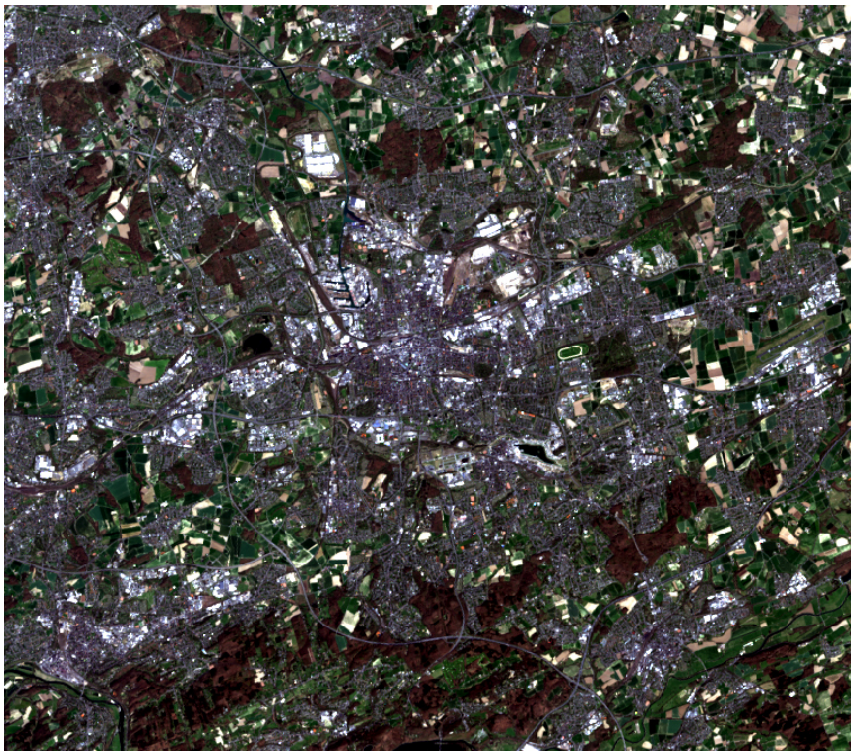
Masterarbeit

Vergleich von verschiedenen Klassifikationsverfahren in der Fernerkundung



08.03.2017

Dozentin:

Prof. Dr. Christine Müller

Zweiter Betreuer:

Prof. Dr. Nguyen Thinh

Autor:

Alexander Marwitz

Inhaltsverzeichnis

1 Einleitung

Die Fernerkundung befasst sich mit der Beobachtung, Kartierung und Interpretation der Erdoberfläche. Im Kontext der Raumplanung wird sie meist zur Erstellung von thematischen Karten aus Satellitenbildern genutzt. Thematische Karten werden in der Raumplanung oft verwendet um Sachverhalte grafisch zu veranschaulichen. Solche Karten befassen sich beispielsweise mit Themen wie dem Wasserhaushalt oder der naturräumlichen Entwicklung eines Gebietes. In vielen Fällen existieren aber keine Karten für das jeweilige Thema beziehungsweise den zu analysierenden Ausschnitt der Welt. Hier können dann eigene Karten zum Beispiel mit Hilfe von Geoinformationssystem-Software wie ArcGIS erstellt werden. Zur Erstellung der Karten bedarf es aber auch hier verschiedener Grundlagenkarten sowie gegebenenfalls weiterer Zusatzinformationen. Zunächst müssen dazu die Grundlagenkarten selbst zur Verfügung stehen. Die Fernerkundung stellt die Mittel zur Erzeugung dieser Karten aus Satellitenbildern bereit. Die Bilder werden von Satelliten erzeugt, die über verschiedene Sensoren die Reflexion elektromagnetischer Strahlung von der Erdoberfläche messen. Diese Strahlung ist nichts anderes als sichtbares beziehungsweise unsichtbares Licht. Das sichtbare Licht teilt sich auf in Rot, Grün und Blau. Unsichtbar (für den Menschen) ist beispielsweise Infrarotlicht. Ein Satellitenbild besteht aus vielen einzelnen Bildpunkten (Pixeln), die jeweils die Messung der Sensoren für einen kleinen quadratischen Teil der Erdoberfläche darstellen. Für diese Art von Rasterdaten ist die Auflösung, also die Angabe wie groß der von einem Pixel dargestellte Erdabschnitt ist, sehr wichtig. Die Auflösung gibt die Detailschärfe an, mit der die Erstellung und die Interpretation der Karten durchgeführt werden können.
Zu Beginn der Kartenerstellung werden in der Regel zunächst einige vorverarbeitende Schritte ausgeführt. Da ein Satellit immer nur einen Ausschnitt der Erde zu einem bestimmten Zeitpunkt erfassen kann, dient die Vorverarbeitung hauptsächlich dazu verschiedene Satellitenbilder zu kombinieren und zu größeren zusammenhängenden Bildern zusammenzufassen. Die Vorverarbeitung ist notwendig, da zum Zeitpunkt der Sensormessung unterschiedliche Voraussetzungen für die jeweiligen Abschnitte der Erde vorliegen. So besitzen Wolken beispielsweise eine großen Einfluss auf die Messwerte. Ebenso wichtig ist der Einstrahlungswinkel der Sonne. Um unerwünschte Einflüsse dieser Art auszugleichen werden radiometrische Korrekturen durchgeführt. Es erfolgen ebenfalls geometrische Korrekturen, um die Rundung der Erdoberfläche

und die Erdrotation auszugleichen. Optional kann das Bild dann noch durch Maßnahmen wie die Erhöhung des Kontrastes weiter verbessert werden. Nach diesen Vorarbeiten erfolgt die Erstellung der Karte. Dazu wird jedem Pixel eine Klasse zugewiesen, die sich aus der Thematik ergibt. Die zu den Pixeln gehörenden Flächen werden dann je nach Klasse in unterschiedlichen Farben dargestellt. Zumeist ist die Klasse eines Areals aber im Vorhinein nicht bekannt und muss erst noch zusätzlich ermittelt werden. Für kleine Karten ist dies gegebenenfalls durch eine Ortsbegehung möglich. Bei größeren Karten wird der zeitliche und finanzielle Aufwand dafür meist zu hoch. Es werden daher Klassifikationsverfahren verwendet, die anhand der Messwerte der Sensoren den Arealen in automatisierter Weise eine Klasse zuordnen. Dies ist die Schnittmenge zur Statistik, die sich ebenfalls verstärkt mit Klassifikationsverfahren befasst. In der Fernerkundung werden typischerweise drei Verfahren verwendet, während in der Statistik zahlreiche andere Verfahren genutzt werden. Aus den Statistikverfahren werden die drei (nach subjektiver Einschätzung des Autors) am häufigsten verwendeten Verfahren ausgewählt und denen der Fernerkundung gegenüber gestellt. Es erfolgt aber auch ein Vergleich der Verfahren untereinander.
In der Fernerkundung wird häufig nach pixelbasierter und segmentbasierter Klassifikation unterschieden. Hier liegt der Fokus auf der segmentbasierten Klassifikation. Bei dieser werden vor der Zuweisung der Klassen Segmente aus den Pixeln erstellt. Die Segmente werden dabei aus benachbarten Pixeln mit ähnlichen Messwerten zusammengefasst. Anschließend wird jedem Segment, und damit allen darin enthaltenen Pixeln, gleichzeitig eine Klasse zugewiesen. Durch die Bildung der Segmente sollen zusätzliche Informationen zur Struktur der Pixel gewonnen werden. Diese Informationen werden bei der Klassifikation verwendet und führen gegebenenfalls zu besseren Ergebnissen. Zur Segmentbildung gibt es verschiedene Vorgehensweisen. Im Folgenden werden hierfür aus der Statistik bekannte Clusterverfahren verwendet. Die verschiedenen Arten der Segmentierung werden dann ebenfalls miteinander verglichen, wobei auch der spätere Einfluss der Verfahren auf die Klassifikationsergebnisse berücksichtigt wird. Abschließend erfolgt eine pixelbasierte Klassifikation, bei der die Pixel direkt klassifiziert werden. Die Ergebnisse werden mit denen der segmentbasierten Klassifikation verglichen. Alle Verfahren werden zunächst mit festen Parametern ausgeführt. Da der Fokus auf der segmentbasierten Klassifikation liegt, wird für die dort verwendeten Klassifikationsverfahren zusätzlich ein Parametertuning durchgeführt. Die obige Vorgehensweise wird auf ein Satellitenbild von

Dortmund und Umgebung angewandt, anhand dessen Bodenbedeckungsklassen bestimmt werden. Bodenbedeckungsklassen werden beispielsweise zur Erstellung von Bebauungsplänen oder Flächennutzungsplänen verwendet. Das Anliegen dieser Arbeit ist es nun eine möglichst gute Vorgehensweise und konkreter ein gutes Klassifikationsverfahren zu finden, dass sich auch für die Klassifikation weiterer Satellitenbilder gut eignet.

Die Details zum Satellitenbild sowie zu den vorliegenden Informationen zur Bodenbedeckung finden sich in Kapitel 2.1. Die inhaltlichen und statistischen Ziele der Arbeit werden in Kapitel 2.2 detailliert dargestellt und konkretisiert. Die theoretischen Aspekte der Verfahren sowie weitere Erläuterungen theoretischer Natur können in Kapitel 3 nachgeschlagen werden. Kapitel 4 beinhaltet die Auswertung der vorliegenden Daten und ist in drei Unterkapitel gegliedert. Diese befassen sich mit der Aufbereitung der Daten, der Segmentierung und der Klassifikation. In Kapitel 5 erfolgen eine Zusammenfassung und ein Ausblick auf mögliche alternative Auswertungsverfahren, deren Untersuchung anhand der Ergebnisse lohnenswert erscheint.

2 Fragestellungen

2.1 Datengrundlage

Die Daten zu dieser Arbeit wurden von Prof. Dr. Nguyen Thinh beziehungsweise Frau Haniyeh Ebrahimi von der Fakultät Raumplanung der Technischen Universität Dortmund zur Verfügung gestellt. Es handelt sich dabei um ein vorverarbeitetes LANDSAT-8-Satellitenbild im TIF-Format, welches die Stadt Dortmund mit Umgebung am 15.04.2015 zeigt. Das geografische Koordinatensystem "Universal Transverse Mercator"(UTM) mit zugehörigem geodätischem Modell "World Geodectic System 1984"(WGS 1984) sorgt für die eindeutige Identifizierbarkeit der dargestellten Erdoberfläche. Der Bereich des Satellitenbildes hat die Koordinaten 382155 - 405555 in horizontaler und 5697075 - 5717685 in vertikaler Richtung. Das Bild besteht aus 535860 Pixeln, die in 687 Zeilen und 780 Spalten angeordnet sind. Die geometrische Auflösung des Bildes liegt bei 30 Metern pro Pixel. Zu jedem Pixel liegen Werte aus sieben Spektralkanälen vor. Sie geben die Reflexion elektromagnetischer Strahlung von der Erdoberfläche in verschiedenen Wellenlängenbereichen an. Diese Bereiche sind in Tabelle 1 näher beschrieben. Die radiometrische Auflösung je-

Tabelle 1: Erläuterung Spektralkanäle eines LANDSAT-8-Satellitenbildes

Kanalnr.	Name	Beschreibung	Wellenlängenbereich in μm
1	Coastal / Aerosol	dunkelblaues / ultraviolettes Licht	0.435-0.451
2	Blau	(sichtbares) blaues Licht	0.452-0.512
3	Grün	(sichtbares) grünes Licht	0.533-0.590
4	Rot	(sichtbares) rotes Licht	0.636-0.673
5	NIR	nahes Infrarotlicht	0.851-0.879
6	SWIR 1	kurzwelliges Infrarotlicht	1.566–1.651
7	SWIR 2	kurzwelliges Infrarotlicht	2.107–2.294

des Kanals liegt bei 16 bit. Dies entspricht $2^{16} = 65536$ unterschiedlichen messbaren Reflexionsintensitäten. Die Messwerte werden daher im Folgenden als sieben stetige Variablen für die Analyse aufgefasst. Die Informationen zu den Spektralkanälen sowie zur Auflösung können im Landsat 8 (L8) Data User Handbook (2016, S. 9 und S. 46f) eingesehen werden. In Abbildung 1 ist eine Echtfarbendarstellung (Kombination der Kanäle 4, 3 und 2) des Satellitenbildes zu sehen. Diese wurde auch für

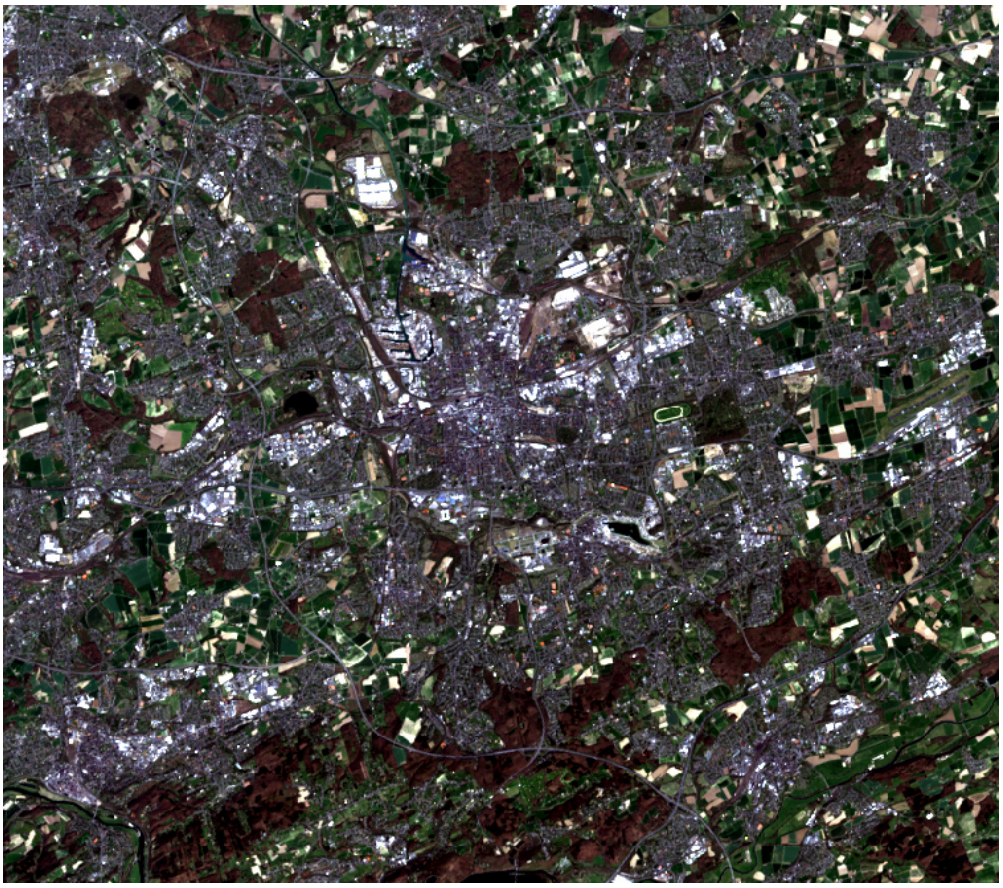


Abbildung 1: Echtfarbendarstellung LANDSAT 8 Satellitenbild Dortmund 15.04.15

das Titelblatt verwendet. Zusätzlich zum Satellitenbild wurden zu jedem Pixel Informationen zur Bodenbedeckung aus CORINE-Daten der aktuellsten Version von 2006 zur Verfügung gestellt. Diese Informationen dienen zur Bewertung während der Analyse und werden als echte Bodenbedeckungsart angenommen. Der zeitliche Unterschied zum Satellitenbild ist zwar relativ groß, sollte aber keine Einschränkung darstellen. Der zeitliche Abstand ist bei allen verwendeten Verfahren identisch, sodass diese vergleichbar sind. Die Bodenbedeckung für Dortmund und Umgebung ist in Abbildung 2 dargestellt. Sie ist in der in Tabelle 2 dargestellten Form codiert. Die Codierung wurde vom Deutschen Zentrum für Luft- und Raumfahrt (DLR) (siehe Quelle [4]) übernommen und beinhaltet ebenfalls einen RGB-Farbcode. Die Farbe der Klassen wird entsprechend diesem Code durch eine Mischung von roten (R), grünen (G) und blauen (B) Farbtönen erzeugt.

In den sieben Spektralkanälen des Satellitenbildes gibt es insgesamt drei fehlende Werte. Bei Kanal 4 handelt es sich um die Pixel 381073 (Zeile 488, Spalte 433) und 497585 (Zeile 637, Spalte 725). Ebenso fehlt der Wert von Pixel 491401 (Zeile 630, Spalte 1) in Kanal 5. Alle fehlenden Angaben werden mit Hilfe der angrenzenden Pi-

Tabelle 2: Codierung Bodenbedeckungsklassen aus CORINE-Daten

Klasse	Codierung CORINE	Bodenbedeckungsart	R/G/B
1	111	Flächen durchgängig städtischer Prägung	211/0/0
2	112	Flächen nicht-durchgängig städtischer Prägung	239/42/71
3	121	Industrie- und Gewerbeflächen	211/0/155
4	122	Straßen und Eisenbahn	126/126/126
5	123	Hafengebiete	155/155/211
6	124	Flughäfen	99/99/99
7	132	Deponien und Abraumhalden	155/13/42
8	141	Städtische Grünflächen	99/239/0
9	142	Sport und Freizeitanlagen	239/71/0
10	211	Nicht bewässertes Ackerland	239/239/126
11	231	Wiesen und Weiden	155/211/13
12	242	Komplexe Parzellenstrukturen	239/211/99
13	243	Landwirtschaft mit natürlicher Bodenbedeckung	183/183/71
14	311	Laubwald	0/183/0
15	312	Nadelwald	0/126/99
16	313	Mischwald	0/126/0
17	321	Natürliches Grasland	155/183/99
18	322	Heiden und Moorheiden	211/211/0
19	512	Wasserflächen	0/155/239

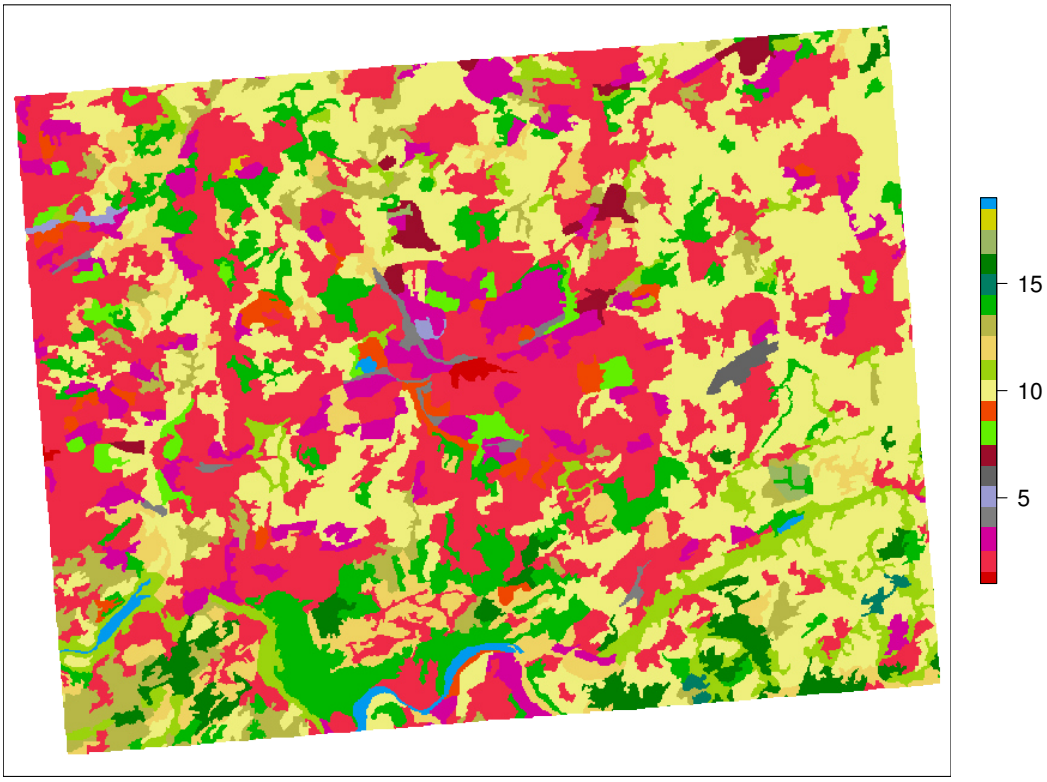


Abbildung 2: CORINE Bodenbedeckung Dortmund 2006

xelwerte des entsprechenden Kanals ersetzt. Dazu wird das arithmetische Mittel der acht (horizontal, vertikal und diagonal) benachbarten Pixel bestimmt. Pixel 491401 liegt am linken Bildrand und hat daher nur fünf benachbarte Pixel aus denen der Mittelwert berechnet wird.

