den obigen Beispielen lässt sich ableiten, dass der Abstand und die Streuung der Merkmalsvektoren verschiedener Klassen unmittelbar mit der Güte der Merkmale zusammenhängen. „Im Allgemeinen kommt es darauf an, ein geeignetes Maß für den *Abstand*

der Verteilungsdichten der Merkmale aus verschiedenen Klassen zu finden, und dabei spielen *alle* Parameter eine Rolle.“ stellt Niemann (2003, S. 247) fest. Aus der Abbildung 3.4 wird auch deutlich, dass der Abstand alleine nicht ausschlaggebend für die Güte eines Merkmals sein kann. Auch im n-dimensionalen Merkmalsraum gilt, dass ein zunehmender Abstand zwischen den Klassen die Trennfähigkeit des Klassifikators steigert, eine hohe Streuungen innerhalb der Klassen (engl. *Within-Class-Scatter* – S_w) kann aber, wie eine Streuung zwischen den Klassen (engl. *Between-Class-Scatter* – S_b) eine Klassifikation empfindlich belasten. Die Formeln zur Berechnung von S_w und S_b können Tönnies (2005) entnommen werden.

Deutlich komplexer erweist sich die Bewertung der Güte unter Berücksichtigung mehrerer Merkmale im mehrdimensionalen Raum, wenn zwar die Kompaktheit noch gegeben ist, die Verteilungsdichtefunktionen sich aber entsprechend überlappen oder Inseln ausbilden. Abbildung 3.5 zeigt Beispiele von Dichtefunktionen im zweidimensionalen Merkmalsraum, deren Trennfunktionen keine Geraden, sondern Polynome n-ten Grades beschreiben. Ausschlaggebend ist dabei nicht der Abstand der Verteilungsdichtefunktionen im Eingangsraum, sondern inwieweit sich die Merkmale bzw. deren Dichtefunktionen in einen höherdimensionalen Merkmalsraum transformieren lassen, so dass sie darin mit möglichst einfach zu beschreibenden Trennebenen separiert werden können. Wie sich im folgenden Kapitel 3.4 und in Kapitel 5 zeigt, ist die Transformation der Merkmale als auch die Beschreibung der Trennebenen das Hauptanliegen von Support Vector Maschinen.

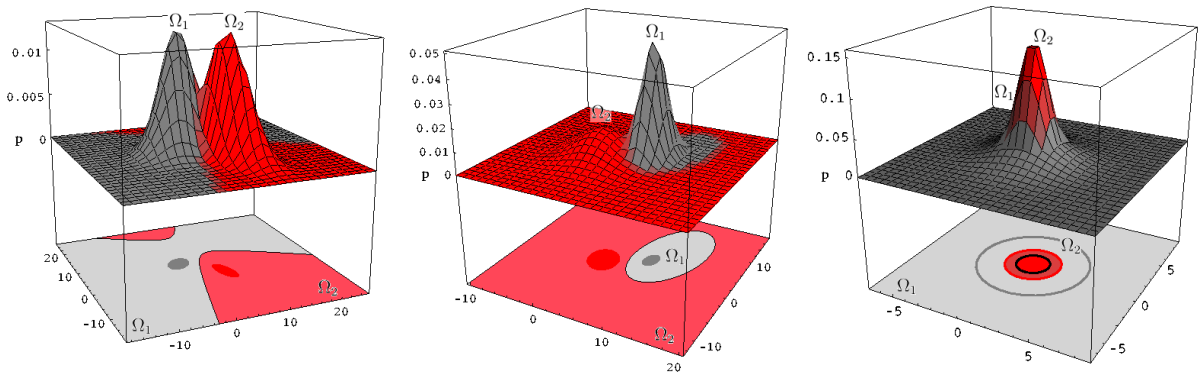


Abbildung 3.5: Beispiel möglicher Merkmalsverteilungen zweier Klassen Ω_1 und Ω_2 im 3-dim. Merkmalsraum. Adaptiert aus Duda et al. (2000).

In weiterführender Literatur verweisen Niemann (2003) und Handels (2000) auf eine Reihe weiterer Verteilungsabstände (Kolmogorow, Matusita, Chernoff, uvm.), sowie Merkmalsbewertungs- und Merkmalsauswahlverfahren, die den Zwei- sowie Mehrklassenfall im n-dimensionalen Merkmalsraum berücksichtigen. In Niemann (2003, Kap. 3.9.3) zeigt der Autor zudem mögliche Auswahlverfahren und diskutiert deren Stärken, Schwächen und etwaige Probleme.

3.4 Klassifizierung

Der Zweck der Klassifizierung besteht darin, Objekte (Bildsegmente) in Klassen einzuordnen. Die Zuordnung zu den Klassen erfolgt - sowohl bei der automatischen als auch bei der manuellen Klassifizierung - anhand von Merkmalen.

Der Fokus der folgenden Betrachtungen liegt auf der *automatischen Klassifizierung mittels maschinellen Lernens*, da diese Art der Klassifizierung in der Fernerkundung aufgrund der speziellen Anforderungen (siehe Kapitel 1.2 und 4.4.1) und großen Datenmengen weit verbreitet ist. Die im nachfolgenden Kapitel 5 eingeführte *Support Vector Machine* wird auch diesem Verfahrenstyp zugerechnet.

3.4.1 Automatische Klassifizierung mittels maschinellem Lernen

Unter einer *automatischen Klassifizierung* versteht man einen technischen Prozess, bei dem eine Kategorisierung von Objekten aufgrund ihrer Objekteigenschaften (Merkmale) ohne menschliches Zutun durchgeführt wird. Eine automatische Kategorisierung ist gemeinhin schneller und im idealen Fall auch treffsicherer und stabiler als eine von Menschen durchgeführte manuelle Kategorisierung bzw. Zuteilung in Klassen[8] (Niemann, 2003). Eine automatische Klassifizierung erfolgt in der Regeln auf Basis numerischer Verfahren (siehe Kapitel 4).

Die Bezeichnung *Maschinelles Lernen* lässt eine *lernende Maschine* vermuten. Der Begriff maschinell bezieht sich hier aber weniger auf reale Maschinen als vielmehr auf Verfahren, in denen künstliche intelligente Systeme in Form von Computerprogrammen aus Trainingsdaten (Objekte mit deren Merkmale) selbständig Gesetzmäßigkeiten lernen, um sie später zur Klassifizierung weiterer Daten (Objekte) zu nutzen. Dieser Lernprozess wird maschinelles Lernen genannt.

Als lernende Maschinen werden hierbei Algorithmen bezeichnet, die ihre bei einer Klassifizierung verwendeten Regeln mit Hilfe von Datenanalysen gewinnen. Dabei versucht ein Algorithmus aus den Merkmalen $x_1, \ldots, x_m$ der Trainingsdaten (Eingangsdaten) und deren Klassenzuweisungen y (Ausgangsdaten) funktionale Zusammenhänge abzuleiten. Die Datenmatrix ist $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_M)^\top$, der korrespondierende Klassenvektor mit den Klassenzuweisungen ist $\mathbf{y} = (y_1, \ldots, y_M)^\top$. Funktionen, die diese Zusammenhänge beschreiben, werden auch *Hypothesen* genannt. Der Lernprozess entspricht der Suche nach der passendsten Hypothese zur Beschreibung der Klassifikation der Daten. Formal kann eine lernende Maschine als eine Funktion f gesehen werden, die Objekte mit m Merkmalen

[8] Die *manuelle Klassifizierung* und ein direkter Vergleich der manuellen mit der automatischen Kategorisierung ist nicht Teil dieser Arbeit und bleibt demnach gänzlich unberücksichtigt.

aus der Menge der Eingangsdaten aus dem Eingangsraum $\mathcal{X}$ auf den Ergebnisraum $\mathcal{Y}$ abbildet (Guyon et al., 2006):

$$f : \mathcal{X} \to \mathcal{Y} \quad \text{für} \quad \mathcal{X} \subset \mathbb{R}^m \quad \text{und} \quad \mathcal{Y} \subset \mathbb{R} \tag{3.7}$$

Sowohl der genaue Ablauf der Lernphase als auch Art und Umfang der Trainingsdaten und ob deren Kategorisierung bekannt ist oder nicht, ist in erster Linie von der Wahl des jeweiligen Algorithmus bzw. Klassifikators abhängig (Niemann, 2003).

In einer groben Unterscheidung unterteilt man aufgrund der Lernphase zwischen unüberwachten (*unsupervised learning problem*) und überwachten (*supervised learning problem*) Klassifizierungsverfahren (Bishop, 2006; De Lange, 2006; Navulur, 2006; Niemann, 2003):

- **Unüberwachte Verfahren** setzen keinerlei klassifizierte Trainings- bzw. keine Referenzdaten voraus. Die Klassifikation erfolgt ausschließlich auf Basis der Merkmalsdaten.
  Sei die Trainingsmenge T eine Teilmenge von X, sei X eine Menge unklassifizierter Daten bestehend aus M Merkmalsvektoren $\mathbf{x}_i$ und sei $\mathbf{x}_i$ ein im Eingangsraum $\mathcal{X}$ definierter Vektor aus m Merkmalen mit $\mathbf{x}_i = (x_i, \ldots, x_m)^\top$, dann ist die Trainingsmenge T (Guyon et al., 2006)

  $$T = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\} \subset X \tag{3.8}$$

  Vergleichbar mit einer einfachen Clusteranalyse (Backhaus et al., 2011) versucht der Klassifikator aus der Merkmalsverteilung eine Häufung der Vektorpunkte im mehrdimensionalen Merkmalsraum zu erkennen und die Vektoren in Cluster zusammenzufassen. Die Anzahl der Cluster wird vorab oder direkt aus den zu klassifizierenden Daten gewonnen. Die unüberwachte Klassifizierung wird in dieser Arbeit nicht näher betrachtet. Dafür sei auf weiterführende Literatur wie Duda et al. (2000), De Lange (2006), Niemann (2003) und Tönnies (2005) verwiesen.

- **Überwachte Verfahren** setzen Trainingsdaten voraus, auf Basis derer der Klassifikator seine Parametrisierung erhält. Grundlage ist die Annahme, dass die Trainingsdaten die zu klassifizierenden Daten repräsentativ beschreiben. Demnach müssen die Trainingsdaten auch klassifiziert sein.
  Sei X die Menge der Eingangsdaten und Y die Menge der Ausgangsdaten und die Trainingsmenge T eine Teilmenge der Objektmenge $D = (X, Y)$, sei (X, Y) eine Menge klassifizierter Daten bestehend aus M Paarungen $(\mathbf{x}_i, y_i)$, sei $\mathbf{x}_i$ ein im Eingangsraum $\mathcal{X}$ definierter Vektor aus m Merkmalen mit $\mathbf{x}_i = (x_i, \ldots, x_m)^\top$ und sei y_i ein mit dem Vektor $\mathbf{x}_i$ stellvertretend für eine Klasse assoziierter Ausgangswert

aus dem Ausgangsraum $\mathcal{Y}$, dann ist die Trainingsmenge T (Guyon et al., 2006)

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\} \subset (X, Y) \tag{3.9}$$

Überwachte Verfahren laufen in drei Phasen ab (Handels, 2000):

1. Die überwachte *Zusammenstellung und Aufbereitung der Trainingsdaten* (optional werden alle zur Verfügung stehenden Trainingsdaten oder nur ein Teil der vorhandenen Daten, sog. *Lernstichproben* für das Training verwendet),
2. dem *Training eines Klassifikators* mit den Trainingsdaten und seine Überprüfung wiederum mit den Trainings- oder eigenen Testdaten, und
3. die eigentliche *Klassifizierung* unter Anwendung des trainierten Klassifikators, also die Zuweisung beliebiger Objekte an bestimmte Klassen.

SVMs verlangen ebenso ein überwachtes Training. Die genannten drei Phasen werden im Zusammenhang mit SVMs im Kapitel 7.3 noch detaillierter diskutiert.

Obige Ausführungen beziehen sich ausschließlich auf Verfahren in Kombination mit maschinellem Lernen. Im Folgenden sei noch eine weitere Klassifikationsmethode angeführt. Sie kommt beispielsweise in den für diese Arbeit entwickelten Algorithmen unter *eCognition* (siehe Kapitel 6) zur Anwendung.

3.4.2 Regelbasierte Klassifizierung

Bei einer *regelbasierten Klassifizierung* werden eine Reihe logischer Regeln definiert, die bei der Klassifikation sequentiell durchlaufen werden. Beispielsweise ist die Klassifikation bestimmter Regionen als Wasser oder Vegetation aufgrund einzelner Spektralkanäle nicht immer eindeutig möglich, eine einfache logische Verknüpfung mehrerer Kanäle[9] kann aber durchaus zu verwertbaren Ergebnissen führen. Der regelbasierten Klassifikation geht immer der Aufbau eines oder mehrerer Regelsätze voraus. Diese Regelsätze beschreiben mit logischen und arithmetischen Operatoren einen Entscheidungsbaum (Bartelme, 2005; Navulur, 2006; Definiens, 2011). Durch die Berechnung einer numerisch ausgedrückten Konfidenz (Vertraulichkeit) für jede Regel kann zudem eine Priorisierung oder Kombination mehrerer Regeln berücksichtigt werden.

Jede automatische Klassifikation setzt die Festlegung eines entsprechenden Regelwerks voraus, mit dem unbekannte Daten einer bestimmten Kategorie zugeordnet werden. Sind die Klassengrenzen zu exakt auf bestimmte (Trainings-)Daten abgestimmt, so werden

[9] Die benutzerdefinierte Regel könnte beispielsweise lauten: `if NIR < 20% and BLUE < 4% then "Wasser"` und `if NDVI > 0.4 then "Vegetation"`.

diese damit zwar gut abgebildet, eine spätere Kategorisierung unbekannter Daten scheitert aber oft, da die Daten aufgrund geringer Abweichungen (Rauschen) oftmals nicht mehr innerhalb der engen Klassengrenzen liegen. Sind die Klassengrenzen zu weitläufig und inexakt, so tritt der umgekehrte Fall ein und die Daten werden falsch zugeordnet. Das Problem jedes Klassifikators ist also die Festlegung der Klassengrenzen als brauchbare Verallgemeinerung, welche sowohl die Trainingsdaten als auch die erwarteten Daten abbildet.

3.5 Generalisierung

Beim maschinellen Lernen werden die Trainingsdaten beispielhaft gelernt. Das Ziel ist dabei, nicht einzelne Zusammenhänge exakt nachzubilden, sondern aus allen im Hypothesenraum möglichen Hypothesen eine allgemein gültige Gesetzmäßigkeit bzw. Hypothese derart abzuleiten, dass der Klassifizierer (engl. *classifier*) zu einem späteren Zeitpunkt unbekannte (Test-)Daten mit geringstmöglichen Fehlern klassifizieren kann (Cristianini & Taylor, 2000). Diese bestmögliche Verallgemeinerung nennt man *Generalisierung* (Duda et al., 2000).

3.5.1 Generalisierungsfähigkeit

Ein Klassifizierer weist eine gute Klassifizierungsleistung auf, wenn er stabile Klassifikationsergebnisse mit einer geringen Fehlerrate liefert. Die Fähigkeit, durch die in der Lernphase mit der Anpassung auf die Lerndaten auch gute Separationsergebnisse mit unbekannten Daten erreicht werden können, wird als *Generalisierungsfähigheit* (engl. *generalization ability*) bezeichnet. Ist die Generalisierungsfähigheit groß, so approximiert die Trennfunktion sowohl gut an die Trainingsdaten als auch an die unbekannten Testdaten. Die Klassifikationsleistung ist bei beiden Datensätzen konstant.

Passt sich die Trennfunktion unzureichend an die Trainingsdaten an, so geht die Klassifizierungsleistung verloren. Möglich ist dabei eine sogenannte Über- oder Unteranpassung (Bishop, 2006; Boser et al., 1992):

- **Überanpassung**: Im Gegensatz zu modellbasierten Methoden, die auf allgemeine Annahmen und Vorhersagen bezüglich der Datenmerkmale beruhen, tendieren modellfreie Methoden dazu, sich in der Lernphase auf die Trainingsdaten zu spezialisieren. Passt sich der Algorithmus zu sehr an die Trainingsdatensätze und an deren Streuung an, wie im linken Beispiel der Abbildung 3.6 mit der roten Linie dargestellt, so spricht man von einer Überanpassung (engl. *overfitting*) oder von einem Übertraining (engl. *overtraining*).

- **Unteranpassung**: Umgekehrt gibt es auch eine Unteranpassung (engl. *underfitting* oder *undertraining*), bei der der Klassifikator die Verteilung der Daten im Merkmalsraum nicht ausreichend erklären kann. In der Regel ist das beschreibende Modell, wie im linken Beispiel der Abbildung 3.6 mit der orangefarbigen Geraden dargestellt, zu einfach und kann die Struktur der (Trainings-)Daten nur unzureichend wiedergegeben.

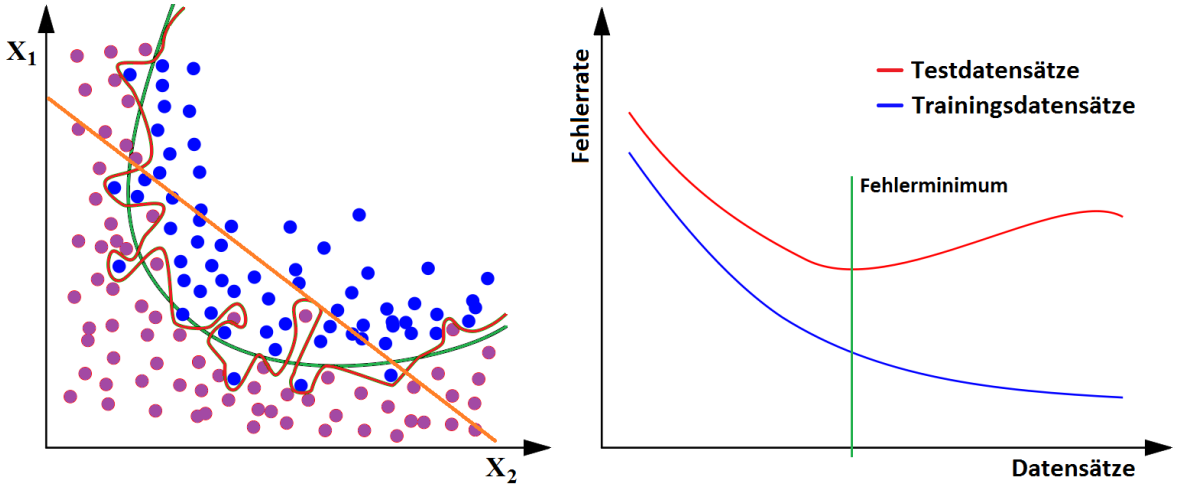


Abbildung 3.6: Links: Beispiel einer Generalisierung im 2-dim. Merkmalsraum. Gute Generalisierung (grüne Linie), Überanpassung (rote Linie) und Unteranpassung (orange Linie).
Rechts: Klassifizierungsfehler bei Trainingsdaten (blaue Linie) und bei Testdatensätzen (rote Linie). Optimale Anzahl der Trainingsdatensätze bei minimalem Testdatenfehler (senkrechte Linie). Adaptiert aus Duda et al. (2000).

Beide Effekte, sowohl eine Überanpassung als auch eine Unteranpassung wirken sich kontraproduktiv auf die erwartete Generalisierungsfähigkeit aus und sind zu vermeiden. In beiden Fällen ist eine Übertragbarkeit der Hypothese auf die Gesamtheit aller Daten nicht gegeben. Eine mögliche Über- oder Unteranpassung kann durch die Steuerung der Anzahl der Trainingsdatensätze vermieden werden. Eine zu hohe Anzahl an Trainingsdatensätzen kann - in Abhängigkeit des Klassifikators - eher zu einem Übertraining führen, eine zu geringe Anzahl von Trainingsdatensätzen im Verhältnis zur Anzahl der Merkmale kann eher zu einer Unteranpassung führen.

Wie sich aus der Erfahrung zeigt, kann bei einer automatischen Klassifizierung mittels maschinellen Lernens eine Änderung der Dimensionalität (siehe auch Kapitel 3.3.5) der Daten eine signifikante Auswirkung auf die Anzahl benötigter Trainingsdatensätze zur Folge haben. Während bei abnehmender Dimensionalität der Daten in der Regel auch weniger Trainingsdaten benötigt werden, um beispielsweise eine Separationsfunktion (siehe Kapitel 4.2) zu beschreiben, steigt der Bedarf an Trainingsdaten mit jeder Zunahme

der Dimension exponentiell (Bishop, 2006). Gesucht ist für das Training demnach jene Datensatzanzahl, bei der durch Überprüfung mit Testdatensätzen ein Fehlerminimum erreicht werden kann. Dieser Zusammenhang ist in der rechten Abbildung 3.6 dargestellt.

Die Generalisierung des Klassifikators kann ohne Berücksichtigung dieser Zusammenhänge mitunter stark leiden (Duda et al., 2000; Bishop, 2006). Support Vector Machines gelten jedoch als weitgehend unempfindlich bezüglich derartiger Probleme (Cristianini & Taylor, 2000; Scholkopf & Smola, 2001), solange in der Parametrisierung keine groben Fehler gemacht werden (vgl. Kapitel 7.8). Diese „Gutmütigkeit“ ist auch darin begründet, dass SVMs versuchen, die Generalisierung zu maximieren und nicht – wie andere Klassifikatoren – die Fehler zu minimieren (vgl. Kapitel 4.2 und 5).

3.5.2 Überprüfung der Generalisierungsfähigkeit

Die Generalisierungsfähigkeit der Klassifikatoren wird vorzugsweise mit Testdaten überprüft, deren Klassifikation bekannt ist. Um den Grad der Verallgemeinerung zu prüfen bzw. um ein Overfitting frühzeitig zu erkennen, gibt es einige Methoden, von denen nur einige hier beispielhaft aufgeführt werden (Duda et al., 2000; Handels, 2000; Lohninger, 2011):

- Bei der **Kreuzvalidierung** werden beispielsweise die aus M Vektoren bestehenden Trainingsdaten T in k ($k \leq M$) gleich große Teilmengen $\tau_1, \ldots, \tau_k$ aufgeteilt. In $i = 1, \ldots, k$ Einzeldurchläufen wird nun jeweils die i-te Teilmenge τ_i für eine Testklassifikation und die verbleibenden Teilmengen τ_j für $j \neq i$ zum Trainieren des Klassifikators verwendet. Nach dem Training wird eine Testklassifikation mit der Teilmenge τ_i durchgeführt, dann werden die Ergebnisse mit den tatsächlichen Klassenangaben verglichen, um daraus eine Fehlerquote zu errechnen. Die Gesamtfehlerquote errechnet sich aus dem arithmetischen Mittelwert der Fehlerquoten der einzelnen Durchläufe. Diese Vorgehensweise wird als k-fache Kreuzvalidierung[10] (engl. *k-fold cross-validation*) bezeichnet.

- Eine Generalisierung kann als Rauschunempfindlichkeit interpretiert werden. Fügt man den Trainingsdaten zunehmend ein **Rauschen** unter gleichzeitiger Überprüfung der Stabilität des Klassifikators hinzu, so kann der Rauschpegel in Beziehung der Generalisierungsfähigkeit des Algorithmus gesetzt und die Generalisierung somit bewertet werden.

- Die einfachste Überprüfung kann mit einem unabhängigen **Monitoring-Datensatz** erfolgen. Das Training wird abgebrochen, sobald die Klassifikationsleistung ein Optimum (oder akzeptables Ergebnis) darstellt.

[10] Beim Sonderfall $k = M$ wird von einer *Leave-One-Out*-Kreuzvalidierung gesprochen.

3.6 Bewertung der Klassifikationsgüte

Zur quantitativen Bewertung der Klassifikationsgüte stehen eine Reihe von Gütemaßen zur Verfügung, von denen einige in dieser Arbeit (siehe Kapitel 7, *Anwendung, Bewertung und Optimierung von SVMs*) zur Anwendung kommen. Die objektive Überprüfung der Klassifikation setzt das Vorhandensein bekannter bzw. klassifizierter Datensätze bzw. Prüfflächen (engl. *ground truth data*) voraus. Diese als Referenzdaten verwendeten Prüfflächen dürfen nicht zuvor schon als Trainingsflächen verwendet worden sein (De Lange, 2006). Steht eine ausreichende Anzahl von vorklassifizierten Datensätzen zur Verfügung, so kann die Aufteilung der Datensätze in Trainingsdaten und Testdaten in einem festen Verhältnis erfolgen. Letztlich bestimmen jedoch die Komplexität der Trainingsdaten und die Datensatzanzahl pro Klasse die optimale Trainings- und Testmenge (vgl. Kapitel 7.4, 7.9.3 und 7.9.4). Die Auswahl hat in allen Fällen zufällig zu erfolgen.

Üblicherweise haben sich lagespezifische Verfahren durchgesetzt, bei denen nicht nur ausschließlich statistische Aussagen über den Anteil korrekt oder nicht-korrekt klassifizierter Datensätze getroffen werden, sondern auch die Abhängigkeiten klassenspezifischer Fehlklassifikationen dokumentiert werden. Dabei wird das Klassifikationsergebnis mit Referenzdaten verglichen. Der Vergleich wird unter Anwendung sogenannter *Konfusionsmatrizen* (oder *Konfusionstabellen*) durchgeführt (De Lange, 2006).

3.6.1 Konfusionsmatrix

Eine einfache Untersuchung der Klassifikation eines überwachten Verfahrens erlaubt die *Konfusions-* oder *Fehlermatrix*. In einer quadratischen Matrix für n Klassen, deren $i = 1 \ldots n$ Zeilen die aktuelle Klassifikation und deren $j = 1 \ldots n$ Spalten die Referenzklassen wiedergeben, werden entweder die absoluten oder die relativen Häufigkeiten x_{ij} der Objekte je Klasse bzw. die bei der Klassifizierung errechnete Zuordnung zu den Klassen eingetragen[11] (Abe, 2010; De Lange, 2006).

Abbildung 3.3 zeigt das Beispiel einer Konfusionsmatrix zur Dokumentation der Klassifizierung von fünf ($n = 5$) Klassen. In der hellblau schattierten Hauptdiagonalen der Konfusionsmatrix werden jene Segmente abgebildet, die korrekt klassifiziert wurden, wohingegen in den restlichen Zellen der Matrix die falsch klassifizierten Objekte eingetragen werden. Aus den Feldern lässt sich die Summe der i-ten Zeile mit $\bar{S}_i = \sum_{j=1}^{n} x_{ij}$, die Summe der j-ten Spalte mit $\check{S}_j = \sum_{i=1}^{n} x_{ij}$ und die Summe $N = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$ aller klassifizierten Objekte errechnen.

[11] Ist der Umfang der Daten zu groß, wobei bei objektbasierten Klassifikationsverfahren dies seltener zutrifft als beispielsweise bei pixelbasierten Verfahren, so kann auch eine zufällig Auswahl von Objekten aus den zur Verfügung stehenden Testdaten zur Bewertung herangezogen werden.

Konfusionsmatrix		Referenzdaten					
		Nadelwald	Abgeerntet	Wiese	Laubwald	Mischwald	Summe
Klassifikation	Nadelwald	1797	0	1	4	402	2204
	Abgeerntet	1	5034	373	100	36	5544
	Wiese	3	931	14286	1516	861	17597
	Laubwald	8	37	739	4357	1941	7082
	Mischwald	1026	39	290	1674	4566	7595
	Summe	2835	6041	15689	7651	7806	40022
	User Acc.	63,39%	83,33%	91,06%	56,95%	58,49%	
	Producer Acc.	81,53%	90,80%	81,18%	61,52%	60,12%	
	Total Acc.	75,06%					
	Kappa	0,6593					

Tabelle 3.3: Beispiel einer Konfusionsmatrix

3.6.2 Gütemaße aus der Konfusionsmatrix

Aus der Konfusionsmatrix lassen sich eine Reihe von Gütemaße aus verschiedenen Perspektiven ableiten (Abe, 2010; De Lange, 2006).

- **Overall Accuracy** (OA): Aus dem Verhältnis der Summe S_K der korrekt klassifizierten Objekte in der Hauptdiagonalen zur Gesamtanzahl N aller Segmente in der Matrix errechnet sich mit

$$OA = \frac{\sum_{i=1}^{n} x_{ii}}{N} \tag{3.10}$$

  die Gesamtgenauigkeit der Klassifikation, die sog. *Overall Accuracy*. Zufällige Übereinstimmungen werden dabei aber ebenso wenig berücksichtigt wie ungünstige Verteilungen der Fehler. Abhilfe schaffen klassenspezifische Gütemaße.

- **User's Accuracy** (UA): Klassenspezifisch ist die aus den Zeilen der Matrix zu berechnende Benutzergenauigkeit (engl. *user's accuracy*):

$$UA_i = \frac{x_{ii}}{\bar{S}_i} \tag{3.11}$$

  Die UA drückt die 1 minus Wahrscheinlichkeit eines Fehlers 1. Art (engl. *commission error*) von irrtümlich einer anderen Klasse zugewiesenen Beobachtungen aus. Mit 100 multipliziert gibt die UA an, wie viel Prozent der einer bestimmten Klasse zugewiesenen Objekte mit den Referenzdaten übereinstimmen.

- **Producer's Accuracy** (PA): Die aus den Spalten der Matrix zu ermittelnde Herstellergenauigkeit (engl. *producer's accuracy*) dokumentiert die 1 minus Wahrscheinlichkeit eines Fehlers 2. Art (engl. *omission error*) bzw. der fälschlicherweise einer Klasse der Referenzdaten nicht zugeordneten Objekte. Berechnet wird sie

ähnlich der UA mit der Gleichung

$$PA_j = \frac{x_{jj}}{\check{S}_j} \tag{3.12}$$

Mit 100 multipliziert gibt die PA an, wie viel Prozent der Objekte der Referenzdaten einer Klasse bei der Klassifizierung auch dieser Klasse zugeordnet wurden.

Sowohl die UA als auch die PA ermöglichen eine Überprüfung, ob die Klassifikationsgüte bei einzelnen Klassen auf Kosten der Klassifikationsgüte anderer Klassen erreicht wurde. Angenommen, alle Segmente eines Bildes würden stur einer einzigen Klasse zugeordnet, so würde bei dieser einen Klasse eine Genauigkeit von 100% erreicht, wohingegen für alle anderen Klassen eine Klassifikationsgenauigkeit von 0% ausgewiesen würde. Die User's Accuracy drückt den klassenspezifischen Anteil der zu viel klassifizierten Objekte, und die Producer's Accuracy jenen der nicht erfassten Objekte einer Klasse aus. Über diese beiden Qualitätsmaße wird eine Ungleichverteilung zwischen den Klassen schnell identifiziert.

- **Kappa**: Zur Klärung der Frage, inwieweit die Güte einer Klassifizierung das Produkt eines Zufalls ist, wird der in der Fernerkundung mittlerweile häufig verwendete *Kappa-Koeffizient* κ aus der KHAT-Statistik[12] errechnet, der sowohl die PA als auch die UA berücksichtigt (Congalton & Green, 2009; De Lange, 2006):

$$\kappa = \frac{N \sum_{i,j=1}^{n} x_{ij} - \sum_{i,j=1}^{n} \bar{S}_i * \check{S}_j}{N^2 - \sum_{i=1}^{n} \bar{S}_i * \check{S}_j} \tag{3.13}$$

Der Kappa-Koeffizient drückt den zufallskorrigierten Anteil der übereinstimmenden Bewertungen zweier Klassifikatoren - des Referenzklassifikators und des zu überprüfenden Klassifikators - aus (Grouven et al., 2007). Kappa-Koeffizienten liegen im Bereich von -1 (keine Übereinstimmung) über 0 (der Anteil der zufällig korrekten und nicht korrekten Klassifikationen ist gleich groß) bis 1 (totale Übereinstimmung). Autoren wie Congalton & Green (2009) und De Lange (2006) interpretieren κ-Werte von > 0.80 als eine gute, < 0.40 als schlechte und den Bereich dazwischen als mäßige Klassifizierungsgenauigkeit. Landis & Koch (1977) unterteilen den Bereich feinmaschiger und leiten sechs Qualitätsstufen von Kappa ab.

Zur Herleitung der Gleichung von κ und dessen Interpretation sei auf weiterführende Literatur wie Congalton & Green (2009) oder Backhaus et al. (2011) verwiesen.

[12] KHAT oder auch Khat steht für „actually $\hat{K}$, an estimate of Kappa. [...] This measure of agreement is based on the difference between the actual agreement in the error matrix (i.e., the agreement between the remotely sensed classification and the reference data as indicated by the major diagonal) and the chance agreement that is indicated by the row and column totals [...] In this way, the KHAT statistic is similar to the more familiar chi-square analysis." (Congalton & Green, 2009, S.105)

4 Numerische Klassifizierung

Meist wird eine automatische Klassifizierung mit Hilfe numerischer Klassifikatoren durchgeführt. Dazu werden die Merkmale numerisch beschrieben und vorzugsweise als mathematische Größe (*Skalar*) dargestellt (Hermes, 2005).

Mathematisch ausgedrückt handelt es sich bei einer Klassifikation immer um die Abbildung $\mathcal{X} \rightarrow \mathcal{Y}$ eines Eingangsraum (Merkmalsraums) $\mathcal{X}$ auf einen Ergebnisraum (Entscheidungsraum) $\mathcal{Y}$. Der *Entscheidungsraum* $\mathcal{Y}$ ist eine Menge, bei der die Elemente jeweils einer bestimmten Objektklasse zuzüglich einer Zurückweisungsklasse für nicht-identifizierbare Objekte entsprechen. Die Elemente sind entweder (Jähne, 2005):

- bei deterministischen Entscheidungen im Zweiklassenfall $y \in \{-1, +1\}$, sie entsprechen einem einfachen *Ja* oder *Nein*, oder bei mehr als zwei Klassen $y \in \{1, \ldots, n\}$ (wobei diese letztlich auch als eine Kombination binärer Klassifizierungen verarbeitet werden; vgl. Kapitel 4.3); oder
- reelle Zahlen $y \in \mathbb{R}, y = (y_1, \ldots, y_n), y_j \geq 0, j = \{1, \ldots, n\}, \sum y_j = 1$, welche die Wahrscheinlichkeit einer Klassenzugehörigkeit angeben und deren Summe 1 ergibt.

Ähnliche Objekte haben durch n Merkmale definierte ähnliche Merkmalsvektoren, die im n-dimensionalen Merkmalsraum eine ähnliche Lage einnehmen. Kommt es im Merkmalsraum zu Häufungen von Merkmalsvektoren, so können sie zu einem n-dimensionalen *Cluster* zusammengefasst werden. Gruppiert bilden sie dann eine zusammengehörige Klasse von Merkmalsvektoren (Hermes, 2005).

4.1 Klassifizierungsansätze

Numerische Klassifikatoren nehmen Objekte entgegen und ordnen diese unter Beachtung der Klassengrenzen bestimmten Klassen zu. Jedes numerische Klassifikationsverfahren lässt sich einem von drei grundsätzlich unterschiedlichen Klassifizierungsansätzen zuordnen (Hermes, 2005; Duda et al., 2000; Niemann, 2003):

- **Klassifizierung mit Distanzfunktionen**: Der erste der drei Ansätze verwendet *geometrische Abstandsmaße* bzw. *Distanzfunktionen*, auf deren Basis die Punkte im n-dimensionalen Raum einer Klasse zugeordnet werden. Die vom Abstandsmaß abgeleitete Regel lautet: ist der Abstand gering, so wird eine größere Ähnlichkeit angenommen, ist der Abstand groß, so wird eine geringere Ähnlichkeit angenommen.

Ein Datum wird dann einer bestimmten Klasse zugeschrieben, wenn der Abstand des Datums zum Repräsentationsvektor der Klasse oder zu den Klassengrenzen einen bestimmten Schwellwert unterschreitet oder im Mehrklassen-Fall, wenn der Abstand des Datums geringer ist als zu den anderen Klassen. Distanzfunktionen können auf Punkte (Euklidsche-, City-Block-, Minkowski-, gewichtete Minkowski-, quadratische Pseudo-Distanzfunktion uvm.), Mengen, Sequenzen oder auch Binärdaten aufsetzen.

- **Klassifizierung mit statistischen Entscheidungstheorien**: Der zweite Klassifikationsansatz baut auf *statistischen Entscheidungstheorien*, bei der eine Klassifikation auf Basis von Wahrscheinlichkeiten erfolgt. Als Beispiel sei hier die Bayes'sche Entscheidungstheorie genannt.
- **Klassifizierung mit Diskriminantenfunktionen**: Der dritte Ansatz, zu dem auch Support Vector Machines gezählt werden, verwendet sogenannte *Trenn-* oder *Diskriminantenfunktionen*. Gesucht wird dabei jene beliebig komplexe, n-dimensionale Funktion, welche die Daten bezüglich ihrer Klassenzuteilung am besten[1] trennt.

Wie im Kapitel 5 gezeigt wird, definieren Support Vector Machines Trennfunktionen oder - im höherdimensionalen Raum - sogenannte Trennflächen, um die Trainingsdaten in zwei Klassen aufzuteilen. Die Basis dafür stellt die Klassifizierung mit Diskriminantenfunktionen bereit, der sich der Abschnitt 4.2 widmet.