Neben den fehlenden Werten gibt es auch einige unplausible Datenpunkte. Da in den sieben Kanälen der Anteil der reflektierten elektromagnetischen Strahlung in Prozent angegeben wird, sollten diese Werte sich zwischen 0 und 100 % (oder 0 und 1) befinden. Es gibt allerdings insgesamt 168 Werte zwischen 1 und 1.681741. Sie finden sich über alle Kanäle hinweg und werden auf die theoretisch maximale Reflexion von 1 gesetzt.

2.2 Inhaltliche und statistische Ziele

Das Hauptanliegen dieser Arbeit ist der Vergleich von Klassifikationsverfahren. Die Verfahren werden genutzt, um aus einem Satellitenbild eine thematische Karte der Bodenbedeckungen des dargestellten Areals herzustellen. Für die Analyse stehen ein Satellitenbild mit sieben Spektralkanälen und eine Einteilung des Bildes in Bodenbe-

deckungsklassen zur Verfügung. Details zur Datengrundlage können dem vorherigen Kapitel 2.1 entnommen werden. Mit den Spektralkanälen ergibt sich ein Klassifikationsproblem mit sieben Variablen sowie mehreren Bodenbedeckungsklassen, in die Datenpunkte eingeordnet werden sollen. Es wird nach einem möglichst fehlerfreien Verfahren gesucht, welches auch bei weiteren Satellitenbildern gut für die Klassifizierung der Bodenbedeckung verwendet werden kann. In der Fernerkundung wird nach pixelbasierter und segmentbasierter Klassifikation unterschieden. Im Folgenden werden zwar beide Arten verwendet, der Fokus liegt jedoch auf der segmentbasierten Klassifikation. Bei dieser werden vor der Klassifikation die Pixel des Satellitenbildes zu Segmenten zusammengefasst. Segmente sind dabei benachbarte Pixel (Bildpunkte) mit ähnlichen Eigenschaften. Statt der einzelnen Pixel werden in der Folge direkt die Segmente klassifiziert. Die gesamte Analyse erfolgt daher in drei Schritten. Der erste Schritt ist die Bildung der Segmente, welche auch als Segmentierung bezeichnet und auf unterschiedliche Arten vollzogen wird. Im zweiten Schritt werden die Segmente mit verschiedenen Verfahren klassifiziert und die Verfahren bewertet. Im dritten Schritt werden die Pixel direkt klassifiziert (pixelbasiert) und die Ergebnisse mit denen der segmentbasierten Klassifikation verglichen. Jeder dieser Schritte stellt ein eigenes Ziel der Arbeit dar. Im Folgenden werden die einzelnen Ziele im Detail beschrieben und mit der Formulierung von Teilzielen konkretisiert.

Das erste Ziel ist die Segmentierung des Satellitenbildes. Zu diesem Zweck werden Clusterverfahren verwendet, die in der Statistik weit verbreitet sind. Die Clusterverfahren berücksichtigen die Nachbarschaftsbeziehungen der Pixel zueinander nicht. Die Pixel werden daher zunächst nur, wie bei Clusterverfahren üblich, in Gruppen eingeteilt. Insgesamt werden zur Erreichung dieses ersten Teilzieles a) vier unterschiedliche Verfahren verwendet. Zunächst wird der recht populäre k-means-Algorithmus von Hartigan und Wong genutzt. Eine als robuster geltende Alternative ist der clara-Algorithmus. Dieser ist speziell bei größeren Datensätzen wie hier gut anwendbar und ist auch unter dem Namen “k-Median“ zu finden. Ebenfalls sehr beliebt ist das hierarchische agglomerative Clustern, da hier im Vorhinein keine Gruppenzahl festgelegt werden muss. Leider ist es bei großen Datensätzen auf Grund mangelnder Rechnerkapazitäten des ausführenden Computers nicht nutzbar. Daher wird eine eigene Abwandlung des Verfahrens verwendet, die das zusätzliche Attribut “partiell“ im Namen trägt. Als Viertes wird Bagged Clustering verwendet. Hierbei werden der k-means-Algorithmus und das hierarchische agglomerative

Clustern kombiniert um die Vorteile beider Verfahren zu vereinen und die Nachteile auszugleichen. Anhand der Gruppen aus den Clusterverfahren werden dann die Segmentierungen erzeugt, indem horizontal oder vertikal benachbarte Pixel aus der gleichen Gruppe einem Segment zugewiesen werden. Dies entspricht Teilziel b) dieses Abschnitts und wird für alle vier Verfahren durchgeführt. Teilziel c) ist der Vergleich der Clusterverfahren hinsichtlich der aus ihnen resultierenden Segmentierungen. Da die Bodenbedeckungen in Form von Klassen vorliegen, kann mit diesen ebenfalls eine Segmentierung bestimmt werden. Diese kann dann mit den Segmentierungen durch die Clusterverfahren verglichen werden. Es erfolgt jeweils eine visuelle Bewertung und eine objektive Distanzmessung zwischen den Segmentierungen. Da im Folgenden die Segmente klassifiziert werden sollen, werden diese pro Verfahren zu einem Datensatz zusammengefügt (Teilziel d)). Die Segmentinformationen werden dabei aus den Messwerten für die zugehörigen Pixel durch Mittelwertbildung zusammengefasst. Zusätzlich werden weitere Informationen bezüglich der Struktur der Segmente ermittelt und in den Datensatz integriert. Als Zusatzinformationen werden die Höhe und Breite eines Segments in Pixeln verwendet. Hinzu kommt die Anzahl der Pixel des Segments, sowie deren Variabilität. In der blauen Box mit dem Titel “Ziel 1“ sind die oben formulierten (Teil-) Ziele nochmals kurz zusammengefasst.

Ziel 1

Segmentierung des Satellitenbildes mittels Clusterverfahren

a) Gruppierung der Pixel mit Clusterverfahren

- k-means-Algorithmus von Hartigan und Wong
- clara-Algorithmus
- partielles hierarchisches agglomeratives Clustern
- Bagged Clustering

b) Bildung der Segmente anhand der Gruppen

c) Bewertung der Segmentbildung der Verfahren

d) Erstellung neuer Datensätze mit Segmenten und zusätzlichen Informationen über diese

Das zweite Ziel besteht aus der Klassifikation der zuvor erstellten Segmente. Für

die Klassifikation wird das `R`-Paket `mlr` (siehe Bischl et al. (2016)) verwendet. Die Nutzung ist zwar komplizierter als bei anderen Paketen, allerdings wird die Durchführung von Klassifikationsverfahren sehr gut vereinheitlicht und damit auch vereinfacht. Das Paket erfordert allerdings eine Einarbeitung des Klassfikationsproblems. Um das Verständnis für die Vorgehensweise zu erhöhen, wird das Paket auch allgemeiner beschrieben, wobei einige Funktionen im Detail erläutert werden. Diese Vorarbeiten zur Nutzung des Paketes stellen das Teilziel 2a) dar. Die Klassifikation der Segmente erfolgt insgesamt mit sechs verschiedenen Verfahren. Dies sind die lineare Diskriminanzanalyse, der Random Forest, die Support Vector Machine sowie das Minimum-Distance,- Maximum-Likelihood- und Quader-Verfahren. Die ersten drei Verfahren werden typischerweise in der Statistik genutzt, während die restlichen drei Verfahren überwiegend in der Fernerkundung verwendet werden. So erfolgt nicht nur ein Vergleich zwischen den einzelnen Verfahren sondern auch zwischen den Verfahrenstypen. Für den Vergleich wird eine vierfache Kreuzvalidierung bei jedem Verfahren durchgeführt. Dazu werden bei den vier Verfahrensdurchführungen der Kreuzvalidierung immer für einen Teil des Datensatzes Vorhersagen erstellt, während die anderen Teile jeweils zur Erstellung der Klassifikationsregel genutzt werden. Für alle Durchführungen wird eine Fehlklassifikationsrate bestimmt, welche den Anteil der falschen Vorhersagen für die Bodenbedeckungsklassen angibt. Der aus den einzelnen Raten resultierende Mittelwert dient der Bewertung des Verfahrens. Die Kreuzvalidierung wird hier genutzt, da sie eine Überanpassung der Klassifikationsregeln an die Daten verhindert (vergleiche Hsu et al. (2003, S.5)). Die Verfahren sollten dadurch auch bei anderen Satellitenbildern möglichst fehlerfrei nutzbar sein. Die Durchführung der Kreuzvalidierung wird im Folgenden abgeändert, da Segmente klassifiziert werden sollen. Die in der Kreuzvalidierung verwendeten Teile des Datensatzes sollten möglichst gleich groß sein. Die Segmente fassen aber die Informationen aus unterschiedlich großen Mengen von Pixeln zusammen. Ein Viertel der Segmente würde daher unter Umständen viel mehr oder auch weniger als ein Viertel der Informationen des Satellitenbildes enthalten. Dies hätte Einfluss auf die Fehlklassifikationsraten und damit die Bewertung der Verfahren sowie deren Verallgemeinerungsfähigkeit. Um dies zu vermeiden erfolgt eine Teilung der Pixel des Satellitenbildes in (etwa gleich große) Viertel. Dazu wird das Bild horizontal und vertikal halbiert. Die Viertelung findet in einem vorverarbeitenden Schritt, also auch noch vor der Segmentierung statt. Ziel 1 wird insofern abgeändert, dass die Bildung

der Segmente in jedem Viertel einzeln durchgeführt wird. Die Segmente eines Viertels werden dann bei der Kreuzvalidierung als ein Teil der Daten verwendet. Teilziel 2b) ist nun die Durchführung von Kreuzvalidierungen zu den sechs Klassifikationsverfahren bei jedem der vier während der Segmentierung erstellten Datensätze. Zunächst werden für alle Verfahren die Standardparameter der genutzten `R`-Funktionen verwendet. Die Ergebnisse zu den Verfahren werden verglichen und intepretiert. Dann wird versucht die Verfahren zu verbessern, indem die verwendeten Parameter variiert werden. Es wird auch von Parametertuning gesprochen, wobei hier eine Gittersuche nach den besten Parameterwerten ausgeführt wird. Dabei ist allerdings zu berücksichtigen, inwiefern ein Verfahren für das Tuning sinnvolle Parameter besitzt. Sind diese vorhanden, wird für jede Parameterwahl wie bei Teilziel 2b) vorgegangen. Es werden also Kreuzvalidierungen auf allen vier Datensätzen durchgeführt, um die gleichen Vorraussetzungen wie bei den Standardparametern zu gewährleisten. Abschließend wird verglichen inwieweit ein eventuelles Parametertuning die Verfahren verbessern kann. Das Parametertuning und der Vergleich der Verfahren bilden zusammen Teilziel 2c). Dieses sowie die anderen Teilziele sind in der blauen Box “Ziel 2“ in kompakter Form dargestellt.

Ziel 2

Klassifikation des Satellitenbildes inklusive Analyse, Optimierung und Vergleich der Ergebnisse

a) Vorarbeiten zur Nutzung des `R`-Paketes `mlr`

b) Klassifikation mit Kreuzvalidierung und Standardparametern sowie Vergleich der Verfahren

- lineare Diskriminanzanalyse
- Random Forest
- Support Vector Machine
- Minimum-Distance-Verfahren
- Maximum-Likelihood-Verfahren
- Quader-Verfahren

c) Parametertuning (falls möglich) und Vergleich der Verfahren

Als drittes Ziel wird ein Vergleich zwischen pixelbasierter und segmentbasierter Klassifikation angestrebt. Zu diesem Zweck wird zunächst eine pixelbasierte Klassifikation des Satellitenbildes durchgeführt. Die Pixel werden also direkt und lediglich anhand der sieben Spektralkanäle klassifiziert. Es werden die schon zuvor genutzten sechs Klassifikationsverfahren eingesetzt. Um die Vergleichbarkeit zu gewährleisten werden die gleichen (voreingestellten) Verfahrensparameter wie bei der segmentbasierten Klassifikation verwendet. Ebenso wird bei jedem Verfahren eine vierfache Kreuzvalidierung durchgeführt. Die Pixel eines Viertels bilden dabei eines der Datensatzteile. Die Bewertung der Verfahren erfolgt wiederum durch während der Kreuzvaliderung bestimmte mittlere Fehlklassifikationsraten. Diese mittleren Raten werden zum Vergleich der Verfahren untereinander eingesetzt. Zusammen mit der Durchführung der pixelbasierten Klassifikation ist dies Teilziel 3a). Da die Klassifikationen unter gleichen Voraussetzungen stattfinden, können die hier berechneten mittleren Fehlklassifikationsraten nun auch mit denen aus der segmentbasierten Klassifikation verglichen werden. Dieses als 3b) definierte Teilziel befasst sich insbesondere auch mit dem Verhältnis der Verfahren innerhalb der Klassifikationen und der Frage ob und warum sich dieses ändert. Eine Übersicht über Ziel 3 bietet die entsprechende blaue Box.

Ziel 3

Vergleich pixelbasierte und segmentbasierte Klassifikation des Satellitenbildes

a) pixelbasierte Klassifikation mit gleichen Voraussetzungen wie segmentbasierte Klassifikation inklusive Vergleich der Verfahren

b) Vergleich pixelbasierte und segmentbasierte Klassifikation

3 Theorie

3.1 Distanzmaße

Die Mehrheit der Clusterverfahren zur Segmentierung als auch der Klassifikationsverfahren benötigen die Angabe, welches Distanzmaß für die Berechnungen verwendet wird. Im Folgenden wird, sofern nicht explizit anders angegeben, die euklidische Distanz verwendet. Sie wird auch als euklidischer Abstand bezeichnet und findet sich beispielsweise in Deitmar (2017, S. 184). Für zwei Vektoren $x_A = (x_{A1}, \ldots, x_{Av})^T$ und $x_B = (x_{B1}, \ldots, x_{Bv})^T$ ist die euklidische Distanz durch

$$D(x_A, x_B) := \sqrt{(x_{A1} - x_{B1})^2 + \ldots + (x_{Av} - x_{Bv})^2} \tag{1}$$

$$= \sqrt{(x_A - x_B)^T (x_A - x_B)} \tag{2}$$

gegeben. Anstelle von $D(x_A, x_B)$ wird alternativ auch die Bezeichnung $||x_A - x_B||$ verwendet. Oft wird auch von der Distanz zwischen zwei Untersuchungseinheiten A und B mit den Messwerten x_A und x_B gesprochen. Diese Distanz $D(A, B)$ entspricht der Distanz $D(x_A, x_B)$ zwischen den Messwerten. Die euklidische Distanz kann ebenfalls als Multiplikation zweier Vektoren, wie in (2) zu sehen, formuliert werden.
Zur Bewertung der durch die Clusterverfahren entstandenen Segmentierungen wird die NIBS-Distanz verwendet. Sie wird ausführlich in Kapitel 3.2.5 vorgestellt.

3.2 Segmentierung / Clustern

Die Segmentierung der Pixel, also die Bildung von zusammenhängenden Pixelgruppen, findet in dieser Masterarbeit durch Clusterverfahren statt. Daher folgen hier zunächst einige generelle Bemerkungen zu diesem Thema. Zusätzlich werden Vorgehensweisen, die bei allen Verfahren gleich sind und allgemeingültige Notationen eingeführt.
Ein Clusterverfahren trennt die Untersuchungseinheiten (beziehungsweise Objekte oder auch Datenpunkte) $1, \ldots, n$ in Gruppen (beziehungsweise Cluster) $G_1, \ldots, G_k$ auf. Die Anzahl der Objekte in einer Gruppe wird mit $n(G_j)$, $j = 1, \ldots, k$, oder alternativ n_{G_j} angegeben. Die Anzahl der Gruppen k wird je nach Verfahren im Vorhinein festgelegt oder auch nicht. Die Aufspaltung in die Gruppen erfolgt an-

hand der Messwertvektoren $x_1, \ldots, x_n$. Jeder Vektor $x_i = (x_{i1}, \ldots, x_{iv})^T$ ist die Realisation eines Zufallsvektors $X_i = (X_{i1}, \ldots, X_{iv})^T$, $i = 1, \ldots, n$, jeweils bestehend aus v Zufallsvariablen. $X_1, \ldots, X_n$ werden als unabhängig identisch verteilt (u.i.v.) angenommen mit $X_i \sim X, i = 1, \ldots, n$.