4.2 Klassifizierung mit Diskriminantenfunktionen

Das Grundproblem jeder maschinellen Klassifizierung, bei der ein Klassifikator auf Basis von Trainingsdaten definiert wird, ist die Festlegung einer verallgemeinerten Beschreibung der Entscheidungsgrenze (engl. *decision boundary*) zwischen den zu trennenden Klassen.

Eine Trennfunktion (*Diskriminanten-* oder *Diskriminanzfunktion*) ist eine mathematische Beschreibung des Grenzverlaufs zu trennender Klassen. Bei einem Klassifikationsprozess wird geprüft, auf welcher Seite der Trennfunktion das zu klassifizierende Datum liegt. Sogenannte *Null-* bzw. zuvor festgelegte *Sprungpunkte* der Trennfunktion bilden die eigentliche Trennfläche zwischen den Klassen. Abgeleitet von der Trennfunktion $g(\mathbf{x})$ lässt sich eine Entscheidungsfunktion $Class(\mathbf{x})$ in Abhängigkeit zum Merkmalsvektor $\mathbf{x}$ festlegen, die als Ergebnis die Klasse als numerischen Wert (beispielsweise -1 oder $+1$) oder die Wahrscheinlichkeit der Zugehörigkeit zu einer bestimmten Klasse liefert (Backhaus et al., 2011; Bishop, 2006).

[1] „Am besten“ wird zu einem späteren Zeitpunkt noch genauer definiert.

Die Komplexität und Dimensionalität der Trennfunktion hängen dabei unmittelbar mit der Dimensionalität des Merkmalsraums $m = \dim(\mathcal{X})$ und mit der Verteilung der Objekte zusammen. Daraus ergeben sich im einfachsten Fall lineare, im aufwendigeren Fall quadratische oder Funktionen höheren Grades (Hermes, 2005). Der Grenzverlauf kann also einer geraden (*lineare Diskriminantenfunktion*) oder einer gekrümmten Trennfläche (*nicht-lineare Diskriminantenfunktion*) entsprechen.

Zur weiteren Verdeutlichung des Prinzips von Diskriminantenfunktion sei das folgende Beispiel diskutiert, das sich vorerst auf den einfachsten Fall, dem einer *linearen Diskriminantenfunktion* beschränkt. Lineare Diskriminantenfunktionen sind vergleichsweise einfach zu berechnen. Klassifikatoren, welche lineare Diskriminantenfunktionen verwenden (engl. *linear machines*), werden dadurch bei Klassifizierungsversuchen mit unbekannten Daten oft für erste Tests vorgezogen (Duda et al., 2000).

Abbildung 4.1 zeigt die Punkte der beiden Klassen Ω_1 und Ω_2, deren Koordinaten aus den Merkmalen $\mathbf{x} = (x_1, x_2)$ definiert werden. Die Trennfunktion $g(\mathbf{x})$ beschreibt die Entscheidungsgrenze des Zweiklassenfalls, nach der ein im gleichen Merkmalsraum $\mathcal{X}$ befindlicher Punkt entweder der Klasse Ω_1 oder der Klasse Ω_2 zugeordnet wird.

Die in Abbildung 4.1 grün dargestellte lineare Diskriminantenfunktion $g(\mathbf{x})$ ist definiert mit (Duda et al., 2000)

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^\top \mathbf{x} + b \\ &= \sum_{i=1}^{m} w_i x_i + b \quad \text{für} \quad m = 2 \text{ im 2D-Fall} \end{aligned} \tag{4.1}$$

wobei $\mathbf{w}$ als Gewichtsvektor (engl. *weight vector*), b als Schwellwertgewicht (engl. *threshold weight* oder *bias*)[2] und m als die Anzahl der Merkmale bzw. die Dimension des Merkmalsraums interpretiert werden kann. Bei einer Klassifikation ist die entscheidende Frage, ob das innere Produkt $\mathbf{w}^\top \mathbf{x}$ über oder unter dem Schwellwert b liegt. In einem Zweiklassenfall entspricht die dargestellte lineare Diskriminantenfunktion $g(\mathbf{x})$ der Regel, sich für Ω_1 zu entscheiden, falls $g(\mathbf{x}) > 0$ bzw. sich für Ω_2 zu entscheiden, falls $g(\mathbf{x}) < 0$ ist. Tritt der Grenzfall $g(\mathbf{x}) = 0$ ein, so obliegt es in der Praxis der jeweiligen Implementierung, sich für eine der beiden Möglichkeiten zu entscheiden, genau genommen wäre dieser Fall *undefiniert*.

Aus obigen Ausführungen und Abbildung 4.1 wird deutlich, dass die Gleichung $g(\mathbf{x}) = 0$ jene Entscheidungsgrenze darstellt, welche die Daten bzw. Punkte der Klassen Ω_1 und Ω_2 separiert. Im einfachsten Fall ist die Diskriminantenfunktion linear, also eine Gerade durch den Merkmalsraum. Im m-dimensionalen Raum ($m > 2$) spricht man von einer

[2] Einige Quellen unterscheiden das *threshold weight* und das *bias* im Vorzeichen. „Sometimes $-b$ is replaced by θ, a quantity known as the *threshold*." (Cristianini & Taylor, 2000, Kapitel 2.1)

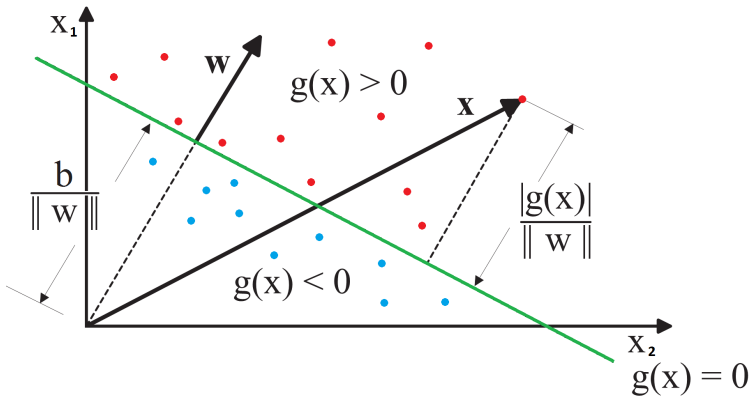


Abbildung 4.1: Darstellung einer linearen Diskriminantenfunktion zur Trennung der Klassen Ω_1 und Ω_2. Adaptiert nach Hermes (2005).

Hyperebene (engl. *hyperplane*), wenn $g(\mathbf{x})$ zwischen den Klassen linear ist. Wenn sowohl $\mathbf{x}_1$ als auch $\mathbf{x}_2$ auf der Hyperebene liegen, so gilt (Duda et al., 2000)

$$\mathbf{w}^\top \mathbf{x}_1 + b = \mathbf{w}^\top \mathbf{x}_2 + b \quad \text{oder} \quad \mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2) = 0 \tag{4.2}$$

und dann ist $\mathbf{w}$ normal zu jedem anderen in der Hyperebene $\mathbf{H}$ liegenden Vektor. Teilt $\mathbf{H}$ nun den Merkmalsraum $\mathcal{X}$ in zwei Hälften, so entstehen sowohl die Entscheidungsregion $\mathcal{R}_1 = \{\mathbf{x} : g(\mathbf{x}) > 0\}$ für die Klasse Ω_1 als auch die Entscheidungsregion $\mathcal{R}_2 = \{\mathbf{x} : g(\mathbf{x}) < 0\}$ für die Klasse Ω_2. Liegt nun $\mathbf{x}$ in $\mathcal{R}_1$, so ist $g(\mathbf{x}) > 0$ und der Normalvektor $\mathbf{w}$ zeigt in die Entscheidungsregion $\mathcal{R}_1$. Anders ausgedrückt gilt auch, dass jedes $\mathbf{x}$ in $\mathcal{R}_1$ auf der positiven Seite und jedes $\mathbf{x}$ in $\mathcal{R}_2$ auf der negativen Seite der Hyperebene $\mathbf{H}$ liegt (Duda et al., 2000; Hermes, 2005).

Die Orientierung der Hyperebene $\mathbf{H}$ wird durch den Normalvektor $\mathbf{w}$ und der Abstand zum Ursprung wird von b bestimmt. Interpretiert man die Diskriminantenfunktion $g(\mathbf{x})$ als algebraisches Abstandsmaß r von $\mathbf{x}$ zur Hyperebene $\mathbf{H}$ und sei $\mathbf{x}_p$ die Normalprojektion von $\mathbf{x}$ auf die Hyperebene $\mathbf{H}$, so gilt

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{4.3}$$

wobei die Norm $\|\mathbf{w}\|$ wie folgt berechnet wird (Niemann, 2003):

$$\|\mathbf{w}\| = \left(\mathbf{w}^\top \mathbf{w}\right)^{\frac{1}{2}} = \sqrt{w_1^2 + w_2^2 + \cdots + w_m^2} \tag{4.4}$$

Wird $g(\mathbf{x}) = 0$ angenommen, so gilt aus der vorherigen Gleichung abgeleitet

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = r \, \|\mathbf{w}\| \tag{4.5}$$

Daraus lässt sich einfach der Normalabstand r ermitteln

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|} \tag{4.6}$$

Das Vorzeichen von r gibt dabei an, auf welcher Seite der Hyperebene der Vektor $\mathbf{x}$ liegt. In gleicher Weise kann der Abstand r_b der Hyperebene $\mathbf{H}$ von Ursprung berechnet werden (Duda et al., 2000).

$$r_b = \frac{\mathbf{b}}{\|\mathbf{w}\|} \tag{4.7}$$

Die exakte Herleitung dieser Gleichungen kann Duda et al. (2000) entnommen werden. Wie Abbildung 4.1 verdeutlicht, liegt bei $b > 0$ der Ursprung auf der negativen Seite der Hyperebene $\mathbf{H}$ und bei $b < 0$ der positiven Seite von $\mathbf{H}$. Ist $b = 0$, dann kreuzt die Diskriminantenfunktion mit $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ den Ursprung.

Wie obige Gleichungen deutlich machen, wird die Position der Hyperebene durch den Bias b und die Orientierung der Hyperebene durch den Normalvektor $\mathbf{w}$ definiert. Zusätzlich ist das Ergebnis der Diskriminantenfunktion vorzeichengleich und proportional zum Abstand von $\mathbf{x}$ zur Hyperebene. Aus diesem Zusammenhang lässt sich aus (4.5) in Abhängigkeit des Vorzeichens eine einfache Klassifizierungsregel für eine binäre Klassifikation mit zwei Klassen Ω_1 und Ω_2 ableiten:

$$\text{Klassifizierungsregel} \quad Class(\mathbf{x}) = \begin{cases} \Omega_1 & \text{falls} \quad \mathbf{w}^\top \mathbf{x} + b \geq 0 \\ \Omega_2 & \text{falls} \quad \mathbf{w}^\top \mathbf{x} + b < 0 \end{cases} \tag{4.8}$$

Ist $g(\mathbf{x}) \geq 0$, dann wird der Vektor $\mathbf{x}$ der Klasse Ω_1 zugerechnet, ist $g(\mathbf{x}) < 0$ dann wird der $\mathbf{x}$ der Klasse Ω_2 zugerechnet. Üblicherweise werden die Räume zu beiden Seiten der Hyperebene mit +1 und -1 bezeichnet (Niemann, 2003). Dies lässt sich durch eine Abänderung der Gleichung (4.8) auf $Class(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$ realisieren.

Bisher ist die Methode zur Berechnung der exakten Position und Orientierung der linearen Trennfunktion bzw. Hyperebene unbehandelt geblieben. Wie beispielsweise Abbildung 4.1 zeigt, gibt es durchaus mehrere Lösungen, die beiden Klassen Ω_1 und Ω_2 mit einer linearen Trennfunktion zu separieren. Abhängig vom jeweiligen Klassifikator ist die optimale Hyperebene aus den unendlich vielen möglichen Hyperebenen unterschiedlich definiert. Bei Support Vector Machines (siehe Kapitel 5) wird beispielsweise die lineare Diskriminantenfunktion so durch die Punktwolke der beiden Klassen gelegt, dass die Abstände der nächstgelegenen Datenpunkte zur Diskriminantenfunktion jeweils maximiert werden und das Risiko einer Fehlklassifikation minimiert wird. Aus diesen Vorgaben kann sowohl die Richtung als auch die exakte Lage der Diskriminantenfunktion eindeutig bestimmt werden (Bishop, 2006).

4.3 Mehrklassen-Klassifizierung

Bei den bisherigen Betrachtungen wurde stets der Zweiklassenfall (*binäre Klassifikation*) betrachtet, bei dem der Klassifikator die beiden zu trennenden Klassen über eine Diskriminantenfunktion separiert. Für mehr als zwei Klassen (*Mehrklassen-Klassifikation*) gibt es eine Reihe von Verfahren, die auf ein Zweiklassenproblem zurückgeführt werden können (Niemann, 2003).

One-Against-All Methode

Die Klassifizierung von n für $n > 2$ Klassen lässt sich mit der Strategie „Eine-gegen-Alle“ (engl. *One-Against-All*[3]) auf n Zweiklassen-Klassifizierungen zurückführen (siehe linke Abbildung 4.2). Dabei wird pro Klasse ein Klassifikator trainiert, der jeweils eine Klasse Ω_i mit allen anderen verbleibenden $n-1$ Klassen vergleicht. Der erste Klassifikator vergleicht die Klasse Ω_1 mit allen anderen zu einer Menge zusammengefassten Klassen $\{\Omega_2, \ldots, \Omega_n\}$, der zweite Klassifikator vergleicht die Klasse Ω_2 mit allen anderen zu einer Menge zusammengefassten Klassen $\{\Omega_1, \Omega_3, \ldots, \Omega_n\}$ usw. (Niemann, 2003).

Der Klassifikator der Klasse Ω_i entspricht einer Entscheidungsfunktion $g_i(\mathbf{x})$ für $i = 1, \ldots, n$. Dabei wird der Vektor $\mathbf{x}$ dann der Klasse Ω_i zugeordnet, wenn $g_i(\mathbf{x}) > 0$, ansonsten wird er den verbleibenden Klassen zugeteilt (Abe, 2010). Die finale Zuweisungsstrategie lautet demnach:

$$\text{Klassifizierungsregel} \quad Class(\mathbf{x}) = \Omega_i \quad \text{falls} \quad g_i(\mathbf{x}) > 0 \text{ und } g_j(\mathbf{x}) < 0 \tag{4.9}$$

$$\text{für} \quad i = 1 \ldots n \text{ und } j = 1 \ldots n \text{ und } j \neq i \tag{4.10}$$

Diese Entscheidungsstrategie hat den Nachteil, dass dabei Regionen entstehen, die keine eindeutige Klassifizierung zulassen. Wenn beispielsweise mehr als eine Entscheidungsfunktion positiv oder keine der Entscheidungsfunktionen positiv ist, so ist keine eindeutige Zuordnung möglich (siehe linke Abbildung 4.2 hellgrau eingefärbter Bereich).

One-Against-One Methode

Eine Alternative ist die Strategie „Jeder-gegen-Jeden“ (engl. *One-Against-One*), welche $n \cdot (n-1)/2$ Zweiklassen-Klassifikatoren mit der Entscheidungsfunktion $g_{ij}(\mathbf{x})$ benötigt, die je eine der möglichen Klassenpaarungen Ω_i und Ω_j abdecken. Dabei ist für jede Paarung nur eine Entscheidungsfunktion notwendig, da gilt $g_{ij}(\mathbf{x}) = -g_{ji}(\mathbf{x})$. Die Entscheidung fällt auf die Klassen Ω_i, wenn $g_{ij}(\mathbf{x}) > 0$ oder fällt auf die Klasse Ω_j, wenn $g_{ij}(\mathbf{x}) < 0$ für $i, j = 1, \ldots, n$ und $i \neq j$.

[3] manchmal auch als *One-Versus-The-Rest* bezeichnet (Camp-Valls & Bruzzone, 2009; Bishop, 2006)

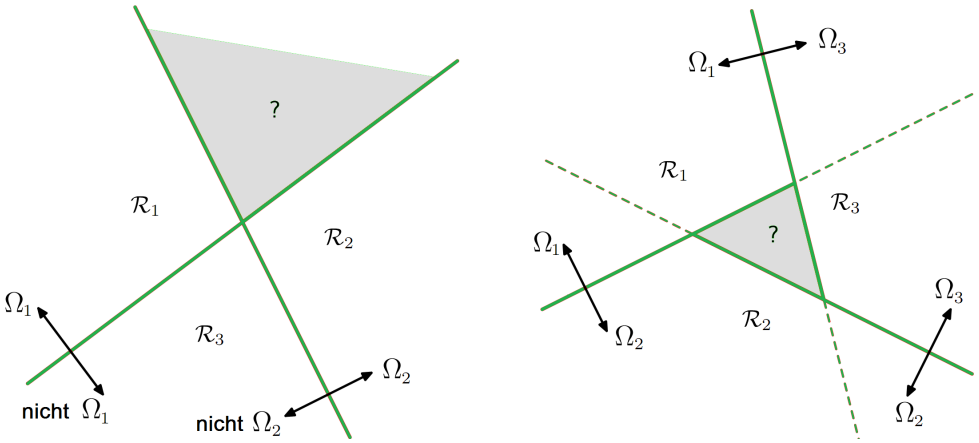


Abbildung 4.2: Mehrklassenklassifizierung mit einem One-Against-All Klassifizierer (links) und einem One-Against-One Klassifizierer. Die grauen Bereiche zeigen jene Regionen an, in denen mit diesen Strategien keine eindeutige Zuordnung erfolgt. Adaptiert aus Bishop (2006).

Da für jede Klasse nun mehrere Klassifikatoren angewendet werden, muss eine Kombination aller Klassifikatoren die Entscheidung für oder gegen eine Klasse fällen (siehe rechte Abbildung 4.2). Eine effiziente Möglichkeit besteht darin, bei jeder Anwendung eines Klassifikators die Entscheidung numerisch zu bewerten und diese Werte über alle Entscheidungen in $k_i(\mathbf{x})$ zu kumulieren. Die endgültige Entscheidung fällt letztlich auf jene Klasse Ω_j mit der höchsten Gesamtbewertung (Abe, 2010; Niemann, 2003).

$$\text{Klassifizierungsregel} \quad Class(\mathbf{x}) = \Omega_j \quad \text{für} \quad j = \underset{i=1,\ldots,n}{\operatorname{argmax}} \; k_i(x) \tag{4.11}$$

$$\text{mit kum. Entscheidungen} \quad k_i(\mathbf{x}) = \sum_{j \neq i, j=1}^{n} \operatorname{sign} g_{ij}(\mathbf{x}) \tag{4.12}$$

Abbildung 4.2 zeigt sowohl links einen *One-Against-All-* als auch rechts einen *One-Against-One*-Klassifizierer. Aus den Abbildungen wird deutlich, dass bei beiden Strategien Bereiche entstehen (in der Abbildung hellgrau eingefärbt), die keine eindeutige Zuordnung ermöglichen.

Indirect Decision Function

Dieser Mangel lässt sich durch *Indirect Decision Functions* beheben, indem die reellen Ergebnisse der Funktionen $g_i(\mathbf{x})$ zur Bewertung der Klassenzugehörigkeit herangezogen werden. Der Vektor $\mathbf{x}$ wird jener Klasse Ω_j zugeordnet, die den größten Funktionswert

ergibt (Abe, 2010):

$$Class(\mathbf{x}) = \Omega_j \quad \text{für} \quad j = \underset{i=1,\ldots,n}{\operatorname{argmax}} \; g_i(\mathbf{x}) \tag{4.13}$$

Die Entscheidungsgrenze zwischen einer Klasse Ω_j und einer angrenzenden Klasse Ω_k liegt bei $g_j(\mathbf{x}) = g_k(\mathbf{x})$. Die daraus im Falle linearer Klassifizierer resultierende $(n-1)$-dimensionale Hyperebene ist wie folgt definiert (Bishop, 2006):

$$(\mathbf{w}_j - \mathbf{w}_k)^\top \mathbf{x} + (b_j - b_k) = 0 \tag{4.14}$$

Die Gleichung der resultierenden Hyperebene ist, wie im Vergleich mit Gleichung (4.1) leicht ersichtlich ist, im Aufbau gleich der einer linearen Diskriminantenfunktion und auch so zu verwenden. Die Entscheidungsregionen sind damit eindeutig einer Klasse zugeordnet. Für detaillierte Herleitungen sei auf weiterführende Quellen wie Bishop (2006) oder Duda et al. (2000) verwiesen. Abe (2010) formuliert noch eine vierte Strategie – die sog. *Decision Tree Strategie* – als weitere Variante der *One-Against-All* Strategie. Sie weist ebenfalls alle Regionen einer bestimmten Klasse zu.

Einen detaillierten Vergleich oben eingeführter und weiterer Methoden im Zusammenhang mit Support Vector Machines diskutieren die Autoren Hsu & Lin (2002) und Melgani & Bruzzone (2008). Sie weisen überraschenderweise die *One-Against-One* Methode in Bezug auf den Berechnungsaufwand und des Klassifizierungsergebnisses als günstiger aus, obwohl die *One-Against-One* Methode mit $n \cdot (n-1)/2$ deutlich mehr Zweiklassen-Klassifikatoren benötigt als die *One-Against-All* Methode mit n Zweiklassen-Klassifikatoren. Der Gesamtrechenaufwand ist aufgrund der kleineren Trainings- und Testmenge trotzdem deutlich geringer, sodass dadurch ein erheblicher Geschwindigkeitsvorteil – vor allem in der Trainingsphase – entsteht. Auch die in vielen Fällen besseren Erkennungsraten sprechen für die Anwendung der *One-Against-One* Methode, auch wenn beispielsweise die Autoren Hsu & Lin (2002) anmerken, dass die *One-Against-All* Methode oft vergleichbar gute Ergebnisse ermöglicht (Niemann, 2003).

4.4 Numerische Klassifikatoren in der Fernerkundung

Automatische Klassifikatoren nutzen verschiedene mathematische und algorithmische Verfahren zur Klassifikation von Objekten. Der Klassifikationserfolg der jeweiligen Klassifikatoren variiert stark in Abhängigkeit der jeweiligen Problem- bzw. Fragestellung.

Wesentlich bei der Auswahl der Klassifikatoren sind die jeweiligen Anforderungen, die sich größtenteils aus den zu klassifizierenden Datenbeständen und aus den applikationsbedingten Vorgaben ableiten lassen. Die folgende kurze Übersicht zeigt die speziellen

Anforderungen aus Sicht von Fernerkundungsapplikationen auf und nennt die gängigsten Klassifikatoren in diesem Anwendungsbereich.

4.4.1 Anforderungen an Klassifikatoren

Eine Landbedeckungsklassifizierung wird im Zusammenhang teilweise recht unterschiedlicher Fragestellungen durchgeführt. Sei es die Überwachung des Anbaus auf landwirtschaftlich genutzten Flächen, die Bestimmung von Versiegelungsflächen in städtischen Gebieten oder die Aufnahme von Veränderungen der Küstenlinie. Trotz der unterschiedlichen Aufgabenstellungen lassen sich die wesentlichsten Anforderungen an Klassifikatoren aus Sicht von Fernerkundungsapplikationen verallgemeinern (Lucht et al., 2009). Die wesentlichsten Anforderungen sind in Tabelle 4.1 zusammengefasst (Duda et al., 2000; Lucht et al., 2009; Niemann, 2003).

Anforderungskriterien aus der Sicht der Fernerkundung sind...
• eine **hohe Anpassungsfähigkeit des Klassifikators** an **stark variierende und komplexe Merkmalsverteilungen**,
• eine möglicherweise **hohe Anzahl unterschiedlicher Klassen**,
• eine mögliche **Unterrepräsentation bestimmter Klassen als Lernmuster**,
• eine (teil-)automatisierte **Parametrisierung des Klassifikators** und
• die **Stabilität des Klassifikators**,
• der **Prozessablauf** (mit/ohne Training, mit/ohne Benutzerunterstützung),
• ein mitunter notwendiges domänenbezogenes **Kontext-Wissen** und
• der für das Training und Klassifizierung notwendige **Prozessierungsaufwand**.

Tabelle 4.1: Spezielle Anforderungskriterien an Klassifikatoren aus der Sicht von Fernerkundungsapplikationen

Nicht alle Klassifikatoren befriedigen die teils speziellen Anforderungen der Fernerkundung im gleichen Maße. Gerade die hohe Anzahl unterschiedlicher Klassen und die stark variierenden Merkmalsverteilungen fordern den Klassifikatoren eine hohe Anpassungsfähigkeit ab. In einschlägiger Literatur (Backhaus et al., 2011; Bishop, 2006; Duda et al., 2000; Liu & Mason, 2009; Niemann, 2003) wird eine Vielzahl von Klassifizierungsverfahren beschrieben, die in speziellen Applikationen ihre Anwendungen finden. In der Fernerkundung haben sich aber nicht alle bekannten Verfahren gleichermaßen behaupten können. Camp-Valls & Bruzzone (2009) und Navulur (2006) widmen

sich ausführlich diesem Thema und diskutieren ausgewählte Lösungsverfahren und Applikationen im Bereich der Fernerkundung. Welche Klassifikatoren sich letztlich etabliert haben, soll die folgende Übersicht zeigen.

4.4.2 Gängige Klassifikatoren in der Fernerkundung

Gemäß den obigen Anforderungskriterien haben sich für die Auswertung von Fernerkundungsdaten und Luftbildern sowohl statistische als auch verteilungsfreie Klassifikatoren als besonders geeignet erwiesen (Niemann, 2003). Dieser Abschnitt gibt einen kurzen Überblick über die gängigsten Klassifikatoren in der Fernerkundung.

Von den statistischen Klassifikatoren kommen oft die *Bayes-* und *Maximum-Likelihood*-Klassifikatoren (Abe, 2010; Duda et al., 2000) zum Einsatz, die auf Dichteberechnungen und Wahrscheinlichkeiten basieren. Sie minimieren das Risiko einer Fehlklassifizierung und schätzen die Wahrscheinlichkeitsverteilung jeder Musterklasse.

Diskriminative Klassifikatoren wie der *Hyperquader*-Klassifikator (Liu & Mason, 2009) oder die Support Vector Machine (Abe, 2010; Guyon et al., 2006; Scholkopf & Smola, 2001; Steinwart & Christmann, 2008; Vapnik, 1995, 1998) und der Abstand messende *k-Nearest-Neighbor*-Klassifikatoren (Abe, 2010; Duda et al., 2000; Navulur, 2006) setzen ebenso wie objektbasierte, hierarchische *Decision Tree*-Klassifikator (Duda et al., 2000; Guyon et al., 2006) und *Künstliche Neuronale Netze* (Guyon et al., 2006) keine bestimmte Verteilung der Merkmale voraus. *Fuzzy-Regeln* (Duda et al., 2000; Bouziani et al., 2010) nutzen unscharfe Mengen und werden ebenso wie Wahrscheinlichkeiten in Klassenzuordnungsstrategien genutzt.

Fast alle verwendeten Klassifikatoren werden den nicht-parametrischen Klassifikationsverfahren (Handels, 2000) zugerechnet, bei denen die Modellstruktur nicht a priori festgelegt sein muss, sondern erst aus den Trainingsdaten durch maschinelles Lernen bestimmt wird (Niemann, 2003). Oft führt erst die Kombination mehrere Methoden oder die sequentielle Abfolge mehrerer Klassifikatoren zum gewünschten Erfolg, wie beispielsweise in Benediktsson et al. (2007), Lee et al. (2007) und Tulyakov et al. (2008) dokumentiert ist. Eine Hilfestellung zur Wahl eines Klassifikators mit Auswahlkriterien und Charakteristiken von Klassifikatoren geben die Seiten 114 bis 115 im Anhang A.

Tabelle 4.2 gibt eine Zusammenfassung der häufig in Fernerkundungsapplikationen verwendeten Klassifikatoren samt Bezeichner[4] und Referenzen zu weiterführender Literatur wieder. Diese Auflistung soll nur einen groben Überblick geben und ist nur als selektiver

[4]Hierbei wurden bewusst die englischen Bezeichner verwendet, da diese in den meisten Fällen auch im deutschsprachigen Raum Verwendung finden.

Auszug zu verstehen, der nicht den Anspruch auf Vollständigkeit erhebt. Die Autoren Lu & Weng (2007) zeigen beispielsweise eine erheblich umfangreichere Übersicht und listen noch eine Reihe weiterer Klassifikatoren samt weiterführender Quellen.

4.4.3 Kombination mehrerer Klassifikatoren

In der Praxis werden meist Kombinationen mehrerer Verfahren verwendet (Lu & Weng, 2007), da die unterschiedlichen Algorithmen nicht für alle Datensätze und Merkmalskombination im gleichen Maße gut geeignet sind (Waske et al., 2009). Ein Verfahrensmix ist dann sinnvoll, wenn ausgewählte Methoden zwar bei bestimmten Datentypen oder

Klassifikator	Referenzen
• Box Classifier	Liu & Mason (2009)
• Decision Tree Classifier	Duda et al. (2000); Guyon et al. (2006)
• Frequency Based Contextual Classifier	Gong & Howarth (1992)
• Fuzzy-Set Classifier	Duda et al. (2000)
• GIS-basierte Classifier	Byeungwoo & Landgrebe (1993)
• Hidden Markov Model Classifier	Duda et al. (2000); Li et al. (2000)
• Iterated Conditional Modes	Byeungwoo & Landgrebe (1993); Kittler & Föglein (1984)
• Künstliches Neuronales Netz	Guyon et al. (2006)
• Lineare Diskriminanzanalyse	Duda et al. (2000)
• Maximum Distance Classifier	Duda et al. (2000); Liu & Mason (2009)
• Maximum Likelihood (Bayes) Classifier	Abe (2010); Duda et al. (2000)
• Nearest Neighbor	Abe (2010); Duda et al. (2000); Navulur (2006)
• Rule Based Classifier	Duda et al. (2000); Bouziani et al. (2010)
• Spectral Mixture Analysis	Camp-Valls & Bruzzone (2009)
• Support Vector Machine	Abe (2010); Guyon et al. (2006); Scholkopf & Smola (2001); Steinwart & Christmann (2008); Vapnik (1995, 1998)

Tabelle 4.2: Häufig in der Fernerkundung eingesetzte Klassifikatoren und Referenzen auf weiterführende Literatur

Teilmengen optimale Zuteilungsergebnisse liefern, andere Methoden aber bei anderen Teilmengen und/oder mit anderen Merkmalen noch bessere Klassifikationsergebnisse ermöglichen. Die Kombination mehrerer Teilergebnisse verschiedener Klassifikatoren führt damit vielfach zu einem besseren Gesamtergebnis (De Lange, 2006).

Moderne Bildverarbeitungsapplikationen, wie beispielsweise das in dieser Arbeit eingesetzte *eCognition* (siehe Kapitel 6), unterstützen die Festlegung sequentieller, hierarchischer und iterativer Prozessabfolgen, mit denen unterschiedlichste Methoden nahezu beliebig kombiniert werden können. Dabei erfolgt die Zerlegung der Datensätze und letztendliche Klassenzuweisung oft schrittweise, indem die Objekte bei jedem Schritt zunehmend detaillierten, hierarchisch geordneten Klassen zugeordnet werden (Ferber, 2003).

In dieser Arbeit stehen Klassifikatoren mit überwachtem Lernverfahren im Vordergrund der Betrachtung, zu denen auch die im folgenden Kapitel eingeführte *Support Vector Machine* gezählt wird.

5 Support Vector Machine

Die *Support Vector Machine* (SVM), dt. *Stützvektormaschine*[1], wird den sog. Kernel-Methoden (engl. *kernel-based methods* oder *kernel methods*) zugerechnet und beschreibt einen speziellen lernenden Algorithmus zur Mustererkennung (Abe, 2010). SVMs können sowohl zur *Klassifizierung* als auch zur *Regressionsanalyse* verwendet werden.

In den folgenden Abschnitten werden sowohl die mathematischen Grundlagen dieses Verfahrens, deren geometrische Deutung, die Sonderfälle und Ausprägungen, als auch die Anwendung von Support Vector Machine (SVMs) zur Klassifikation erläutert. Am Beginn beschreibt eine kurze Einführung die Entwicklung von SVMs und deren Anwendungsspektrum. Beide Abschnitte geben auch einen Überblick über grundlegende und ausgewählte Literatur, die im Rahmen dieser Arbeit verwendet wurde.

5.1 Historische Entwicklung

Obwohl die theoretischen Grundlagen der Support Vector Machines (SVMs) und deren Varianten und Erweiterungen erstmals schon 1964/65 von Vapnik und Chervonenkis publiziert wurden (Sewell, 2011), blieben sie bis zum Beginn der 1990er Jahre weitgehend unbeachtet (Steinwart & Christmann, 2008). Der Grundstein der heute verwendeten Algorithmen wurde schon deutlich früher in Publikationen wie beispielsweise Duda & Hart (1973) gelegt und dann von Vapnik & Chervonenkis (1974) in der Beschreibung der *Statistischen Lerntheorien* zusammengefasst und auf SVMs erweitert.

Steigende Beachtung fanden Support Vector Machines ab Mitte der 1970er Jahre, als mehr und mehr Daten mittels mathematischer Verfahren auf Regelmäßigkeiten, Ähnlichkeiten oder Gesetzmäßigkeiten untersucht wurden, um sie computergestützt zu klassifizieren oder einfache Muster in den Daten zu erkennen. Erst spätere Publikationen wie beispielsweise Boser et al. (1992), Vapnik (1995), Vapnik (1998), Cristianini & Taylor (2000) und Scholkopf & Smola (2001) verhalfen den SVMs zur letztlich entscheidenden Verbreitung.

Wie aktuell das Thema SVM mittlerweile geworden ist, zeigen aktuelle Publikationen und Bücher zu diesem Thema, wie beispielsweise Abe (2010), Camp-Valls & Bruzzone (2009), Steinwart & Christmann (2008) und Wang (2005), um nur einige zu nennen.

[1] Manchmal wird SVM auch wörtlich mit *Support Vektor Maschine* (Niemann, 2003) übersetzt.

Die letztgenannten Quellen führen auch in die mathematischen Grundlagen von SVMs und Kernel-Methoden ein und werden in dieser Arbeit vermehrt referenziert.

5.2 Anwendungsgebiete

Ursprünglich wurden SVMs zur Lösung binärer Klassifikations- und Regressionsprobleme entwickelt (Guyon et al., 2006), heute finden sich Support Vector Machines in einer Reihe von Applikationen unterschiedlichster Domänen wieder (Scholkopf & Smola, 2001). Typische Anwendungsgebiete der Mustererkennung und heute auch von SVMs sind im Bereich der Spracherkennung, Text- bzw. Schrifterkennung, Gesichtserkennung oder in der Bildverarbeitung und -analyse zu finden (Burges, 1998). Suchmaschinen und Filter nutzen Stützvektormaschinen beispielsweise für Information Retrieval (Manning et al., 2009) oder zum Data Mining (Burbidge & Buxton, 2001) oder zur Bewertung (*Ranking*) von Email-, Seiten- und Dokumenteninhalten (Chapelle & Keerthi, 2010; Cristianini & Taylor, 2000; Ferber, 2003). Biologen greifen zur Klassifikation von Genen über die Suche nach Mustern als Unterscheidungsmerkmal von biologischen Sequenzen auch gerne auf SVMs zurück (Ben-Hur et al., 2008; Byvatov & Schneider, 2003; Cristianini & Taylor, 2000; Noble, 2004) und auch in der Fernerkundung (Mountrakis et al., 2010) finden SVMs immer mehr Anwendungsmöglichkeiten, auch wenn deren Verwendung noch eher verhalten wirkt (Heinert, 2010a).

Gründe für die stete Verbreitung von SVMs sind deren günstige Eigenschaften, die Support Vector Machines von anderen lernenden Algorithmen wie beispielsweise Neuronalen Netzen (engl. *neural networks*) deutlich unterscheiden. Gerade deren Generalisierungsfähigkeit auch bei vergleichsweise wenig Trainingsdaten und deren Robustheit gegenüber dem sog. Übertraining (siehe Kapitel 3.5) lassen SVMs unter bestimmten Umständen als bevorzugte Klassifikatoren erscheinen (Abe, 2010; Burges, 1998; Mountrakis et al., 2010; Steinwart & Christmann, 2008).

In den Geowissenschaften gelten SVMs immer noch als weniger bekannt und selbstverständlich in ihrer Anwendung als beispielsweise Entscheidungsbäume (engl. *decision trees*) und die verschiedensten Varianten neuronaler Netzwerke sowie Fuzzy-Systeme (Heinert, 2010a). Eine Überprüfung, in welchem Ausmaß und unter welchen Voraussetzungen die in einschlägiger Literatur beschriebenen Vorteile auch für eine Verwendung von SVMs zur Landnutzungsklassifikation zutreffen, ist ein wesentliches Anliegen dieser Arbeit.

Bevor mit der Einführung in die Grundtypen von Support Vektor Maschinen begonnen werden kann, sei die allgemeine Problemstellung genauer definiert.

5.3 Allgemeine Problemstellung

In den nachfolgenden Beispielen sei stets als Ausgangssituation eine Menge von Daten angenommen. Mit den vollständigen Daten oder einer Untermenge von Daten, den sogenannten Trainingsdaten, wird die SVM trainiert, sodass sie zu einem späteren Zeitpunkt in der Lage ist, auf Basis der gelernten Klassifikationsregeln unbekannte Daten möglichst fehlerfrei zu klassifizieren.

Die in der Menge D enthaltenen Trainingsdaten $(\mathbf{x}_i, y_i)$ setzen sich aus M Datenpaaren $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)$ zusammen. $\mathbf{x}_i \in \mathcal{X}$ entsprechen den beobachteten Merkmalen (Eigenschaften), $\mathcal{X}$ bezeichnet den Eingangsraum. Jeder Datensatz $\mathbf{x}_i$ hat m Merkmale, die als Vektor $(x_1, \ldots, x_m)^\top$ im m-dimensionalen Eingangsraum $\mathcal{X} \subset \mathbb{R}^m$ interpretiert werden. Diese Vektoren adressieren je einen Datenpunkt. $\mathcal{Y}$ bezeichnet den Ergebnisraum, wobei $y_i \in \{-1, +1\}$ den beobachteten Klassenlabeln (Ausprägungen) entsprechen.

Vorerst wird nur eine binäre Klassifikation angenommen, das heißt, jedes Klassenlabel kann nur entweder $+1$ oder -1 sein. Mit den Mengen $D_+ = \{(\mathbf{x}, y) \in D : y = -1\}$ und $D_- = \{(\mathbf{x}, y) \in D : y = -1\}$ werden alle jene Daten zusammengefasst, deren Klassenlabel $+1$ oder -1 ist. Datenpunkte werden entsprechend dem Vorzeichen einer Funktion $y(\mathbf{x})$ des Klassifikators klassifiziert.

5.4 Lineare SVM für linear separierbare Muster

Als Einführung in Support Vector Machines wird exemplarisch die einfachste SVM diskutiert, die sogenannte *Hard-Margin Support Vector Machine*[2]. In folgenden Herleitungen sei angenommen, dass die Menge jener Punkte mit negativem Label $D_- = \{(\mathbf{x}_i, y_i) : y_i = -1\}$ und jene mit positivem Label $D_+ = \{(\mathbf{x}_i, y_i) : y_i = +1\}$ exakt linear separierbar sind und damit keine Klassifizierungsfehler zu berücksichtigen sind.

Hyperebene und Trennbereiche

Ausgangspunkt der folgenden Betrachtungen sind die in Kapitel 4.2 gezeigten Gleichungen zur Definition einer Hyperebene H mit dem Gewichtsvektor $\mathbf{w}$ und dem Bias b:

$$H(\mathbf{x}) \quad = \quad \mathbf{w}^\top \mathbf{x} + b = 0 \tag{5.1}$$

Nimmt man an, dass eine Hyperebene, definiert durch die Vektoren $\mathbf{w}$ und b zwei exakt linear separierbare Klassen Ω_1 und Ω_2 trennt, so gilt für alle Muster aus den Trainingsdaten

[2] in einigen Publikationen wird sie auch als *Maximal Margin* SVM bezeichnet

$\mathbf{X}_i$ für $i = 1, \ldots, M$ folgende Gleichung (Abe, 2010):

$$\mathbf{w}^\top \mathbf{x_i} + b = \begin{cases} > 0 & \text{für alle } \mathbf{x_i} \text{ mit } y_i = +1 \\ < 0 & \text{für alle } \mathbf{x_i} \text{ mit } y_i = -1 \end{cases} \tag{5.2}$$

Aufgrund der Annahme, dass die Trainingsdaten linear trennbar sind, ist vorerst der spezielle Fall $\mathbf{w}^\top \mathbf{x} + b = 0$ nicht zu berücksichtigen.

Maximierung der Marge

Die Aufgabe während des Trainings der SVM besteht nun darin, die Hyperebene H optimal durch die Datenpunktwolke zu legen, so dass der Abstand zwischen der Hyperebene und den Punkten der binären Klassen $+1$ und -1 maximal groß ist. Zu beiden Seiten um die zwischen den Stützvektoren der beiden Klassen liegenden optimalen Hyperebene soll sich ein in seiner Breite maximierter Trennbereich (engl. *margin*) befinden, der frei von Vektoren aus den Trainingsdaten ist (siehe Abbildung 5.1).

Aufgrund der Forderung, der Rand des zu maximierenden Trennbereichs zu beiden Seiten der Hyperebene solle exakt auf die Werte ± 1 fallen, wird (5.2) zu folgenden Ungleichungen umgeformt (Abe, 2010):

$$\mathbf{w}^\top \mathbf{x_i} + b = \begin{cases} \geq +1 & \text{für alle } \mathbf{x_i} \text{ mit } y_i = +1 \\ \leq -1 & \text{für alle } \mathbf{x_i} \text{ mit } y_i = -1 \end{cases} \tag{5.3}$$

Abbildung 5.1 zeigt zwei Trennfunktionen, die obige Bedingungen (5.3) erfüllen. Der maximierte Trennbereich lässt die SVM später auch unbekannte in diesem Bereich liegende

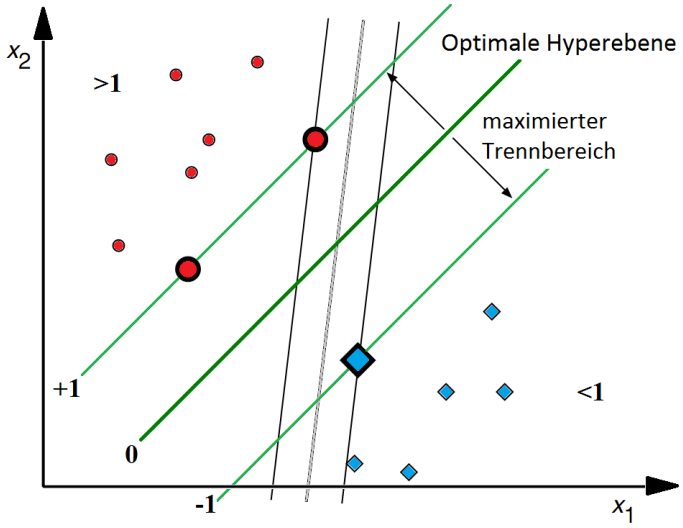


Abbildung 5.1: Optimal separierende Hyperebene auf drei Stützvektoren mit maximiertem Trennbereich. Adaptiert aus Abe (2010).

Vektoren noch korrekt zuordnen. Für diese innerhalb des Trennbereichs befindliche Muster gilt $-1 < \mathbf{w}^\top \mathbf{x_i} + b < +1$. Die Ungleichungen (5.3) multipliziert mit dem Resultat y_i lassen sich in den folgenden äquivalenten Ausdruck umformen (Burges, 1998):

$$y_i(\mathbf{w}^\top \mathbf{x_i} + b) \geq 1 \quad \text{für} \quad i = 1, \ldots, M \tag{5.4}$$

Die Umformung der Gleichung auf euklidische Entfernungen erfolgt - wie schon bei den Gleichungen (4.3) bis (4.7) gezeigt - durch eine Division mit der *Euklidschen Norm* $\|\mathbf{w}\|$ - siehe auch (4.4).

$$y_i \left(\frac{\mathbf{w}^\top}{\|\mathbf{w}\|} \mathbf{x_i} + \frac{b}{\|\mathbf{w}\|} \right) \geq \frac{1}{\|\mathbf{w}\|} \tag{5.5}$$

Aus obiger Gleichung erkennt man die *Hessesche Normalform* der Hyperebene.

Formulierung des Optimierungsproblems

Der Abstand der Stützvektorpunkte am Rand des Trennbereichs zur Hyperebene wird mit der Gleichung

$$\left| \frac{\mathbf{w}^\top}{\|\mathbf{w}\|} \mathbf{x_i} + \frac{b}{\|\mathbf{w}\|} \right| = \frac{1}{\|\mathbf{w}\|} = \|\mathbf{w}\|^{-1} \tag{5.6}$$

ermittelt. Die zu maximierende Breite des Trennbereichs ist mit dem doppelten Abstand der Hyperebene zu den Stützvektoren mit $2 \cdot \|\mathbf{w}\|^{-1}$ umgekehrt proportional zur Länge des Normalvektors. Der Forderung nach einem maximierten Trennbereich lässt sich nun durch die Formulierung eines Optimierungsproblems samt Nebenbedingungen darstellen (Bishop, 2006; Burges, 1998):

$$\text{Optimierungsproblem} \quad (\mathbf{w}^*, b^*) = \underset{\mathbf{w} \in \mathcal{F}, b \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{w}\|^2 \tag{5.7}$$

$$\text{mit der Bedingung} \quad y_i(\mathbf{w}^\top \mathbf{x_i} + b) - 1 \geq 0 \quad \text{für alle} \quad i = 1, \ldots, M \tag{5.8}$$

Die mit (5.8) vorweggenommenen notwendigen Nebenbedingungen für $\mathbf{w}$ und b bedürfen noch weiterer Erläuterungen.

Die geometrische Deutung des Optimierungsproblems $\frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w}^\top \mathbf{w}$ belegt, dass sich die Hyperebene, wie in Abbildung 5.2.a dargestellt, damit immer weiter von den Trainingsdaten entfernen würde, ohne sie voneinander zu trennen. Weitere Randbedingungen müssen also die gesuchte Hyperebene zwischen die zu trennenden Klassen zwingen.

Realisiert wird dies durch eine zusätzliche Forderung, wonach alle orthogonalen Abstände

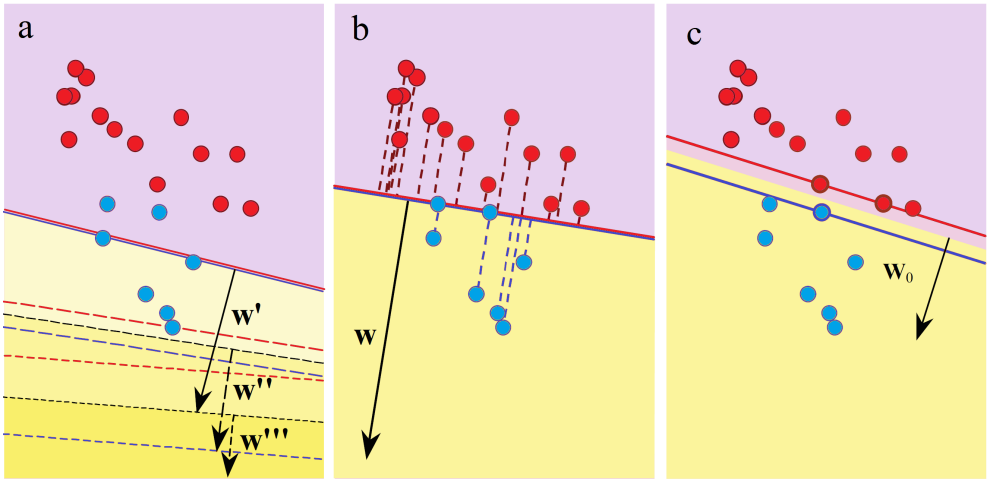


Abbildung 5.2: Drift der Hyperebene aufgrund der Optimierungsbedingungen: (a) bei uneingeschränkter Minimierung von $\mathbf{w}^\top\mathbf{w}$ Abdrift über die Muster hinweg; (b) bei der Maximierung aller $y_i(\mathbf{w}^\top\mathbf{x_i} - b) - 1$ wandert die Hyperebene an die Seite der kleineren Stichprobe; (c) bei Kombination beider Bedingungen bildet sich ein Trennbereich zwischen den beiden Klassen aus. Adaptiert aus Heinert (2010a).

auf beiden Seiten des Trennbereichs gemeinsam zu maximieren sind (Heinert, 2010a):

$$\sum_i d_i = \sum_{i=0}^{M} y_i \left(\frac{\mathbf{w}^\top}{\|\mathbf{w}\|}\mathbf{x_i} + \frac{b}{\|\mathbf{w}\|} \right) - \frac{1}{\|\mathbf{w}\|} \tag{5.9}$$

Das Ergebnis dieser Bedingung ist in Abbildung 5.2.b dargestellt. Die Hyperebene liegt nun zwischen den beiden Klassen an der Seite der kleineren Stichprobe. Durch das Zusammenwirken der Minimierung und der Maximierungsbedingung wandert die Hyperebene letztendlich zwischen die Punkte der beiden Klassen und bildet einen Trennbereich aus, siehe Abbildung 5.2.c. Nur wenn die Klassenzuordnung y_i aller Trainingsvektoren korrekt sind, andernfalls wäre das Resultat $d_i \leq -1$, und wenn keiner der Trainingsvektoren in den Trennbereich fällt, andernfalls wäre das Resultat $-1 < d_i < 0$, ist jede Bedingung für sich erfüllt (Heinert, 2010a) und die **Nebenbedingung** zu (5.7) kann auch ohne die Normierung $\|\mathbf{w}\|$ entsprechend (5.8) formuliert werden.

Mit Hilfe obiger Herleitungen und der Abbildung 5.1 lassen sich die Kriterien zur Bestimmung der optimalen Hyperebene $H : D(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b = 0$ für linear separierbare Trainingsdaten zweier Klassen festlegen (Niemann, 2003). Mit der Minimierungsbedingung (5.7) und den aus der Anzahl der Trainingspaare resultierenden M Nebenbedingungen (5.8) ist das Optimierungsproblem vollständig definiert.

Lösung des Optimierungsproblems

Die Lösung des quadratischen Optimierungsproblems bedarf jedoch einiger Vereinfachungen, um den numerischen Aufwand und Speicherbedarf in praktikable Bereiche zu bringen (Platt, 1999). Zur Vereinfachung des Optimierungsproblems werden positive *Lagrange-Multiplikatoren* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)$ eingeführt und jede der Nebenbedingungen (5.8) damit multipliziert:

$$\alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x_i} + b) - 1 \right) = 0 \tag{5.10}$$

Damit ergibt sich die Formulierung des Optimierungsproblems aus (5.7) und (5.8). Die *Lagrange-Gleichung* (Bishop, 2006)

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=0}^{M} \alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x_i} + b) - 1 \right) \tag{5.11}$$

ist bezüglich $\mathbf{w}$ zu minimieren, bezüglich b in Abhängigkeit des Vorzeichens von $\sum_{i=1}^{M} \alpha_i y_i$ zu minimieren oder zu maximieren, und bezüglich α_i zu maximieren. Dazu werden die Ableitungen nach $\mathbf{w}$ und b jeweils 0 gesetzt. Wir finden nunmehr die Lösung der unter (5.11) formulierten *Lagrange-Gleichung* (Abe, 2010)

$$\text{unter den Bedingungen} \quad \frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{M} \alpha_i y_i \mathbf{x_i} = 0 \tag{5.12}$$

$$\text{und} \quad \frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{b}} = \sum_{i=1}^{M} \alpha_i y_i = 0 \tag{5.13}$$

$$\text{und} \quad \alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x_i} + b) - 1 \right) = 0 \quad \text{für} \quad i = 1, \ldots, M \tag{5.14}$$

$$\text{und} \quad \alpha_i \geq 0 \quad \text{für} \quad i = 1, \ldots, M \tag{5.15}$$

Die *Karush-Kuhn-Tucker-Bedingungen* (KKT-Bedingungen) garantieren die Optimalität der Lösung und ordnen jeder relevanten Nebenbedingung ein $\alpha_i \geq 0$ zu, die nicht scharfen Nebenbedingungen gehen aufgrund von $\alpha_i = 0$ nicht mehr in die Lösung ein (Pfeifenberger, 2010). Durch Substitution der Gleichungen (5.12) und (5.13) in die Lagrange-Gleichung (5.11) erhält man die Gesamtlösungsgleichung in dualer Form:

$$\text{Maximiere} \quad L(\boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \tag{5.16}$$

$$\text{unter den Bedingungen} \quad \sum_{i=1}^{M} \alpha_i y_i = 0 \tag{5.17}$$

$$\text{und} \quad \alpha_i \geq 0 \quad \text{für alle} \quad i = 1 \ldots M \tag{5.18}$$

Das vorliegende Optimierungsproblem ist gegenüber der Gleichung (5.9) mit den Nebenbedingungen (5.8) stark vereinfacht. Zur Lösung des nunmehr *konvexen, quadratischen Optimierungsproblems* (Cristianini & Ricci, 2008; Bishop, 2006; Burges, 1998) gibt es eine Reihe numerischer Verfahren, die an dieser Stelle jedoch nicht weiter diskutiert werden. Hierfür sei auf weiterführende Literatur verwiesen. Heinert (2010a) und Bishop (2006) empfehlen beispielsweise die dafür oft verwendete *sequentielle Minimierungsoptimierung* (engl. *Sequential Minimal Optimization* – SMO) von Platt (1999). Die Autoren Chang & Lin (2011) verweisen ebenso auf einige in der Praxis bewährte Verfahren, die teilweise auch in der Implementierung der frei verfügbaren Bibliothek LibSVM (siehe Kapitel 6.1) umgesetzt wurden.

Anwendung der Entscheidungsfunktion

Aus den Gleichungen (5.16) bis (5.18) lassen sich mit den Trainingsvektoren die optimalen Lagrange-Multiplikatoren $\alpha_1^*, \ldots, \alpha_M^*$ berechnen. Da aus Gleichung (5.16) bzw. (5.11) folgt, dass infolge der Lagrange-Multiplikatoren α_i die Vektoren selektiv und nicht gleichermaßen an der Lösung beteiligt sind, sind jene Daten, die ein positives $\alpha_i > 0$ aufweisen, in der Menge der Support Vector Indizes S, wohingegen alle anderen Vektoren $\alpha_i = 0$ keinen Einfluss auf das Modell nehmen (Abe, 2010). In der Praxis werden nur $\alpha_i > \varepsilon$ verwendet, wobei ε ein kleiner positiver Wert ist (vgl. Kapitel 7.9.5). In einem weiteren Schritt wird mit $S = \{i : \alpha_i > \varepsilon\}, |S| \ll |X|$ entsprechend Gleichung (5.12) der optimale Gewichtsvektor $\mathbf{w}^*$ ermittelt (Burges, 1998):

$$\mathbf{w}^* = \sum_{i=1}^{M} \alpha_i^* y_i \mathbf{x}_i \tag{5.19}$$

Setzt man die Gleichung des optimalen Gewichtsvektors $\mathbf{w}^*$ (5.19) in die ursprüngliche Gleichung der Hyperebene (5.1), so ergibt sich die Entscheidungsfunktion (Abe, 2010)

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i^* y_i \mathbf{x}_i^\top \mathbf{x} + b \tag{5.20}$$

Den Bias b erhält man aus einem beliebigen Support Vector $\mathbf{x}_i$ mit der Gleichung $b = y_i - \mathbf{w}^\top \mathbf{x}_i$ für $i \in S$ oder, wie Abe (2010) ausdrücklich vermerkt, aufgrund möglicher numerischer Ungenauigkeiten vorzugsweise aus dem Mittelwert aller Stützvektoren.

$$b^* = \frac{1}{|S|} \sum_{i \in S} \left(y_i - \mathbf{w}^\top \mathbf{x}_i \right) \tag{5.21}$$

Die Klassifizierungsregel für zwei Klassen Ω_1 und Ω_2 lautet

$$Class(\mathbf{x}) = \begin{cases} \Omega_1 & \text{falls} \quad D(\mathbf{x}) > 0 \\ \Omega_2 & \text{falls} \quad D(\mathbf{x}) < 0 \end{cases} \tag{5.22}$$

Für die schrittweise Herleitung obiger Gleichungen und den Verweis auf weiterführende Literatur wird das Studium von Abe (2010), Bishop (2006), Heinert (2010a) und Niemann (2003) empfohlen.

5.5 Lineare SVM für nicht linear separierbare Muster

Bisher wurde davon ausgegangen, dass die Trainingsdaten exakt linear separierbar sind. In der Praxis ist dies jedoch kaum der Fall, da die Trainingsdaten einem gewissen Fehler durch Rauschen unterliegen und damit Falschklassifizierungen mit einer *harten* Klassifizierungsgrenze kaum zu vermeiden sind. Mit einer einfachen Modifikation der in Abschnitt 5.4 eingeführten Vorgehensweise kann eine gelegentliche Falschklassifizierung zugelassen werden, obwohl die Trainingsmuster a priori linear untrennbar wären.

Aufweichung der Trennbereiche

Die Klassifizierungsgrenzen werden „aufgeweicht“, man spricht deshalb auch von einer *Soft-Margin Support Vector Machine*. Dazu werden positive Schlupfvariablen ξ_i (engl. *slack variables*) eingeführt. Sie erfüllen die Funktion von Fehlertermen und geben die Abweichung des jeweiligen Vektors $\mathbf{x}_i$ von der Trennfläche an (siehe Abbildung 5.3).

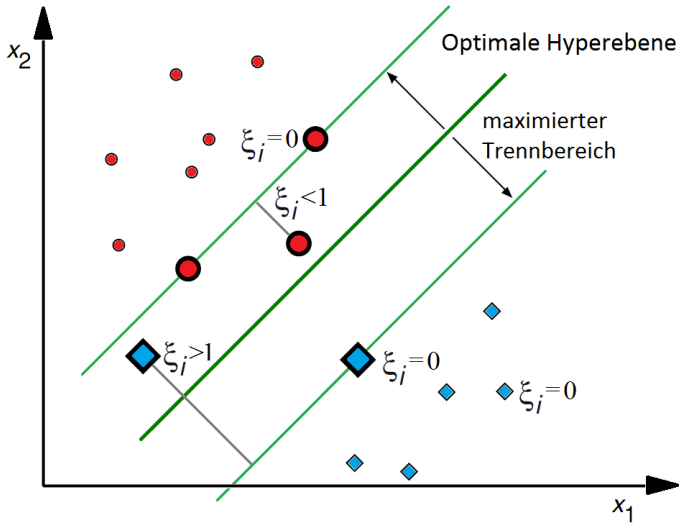


Abbildung 5.3: Anwendung einer Soft-Margin bei nicht-trennbaren Trainingsdaten. Adaptiert aus Abe (2010).

Diese zusätzlichen Schlupfvariablen schwächen die Nebenbedingungen (5.3) derart ab, dass Trainingsvektoren auch innerhalb des maximierten Trennbereichs liegen können. Für korrekt klassifizierte Vektoren gilt $\xi_i = 0$. Für Vektoren, die auf der korrekten Seite der Hyperebene innerhalb des Trennbereichs liegen, gilt $0 < \xi_i \leq 1$. Für jene Vektoren die

innerhalb des Trennbereichs, aber schon auf der falschen Seite der Hyperebene liegen, gilt $\xi_i > 1$. Sie werden auch als falsch klassifiziert gewertet (Bishop, 2006). Die Abschätzung des gesamten Fehlers für alle Vektoren beträgt demnach $\sum_i \xi_i$, entsprechend erscheint eine Anpassung der Minimierungsbedingung (5.7) auf

$$(\mathbf{w}^*, b^*, \xi^*) = \underset{\mathbf{w}\in\mathcal{F}, b\in\mathbb{R}, \xi\in\mathbb{R}^M}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{p}\sum_{i=1}^{M}\xi_i^p \tag{5.23}$$

durchaus sinnvoll. Der abstrakte Parameter $C > 0$ reguliert den Einfluss der Schlupfvariablen und wird so zur Stellgröße bei Soft-Margin SVMs. Mit dem als Fehlergewicht interpretierten C kann sowohl die Größe des Trennbereichs als auch die Anzahl der Trainingsfehler reguliert werden. Der Parameter p wird üblicherweise auf 1 oder 2 fixiert. Größere Werte sind möglich, Berechnungen werden bei zunehmendem p jedoch komplexer. In den folgenden Herleitungen wird von $p = 1$ ausgegangen. Derartige SVMs werden als „L1 Soft-Margin Support Vector Maschine“ bezeichnet[3] (Abe, 2010).

Abbildung A.1 auf Seite 113 verdeutlicht die Abhängigkeit der Breite des Trennbereichs vom Fehlergewicht C in einigen Beispielen und zeigt die Auswirkungen bei linear separierbaren (Beispiel a und b) und linear nicht separierbaren Mustern (Beispiel c und d). Wird C größer und damit der Trennbereich schmäler, so vergrößert sich die Komplexität des Klassifikators durch die Tatsache, dass mehr Lagrange-Multiplikatoren $\alpha_i > 0$ werden und damit $|S|$ größer wird. Dabei besteht die Gefahr des Overfittings. Der Fall $C \to \infty$ entspricht dem im Abschnitt 5.4 eingeführten Fall für linear separierbare Muster. Wird ein kleines C gewählt und damit der Trennbereich breiter, so leidet die Generalisierungsfähigkeit des Klassifizierers, was wiederum zum Underfitting führen kann (Bishop, 2006; Cristianini & Taylor, 2000; Manning et al., 2009).

Anpassung des Optimierungsproblems

Aus obigen Anpassungen ergeben sich die geänderten Bedingungen (Niemann, 2003):

$$\begin{aligned} \mathbf{w}^\top\mathbf{x}_i + b &\geq +1 - \xi_i \quad \text{für} \quad y_i = +1 \\ \mathbf{w}^\top\mathbf{x}_i + b &\leq -1 + \xi_i \quad \text{für} \quad y_i = -1 \end{aligned} \tag{5.24}$$

$$\begin{aligned} \text{und somit} \quad y_i(\mathbf{w}^\top\mathbf{x}_i + b) &\geq 1 - \xi_i \quad \text{für} \quad i = 1, \ldots, M \\ \xi_i &\geq 0 \quad \text{für} \quad i = 1, \ldots, M \end{aligned} \tag{5.25}$$

Mit der Einführung zusätzlicher Lagrange-Multiplikatoren β_i wird verhindert, dass die Schlupfvariablen negativ werden. Die vollständige Lagrange-Gleichung lautet nun (Nie-

[3] $p = 1$ kommt am häufigsten zum Einsatz. Wenn $p = 2$, werden diese SVMs folgerichtig mit „L2 Soft-Margin Support Vector Maschine“ bezeichnet (Abe, 2010, Seite 56-59).

MANN, 2003)

$$L(\mathbf{w}, b, \phi, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=0}^{M} \xi_i - \sum_{i=0}^{M} \alpha_i \left(y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i \right) - \sum_{i=0}^{M} \beta_i \xi_i \quad (5.26)$$

Die Lösung des Optimierungsproblems erfolgt im Prinzip gleich wie schon in Kapitel 5.4 mit den Gleichungen (5.12) bis (5.16). Einzig die partielle Ableitung nach $\mathbf{w}$ (5.12)und nach b (5.13) und die zusätzlich erforderliche partielle Ableitung nach ξ für $\xi = (\xi_1, \ldots \xi_M)$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha}, \xi)}{\partial \xi} = C - \alpha_i - \beta_i = 0 \quad (5.27)$$

führen gemeinsam mit den KKT-Bedingungen zur Änderung der Bedingung (5.18) auf die Bedingung (BURGES, 1998)

$$0 \leq \alpha_i \leq C \quad \text{für alle} \quad i = 1 \ldots M \quad (5.28)$$

Für eine schrittweise Herleitung der Karush-Kuhn-Tucker-Lösungsbedingungen sei wieder auf BURGES (1998) oder ABE (2010) verwiesen.

Anwendung der Entscheidungsfunktion

Zur Schätzung des Parameters C werden vorzugsweise heuristische Verfahren genutzt. Indikatorgrößen sind im Wesentlichen die Anzahl der Support-Vektoren und die sogenannte VC-Dimension (VAPNIK, 1998) des Modells, ein numerisches Maß, das die Komplexität des Klassifikators bewertet (HEINERT, 2010a).

Der optimale Gewichtsvektor $\mathbf{w}^*$ kann, wie jener linear trennbarer Trainingsvektoren mit Gleichung (5.19) berechnet werden. Der Bias b^* jedoch kann nicht mehr aus einem beliebigen Stützvektor abgeleitet werden, da dessen Werte y_i einer Streuung unterliegen können. Aus diesem Grund muss der optimale Bias b^* nun aus dem Mittelwert aller *unbounded* Support Vector Indizes $U = \{i : 0 < \alpha_i < C\}$ berechnet werden[4] (ABE, 2010).

$$b^* = \frac{1}{|U|} \sum_{i \in U} \left(y_i - \mathbf{w}^\top \mathbf{x}_i \right) \quad (5.29)$$

Sowohl die Entscheidungsfunktion (5.20) als auch die Klassifizierungsregel (5.22) für zwei Klassen Ω_1 und Ω_2 sind gleich denen einer Hard-Margin Support Vector Machine.

[4] Im Gegenzug werden als *bounded support vectors* jene Stützvektoren bezeichnet, für die $\alpha_i = C$ gilt.

5.6 Kernel-SVM für nicht linear separierbare Muster

Die bisherigen Betrachtungen in den vorhergehenden Abschnitten und die Herleitung der Klassifikatoren unter Verwendung von Hyperebenen als Trennfunktionen gingen davon aus, dass die Muster exakt linear trennbar (Abschnitt 5.4) oder zumindest annähernd linear trennbar (Abschnitt 5.5) sind. Betrachtet man die zugrundeliegenden Gleichungen beider Fälle, so erkennt man, dass sich die Beschränkung auf Hyperebenen nicht zwingend notwendig ist. Da immer nur Skalarprodukte der Merkmalsvektoren berechnet werden müssen, können die Gleichungen einfach auf Trennflächen höherer Ordnung angepasst werden.

Transformiert man den Merkmalsvektor $\mathbf{x} \in \mathbb{R}^m$ mit einer nichtlinearen *Abbildungsfunktion* $\phi(\mathbf{x}) = (\phi(x_1), \ldots, \phi(x_m))^\top$ in einen höherdimensionalen Raum $\tilde{\mathbf{x}} \in \mathbb{R}^l$ mit $l > m$, so bleiben die Gleichungen weiterhin gültig, werden anstatt mit $\mathbf{x}$ nun aber mit $\phi(\mathbf{x})$ berechnet. Die Gleichung der Hyperebene $H(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ wird nun zu einer allgemein gültigen Gleichung einer nicht linearen Trennebene umformuliert (Niemann, 2003).

$$H'(x) = \mathbf{w}^\top \phi(\mathbf{x}) + b \tag{5.30}$$

Einen Vorteil hat diese Merkmalsabbildung (engl. *feature mapping*) dann, wenn in diesem höherdimensionalen Raum eine lineare Trennung der transformierten Merkmalsvektoren durchgeführt werden kann, denn das *Theorem von Cover* (Scholkopf & Smola, 2001) besagt, dass durch die Transformation von Daten in einen höherdimensionalen Raum die Anzahl der möglichen linearen Trennebenen immer erhöht wird. Ein einfaches Beispiel in Abbildung 5.4 soll dies verdeutlichen.

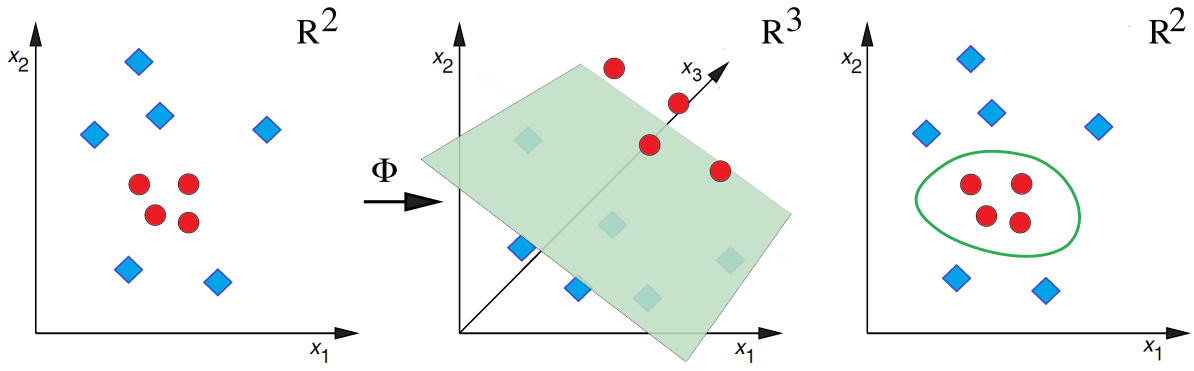


Abbildung 5.4: Lineare Trennung der Datensätze durch Transformation in einen höherdimensionalen Raum: nicht linear separierbare Daten (links) werden in einen höherdimensionalen Raum transformiert (Mitte), in dem sie dann linear separierbar sind. Die Hyperebene im höhertransformierten Raum entspricht einer nicht linearen Trennfunktion im Eingangsraum (rechts).

Abbildung 5.4 links zeigt die Punkte der beiden Klassen Ω_1 und Ω_2, die einen zweidimensionalen Raum $\mathbb{R}^2$ aufspannen. Egal, wie eine Hyperebene H durch die Punkte gelegt werden würde, in $\mathbb{R}^2$ sind die beiden Klassen nicht linear separierbar. Erweitert man nun den Merkmalsraum mithilfe einer Abbildungsfunktion $\phi(\mathbf{x})$ um eine weitere geeignete, dritte Dimension, so können dadurch die Datenpunkte in $\mathbb{R}^3$ mithilfe einer Hyperebene (Abbildung 5.4 Mitte) getrennt werden. Durch die Transformation entspricht die Hyperebene im höhertransformierten Raum einer nicht linearen Trennfunktion im Eingangsraum (rechte Abbildung 5.4).

Umgelegt auf die Erkenntnisse der Abschnitte 5.4 und 5.5 müsste die Entscheidungsfunktion (5.20) ebenfalls in den neuen höherdimensionalen Abbildungsraum $\mathbb{R}^l$ überführt werden.

$$D(\mathbf{x}) = \sum_{i \in S} \alpha_i^* y_i \phi(\mathbf{x_i})^\top \phi(\mathbf{x}) + b \tag{5.31}$$

Der wesentliche Nachteil einer derartigen Lösung wird klar, wenn man obige Gleichung (5.31) betrachtet. Die Berechnung des Skalarprodukts zur Transformation großer Datenmengen in einen höherdimensionalen Raum ist aufwendig und teuer. Zudem ist die Vorhersage einer Abbildungsfunktion ϕ, die eine lineare Separierbarkeit im höherdimensionalen Raum $\mathbb{R}^{\tilde{d}}$ möglich machen soll, nicht ohne Weiteres möglich (Burges, 1998).

Deutlich erfolgversprechender ist stattdessen die Hintransformation der Trainingsdaten $\mathbf{x} \in \mathcal{X}$ in einen höherdimensionalen Merkmalsraum $\mathcal{F}$, die Bestimmung der trennenden Hyperebene H' in diesem temporären Merkmalsraum, und die Rücktransformation der Hyperebene vom Merkmalsraum $\mathcal{F}$ in den Zustandsraum $\mathcal{X}$ der aktuellen Trainingsdaten (Abbildung 5.4 rechts). Diese Vorgangsweise klingt vorweg noch deutlich rechenintensiver, sie lässt sich jedoch stark vereinfachen. Dazu bedient man sich eines „Tricks“ mit sogenannten *Kernelmethoden*[5].

5.7 Kernelmethoden

SVMs werden den sog. Kernelmethoden (engl. *kernel methods*) zugerechnet, da sie sich eines Kerns (engl. *kernel*) bedienen (Abe, 2010). Treten in einem beliebigen Algorithmus die Skalarprodukte transformierter Vektoren $\phi(\mathbf{x})^\top \phi(\mathbf{z})$ auf, so können diese unter bestimmten Umständen auf Kernelevaluierungen untransformierter Vektoren zurückgeführt werden. Diese Vorgangsweise lässt sich immer dann anwenden, wenn eine *Kernfunktion* $\mathcal{K}$ existiert, die wie folgt definiert ist (Steinwart & Christmann, 2008):

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{z})^\top \phi(\mathbf{x}) \tag{5.32}$$

[5] Manchmal auch *Kernelfunktionen* oder *Kernfunktionen* genannt

Die Kernfunktion $\mathcal{K}(\mathbf{x}, \mathbf{z})$ ist äquivalent dem Skalarprodukt[6] der Abbildungsfunktion $\phi(\mathbf{z})^\top \phi(\mathbf{x})$, obwohl ϕ in $\mathcal{K}$ nicht explizit berechnet werden muss. Die Abbildungsfunktion $\phi : \mathcal{X} \to \mathcal{F}$ transformiert die Eingangsvektoren $(\mathbf{x}, \mathbf{z}) \in \mathcal{X}$ und $\mathcal{X} \subset \mathbb{R}$ oder $\mathbb{C}$ vom Eingangsraum $\mathcal{X}$ in den höherdimensionalen Merkmalsraum $\mathcal{F}$.