Im Gegensatz zur noch folgenden Klassifikation ist zu Beginn eines Clusterverfahrens nichts, eventuell abgesehen von der Anzahl, über die Gruppen bekannt. Die Eigenschaften, die bei allen Objekten einer Gruppe ähnlich sind, werden erst nach der Durchführung des Verfahrens als solche identifiziert und können dann entsprechend interpretiert werden. Dies ist der grundlegende Ansatz zur Unterscheidung zwischen Cluster- und Klassifikationsverfahren. Im Bereich des maschinellen Lernens wird auch von unüberwachtem und überwachtem Lernen gesprochen. Die überwachten Verfahren steuern nur die Einteilung in vorhandene Gruppen, während ein unüberwachtes Verfahren selbstständig Gruppen erzeugt.

Clusterverfahren fassen also ähnliche Objekte anhand ihrer Messwerte zu Gruppen zusammen. Hier sind die Objekte die Pixel eines Satellitenbildes. Die Pixel stehen nicht nur über ihre ähnlichen Messwerte in Verbindung, sondern auch über ihre Position im Bild. Nach der Clusterung werden also zusammenhängende Pixel (waagerecht oder senkrecht benachbart) mit der selben Klasse zu einem Segment zusammengefasst. Auf diese Weise entstehen s Segmente $S_1, \ldots, S_s$.

3.2.1 k-means-Algorithmus (von Hartigan und Wong)

In der Statistik ist ein k-means-Algorithmus eine der Standardvorgehensweisen im Bereich der Clusteranalyse. Hinter diesem Schlagwort stehen allerdings mehrere verschiedene Algorithmen mit der gleichen Grundlage. Generell werden k Zentren (auch Zentroide oder Mittelwerte) bestimmt. Jedes Zentrum soll eine Gruppe möglichst gut repräsentieren. Die Objekte werden dann der Gruppe zugeordnet, zu deren Zentrum sie die kleinste Distanz besitzen. Die Algorithmen unterscheiden sich nun bezüglich des Suchens und Findens der "besten" Zentren. Hier wird lediglich der Hartigan-Wong-Algorithmus beschrieben. Die Informationen hierzu, sowie zu den weiteren Angaben dieses Kapitels stammen aus Hartigan und Wong (1979). Da keine weiteren k-means-Algorithmen verwendet werden, wird der Hartigan-Wong-Algorithmus im Folgenden auch als der k-means-Algorithmus bezeichnet. Für die Berechnungen müssen zunächst k (Start-) Zentren vorgegeben werden. Hartigan und Wong las-

sen dabei offen auf welche Weise dies geschieht. Zunächst wird dann das Zentrum mit der kleinsten ($CEN_1(i)$) und der zweitkleinsten ($CEN_2(i)$) Distanz zu jedem Objekt i, $i = 1, \ldots, n$, bestimmt. Jedes Objekt wird nun vorläufig seinem nächsten Zentrum $CEN_1(i)$ und dem zugehörigen Cluster zugeordnet. Dann werden die Zentren aktualisiert. Dazu wird das arithmetische Mittel aus den Messwerten der dem Zentrum zugeordneten Punkte bestimmt und als neues Zentrum für diese Gruppe von Objekten verwendet. Im Folgenden wird versucht die Zentren weiter zu verbessern. Dazu wird schrittweise immer ein Objekt einer anderen Gruppe zugeordnet und die Zentren anschließend, wie oben beschrieben, aktualisiert. Diese Umordnung kann nun auf zwei verschiedene Weisen erfolgen. Es gibt die schnelle (quick-transfer, QTRAN) und die optimale (optimal-transfer, OPTRA) Variante. Letztere wird zuerst durchgeführt. Die Objekte $1, \ldots, n$ werden der Reihenfolge nach betrachtet. Für ein Objekt i welches sich beispielsweise in Cluster G_1 befindet, werden die in (3)-(4) dargestellten Werte berechnet.

$$Q_1 = \frac{n(G_1) \cdot D(i, G_1)}{n(G_1) - 1} \tag{3}$$

$$Q_2^j = \frac{n(G_j) \cdot D(i, G_j)}{n(G_j) - 1} \quad , \quad j \in J \subseteq \{2, \ldots, k\} \tag{4}$$

Dabei ist $n(G_1)$ die Anzahl der Objekte die aktuell dem Cluster G_1 zugeordnet ist und $D(i, G_1)$ die euklidische Distanz zwischen i und dem Zentroid des Clusters G_1. Analoges gilt für die Cluster G_j, $j \in J$ mit $j \neq 1$. Die Indexmenge J wird dabei über das sogenannte "live set" bestimmt. Zu Beginn der ersten OPTRA-Phase sind alle Cluster im live set. Q_2^j wird also zunächst für alle Cluster $j \neq 1$ berechnet. Dann wird das Minimum $Q_2^{min} = \min_j Q_2^j$ bestimmt. Falls es kleiner als Q_1 ist, wird nun das Zentrum des zu Q_2^{min} gehörigen Clusters G_{min} zu $CEN_1(i)$, also dem nächstgelegenen Clusterzentrum. Im Gegenzug wird das bisher so bezeichnete Zentrum des Clusters G_1 nun zu $CEN_2(i)$. Das Objekt i ist also praktisch von Cluster G_1 zu G_{min} gewechselt beziehungsweise transferiert worden. Im Anschluss werden die Clusterzentren der betroffenen Cluster, wie beschrieben, aktualisiert. Ist Q_2^{min} größer oder gleich Q_1 so geschieht nichts. Es werden nun, wie bereits gesagt, die Objekte $1, \ldots, n$ der Reihenfolge nach durchlaufen. In weiteren OPTRA-Phasen wird zunächst das live set angepasst. Es besteht dann aus den Clustern, die innerhalb der letzten QTRAN-Phase aktualisiert wurden und aus allen Clustern, zwischen denen während der letzten n Schritte einer OPTRA-Phase ein Transfer stattgefunden hat. Das live set wird dabei nach jedem Schritt der OPTRA-Phase aktualisiert. Gehört

ein Objekt zu einem Cluster im live set, so befinden sich in J weiterhin, wie oben, alle anderen Cluster. Ist dies nicht der Fall, reduziert sich die Menge auf alle Cluster die sich im live set befinden. Nach Beendigung eine OPTRA-Phase wird überprüft, ob das live set leer ist. Falls ja, wird der Algorithmus beendet, falls nicht, folgt eine QTRAN-Phase. Diese verläuft ähnlich zu einer OPTRA-Phase. Die Menge J ist allerdings auf das Cluster eingeschränkt, welches dem Objekt am zweitnächsten ($CEN_2(i)$) ist. Das live set hat für die QTRAN-Phase praktisch also keine Bedeutung. Nach der schrittweisen Bearbeitung der n Objekte wird beobachtet, ob mindestens eines davon zwischen zwei Clustern transferiert wurde. Ist dies der Fall, folgt eine weitere QTRAN-Phase. Andernfalls schließt sich eine OPTRA-Phase an. Der k-means-Algorithmus von Hartigan und Wong ist in `R` mit der Funktion `kmeans()` aus dem Paket `stats` (siehe R Core Team (2016)) umgesetzt. Die Startzentren können über den Parameter `centers` bestimmt werden. Es kann auch eine einzelne Zahl, die Anzahl der zu bestimmenden Cluster, angegeben werden. Die Startzentren sind dann k zufällig ausgewählte Objekte.

3.2.2 clara-Algorithmus

Die Bezeichnung clara steht für **c**lustering **lar**ge **a**pplications. Der clara-Algorithmus eignet sich also besonders bei großen Datenmengen. Der Datensatz wird in mehrere Teil-Datensätze aufgeteilt, die einzeln geclustert werden und deren Ergebnisse dann kombiniert werden. Die Clusterung der Teil-Datensätze erfolgt mit dem pam-Algorithmus (**p**artitioning **a**round **m**edoids), weshalb dieser hier zunächst eingehender beschrieben wird. Die verwendeten Informationen sowohl zum pam- als auch zum clara-Algorithmen stammen aus Kaufman und Rousseeuw (2005) und finden sich in den Kapitel 2 (pam) und 3 (clara).

Der pam-Algorithmus bildet k Gruppen um die entsprechende Anzahl Zentren, die sogenannten Medoide, herum. Diese Medoide sollen repräsentative Datenpunkte sein, die die Objekteigenschaften der Gruppe gut wiedergeben. Jedes Objekt wird der Gruppe zugeordnet, zu dessen Medoid die kleinste (euklidische) Distanz besteht. Der Algorithmus wird wegen seiner Ähnlichkeit zum k-means-Algorithmus auch k-medoids (oder k-median) genannt. Er gilt jedoch als robuster, da sich die Zentren, anders als bei k-means, aus der Menge der Objekte ergeben. pam wird in zwei Phasen ausgeführt. In der BUILD-Phase werden schrittweise k Datenpunkte (Objekte)

als (Start-) Medoide ausgewählt. In der SWAP-Phase wird sukzessive ein Medoid gegen einen anderen Datenpunkt ausgetauscht, bis sich das Clusterungsergebnis nicht mehr weiter verbessern lässt.

In der BUILD-Phase wird zunächst das Objekt selektiert, welches die Summe der Distanzen zu allen anderen Objekten minimiert. In jedem weiteren Schritt wird jeweils das Objekt ausgewählt, welches die Summe der Distanzen aller nicht selektierten Objekte zum jeweils nächstgelegenen selektierten Objekt minimiert. Dazu werden die folgenden Schritte so lange wiederholt, bis k Datenpunkte als Medoide ausgewählt wurden. Zunächst wird ein Objekt i, $i \in \{1, \ldots, n\}$, das noch nicht selektiert wurde betrachtet. Die euklidische Distanz $D(j,i) = ||x_j - x_i||$ zu einem anderen nicht selektierten Objekt j, $j \in 1, \ldots, n$, $j \neq i$, wird berechnet. Außerdem wird die kleinste Distanz von j zu einem der bislang ausgewählten Datenpunkt (bezeichnet als D_{j1}) betrachtet. Die Differenz $D_{j1} - D(j,i)$ gibt dann, falls positiv, die Verbesserung durch Selektion von i an. Der Gewinn GN_{ij} durch die Selektion von i bezüglich j wird daher wie in (5) angegeben.

$$GN_{ij} = \max\{D_{j1} - D(j,i), 0\} \tag{5}$$

$$GN_i = \sum_j GN_{ij} \tag{6}$$

Der Gesamtgewinn einer Selektion von i ist entsprechend in (6) zu sehen. Es wird nun das Objekt i selektiert, welches den Gesamtgewinn maximiert. Nachdem k Datenpunkte als Startmedoide ausgesucht wurden, endet die BUILD-Phase.

Es folgt die SWAP-Phase in der versucht wird, die aus den aktuellen Medoiden resultierende Clusterung weiter zu verbessern. Dazu sei zunächst angemerkt, dass die Qualität der Clusterung analog zur BUILD-Phase angegeben wird. Es wird also die Summe der Distanzen jedes nicht selektierten Objektes zum nächstgelegenen Medoid berechnet. In jedem Schritt der SWAP-Phase werden nun die Vor- beziehungsweise Nachteile einer Vertauschung der Rollen eines Medoids i und eines nicht selektierten Datenpunktes h bestimmt. Zunächst wird, ähnlich zum Gewinn von oben, ein Verlust GN_{jih} durch die Vertauschung bezüglich eines ebenfalls nicht selektierten Punktes j berechnet. Dazu müssen allerdings verschiedene Fälle bezüglich der Lage der drei Objekte j, i und h unterschieden werden. Die einzelnen Fälle sollen durch Abbildung 3 verdeutlicht werden. Die Lage der Punkte wird dort zur Vereinfachung eindimensional dargestellt. Für den Fall a) in dem sowohl i als auch h weiter von j entfernt sind als der zu j nächstgelegene Medoid M_{j1} ist der Verlust 0. Der Tausch

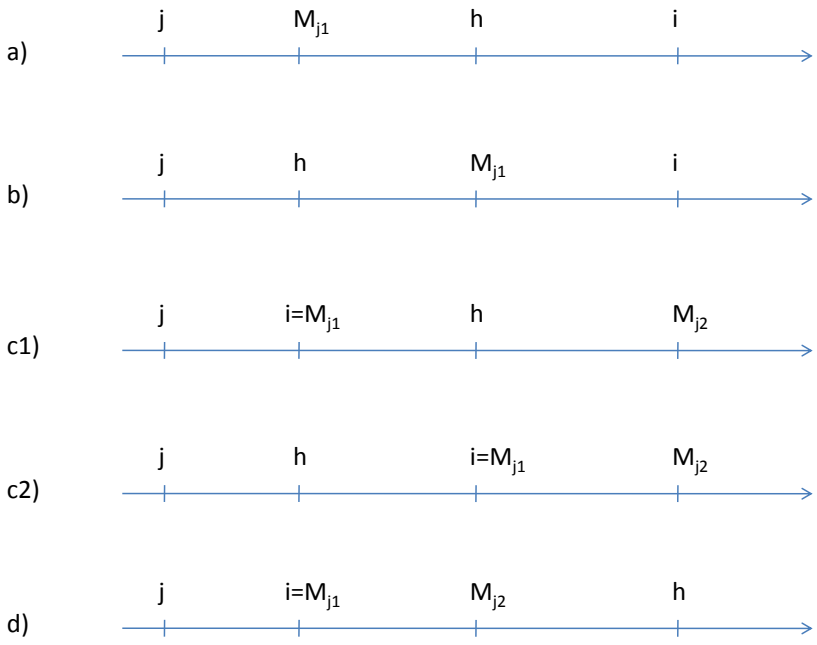


Abbildung 3: Fallunterscheidung bei Berechnung des Verlusts in der SWAP-Phase des clara-Algorithmus (eigene Grafik)

der Rollen von i und j ist für die Distanz zum nächsten Medoid nicht relevant. Im Fall b) liegt h näher an j als der nächstgelegene Medoid M_{j1}, der aber nicht i ist. Hier gilt $GN_{jih} = D(j,h) - D_{j1}$, wobei D_{j1} wiederum die Distanz zum nächsten Medoid ist. Ist i der Medoid mit der kleinsten Distanz zu j (also $i = M_{j1}$), so ist die Lage von h entscheidend. Liegt h näher an j als der zweitnächste Medoid M_{j2}, so ist $GN_{jih} = D(j,h) - D(j,i)$ (Fälle c1) und c2)). Bei c2) wird der Verlust negativ, stellt also praktisch einen Gewinn dar. Ist die Distanz (D_{j2}) von j zum zweitnächsten Medoid hingegen kleiner als die von j zu h (siehe Fall d)), so gilt $GN_{jih} = D_{j2} - D(j,i)$. Der Gesamtverlust bei einem Tausch von i und h ist in (7) dargestellt und sollte nun über alle möglichen Kombinationen minimiert werden.

$$GN_{ih} = \sum_{j} GN_{jih} \tag{7}$$

Ist das Minimum negativ tauschen die entsprechenden Objekte i und h ihre Rollen und die SWAP-Phase wird wiederholt. Andernfalls stoppt der Algorithmus an dieser Stelle und die Clusterung erfolgt anhand der aktuellen Medoide.

Der clara-Algorithmus zieht nun zufällig Objekte aus dem Datensatz und gruppiert

diese mit dem pam-Algorithmus. Die Anzahl der Objekte ist, wie in Kaufman und Rousseeuw (2005) angegeben, $40 + 2k$ bei k zu bildenden Clustern. Das Ziehen erfolgt dabei ohne Zurücklegen. Dieses Vorgehen wird mehrfach wiederholt, wobei eine Ziehung von den Ergebnissen der vorherigen Clusterungen abhängt. Bei der ersten Ziehung gibt es also keine Abhängigkeit. Bei jeder weiteren Ziehung werden die Medoide der bisher besten Clusterung als erste k Objekte verwendet und die restlichen $40+k$ Objekte "normal" gezogen. Die beste Clusterung bestimmt sich dabei über die kleinste durchschnittliche Distanz der Objekte zu ihren jeweils nächsten Medoiden. Dazu werden die Objekte des kompletten Datensatzes und nicht nur die der Ziehung betrachtet. Nachdem alle Berechnungen durchgeführt wurden, wird die Clusterung anhand der Medoide vorgenommen, die die kleinste Durchschnittsdistanz verursachen.

Eine Umsetzung der Algorithmen in R ist im Paket `cluster` (siehe Maechler et al. (2015)) mit den Funktionen `pam()` und `clara()` enthalten. Die Einstellungen `samples` und `sampsize` ermöglichen dabei die Steuerung des Algorithmus. Die erste Einstellung gibt die Anzahl der Ziehungen aus dem Datensatz an, während die Zweite die Anzahl der Objekte einer Ziehung verändert. Die Standards sind 5 Ziehungen mit jeweils $40 + 2k$ Objekten. Zu beachten ist auch, dass die Einstellung `pamLike=TRUE` gewählt werden sollte, die leider kein Standard ist. Sie garantiert die Nutzung des originalen oben beschriebenen pam-Algorithmus bei jeder Ziehung.

3.2.3 Hierarchisches agglomeratives Clustern

Generelle Informationen zu hierarchischem agglomerativem Clustern sowie die Angaben zum verwendeten Algorithmus von Ward finden sich in Fahrmeir et al. (1996, S. 457ff).