Welche Eigenschaften eine Funktion besitzen muss, damit sie als Kernfunktion geeignet ist, wird durch das *Theorem von Mercer* (Niemann, 2003) beschrieben. Es erklärt, dass eine Funktion dann für einen Kernel geeignet ist, wenn sie stetig, symmetrisch und positiv semidefinit (engl. *positive semi-definite* – PSD) ist (Graf, 2010). Die Einführung ausgewählter Kernel im folgenden Abschnitt 5.7.1 wird jedoch am Beispiel des Sigmoid-Kernels schon zeigen, dass vermehrt auch Kernel zum Einsatz kommen, die nur bedingt positiv definit (engl. *conditionally positive definite* – CPD) sind. Dabei handelt es sich um Kernel, die nur in bestimmten Funktionsbereichen die notwendigen Eigenschaften einer Kernfunktion mitbringen, außerhalb dieser Bereiche aber nicht als Kernfunktion geeignet sind (Boughorbel et al., 2005). Diese Einschränkung ist bei der Parametrisierung der Kernel zu beachten.

Ein Beispiel soll die Funktionsweise einer PSD-Kernfunktion verdeutlichen. Angenommen sei die Kernfunktion $\mathcal{K}(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$ für $\mathbf{X} \subseteq \mathbb{R}$ im 2-dimensionalen Eingangsraum mit $\mathbf{x} = (x_1, x_2)$ und $\mathbf{z} = (z_1, z_2)$ für $(\mathbf{x}, \mathbf{z}) \in \mathcal{X}^2$. Die Gleichung wird schrittweise in ein Skalarprodukt umgeformt:

$$\begin{aligned}
\mathcal{K}(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^\top \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 && (5.33)\\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 && (5.34)\\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^\top && (5.35)\\
&= \phi(\mathbf{x})^\top \phi(\mathbf{z}) && (5.36)
\end{aligned}$$

Wir sehen im direkten Vergleich von (5.35) und (5.36), dass die Transformationsfunktion ϕ die Form $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$ bzw. $\phi(\mathbf{z}) = (z_1^2, \sqrt{2} z_1 z_2, z_2^2)$ hat. Am Ergebnis der Umformung erkennen wir weiters, dass durch die richtige Wahl der Kernfunktion $\mathcal{K}(\mathbf{x}, \mathbf{z})$ ein äquivalentes Skalarprodukt $\phi(\mathbf{x})^\top \phi(\mathbf{z})$ entstanden ist, womit die Forderung aus Gleichung (5.32) erfüllt ist.

Der Kunstgriff, Gleichungen mit geeigneten Skalarprodukten drastisch zu vereinfachen und deren Berechnung zu beschleunigen, besteht nun darin, durch eine Substitution das aufwendige Transformieren und hochdimensionale Skalarprodukt durch eine vergleichsweise einfache Kernfunktion zu ersetzen, ohne dass das Ergebnis der Funktion damit

[6] Steinwart & Christmann (2008) weisen auf Seite 112 explizit darauf hin, dass die Reihenfolge der Faktoren vor allem bei $\mathcal{X} \subseteq \mathbb{C}$ relevant ist, da das Skalarprodukt in einem komplexen Vektorraum nicht kommutativ ist. Im reellen Fall ist die Reihenfolge nicht relevant. Dies ist wohl auch der Grund dafür, dass ein Großteil der Literatur die naheliegendere, wenngleich nicht ganz korrekte Schreibweise $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$ ausweist.

verändert wird. Mit diesem *Kernel-Trick* (Burges, 1998) kann die Hyperebene direkt aus dem Eingangsraum der Trainingsdaten berechnet werden, ohne die Transformation selbst ausführen zu müssen. Vielmehr erfolgt die Hin- und Rücktransformation implizit in der Kernelfunktion, die als Repräsentant der Hyperebene im höherdimensionalen Merkmalsraum betrachtet werden kann (Heinert, 2010a). Der gänzliche Verzicht auf das Skalarprodukt hat auch zur Folge, dass die Abbildungsfunktion ϕ selbst zu keinem Zeitpunkt der Berechnung bekannt sein muss (Abe, 2010) und in der Praxis oftmals auch nicht bekannt ist.

5.7.1 Ausgewählte Kernel

Einer der großen Vorteile der Support Vector Machines ist die Nutzung von Kernel zur Steigerung der Generalisierungsleistung und als positives Nebenprodukt auch zur Steigerung der Klassifizierungsgeschwindigkeit. Die Auswahl des Kernels und die Entwicklung neuer und auf die jeweiligen Anwendungen noch passenderen Kernfunktionen kommen dabei eine zentrale Bedeutung zu. So dokumentieren auch aktuelle Quellen (Abe, 2010; Bishop, 2006; Steinwart & Christmann, 2008; Souza, 2010) die stetige Entwicklung in diesem Bereich.

Mittlerweile gibt es eine Vielzahl unterschiedlicher Kernel, die in SVMs eingesetzt werden. Allein Souza (2010) listet 25 unterschiedliche Kernelmethoden und diskutiert deren Eigenschaften. Die Wahl des am besten geeigneten Kernels hängt im hohen Maße von den Eingangsdaten und der Parametrisierung der Kernel ab. Die Feinabstimmung der Parameter kann jedoch schnell zu einer langwierigen und mühsamen Aufgabe werden. Die automatische Auswahl eines Kernels ist trotzdem teilweise möglich, wie Howley & Madden (2005) diskutieren.

Im Folgenden werden ausgewählte Kernel gezeigt, die aufgrund ihrer Generalisierungsleistung, aber auch wegen ihrer „Gutmütigkeit“ vermehrt zur Anwendung kommen. Für die folgenden Beispielkernel sei $\mathcal{X} \subset \mathbb{R}^M$ angenommen. Liegen die Eingabedaten nicht in $\mathcal{X} \subset \mathbb{R}^M$, so können die Daten in einen Merkmalsraum $\mathcal{F} \subset \mathbb{R}^M$ abgebildet werden. Wenn nicht anders referenziert, wurde Abe (2010) als Quelle herangezogen.

Linear-Kernel

Der lineare Kernel ist der einfachste aller Kernel und wie folgt definiert:

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \quad = \quad \mathbf{x}^\top \mathbf{z} \tag{5.37}$$

Er wird immer dann verwendet, wenn die Trainingsdaten auf deren lineare Separierbarkeit überprüft werden sollen, oder wenn das Klassifikationsproblem linear separierbare

Trainingsdaten aufweist. Sollten die Daten linear trennbar sein, so sollte darauf verzichtet werden, die Daten in einen höherdimensionalen Merkmalsraum zu transformieren. Dies würde bei ungünstiger Parametrisierung nur eine Überanpassung der SVM fördern.

Polynom-Kernel

Ein Polynom-Kernel wird häufig verwendet, um einfache nichtlineare Klassifizierungsprobleme zu lösen. Der Kernel ist wie folgt definiert:

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \quad = \quad (\mathbf{x}^\top \mathbf{z} + c)^d \tag{5.38}$$

Indem $c = 0$ gesetzt wird, kann ein homogener Polynom-Kernel erzeugt werden. Wird zudem $d = 1$ gesetzt, so entspricht er dem Linear-Kernel. Der Polynom-Kernel entspricht Mercers Bedingungen. Nachteilig bei Polynom-Kernel ist die Tatsache, dass schon zwei Parameter c und d während des Trainings optimiert werden müssen, was den Trainingsvorgang verlangsamt. Kritisch können Polynom-Kernel dann werden, wenn der Grad d der Funktion übermäßig erhöht wird. Dann tendiert der Klassifikator leicht zum Overfitting.

Sigmoid-Kernel

Sigmoid-Kernel - manchmal auch als Hyperbolic Tangent Kernel bezeichnet - finden in Bereichen Anwendung, in denen sonst Klassifikatoren auf Basis neuronaler Netze Fuß gefasst haben. In der Tat sind SVMs, die Sigmoid Kernel verwenden, äquivalent einem zweilagigen Perzeptron[7] (Niemann, 2003). Der Sigmoid-Kernel ist wie folgt definiert:

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \quad = \quad \tanh(\kappa \mathbf{x}^\top \mathbf{z} + \theta) \tag{5.39}$$

Die Gefahr bei Sigmoid-Kernel ist, dass bei bestimmten Konfigurationen der Optimierungsalgorithmus nicht konvergiert. Der Grund dafür ist, dass Sigmoid-Funktionen die Bedingungen Mercers nur bei einer bestimmten Parametrisierung von κ und θ einhalten, denn Sigmoid-Kernel sind nur bedingt positiv definit (Vapnik, 1995). Für nähere Details sei auf Lin & Lin (2003) verwiesen, die zeigen, dass nur wenn $\kappa > 0$ und $\theta < 0$ und θ klein genug ist, $\mathcal{K}$ bedingt positiv definit ist und dass Sigmoid-Kernel in ihrem Verhalten Ähnlichkeiten zu RBF-Kernel aufweisen können.

RBF-Kernel

Eine radiale Basisfunktion (engl. *Radial Basic Function* – RBF) ist eine reelle Funktion, deren Wert ausschließlich die Distanz $\phi(\mathbf{x}) = \|\mathbf{x}\|$ vom Punkt $\mathbf{x}$ zum Nullpunkt oder mit $\phi(\mathbf{x}) = \|\mathbf{x} - \mathbf{z}\|$ von $\mathbf{x}$ zu einem beliebigen anderen Punkt $\mathbf{z}$ angibt. Kernel mit radialer

[7] Als Perzeptron versteht man ein vereinfachtes künstliches neuronales Netz.

Basisfunktion – kurz als RBF-Kernel bezeichnet – können in der Form

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \quad = \quad f(\, d(\mathbf{x}, \mathbf{z})\,) \tag{5.40}$$

geschrieben werden, wobei hier d eine Metrik[8] auf $\mathcal{X}$ und f eine Funktion in $\mathbb{R}_0^+$ ist. Üblicherweise leitet sich eine Metrik aus der Norm $d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|$ ab (Scholkopf & Smola, 2001).

RBF-Kernel weisen die besondere Eigenschaft aus, dass Trennebenen nicht zusammenhängend sein müssen. Beispiele für RBF-Kernel sind sogenannte B-Spline-Kernel (Bishop, 2006) oder die noch häufiger verwendeten **eRBF-Kernel** bzw. **Gauss-Kernel**, wie beispielsweise

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) \quad = \quad \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{z}\right\|^2\right) \tag{5.41}$$

für $\gamma > 0$. Oft wird der Gauss-Kern mit $\gamma = 1/(2\sigma^2)$ für $\sigma \neq 0$ parametrisiert (Scholkopf & Smola, 2001).

Im Allgemeinen gehören RBF-Kernel aufgrund ihrer positiven Eigenschaften zur ersten Wahl, wenn nicht-lineare Trainingsdaten vorliegen. Der Grund dafür ist die vergleichsweise einfache Parametrisierung der SVM und deren „gutmütiges“ Verhalten. Bei genauerer Untersuchung können lineare Kernel als Spezialfälle von entsprechend parametrisierten RBF-Kernel interpretiert werden. So nähern sich Gauss-Kernel mit zunehmendem σ in ihrem Trennverhalten immer mehr linearen Kernel an (Keerthi & Lin, 2003). Zudem verhält sich ein Sigmoid-Kernel mit entsprechender Parametrisierung wie ein RBF-Kernel (Lin & Lin, 2003). Hsu et al. (2010) legen die Verwendung von RBF-Kernel nahe, um erste Klassifikationsversuche durchzuführen, wenn keine Erfahrungswerte für bestimmte Eingangsdaten vorliegen.

5.7.2 Konstruktion eigener Kernel

Zur Konstruktion eigener Kernel sei erwähnt, dass sich Kernel ähnlich einem Baukastensystem auch durch die Kombination bestehender gültiger Kernel bilden lassen. Anhang A auf Seite 116 zeigt eine Liste gängiger Kernel-Bausteine, die in Kombination wiederum einen gültigen Kernel ergeben. Bishop (2006) zeigt auch einige Beispiele dazu. Eine ausführliche Diskussion des Kernel-Engineerings ist ebenso in Shawe-Taylor & Cristianini (2004) zu finden.

[8] Unter Metrik versteht man in der Mathematik eine semipositive, symmetrische, der Dreiecksungleichung genügende Abstandsfunktion.

6 Implementierung

Die Umsetzung der in den Kapiteln 3 bis 5 eingeführten Grundlagen und Algorithmen erfolgte mit der Entwicklung eines *eCognition*-Plug-Ins (Abschnitt 6.3) zur Klassifizierung von Bildsegmenten auf Basis von Support Vector Machines. Als Ausgangsbasis der Implementierung wurde die Bibliothek LibSVM (Abschnitt 6.1), eine Toolsammlung und Bibliothek zur Klassifizierung mit SVMs und das *Definiens Developer SDK 1.5.x* für eCognition (Abschnitt 6.2) verwendet. Die Implementierung, das Debugging der Applikationen und die Evaluierung der Testergebnisse erfolgte mit Hilfe von MATLAB in der im Anhang A dokumentierten Entwicklungs- und Testumgebung.

Der Fokus der Arbeit liegt auf Support Vector Machines und deren Anwendung. Aus diesem Grund dienen die folgenden Abschnitte nur einer groben Orientierung bzgl. der Implementierungszusammenhänge und verwendeten Werkzeuge. Auf die Beschreibung von Implementierungsdetails wird bewusst verzichtet. Für weitere Informationen sei auf die jeweiligen Entwicklerdokumentationen und Bibliotheken verwiesen.

6.1 Bibliothek LibSVM

LibSVM[1] – in dieser Arbeit wurde die Version 3.1 von April 2011 verwendet – ist eine Bibliothek und Werkzeugsammlung der beiden Autoren Chang & Lin (2011). Die Bibliothek ist mitsamt der Quelltexte in den Programmiersprachen C und Java frei verfügbar und bietet eine Anbindung an eine Reihe weiterer Frameworks, Programmier- bzw. Skriptsprachen, wie beispielsweise Python, C#/.NET. und MATLAB (Abschnitt 6.4).

Die LibSVM Suite bietet gegenüber anderen frei verfügbaren Werkzeugen und Bibliotheken[2] eine Reihe von Vorteilen, welche in der Vorauswahl der Werkzeuge die Entscheidung für LibSVM begründen[3]. Neben den vergleichsweise kurzen Trainings- und Klassifikationszeiten auch bei großen Datenmengen spricht für LibSVM, dass die Bibliothek vollständig als Quelltext in der Programmiersprache C vorliegt. Eine Anbindung an *Definiens De-*

[1] LIBSVM – A Library for Support Vector Machines: http://www.csie.ntu.edu.tw/~cjlin/libsvm/ (Zugriff: 2011-07-07)

[2] Eine ausführliche Liste verfügbarer Support Vector Machine Software findet sich unter http://www.support-vector-machines.org/SVM_soft.html (Zugriff: 2011-10-02)

[3] Die Evaluierung verschiedener SVM-Bibliotheken ist ausdrücklich nicht Teil dieser Arbeit. Hierfür sei auf die Dokumentation der jeweiligen SVM-Bibliotheken verwiesen, in der die Frage „der besseren SVM Implementierung“ hinreichend diskutiert wird.

veloper XD erfolgt über C++ Wrapper (siehe Abschnitt 6.3). Sowohl die Datentypen als auch die Funktionsschnittstellen sind ohne weitere Vorkehrungen kompatibel und uneingeschränkt verwendbar. Zudem bestehen keinerlei zusätzliche Abhängigkeiten zu weiteren (beispielsweise numerischen Mathematik-)Bibliotheken.

Grundsätzlich könnte angenommen werden, dass die Klassifizierungsergebnisse bei allen Bibliotheken etwa gleich sind, weil sie allesamt SVMs wie in Kapitel 5 beschrieben implementieren. Dies ist aber nicht der Fall. So zeigen sich gerade bei Mehrklassenproblemen (siehe Kapitel 4.3) teils erhebliche Unterschiede zwischen den Klassifikationsergebnissen. Begründet werden kann dies mit den Unterschieden in der Umsetzung der Mehrklassen-Lösungsverfahren. Die in dieser Arbeit ausgewiesenen Ergebnisse beziehen sich - wenn nicht anders angegeben - immer auf LibSVM und auf die von der LibSVM favorisierte *One-Against-One* Methode (siehe Kapitel 4.3). Etwaige Laufzeitunterschiede bei binären Klassifikationen begründen sich durch die unterschiedliche Herangehensweise zur Lösung des quadratischen Optimierungsproblems (siehe Kapitel 5.4 bis 5.6).

LibSVM ist vergleichsweise einfach zu benutzen und bietet dennoch eine breite Palette verfügbarer Optionen, um die SVM an die jeweiligen Bedürfnisse des Benutzers bzw. an die jeweiligen Problemstellungen anzupassen. Neben den Bibliotheksfunktionen zur Einbindung von Support Vector Machines in eigene Applikationen werden auch Tools[4] mitgeliefert, welche Datendateien in einem LibSVM spezifischen Daten(text-)format lesen, verarbeiten und schreiben (Hsu et al., 2010; Chang & Lin, 2011).

Bei den Tools handelt es sich um Konsolenapplikationen, die in der Reihenfolge ihrer Anwendung für folgende Aufgaben verwendet werden:

- `svm-scale` - Lineare Skalierung der Merkmalswerte des Trainingsdatensatzes (Default auf den Bereich von -1 bis +1)
- `grid.py` - Python-Skript zum Tuning der Parameter C und γ unter Anwendung eines RBF-Kernels
- `svm-train` - Trainieren der SVM auf Basis vorhandener Trainingsdatensätze oder die Berechnung einer k-fachen Kreuzvalidierung (siehe Abschnitt 3.5.2) aus k Teilmengen der Trainingsdatensätze
- `svm-predict` - Klassifizierung von Datensätzen auf Basis der trainierten SVM

In dieser Arbeit wurden diese Werkzeuge größtenteils aus MATLAB heraus genutzt, um die Optimierung der SVM-Parametrisierung und Evaluierung der Klassifikationen voranzutreiben.

[4] Als *Tools* (dt. Werkzeuge) werden kleine Computerprogramme oder Dienstprogramme bezeichnet.

6.2 Definiens Developer XD für eCognition

Ein Mitgrund für die Verbreitung und Akzeptanz der Object Based Image Analysis (siehe Abschnitt 2.7) in der Fernerkundung und medizinischen Bildverarbeitung dürfte neben den in den einführenden Kapiteln 2.6, 2.7 und 3.3 genannten Vorteilen auch die parallel erfolgte Entwicklung der in der Fernerkundung und medizinischen Bildverarbeitung weit verbreiteten Software *eCognition* von Definiens/Trimble[5] sein (Blaschke & Lang, 2006; Navulur, 2006).

eCognition ist auf OBIA und die darauf aufbauenden Algorithmen spezialisiert und gibt es mittlerweile in mehreren branchenspezifischen Ausführungen. Mit allen *eCognition* Derivaten können Bilder segmentiert und die über mehrere Ebenen (engl. *layer*) verteilten hierarchisch organisierten Bildobjekte auf Basis der Objekteigenschaften analysiert und weiterverarbeitet werden (Definiens, 2011).

Definiens bindet einen Großteil der applikationsinternen Algorithmen als *Algorithm Plug-In* und Merkmalsdefinitionen als *Feature Plug-In* in die Hauptapplikation von eCognition ein. Damit ist die Applikation für bestimmte Märkte weitgehend individuell konfigurierbar, erweiterbar und wartbar. Bis zum Sommer 2011 hat *Definiens* bzw. *Trimble* aber noch keine *Support Vector Machine* Unterstützung für eCognition zur Verfügung gestellt[6], so wurde im Rahmen dieser Arbeit ein *eCognition SVM Algorithm Plug-In* entwickelt.

6.3 Entwicklung eines eCognition Algorithm Plug-In für SVMs

Definiens vertreibt ein Softwareentwicklungspaket (engl. *Software Development Kit* – SDK) mit der Bezeichnung *Definiens Developer XD*[7], um *eCognition*-spezifische Prozessabläufe und eigene Erweiterungsmodule in C++ zu implementieren. Das SDK besteht im Wesentlichen aus einem Prozesseditor, einigen Windows-Bibliotheken im Binärformat, einer Reihe von C++ include-Dateien zur Deklaration der Datentypen und der Funktionen der Programmierschnittstelle (engl. *Application Programming Interface* – API), einer nicht sonderlich umfangreichen Dokumentation und einigen wenigen Beispieldateien.

[5] eCognition Software: http://www.ecognition.com/ (Zugriff: 2011-06-01): ursprünglich von Definiens AG, http://www.definiens.com, seit 2010 Trimble Navigation Ltd. http://www.trimble.com

[6] Zeitgleich mit Abschluss dieser Arbeit veröffentlichte *Trimble* die Zwischenversion 8.7 von eCognition, die erstmals auch einen – in seiner Anwendung jedoch stark eingeschränkten – SVM-basierten Klassifikator zur Verfügung stellt. Trimbles Implementierung basiert auf der Open Source Bibliothek *OpenCV*, http://opencv.willowgarage.com/wiki/ (Zugriff: 2011-10-27). Aufgrund der starken Beschränkungen bzgl. der Parametrisierung der SVM kann diese erste Version von Trimble jedoch nur als prototypische Erweiterung eingestuft werden.

[7] Definiens Developer XD: http://developer.definiens.com/ (Zugriff: 2011-10-31)

Mit dem *Definiens Developer XD* SDK lassen sich Erweiterungsmodule, sogenannte *Plug-Ins*[8] – auch als *Components* bezeichnet – für eCognition entwickeln, die aus der Sicht des Benutzers völlig transparent in das Programmpaket integriert werden können. Dabei handelt es sich um eine dynamisch nachladbare Bibliothek (engl. *dynamic link library* – DLL), die eine definierte Schnittstelle zur Verfügung stellen muss.

Im Folgenden wird kurz die Einbindung des Plug-Ins in eCognition, die Implementierung des Prozessablaufs, sowie die SVM-Benutzerschnittstelle erläutert.

6.3.1 Einbindung des Plug-Ins in eCognition

Die Aufgabe eines eCognition-Entwicklers besteht nun darin, mit Hilfe der dokumentierten *Definiens Engine API* (Definiens, 2008) eine sogenannte *Algorithm Component* zu implementieren. Der Export bestimmter Funktionen aus dem Plug-In stellt sicher, dass das DLL während der Laufzeit dynamisch nachgeladen und während der Prozessierung eines Images mit bestimmten Funktionsaufrufen angesprochen werden kann. Das Plug-In-DLL liefert optional alle Informationen zur Abhandlung eines etwaigen Benutzerdialogs, interagiert während eines Prozessablaufs mit der Hauptapplikation und hat Zugriff auf wesentliche Teile der hierarchisch organisierten Datenstrukturen innerhalb von eCognition. Abbildung 6.1 zeigt schematisch die Anbindung des SVM-Plug-Ins an die Hauptapplikation über definierte Schnittstellen (blaue UML-Schnittstellensymbole) und die Datenflüsse (grüner Pfeil) zwischen der Applikation und dem Plug-In.

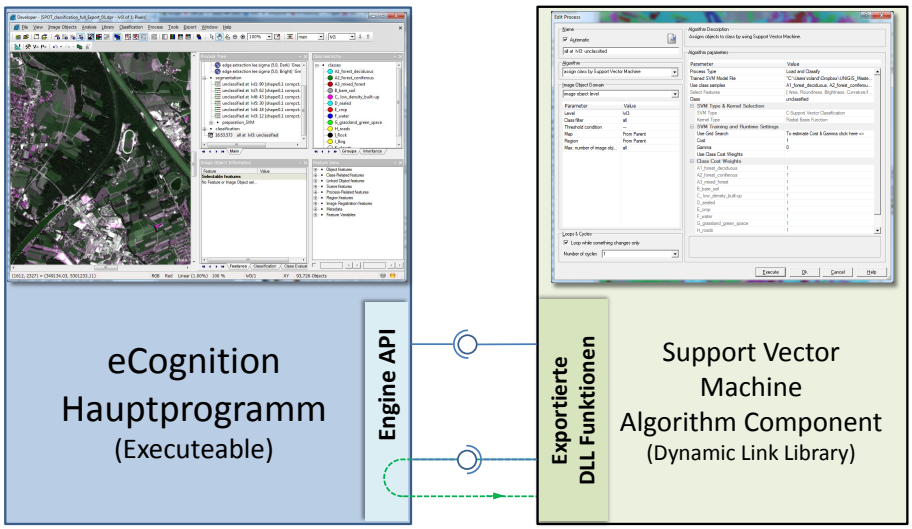


Abbildung 6.1: Dynamische Anbindung des SVM Algorithm Plug-In an eCognition

[8] Unter einem *Plug-In* (engl. *to plug in* - dt. *einstöpseln*, *anschließen*) versteht man ein Computerprogramm oder eine dynamisch nachladbare Bibliothek, die in ein anderes Softwareprodukt eingeklinkt werden, um dessen Funktionalität zu erweitern.

6.3.2 Implementierung des Prozessablaufs

Die Entwicklung eines eCognition Plug-Ins erfolgt in den Schritten (i) Registrierung des Plug-Ins zur Einbindung in die eCognition-Umgebung, (ii) Definition der Datentypen und Parameter zur Steuerung des Algorithm Plug-Ins, daraus wird auch von eCognition die Default-Benutzerschnittstelle abgeleitet, (iii) Implementierung der Plug-In spezifischen Benutzerdialoge (optional) und (iv) Implementierung der Funktionen zur Einbindung der Algorithmen in den eCognition-Prozessablauf.

Der Kernteil des Plug-Ins besteht aus den Einsprungfunktionen `AlgrInit`, wird am Beginn des Prozessablaufs aufgerufen, `algrExec`, wird während des Prozessablaufs für jedes Objekt aufgerufen und `algrDone`, wird am Ende des Prozessablaufs von eCognition aufgerufen. Innerhalb dieser drei Methoden werden in Abhängigkeit des vom SVM-Plug-In festgelegten Prozesstyps *Train and Save* oder *Load and Classify* unterschiedliche Methoden des Plug-Ins verwendet. Abbildung 6.2 zeigt den sequentiellen Ablauf der implementierten Methoden, deren Bezeichnung selbsterklärend sind.

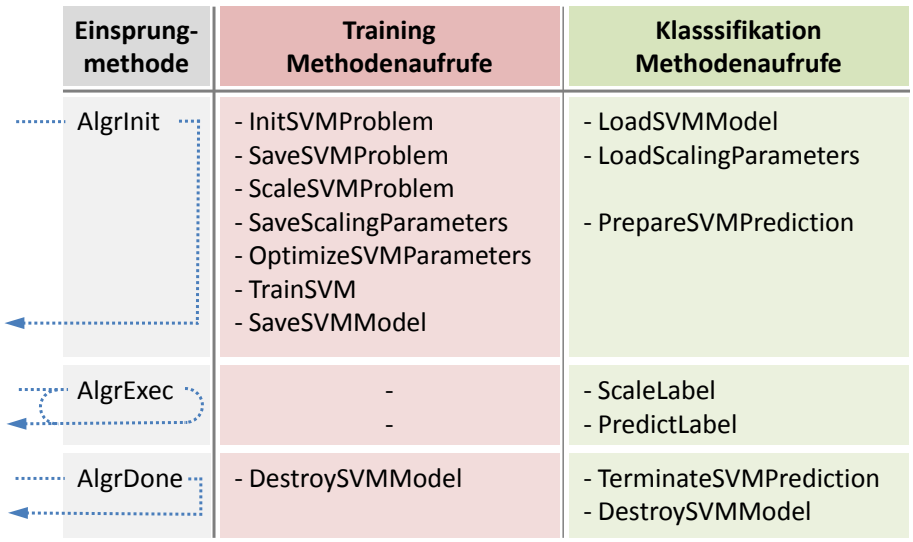


Abbildung 6.2: Ablauf eines Trainings und einer Klassifizierung innerhalb des SVM Algorithm Plug-In für eCognition

6.3.3 Benutzerschnittstelle

Die Default-Benutzerschnittstelle wird von eCogntion aus den zuvor definierten Datentypen und Parameter generiert. Abbildung 6.3 zeigt die Detaildarstellung der grafischen Benutzerschnittstelle der für diese Arbeit entwickelten SVM-Algorithm-Komponente. Der Benutzer wählt über (A) den SVM-Algorithmus aus und bestimmt mit den Dialogelementen (B) die in Abhängigkeit zur gewählten SVM verfügbaren Parameter.

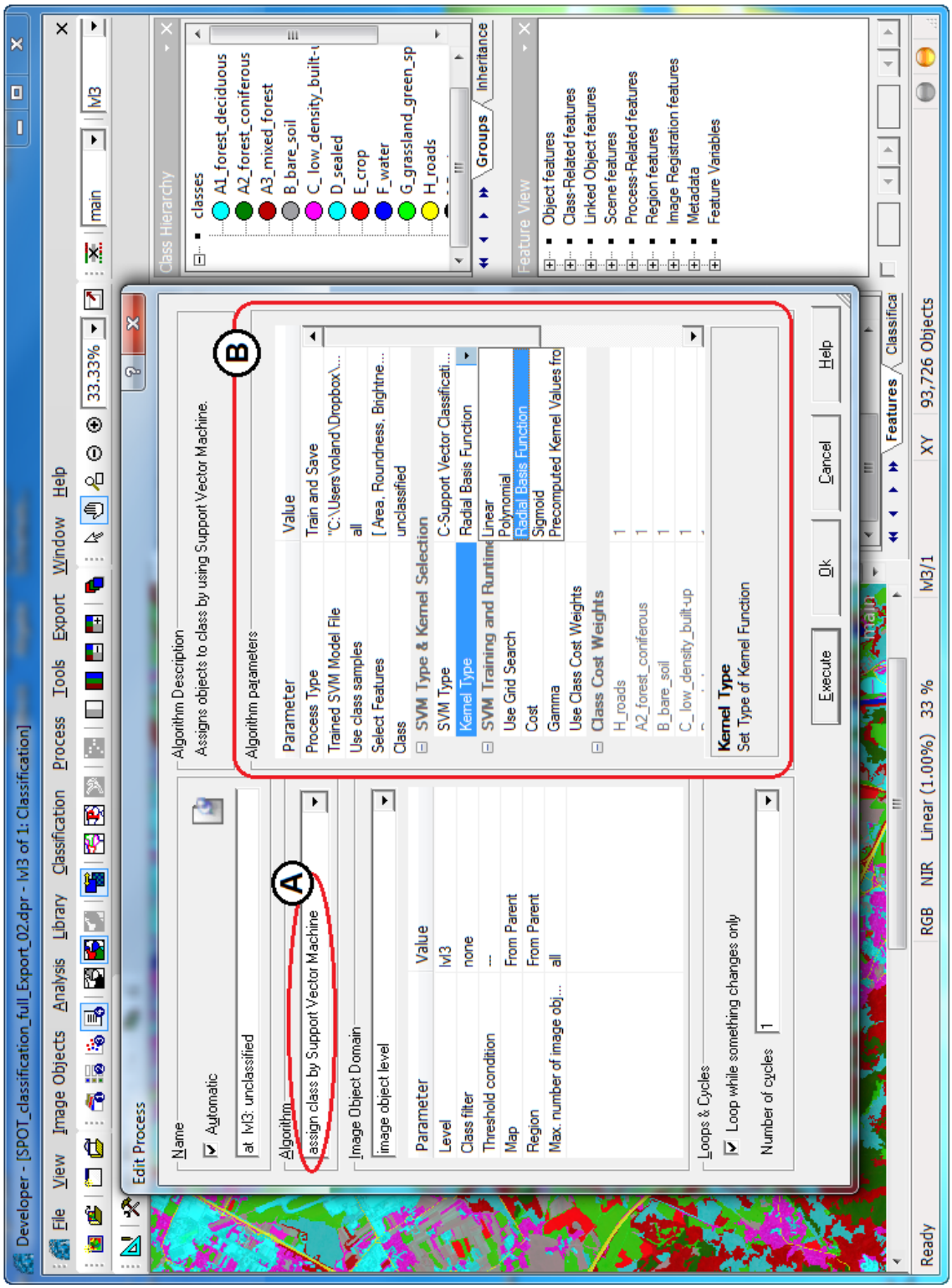


Abbildung 6.3: SVM Algorithm Plug-In Benutzerschnittstelle (Screenshot)

6.4 MATLAB, -Skriptdateien und -Executables

MATLAB[9] ist ein im technischen Umfeld weitverbreitetes kommerzielles Softwarepaket der Firma *The MathWorks Inc.*[10] zur numerischen Lösung mathematischer Probleme und zur grafischen Aufbereitung der Ergebnisse.

Eine Stärke von MATLAB liegt in der konsequenten Verarbeitung numerischer Aufgaben auf Basis von Vektoren und Matrizen, dessen Leistungsfähigkeit bei numerischen Simulationen und dem breiten Zusatzangebot spezieller Funktionsbibliotheken (genannt *Toolboxes*) für eine Reihe numerischer Problemstellungen. MATLAB bietet eine eigene proprietäre Skriptsprache und erlaubt darüber die Einbindung und den Aufruf eigener C-Funktionen.

Über spezielle Wrapper lassen sich auch die Funktionen der LibSVM in MATLAB einbinden, aus MATLAB-Skriptcode[11] ansprechen und deren Ergebnisdaten in MATLAB weiterverarbeiten. Zu diesem Zweck wurden sogenannte *Matlab Executables* (MEX-Dateien) (MathWorks, 2011) – spezielle DLLs – erstellt, mit denen die Funktionen der LibSVM über MATLAB kompatible Schnittstellen angesprochen werden können.

Im Rahmen dieser Arbeit wurden ebenso eine Reihe von MATLAB-Skriptdateien entwickelt. Deren Aufgaben sind im Wesentlichen

- die Auswahl und Aufbereitung der Eingangsdaten,
- die Optimierung und Parametrisierung der SVM während des Trainings,
- die Simulation von Klassifizierungsprozessen mit einer LibSVM-MEX-Anbindung
- die Auswertung und Visualisierung der Trainings- und Klassifizierungsergebnisse.

Den vollständigen Quelltext der MATLAB-Skriptdateien abzudrucken, würde den Umfang dieser Arbeit bei weitem sprengen. Die wesentlichsten Dateien und die im nachfolgenden Kapitel 7 zur Evaluierung verwendeten Masterdatensätze stehen dem interessierten Leser jedoch unter `http://www.users.fh-salzburg.ac.at/~rgraf/downloads/Downloads/Objektklassifikation_mit_SVM/` zur Verfügung. Voraussetzung zum Ablauf der Scriptdateien ist die Installation der LibSVM-Bibliotheken und die Einbindung in MATLAB entsprechend der LibSVM-Dokumentation und der LibSVM-README-Dateien[12].

[9] Der Name MATLAB ist abgeleitet von den beiden Begriffen *MATrix LABoratory*.
[10] MathWorks Inc: `http://www.mathworks.de/` (Zugriff: 2011-10-29)
[11] Manchmal auch als Scriptcode oder einfach als Script bezeichnet
[12] Die Ausführung der Scriptdateien ist mit den Originaldateien der LibSVM ohne weiteres möglich. Mitunter müssen die MEX-Dateien neu kompiliert werden, wofür entsprechende Entwicklungswerkzeuge (Visual Studio oder GCC) notwendig sind. Die Anleitung ist den LibSVM- und MATLAB-Dokumentationen zu entnehmen. An dieser Stelle sei vorweg festgehalten, dass in den LibSVM-Quelltexten für diese Arbeit einige Adaptierungen vorgenommen wurden, um die Prozessierungszeiten zu verkürzen und weitere Eingriffe in den Prozessierungsablauf vorzunehmen. Vgl. Kapitel 7.9.6.

7 Anwendung, Bewertung und Optimierung von SVMs

Die bisherigen Ausführungen haben die theoretischen Grundlagen beschrieben, die Anwendung von SVMs zur objektbasierten Klassifikation und eine objektive Prüfung von Klassifizierungsergebnissen sind bislang jedoch ausgeblieben. Dieses Kapitel zeigt die praktische Anwendung von SVMs und beschreibt sowohl ausgewählte Methoden zur Parametrisierung von SVMs als auch zur Evaluation der Ergebnisse.

7.1 Testgebiete und -datensätze

Die Analysen wurden mit Datensätzen auf Basis eines segmentierten und klassifizierten SPOT[1]-Satellitenbildes (siehe Abbildung 7.1) der *Stadt Salzburg und Umgebung* durchgeführt. Die Daten wurden in Form eines *eCognition*-Projekts vom *Zentrum für Geoinformatik*[2] (Z_GIS) der Universität Salzburg zur Verfügung gestellt.

Die Eckdaten der Testdatensätze lauten wie folgt:

- Das **Satellitenbild** weist eine **Auflösung von** 3492×4684 **Pixel** auf.
- Das Bild entspricht einer **Fläche von** $18,483 \times 24,792\ km = 458,26\ km^2$.
- Der Segmentierungsprozess hat aus den Bilddaten **93724 Objekte** erzeugt.
- Die Segmente wurden insgesamt **13 Klassen** zugeteilt.

Tabelle 7.1 auf Seite 77 listet die vorhandenen Klassen und die Anzahl der Segmente pro Klasse. Auf Pixelbasis stehen die in Tabelle 7.2 angeführten sechs Bildebenen (engl. *Image Layer*) zur Verfügung. Sie entsprechen den Bändern 1 bis 4 (Layer 1 bis 4) oder sind das Ergebnis eines im *eCognition* Segmentierungsprozess verwendeten Kantenfilters (Layer 5 und 6).