Das hierarchisch agglomerative Clustern kombiniert schrittweise zwei Gruppen von Objekten zu einer einzigen Gruppe. Zu Beginn bildet jedes Objekt seine eigene Gruppe. Dann werden die zwei Objekte die sich am ähnlichsten sind zu einer Gruppe zusammengefasst. Ähnlichkeit kann hierbei durch verschiedene Maße gemessen werden. Im Folgenden sind die ähnlichsten Objekte (beziehungsweise Gruppen) diejenigen mit der kleinsten euklidischen Distanz. Die Distanzen zwischen zwei Gruppen werden nun noch für den nächsten Schritt angepasst. Es werden dabei nur die Distanzen der soeben zusammengestellten Gruppe zu den anderen Gruppen berech-

net, da nur hier eine Veränderung vorliegt. Sind die Gruppen G_l und G_m gerade zur Gruppe G kombiniert worden, so lässt die die Distanz zu einer anderen Gruppe G_a wie in (8) dargestellt bestimmen.

$$\begin{aligned} D(G, G_a) =& \tau_1 D(G_l, G_a) + \tau_2 D(G_m, G_a) \\ &+ \tau_3 D(G_l, G_m) + \tau_4 |D(G_l, G_a) - D(G_m, G_a)| \end{aligned} \tag{8}$$

D steht dabei wiederum für die euklidische Distanz (in Abhängigkeit der beiden Gruppen). Allerdings gelten die Angaben auch allgemein für andere Distanzmaße. Die verschiedenen hierarchischen agglomerativen Algorithmen unterscheiden sich nun lediglich in der Wahl der Parameter $\tau_1, \ldots, \tau_4$. Der im Folgenden verwendete Algorithmus von Ward trifft eine recht komplexe Wahl, indem die Anzahlen der Objekte in den Gruppen in Beziehung zueinander gesetzt werden. Die entsprechende Wahl der Parameter ist in (9) - (12) zu sehen.

$$\tau_1 = \frac{n_{G_l} + n_{G_a}}{n_{G_l} + n_{G_m} + n_{G_a}} \tag{9}$$

$$\tau_2 = \frac{n_{G_m} + n_{G_a}}{n_{G_l} + n_{G_m} + n_{G_a}} \tag{10}$$

$$\tau_3 = -\frac{n_{G_a}}{n_{G_l} + n_{G_m} + n_{G_a}} \tag{11}$$

$$\tau_4 = 0 \tag{12}$$

Dabei stehen n_{G_l}, n_{G_m} und n_{G_a} für die jeweilige Anzahl der Objekte in den Gruppen G_l, G_m und G_a. Die Parameter lassen sich somit als Anteile an der Anzahl aller verwendeten Objekte $(n_{G_l} + n_{G_m} + n_{G_a})$ interpretieren. τ_1 ist beispielsweise der Anteil der Objekte aus den Gruppen G_l und G_a. Gibt es in diesen Gruppen viele Objekte erhält die Distanz zwischen den Gruppen entsprechend auch eine hohe Bedeutung. Es werden nun so lange Gruppen zusammengefasst, bis sich alle Objekte in einer einzigen Gruppe befinden. Der Anwender kann wählen in wie viele Gruppen / Cluster die Datenpunkte unterteilt werden sollen. Dies kann beispielsweise über grafische Darstellungen des Clusterungsverlaufs erfolgen. Eine andere hier angewandte Wahl der Gruppenanzahl erfolgt über die Betrachtung der jeweiligen kleinsten Distanzen bei jeder Fusion zweier Gruppen. Steigen diese Distanzen zu stark an, deutet dies darauf hin, das sehr unterschiedliche Gruppen zusammengeführt wurden. Folglich sollte eher die Einteilung der Gruppen vor diesem Fusionsschritt verwendet werden. Liegt ein sehr großer Datensatz vor, kann es beim hierarchischen agglomerativen Clustern am Anfang zu Problemen kommen. Vor dem ersten Schritt muss für jedes

Paar von Objekten die Distanz zwischen diesen bekannt sein, damit die Objekte mit der kleinsten Distanz zusammengefasst werden können. Die Distanzen der Paare werden zumeist in einer Distanzmatrix angegeben, bei der die Zeilen und Spalten jeweils für ein Objekt stehen. Da die Distanzen eines Objektes zu sich selbst unerheblich und die restlichen Distanzpaare symmetrisch sind, müssen also mindestens $(\sum_{i=1}^{n} i)$ Werte angegeben werden. Bei der Berechnung am Computer kann dabei die Kapazität des Arbeitsspeichers überschritten werden, sodass das Verfahren nicht angewendet werden kann. Eine Idee zur Lösung dieses Problems ist die gleichmäßige Aufteilung der Daten in mehrere Teildatensätze. Diese werden so groß gewählt, dass der Ward-Algorithmus normal ausgeführt werden kann. Für jeden Teil wird dann die Gruppenanzahl einzeln bestimmt. Dazu wird ein fester Grenzwert vorgegeben. Ist eine minimale Distanz größer oder gleich diesem Grenzwert, werden die vor diesem Schritt bestehenden Gruppen verwendet. Nach dieser Entscheidung wird nun innerhalb jeder Gruppe das arithmetische Mittel über die Messwerte der Objekte gebildet und als neues Objekt verwendet. Aus einem Teildatensatz entstehen also genau so viele neue Objekte wie Gruppen gebildet wurden. Die neuen Objekte jedes Teildatensatzes werden zu einem neuen Gesamtdatensatz kombiniert. Auf diesen wird nun nochmals der Ward-Algorithmus angewendet und erneut anhand des Grenzwertes die Gruppenanzahl bestimmt. Die daraus resultierende Gruppierung der neuen Objekte wird nun weiterverwendet. Die Gruppe eines neuen Objektes wird jedem an dessen Entstehung beteiligtem originalem Objekt zugewiesen. Dadurch entsteht die endgültige Gruppierung, die für weitere Untersuchungsschritte verwendet werden kann. Da die Daten Teil für Teil bearbeitet werden, könnte die Vorgehensweise, welche aus eigenen Überlegungen heraus entstand, als partielles hierarchisches agglomeratives Clustern (kurz: phclust) bezeichnet werden. Einen guten Überblick bietet Abbildung 4 in der diese Vorgehensweise beispielhaft für 15 Objekte dargestellt wird. Der Ward-Algorithmus kann in `R` mit der Funktion `hclust()` angewendet werden. Dazu muss der Parameter `method` mit der Einstellung `"ward.D2"` verwendet werden. Auf der Basis der Funktion `hclust()` wurde für das partielle hierarchische agglomerative Clustern eine eigene Funktion `phclust()` erstellt. Der `R`-Code hierzu befindet sich im Anhang im Kapitel B.1 ab Seite 102.

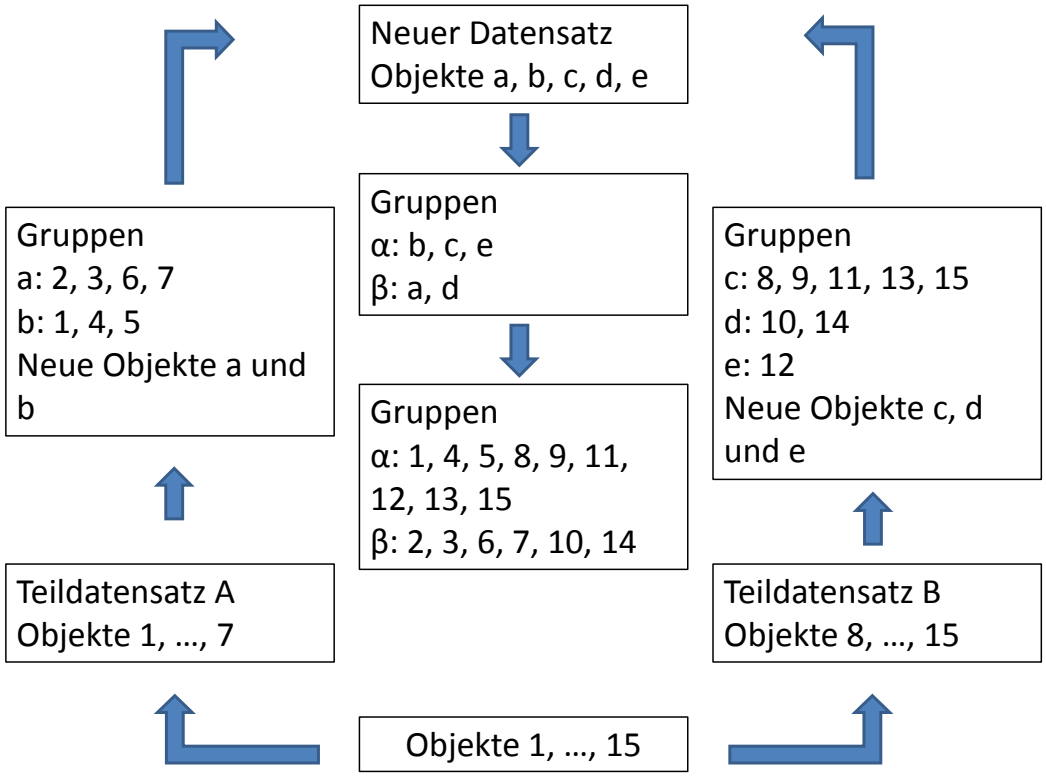


Abbildung 4: Beispiel für die Vorgehensweise bei partiellem hierarchischem agglomerativem Clustern mit 15 Objekten (eigene Grafik)

3.2.4 Bagged Clustering

Das Verfahren Bagged Clustering kombiniert ein partitionierendes Clusterverfahren (wie k-means) mit einem hierarchischen Clusterverfahren. Dabei sollen die Vorteile beider Verfahrenstypen kombiniert und die Nachteile ausgeglichen werden. Eine detailierte Beschreibung des Verfahrens inklusive aller hier verwendeten Informationen kann in Leisch (1999) eingesehen werden.

Zunächst werden aus dem vorhandenen Datensatz ζ Teildatensätze konstruiert, indem jeweils eine Zufallsauswahl mit Zurücklegen ausgeführt wird. Es wird auch von "bootstrap samples" gesprochen. Durch das Zurücklegen kann ein Objekt mehrfach sogar im gleichen Teildatensatz vorkommen. Allerdings kann so angenommen werden, dass die Objekte der Teildatensätze unabhängig voneinander sind. Dies ist eine wichtige Annahme, damit das nun folgenden Bagging, welches namensgebend für dieses Verfahren ist, sinnvoll verwendet wird. Beim Bagging werden die Ergebnisse mehrerer Clusterverfahren, mit unterschiedlichen Teilen der Daten durchgeführt, miteinander kombiniert. Dadurch sollen komplexe Situationen in den Daten besser analysiert werden können. Beim Bagged Clustering wird nun immer das gleiche

Clusterverfahren auf jeden der zufällig konstruierten Teildatensätze angewandt. Im Folgenden wird der in Kapitel 3.2.1 vorgestellte k-means-Algotithmus von Hartigan und Wong verwendet. Aus jedem Teildatensatz ergeben sich k_{base} Zentren. Die insgesamt $\zeta \cdot k_{base}$ Zentren werden nun als neuer Datensatz betrachtet. Mit diesem wird eine hierarchische agglomerative Clusterung, wie in Kapitel 3.2.3 beschrieben, durchgeführt. Dabei wird im Folgenden der Algorithmus von Ward verwendet. Es wird eine im Vorhinein durch den Anwender festgelegte Anzahl von k Clustern bestimmt. Die Objekte $1, \ldots, n$ des ursprünglichen Datensatzes werden nun dem jeweils nächstgelegenen Zentrum $CEN_1(i)$, $i = 1, \ldots, n$, zugeordnet. Als Gruppenzugehörigkeit für Objekt i, $i = 1, \ldots, n$, wird das Cluster G_j, $j \in \{1 \ldots, k\}$, verwendet, zu dem $CEN_1(i)$ gehört.
Bagged Clustering kann in `R` mit der Funktion `bclust()` aus dem Paket `e1071` (siehe Meyer et al. (2015)) durchgeführt werden. Der k-means-Algorithmus von Hartigan und Wong ist bereits voreingestellt. Der Ward-Algorithmus zum hierarchischen agglomerativen Clustern wird mit dem Parameter `hclust.method="ward.D2"` eingestellt. Die Anzahl k der Gruppen kann mit dem Parameter `centers` festgelegt werden. Die Anzahl Zentren pro Teildatensatz wird über den Parameter `base.centers` gesteuert. Zuletzt kann die Zahl der Teildatensätze mit dem Parameter `iter.base` eingestellt werden.

3.2.5 NIBS-Distanz

Zur Bewertung der Segmentierungen durch die Clusterverfahren werden diese mit den "echten" Segmenten aus den CORINE-Daten verglichen. Die NIBS-Distanz (**n**ot **i**nside **b**oth **s**equences) gibt an, wie unterschiedlich zwei Segmentierungen sind. Sie ist aus eigenen Überlegungen auf Basis der Längste-Gemeinsame-Teilsequenz-Distanz (englisch: Longest Common Subsequence distance, kurz: LCS) entstanden. Die LCS-Distanz, zu finden in Navarro (2001, S. 37), ist ein Distanzmaß aus dem Bereich Zeichen- und Worterkennung. Die Distanz zählt, wie viele Operationen mindestens notwendig sind, um eine Sequenz in eine Andere zu überführen. Eine Sequenz ist dabei eine Anordnung von Buchstaben beziehungsweise ein Wort. Es kann jedoch auch eine Reihe von Zahlen (explizit Pixelnummern) verwendet werden. Als Operationen stehen zwei Möglichkeiten zur Auswahl. Zum Einen kann ein Objekt (Buchstabe oder Zahl) an eine beliebige Stelle einer Sequenz eingefügt werden. Zum

Anderen kann ein Objekt aus einer Sequenz entfernt werden. Abweichend von der LCS-Distanz ist die Reihenfolge der Objekte innerhalb einer Sequenz im Folgenden nicht relevant. Die Distanz zwischen zwei Sequenzen entspricht dann der Anzahl an Objekten, die in genau einer der beiden Sequenzen vorkommen. Im Falle zweier gleicher Sequenzen ist die Distanz 0 (untere Schranke). Sind die Sequenzen komplett verschieden ist die Distanz die Summe der Längen beider Sequenzen (obere Schranke). Beim Vergleich der Segmentierungen aus den CORINE-Daten und den Clusterverfahren werden nun alle Sequenzpaare betrachtet, die mindestens ein gemeinsames Objekt besitzen. Gibt es bei der Sequenz aus den CORINE-Daten und der Sequenz aus dem Clusterverfahren mehr als ein gemeinsames Objekt, so geht die Distanz trotzdem nur einmalig ein. Die Distanzen zu den Sequenzpaaren werden nun aufsummiert und ergeben so die NIBS-Distanz. Abbildung 5 soll die obige Vorgehensweise zur Bestimmung der Distanz an einem kleinen Beispiel verdeutlichen. Statt den Segmentierungen aus den CORINE-Daten und den Clusterverfah-

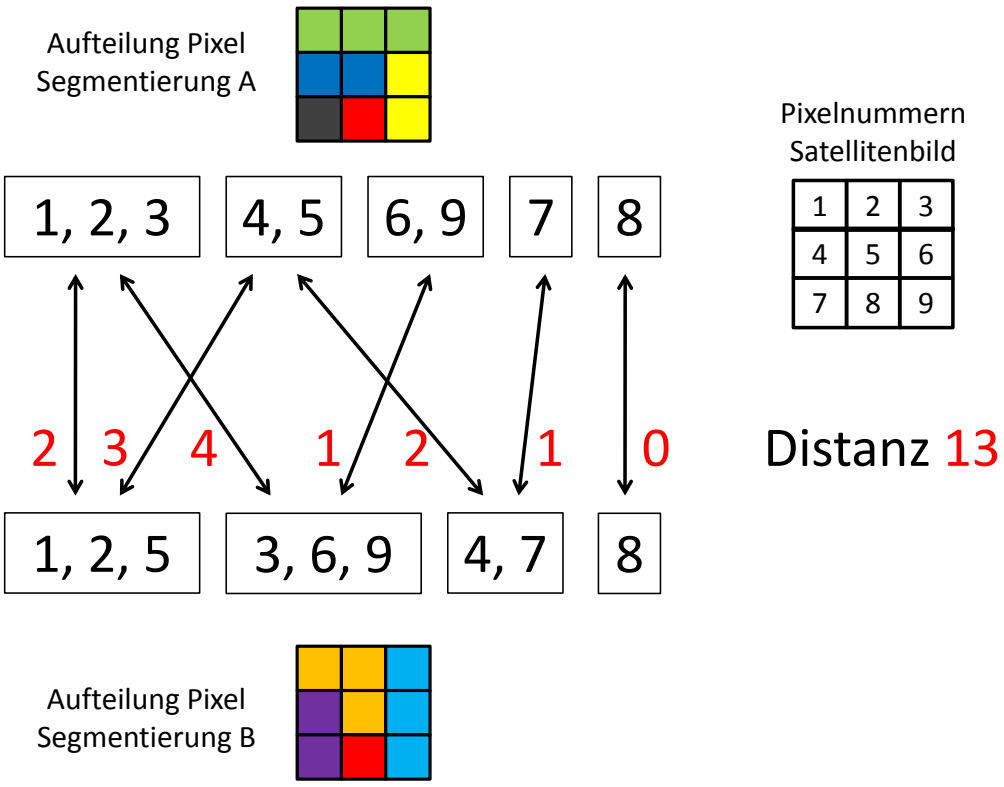


Abbildung 5: Beispiel NIBS-Distanz für 3x3 Satellitenbild (eigene Grafik)

ren können auf die dargestellte Weise auch beliebige andere Mengen von Segmenten verglichen werden. Allgemeiner können auch zwei beliebige Mengen von Sequenzen

betrachtet werden. In den Sequenzen einer Menge sollte jedes Objekt dabei aber nur einmal vorhanden sein. Die Segmentierungen der CORINE-Daten und eines Clusterverfahrens sollten möglichst ähnlich sein. Bei der Bewertung der Clusterverfahren wird also nach einer minimalen NIBS-Distanz gesucht.

In `R` finden sich diverse Funktionen unter dem Begriff Longest Common Subsequence. Ein Teil davon behandelt die Thematik des Longest Common Subsequence Problem, welches nicht mit der Distanz verwechselt werden sollte. Die anderen Funktionen sind für die Konstruktion der NIBS-Distanz nur schwer verwendbar. Darum, aber auch auch rechenzeittechnischen Gründen, wird die eigene Funktion `nibs()` zur Berechnung der NIBS-Distanz genutzt. Sie ist im Anhang in Kapitel B.1 ab Seite 108 zu finden.