Bei den Evaluierungen wurden pro Segment bis zu maximal **46 Merkmale** verwendet. Eine Liste aller aus *eCognition* exportierten Merkmale zeigt Tabelle A.3 auf Seite 118.

Laut Z_GIS wurden die in der Tabelle 7.3 angeführten Merkmale (Features) bzw. davon abgeleitete Größen zur Klassifizierung herangezogen. Für jedes Segment wurden aus

[1] SPOT: Satellite Pour l'Observation de la Terre; http://www.spotimage.com (Zugriff: 2011-10-01)
[2] Centre for GeoInformatics (Z_GIS): http://www.zgis.at (Zugriff: 2011-10-01)

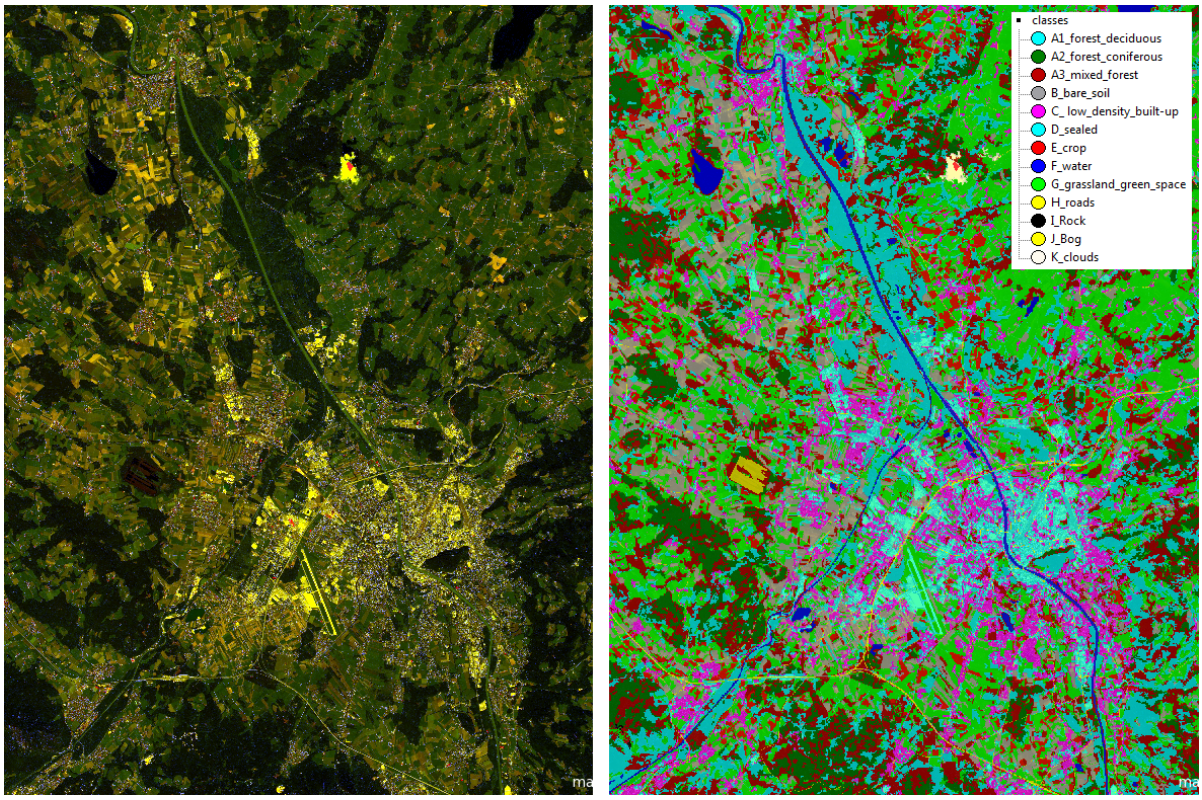


Abbildung 7.1: Testgebiet und Testdatensätze: Satellitenbild von Salzburg und Umgebung. Links: Falschfarbendarstellung des Testgebiets. Rechts: Segmentiertes und klassifiziertes Bild des Testgebiets mit 93724 Segmenten, zugeteilt in 13 Klassen.

dem grünen Band 3 die Standardabweichung (engl. *standard deviation* – `Stddev`) und das Maximum (`Max`) der Pixelwerte errechnet. Ebenso wurde aus dem nahen Infrarot (engl. *near-infrared* – `NIR` = Band 1) und dem sichtbaren Rot (engl. *red* = Band 2) der Vegetationsindex (engl. *normalized differenced vegetation index* – `NDVI`) als Unterscheidungsmerkmal herangezogen (Kröber M. et al., 2011):

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{7.1}$$

Weiters wurden sowohl der Mittelwert (engl. *mean*) aus den Pixelwerten des kurzwelligen Infrarots (engl. *short-wavelength infrared* – `SWIR`) und des nahen Infrarots als auch die Helligkeit (engl. *brightness*) über einen gewichteten Mittelwert aus den Kanälen 1 bis 4 herangezogen. Diese Merkmalswerte wurden bei der Segmentierung und anschließenden Klassifizierung allerdings in Abhängigkeit zur Klasse selektiv bzw. in unterschiedlichen Prozessierungsstufen und Kombinationen angewendet.

Die Objekte der Klasse `H_roads` (Straßen) sind lt. Z_GIS nicht aufgrund ihrer spektralen

Klassenname	ID: Bezeichner	Segmente
Straßen	12: H_roads	3234
Versiegelt	54: D_sealed	12014
Lockere Bebauung	53: C_low_density_built-up	15762
Feld	55: E_crop	1839
Abgeerntetes Feld	52: B_bare_soil	10763
Grün oder Wiese	57: G_grassland_green_space	21985
Nadelwald	51: A2_forest_coniferous	3792
Laubwald	59: A1_forest_deciduous	11674
Mischwald	60: A3_mixed_forest	11477
Moor	61: J_Bog	131
See oder Fluss	56: F_water	846
Fels	58: I_Rock	19
Wolken	62: K_clouds	188

Tabelle 7.1: Klassifikation der Testdaten

Bildebene	Quelle / Beschreibung
NIR	Band 1 / Near InfraRed: 780 nm bis 1400 nm
Red	Band 2 / Rot: 0.61 bis 0.68 μm
Green	Band 3 / Grün: 0.50 bis 0.59 μm
SWIR	Band 4 / Short-Wavelength InfraRed: 1.4 bis 3.0 μm
lee_sigma_dark	Ergebnis aus Kantenfilter 1
lee_sigma_bright	Ergebnis aus Kantenfilter 2

Tabelle 7.2: Image Layer der Testdaten

Berechnung	Quelle
Mean(NIR)	Mittelwert aus Band 1, Pixelwerte
Mean(SWIR)	Mittelwert aus Band 4, Pixelwerte
Max(Green)	Maximum aus Band 3, Pixelwerte
Brightness	Gewichteter Mittelwert aus Band 1 bis 4
Stddev(Green)	Standardabweichung aus Band 3 Pixelwerte
Stddev(lee_sigma_bright)	Standardabweichung des Kantenfilters 2
NDVI=((NIR-Red)/(NIR+Red)	NIR = Band 1, Red = Band 2
Ratio(VIS,SWIR)	Verhältnis von Band 2 zu Band 4

Tabelle 7.3: Benutzte Merkmale zur Klassifizierung der Testdaten

Eigenschaften klassifiziert worden, sondern wurden auf Basis externer Informationen festgelegt. Wie aus dem Bildmaterial hervorgeht, wurden dabei keine Nebenstraßen oder Wege berücksichtigt. So sind kleinere Straßen und Wege noch als Grünfläche oder Wald klassifiziert, selbst wenn sie auf dem Pixelbild deutlich sichtbar als Weg zu identifizieren gewesen wären. Dieser Umstand muss als *ground truth data* Mangel gewertet werden.

7.2 Phasen einer Klassifizierung mit SVMs

Der Ablauf einer SVM-basierten Klassifizierung verläuft in den Phasen *Datenaufbereitung*, *Parametrisierung* und *Klassifizierung*, die in mehreren Schritten sequentiell und/oder iterativ durchlaufen werden. Abbildung 7.2 stellt den Ablauf dieser Phasen und ihre einzelnen Schritte symbolisch dar.

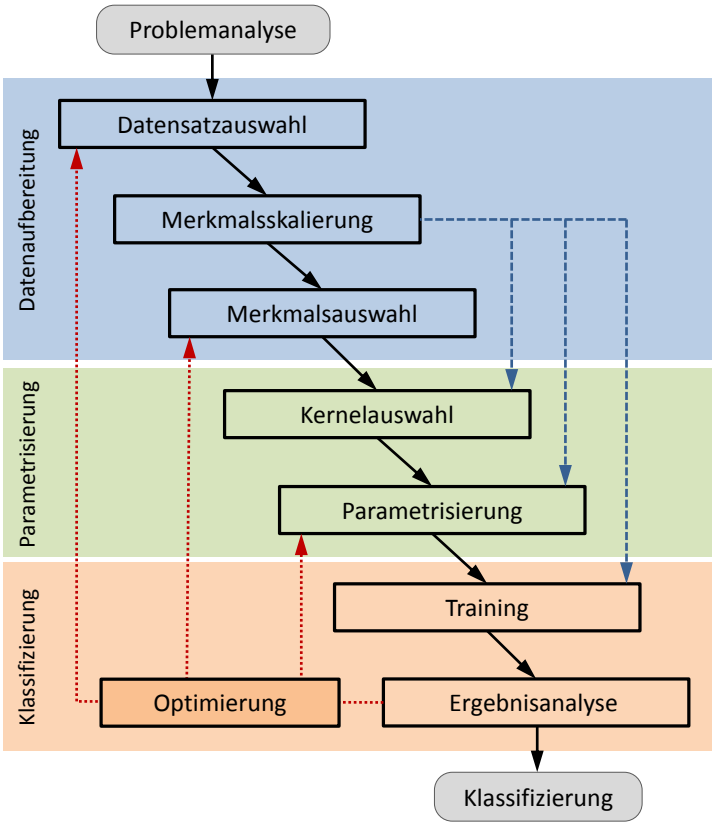


Abbildung 7.2: Phasen eines SVM-basierten Mustererkennungssystems

Die Phasen eines SVM-basierten Mustererkennungssystems werden in acht Teilaufgaben zerlegt, wobei einige davon je nach Anforderung optional sind.

1. **Auswahl der Datensätze**: legt jene Vektoren fest, die zum Trainieren und Testen der Support Vector Machine verwendet werden (Kapitel 7.4).

2. **Skalierung (Standardisierung) der Datensätze**: transformiert alle Merkmale in einen einheitlichen und geeigneten Wertebereich (Kapitel 7.5).
3. **Auswahl der Merkmale**: legt die Dimension des Merkmalsraums fest und bestimmt jene Merkmale, die zur eindeutigen Unterscheidung der jeweiligen Klassen am geeignetsten erscheinen (Kapitel 7.6).
4. **Auswahl des Kernels**: bestimmt die Funktion zur Transformation der Merkmale in einen höherdimensionalen Merkmalsraum, um die Vektoren bestmöglich mit einer Hyperebene zu separieren (Kapitel 7.7).
5. **Parametrisierung der SVM**: legt in erster Linie die Funktionsparameter der Kernel und den Einfluss der Schlupfvariablen C der SVM fest (Kapitel 7.8).
6. **Training**: berechnet die Stützvektoren und SVM-Parameter zur Festlegung der Hyperebene (Kapitel 4 und 5) und liefert als Ergebnis das mathematische Modell[3].
7. **Qualitätsanalyse**: überprüft die Generalisierungsleistung (Kapitel 3.5) der SVM und nimmt eine Bewertung der Klassifikation (Kapitel 3.6) mittels Testdaten vor[4].
8. **Optimierung**: optimiert die Daten- und/oder Parametrisierung auf Basis der Qualitätsanalyse und spezieller Problemanforderungen (Kapitel 7.9).

Diese Teilaufgaben werden im Normalfall sequentiell durchlaufen. Stehen nur wenige Merkmale zur Verfügung oder sind einige Kriterien schon vorweg bekannt, so werden einzelne Aufgaben - wie beispielsweise die Selektion der Merkmale - übersprungen. Symbolisiert wird dies in Abb. 7.2 mit den blauen Linien. Die rot punktieren Linien weisen auf den iterativen Charakter des SVM-basierten Mustererkennungssystems hin. Wird aus den Ergebnissen der Qualitätsanalyse eine Anpassung der Daten oder - wahrscheinlicher noch - eine Anpassung der Parametrisierung des Klassifikators notwendig, so werden einzelne Teilaufgaben solange wiederholt, bis ein zufriedenstellendes Ergebnis erreicht wird oder keine Verbesserung der Ergebnisse mehr erreicht werden kann.

7.3 Methodische Vorgangsweise

Wie im vorangegangenen Kapitel beschrieben, wurde ein *eCognition Algorithm Plug-In* (Kapitel 6.3) entwickelt, mit dem innerhalb der *eCognition Software Suite* mit Hilfe von Support Vector Machines Bildsegmente klassifiziert werden können. In den nachfolgenden Abschnitten wird gezeigt, inwieweit SVMs den Ansprüchen gerecht werden können,

[3] Dem Training wird in diesem Kapitel kein eigener Abschnitt gewidmet, da das eigentliche Training ohne manuellen Eingriff erfolgt.

[4] Der Ergebnisanalyse wird in diesem Kapitel kein eigener Abschnitt gewidmet. Die Bewertung und Interpretation der Ergebnisse (Kapitel 3.6) findet in den jeweiligen Teilaufgaben statt.

welchen Einfluss die einzelne Stellgrößen auf das Ergebnis haben und wie die Qualität einer Klassifikation verbessert werden könnte.

Das SVM Plug-In stellt Datenschnittstellen zur Verfügung, die eine Weiterverarbeitung der Daten mit LibSVM Tools (Kapitel 6.1) und/oder eine Qualitätsanalyse mit MATLAB (Kapitel 6.4) ermöglichen.

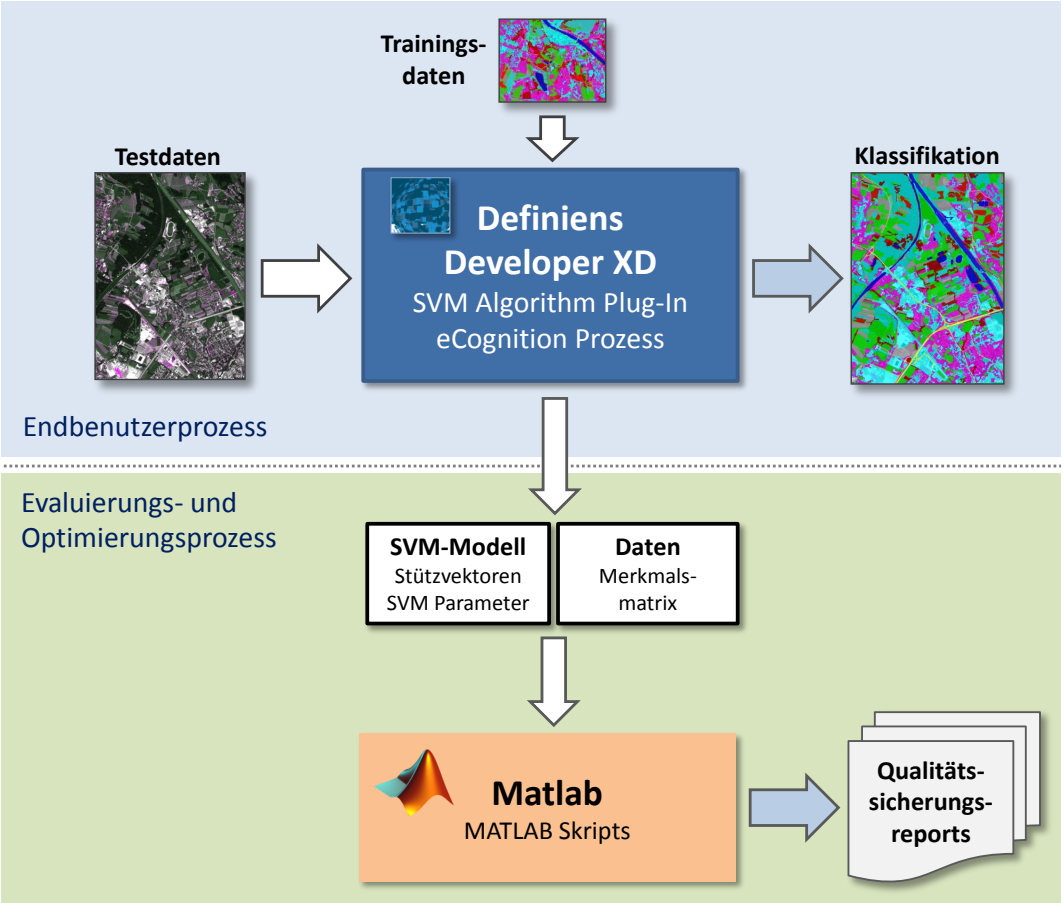


Abbildung 7.3: Prozessablauf einer Klassifizierung und Evaluierung mittels eCognition und MATLAB

Abbildung 7.3 zeigt einen typischen Prozessablauf einer Evaluierung mittels eCognition und MATLAB. Die zur Verfügung stehenden Trainings- bzw. Testdaten werden in eCognition importiert und prozessiert. Optional exportiert das SVM Plug-In modellrelevante Einstellungen und die verarbeiteten Testdaten (Labels, Vektormatrix, Metadaten) in einem LibSVM kompatiblen Datenformat oder in einfache CSV[5]-Dateien. Diese Daten werden zur Evaluierung und Optimierung in MATLAB geladen und in MATLAB-Skriptdateien weiterverarbeitet.

[5]CSV steht für *Comma Separated Values*

7.4 Auswahl der Trainings- und Testdatensätze

Eine sorgfältige Auswahl vorklassifizierter Datensätze spielt gerade bei lernenden Algorithmen eine wesentliche Rolle. Die Datensatzauswahl muss in Abhängigkeit der Daten und der Dimension des Merkmalsraums einen entsprechenden Umfang besitzen, um die Entscheidungsgrenzen vollständig beschreiben zu können. Zudem sollten die Beispieldaten (engl. *sample objects*) sowohl repräsentativ für die später zu klassifizierenden Datensätze als auch ausgewogen in deren Klassenzusammensetzung sein.

Ein Selektionsverfahren legt jene Vektoren fest, die zum Trainieren und Testen der Support Vector Machine verwendet werden. Als Ausgangspunkt der Überlegungen gelten die Annahmen: (i) je mehr Trainingsdaten, desto näher wird die gelernte Trennfunktion an der „wahren“ Entscheidungsgrenze liegen, d.h. der Klassifikator macht weniger Fehler; (ii) je mehr Testdaten, desto genauer wird die Fehlerabschätzung bei den Tests ausfallen können oder - anders ausgedrückt - desto vertrauenswürdiger sind die Testergebnisse. Möglich ist beispielsweise eine zufällige Teilung der zur Verfügung stehenden Datensätze im Verhältnis $^2/_3$ zum Trainieren und $^1/_3$ zur Überprüfung der Generalisierungsfähigkeit des Klassifikators[6]. Gelten muss in jedem Fall, dass die Merkmalsverteilung der Trainingsdaten gleich jener der Testdaten und der späteren zur klassifizierenden Daten sein muss, sodass repräsentativ *gesampelt* wird.

Es gibt keine allgemeingültigen Regeln, wie viele Datensätze pro Klasse notwendig sind, um zwei oder mehrere Klassen zu separieren, da dies von der Komplexität der Daten, deren Merkmalen und des eingesetzten Klassifikators abhängt. Es gibt aber einige in der Fachliteratur dokumentierte Zusammenhänge (Camp-Valls & Bruzzone, 2009):

- Je mehr Klassen, desto mehr Trainingsdatensätze.
- Je komplexer die Merkmalsverteilungen, desto mehr Trainingsdatensätze.
- Je mehr Trainingsdatensätze, desto länger die Prozessierungsdauer.

Als konventionelle Hilfestellung mag - ohne auf Abschnitt 7.9.3 vorzugreifen - vorab die Empfehlung gelten, so viele Trainings- und Testdatensätze wie möglich zu sammeln (Camp-Valls & Bruzzone, 2009). Die bei den meisten Klassifikatoren bestehende Gefahr, dass zu viele Trainingsdatensätze zu einem Overfitting führen, ist bei SVMs in diesem Ausmaß nicht zu befürchten, da hierbei nur Randpunkte der Klassen die Stützvektoren bilden, und die restlichen Vektoren ohnehin unberücksichtigt bleiben.

Wie sich in den folgenden Abschnitten zeigt, erkennt man erst bei der Qualitätsanalyse, ob die Auswahl und der Umfang der Trainings- und Testdaten den Ansprüchen genügen.

[6] Bei einer n-fach Kreuzvalidierung (engl. *n-fold cross validation*) mit beispielsweise $n = 5$ wird bei einem Validierungsdurchgang das Verhältnis $^4/_5$ Trainings- zu $^1/_5$ Testdatenanteil verwendet (Kapitel 3.5.2).

7.5 Standardisierung der Datensätze

Support Vector Machines bestimmen die Lage der Trennebene während des Trainings über die Maximierung des Abstands der Ebene zu den jeweiligen Stützpunkten (siehe Kapitel 5.4). Um die Dominanz einzelner Merkmale zu verhindern und alle Merkmale mit gleicher Gewichtung in die Abstandsberechnungen einfließen zu lassen, sollten die Trainingsdaten derart skaliert werden, dass alle Merkmalswerte in einem ähnlichen Wertebereich abgebildet werden (Hsu et al., 2010). Hierfür stehen eine Reihe von Standardisierungs- bzw. Normierungsmöglichkeiten zur Verfügung, die Merkmale auf gleiche Skalenniveaus und Größenordnungen bringen, und damit Merkmale auch klassenübergreifend vergleichbar machen.

- **Z-Transformation**: Eine einfache Möglichkeit besteht darin, von allen Merkmalswerten den Mittelwert zu subtrahieren und durch die Standardabweichung zu dividieren:

$$\tilde{x} = \frac{x - \mu}{\sigma} \tag{7.2}$$

  Dadurch erhalten wir für alle Merkmale eine *standardisierte Verteilung* der Merkmalswerte mit dem Mittelwert $\mu = 0$ und der Standardabweichung $\sigma = 1$.

  Wird anstatt des Mittelwerts der Median herangezogen, so haben Ausreißer in den Trainingsdaten einen geringeren Einfluss auf deren Skalierung. Abbildung A.2 im Anhang A auf Seite 120 zeigt die über alle Klassen berechneten standardisierten Verteilungen mehrerer Merkmale der 93724 verfügbaren Testdatensätze.

- **Lineare Skalierung**: Einfacher und weniger aufwendig ist eine lineare Skalierung der Merkmalswerte mithilfe der Minima `min` und Maxima `max` aller Merkmalswerte, so dass diese immer in einem konstanten Intervall - oft wird das Intervall $[0, 1]$ oder $[-1, +1]$ verwendet - abgebildet werden.

$$\tilde{x} = \frac{x - \min}{\max - \min} \tag{7.3}$$

- **Atan-Skalierung**: Die Skalierung über einen Arcustangens zwingt die Merkmalswerte in einen Intervall von $[-\frac{\pi}{2}, +\frac{\pi}{2}]$, skaliert die ursprünglichen Werte nun aber in Abhängigkeit des Abstands des Merkmalswerts zum Nullpunkt. In der Regel geht eine Z-Transformation der Daten voraus, um die Merkmalswerte um den Nullpunkt anzuordnen. Entsprechend dem Verlauf der Arcustangens-Funktion werden hernach zentraler gelegene Werte nahezu linear skaliert, Ausreißer jedoch mit zunehmendem Abstand zum Nullpunkt stark abgeschwächt.

- **Normaleneinheitsvektor**: Nicht unerwähnt soll eine weitere - in dieser Arbeit jedoch nicht evaluierte - Möglichkeit bleiben, welche die Autoren Ben-Hur et al.

(2008) beschreiben. Dabei werden die Merkmale aller Eingangsvektoren nicht separat skaliert, sondern jeder Eingangsvektor wird entsprechend der Formel

$$\tilde{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|} \tag{7.4}$$

mit der euklidischen Norm (Länge, Betrag) des Vektors dividiert, um einen *Normaleneinheitsvektor* zu erhalten, dessen Vektorlänge auf 1 normiert ist.

Welche Methode sich letztlich als die geeignetste erweist, hängt stark von der Verteilung und Qualität der Merkmalswerte ab. Als weitgehend unkritisch ist sicherlich die lineare Skalierung anzusehen, wenngleich sie gerade bei Ausreißern keine wirkungsvollen Gegenmaßnahmen ergreift. In diesen Fällen ist eine Atan-Skalierung vorzuziehen. LibSVM stellt eine Bibliotheksfunktion und das Kommandozeilenwerkzeug `svm-scale` zur Verfügung, um Merkmalswerte ausschließlich linear zu skalieren (Hsu et al., 2010).

Ob die Merkmalswerte im Eingangsraum oder im Merkmalsraum skaliert werden ist unwesentlich, solange für alle Merkmale die gleiche Vorgangsweise gilt. Für alle Methoden gilt, dass die beim Training ermittelten Skalierungsparameter in jedem Fall gesichert werden müssen, da die zu einem späteren Zeitpunkt zu klassifizierenden Daten unbedingt mit der gleichen Methode und Parametrisierung skaliert werden müssen.

7.6 Auswahl der Merkmale

Klassifikatoren separieren die Objekte auf Basis ihrer Merkmale. Weist ein Objekt beispielsweise bestimmte Merkmale vermehrt oder andere Merkmale nur schwach auf, so kann das Objekt durch die Kombination der Merkmalsinformationen im günstigen Fall von anderen Objekten unterschieden und ebenso einer bestimmten Klasse zugeordnet werden. Sinnvoll hierfür ist die Auswahl jener Merkmale oder Merkmalskombinationen, die vorweg schon eine Unterscheidung zulassen.

Werden die Trainingsdaten nach Klassen zusammenfasst und daraus für jedes Merkmal die Wahrscheinlichkeitsdichtefunktionen (engl. *propability density function*) ermittelt, so erhält man die *wahrscheinliche Verteilung der Merkmalswerte* über den Wertebereich der jeweiligen Merkmale über alle Klassen. Diese Darstellungsform lässt sich auch nutzen, um die Trennfähigkeit der jeweiligen Merkmale sichtbar zu machen.

Abbildung 7.4 zeigt die Dichtefunktionen der beiden Merkmale `MaxNIR` (links) und `Length/Width` (rechts) des Testgebiets. Die Skalierung erfolgt auf Basis des Medians und der Standardabweichung.

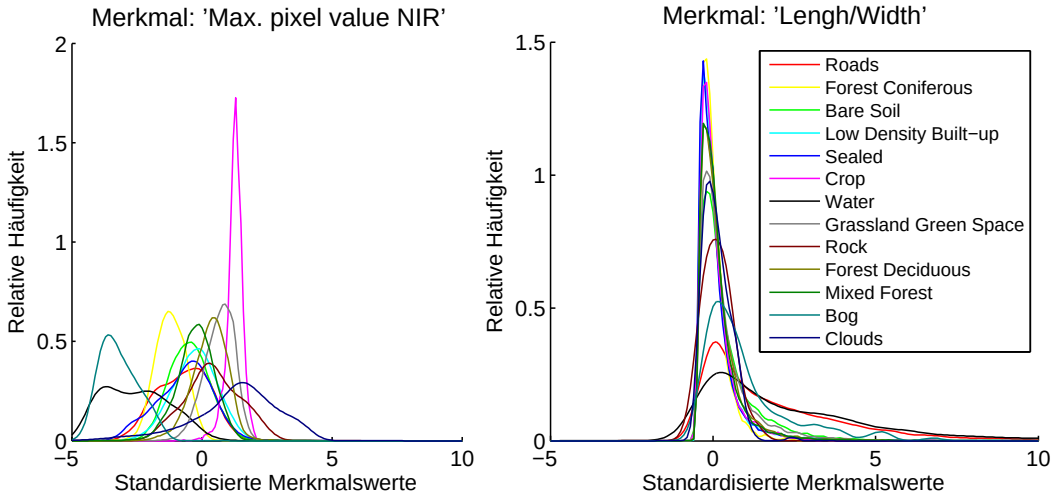


Abbildung 7.4: Dichtefunktionen der Merkmale MaxNIR und Length/Width.

Wie sollten die Dichtefunktionen nun optimalerweise aussehen? Hierbei entscheidet weniger die Form einer einzelnen Funktion als vielmehr die Lage der Dichtefunktionen zueinander. Gewünscht ist eine Verteilung der Merkmale, so dass aufgrund des Merkmalswerts schon eine mit hoher Wahrscheinlichkeit korrekte Klassifizierung erfolgen kann. Die Merkmalsbereiche, im Diagramm als die Breite der Dichtefunktionen sichtbar, sollten sich demnach möglichst wenig überlappen.

Beispiel: Wie aus den Abbildungen 7.4 hervorgeht, eignet sich das von NIR abgeleitete Merkmal deutlich besser zur Unterscheidung der einzelnen Klassen als das Längen-Breiten-Verhältnis der einzelnen Bildsegmente. Die Merkmalsdichtefunktionen der linken Abbildung lassen eine Separierung nach Klassen, wenn schon nicht für alle Klassen, so zumindest im eingeschränkten Umfang zu. Die Trennfähigkeit des rechten Merkmalswerts ist deutlich geringer. Die Merkmalsdichtefunktionen häufen sich alle um den Achsenursprung und lassen aufgrund der Überlappungen für sich alleine betrachtet kaum eine Unterscheidung zu.

Abbildung A.3 im Anhang A auf Seite 121 zeigt die standardisierten Verteilungen ausgewählter Merkmalswerte für alle Klassen. Die Diagramme bestätigen auch, dass die Verteilungen einiger Merkmale kaum eine Unterscheidung, andere eine vergleichsweise gute Unterscheidung der Klassen zulassen.

7.7 Auswahl des Kernels

Betrachtet man die Punktwolke eines mehrdimensionalen Klassifizierungsproblems in einem 2- oder 3-dimensionalen Merkmalsraum (siehe auch Abbildung 3.1), so ist leicht

einzusehen, dass nicht jede Funktion im gleichen Maße zur Beschreibung einer Trennebene geeignet ist. Handelt es sich um n-dimensionale Merkmalsräume ($n \gg 2$), so wird die Vorstellung der Hyperebene schwieriger und deren Visualisierung gar unmöglich. Durch die Einführung von Kernel in SVMs steigt die Komplexität zudem, indem die Eingangsdaten X implizit mit einer nicht-linearen Funktion $\phi(X)$ in einen höherdimensionalen Merkmalsraum $\mathcal{F}$ transformiert werden, um darin eine lineare Trennung der Klassen vorzunehmen. Wie wird die Kernel-Funktion nun ausgewählt?

Obwohl die Literatur (Bishop, 2006; Cristianini & Taylor, 2000; Shawe-Taylor & Cristianini, 2004) bestimmte Kernel und deren Kombinationen diskutiert, zeigt sich schnell, dass in der Praxis oft nur einige wenige Kernel zur Anwendung kommen (siehe Kapitel 5.7.1) und aufgrund der mathematischen Komplexität heuristische Methoden die Kernelauswahl dominieren (Chapelle & Vapnik, 1999). Die Autoren Hsu et al. (2010) von LibSVM empfehlen die Verwendung des RBF-Kernels, wenn nicht explizite Gründe dagegensprechen.

Beispiel: Aus den Testdaten wurden für die Klassen `[12 53 54]` mit `[1:42]` Merkmalen[7] mit insgesamt `3×3234=9702` Vektoren eine Cross Validation Accuracy (CVA) mit unterschiedlicher Parametrisierung für C und γ für verschiedene Kernel gerechnet.

Ergebnisse: Tabelle 7.4 zeigt die Zusammenfassung der Ergebnisse, wobei für jeden Kernel jene Parametrisierung mit dem besten Ergebnis gelistet ist. Wie aus dem Ergebnis hervorgeht, liefert der RBF-Kernel zwar mit 96, 91% die beste Cross Validation Accuracy, die anderen drei Kernel liefern aber durchaus vergleichbar gute Ergebnisse. Die Anzahl der Stützvektoren, ebenfalls ein Qualitätskriterium wie sich im nächsten Abschnitt noch zeigen wird, ist oft bei jenen Kernel- und Parameterkombinationen am geringsten, die auch die höchste CVA ausweisen. Auffallend ist, dass der Linear-Kernel zwar eine CVA gleichauf mit den anderen bietet, die Prozessierungsdauer aber die der anderen Kernel deutlich übersteigt.

Ergebnis	RBF	Polynom	Sigmoid	Linear
Cross Validation Accuracy (in %)	96,91	96,77	94,93	95,31
mit Stützvektoren (max. 9702)	902	909	1445	1196
nach Prozessierungsdauer (in Sek.)	1748	4263	1100	6480

Tabelle 7.4: Cross Validation Accuracy bei Anwendung unterschiedlicher Kernel

Beispiel: Aus den zuvor schon verwendeten Klassen `[12 53 54]` mit `[1:42]` Merkmalen wurden je Klasse exakt `3000` Vektoren zufällig ausgewählt. Mit den zuvor berechneten

[7] In den folgenden Beispielberechnungen wird die Auswahl der Trainings- und Testdaten bzw. Merkmale in der MATLAB-Notation für Vektoren beschrieben. Vektoren werden in eckigen Klammern definiert. Die Kurzschreibweise [1:42] beschreibt einen 42-dim. Vektor der Form [1 2 3 ... 42].

optimalen Parametern C und γ wurden die selben Kernel trainiert und dann die insgesamt `31010` verbleibenden Vektoren der drei Klassen klassifiziert.

Ergebnisse: Tabelle 7.5 zeigt, dass der RBF-Kernel mit $81,26\%$ die beste Overall Accuracy ausweist. Die Anzahl der Stützvektoren ist bei allen Kernel etwa $^1/_3$ der Trainingsvektoren, wohingegen die Trainingszeit beim Polynomial-Kernel ungleich länger ist als bei allen anderen Kernel. Der Algorithmus konvergiert bei diesem Kernel nur sehr langsam.

Ergebnis	RBF	Polynom	Sigmoid	Linear
Overall Accuracy (in %)	81,26	70,65	78,73	79,60
Stützvektoren (max. 9000)	3139	2745	3624	3358
Trainingsdauer (in Sek.)	17,4	12717,8	3,4	754,2
Klassifizierungsdauer (in Sek.)	2,8	2,0	3,5	2,3

Tabelle 7.5: Overall Accuracy bei Anwendung verschiedener Kernel

Dieses Ergebnis und weitere Versuche mit weniger Merkmalen lassen die Vermutung zu, dass neben den Daten auch die Anzahl der Dimensionen (=Merkmale) einen wesentlichen Einfluss auf die Wahl des Kernels haben. Je höher die Dimension des Merkmalsraums, desto eher liefert auch ein linearer Kernel befriedigende Ergebnisse. Die Vermutung, dass ein linearer Kernel wegen seiner Einfachheit eine schnellere Prozessierung zulässt, kann nicht bestätigt werden. Die schnellere Ausführung einer linearen Kernel-Methode wird - wie im vorliegenden Beispiel - durch das schleppende Konvergenzverhalten des Trainingsalgorithmus bei Linear-Kernel mitunter zunichte gemacht.

Weitere Versuche mit anderen Daten haben ebenso gezeigt, dass sowohl der Polynom- als auch der Sigmoid-Kernel mitunter zu keinem Ergebnis in akzeptabler Ausführungszeit kommen. Bei Tests wurden Berechnungen teilweise nach mehr als 24 Stunden erfolglos abgebrochen. Ein Blick in den Quelltext der SVM bestätigt, dass auch hierbei der Algorithmus zu langsam konvergiert. Eine Manipulation der Abbruchbedingungen brachte keine allgemein gültigen Verbesserungen[8]. Eine Regel, wann eine Berechnung bzw. Parametrisierung Erfolg hat und wann nicht, konnte für diese beiden Kernel nicht zielsicher bestimmt werden.

Aus obigen Erkenntnissen abgeleitet kann der RBF-Kernel als vergleichsweise unkomplizierter Allround-Kernel gelten, der zudem oft die besten Ergebnisse bei guten oder mittleren Prozessierungszeiten liefert.

[8] Andere Implementierungen, wie beispielsweise die MATLAB-interne SVM, sehen dafür die Festlegung einer maximalen Anzahl an Iterationen als zusätzliche Abbruchbedingung vor, um dieses Problem zu entschärfen oder zumindest einen Abbruch zu erzwingen.

7.8 Parametrisierung der SVM

Wie aus dem Kapitel 5 hervorgeht, besitzt eine Support Vector Machine eine Reihe von Parametern, die das Ergebnis einer Klassifizierung in Abhängigkeit der zu lernenden oder klassifizierenden Datensätze mehr oder weniger stark beeinflussen. Wie groß der Einfluss dieser Parameter ist und wie eine *günstige* Parametrisierung erzielt werden kann, sollen folgende Beispiel zeigen.

7.8.1 Einfluss der Parametrisierung am Beispiel eines RBF-Kernels

Am deutlichsten sichtbar wird der Einfluss der Parameter bei der Verwendung von Kernel (Kapitel 5.7.1), über deren Gleichungen funktionsabhängige Parameter direkt in das Ergebnis der Kernel-Funktion und die Klassifikation einwirken. Ein häufig genutzter Kernel ist der sogenannte *eRBF-* oder *Gauss-Kernel* der Form $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{z}\right\|^2\right)$. Dabei bestimmen der Parameter γ des Gauss-Kernels (5.41) und das Fehlergewicht C als Stellgröße bei Soft-Margin SVMs (5.23) im Wesentlichen die Anpassungsfähigkeit der SVM an die Problemstellung. Abbildung 7.5 visualisiert diesen Zusammenhang.

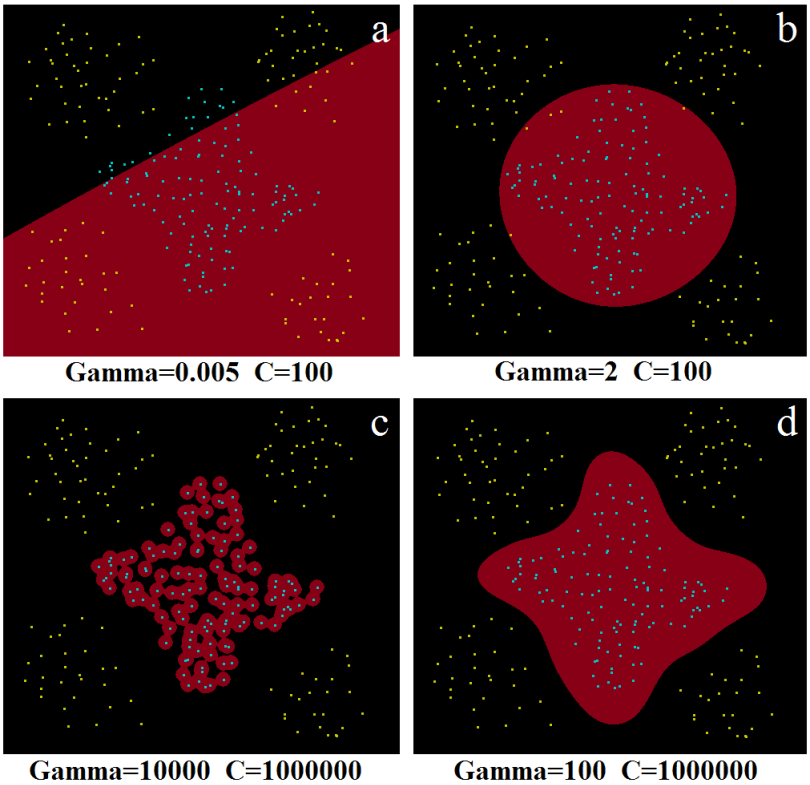


Abbildung 7.5: Einfluss der Parameter C und γ auf eine binäre Klassifikation. Bild (a): γ zu klein, C zu klein; Bild (b): γ passend, C zu klein; Bild (c): γ zu groß, C passend; Bild (d): γ passend, C passend

Ist γ im Beispiel der Abbildung 7.5 zu klein gewählt, so sind die Gauss-Glocken derart in die Breite gezogen, dass sie sich überlappen und die Konturen verschwinden. In Kombination mit einer zu großen Toleranz beim Auftreten von Fehlklassifikationen durch ein zu niedrig gewähltes C wird die Trennebene nahezu linear (Abbildung 7.5.a). Ist γ zu groß gewählt (Abbildung 7.5.c), so legen sich die Gauss-Glocken zu eng an die Punkte. Es kommt zwar aufgrund eines korrekt eingestellten Parameters C zu keinen Fehlklassifikationen, jedoch zu einem Übertraining der SVM, weil die Trainingsdaten zu exakt abgebildet werden. Ist die Einstellung der Parameter C und γ korrekt, so generalisiert die SVM gut und trennt die beiden Klassen korrekt (Abbildung 7.5.d). Dieses Beispiel belegt, dass eine korrekte Wahl der Parameter eine wesentliche Voraussetzung für eine gute Generalisierung einer SVM bildet.

7.8.2 Wahl der Parameter C und γ am Beispiel eines RBF-Kernels

Wie können die Kernel-Parameter und das Fehlergewicht C nun aber berechnet werden? Leider kann weder aus den Daten, noch durch die geschickte Wahl eines bestimmten Kerneltyps die optimale Parametrisierung vorhergesagt werden, denn „...the parameter C has no intuitive meaning.“ (Shawe-Taylor & Cristianini, 2004, S. 225). Stattdessen empfehlen Autoren wie Chang & Lin (2011) eine numerische Parametersuche, bei der aus den Trainingsdaten mit unterschiedlichen Parameterkombinationen jeweils eine n-fache Kreuzvalidierung (Kapitel 3.5.2) durchgeführt wird, um aus allen Ergebnissen eines gedachten *Cross Validation Accuracy* Netzes das CVA-Maximum zu ermitteln.

Beispiel: In einer Beispielrechnung wurde die Cross Validation Accuracy mit unterschiedlicher Parametrisierung für 54492 Vektoren aus den 8 Klassen `[12 51 52 54 55 56 57 58]` und `[1:42]` Merkmalen ermittelt. Berechnet wurde eine 5-fach Cross Validation Acccuracy jeweils für das Fehlergewicht C für den Bereich von 2^{-12} bis 2^{12} und den RBF-Kernelparameter γ für den Bereich 2^{-6} bis 2^{12}.

Ergebnisse: Wie im linken Contour-Plot des CVA-Grids der Abbildung 7.6 sichtbar wird, ist mit der gewählten Datenauswahl die Parametrisierung der SVM wesentlich für den Klassifizierungserfolg verantwortlich. Je nach Wahl der Parameter C und γ wird beim Trainingsvorgang eine CVA im Bereich von $\leq 45\%$ bis $\geq 85\%$ ausgewiesen. Wählt man bei den verwendeten Testdaten C im Bereich von 2^{-3} bis 2^{12} und γ im Bereich von 2^{-12} bis 2^{-3}, so sind die besten Ergebnisse bei einer Klassifizierung zu erwarten. Die CVA-Maximum wurde bei $C = 2^6$ und $\gamma = 2^{-3}$ erreicht (siehe roter Markierungspunkt).

Abbildung 7.6 rechts stellt die Anzahl der Support-Vektoren gegenüber, die bei den jeweiligen Parameterkombinationen notwendig waren, um die Trennebenen zu stützen. Bemerkenswert, wenngleich nachvollziehbar dabei ist die Tatsache, dass jene Kombinationen mit der besten Übereinstimmung auch die geringste Anzahl von Support-Vektoren

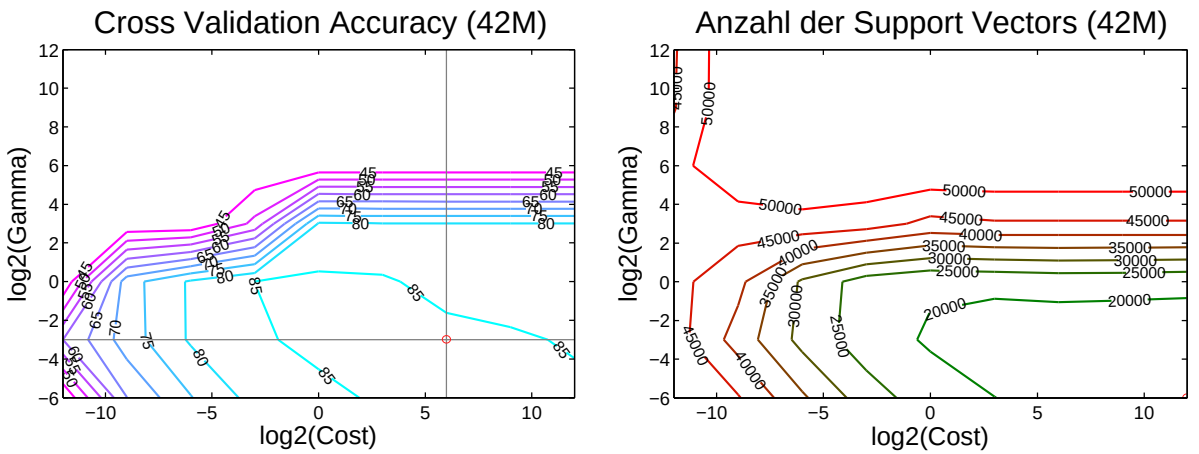


Abbildung 7.6: Cross Validation Accuracy und Anzahl der Stützvektoren für 54492 Vektoren mit 42 Merkmalen aus 8 Klassen

benötigten. Dieser Effekt kann auch mit anderen Daten und Problemstellungen nachvollzogen werden. Dies belegt wiederum, dass auch die Anzahl der Support-Vektoren zur Festlegung der Hyperebene als Indikator einer effizienten Klassifizierung herangezogen werden kann.

Die Methode führt zwar letztlich zum Erfolg, ein Nachteil ist jedoch der hohe Rechenaufwand. Für ein Netz wie in Abbildung 7.6 mit einer Schrittweite von 3 müssen immerhin $9 \times 7 = 63$ vollständige 5-fach Kreuzvalidierungen berechnet werden. Daraus ergeben sich 315 Trainings- und Testdurchgänge. Die Schrittweite sollte demnach nicht zu klein gewählt werden. Berechnungen mit Schrittweiten von 3 bis 5 im Bereich von -5 bis 15 für C und γ sind durchaus üblich. Werden zu große oder kleine Grenzwerte gewählt, so kommt es ohnehin zu einer Über- oder Unteranpassung, wie Abbildung 7.5 belegt.

Diese Methode der Parameteroptimierung eignet sich am ehesten beim Einsatz eines Linear- oder eines RBF-Kernels. Sigmoid- oder Polynom-Kernel bedürfen schon einer Iteration durch drei Variablen, wodurch der Aufwand und damit die Prozessierungsdauer noch einmal deutlich steigt. Zudem reagieren diese beiden Kernel empfindlich auf die Parametrisierung. Auf Maßnahmen gegen eine hohe Prozessierungsdauer wird auch im Abschnitt 7.9.6 noch einmal eingegangen.

7.8.3 Abhängigkeit der Parameter vom Kernel

Die folgende Untersuchung mit mehreren Kerneltypen soll Klarheit schaffen, inwieweit die ermittelte Parametrisierung kernelabhängig ist.

Beispiel: Für die gleichen Klassen und Merkmale wie in den Beispielen aus Abschnitt 7.7 wurde eine Parameteroptimierung mittels einer CVA berechnet. Beim Sigmoid- und

Polynomial-Kernel wurde aufgrund des Rechenaufwands auf die Variation des dritten Funktionsparameters verzichtet und dieser jeweils auf 0 festgelegt.

Ergebnisse: Abbildungen 7.7 zeigt die CVA einzelner Kernel. Die Diagramme belegen, dass jeder Kernel seine eigene Parametrisierung und damit Optimierung benötigt. Die Fadenkreuze markieren die optimale Parameterkombination, bei der die Cross Validation Accuracy ein Maximum erreicht. Deutlich wird auch, dass der Linear-Kernel nur den Parameter C besitzt. Dementsprechend ist der Aufwand der Parameteroptimierung in der Regel geringer als bei Kernel mit zwei oder gar drei Variablen.

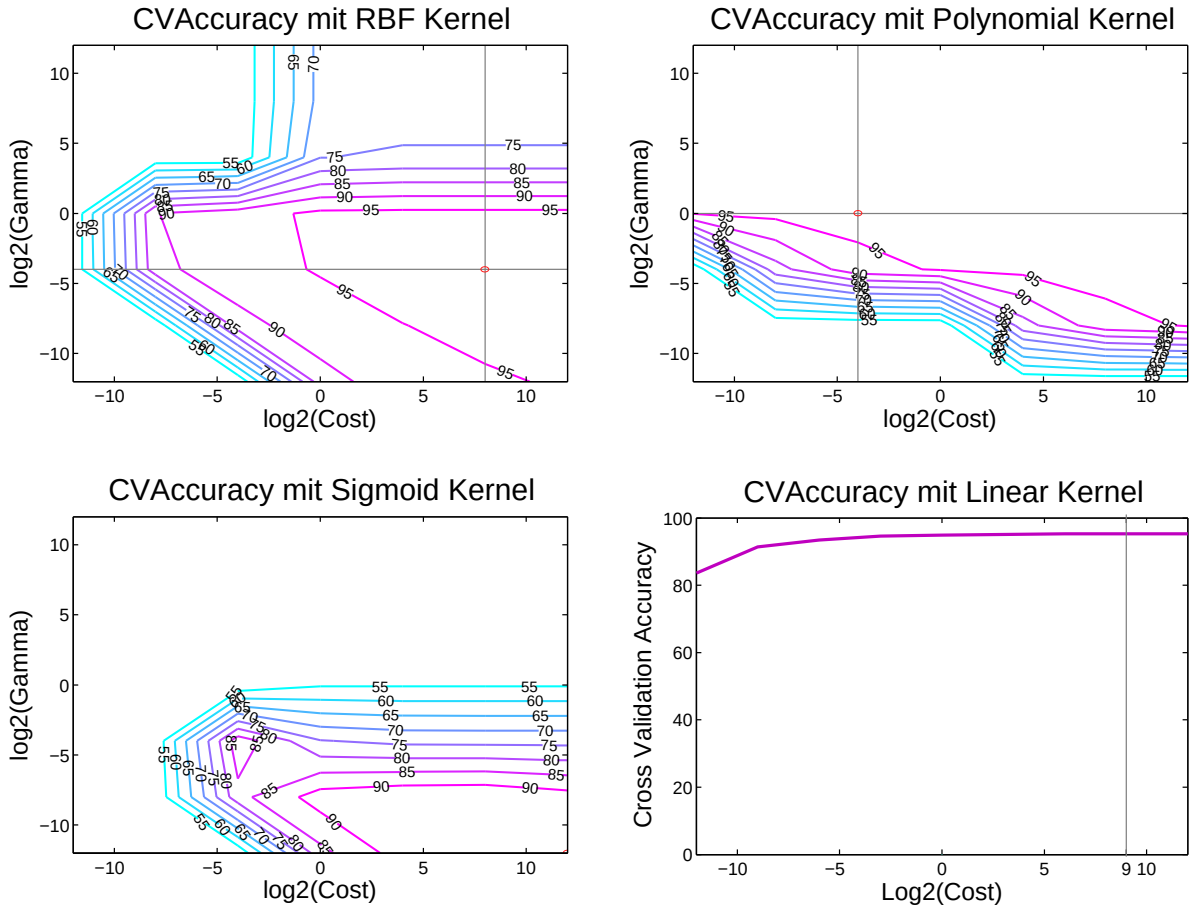


Abbildung 7.7: Cross Validation Accuracy in Abhängigkeit zum Kernel gerechnet mit 9702 Vektoren mit 42 Merkmalen aus 3 Klassen

7.8.4 Berechnung des Parameters γ am Beispiel eines RBF-Kernels

Aufgrund des hohen Rechenaufwands liegt der Wunsch nahe, zumindest einen der Parameter exakt oder zumindest näherungsweise berechnen zu können. Damit könnte die Bestimmung der verbleibenden Parameter deutlich beschleunigt werden. Die Autoren Wang et al. (2003) widmen sich diesem Anliegen und beschreiben ein Verfahren für

den RBF-Kernel (5.40), mit dem der Parameter Gamma $\gamma = 1/2\sigma^2$ über die Minimierung von Sigma σ in einem iterativen Verfahren gewonnen wird. Die Komplexität des Verfahrens wächst aber mit $O(4n^2)$ quadratisch mit der Anzahl n der Trainingsvektoren, womit auch diese Methode unter Berücksichtigung der Berechnungsdauer nur für einen eingeschränkten Problemkreis nützlich ist.

7.9 Optimierungsmöglichkeiten

Betrachtet man die Ergebnisse der vorangegangen Beispiele, so liegt es nahe, nach weiteren Optimierungsmöglichkeiten zu fragen. Ziel der Optimierungen soll (i) das Erreichen besserer Klassifikationen, (ii) eine Vereinfachung bei der Auswahl der Trainingsdaten und (iii) eine Verkürzung der Prozessierungsdauer sein. Die folgenden Abschnitte beleuchten diese Zielsetzungen unter Anwendung der vorhandenen Testdaten.

7.9.1 Transformation der Merkmale

Die *Hauptkomponentenanalyse* (HKA) (engl. *Principal Components Analysis*[9] – PCA) ist ein Verfahren der multivariaten Statistik mit dem Ziel, mehrdimensionale Datensätze derart zu komprimieren, dass bestimmte statistische Größen durch eine geringere Anzahl von Linearkombinationen, den sogenannten *Hauptkomponenten*, beschrieben werden können. Stark vereinfacht ausgedrückt und auf unseren Anwendungsfall bezogen erklärt kann die PCA mehrdimensionale, teils stark korrelierende Datensätze so vereinfachen, dass die Datensätze unkorreliert und in möglichst wenigen Dimensionen dargestellt werden können und möglichst viel der ursprünglichen Information trotzdem erhalten bleibt (Abe, 2010; Backhaus et al., 2011).

Realisiert wird diese Merkmalsreduktion durch ein konvergierendes, iteratives Verfahren, in dem eine „lineare Abbildung n-dimensionaler Merkmalsvektoren auf eine m-dimensionale Hyperebene“ (Handels, 2000) durchgeführt wird. Dabei wird die Hyperebene so positioniert, dass die Varianzen der einzelnen Komponenten maximiert werden und die Komponentenachsen orthogonal zueinander stehen. In einschlägiger Fachliteratur wird diese Forderung als *Varianzkriterium* bezeichnet (Backhaus et al., 2011; Handels, 2000).

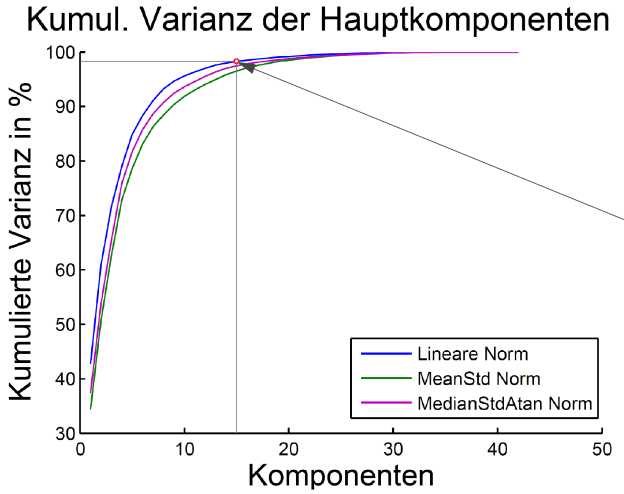
Das Ergebnis einer HKA ist eine durch möglichst wenige Faktoren bzw. Hauptkomponenten beschriebene Matrix $\mathbf{P}$ als Reproduktion der Merkmalsmatrix $\mathbf{M}$. Zum Unterschied von $\mathbf{M}$ kommt es durch die Transformation zu einer Ordnung der Hauptkomponenten

[9] Die PCA findet sich in einschlägiger Fachliteratur zur Bildverarbeitung oft auch unter der Bezeichnung *Karhunen-Loève-Transformation* (engl. *Karhunen-Loève expansion*).

nach abnehmender Standardabweichung, die als transformierte Merkmale interpretiert werden können.

Beispiel: Gerechnet wurde eine PCA über alle Vektoren der Testdaten mit 42 Merkmalen. Die Testdaten wurden vor der PCA mit unterschiedlichen Verfahren standardisiert (siehe Kapitel 7.5).

Ergebnis: Abbildung 7.8 zeigt die kumulierte Varianz der Hauptkomponenten der Ergebnismatrix. Wie aus der Abbildung und der rechts nebenstehenden Eigenwert-Tabelle hervorgeht, umfasst die erste Komponente schon rund 42, 7% der Gesamtvarianz. Schon die ersten 15 der 42 Komponenten beschreiben die Ausgangsmatrix zu 98.3%.



Eigenwert-Tabelle

Kom.	Total	%.Var.	Kum.Var
1	1,554	42,694	42,694
2	0,660	18,142	60,836
3	0,389	10,682	71,518
4	0,272	7,484	79,003
5	0,213	5,857	84,859
...			
10	0,034	0,934	95,607
...			
15	0,010	0,284	98,296
...			
19	0,006	0,165	99,054
...			
38	0,000	0,002	99,998
39	0,000	0,001	100,000
...			
42	0,000	0,000	100,000

Abbildung 7.8: Kumulierte Varianz der Hauptkomponenten

Die dekorrelierten Komponenten der Ergebnismatrix **P** gelten als Ersatz für die ursprünglichen Merkmale (Backhaus et al., 2011). Wesentlich ist der Hinweis, dass die Hauptkomponenten nicht als Äquivalent der Merkmale gelten können. Anders ausgedrückt lässt sich aus der Ergebnismatrix **P** nicht bestimmen, welchen Anteil die Merkmale der ursprünglichen Merkmalsmatrix **M** an **P** haben, dazu dient die Transformationsmatrix.

Neben der Hauptkomponentenanalyse gibt es noch einige weitere Transformationsverfahren, die ähnlich der PCA die Merkmalsvektoren dekorrelieren. Die Kernel-HKA (engl. *Kernel Principal Component Analysis - Kernel-PCA*) gilt als Erweiterung der PCA, führt jedoch wie bei Kernel-SVMs bestimmte Kernel-Methoden in die Berechnung ein, um eine lineare Abbildung der Merkmalsvektoren zu umgehen. Damit müssen die Merkmale der Vektoren nicht zwangsweise linear korrelieren, sondern können eine Transformation in einen mehrdimensionalen Merkmalsraum erfahren. Als Anwendungsgebiet können die selben Kriterien wie bei Kernel-SVMs gelten, also nicht lineare separierbare Muster (Scholkopf et al., 1999). Einen umfangreichen Vergleich mehrerer Verfahren geben die Autoren van der Maaten et al. (2009) in einem bisher unveröffentlichten, jedoch online verfügbaren Artikel.

7.9.2 Reduktion der Merkmale

Nachdem, wie im vorangegangenen Beispiel gezeigt, mit einer Hauptkomponentenanalyse die Komponenten nach deren Varianz geordnet werden und die Hauptkomponenten als Ersatz für die ursprünglichen Merkmale gelten, liegt die Idee nahe, anstatt der ursprünglichen Eingangsmatrix die Ergebnismatrix (=Ersatzmatrix) der PCA zur Klassifikation heranzuziehen.

Beispiel: Nach einer Hauptkomponentenanalyse der Eingangsdatensätze aus den Beispielen des Kapitels 7.8.2 wird ein CVA-Grid gerechnet. In diesem Fall werden jedoch nur 15 Hauptkomponenten anstatt der 42 vorhandenen Merkmale verwendet.

Ergebnisse: Abbildung 7.9 zeigt die Contour-Plots der Cross Validation Accuracy (links) und der Anzahl der Stützvektoren (rechts) für die jeweilige Parametrisierung. Im direkten Vergleich mit den Ergebnissen der Abbildung 7.6 aus Kapitel 7.8.2 fällt auf, dass die Isolinien gleicher CVA gerechnet mit der transformierten Ersatzmatrix mit nur 15 Merkmalen nahezu ident mit den ursprünglichen Isolinien der Berechnung mit der vollen Merkmalsanzahl verlaufen. Die maximale CVA liegt bei gleicher Parametrisierung bei der PCA-Variante mit $86,05\%$ auch nur knapp unter den $86,41\%$ der ursprünglichen Optimierung.

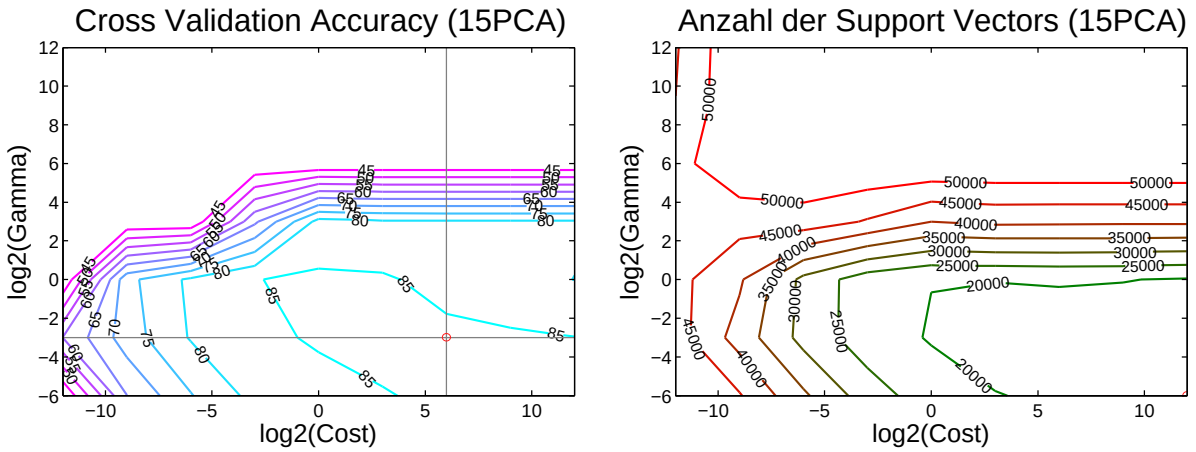


Abbildung 7.9: Cross Validation Accuracy und Anzahl der Stützvektoren für 54492 Vektoren mit 15 Hauptkomponenten aus 8 Klassen (vgl. Abb.7.6)

Überraschend an diesem Vergleich ist, dass sich selbst die Anzahl der Stützvektoren kaum ändert. Das Minimum liegt bei der PCA bei 16962 Stützvektoren und bei der ursprünglichen Variante mit der vollen Merkmalsanzahl bei 16992. Dieses Ergebnis wurde mit den selben Daten mehrfach verifiziert, sodass diese Ähnlichkeit damit erklärt werden könnte, dass die transformierte Hülle der Punktwolke im Gesamtkontext aller Vektoren eine ähnliche Form haben muss. Diese Vermutung gäbe genug Stoff für weitere Untersuchungen in Fortsetzung dieser Arbeit.

Die Gesamtdauer zur Berechnung der PCA-basierten CVA-Grids verkürzte sich auf $49,34\%$, also die Hälfte der ursprünglichen Prozessierungsdauer. Zwei Fragen sind bislang offengeblieben: (i) Wie weit kann die Anzahl der Merkmale reduziert werden, ohne einen negativen Einfluss auf die Erkennungsrate zu erfahren? (ii) Welchen Einfluss hat die Merkmalsreduktion auf die Prozessierungszeit? Die zweite Frage ist insofern nicht abwegig, da die Reduktion der Merkmale das Modell soweit verändert, dass der Trainings-Algorithmus der verwendeten RBF-SVM mitunter nicht mehr konvergiert. Zur Beantwortung dieser Fragen wurde wieder eine Beispielrechnung angestellt.

Beispiel: Mit aus dem vorherigen Beispiel ermittelten optimalen Parametrisierung wurde jeweils eine RBF-SVM trainiert und daraufhin Datensätze damit klassifiziert. Begonnen wurde mit der ersten Hauptkomponente. Nach jedem Zyklus wurde eine Hauptkomponente hinzugefügt und der Trainings- und Klassifikationsvorgang wiederholt.

Ergebnisse: Das im Kapitel 3.3.4 beschriebene Peaking-Phänomen wird mit den verwendeten Datensätzen bestätigt (siehe Abbildung 7.10). Mit einer einzigen Hauptkomponente, also $42,7\%$ der Gesamtvarianz (Abbildung 7.8), wird schon eine Overall Accuracy von 76% erreicht. Das Maximum wird mit 6 Hauptkomponenten (=Merkmalen) mit einer Overall Accuracy von 81% bei einer kumulierten Varianz von $88,36\%$ erreicht. Danach sinkt die Overall Accuracy wieder stetig bei sukzessiver Erweiterung der Merkmalsmatrix um je eine Hauptkomponente. Die Hinzunahme weiterer Hauptkomponenten wirkt sich nach dem Peak also kontraproduktiv aus.

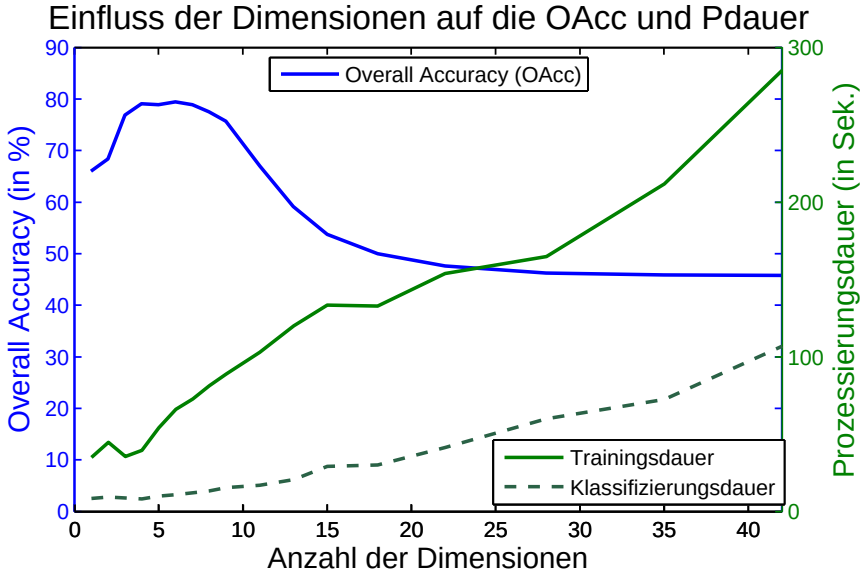


Abbildung 7.10: Overall Accuracy und Prozessierungsdauer in Abhängigkeit der Anzahl verwendeter Hauptkomponenten

Wie der Abbildung 7.10 auch zu entnehmen ist, steigen die Trainings- und Klassifizierungsdauer nahezu linear. Schwankungen sind dann zu erkennen, wenn der Trainingsalgorithmus

mehr oder weniger schnell konvergiert. Dieses Ergebnis bestätigt, dass die Anzahl der Merkmale nicht zwangsläufig das Klassifizierungsergebnis positiv beeinflusst, sondern „weniger oft mehr ist“. Die Prozessierungsdauer nimmt bei einer Dimensionsreduktion des Merkmalsraums in den meisten Fällen ab.

Neben der Reduktion des Merkmalsraums über eine Hauptkomponentenanalyse bietet beispielsweise auch der *plus l-take away r* Algorithmus als eine Kombination von *sequentiellem Vorwärtsalgorithmus* (SFS) und *sequentieller Rückwärtsselektion* (SBS) und seine Erweiterung, der *sequential floating forward search* (SFFS) Algorithmus (Pudil et al., 1994) die Möglichkeit einer gezielten Merkmalsreduktion. Für weitere Informationen dazu sei auf Guyon et al. (2006) verwiesen.

7.9.3 Reduktion der Trainingsdatensätze

In Kapitel 7.4 wurde die Auswahl der Datensätze zwar diskutiert, der Trainingsdatensatzumfang (engl. *sample size*) wurde jedoch nicht vollends geklärt. „...a 'bigger is better' attitude is often held by researchers“ halten (Camp-Valls & Bruzzone, 2009, S. 92) fest, verweist jedoch auch auf eine Klassifikator unabhängige mögliche näherungsweise Bestimmung der „required sampe size“. Gegen die Maximierung der Testdatenmenge spricht die Prozessierungsdauer, die mit dem Quadrat der Trainingsdatensatzanzahl zunimmt. Die aus den Forderungen nach einem Maximum der Erkennungsrate bei einem Minimum an Prozessierungsdauer abgeleitete Regel bzgl. der Anzahl der Trainingsdatensätze lautet: So wenig wie möglich und so viele wie notwendig. Den Einfluss der Trainingsdatensätze auf das Klassifikationsergebnis zeigt das folgende Beispiel.

Beispiel: Aus den Testdaten wurden für die Klassen `[12 53 54]` mit `[1:42]` Merkmalen aus jeder der drei Klassen jeweils `[500 1000 1500 2000 2500 3000]` Vektoren zufällig ausgewählt. Die Parametrisierung mit C und γ für einen RBF-Kernel wurde über ein CVA-Grid ermittelt. Mit den besten Parametersettings wurde danach ein Training und daraufhin eine Klassifizierung mit den verbleibenden `234+12762+9014=22010` Vektoren durchgeführt und die Overall Accuracy berechnet.

Ergebnisse: Tabelle 7.6 listet die wichtigsten Ergebnisse dieses Versuchs. Auffallend dabei ist, dass die im CVA-Grid maximal erreichte CVA mit einem Minimum von $81,66\%$ und einem Maximum von $84,75\%$ nicht stärker von der Anzahl der verwendeten Trainingsvektoren abhängt. Der Anteil der Stützvektoren ist bei jenem Rechendurchgang mit den `3x500=1500` Vektoren mit $41,07\%$ am höchsten. Beide Effekte würden sich damit erklären lassen, dass die verwendeten Vektoren die Klassenhüllen der Punktwolke im mehrdimensionalen Merkmalsraum gleichmäßig repräsentieren und sich an den Rändern der jeweiligen Klassen im Verhältnis in etwa die gleiche Anzahl an Stützvektoren bilden als im Inneren der jeweiligen Punktwolken. Der Anteil der Stützvektoren nimmt trotz einer

Ergebnis / Vektoren	1500	3000	4500	6000	7500	9000
Cross Validation Accuracy (in %)	81,66	83,66	84,20	84,56	85,04	84,75
mit Stützvektoren (abs.)	616	1153	1643	2208	2620	3201
mit Stützvektoren (in %)	41,07	38,43	36,51	36,80	34,93	35,57
bei log2(C)	8	8	12	4	12	12
bei log2(Gamma)	-8	-8	-8	-4	-8	-8
nach Prozessierungsdauer (in Sek.)	78	296	665	1158	1903	3104
Overall Accuracy (in %)	78,56	79,67	80,59	80,64	80,60	80,93

Tabelle 7.6: Cross Validation Accuracy bei Anwendung unterschiedlicher Kernel

Erhöhung der Trainingsvektoren ab, wenn die Hülle der Punktwolke schon mit wenigen Trainingsvektoren ausreichend beschrieben wird. Diese Hypothese wäre in weiteren Untersuchungen zu klären.

Die optimale Parametrisierung für C und γ ändert sich zwar in Abhängigkeit der Anzahl der Trainingsvektoren, wie aus Tabelle 7.6 und den Konturdiagrammen in Abbildung 7.11 aber auch hervorgeht, liegt das Optimum immer im selben Bereich. Ab etwa 4500 Vektoren aufwärts sind die Regionen gleicher CVA ähnlich ausgebildet. Wie Tabelle 7.6 ausweist, ist die Zeitersparnis bei der Parametersuche mit wenigen Vektoren deutlich, nimmt doch die Prozessierungsdauer bei n Vektoren mit etwa $O(n^2)$ zu. Eine Optimierung durch eine Reduktion der Trainingsvektoren kann bei entsprechenden Daten also durchaus sinnvoll sein.