3.3 Klassifikation

Wie beim Clustern folgen auch hier zu Beginn einige allgemeine Erläuterungen zu Klassifikationsverfahren. Sie beschreiben den Kontext sowie Vorgehensweisen, die bei allen Verfahren gleich sind. Zusätzlich werden verfahrensübergreifende Notationen eingeführt. Ein Klassifikationsverfahren unterteilt die Untersuchungseinheiten (Objekte / Datenpunkte) $1, \ldots, n$ in die Klassen $C_1, \ldots, C_c$. Die Anzahl der Objekte in Klasse C_j, $j = 1, \ldots, c$, wird mit $n(C_j)$ beziehungsweise n_{C_j} bezeichnet. Die Klassifikation unterscheidet sich nun dadurch vom Clustern, dass die c Klassen schon vor Beginn des Verfahrens festgelegt sind. Objekte sollen lediglich anhand ihrer Messwerte in eine der Klassen einsortiert werden. Die Messwerte der Objekte bei den v Variablen werden dabei wie beim Clustern mit $x_i = (x_{i1}, \ldots, x_{iv})^T$, $i = 1, \ldots, n$, bezeichnet. Sie sind erneut Realisationen der unabhängig identisch verteilten Zufallsvektoren $X_i = (X_{i1}, \ldots, X_{iv})^T$ mit $X_i \sim X$, $i = 1, \ldots, n$. Die v Variablen werden auch als Einflussvariablen oder Features bezeichnet. Bei manchen Verfahren ist es übersichtlicher die Messwerte nach den Variablen getrennt zu verwenden. Hierfür wird die in (13) zu sehende Notation verwendet.

$$\begin{aligned} \tilde{x}_1 &= (x_{11}, \ldots, x_{n1}) \\ \vdots &\qquad\quad \vdots \\ \tilde{x}_v &= (x_{1v}, \ldots, x_{nv}) \end{aligned} \tag{13}$$

Sollen verschiedene Verfahren bezüglich der Richtigkeit ihrer Vorhersagen verglichen werden, muss die Klassenzugehörigkeit aller Objekte bekannt sein. Dabei wird die

Klassenzugehörigkeit durch die diskreten Zufallsvariablen $Y_1, \ldots, Y_n$ mit $Y_i \sim Y$, $i = 1, \ldots, n$, ausgedrückt. Die möglichen Ausprägungen von Y sind die Klassen $C_1, \ldots, C_c$. Die Klassenzugehörigkeit der Objekte $1, \ldots, n$ wird durch den Vektor $y = (y_1, \ldots, y_n)^T$ angegeben, der Realisationen von Y enthält. Eine Vorhersagefunktion für die Zufallsvariable Y wird mit $\hat{Y}$ bezeichnet. Die Vorhersagen für die Klassenzugehörigkeit der Objekte $1, \ldots, n$ werden mit $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)^T$ angegeben. Die Vorhersagen hängen von den Daten ab und werden oft in Form einer Entscheidungsregel $\hat{Y} = \delta(X)$ beziehungsweise $\hat{y}_i = \delta(x_i)$ oder als Funktion $\hat{Y} = f(X)$ beziehungsweise $\hat{y}_i = f(x_i)$, $i = 1, \ldots, n$, verwendet. Die Objekte $1, \ldots, n$ werden nun zum Vergleich der Verfahren in zwei Teile aufgeteilt. Der eine Teil dient dem Verfahren zum Lernen, in welcher Beziehung die Objekte zu den Klassen stehen und welche Eigenschaften diese besitzen. Es wird hier auch vom "Trainieren“ und bei den Objekten dieses Teils der Daten vom Trainingsdatensatz gesprochen. Die tr Messwertvektoren der Trainingsobjekte werden mit $x^{TR} = (x_1^{TR}, \ldots, x_{tr}^{TR})^T$ bezeichnet. Der andere Teil der Daten wird als Testdatensatz bezeichnet. Für die hierin vorhandenen te Objekte und deren Messwertvektoren $x^{TE} = (x_1^{TE}, \ldots, x_{te}^{TE})^T$ wird die Vorhersagequalität “getestet“. Hierzu wird jeweils eine der Klassen vorhergesagt und mit der “echten“ Klasse verglichen. Der Anteil falscher Vorhersagen wird mit der Fehlklassifikationsrate angegeben. Um diese unabhängiger von der Wahl der Trainings- und Testdaten zu machen, kann eine Kreuzvalidierung durchgeführt werden. Diese wird im Kapitel 3.3.1 näher erläutert.

3.3.1 Kreuzvalidierung

Die folgenden Details zur Kreuzvalidierung sind beispielsweise in Witten et al. (2011, S. 152ff) zu finden. Bei einer λ-fachen Kreuzvalidierung werden die Daten in λ möglichst gleich große Teile aufgespalten. Jeder Teil der Daten wird einmal als Testdatensatz beim Klassifikationsverfahren genutzt. Alle anderen Teile der Daten sind zusammen jeweils der Trainingsdatensatz. Es wird also untersucht, wie gut das Verfahren ist, wenn ein Teil der Daten fehlt. Um einen Gesamteindruck zu erhalten, wird für alle λ Durchführungen des Verfahrens die Fehlklassifikationsrate berechnet, welche den Anteil falscher Vorhersagen angibt. Das arithmetische Mittel der Raten heißt dann mittlere Fehlklassifikationsrate (mean missclassification error, kurz mmce). Die Werte der mmce stammen genau wie die Werte der einzelnen Fehl-

klassifikationsraten aus dem Intervall $[0, 1]$. Der beste Wert 0 bedeutet, dass alle Vorhersagen den echten Klassen entsprechen. Beim schlechtesten Wert 1 hingegen sind 100 % (also alle) Vorhersagen falsch.
Die Kreuzvalidierung kann innerhalb des Pakets `mlr` mit der `R`-Funktion `makeResampleDesc()` eingestellt werden. Die Anzahl der Teile in die aufgespalten wird, kann mit dem Parameter `iter` verändert werden. Die gewählte Form der Kreuzvalidierung wird dann bei der Durchführung eines Klassifikationsverfahrens mit angegeben. Die Aufteilung der Daten erfolgt dabei zufällig. Eine größere Kontrolle über die Aufteilung kann durch die Funktionen `makeResampleInstance()` oder `makeFixedHoldoutInstance()` erreicht werden. Bei letztgenannter Funktion muss die Kreuzvalidierung aber manuell ausgeführt werden. Der mmce kann unter Angabe von Vorhersagewerten mit der Funktion `performance()` ermittelt werden.

3.3.2 Lineare Diskriminanzanalyse

Die lineare Diskriminanzanalyse ist ein Verfahren, welches aus der statistischen Entscheidungtheorie heraus als eine spezielle Bayes-Regel entwickelt wurde. Dieser Hintergrund wird hier zunächst näher erläutert, da er zum Verständnis des Verfahrens elementar ist. Die Informationen zur statistischen Entscheidungstheorie sind aus Friedman et al. (2009, S. 18ff) entnommen worden. Details zur anschließenden linearen Diskriminanzanalyse können in Friedman et al. (2009, S. 106ff) nachgelesen werden. Die Entscheidungstheorie soll die Güte einer statistische Entscheidung bewerten. Im Fall der Klassifikation soll die Vorhersage $\hat{Y} = f(X)$ der Klasse mittels X bewertet werden. Die Bewertung erfolgt dabei anhand einer Verlustfunktion L. Hier wird der "0-1-Verlust", wie in (14) dargestellt, verwendet.

$$L(Y, \hat{Y}) = I(Y \neq \hat{Y}) = \begin{cases} 1 & , Y \neq f(X) \\ 0 & , Y = f(X) \end{cases} \tag{14}$$

Die Klassenzugehörigkeit Y eines Objekts ist dabei meist unbekannt. Daher wird statt des Verlusts der erwartete Verlust, auch Risiko genannt, betrachtet. Das Risiko kann, wie in Friedman et al. (2009, S.18) erläutert und in (15) dargestellt, umgeformt werden. Die Umformung geschieht auf Basis der Definition einer bedingten Verteilung (siehe hierfür Czado und Schmidt (2011, S. 20f)).

$$R(f(X)) = E[L(Y, f(X))] = E_X E_{Y|X}[I(Y \neq f(X))] \tag{15}$$

$$= E_X \left[\sum_{j=1}^{c} I(C_j \neq f(X)) \cdot P(C_j|X = x) \right] \tag{16}$$

$$= E_X[1 - P(f(X)|X = x)] \tag{17}$$

Da Y eine diskrete Zufallsvariable mit den Ausprägungen $C_1, \ldots, C_c$ ist, kann der innere Erwartungswert als Summe geschrieben werden (siehe (16)). Anschließend kann die Gegenwahrscheinlichkeit betrachtet werden, da nur eine Indikatorfunktion (nämlich die mit $C_j = f(X)$) zu 0 wird. f wird nun so gewählt, dass das Risiko minimiert wird. Damit ist f eine sogenannte Bayes-Regel. Nach Umformungen wird für eine Beobachtung $\check{x}$ die in (18) zu sehende Entscheidung getroffen.

$$f(\check{x}) = \underset{C \in \{C_1, \ldots, C_c\}}{\operatorname{argmax}} P(Y = C|X = \check{x}) \tag{18}$$

Es wird also die Klasse vorhergesagt, die bei gegebenen Daten die höchste Wahrscheinlichkeit erreicht. Die Wahrscheinlichkeiten werden auch als "a-posteriori" bezeichnet. Sie können über das Bayes Theorem umgeformt werden (siehe (19)).

$$P(Y = C_i|X = x) = \frac{P(Y = C_i) \cdot P(X = x|Y = C_i)}{\sum_{j=1}^{c} P(Y = C_j) \cdot P(X = x|Y = C_j)} \quad , i = 1, \ldots, c \tag{19}$$

Der Nenner ist dabei für alle C_i gleich und spielt daher zur Bestimmung des Maximums keine Rolle. Im Gegensatz zu den a-posteriori-Wahrscheinlichkeiten können die Wahrscheinlichkeiten des Zählers nun relativ einfach geschätzt werden. Um $\pi_{C_i} := P(Y = C_i)$, die "a-priori-Wahrscheinlichkeit" für Klasse C_i, zu schätzen wird zumeist die relative Häufigkeit der Klassen verwendet (siehe (22)). Alternativ kann auch eine Gleichverteilung über alle Klassen angenommen werden.

Die Schätzung von $P(X = x|Y = C_i)$ erfolgt bei vielen Klassifikationsverfahren auf unterschiedlichem Wege und charakterisiert diese dadurch auch. Einige Verfahren treffen Annahmen für die Verteilung von $X|Y = C_i$. Hier wird dann statt $P(X = x|Y = C_i)$ die übliche Notation für eine Dichte $f_{C_i}(x)$ verwendet. Der Index C_i signalisiert hierbei, dass es für jede Klasse C_i, $i = 1, \ldots, c$, solch eine Dichte gibt. Die lineare Diskriminanzanalyse unterstellt hier eine Normalverteilung. Da v Variablen vorliegen ist es eine multivariate Normalverteilung. Wegen der Annahme müssen nur noch Erwartungswerte und Kovarianzmatrizen geschätzt werden. Bei letzteren wird zudem angenommen, dass sie für alle Klassen gleich sind. Die Schätzungen erfolgen daher wie in (20) und (21) zu sehen.

$$\hat{\mu}_{C_i} = \frac{1}{n_{C_i}} \sum_{j:\, y_j = C_i} x_j \quad , i = 1, \ldots, c \tag{20}$$

$$\hat{\Sigma} = \sum_{i=1}^{c} \sum_{j:\, y_j = C_i} (x_j - \hat{\mu}_{C_i})(x_j - \hat{\mu}_{C_i})^T/(n-c) \quad , i = 1, \ldots, c \tag{21}$$

$$\hat{\pi}_{C_i} = n_{C_i}/n \quad , i = 1, \ldots, c \tag{22}$$

Dabei ist n_{C_i} die Anzahl der Objekte in Klasse C_i, $i = 1, \ldots, c$. Für eine Beobachtung $\check{x}$ wird nun für jede Klasse die a-posteriori-Wahrscheinlichkeit, wie in (23) zu sehen, bestimmt.

$$f_{C_i}(\check{x}) := \check{x}^T \hat{\Sigma}^{-1} \hat{\mu}_{C_i} - \frac{1}{2} \hat{\mu}_{C_i}^T \hat{\Sigma}^{-1} \hat{\mu}_{C_i} + \ln \hat{\pi}_{C_i} \quad , i = 1, \ldots, c \tag{23}$$

$$f(\check{x}) = \underset{C \in \{C_1, \ldots, C_c\}}{\operatorname{argmin}} f_C(\check{x}) \tag{24}$$

Die Formel lässt sich dabei herleiten durch das Einsetzen der Dichte einer multivariaten Normalverteilung und dem Vergleich der Dichten zweier Klassen. Der Mehrklassenfall wird dabei aus dem Zweiklassenfall gefolgert. Es wird nun die Klasse mit der höchsten Wahrscheinlichkeit vorhergesagt (siehe (24)). Anschaulich wird durch die oben beschriebene Vorgehensweise der Messraum durch Hyperebenen unterteilt. Diese Hyperebenen sind der Grenzbereich in dem das Maximum der f_{C_i}, $i = 1, \ldots, c$, von zwei (oder mehr) Klassen erzielt wird.
Eine Umsetzung der linearen Diskriminanzanalyse in `R` befindet sich mit der Funktion `lda()` im Paket `MASS` (siehe Venables und Ripley(2002)).

3.3.3 Random Forest

Die nachfolgenden Informationen zum Random Forest stammen aus Friedman et al. (2009, S. 587ff). Der Random Forest, zu deutsch Zufallswald genannt, ist ein Ensembleverfahren. Das bedeutet, dass die Resultate aus vielen Klassifikationsverfahren kombiniert werden, um zu einem besseren Gesamtergebnis zu gelangen. Die zu kombinierenden Verfahren sind in der Regel recht einfach und daher auch schnell zu berechnen, können aber in der Kombination durchaus komplexe Sachverhalte wiedergeben. Die schnellere Berechnung ist hierbei der Hauptvorteil gegenüber aufwändigeren Verfahren wie beispielsweise der Support Vector Machine (siehe Kapitel 3.3.4).
Beim Random Forest werden nun B Entscheidungsbäume (decision trees) miteinander kombiniert. Jeder Baum erstellt für ein Objekt die Vorhersage einer Klasse. Der Random Forest prognostiziert nun die Klasse, die unter allen Bäumen am häufigsten

vorhergesagt wird. Um das Vorgehen zu verdeutlichen, wird im Folgenden der Entscheidungsbaum CART (classification and regression tree) näher erläutert. CART ist der Standard zur Konstruktion eines Random Forest. Die Informationen zu Entscheidungsbäumen und zum CART stammen aus Ripley (1996, S. 213ff).
Der Entscheidungsbaum unterteilt den Messraum $\mathbb{R}^v$ in Regionen genannte Polygone. Dazu werden schrittweise Grenzwerte für die Variablen bestimmt. Ein Grenzwert stellt eine Hyperebene dar, durch die die Objekte zunächst in zwei und dann mehrere Teile getrennt werden. Jede weitere Hyperebene führt zu einer weiteren Unterteilung des Raumes in Regionen. Diese Unterteilung kann auch in einer einem Baum ähnlichen Struktur dargestellt werden. Ein kleines allgemeines Beispiel für 40 Datenpunkte in fünf Dimensionen sowie zwei Klassen (C_1,C_2) ist in Abbildung 6 zu sehen. Am Anfang befinden sich dort alle Objekte in der Wurzel. Die Aufteilung erfolgt an-

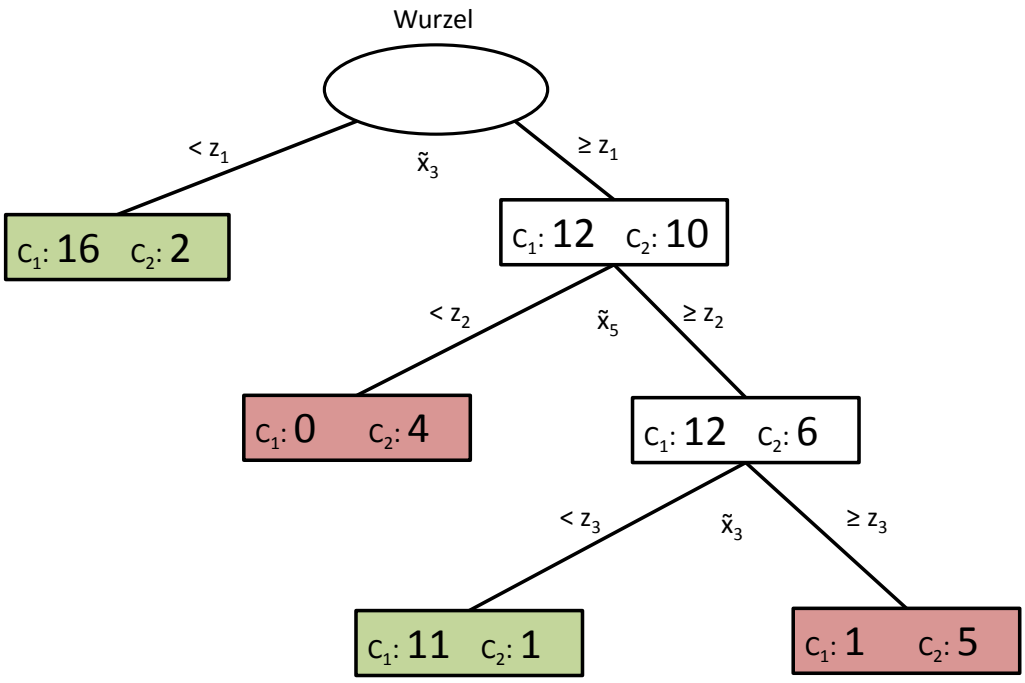


Abbildung 6: Beispiel für die Struktur eines Entscheidungsbaumes (eigene Grafik)

hand des dritten Messwertes jedes Objektes. Diese dritten Messwerte sind im Vektor $\tilde{x}_3 = (x_{13}, \ldots, x_{n3})^T$ zusammengefasst. Die Einträge werden mit dem Grenzwert z_1 verglichen. Dies führt zur Teilung der Daten, je nachdem ob der Messwert kleiner oder größer gleich dem Grenzwert ist. Im Beispiel ist eine Aufteilung in 18 und 22

Objekte zu sehen. Es wird auch davon gesprochen, dass die Wurzel sich verzweigt und zwei Knoten bildet. Der rechte Knoten mit den 22 Beobachtungen verzweigt sich weiter anhand der fünften ($\tilde{x}_5 = (x_{15}, \ldots, x_{n5})^T$) und dann nochmals anhand der dritten Messwerte der Objekte. Die gleiche Variable kann also mehrfach zur Aufteilung verwendet werden. Zu beachten ist, das dabei unterschiedliche Grenzwerte (im Beispiel z_1 und z_3) benutzt werden. Es gibt nun vier Knoten, die sich nicht weiter verzweigen. Dies kann dadurch vorkommen, dass alle Objekte in einem Knoten zur selben Klasse gehören und eine weitere Unterteilung damit nicht sinnvoll wäre. Andererseits kann unter Umständen auch keine weitere Aufteilung der Objekte gefunden werden, die bezüglich deren Klassenzugehörigkeit besser wäre. Zuletzt kann auch im Algorithmus festgelegt werden, dass sich für eine Aufteilung eine Mindestanzahl an Objekten im Knoten befinden muss. In allen Fällen wird von einem Terminalknoten gesprochen. Ein zu klassifizierender Datenpunkt wird von der Wurzel ausgehend bis zu einem Terminalknoten anhand seiner Messwerte einsortiert. Für dieses Objekt wird dann die Klasse vorhergesagt, welche im Terminalknoten am häufigsten vorkommt. Wäre beispielsweise der dritte Messwert des Objekts kleiner als z_1 (linke Verzweigung), so würde Klasse C_1 vorhergesagt. Die Terminalknoten sind in der Grafik der Übersichtlichkeit halber farblich gekennzeichnet. In den grünen Knoten wird Klasse C_1 und in den roten Knoten Klasse C_2 prognostiziert. Alternativ kann die Grafik auch um 180 Grad gedreht und als Stamm mit sich verzweigenden Ästen interpretiert werden.