Konf.matrix		Referenzdaten			Summe	Omiss. Error	Prod. Acc.
		Strassen(12)	Versiegelt(53)	Lock.Bau(54)			
Klassifikation	Strassen(12)	216	750	628	1594	86,45%	13,55%
	Versiegelt(53)	8	10355	1665	12028	13,91%	86,09%
	Lock.Bau(54)	10	1657	6721	8388	19,87%	80,13%
	Summe	234	12762	9014	22010		
	Comm.Error	7,69%	18,86%	25,44%			
	User's Acc.	92,31%	81,14%	74,56%			
	Total Acc.	78,56%					
	Kappa	0,5927					

Tabelle 7.7: Konfusionsmatrix für 3×3000 Trainingsvektoren aus 3 Klassen mit 42 Merkmalen und darauf folgender Klassifikation von 22010 Vektoren

Tabelle 7.7 zeigt Detailergebnisse der durchgeführten Klassifikationen mit den 22010 Testdatensätzen nach einem Training mit 9000 Vektoren in einer Konfusionsmatrix. Die Overall Accuracy liegt etwa 4% unter der Cross Validation Accuracy, was auf eine gute Generalisierung hinweist. Die Sensitivität liegt mit der User's Accuracy für alle drei Klassen im Bereich von $94,87\%$ (*Straßen*-Segmente) bis $76,20\%$ (*Lock.Bebauung*-Segmente). Die Producer's Accuracy zeigt bei Straßen-Segmenten einen niedrigen Wert von nur $18,93\%$

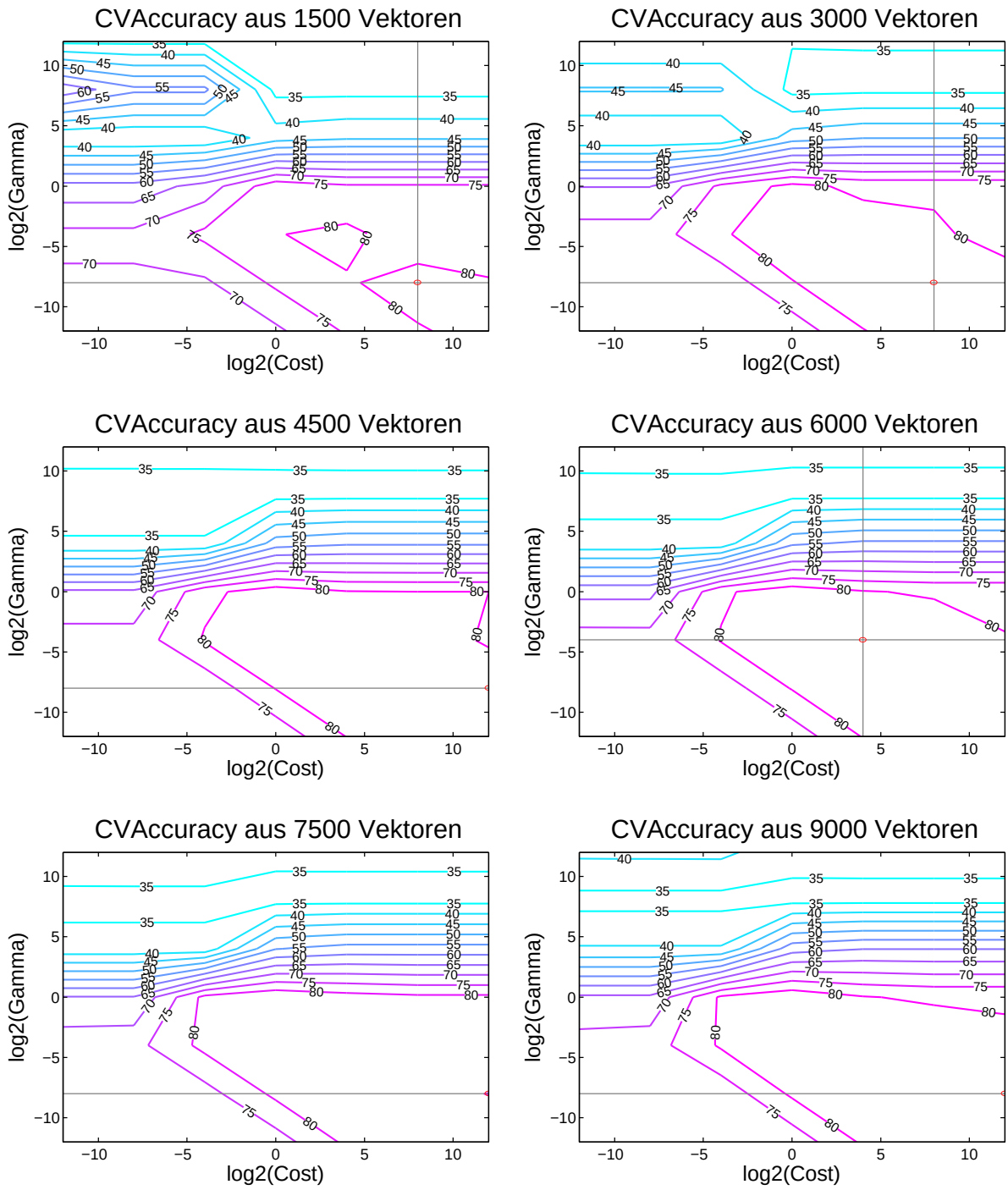


Abbildung 7.11: Cross Validation Accuracy in Abhängigkeit zur Vektorenanzahl gerechnet für 1500 bis 9000 Vektoren mit 42 Merkmalen aus 3 Klassen

auf. Dies liegt in erster Linie daran, dass sich jede fehlerhafte Klassifikation eines Segments als Straßen-Segment aufgrund der geringen Anzahl real vorhandener Strassen-Segmente überproportional stark auf die Producer's Accuracy für Straßen-Segmente auswirkt. Dieser Effekt tritt immer dann auf, wenn eine Klasse in einer Matrix besonders unterrepräsentiert

ist. In Relation zu den jeweiligen anderen Klassen sind die Falschklassifikationen jedoch im akzeptablen Bereich, wie die Werte des Omission Errors bestätigen.

7.9.4 Vermeidung des Class Imbalance Problems

Lernende Klassifikatoren, wie beispielsweise der *Decision Tree* Klassifikator, reagieren oft empfindlich auf eine Ungleichverteilung der Anzahl der Trainingsdatensätze pro Klasse. In der Praxis tritt dieser Fall sehr oft auf, zumal in den meisten Anwendungen die Trainingsdaten nicht generiert, sondern gesammelt werden. Die Anzahl der verfügbaren Trainingsdatensätze pro Klassen entspricht der Verteilung der einzelnen Klassen in der Realität. Eine Reihe von wissenschaftlichen Arbeiten (Japkowicz & Stephen, 2002; Lessmann, 2004) widmen sich seit Jahren diesem Problem - auch im Zusammenhang mit Support Vector Machines - und diskutieren mögliche Gegenmaßnahmen:

- **Upsizing**: Für jene Klassen, die in der Trainingsmenge schwach repräsentiert sind, werden durch ein *Resampling* oder durch eine einfache Duplikation zufällig ausgewählter Datensätze zusätzliche Trainingsdatensätze erzeugt.
- **Downsizing**: Von stark repräsentierten Klassen der Trainingsmenge wird nur ein Teil der verfügbaren Trainingsdatensätze verwendet.
- **Weighting**: Die ungleiche Verteilung der Klassen wird durch eine unterschiedliche Gewichtung der Klassen während des Trainingsvorgangs berücksichtigt.

Wie wirksam diese Methoden sein können, wird anahnd von Beispielen versucht.

Beispiel: Aus den Testdaten wurden für die Klassen `[12 52 53 54]` mit `[1:42]` Merkmalen aus jeder der Klassen jeweils `1000` Vektoren als Testdaten gesichert, die restlichen Daten wurden als Trainingsdaten verwendet. Die Trainingsdaten weisen mit `2234+9763+14762+11014=37773` Vektoren eine deutliche Ungleichverteilung auf. Nun wurde diese Ungleichverteilung durch Downsizing schrittweise reduziert. Für jedes Setting wurde ein CVA-Grid berechnet, mit den optimalen Parametern eine RBF-SVM trainiert und damit die Testdaten klassifiziert.

Ergebnisse: Wie die Tabelle 7.8 ausweist, wirkt sich die Reduktion der Vektoren in der Beispielrechnung positiv auf die Klassifikation aus. Beim Versuch der Spalte 1 mit einer massiven Ungleichverteilung der Klassen wird das Ergebnis der CVA-Grid-Optimierung nach erst nach 86:44 Min. geliefert. Eine Overall Accuracy (OA) von 83,13% konnte nach einer Trainingsdauer von 04:39 Min. erzielt werden. Wird die Anzahl der Vektoren der Klassen 53 und 54 schrittweise reduziert (Spalte 2 bis 5), so kommt es nicht nur zu einer massiven Verkürzung der Prozessierungszeiten, sondern auch zu einer stetigen Verbesserung der OA. Letztlich konnten bei einer Angleichung aller Klassen auf je 2234

Vektoren nach 02:31 Min. schon die optimalen Parameter aus dem CVA-Grid entnommen werden und nach einer Trainingsdauer von nur 2 Sekunden eine um 2, 37% bessere Overall Accuracy erreicht werden. Die Trainingsdatenzusammenstellung der Spalte 5∗ lieferte – bei gleicher Vektorenanzahl über alle Klassen – das beste Ergebnis.

Ergebnis / Versuch	**1**	**2**	**3**	**4**	**5***	**6**
Vektoren der Klasse 12	2234	2234	2234	2234	2234	2234
Vektoren der Klasse 53	14762	11630	8498	5366	2234	1607
Vektoren der Klasse 54	11014	8819	6624	4429	2234	1795
CVA-Grid Dauer (Min:Sek)	86:44	50:47	24:43	10:20	02:31	01:46
Max aus CVA-Grid (in %)	83,66	83,58	83,37	83,38	84,85	80,85
Anzahl der Stützvektoren	10497	8551	6717	4729	2532	1990
Stützvektoren (in %)	37,47	37,69	38,70	39,31	37,77	35,30
Trainingsdauer (Min:Sek)	04:39	03:03	01:34	00:55	00:02	00:11
Klassifikationsdauer (Min:Sek)	00:01	00:01	00:01	00:01	00:01	00:01
Overall Accuracy (in %)	83,13	84:06	84,33	84,63	85,50	84,83

Tabelle 7.8: Einfluss des Class Imbalance auf die Prozessierungsdauer, die Parameteroptimierung und die Klassifikationsergebnisse

Wie Tabelle 7.8 Spalte 6 zeigt, wird das Ergebnis sofort wieder schlechter, wenn die Vektoren der Klassen 53 und 54 weiter reduziert werden. Dieser Effekt ließ sich auch mit anderen Testdatenkombinationen nachvollziehen. Versuche mit anderen Kernel zeigten durchaus unterschiedliche Effekte. So hat sich die Veränderung der Vektorenanzahl beim Sigmoid-Kernel kaum ausgewirkt, wohingegen der Linear- und der Polynom-Kernel – wohlgemerkt bei den verwendeten Testdaten – noch deutlich positiver reagierte und Ergebnisverbesserungen von bis zu 10% auswies.

Die Ergebnisse bestätigen, dass auch mit einer SVM mitunter bessere Ergebnisse erreicht werden können, wenn eine Class Imbalance vermieden wird. Ein Downsizing wirkt schon alleine durch eine mitunter massive Verkürzung der Prozessierungsdauer positiv aus. Wie Berechnungen mit anderen Klassen aber auch offenlegten, gilt dies natürlich nur dann, wenn die Anzahl der Trainingsvektoren ausreichend ist. Stehen ohnehin schon zu wenige Trainingsvektoren zur Verfügung, so wirkt sich mitunter jede Reduktion der Vektorenanzahl negativ auf das Klassifizierungsergebnis aus. Eine exaktere Differenzierung der Effekte ist über eine Konfusionsmatrix möglich, mit der die Auswirkung auf jede Klasse einzeln nachgewiesen wird. Dies gilt natürlich nur dann, wenn die Kosten einer Fehlklassifizierung für alle Klassen gleich ist. Ansonsten wäre die Multiplikation der Konfusionsmatrix mit einer Kostenmatrix sinnvoll, in der die tatsächlich anfallenden Kosten je Klassenkombination eingetragen werden.

Keine deutliche Verbesserung der Klassifizierungsergebnisse konnte mit einem von LibSVM unterstützten C-Weighting erreicht werden, mit dem der Parameter C für jede Klasse Ω

auf Basis der Vektoranzahl N_Ω pro Klasse und der maximalen Vektoranzahl N_{max} über aller Klassen gewichtet wird:

$$C_\Omega = C * \frac{N_{max}}{N_\Omega} \tag{7.5}$$

Im vorherigen Beispiel hatte die klassenabhängige Gewichtung kaum Auswirkungen auf das Ergebnis. Die Overall Accuracy legte im Vergleich zu einem über alle Klassen einheitlichen C um lediglich 0.3% zu.

7.9.5 Manipulation des Parameters ε

Je nach Anzahl der Trainingsvektoren und der Merkmale können die Trainingsdauer und im Besonderen die Dauer zur Optimierung der Trainingsparameter mit der Berechnung eines CVA-Grids überproportional ansteigen. Der Grund für einen langen Trainingsprozess sind einerseits die wiederholten und aufwendigen Berechnungen, andererseits auch, dass der Algorithmus unter bestimmten Voraussetzungen nur langsam konvergiert. Das erste Problem kann mit der Sicherung der Zwischenergebnisse für nachfolgende Iterationen gemildert werden. LibSVM verwendet dafür einen sogenannten *dynamischen Cache* (Chang & Lin, 2011), dessen Aufgabe es ist, möglichst viele der aufwendig berechneten Zwischenergebnisse kurzfristig im Speicher abzulegen. Für alle Berechnungen in dieser Arbeit wurde der Cache auf 1.5 GByte limitiert[10].

Der zweite Grund für eine lange Trainingszeiten ist der Parameter C. Wird dieser groß gewählt, so konvergiert der Algorithmus langsamer (Chang & Lin, 2011). Dieser Parameter ist jedoch nicht beliebig variierbar, wenn ein optimales Ergebnis erreicht werden soll.

Eine zusätzliche Möglichkeit ist die Manipulation jenes Abbruchkriteriums ε (Epsilon), das die maximale Differenz der optimalen Lage der Hyperebene zur aktuell erreichten Lage festlegt. Wird dieser Abstand unterschritten, so bricht der Trainingsprozess den Optimierungsalgorithmus erfolgreich ab. Die Defaulteinstellung für ε ist 0.001. Wird ε vergrößert, so bricht der Algorithmus vorzeitig ab, auf die Gefahr hin, schlechtere Ergebnisse bei der Klassifikation einzufahren. Welche Auswirkung hat nun die Manipulation von ε auf das Ergebnis und auf die Trainingsdauer? Um dies festzustellen, wurde ein Beispiel gerechnet.

Beispiel: Aus den Testdaten wurde für insgesamt `41773` Vektoren der Klassen `[12 52 53 54]` mit `[1:42]` Merkmalen ein gesamter Zyklus, bestehend aus Parameteroptimierung (CVA-Grid), Training und Klassifizierung unter Änderung des Abbruchkriteriums ε simuliert.

[10] LibSVM bietet noch weitere Optionen zur Optimierung des Caches.

Ergebnisse: Aus Tabelle 7.9 geht hervor, dass die Klassifikationsleistung bei vergrößertem ε bei den verwendeten Beispieldaten kaum leidet, wohingegen die Prozessierungsdauer deutlich reduziert werden kann. Die Dauer zur Berechnung eines CVA-Grids sinkt stetig von 48:51 Minuten bis 23:47 Minuten, wohingegen die Overall Accuracy um nur 0.5% auf $74,70\%$ sinkt. Auffallend ist die schlagartige Verkürzung der Trainingsdauer ab $\varepsilon \geq 0.01$. Dies könnte damit erklärt werden, dass die Hyperebene sehr schnell eine sehr gute Lage einnimmt und auch damit, dass bei vielen Dimensionen schneller größere Abweichungen auftreten, selbst wenn die Genauigkeit schon hoch ist. Der Algorithmus konvergiert dann sehr rasch, benötigt dann aber viele Iterationsschleifen für eine nur geringfügige Verbesserung der Ergebnisse.

Ergebnis / Epsilon	**0.0001**	**0.001***	**0.01**	**0.1**	**0.5**	**1.0**
CVA-Grid Dauer (Min:Sek)	48:51	40:11	33:03	27:44	25:04	23:47
Max aus CVA-Grid (in %)	80,10	79,88	79,91	80,03	80,02	79,66
Trainingsdauer (Min:Sek)	01:21	00:54	00:04	00:06	00:03	00:03
Klassifikationsdauer (Min:Sek)	00:06	00:06	00:06	00:06	00:06	00:06
Overall Accuracy (in %)	74,94	74,94	74,86	74,84	74,89	74,70

Tabelle 7.9: Einfluss des Abbruchkriteriums ε auf die Prozessierungsdauer, die Parameteroptimierung und die Klassifikationsergebnisse

Eine Verringerung des Abbruchkriteriums auf $\varepsilon = 0.0001$ bringt keine Vorteile. Ganz im Gegenteil: Die Prozessierungsdauer steigt weiter, das Ergebnis ist aber kaum besser als mit der Defaulteinstellung $\varepsilon = 0.001$. Nach einer weiteren Vergrößerung des Abbruchkriteriums auf $\varepsilon > 2$ fällt die Klassifizierungsleistung drastisch ab. Eine behutsame Vergrößerung von ε gerade zur Abschätzung der Parameter mittels CVA-Grid ist aber - in Anbetracht der Zeitersparnis - durchaus zu empfehlen.

7.9.6 Parallelisierung der Algorithmen

Obige Ergebnisse belegen, dass die Reduktion der Merkmale und der Trainingsvektoren eine wesentliche Beschleunigung des Trainingsprozesses zur Folge haben kann. Große Datenmengen und eine ungünstige Daten- und Parameterkonstellationen münden bei aktueller Hardwareausstattung (siehe Anhang A) trotzdem schnell in eine mehrstündige Wartezeit. Die Berechnung eines CVA-Grids für einen RBF-Kernel für eine vollbesetzte Matrix mit 54492 Vektoren aus 8 Klassen mit 42 Merkmalen benötigte auf dem Testsystem gar bis zu 40 Stunden. Dabei hat gerade der Trainingsvorgang ein großes Optimierungspotential, wie dieses Kapitel belegt.

Eine Parallelisierung des Algorithmus wirkt sich bei umfangreichen Matrizen ebenso besonders positiv auf die Verarbeitungsgeschwindigkeit aus. Um dies zu belegen, wurden zwei Versuche bzw. Codeanpassungen angestellt:

Adaption 1: Für Testzwecke wurde die LibSVM vom Autor auf *OpenMP* (OpenMP.org, 2011) umgestellt und die Bibliotheksmodule sowohl für Matlab als auch für *eCognition* neu kompiliert und gelinkt. Mit einem Mehrkern-Prozessors vom Typ Intel Core 2 Quad Q9550 (2,83GHz, 1333MHz, 12MB)[11] konnte so die Prozessierung gegenüber der ursprünglichen Ausführung deutlich beschleunigt werden, da durch diese Anpassung zeitkritische Schleifen innerhalb der LibSVM in vier parallel laufenden Programmfäden (engl. *threads*) gleichzeitig ausgeführt werden. Messungen ergaben, dass damit bei größeren Datenmengen de-facto eine Verdoppelung der Ausführungsgeschwindigkeit erreicht werden kann.

Adaption 2: Eine nochmalige Beschleunigung kann der Einsatz eines Grafikprozessors (engl. *graphics processing unit* – GPU) mit sich bringen. Im direkten Vergleich mit einer voll ausgelasteten 4-Kern-Architektur des oben genannten Typs konnten einige der gezeigten Beispielberechnungen mit einer für eine GPU adaptierten LibSVM[12] (Athanasopoulos et al., 2011) und einer NVIDIA GeForce GTX 580[13] die Ausführungsdauer noch einmal - je nach Größe der Merkmalsmatrix - auf bis zu 30% der ursprünglichen Prozessierungsdauer reduziert werden.

Nur Teile des SVM-Algorithmus sind parallelisierbar, einer Beschleunigung der GPU Kernel sind demnach Grenzen gesetzt. Grundsätzlich gilt aber: Je größer die Merkmalsmatrix, desto mehr können parallelisierte Algorithmen auf GPUs oder auf Mehrkernprozessoren ihr Potential ausschöpfen. Ist die Anzahl der Vektoren zu gering, so wirkt sich die Parallelisierung weniger positiv auf die Prozessierungsdauer aus. Gerade bei GPUs tritt dieser Effekt verstärkt zutage, da ein Teil des Zeitgewinns als Overhead für den Datentransfer zur und aus der Grafikkarte und zur Synchronisation der einzelnen Threads wieder verloren geht. Bei GPUs wird mitunter bei kleinen Datenmengen durch diesen Overhead die Laufzeit sogar wieder verlängert.

[11] Intel: http://ark.intel.com/products/33924/Intel-Core2-Quad-Processor-Q9550-(12M-Cache-2_83-GHz-1333-MHz-FSB) (Zugriff: 2011-07-02)
[12] GPU-accelerated LIBSVM Version 1.1: http://mklab.iti.gr/project/GPU-LIBSVM (Zugriff: 2011-09-28)
[13] Nvidia: http://www.nvidia.com/object/product-geforce-gtx-580-us.html (Zugriff: 2011-11-02)

8 Zusammenfassung und Ausblick

Die vorliegende Arbeit beschreibt die Grundlagen der objektbasierten Klassifikation mit Support Vector Machines (SVMs). Fernerkundungsbilder bilden die Datenbasis der praktischen Beispiele und Evaluierung, auf eine eingehende Diskussion fernerkundungsspezifischer Klassifikationsmethoden wurde jedoch bewusst verzichtet, um den Fokus ganz auf SVM-relevante Details zu lenken. Die folgende Zusammenfassung skizziert kurz den Inhalt der Thesis, bevor sie sich dann gänzlich den im Kapitel 7, *Anwendung, Bewertung und Optimierung*, gewonnenen Erkenntnissen und der Beantwortung der in der Einleitung (Kapitel 1.4) definierten Fragen widmet.

8.1 Zusammenfassung und Schlussfolgerungen

Nach Erklärung der Motivation und Problemstellung (Kapitel 1) beginnt die Arbeit mit den Grundlagen der Bildverarbeitung (Kapitel 2) und einer kurzen Einführung in die Prinzipien der Klassifizierung (Kapitel 3). Erläuterungen zu Methoden der numerischen Klassifizierung (Kapitel 4) bereiten den Leser auf das mathematische Basiswissen von *Support Vector Machines* (Kapitel 5) und Kernelmethoden vor. Mit Hilfe eines für die Software *eCognition* entwickelten Plug-Ins und mehrerer MATLAB-Skriptdateien (Kapitel 6) wurden ausgewählte Bildobjekte mit Support Vector Machines klassifiziert, die Klassifikationsgüte bewertet und die Klassifikationsprozesse optimiert (Kapitel 7).

Die abschließende Interpretation sowohl der Evaluierungsergebnisse als auch der Optimierungsbestrebungen lassen folgende Schlussfolgerungen zu, die auch die eingangs definierten Fragen beantworten:

• Die Festlegung der Trainings- und Testdatensatzgröße (Kapitel 7.4) ist bei SVMs prinzipbedingt unkritisch, denn SVMs neigen kaum zu einem Übertraining. Positiv auf die Generalisierungsleistung wirkt sich die Vermeidung einer Ungleichverteilung der Klassen (Kapitel 7.9.4) aus. Eine gezielte Reduktion der Datensatzanzahl (Kapitel 7.9.3) senkt die Prozessierungsdauer merklich, bleibt jedoch nur ohne negative Folgen, solange das Klassifizierungsproblem noch hinreichend beschrieben ist. Als problematisch erweist sich die konkrete Festlegung der Trainingsdatensatzanzahl, zumal diese im direkten Zusammenhang mit der Anzahl der Klassen, der Dimension des Merkmalsraums und der Komplexität der Merkmalsverteilungen stehen.

• Die Auswahl der Merkmale (Kapitel 7.6) ist ebenso wesentlich. Sie einzig über deren Einzelseparationsfähigkeit zu bestimmen ist nicht zielführend, da oft erst die Kombination mehrerer Merkmale eine Separation im hochdimensionalen Merkmalsraum möglich macht. Von der Idee, die Dimensionalität des Merkmalsraums grundsätzlich zu maximieren, muss jedoch entschieden abgeraten werden. Oft führt eine Dimensionreduktion zum sog. Peaking-Phänomen (Kapitel 3.3.4), das heißt, die Reduktion der Merkmale (Kapitel 7.9.2) steigert entgegen der intuitiven Erwartungen die Leistung des Klassifikators.

• Als zur Merkmalsreduktion hilfreich erweist sich oft eine Transformation der Merkmale (Kapitel 7.9.1). Nach einer einfachen Hauptkomponentenanalyse (HKA) liegen die transformierten Merkmale (Hauptkomponenten) nach deren Varianz aufsteigend gereiht vor. Eine Klassifikation mit den höchstwertigsten Hauptkomponenten brachte mit den Testdaten oft vergleichbar gute oder gar bessere Klassifikationsergebnisse bei deutlich weniger Prozessierungsaufwand als mit allen verfügbaren und untransformierten Merkmalen.

• Mit Kernel kann der Klassifikator an die Problemstellung angepasst werden. Tests mit mehreren Kerneltypen (Kapitel 7.7) bestätigten, dass der RBF-Kernel als guter Allround-Kernel gelten kann, der sowohl bei einfachen als auch komplexen Merkmalsverteilungen eine vergleichsweise stabile Klassifikationsleistung erbringt. Der Linear-Kernel brachte zunehmend bessere Leistungen bei steigender Dimensionalität des Merkmalsraums. Sowohl der Sigmoid- als auch der Polynom-Kernel erwiesen sich bzgl. deren Parametrisierung als kritisch. Eine ungünstige Festlegung der Funktionsvariablen führt schnell dazu, dass bei beiden Kerneltypen die SVMs sehr langsam konvergieren und damit das Abbruchkriterium in einem akzeptablen Zeitraum nicht mehr erreicht wird.

• Die Parametrisierung der Kernel-SVM (Kapitel 7.8) ist absolut wesentlich für den erfolgreichen Einsatz von SVMs, jedoch leider weder trivial, da kaum vorhersehbar, noch unaufwendig. Mit einem heuristischen Verfahren zur Maximierung der Kreuzvalidierung können die Variablen C und γ eines RBF-Kernels zwar weitgehend automatisch geschätzt werden, der algorithmische Aufwand spiegelt sich jedoch in der Rechenzeit wieder. Sowohl Sigmoid- als auch Polynom-Kernel haben drei frei wählbare Parameter, die eine Parameterabschätzung noch aufwendiger und die beiden Kernel noch unattraktiver machen. Durch die Optimierung der Merkmalsanzahl (Kapitel 7.9.2) und der Vektorenanzahl (Kapitel 7.9.3) kann der Vorgang spürbar beschleunigt werden. Wirkung zeigt auch die Aufweitung des Abbruchkriteriums ε. Wird ε vergrößert, so bricht der zu Beginn schnell konvergierende SVM-Algorithmus mit einer beträchtlichen Zeitersparnis bei einer kaum spürbaren Verschlechterung der Overall Accuracy deutlich früher ab.

Diese Evaluierungsergebnisse belegen einerseits den Erfolg der Anwendung und die Leistungsfähigkeit von Support Vector Machines zur objektbasierten Klassifikation von Fernerkundungsbildern, zeigen andererseits aber auch die Grenzen der derzeitigen Implementierung und den Bedarf einer Fortführung der Entwicklung auf.

8.2 Ausblick auf mögliche Weiterentwicklungen

Gerade die Automatisierung des Klassifikationsprozesses und im Besonderen die Parametrisierung bedürfen in Zukunft noch einiger weiterer Verbesserungen, um sie für den gemeinen Anwender wirklich praxistauglich zu machen. Eine aus der Sicht des Anwenders vollautomatische Klassifizierung ohne oder mit nur geringfügigen manuellen Eingriffen scheint unter bestimmten Voraussetzungen und mit einer entsprechenden softwareseitigen und hardwareseitigen Optimierung und entsprechendem Aufwand durchaus möglich.

Die Vorverarbeitung und Optimierung der Trainings- und Testdatensätze lassen sich zu einem guten Teil automatisieren. Mit einer Teilmenge der Datensätze und einer schrittweisen Reduktion der Merkmale nach einer Hauptkomponententransformation lässt sich die optimale Anzahl der Merkmale (Hauptkomponenten) gut schätzen. Eine Berücksichtigung des Class-Imbalance-Problems ist zumindest mit einem Upsizing der Datensätze auch einfach und weitgehend automatisiert möglich. Auch hier kann mit einer Teilmenge der Testdaten vergleichsweise schnell überprüft werden, wie sich eine Änderung der Klassenverteilung auf das Klassifikationsergebnis auswirkt.

Die automatische Bestimmung der Kernel-Parameter für RBF-Kernel ist rechenaufwendig, aber möglich, zumal RBF-Kernel meist gutmütig auf unterschiedliche Parametrisierungsvarianten reagieren und damit die Parameter gut mit heuristischen Methoden geschätzt werden können.

Letztlich generieren alle genannten Automatisierungsbestrebungen gerade bei komplexen Merkmalsmatrizen einen beträchtlichen rechentechnischen Aufwand und verlangen damit nach einer Optimierung der zugrundeliegenden Prozesse. Einzelne Algorithmen für die Nutzung spezieller Hardware zu parallelisieren haben schon im Rahmen dieser Arbeit die Vorteile offengelegt.

Die weitere und konsequente Verbesserung und Automatisierung der komplexen Optimierungsschritte und die Parallelisierung aller Prozesse könnte selbst aufwendige mathematische Modelle wie Support Vector Machines in naher Zukunft noch attraktiver werden lassen und für die automatisierte Auswertung von Fernerkundungsdaten vermehrt in den Fokus der Anwender rücken.

Quellenverzeichnis

Abe, S. (2010[2]): Support Vector Machines for Pattern Classification. London: Springer-Verlag.

Athanasopoulos, A., Dimou, A., Mezaris, V. & I. Kompatsiaris (2011): GPU Acceleration for Support Vector Machines. Delft, The Netherlands. <http://www.iti.gr/~bmezaris/publications/wiamis11.pdf> (Zugriff: 2011-09-28).

Baatz, M., Hoffmann, C. & G. Willhauck (2008): Progressing from object-based to object-oriented image analysis. In: Blaschke, T., Lang, S. & G.J. Hay (Hrsg.): Object-Based Image Analysis. Lecture Notes in Geoinformation and Cartography. Berlin Heidelberg: Springer-Verlag, S. 29–42.

Backhaus, K., Erichson, B., Plinke, W. & R. Weiber (2011[13]): Multivariate Analysemethoden: eine anwendungsorientierte Einführung. Berlin: Springer-Verlag.

Baldenhofer, K. (2011): Lexikon der Fernerkundung. Online-Ausgabe: <http://www.fe-lexikon.info/> (Stand: Mai 2011; Zugriff: 2011-06-11).

Bartelme, N. (2005[4]): Geoinformatik - Modelle, Strukturen, Funktionen. Berlin: Springer-Verlag.

Beckmann, P., Ebers, O., Ebers, T., Hoffmann, R., M., P. & R. Spiridonidou (2007): Technischer Bericht des Grundlagenforschungsprojekts: Quantitative und qualitative Beschreibung von Personenbewegungen - Teilprojekt: 3D-Gesichtserkennung. Technische Universität Berlin - Institut für Mathematik. <http://www.math.tu-berlin.de/~plaue/EFRE_report.pdf> (Zugriff: 2011-08-12).

Bellman, R.E. (1961): Adaptive control processes - A guided tour. Princeton: Princeton University Press.

Ben-Hur, A., Ong, C., Sonnenburg, S., Schölkopf, B. & G. Rätsch (2008): Support Vector Machines and Kernels for Computational Biology. In: PLoS Comput Biol 4, 10, S. e1000173.

Benediktsson, J.A., Chanussot, J. & M. Fauvel (2007): Multiple classifier systems in remote sensing: from basics to recent developments. Proceedings of the 7th international conference on Multiple classifier systems. Berlin: Springer-Verlag, S. 501–512.

Bishop, C.M. (2006): Pattern Recognition and Machine Learning (Information Science and Statistics). New York: Springer-Verlag.

Blaschke, T. (2000): Objektextraktion und regelbasierte Klassifikation von Fernerkundungsdaten: Neue Möglichkeiten für GIS-Anwender und Planer. 5. Symposium „Computergestützte Raumplanung“ (CORP 2000). Wien.

Blaschke, T. (2010): Object based image analysis for remote sensing. In: Isprs Journal of Photogrammetry and Remote Sensing 65, S. 2–16.

Blaschke, T. & S. Lang (2006): Object based image analysis for automated information extraction - A synthesis. Measuring the Earth II ASPRS Fall Conference 6-10 November

2006. San Antonio, Texas.

Blaschke, T., Lang, S. & G. Hay (Hrsg.) (2008): Object-Based Image Analysis. Berlin: Springer-Verlag.

Blaschke, T. & J. Strobl (2001): What's wrong with pixels? Some recent developments interfacing remote sensing and GIS. In: GIS - Zeitschrift für Geoinformationssysteme 14, 6, S. 12–17.

Boser, B.E., Guyon, I.M. & V.N. Vapnik (1992): A Training Algorithm for Optimal Margin Classifiers. Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory. New York, USA: ACM Press, S. 144–152.

Boughorbel, S., Tarel, J.P. & N. Boujemaa (2005): Conditionally Positive Definite Kernels for SVM Based Image Recognition. In: Multimedia and Expo, IEEE International Conference on S. 113-116.

Bouziani, M., Goita, K. & D. He (2010): Rule-Based Classification of a Very High Resolution Image in an Urban Environment Using Multispectral Segmentation Guided by Cartographic Data. In: IEEE Transactions on Geoscience and Remote Sensing 48, 8, S. 3198–3211.

Bungartz, H.J. (2002): Einführung in die Computergraphik: Grundlagen, geometrische Modellierung, Algorithmen. Braunschweig: Vieweg.

Burbidge, R. & B. Buxton (2001): An Introduction to Support Vector Machines for Data Mining. Keynote Papers, Young OR12, University of Nottingham. Nottingham: Operational Research Society, S. 3–15.

Burges, C. (1998): A tutorial on support vector machines for pattern recognition. In: Data mining and knowledge discovery 2, 2, S. 121–167.

Byeungwoo, J. & D. Landgrebe (1993): Classification of Multispectral Image Data with Spatial-Temporal Context. Purdue University School of Electrical Engineering.

Byvatov, E. & G. Schneider (2003): Support vector machine applications in bioinformatics. In: Applied bioinformatics 2, 2, S. 67–77.

Camp-Valls, G. & L. Bruzzone (Hrsg.) (2009): Kernel Methods for Remote Sensing Data Analysis. UK: John Wiley & Sons, Ltd.

Chang, C.C. & C.J. Lin (2011): LIBSVM: A library for support vector machines. In: ACM Transactions on Intelligent Systems and Technology 2, S. 27:1–27:27. <http://www.csie.ntu.edu.tw/~cjlin/libsvm> (Zugriff: 2011-01-17).

Chapelle, O. & S.S. Keerthi (2010): Efficient algorithms for ranking with SVMs. In: Information Retrieval 13, S. 201–215.

Chapelle, O. & V. Vapnik (1999): Model Selection for Support Vector Machines. NIPS. Red Bank, S. 230-236.

Congalton, R. & K. Green (2009[2]): Assessing the Accuracy of Remotely Sensed Data: Principles and Practices. Boca Raton: CRC Press.

Cristianini, N. & E. Ricci (2008): Support Vector Machines. In: Kao, M. (Hrsg.): Encyclopedia of Algorithms. New York: Springer-Verlag.

Cristianini, N. & S.J. Taylor (2000): An introduction to support vector machines. Cambridge: Cambridge University Press.

De Lange, N. (2006²): Geoinformatik in Theorie und Praxis. Berlin Heidelberg: Springer-Verlag.

Definiens (2008): Definiens Developer XD 1.0 SDK Engine API - User Guide. München: Definiens AG.

Definiens (2011): Definiens Developer XD 1.5.1 Documentation - User Guide. München: Definiens AG.

DIN18716-3 (1997): Photogrammetrie und Fernerkundung - Teil 3: Begriffe der Fernerkundung. Berlin: DIN Deutsches Institut für Normung.

Duda, R.O. & P.E. Hart (1973): Pattern Classification and Scene Analysis. New York: John Wiley and Sons.

Duda, R.O., Hart, P.E. & D. Stork (2000²): Pattern Classification. New York: John Wiley and Sons.

Ferber, R. (2003): Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web. Heidelberg: dpunkt.verlag.

Gevantmakher, M. & C. Meinel (2004): Medizinische Bildverarbeitung - eine Übersicht. In: Universität Trier, Mathematik/Informatik, Forschungsbericht S. 1-29.

Gong, P. & P. Howarth (1992): Frequency-Based Contextual Classification and Gray-Level Vector Reduction for Land-Use Identification. In: Photogrammetric Engineering and Remote Sensing 58, 4, S. 423–437.

Graf, A. (2010): Nichtlineare Kernelmethoden zur Mustererkennung. Unveröffentlichte Masterthesis. Puch : Fachhochschule Salzburg.

Grouven, U., Bender, R., Ziegler, A. & S. Lange (2007): Der Kappa-Koeffizient. In: DMW - Deutsche Medizinische Wochenschrift 132, S. 65–68.

Guyon, I., Gunn, S., Nikravesh, M. & L. Zadeh (Hrsg.) (2006): Feature Extraction, Foundations and Applications. Berlin Heidelberg: Springer-Verlag.

Handels, H. (2000²): Medizinische Bildverarbeitung. Wiesbaden: Vieweg+Teubner Verlag.

Heinert, M. (2010a): Support Vector Machines - Teil 1: Ein theoretischer Überblick. In: Reiterer, A., Egly, U., Heinert, M. & B. Riedel (Hrsg.): Zeitschrift für Geodäsie, Geoinformation und Landmanagement 3. Vogtsburg-Oberrotweil, Germany: DVW e.V. - Gesellschaft für Geodäsie, Geoinformation und Landmanagement, S. 179-189.

Heinert, M. (2010b): Support Vector Machines - Teil 2: Praktische Beispiele und Anwendungen. In: Reiterer, A., Egly, U., Heinert, M. & B. Riedel (Hrsg.): Zeitschrift für Geodäsie, Geoinformation und Landmanagement 5. Vogtsburg-Oberrotweil, Germany: DVW e.V. - Gesellschaft für Geodäsie, Geoinformation und Landmanagement, S. 308-314.

Hermes, T. (2005): Digitale Bildverarbeitung - Eine praktische Einführung. München: Hanser.

Howley, T. & M.G. Madden (2005): The Genetic Kernel Support Vector Machine: Description and Evaluation. In: Artificial Intelligence Review 24, S. 379–395.

Hsu, C.W., Chang, C.C. & C.J. Lin (2010): A Practical Guide to Support Vector Classification. In: Bioinformatics 1, 1, S. 1–16. <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf> (Zugriff: 2011-04-03).

Hsu, C.W. & C.J. Lin (2002): A comparison of methods for multiclass support vector machines.

In: IEEE Transactions on Neural Networks 13, 2, S. 415–425.

Huang, L. & L. Ni (2008): Object-Oriented Classification of High Resolution Satellite Image for Better Accuracy. In: Zhang, J. & M. F.Goodchild (Hrsg.): Proceedings of the 8th International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences. Shanghai: ISARA - International Spatial Accurcy Research Association, S. 211–218.

Jacobson, K. (2006): Hochauflösende Satellitenbilder - ein gleitender Übergang zu Luftbildern. Festschrift zur 125-Jahr-Feier Geodäsie und Geoinformation: Wissenschaftliche Arbeiten der FR Geodäsie und Geoinformatik der Leibniz-Universität Hannover. Band 263. Hannover: Leibniz-Universität Hannover, S. 199–208.

Japkowicz, N. & S. Stephen (2002): The Class Imbalance Problem: A Systematic Study. In: Intelligent Data Analysis 6, 5, S. 429-449.

Jähne, B. (2005[6]): Digitale Bildverarbeitung. Heidelberg: Springer-Verlag.

Keerthi, S.S. & C.J. Lin (2003): Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. In: Neural Computation 15, S. 1667-1689.

Kittler, J. & J. Föglein (1984): Contextual classification of multispectral pixel data. In: Image Vision Computing 2, 1, S. 13-29.

Kröber M. et al. (2011): Wikiversity - FE Auswerteverfahren 1/Vegetationsindizes/Vegetationsindizes. Wikiversity - Plattform zum gemeinschaftlichen Lernen, Lehren, Nachdenken und Forschen. San Francisco: Wikimedia Foundation Inc. <http://de.wikiversity.org/wiki/Projekt:FE_Auswerteverfahren_1/Vegetationsindizes/Vegetationsindizes> (Stand: 2011-05-11; Zugriff: 2011-09-07).

Landis, J.R. & G.G. Koch (1977): The measurement of observer agreement for categorical data. In: Biometrics 33, 1, S. 159–174.

Lee, W.J., Verzakov, S. & R. Duin (2007): Kernel combination versus classifier combination. Proceedings of the 7th international conference on Multiple classifier systems. Berlin: Springer-Verlag, S. 22–31.

Lehman, T., Oberschelp, W., Pelikan, E. & R. Repges (1997): Bildverarbeitung in der Medizin. Berlin: Springer-Verlag.

Lehmann, T., Hiltner, J. & H. Handels (2005): Medizinische Bildverarbeitung. In: Lehmann, T.M. (Hrsg.): Handbuch der Medizinischen Informatik. München: Carl Hanser Verlag, S. 362–422.

Lessmann, S. (2004): Solving Imbalanced Classification Problems with Support Vector Machines. IC-AI - Proceedings of the International Conference on Artificial Intellicence. Las Vegas, USA: CSREA Press, S. 214-220.

Li, J., Najmi, A. & R. Gray (2000): Image classification by a two dimensional hidden Markov model. In: Acoustics, Speech, and Signal Processing, IEEE International Conference on 6, S. 3313-3316.

Lin, H.T. & C.J. Lin (2003): A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods. National Taiwan Univerity - Dept. of Computer Science and Information Engineering. <http://www.csie.ntu.edu.tw/~cjlin/papers/tanh.pdf>

(Zugriff: 2011-06-02).

Liu, J. & P. Mason (2009): Essential Image Processing and GIS for Remote Sensing. Essential (John Wiley & Sons). London, UK: Wiley-Blackwell.

Lohninger, H. (2011): Grundlagen der Statistik (elektronisches Lehrbuch). Pressbaum: Epina eBook. Online Ausgabe: <http://www.statistics4u.info/fundstat_germ/index.html> (Zugriff: 2011-07-17).

Lu, D. & Q. Weng (2007): A survey of image classification methods and techniques for improving classification performance. In: Int. Journal of Remote Sensing 28, S. 823–870.

Lucht, C., Nestler, R. & K. Franke (2009): Landnutzungsklassifikation auf multisensorischen und multispektralen Bilddaten - Anforderungen und Lösungsansätze. Beitrag zum: 15. Workshop Farbbildverarbeitung, 08.-09. Oktober 2009. Berlin.

van der Maaten, L.J.P., Postma, E.O. & H.J. van den Herik (2009): Dimensionality Reduction: A Comparative Review. Unveröffentlichtes Manuskript.

Manning, C., Raghavan, P. & H. Schütze (2009): An Introduction to Information Retrieval. Cambridge: Cambridge University Press. Online Ausgabe: <http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf> (Zugriff: 2011-04-08).

MathWorks (2011): MEX-files Guide - MATLAB Product Support (Online). <http://www.mathworks.de/support/tech-notes/1600/1605.html> (Zugriff: 2011-04-09).

Melgani, F. & L. Bruzzone (2008): Classification of Hyperspectral Remote Sensing Images With Support Vector Machines. In: IEEE Transactions on Geoscience and Remote Sensing 42, 8, S. 1778–1790.

Möller, M. (2011): Systemvergleich hochauflösender optischer Satellitenfernerkundungssensoren. In: Strobl, J., Blascke, T. & G. Griesebner (Hrsg.): Angewandte Geoinformatik - Beiträge zum 23. AGIT-Symposium Salzburg. Berlin: Wichmann, S. 66–75.

Mountrakis, G., Im, J. & C. Ogole (2010): Support vector machines in remote sensing: A review. In: ISPRS Journal of Photogrammetry and Remote Sensing 66, 3, S. 1–13.

Navulur, K. (2006): Multispectral image analysis using the object-oriented paradigm. London: CRC Press.

Niemann, H. (2003^2): Klassifikation von Mustern. Berlin: Springer-Verlag.

Noble, W.S. (2004): Support vector machine applications in computational biology. Computational molecular biology. Cambridge: MIT Press. Kapitel 3.

OpenMP.org (2011): The OpenMP API specification for parallel programming. <http://openmp.org/wp/> (Zugriff: 2011-11-03).

Peijun Li, Jiancong Guo, H.X. & X. Xiao (2008): Multilevel object based image classification over urban area based hierarchical image segmentation and invariant moments. In: International Archives of Photogrammetry XXXVIII-4/C1.

Pfeifenberger, M. (2010): Lineare Support Vector Machines. Unveröffentlichte Diplomarbeit. Puch: Fachhochschule Salzburg.

Platt, J.C. (1999): Fast training of support vector machines using sequential minimal optimization. Cambridge, MA, USA: MIT Press, S. 185–208.

Pudil, P., Ferri, F.J., Novovicova, J. & J. Kittler (1994): Floating Search Methods

for Feature Selection with Nonmonotonic Criterion Functions. In Proceedings of the 12th International Conference on Pattern Recognition (IAPR). Jerusalem , Israel, S. 279–283.

Rostock, U. (2011): Geoinformatik Service - Lexikon. Online-Ausgabe: <http://www.geoinformatik.uni-rostock.de/> (Stand: Dezember 2003; Zugriff: 2011-06-19).

Scholkopf, B. & A.J. Smola (2001): Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA, USA: MIT Press.

Scholkopf, B., Smola, A.J. & K.R. Müller (1999): Kernel principal component analysis. In: Advances in kernel methods: support vector learning S. 327–352.

Sewell, M. (2011): Support Vector Machines - The History of Support Vector Machines. <http://www.svms.org/history.html> (Zugriff: 2011-05-21).

Shawe-Taylor, J. & N. Cristianini (2004): Kernel Methods for Pattern Analysis. Cambridge: Cambridge University Press.

Souza, C. (2010): /cesarsouza/blog - Kernel Functions for Machine Learning Applications. <http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html> (Zugriff: 2011-08-23).

Steinwart, I. & A. Christmann (2008): Support Vector Machines. Berlin Heidelberg: Springer-Verlag.

Tönnies, K. (2005): Grundlagen der Bildverarbeitung. München: Pearson Education Inc..

Tulyakov, S., Jaeger, S., Govindaraju, V. & D. Doermann (2008): Review of Classifier Combination Methods. In: Simone Marinai, H.F. (Hrsg.): Studies in Computational Intelligence: Machine Learning in Document Analysis and Recognition. Berlin: Springer, S. 361-386.

Uhl, A. (2005): Grundlagen der Bildverarbeitung. Vorlesungsunterlagen WS2005/2006. Salzburg: Paris Lodron Universität, <http://www.cosy.sbg.ac.at/~uhl/imgProcess.pdf> (Zugriff: 2011-04-28).

Vapnik, V. (1995): The nature of statistical learning theory. New York: Springer-Verlag.

Vapnik, V. (1998): Statistical Learning Theory. Chichester: Wiley-Interscience.

Vapnik, V. & A. Chervonenkis (1974): Theory of Pattern Recognition (in Russisch). Moscow: Nauka. (Deutsche Übersetzung: W. Wapnik & A. Tscherwonenkis, Theorie der Zeichenerkennung, Akademie-Verlag, Berlin, 1979)..

Wang, L. (2005): Support Vector Machines: Theory and Applications. Studies in Fuzziness and Soft Computing 177. Heidelberg, Germany: Springer Berlin.

Wang, W., Xu, Z., Lu, W. & X. Zhang (2003): Determination of the spread parameter in the Gaussian kernel for classification and regression. In: Neurocomputing 55, 3-4, S. 643–663.

Waske, B., Fauvel, M., Benediktsson, J. & J. Chanussot (2009): Machine Learning Techniques in Remote Sensing Data Analysis. In: Camps-Valls, G. & L. Bruzzone (Hrsg.): Kernel Methods for Remote Sensing Data Analysis. Chichester, UK: John Wiley & Sons, Ldt, S. 3–24.

Abkürzungsverzeichnis

API	Application Programming Interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
CNES	Centre National d'Études Spatiales
CPD	Conditionally Positive definite
DLR	Deutschen Zentrums für Luft und Raumfahrt
FCC	False Color Composite
FIR	Far Infrared (15 μm bis 1 mm)
GPU	Graphics Processing Unit
HKA	Hauptkomponentenanalyse
HMM	Hidden Markov Model
KKT	Karush-Kuhn-Tucker
KNN	Künstliches Neuronales Netz
LWIR	Long-Wavelength Infrared (8 bis 15 μm)
MEX	Matlab Executables
MNDVI	Modified Normalized Difference Vegetation Index
MSAVI	Modified Soil Adjusted Vegetation Index
MODIS	Moderate-resolution Imaging Spectroradiometer
NDVI	Normalized Differenced Vegetation Index
NIR	Near-Infrared (780 nm bis 1400 nm)
OBIA	Object Based Image Analyzing
PCA	Principal Component Analysis
PSD	Positive Semi-definite
RBF	Radial Basic Function
SMO	Sequential Minimal Optimization
SPOT	Satellite Pour l'Observation de la Terre
SVM	Support Vector Machine (Stützvektormaschine)
SWIR	Short-Wavelength Infrared (1.4 bis 3.0 μm)
TCC	True Color Composite
VIS	Visible

A Tabellen und Abbildungen

Soft-Margin SVM in Abhängigkeit des Werts C

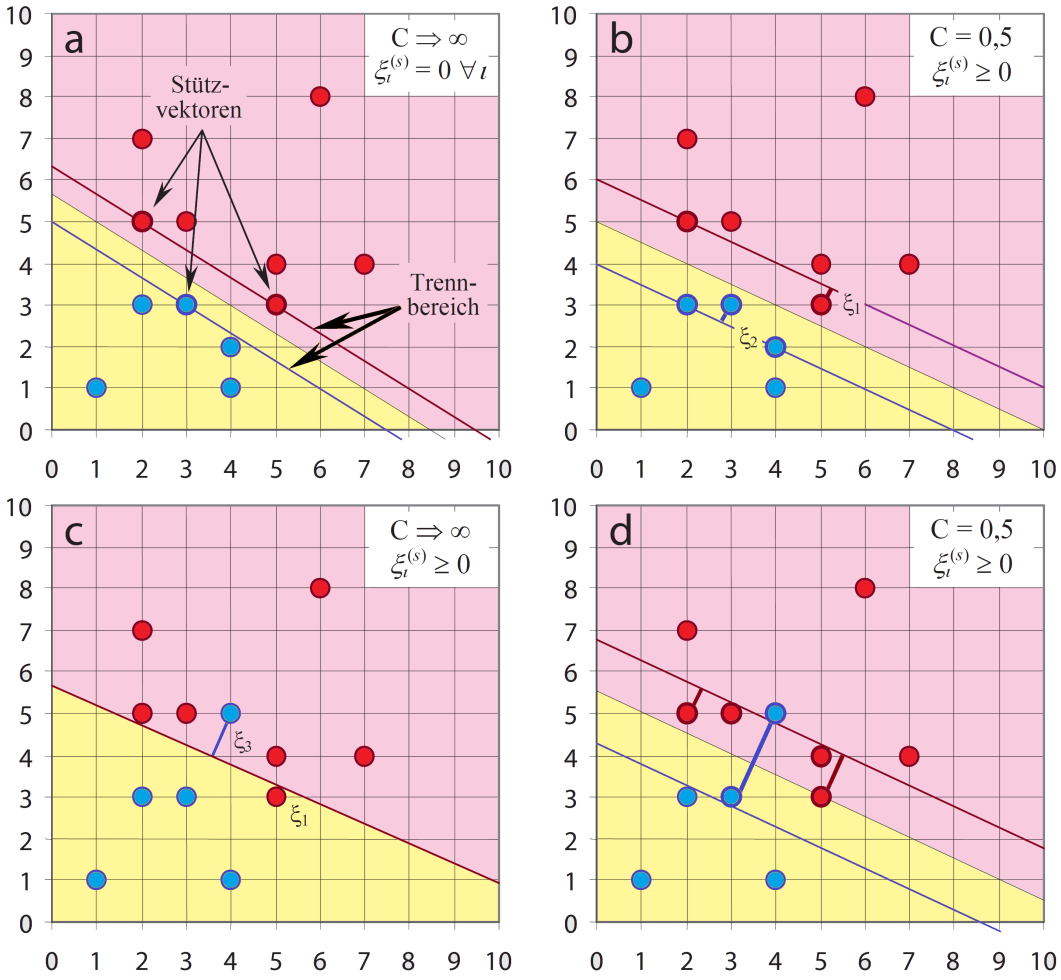


Abbildung A.1: Trennung eines Musters mit Soft-Margin SVMs in Abhängigkeit des abstrakten Fehlergewichts C: Bild (a) und (b) zeigen linear separierbare Muster, Bild (c) und (d) zeigen linear nicht separierbare Muster. Bei einem sehr großen C existiert kein Trennbereich (a und c), wenn $C \to \infty$ entspricht die SVM einer Hard-Margin SVM. Je kleiner C wird (b und d), umso mehr Stützvektoren sind im Trennbereich zwischen den beiden Klassen. Bei unendlich kleinem C sind alle Muster innerhalb des Trennbereichs. Adaptiert aus Heinert (2010b).

Kriterien zur Auswahl von Klassifikationsmethoden

Kriterien	Kategorie	Charakteristik	Möglicher Klassifikator
Sind Trainigs-daten vorhanden oder nicht?	Klassifikatoren mit unterstütztem Lernen	Landbedeckungsklassen sind definiert. Klassifizierte Referenzdaten sind vorhanden und können zum Trainieren verwendet werden. Die Signatur der Trainingsdaten bestimmt die Parametrisierung des Klassifikators.	Maximum Likelihood (ML) Minimum Distance (MD) Künstliches Neuronales Netzwerk (kNN) Decision Tree (DT)
	Klassifikatoren ohne unterstütztes Lernen	Cluster-basierte Algorithmen werden zur Aufteilung der spektralen Bilder auf Basis statistischer Größen benutzt. Der Benutzer hat in einem nachfolgen Arbeitsschritt die generierten Klassen bei etwaigen Inhomogenitäten zu größeren Klassen zusammenzufassen und die Klassen mit einem Bezeichner zu versehen.	K-Means Clustering Iterative Self-Organizing Data Analysis Technique (ISODATA)
Werden Parameter wie beispielsweise der Mittelwert-vektor oder die Kovarianzmatrix benutzt oder nicht?	Parameterische Klassifikatoren	Gauss-Verteilung der Merkmalswerte werden angenommen. Die Parameter (z.B. Mittelwertvektor, Kovarianzmatrix) wird aus den Trainingsdaten generiert. Probleme bei kleinsegmentigen komplexen Bildern, da parametrische Klassifikatoren dann zu *'noisy results'* neigen. Die Integration ergänzender, kontextabhängiger nicht statistischer Daten ist schwierig.	Maximum Likelihood (ML) Lineare Diskriminanten Analyse (LDA)
	Nicht-parametrische Klassifikatoren	Kein Vorwissen bzgl. der Daten notwendig. Die Parametrisierung der Klassifikatoren basiert auf keine statischen Annahmen oder Parametern. Gut geeignet zur Beimengung zusätzlicher nicht-fernerkundeten (thematischen) Merkmalsdaten.	Künstliches Neuronales Netzwerk (kNN) Decision Tree (DT) Support Vector Machine (SVM) Evidential Reasoning Experten Systeme
Welche Bildpunkte werden verwendet?	Pixel-basierte Klassifikatoren	Pixelbasierte Klassifizierer verwenden die spektralen Informationen der Bildpunkte oder deren Kombination aus den Trainingsdaten. Unberücksichtigt bleiben dabei oft sog. Mischpixel, die aufgrund der Auflösung der Sensoren an den Klassengrenzen Zwischenwerte einnehmen, die keiner Klasse zugerechnet werden können.	Anwendung der meisten genannten Klassifikatoren wie Maximum Likelihood, Maximum Distance, Künstliche Neuronale Netzwerke, Decision Tree und Support Vector Machine.
	Subpixel-basierte Klassifikatoren	Der Spektralwert eines Pixels wird als Kombination mehrerer Subpixel-Spektralwerte angenommen, die den jeweiligen Klassen zugeordnet werden.	Fuzzy-Set-Klassifikator spezieller Subpixel Klassifikator (z.B. Erdas IMAGINE) Spectral Mixture Analysis (SMA)

Tabelle A.1: Kriterien zur Auswahl von Klassifikatoren - Blatt 1. Überarbeitet und ergänzt aus Lu & Weng (2007).

Kriterien	Kategorie	Charakteristik	Möglicher Klassifikator
Welche Art der Pixelinformation wird verwendet?	Objekt-basierte Klassifikatoren	Segmentierung des pixelbasierten Bildes fasst die einzelnen Bildpunkte zu Objekten zusammen. Aus spektralen Informationen der Pixel werden Mittelwerte, Minima, Maxima, Standardabweichungen, usw. weiterverabeitet. Klassifikation auf Basis der Objekte. Neben spektralen Merkmalen können auch die Form, Textur, Größe usw. verwendet werden.	Nearest Neighbor Künstliche Neuronale Netze Support Vector Machine
	Per-Field Klassifikatoren	Überlagerung von raster- und vektorbasierten Daten. Vektordaten definieren Felder, deren spektrale Eigenschaften aus dem Image errechnet werden. Vorteil ist die Unabhängigkeit bzgl. spektraler Inhomogenitäten. Nachteil ist, dass a priori die zum Image passenden Feldgrenzen bekannt sein müssen; ansonsten ähnlich wie objekt-basierte Klassifizierung.	GIS-basierte Klassifikatoren ansonsten gleich wie objektbasierte Klassifikatoren.
Ist eine definitive Klassifikationentscheidung erwünscht oder nicht?	Harte Klassifikatoren	Jeder Bildpunkt oder jedes Segment wird eindeutig einer Klasse zugeschrieben. Gerade bei pixelbasierten Klassifikationen oder kleinen Segmenten kann eine Flächenberechnung aufgrund der Randbereiche (Mischpixelproblem) große Abweichungen zur Folge haben.	Anwendung der meisten genannten Klassifikatoren, wie Maximum Likelihood, Maximum Distance, Künstliche Neuronale Netzwerke, Decision Tree und Support Vector Machine.
	Weiche Klassifikatoren	Jedem Bildpunkt oder Segment wird ein Wert zugewiesen, der die Ähnlichkeit zu einer Klasse angibt. Das Klassifikationsergebnis gibt mehr Information preis und die Klassifikationsergebnisse sind in der Regel besser, gerade wenn die Auflösung relativ gering ist und damit eine große Anzahl von Mischpixel vorliegen.	Fuzzy-Set-Klassifikatoren Subpixel und Spectral Mixture Klassifikatoren
Wird zusätzliche raumbezogene Information verwendet oder nicht?	Spektrale Klassifikatoren	Ausschließlich die spektralen Informationen werden zur Klassifizierung herangezogen. Problematisch ist dabei eine mögliche Inhomogenität der spektralen Daten innerhalb einer Klasse, die sich mitunter auf das Klassifizierungsergebnis durch *'noisy results'* niederschlägt.	Maximum Likelihood Minimum Distance Künstliches Neuronales Netzwerk
	Kontextuelle Klassifikatoren	Berücksichtigung räumlicher Nachbarschaftsbeziehungen auf Pixel- oder Segmentebene.	Iterated Conditional Modes (ICM) Point-To-Point Contextual Correction Frequency-based Contextual Classifier
	Spektral-kontextuelle Klassifikatoren	Sowohl die spektralen als auch raumbezogenen Informationen fließen in die Klassifikation ein. Mit parametrischen oder nicht-parametrischen Klassifikatoren wird eine Vorklassifizierung durchgeführt, bevor in einem zweiten Arbeitsschritt kontextuelle Informationen einfließen, die die endgültige Klassifikation bestimmen.	Kombination (nicht-)parametrischer und kontextueller Algorithmen

Tabelle A.2: Kriterien zur Auswahl von Bildklassifikationsmethoden - Blatt 2. Überarbeitet und ergänzt aus Lu & Weng (2007).

Kernelkombinationen zur Konstruktion eigener Kernel

Sind zwei gültige Kernel $\mathcal{K}_1(\mathbf{x}, \mathbf{z})$ und $\mathcal{K}_2(\mathbf{x}, \mathbf{z})$ gegeben, so können daraus mit den folgenden Gleichungen neue, wiederum gültige Kernel erzeugt werden (Bishop, 2006):

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = c\mathcal{K}_1(\mathbf{x}, \mathbf{z}) \tag{A.1}$$
für $c > 0$
für c ist konstant

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})\mathcal{K}_1(\mathbf{x}, \mathbf{z})f(\mathbf{z}) \tag{A.2}$$
für $f(\cdot)$ ist eine Funktion

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = q(\mathcal{K}_1(\mathbf{x}, \mathbf{z})) \tag{A.3}$$
für $q(\cdot)$ ist ein Polynom mit Koeffizienten ≥ 0

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(\mathcal{K}_1(\mathbf{x}, \mathbf{z})) \tag{A.4}$$

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z}) + \mathcal{K}_2(\mathbf{x}, \mathbf{z}) \tag{A.5}$$

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z})\mathcal{K}_2(\mathbf{x}, \mathbf{z}) \tag{A.6}$$

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_3(\phi(\mathbf{x}), \phi(\mathbf{z})) \tag{A.7}$$
für $\mathcal{K}_3(\cdot, \cdot)$ ist ein gültiger Kernel auf $\mathbb{R}^n$
für ϕ ist eine Abbildung von $\mathcal{X}$ auf $\mathbb{R}^n$

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{A}\mathbf{z} \tag{A.8}$$
für $\mathbf{A}$ ist eine positive semidefinite Matrix

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_a(\mathbf{x}_a, \mathbf{z}_a) + \mathcal{K}_b(\mathbf{x}_b, \mathbf{z}_b) \tag{A.9}$$
für $\mathbf{x}_a$ und $\mathbf{x}_b$ bestehen aus Komponenten von $\mathbf{x}$
für $\mathbf{z}_a$ und $\mathbf{z}_b$ bestehen aus Komponenten von $\mathbf{z}$
für $\mathcal{K}_a$ und $\mathcal{K}_b$ sind gültige Kernel auf $\mathbb{R}^{\tilde{n}}$

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_a(\mathbf{x}_a, \mathbf{z}_a)\mathcal{K}_b(\mathbf{x}_b, \mathbf{z}_b) \tag{A.10}$$
für $\mathbf{x}_a$ und $\mathbf{x}_b$ bestehen aus Komponenten von $\mathbf{x}$
für $\mathbf{z}_a$ und $\mathbf{z}_b$ bestehen aus Komponenten von $\mathbf{z}$
für $\mathcal{K}_a$ und $\mathcal{K}_b$ sind gültige Kernel auf $\mathbb{R}^{\tilde{n}}$

Eine ausführliche Diskussion zum Thema *Kernel-Engineerings* findet sich in Shawe-Taylor & Cristianini (2004).

Entwicklungs- und Testumgebung

Die Implementierung und das Debugging der Applikationen und Scripte, sowie die Evaluation der Ergebnisse erfolgte in folgender Umgebung:

- **Hardware**: Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz[1]
  8,00 GB RAM, 2 TB Festplatte
  Optional: Grafikkarte NVIDIA GeForce GTX 580[2]
- **Windows 7 Enterprise**: 64 Bit (US engl.),
  Windows Version 6.1.7601, Service Pack 1
- **Definiens Developer XD**[3]: Version 1.5.1 (Build 1736)
- **Visual Studio 2010 Ultimate**[4] und **Visual Studio 2005 Prof.**:
  Version 10.0.30319.1 RTMRel, Service Pack 1[5]
  aus Kompatibilitätsgründen mit dem Definiens Developer XD musste das Platform Toolset 8.0 aus Visual Studio 2005 nachinstalliert und manuell in Visual Studio 2010 integriert werden
- **LIBSVM**[6]: Version 3.1 (April 2011)
  Optional: **GPU-accelerated LIBSVM**[7]: Version 1.1 (10/12/2010)
- **Mathlab**[8]: 7.12.0.625 (R2011a), 64 Bit

[1] Intel: http://ark.intel.com/products/33924/Intel-Core2-Quad-Processor-Q9550-(12M-Cache-2_83-GHz-1333-MHz-FSB) (Zugriff: 2011-07-02)
[2] Nvidia: http://www.nvidia.com/object/product-geforce-gtx-580-us.html (Zugriff: 2011-11-02)
[3] http://developer.definiens.com/ (Zugriff: 2011-04-25)
[4] Visual Studio 2010 von Microsoft: http://www.microsoft.com/germany/visualstudio/products/team/visual-studio-ultimate.aspx (Zugriff: 2011-06-05)
[5] Visual Studio 2010 - Service Pack 1: siehe http://msdn.microsoft.com/de-de/vstudio/aa718359 (Zugriff: 2011-05-06)
[6] LIBSVM – A Library for Support Vector Machines: http://www.csie.ntu.edu.tw/~cjlin/libsvm/ (Zugriff: 2011-07-07)
[7] GPU-accelerated LIBSVM – LIBSVM Accelerated with GPU using the CUDA Framework : http://mklab.iti.gr/project/GPU-LIBSVM/ (Zugriff: 2011-09-28)
[8] MATLAB von The MathWorks, Inc.: siehe http://www.mathworks.de/ (Zugriff: 2011-05-07)

Verfügbare Merkmale

Aus den in eCognition zur Vefügung stehenden segmentierten Testdaten wurden für jedes der 93724 Objekte die in der folgenden Tabelle aufgelisteten 46 Eigenschaften exportiert und bei den jeweiligen Beispielberechnungen optional verwendet.

ID	Bezeichner
1	Area
2	Roundness
3	Brightness
4	Curvature/length (only main line)
5	Compactness
6	Max. pixel value Green
7	Max. pixel value NIR
8	Max. pixel value Red
9	Max. pixel value SWIR
10	Mean Green
11	Mean NIR
12	Mean Red
13	Mean SWIR
14	Mean lee_sigma_bright
15	Mean lee_sigma_dark
16	Length of main line (no cycles)
17	Standard deviation Green
18	Standard deviation NIR
19	Standard deviation Red
20	Standard deviation SWIR
21	Standard deviation lee_sigma_bright
22	Standard deviation lee_sigma_dark
23	Average branch length
24	Compactness (polygon)
25	Length/Width
26	Length
27	Rectangular Fit
28	Density
29	Length/Width (only main line)
30	Asymmetry

ID	Bezeichner
31	HSI Transformation Hue(R='SWIR',G='NIR',B='Red')
32	HSI Transformation Intensity(R=SWIR,G=NIR,B=Red)
33	HSI Transformation Saturation(R=SWIR,G=NIR,B=Red)
34	Width
35	GLCM Homogeneity (all dir.)
36	MNDVI
37	MSAVI
38	MSI
39	NDVI
40	Border length
41	GLCM Homogeneity Green (all dir.)
42	GLCM Homogeneity NIR (all dir.)
43	GLCM Homogeneity SWIR (all dir.)
44	Ratio Green
45	Ratio NIR
46	Ratio SWIR

Tabelle A.3: Liste der 46 aus den Testdaten generierten Merkmale

Standardisierte Verteilungen der Merkmalswerte

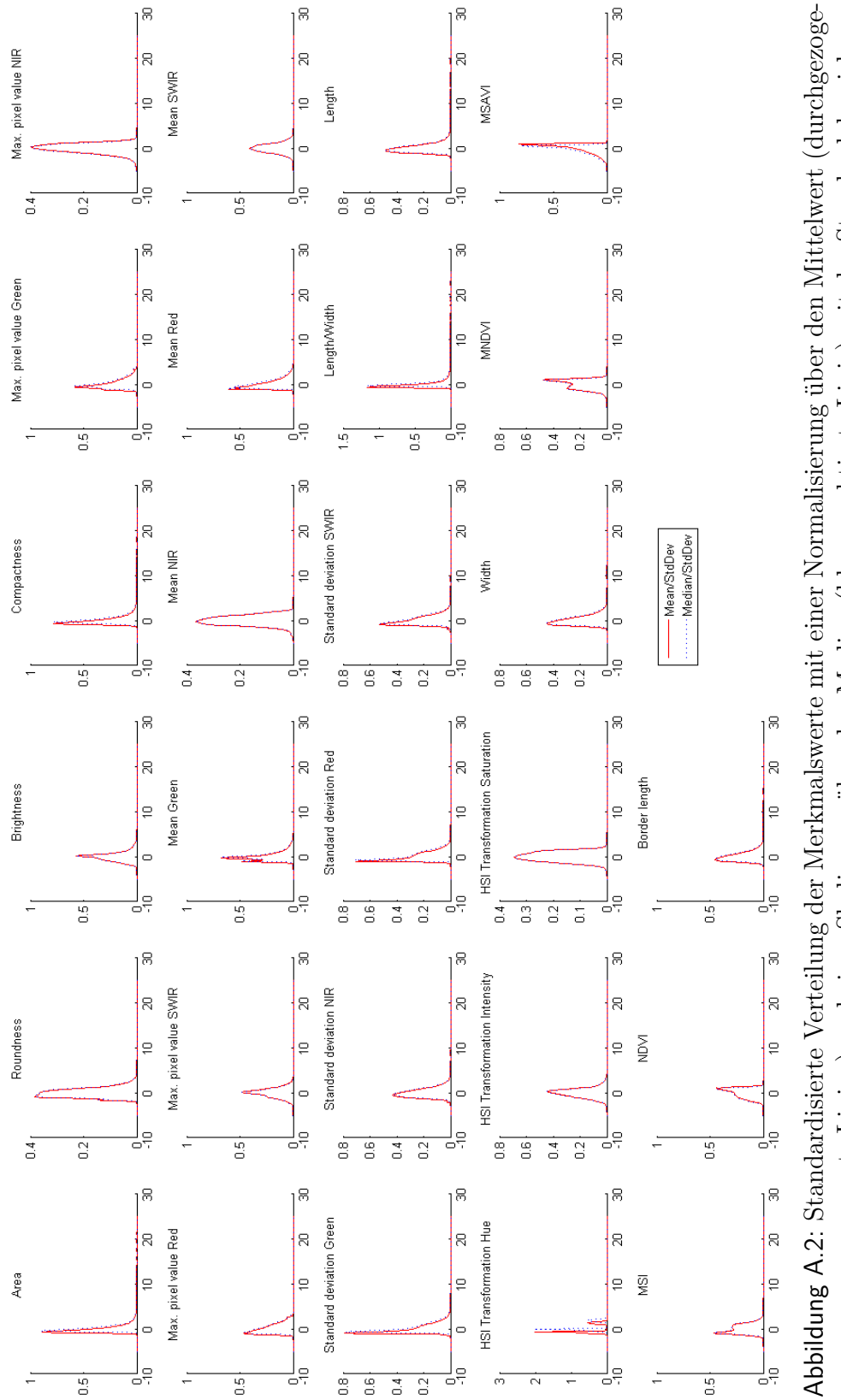


Abbildung A.2: Standardisierte Verteilung der Merkmalswerte mit einer Normalisierung über den Mittelwert (durchgezogene rote Linien) und einer Skalierung über den Median (blau punktierte Linie) mit der Standardabweichung über die Datensätze aller Klassen

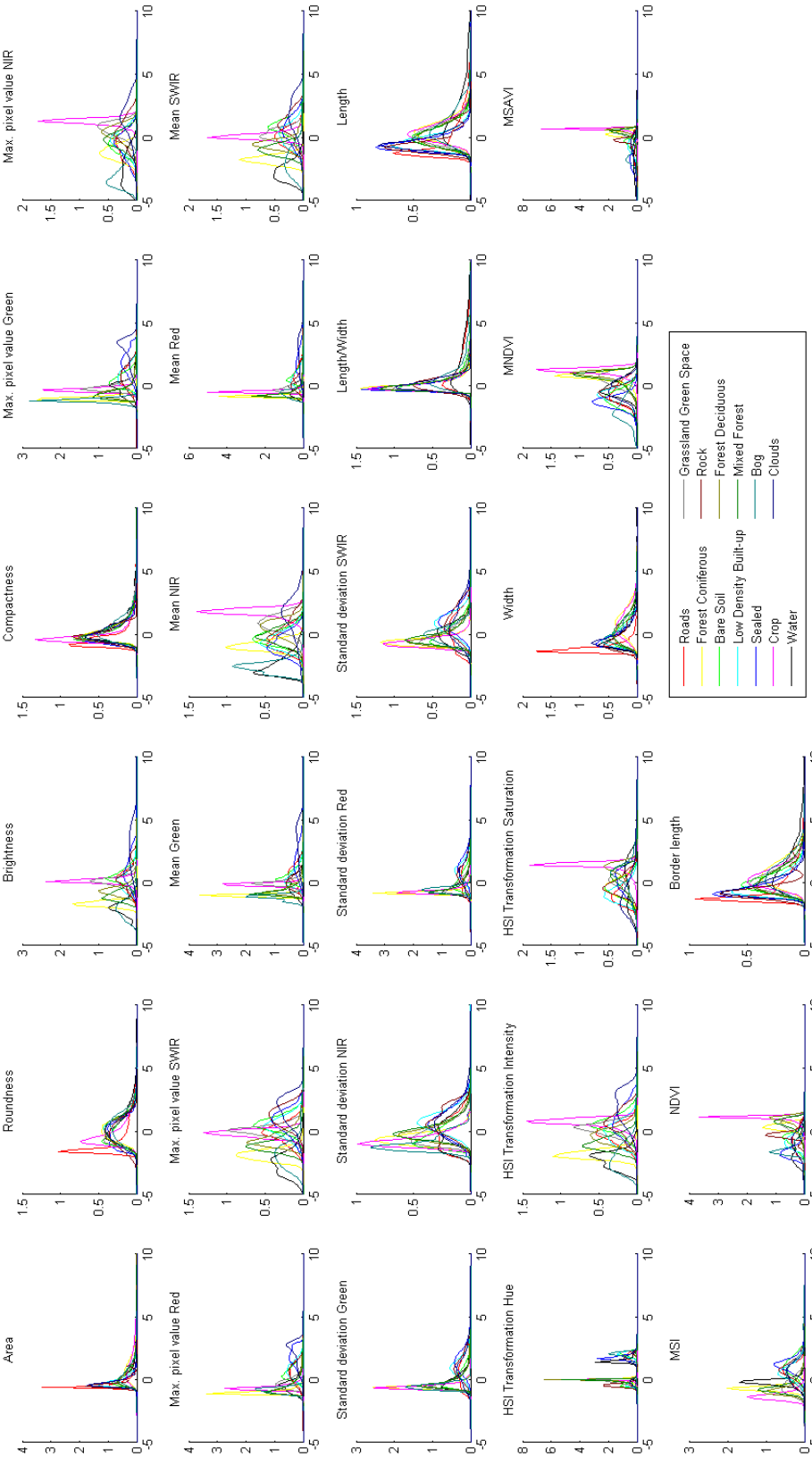


Abbildung A.3: Standardisierte Verteilung der Merkmalswerte, gesplittet nach Klassen mit einer Normalisierung über den Median und die Standardabweichung