Die jeweilige Variable und der zugehörige Grenzwert nach denen die Objekte aufgespalten werden bestimmen sich mittels des Gini-Index. Dieser gibt die Unreinheit eines Knotens bezüglich der Klassen wieder und kann für einen Knoten kn wie in (25) zu sehen berechnet werden.

$$Gini(kn) = 1 - \sum_{j=1}^{c} P_{kn}(C_j)^2 \tag{25}$$

Dabei steht $P_{kn}(C_j)$ für die Wahrscheinlichkeit von Klasse C_j in Knoten kn, welche durch die empirische relative Häufigkeit geschätzt wird. Der optimale Wert ist das Minimum bei 0. Es wird erreicht, wenn der Knoten nur Objekte aus einer Klasse enthält. Das Ziel der Objektaufteilung ist nun eine möglichst große Reduzierung des Indexes vom Ausgangsknoten (Mutterknoten) gegenüber den beiden entstehenden neuen Knoten (Tochterknoten). Für einen Mutterknoten m und die Tochterknoten

t_1 und t_2 lässt sich die Reduzierung durch die Differenz in (26) darstellen.

$$\Delta Gini(m, t_1, t_2) = Gini(m) - p_{t_1} \cdot Gini(t_1|Z_{t_1}) - p_{t_2} \cdot Gini(t_2|Z_{t_2}) \qquad (26)$$

Dabei sind p_{t_1} und p_{t_2} die empirischen relativen Häufigkeiten mit denen ein Objekt des Mutterknotens dem jeweiligen Tochterknoten zugeordnet wird. Die Objekte werden mit dem Grenzwert z und der zugehörigen Variablen in die Gruppen Z_{t_1} und Z_{t_2} eingeteilt und an den jeweiligen Tochterknoten weitergegeben. Die Grenzwerte und die Variablen werden so variiert, dass alle möglichen Aufteilungen der Objekte auf die Tochterknoten einmalig betrachtet werden. Es wird die Grenzwert-Variablen-Kombination gewählt, die die Differenz maximiert. Sollte die größte Differenz kleiner gleich Null sein wird der Mutterknoten nicht geteilt.
Ein Entscheidungsbaum sucht sich also immer vorwärts gerichtet die beste Variable mit Grenzwert aus und kann diesen Schritt im Nachhinein nicht mehr korrigieren. Dies ist besonders bei hohen Korrelationen zwischen den Variablen kritisch zu sehen. Diese Schwäche soll durch die Kombination mehrerer Bäume zu einem Wald vermindert werden. Damit die Bäume eines Random Forest jedoch nicht alle gleich sind, werden die Voraussetzungen verändert. Für jeden Baum wird aus den Variablen eine Zufallsauswahl getroffen. Die übrigen Variablen stehen dann für die Konstruktion des Baumes nicht zur Verfügung. Ebenso wird aus den Trainingsdaten eine Zufallsauswahl mit Zurücklegen gezogen, die dann für die Konstruktion eines Baumes verwendet wird. Dadurch werden einige Objekte des Trainingsdatensatzes mehrfach für den gleichen Baum verwendet. Sie erhalten also mehr Einfluss auf die Entscheidung als andere Objekte. Die Zufallsauswahlen werden auch als “bootstrap samples“ bezeichnet. Ein Ensembleverfahren wie der Random Forest ist auch unter dem Oberbegriff “Bagging“ zu finden. Wie bei dieser Art Verfahren üblich weist der Random Forest einem Objekt nun die Klasse zu, die mit den einzelnen Bäumen am häufigsten vorhergesagt wurde. Bei einem Gleichstand kommt es zur zufälligen Vorhersage einer der beteiligten Klassen.
Mit der Funktion `randomForest()` aus dem gleichnamigen Paket (siehe Liaw und Wiener (2002)) kann in `R` standardmäßig ein Wald mit 500 CART-Bäumen erzeugt werden. Die Anzahl der Bäume kann mit dem Parameter `ntree` verändert werden. Die Anzahl der Objekte, die aus den Trainingsdaten gezogen werden, kann mit dem Parameter `sampsize` verändert werden. Die Zahl der zu ziehenden Variablen kann über den Parameter `mtry` eingestellt werden.

3.3.4 Support Vector Machine

Die Support Vector Machine (SVM) wird im Folgenden in mehreren Schritten erläutert. Zunächst wird ähnlich zur linearen Diskriminanzanalyse eine trennende Hyperebene für zwei trennbare Klassen bestimmt. Dann wird auf den Fall zweier nicht trennbarer Klassen verallgemeinert. Zuletzt wird darauf eingegangen, wie mehr als zwei Klassen gehandhabt werden können. Die Informationen zur Support Vector Machine für zwei Klassen stammen aus Friedman et al. (2009). Die Details zu trennbaren Klassen finden sich dabei größtenteils auf den Seiten 129ff, wo es allgemein um (optimal) trennende Hyperebenen geht. Im eigentlichen Kapitel zur SVM ab Seite 417 wird dies kurz aufgegriffen und dann auf die Situation nicht trennbarer Klassen übertragen. Die Angaben zur Berechnung der SVM bei mehr als zwei Klassen (die "1 gegen 1"-Methode) findet sich in Hastie und Tibshirani (1998, S. 451).
Zunächst wird angenommen, dass es zwei Klassen 1 und -1 gibt, die sich linear durch die Hyperebene H voneinander trennen lassen. Es gibt normalerweise mehrere Hyperebenen, die die Klassen trennen. Daher wird nach einer optimalen Ebene gesucht, die den Abstand zu den nächstgelegenen Objekten der Klassen 1 und -1 maximiert. Eine Hyperebene lässt sich durch die in (27) zu sehende Gleichung darstellen.

$$w^T x + b = 0 \tag{27}$$

$$w^T x_{pos} + b = 1 \tag{28}$$

$$w^T x_{neg} + b = -1 \tag{29}$$

Alle Punkte x des Messraums, die die Gleichung erfüllen, liegen auf der Ebene. Dabei ist w ein auf der Hyperebene senkrecht stehender Normalenvektor und $b \in \mathbb{R}$. Sie werden so gewählt, dass für die nächstgelegenen Punkte x_{pos} der Klasse 1 und x_{neg} der Klasse -1 die Gleichungen (28) und (29) gelten. Alle weiteren Punkte der Klassen 1 und -1 ergeben nach Einsetzen in $w^T x + b$ Werte größer als 1 oder kleiner als -1. Daher wird im Folgenden auch von positiven und negativen Klassen, Objekten und so weiter gesprochen. Die nächstgelegenen Objekte der beiden Klassen heißen Stützvektoren (englisch: support vectors). Dies können gegebenenfalls auch mehr als zwei Datenpunkte sein, falls mehrere Punkte einer Klasse der Hyperebene am nächsten sind. Durch die Stützvektoren verlaufen parallel zu H die zwei Hyperebenen H_1 und H_{-1}. Der Abstand zwischen ihnen wird als Margin bezeichnet. Mit

obiger Wahl von w und b lässt er sich berechnen als $2/||w||$. Der Raum zwischen den Ebenen wird oft ebenfalls Margin genannt. Für einen möglichst großen Margin muss nun also $||w||$ minimiert werden. Daraus kann das in (30) - (33) zu sehende Optimierungsproblem formuliert werden.

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 && (30)\\ u.d.N. \quad & w^T x_i + b \geq 1 \quad , y_i = 1 && (31)\\ & w^T x_i + b \leq -1 \quad , y_i = -1 && (32)\\ & bzw. \\ & y_i(w^T x_i + b) - 1 \geq 0 && (33)\\ & \forall i = 1, \ldots, n \end{aligned}$$

Die Nebenbedingungen in (31) und (32) können zu (33) zusammengefasst werden. In ihrer ursprünglichen Form verdeutlichen sie jedoch, dass keine Objekte im Margin liegen dürfen und alle positiven Objekte jenseits von H_1 und alle negativen jenseits von H_{-1} liegen müssen. Das vorliegende konvexe, quadratische Optimierungsproblem (siehe Friedman et al. (2009, S. 132f)) lässt sich mit der Lagrange-Funktion lösen. Informationen zur Lagrange-Funktion sind beispielsweise in Papageorgiou (1996, S. 79ff bzw. S. 97ff) zu finden. Sie hat hier die in (34) angegebene Form, wobei $\alpha = (\alpha_1, \ldots, \alpha_n)^T$ die Lagrange-Multiplikatoren sind.

$$LAG(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \quad (34)$$

Jede Lösung des Optimierungsproblems muss die Karush-Kuhn-Tucker-Bedingungen (35) - (38) erfüllen. Dadurch wird aus dem primalen ein duales Optimierungsproblem.

$$\begin{aligned} \frac{\partial LAG}{\partial w_j} = w_j - \sum_{i=1}^{n} \alpha_i y_i x_{ij} \stackrel{!}{=} 0 \quad , j = 1, \ldots, v \\ \text{bzw. in Vektorform} \quad w = \sum_{i=1}^{n} \alpha_i y_i x_i \quad (35) \\ \frac{\partial LAG}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i \stackrel{!}{=} 0 \quad (36) \\ \alpha_i \geq 0 \quad \forall i = 1, \ldots, n \quad (37) \\ \alpha_i (y_i(w^T x_i + b) - 1) = 0 \quad \forall i = 1, \ldots, n \quad (38) \end{aligned}$$

Werden die Bedingungen in die Lagrange-Funktion eingesetzt vereinfacht sich diese dadurch wie in (39) - (42) zu sehen.

$$LAG(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \tag{39}$$

$$= \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i y_i w^T x_i - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i \tag{40}$$

$$\overset{(36)}{=} \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i y_i w^T x_i + \sum_{i=1}^{n} \alpha_i \tag{41}$$

$$\overset{(35)}{=} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \alpha_i \tag{42}$$

Die Lagrange-Funktion hängt somit nur noch von α ab und hat abschließend die Form (43).

$$LAG(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{43}$$

Aus (38) lässt sich erkennen, dass $\alpha_i = 0$ gelten muss, falls $y_i(w^T x_i + b) > 1$ ist. Andererseits ist $\alpha_i > 0$, falls $y_i(w^T x_i + b) = 1$ gilt. Somit ist klar, dass alle Objekte i, für die $\alpha_i > 0$ gilt, auf den Hyperebenen H_1 oder H_{-1} liegen und damit Stützvektoren sind. Die Vorhersageregel für einen Punkt $\check{x}$ lautet nun

$$f(\check{x}) = sign(\hat{w}^T \check{x} + \hat{b}) = \begin{cases} 1 & , \hat{w}^T \check{x} + \hat{b} \geq 0 \\ -1 & , \hat{w}^T \check{x} + \hat{b} < 0 \end{cases} \tag{44}$$

mit den aus den Gleichungen (28) und (29) bestimmten Schätzern $\hat{w}$ und $\hat{b}$. Für $\hat{w}$ gilt (35), sodass die Regel durch Einsetzen die Form (45) hat.

$$f(\check{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i (x_i^T \check{x}) + \hat{b}) \tag{45}$$

Es ist erkennbar, dass $\hat{w}$ aus einer Linearkombination der Stützvektoren ($\alpha_i \neq 0$) besteht, woraus sich deren Name herleiten lässt.

In der Praxis sind zwei Klassen meist nicht linear trennbar. Daher wird das Optimierungsproblem nun für diesen Fall erweitert. Einigen Datenpunkten wird nun erlaubt, sich innerhalb des Margin (also zwischen H_1 und H_{-1}) zu befinden. Sie dürfen sogar auf der "falschen" Seite der trennenden Hyperebene H liegen und können in diesem Fall sogar außerhalb des Margin sein. Zu diesem Zweck werden Schlupfvariablen (englisch: slack variables) ξ_i, $i = 1, \ldots, n$ eingeführt. Sie geben an, wie weit der Datenpunkt in den Margin beziehungsweise über H hinaus reicht.

Für alle Datenpunkte, die auf der richtigen Seite außerhalb des Margin liegen hat die Schlupfvariable den Wert 0. Die Nebenbedingungen des Optimierungsproblems werden wie in (47) zu sehen angepasst.

$$\min \quad \frac{1}{2}||w||^2 + C_{SVM} \sum_{i=1}^{n} \xi_i \tag{46}$$

$$u.d.N. \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n \tag{47}$$

$$\xi_i \geq 0 \quad \forall i = 1, \ldots, n \tag{48}$$

Natürlich sollten nicht zu viele Variablen zu weit auf der falschen Seite liegen. Daher wird der zu minimierende Term wie in (46) dargestellt ergänzt. Der Parameter $C_{SVM} > 0$ wird dabei auch als Kosten bezeichnet und beschränkt die Größe der Schlupfvariablen (in der Summe) nach oben. Er wird durch den Anwender gewählt und kann durch Tunen zu unterschiedlichen Klassifikationsergebnissen der Support Vector Machine führen. Das obige Optimierungsproblem kann nun wie zuvor mit der Lagrange-Funktion in (49) gelöst werden.

$$\begin{aligned} LAG(w, b, \xi, \alpha, \beta) =& \frac{1}{2}||w||^2 + C_{SVM} \sum_{i=1}^{n} \xi_i \\ & - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - (1 - \xi_i)] - \sum_{i=1}^{n} \beta_i \xi_i \end{aligned} \tag{49}$$

$\alpha = (\alpha_1, \ldots, \alpha_n)^T$ und $\beta = (\beta_1, \ldots, \beta_n)^T$ sind dabei die Lagrange-Multiplikatoren. Durch Einsetzen der Karush-Kuhn-Tucker-Bedingungen (50) - (55)

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{50}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{51}$$

$$\frac{\partial LAG}{\partial \xi_i} = C_{SVM} - \alpha_i - \beta_i \stackrel{!}{=} 0 \quad , i = 1, \ldots, n \tag{52}$$

$$\alpha_i(y_i(w^T x_i + b) - (1 - \xi_i)) = 0 \quad , i = 1, \ldots, n \tag{53}$$

$$\beta_i \xi_i = 0 \quad , i = 1, \ldots, n \tag{54}$$

$$y_i(w^T x_i + b) - (1 - \xi_i) \geq 0 \quad , i = 1, \ldots, n \tag{55}$$

kann die Lagrange-Funktion erneut vereinfacht werden. Die Schritte sind im Einzelnen in (56) - (61) dargestellt.

$$LAG(w, b, \xi, \alpha, \beta) = \frac{1}{2}||w||^2 + C_{SVM} \sum_{i=1}^{n} \xi_i$$

$$- \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - (1 - \xi_i)] - \sum_{i=1}^{n} \beta_i \xi_i \tag{56}$$

$$\overset{(54)}{=} \frac{1}{2} w^T w + C_{SVM} \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - (1 - \xi_i)] \tag{57}$$

$$= \frac{1}{2} w^T w + C_{SVM} \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i y_i w^T x_i - b \sum_{i=1}^{n} \alpha_i y_i$$

$$+ \sum_{i=1}^{n} \alpha_i (1 - \xi_i) \tag{58}$$

$$\overset{(51)}{=} \frac{1}{2} w^T w + C_{SVM} \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i y_i w^T x_i + \sum_{i=1}^{n} \alpha_i (1 - \xi_i) \tag{59}$$

$$\overset{(52)}{=} \frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i \xi_i + \sum_{i=1}^{n} \beta_i \xi_i - \sum_{i=1}^{n} \alpha_i y_i w^T x_i$$

$$+ \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \xi_i \tag{60}$$

$$\overset{(54)}{=} \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i y_i w^T x_i + \sum_{i=1}^{n} \alpha_i \tag{61}$$

(61) und (41) sind nun gleich, sodass die Lagrange-Funktion hier letztlich die gleiche Form wie in (43) annimmt. Der Unterschied zur Situation ohne Schlupfvariablen ist, das hier $0 \leq \alpha_i \leq C_{SVM}$, $i = 1, \ldots, n$, gilt. Die Lagrange-Multiplikatoren sind also zusätzlich nach oben beschränkt durch die Kosten C_{SVM}. Die Entscheidungsregel hat ebenfalls das gleiche Aussehen wie zuvor (siehe (45)). Allerdings unterscheiden sich auch hier die α_i von denen in der Situation trennbarer Klassen. Zusätzlich zu den Objekten auf H_1 oder H_{-1} ist auch bei den Datenpunkten mit positivem ξ_i der Wert des zugehörigen α_i größer als Null. Die Datenpunkte im Margin beziehungsweise auf der falschen Seite sind also zusätzliche Stützvektoren.

Neben den Schlupfvariablen gibt es für nicht trennbare Klassen auch den Ansatz der Kernfunktionen. Bei genauer Betrachtung der Lagrange-Funktion (43) sowie der Entscheidungsregel (45) ist erkennbar, dass die Daten jeweils über ein Skalarprodukt ($x_i^T x_j$ sowie $x_i^T \check{x}$) einfließen. Die Idee der Kernfunktionen ist nun die Verwendung einer Funktion $\phi(x)$ statt der Daten selbst. Dadurch sollen die linear nicht trennbaren Daten zu linear trennbaren Daten transformiert werden. Es ergeben sich die Lagrangefunktion in (62) und für einen Punkt $\check{x}$ die Entscheidungsregel in (63).

$$LAG(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j < \phi(x_i), \phi(x_j) > \tag{62}$$

$$f(\check{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i < \phi(x_i), \phi(\check{x}) > + \hat{b}) \tag{63}$$

Dabei steht $< ., . >$ für das Skalarprodukt im transformierten Messraum. Von Interesse ist nun nicht die Transformation selbst, sondern die zugehörige Kernfunktion (englisch: kernel function). Diese ist für zwei Punkte x_A und x_B in (64) angegeben.

$$\kappa(x_A, x_B) =< \phi(x_A), \phi(x_B) > \tag{64}$$

Dabei muss κ eine symmetrische positiv semi-definite Funktion sein. Die Entscheidungsregel ist dann

$$f(\check{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i \kappa(x_i, \check{x}) + \hat{b}) \quad , \text{mit } 0 \leq \alpha_i \leq C_{SVM} \quad , i = 1, \ldots, n, \tag{65}$$

und der vom Anwender gewählten Kernfunktion κ. Häufig genutzte Kernfunktionen sind ein Polynom r-ten Grades oder eine Radial-Basis-Funktion (siehe (66) beziehungsweise (67)).

$$\kappa(x_A, x_B) = (1+ < x_A, x_B >)^r \tag{66}$$

$$\kappa(x_A, x_B) = \exp(-\gamma ||x_A - x_B||^2) \tag{67}$$

γ ist neben C_{SVM} ein weiterer Tuningparameter, der Einfluss auf das Ergebnis einer SVM mit Radial-Basis-Kern ausübt. Ist der Paramter klein, werden realtiv glatte Grenzen erzeugt, wobei aber die Gefahr von Underfitting besteht. Wird γ hingegen zu groß gewählt könnte es zu einer Überanpassung an die Daten kommen. Der Radial-Basis-Kern wird im Folgenden aber wegen der Flexibilität durch den Parameter nichtsdestotrotz verwendet.

Bisher wurde die SVM nur für zwei Klassen beschrieben. Für die Erweiterung auf ein Mehrklassenproblem gibt es in der Literatur unterschiedliche Ansätze. Hier wird die "1 gegen 1"-Methode verwendet. Dabei wird jeweils eine SVM für zwei Klassen durchgeführt. Bei c Klassen gibt es $c(c-1)/2$ mögliche Paare von Klassen. Es werden also $c(c-1)/2$ Support Vector Maschinen ausgeführt, die ebenso viele Entscheidungen beziehungsweise Vorhersagen treffen. Für einen neuen Datenpunkt wird die Klasse, die dabei am häufigsten auftritt vorhergesagt.

Die `R`-Funktion `svm()` aus dem Paket `e1071` (siehe Meyer et al. (2015)) führt eine Support Vector Machine wie oben beschrieben aus. Dabei werden Schlupfvariablen und die Radial-Basis-Funktion als Kernfunktion verwendet. Die Parameter `cost` und `gamma` entsprechen dabei C_{SVM} und γ von oben und können getuned werden.

3.3.5 Minimum-Distance-Verfahren

Die nachfolgenden Informationen zum Minimum-Distance-Verfahren sind aus Nischwitz et al. (2011, S. 442f) entnommen worden. Das Verfahren funktioniert ähnlich wie ein Clusterverfahren. Für jede Klasse C_i, $i = 1, \ldots, c$ wird ein repräsentativer Klassenmittelpunkt / Zentroid bestimmt. Neue Objekte werden dann der Klasse des nächstgelegenen Zentroids zugeordnet. Die Mittelpunkte sind hier allerdings während des ganzen Verfahrens die gleichen und werden nicht wie beim Clustern aktualisiert. Sie werden aus allen Objekten einer Klasse bestimmt, indem das arithmetische Mittel, wie in (68), gebildet wird.

$$\hat{\mu}_{C_i} = \frac{1}{n_{C_i}} \sum_{j:\, y_j = C_i} x_j \quad , i = 1, \ldots, c \tag{68}$$

Für ein zu klassifizierendes Objekt mit Messwertvektor $\check{x}$ werden dann die Distanzen zu allen Klassenmittelpunkten, wie in (69) zu sehen, bestimmt.

$$D(\check{x}, \hat{\mu}_{C_i}) = ||(\check{x} - \hat{\mu}_{C_i})|| = \sqrt{(\check{x} - \hat{\mu}_{C_i})^T (\check{x} - \hat{\mu}_{C_i})} \quad , i = 1, \ldots, c \tag{69}$$

$$f(\check{x}) = \underset{C \in \{C_1, \ldots, C_c\}}{\operatorname{argmin}} D(\check{x}, \hat{\mu}_C) \tag{70}$$

Nun wird die Klasse mit der kleinsten Distanz vorhergesagt (siehe (70)). Wie angegeben wird standardmäßig die euklidische Distanz verwendet. Andere Distanzmaße kommen kaum zum Einsatz, obwohl ihre Verwendung ebenfalls mögliche wäre. Zu beachten ist ebenfalls, dass die Streuung der Messwerte bei diesem Verfahren nicht berücksichtigt wird.

Eine adäquate `R`-Funktion ist nicht verfügbar, daher wurde jeweils eine eigene Funktion für die Erstellung des Modells sowie für die Vorhersage konstruiert. Der `R`-Code zu den beiden Funktionen `mindist.mod()` und `mindist.predict()` ist im Anhang in Kapitel B.2 ab Seite 112 zu sehen.

3.3.6 Maximum-Likelihood-Verfahren

Die Informationen zu diesem Kapitel stammen aus Nischwitz et al. (2011, S. 448ff). Das Maximum-Likelihood-Verfahren hat Ähnlichkeit mit der linearen Diskriminanzanalyse (LDA). Beide Verfahren leiten sich aus der statistischen Entscheidungstheorie her. Entsprechende Literatur ist im Kapitel 3.3.2 zur LDA genannt worden. Ebenfalls bei beiden Verfahren wird die Annahme getroffen, dass die Objekte einer

Klasse jeweils aus einer multivariaten Normalverteilung stammen. Um die Verteilungen eindeutig angeben zu können, müssen also lediglich die Erwartungswertvektoren und Kovarianzmatrizen für jede Klasse angegeben werden. Im Gegensatz zur LDA wird hier aber angenommen, dass sich die Kovarianzen der einzelnen Klassen unterscheiden. Da die Erwartungswerte und Kovarianzen im Allgemeinen unbekannt sind, werden die in (71) und (72) dargestellten Schätzer verwendet.

$$\hat{\mu}_{C_i} = \frac{1}{n_{C_i}} \sum_{j:\, y_j = C_i} x_j \qquad , i = 1, \ldots, c \quad (71)$$

$$\begin{aligned} \hat{\Sigma}_{C_i} &= \left[\widehat{Cov_{C_i}}(\tilde{x}_{h_1}, \tilde{x}_{h_2})\right]_{h_1,h_2=1,\ldots,v} \\ &= \begin{pmatrix} \widehat{Var_{C_i}}(\tilde{x}_1) & \widehat{Cov_{C_i}}(\tilde{x}_1, \tilde{x}_2) & \cdots & \widehat{Cov_{C_i}}(\tilde{x}_1, \tilde{x}_v) \\ \widehat{Cov_{C_i}}(\tilde{x}_2, \tilde{x}_1) & \widehat{Var_{C_i}}(\tilde{x}_2) & & \widehat{Cov_{C_i}}(\tilde{x}_2, \tilde{x}_v) \\ \vdots & & \ddots & \vdots \\ \widehat{Cov_{C_i}}(\tilde{x}_v, \tilde{x}_1) & \widehat{Cov_{C_i}}(\tilde{x}_v, \tilde{x}_2) & \cdots & \widehat{Var_{C_i}}(\tilde{x}_v) \end{pmatrix} \qquad , i = 1, \ldots, c \quad (72) \end{aligned}$$

n_{C_i} ist erneut die Objektanzahl in Klasse C_i, $i = 1, \ldots, c$. Für die geschätzte Kovarianzmatrix einer Klasse C_i werden empirische Varianzen und Kovarianzen berechnet (siehe (73) sowie (74)), wobei $\widehat{Cov_{C_i}}(\tilde{x}_j, \tilde{x}_j) = \widehat{Var_{C_i}}(\tilde{x}_j)$ für alle $j \in \{1, \ldots, v\}$ gilt. Zu beachten ist hier, dass die Realisationen nach den einzelnen Variablen aufgeteilt werden. Die verwendete Notation ist in der Einführung zu Klassifikationsverfahren (Kapitel 3.3) in (13) dargestellt.

$$\widehat{Var_{C_i}}(\tilde{x}_{h_1}) = \frac{1}{n_{C_i} - 1} \sum_{j:\, y_j = C_i} (x_{jh_1} - \frac{1}{n_{C_i}} \sum_{l:\, y_l = C_i} x_{lh_1})^2$$

$$, h_1 \in \{1, \ldots, v\} \text{ und } i = 1, \ldots, c \quad (73)$$

$$\widehat{Cov_{C_i}}(\tilde{x}_{h_1}, \tilde{x}_{h_2}) = \frac{1}{n_{C_i} - 1} \sum_{j:\, y_j = C_i} (x_{jh_1} - \frac{1}{n_{C_i}} \sum_{l:\, y_l = C_i} x_{lh_1})(x_{jh_2} - \frac{1}{n_{C_i}} \sum_{l:\, y_l = C_i} x_{lh_2})$$

$$, h_1, h_2 \in \{1, \ldots, v\}, h_1 \neq h_2 \text{ und } i = 1, \ldots, c \quad (74)$$

Die Basis für die Darstellung der Kovarianz stammt aus Fahrmeir (1996, S. 50). Auf Grund der Klassen erscheint sie hier in modifizierter Form. Für einen zu klassifizierenden Datenpunkt mit Messwertvektor $\check{x}$ kann nun für jede Klasse $f_{C_i}(\check{x})$, $i = 1, \ldots, c$, bestimmt werden. f_{C_i} ist dabei die Dichte einer multivariaten Normalverteilung mit Erwartungwert $\hat{\mu}_{C_i}$ und Kovarianz $\hat{\Sigma}_{C_i}$. Für die Vorhersage der Klasse von $\check{x}$ werden nun noch a-priori-Wahrscheinlichkeiten benötigt, die durch die relativen Häufigkeiten der Klassen $\hat{\pi}_{C_i} = n_{C_i}/n$, $i = 1, \ldots, c$, geschätzt werden. Die

Vorhersage ist dann durch (75) gegeben.

$$f(\check{x}) = \underset{C \in \{C_1,\ldots,C_c\}}{\operatorname{argmax}} \hat{\pi}_C \cdot f_C(\check{x}) \tag{75}$$

Die a-priori-Wahrscheinlichkeiten können als Gewichtung der Dichten interpretiert werden.

Es konnte keine passende `R`-Funktion gefunden werden. Daher wird das Verfahren durch eigene Funktionen für das Modell und die Vorhersage umgesetzt. Diese sind im Anhang in Kapitel B.2 unter den Namen `maxlike.mod` und `maxlike.predict` ab Seite 115 zu finden.

3.3.7 Quader-Verfahren

Das Verfahren der Klassifikation mit Quadern ist in der Literatur auch unter den Stichworten "parallelepiped classifier" oder "box classifier" zu finden. Die meisten der nachfolgenden Informationen sind aus Nischwitz et al. (2011, S. 453ff) entnommen worden. Sind andere Quellen verwendet worden, werden diese im Zusammenhang genannt.

Beim Quader-Verfahren werden v-dimensionale Quader für den Messraum erstellt. Jede Klasse soll durch einen Quader repräsentiert werden. Die allgemeine Form eines Quaders ist in (76) zu sehen.

$$q_{C_i} = \{(u_1^{C_i}, o_1^{C_i}), (u_2^{C_i}, o_2^{C_i}), \ldots, (u_v^{C_i}, o_v^{C_i})\} \tag{76}$$

Für jede Dimension sind die zwei Begrenzungen des Quaders mit $u_j^{C_i}$ und $o_j^{C_i}$, $j = 1, \ldots, v$, bezeichnet. Diese Begrenzungen können nun beispielsweise wie in (77) und (78) gewählt werden.

$$u_j^{C_i} = \hat{\mu}_j^{C_i} - \eta\sqrt{\widehat{Var_{C_i}}(\tilde{x}_j)} \tag{77}$$

$$o_j^{C_i} = \hat{\mu}_j^{C_i} + \eta\sqrt{\widehat{Var_{C_i}}(\tilde{x}_j)} \tag{78}$$

$$\text{mit} \quad \hat{\mu}_j^{C_i} = \frac{1}{n_{C_i}} \sum_{l:\, y_l = C_i} x_{lj} \tag{79}$$

$$\text{und} \quad \widehat{Var_{C_i}}(\tilde{x}_j) = \frac{1}{n_{C_i}} \sum_{l:\, y_l = C_i} (x_{lj} - \hat{\mu}_j^{C_i})^2 \tag{80}$$

Zumeist wird jedoch die in (81) und (82) dargestellte Wahl getroffen.

$$u_j^{C_i} = \min_{l:\, y_l = C_i} x_{lj} \tag{81}$$

$$o_j^{C_i} = \max_{l:\, y_l = C_i} x_{lj} \tag{82}$$

Sie ist beispielsweise in Lillesand et al. (2008, S.552) zu finden.
Für einen Datenpunkt mit den Messwerten $\check{x} = (\check{x}_1, \ldots, \check{x}_v)$ soll nun eine Vorhersage für die Klassenzugehörigkeit bestimmt werden. Dazu wird zunächst überprüft, in welchen Quadern q_{C_i}, $i = 1, \ldots, c$, der Punkt liegt. Sind die Ungleichungen

$$u_j^{C_i} \leq \check{x}_j \leq o_j^{C_i} \quad , j = 1, \ldots, v \tag{83}$$

erfüllt, befindet sich der Punkt im Quader der Klasse C_i. Für die Vorhersage der Klasse gilt es nun drei Fälle zu unterscheiden. Liegt $\check{x}$ in genau einem der Quader, wird die zugehörige Klasse vorhergesagt. Ist das Objekt in keinem Quader, kann eine eigene Sonderklasse "unknown" zugewiesen werden (siehe Lillesand et al. (2008, S.552)). Dies ist jedoch für die Praxis nicht wünschenswert und würde den Vergleich verschiedener Klassifikationsverfahren erschweren. Daher wird hier zur Erstellung der Vorhersage das Minimum-Distance- oder das Maximum-Likelihood-Verfahren angewandt. Dies geschieht ebenfalls, wenn ein Objekt im Überschneidungsbereich mehrerer Quader liegen sollte. Dann werden für die Verfahren aber nur die zu diesen Quadern gehörenden Klassen berücksichtigt. Das Quader-Verfahren soll gegenüber dem Minimum-Distance- und dem Maximum-Likelihood-Verfahren vor allem Rechenzeitvorteile generieren. Dies wird nur erreicht, wenn sich eine ausreichend große Anzahl von Objekten in nur einem Quader befindet.
In `R` wurde leider keine Funktion gefunden, die das Quader-Verfahren in angemessener Form ausführt. Daher werden die selbst erstellten Funktionen `quader.mod` für das Modell und `quader.predict` für die Vorhersage verwendet. Sie sind im Detail im Anhang in Kapitel B.2 ab Seite 119 zu finden.

4 Auswertung

Alle Berechnungen dieses Kapitels wurden mit R 3.3.2 (siehe R Core Team (2016)) durchgeführt. Zusätzlich benötigte Pakete werden an den entsprechenden Stellen genannt.

4.1 Vorverarbeitung Daten

Vor der Segmentbildung und der Klassifikation müssen die Daten zunächst in die richtige Form für die Weiterverarbeitung gebracht werden. Die Beschreibung hierzu findet sich im Folgenden. Zuerst wird das Satellitenbild von Dortmund eingelesen. Dazu wird die Funktion `brick()` aus dem Paket `raster` (siehe Hijmans (2015)) verwendet. Das Paket beinhaltet eine Vielzahl von Funktionen, die zur Bearbeitung von Bild-Dateien und speziell Satellitenbildern sehr nützlich sind. Die anderen Funktionen dieses Kapitels stammen sofern nicht anders angegeben ebenfalls aus diesem Paket. Die Funktion `brick()` erstellt nun mittels Eingabe des Dateipfades aus einer TIF-Datei ein eigenes Datenformat "(Raster) Brick" in dem die Informationen abgespeichert werden. Die Pixelwerte jedes Kanals des Bildes werden in einem sogenannten Layer abgelegt. Die Pixelwerte liegen dort Bildzeile für Bildzeile in Form eines Vektors vor. Zusätzliche werden weitere Informationen über das Satellitenbild abgespeichert. Zunächst sind dies die Pixelanzahl (535860) und die Zahl der Zeilen (687) und Spalten (780) in die das Bild aufgeteilt ist. Diese Angaben werden öfter benötigt und können mit den Funktionen `ncell()`, `nrow()` und `ncol()` recht einfach ermittelt werden. Zu den Angaben des Layers gehört auch das geografische Referenzkoordinatensystem dem die Daten zugeordnet sind. Es gibt an, welchen Bereich der Erdoberfläche das Bild darstellen soll. Dadurch können verschiedene Satellitenbilder in Beziehung zueinander gesetzt werden. Es können beispielsweise Informationen überlappender Bereiche kombiniert und gegebenenfalls größere Aufnahmen erzeugt werden, als dies technisch möglich wäre. Das Koordinatensystem ist hier "Universal Transverse Mercator" (UTM). Da die Systeme im Laufe der Zeit verbessert wurden, wird zusätzlich auch das geodätische Modell (Kartenbezugssystem) angegeben. Dieses beschreibt die technischen Bedingungen näher und ist hier das "World Geodectic System 1984" (WGS 1984). Der Bereich in dem das Bild sich befindet ist unter dem Begriff `extent` durch die Werte `xmin=382155`, `xmax=405555`,

`ymin=5697075` und `ymax=5717685` angegeben. Ebenfalls im Layer angegeben ist die geometrische Auflösung (`resolution`), die hier 30 Meter pro Pixel beträgt. Zuletzt wird auch der minimale und maximale Pixelwert im Layer angegeben. Alle Angaben des Layers werden auch beim zugehörigen Brick angezeigt. Zusätzlich kommt noch die Anzahl der im Brick vorhandenen Layer hinzu, welche hier zunächst 7 ist. Sie kann mit der Funktion `nlayers()` seperat eingesehen werden.

Nach dem Einlesen der Daten werden diese nun zunächst auf fehlende Werte hin überprüft. Wie in Kapitel 2.1 zur Datengrundlage bereits angegeben, werden die drei fehlenden Werte aus den Layern 4 und 5 durch die arithmetischen Mittel der Nachbarpixel aus dem jeweiligen Layer ersetzt. Dann werden alle unplausiblen Werte über 1 auf diese theoretische Obergrenze gesetzt. Details werden ebenfalls in Kapitel 2.1 angegeben.

Um sich einen ersten Überblick über die Daten zu verschaffen, kann das Satellitenbild nun in unterschiedlichen Grafiken dargestellt werden. Ein Echtfarbenbild, wie Abbildung 1 aus Kapitel 2.1 (Seite 5) kombiniert verschiedene Kanäle (hier Rot, Grün und Blau). Es kann mit der Funktion `plotRGB()` erzeugt werden, indem die Layernummer des roten (4), grünen (3) und blauen (2) Kanals angegeben werden. Zudem werden die Parameter `scale=1` und `stretch="lin"` gesetzt. Sie geben den maximalen Wert eines Pixels an beziehungsweise verbessern den Kontrast des Bildes durch eine lineare Streckung der Pixelwerte. Neben der Kombination mehrerer Kanäle kann jeder Layer auch einzeln in einer Grauwertdarstellung betrachtet werden. Sie stellt den Reflexionsanteil dar, der am Sensor für diesen Kanal angekommen ist. Für die Grafik wird die Funktion `plot()` unter Angabe des Bildes und einer Layernummer verwendet. Ein Beispiel ist in Abbildung 7 für Kanal 5 mit nahem Infrarotlicht (NIR) zu sehen. Die Darstellung der übrigen Kanäle ist in Abbildung 21 im Anhang in Kapitel A auf Seite 95 zu finden.

Zu dem Brick für das Satellitenbild mit den bisher 7 Layern können (beliebig viele) weitere Layer hinzugefügt werden. Dies geschieht mit der Funktion `setValues()`, indem das Satellitenbild, die Layernummer und ein Vektor mit den Werten der Pixel für diesen Layer angegeben werden. Zunächst werden zwei Layer (8 und 9) mit den horizontalen und vertikalen Koordinaten der Pixel erstellt. Diese werden später bei der Kontruktion der Datensätze mit den Segmenten benötigt, um die Breite und Höhe eines Segments einfach berechnen zu können. Die Koordinaten können mittels der Funktion `rasterToPoints()` aus dem Satellitenbild ermittelt werden. Für die

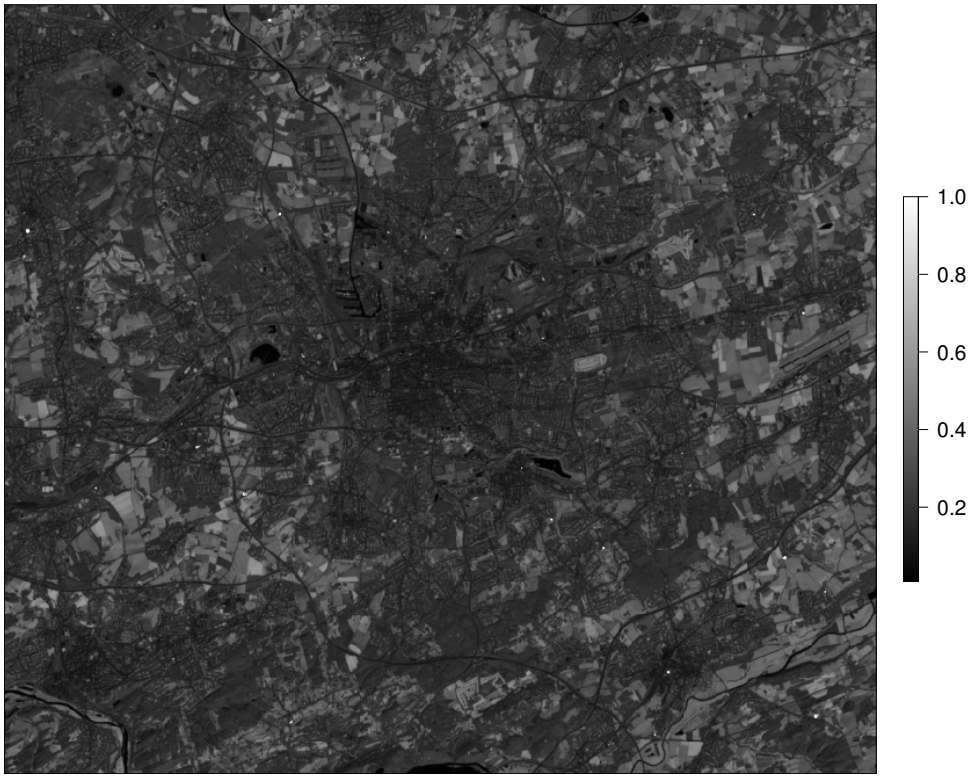


Abbildung 7: Grauwertdarstellung Kanal 5 (NIR)

Klassifikation werden die “echten“ Bodenbedeckungen für jeden Pixel benötigt. Sie werden als zehnter Layer an das Satellitenbild angehängt. Dazu wird die Datei mit den CORINE-Daten auf die gleiche Weise wie das Satellitenbild eingelesen. Es liegt allerdings nur ein Layer vor, welcher eine der 19 Bodenbedeckungsarten angibt. Wie beim Vergleich der Abbildungen 1 und 2 (Kapitel 2.1, S. 5 und 7) zu erahnen ist, stellen die CORINE-Daten einen größeren Bereich um Dortmund herum dar als das Satellitenbild. Die Funktion `crop()` schneidet nun die CORINE-Daten auf den Bereich des Satellitenbildes zu, sodass nur die für das Bild relevanten Informationen übrig bleiben. Dieses Vorgehen funktioniert nur richtig, wenn das gleiche geografische Koordinatensystem und das gleich geodätische Modell vorliegen. Dies ist hier aber gegeben. Die resultierenden 535860 Angaben zur Klasse der Bodenbedeckung werden nun noch in Layer 10 des Satellitenbildes abgelegt. Zu beachten ist, dass Klasse 15 (Nadelwald) nur außerhalb des Bereichs des Satellitenbildes vorkommt. Sie wird durch das Zuschneiden komplett entfernt und hat für die weitere Analyse keine Bedeutung mehr. Neben der Darstellung der Bodenbedeckungen in Abbildung 2 (Kapitel 2.1, S. 7) kann für einen ersten Eindruck auch angegeben werden, wieviele Pixel jeder der 18 vorhandenen Klassen zugeordnet sind. Dies ist in Tabelle 3

Tabelle 3: Anzahl Pixel in Bodenbedeckungsklassen

Klasse	1	2	3	4	5	6	7	8	9
Anzahl Pixel	1465	192792	38955	4856	613	2425	6131	6713	9906
Klasse	10	11	12	13	14	16	17	18	19
Anzahl Pixel	149319	17561	26561	16715	50170	9391	778	401	1108

zu sehen, wobei Klasse 15 schon weggelassen wurde. Besonders viele Pixel gehören zu den Klassen 2 (Flächen nicht-durchgängig städtischer Prägung) und 10 (Nicht bewässertes Ackerland). Dies ist nicht weiter verwunderlich, da nicht nur das Zentrum, sondern auch das Umland von Dortmund auf dem Satellitenbild erfasst ist. Andere Klassen wie beispielsweise Klasse 5 (Hafengebiete) erscheinen auf Grund ihrer sehr speziellen Bodenbedeckung eher selten.

Während der Klassifikation soll eine 4-fache Kreuzvalidierung durchgeführt werden. Daher werden die Daten nun deterministisch in vier Teile aufgeteilt. Die Daten werden dabei geviertelt, wie in Abbildung 8 anhand der roten Linien illustriert. Das

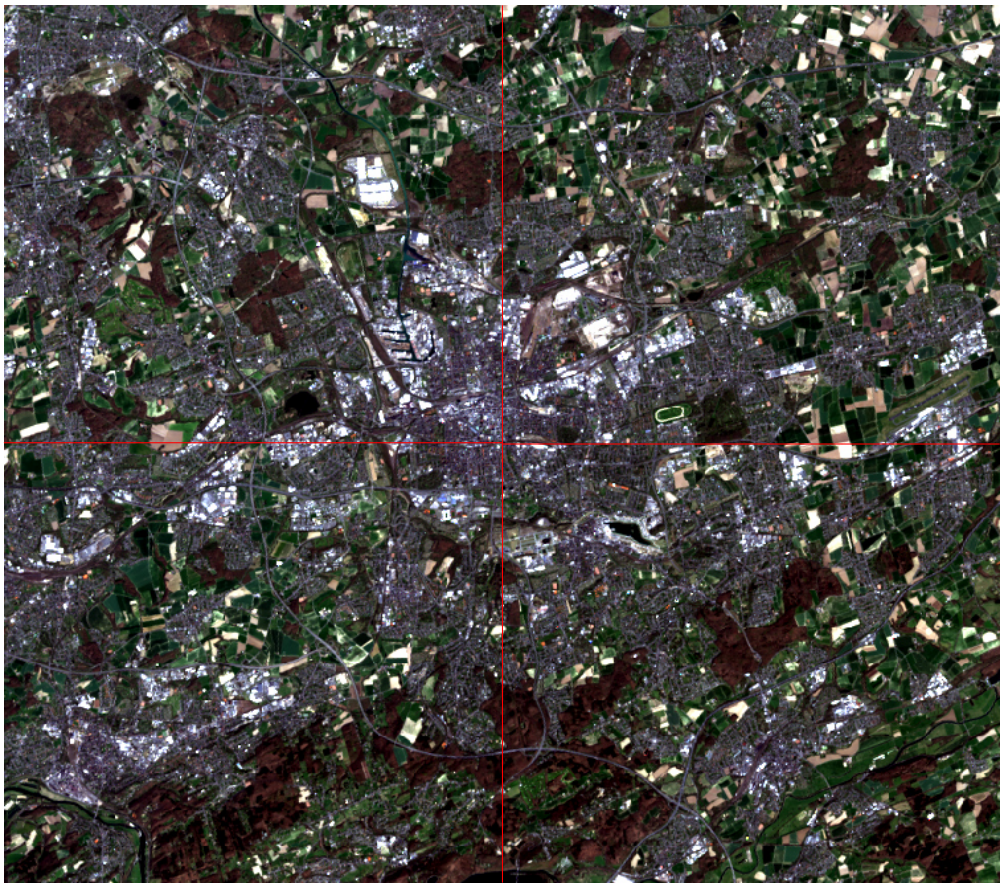


Abbildung 8: Aufteilung des Satellitenbildes in vier Teile

Satellitenbild wird vertikal zwischen den Zeilen 390 und 391 getrennt. Da das Bild 687 Zeilen hat, kann keine Trennung exakt bei der Hälfte erfolgen. Der linke Teil wird zwischen den Zeilen 344 und 345 aufgespalten, der rechte zwischen den Zeilen 343 und 344. Die vier Teile werden in der weiteren Auswertung auch als ol (oben links), or (oben rechts), ul (unten links) und ur (unten rechts) bezeichnet. Alle vier Teile werden jeweils als eigener Brick mit den zugehörigen Daten der 10 Layer abgelegt. Im Folgenden werden für jeden Teil der Daten die Cluster- und Klassifikationverfahren separat ausgeführt und die Ergebnisse zusammen ausgewertet. Mit den Clusterverfahren werden für jedes Viertel Gruppen bestimmt aus denen dann Segmente gebildet werden. Während der Klassifikation werden dann die Segmente aus jeweils drei der Viertel zum Trainieren verwendet, während die Segmente des übrigen Viertels die Testdaten sind. Die Teilung des Satellitenbildes erfolgt nicht direkt vor der Klassifikation, da jede Segmentinformation eine unterschiedliche Anzahl Pixel repräsentiert. Würden die Segmente einfach in vier Teile aufgeteilt (egal ob zufällig oder nicht), unterscheiden sich die Viertel deutlich bezüglich der Anzahl repräsentierter Pixel. Statt 25 % des Satellitenbildes als Testdaten zu verwenden wären es so vielleicht nur 5 oder aber 50 %. Dies würde die Ergebnisse der Klassifikation verfälschen. Die Viertelung erfolgt auch noch vor der Segmentierung, da diese dadurch eventuell flexibler wird. Die Bodenbedeckungsklassen sind unregelmäßig über das Satellitenbild verteilt. Nicht jede Klasse ist auch in jedem Viertel vorhanden. Die Anzahl der Gruppen, die durch die Clusterverfahren bestimmt werden sollen wird der tatsächlich vorhandenen Klassenzahl in jedem Viertel angepasst. In den Vierteln ol, or, ul und ur gibt es 16, 13, 12 und 14 Klassen.

4.2 Segmentierung / Clusterverfahren

Aus den Pixeln des Satellitenbildes, die hier die Untersuchungseinheiten/ Objekte $1, \ldots, 535860$ sind, werden nun mittels verschiedener Clusterverfahren Segmente bestimmt. Die Pixel sind hierbei zeilenweise nummeriert. Die Clusterverfahren arbeiten mit den Messdaten $x_1, \ldots, x_{535860}$. Für Pixel i besteht $x_i = (x_{i1}, \ldots, x_{iv})^T$ aus den $v = 7$ Reflexionsintensitäten für jeden Kanal des Satellitenbildes. Die Lage der Pixel zueinander wird durch die Messwerte nicht berücksichtigt. Daher werden zunächst nur Gruppen $G_1, \ldots, G_k$ für die Pixel anhand der Clusterverfahren bestimmt. Dann werden alle horizontal und vertikal benachbarten Pixel mit der selben

Gruppenzugehörigkeit als ein Segment festgelegt. Zuletzt werden für jedes Segment die Informationen der zugehörigen Pixel kombiniert sowie weitere Informationen über die Struktur des Segments ermittelt. Die Segmente (beziehungsweise deren Informationen) bilden dann neue Datensätze, die in der anschließenden Klassifikation verwendet werden.

Als Erstes erfolgt also die Bildung der Gruppen anhand der vier Clusterverfahren. Der k-means-Algorithmus von Hartigan und Wong (im Folgenden auch kurz k-means) wird mit der Funktion `kmeans()` ausgeführt. Er wird einzeln auf die Teildatensätze ol, or, ul und ur, also auf jeweils ein Viertel des Satellitenbildes, angewendet. Die Anzahl k der zu bestimmenden Zentren wird mit dem Parameter `centers` auf 16, 13, 12 beziehungsweise 14 gesetzt. Dies entspricht der Anzahl der Bodenbedeckungsklassen aus den CORINE-Daten, die im jeweiligen Teil des Satellitenbildes vorhanden sind. Um die Ergebnisse zu stabilisieren wird die maximale Anzahl an Iterationen des Algorithmus vom Standard 10 auf `iter.max=200` erhöht. Werden nur wenige Iterationen zugelassen kommt es zu Warnmeldungen der Form `"did not converge in 10 iterations"`. Nichtsdestotrotz kommt es weiterhin zu anderen Warnmeldungen des Typs `"Quick-TRANSfer stage steps exceeded maximum (= 6708000)"`. Zum Einen lässt sich dies durch keinen Parameter anders einstellen. Zum Anderen scheint die Anzahl der verwendeten QTRAN-Schritte ausreichend hoch um stabile Ergebnisse zu ermöglichen. Daher wird diese Warnmeldung hier lediglich erwähnt. Die ermittelten Gruppen für das obere linke Viertel (ol) des Bildes sind in Abbildung 9 dargestellt. Hier und im Weiteren erfolgt die Bezeichnung

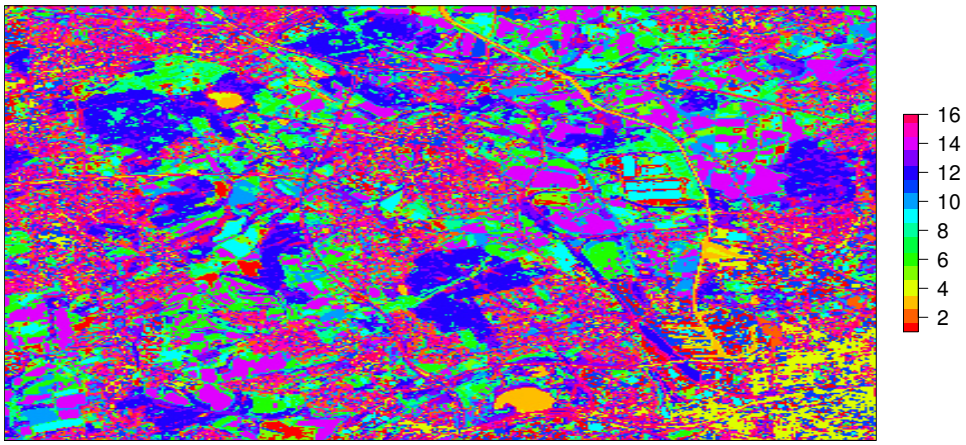


Abbildung 9: Gruppen anhand des k-means-Algorithmus im oberen linken Teil des Satellitenbildes

der Gruppen lediglich anhand ihrer Nummern. Die Gruppen in den anderen Vierteln or, ul und ur sind in den Abbildungen 22 - 24 in Kapitel A des Anhangs zu finden. Zu beachten ist dabei, dass die Farbgebung nur der Unterscheidung der Gruppen innerhalb eines Viertels dient. Da die Bildung der Gruppen auf unterschiedlichen Startzentren beruht, kann zwischen den vier Teilen keine farbliche Beziehung hergestellt werden. Anhand der Grafik soll vielmehr die Lage und Struktur der Gruppen erkennbar werden. In Abbildung 9 sind unterschiedliche Typen von Gruppen erkennbar. Es gibt Gruppen, die über den ganzen Bildausschnitt verteilt sind, wie beispielsweise Gruppe 6 (mittlerer grüner Farbton). Andere Gruppen wie die Gelbe (4) sind fast nur in bestimmten Bereichen, hier in der rechten unteren Ecke, zu finden. Zudem kann unterschieden werden zwischen Gruppen, die sehr oft abgebildet sind (Gruppe 15 in Magenta) und Gruppen, die nur selten erscheinen (Gruppe 3 in Gold). Zuletzt gibt es Gruppen mit (großen) zusammenhängenden Gebieten (Gruppe 14 in Lila). Dem Gegenüber stehen sehr pixelig verteilte Gruppen, die stark mit anderen Gruppen vermischt sind. Beispielsweise sind in den Gebieten der magentafarbenen Gruppe 15 sehr viele kleine Haufen oder einzelne Pixel anderer Gruppen zu sehen. Die großen Bereiche in denen sich diese Gruppe befindet, werden dadurch vielfach aufgespalten. Dies führt später zu wesentlich kleineren Segmenten. Die Anhäufung von kleinen Pixelhaufen und einzelnen Pixeln wird in der Literatur (siehe Lange (2013, S. 445)) auch als “Salz-und-Pfeffer-Effekt“ bezeichnet.

Analog zum k-means-Algorithmus können die Gruppen nun auch mit den anderen drei Clusterverfahren berechnet und dargestellt werden. Für den clara-Algorithmus wird die Funktion `clara` aus dem Paket `cluster` (siehe Maechler et al. (2015)) verwendet. Zusätzlich wird der Parameter `pamLike=TRUE` gesetzt. Die Parameter `samples` und `sampsize` werden bei den Standards von 5 und $40 + 2k$ belassen. Der Standard für sampsize ist genau genommen eigentlich $\min(n, 40 + 2k)$, wobei das Minimum aber mit $k = 16$ (beziehungsweise 13, 12 oder 14) Gruppen und nicht durch $n = 535860$ erreicht wird. Die Struktur der Gruppen unter Verwendung des clara-Algorithmus ist für den unteren linken Teil des Satellitenbildes in Abbildung 10 dargestellt. Auffällig sind hier einige Bereiche beispielsweise der gelben Gruppe 3 in der Mitte des Bildes in denen es nahezu keine Einschlüsse, also Pixel anderer Gruppen im Innern, gibt. Die drei anderen Viertel sind in den Abbildungen 25 - 27 in Kapitel A des Anhangs zu sehen.

Das partielle hierarchische agglomerative Clustern (kurz: phclust) wird mit der ei-

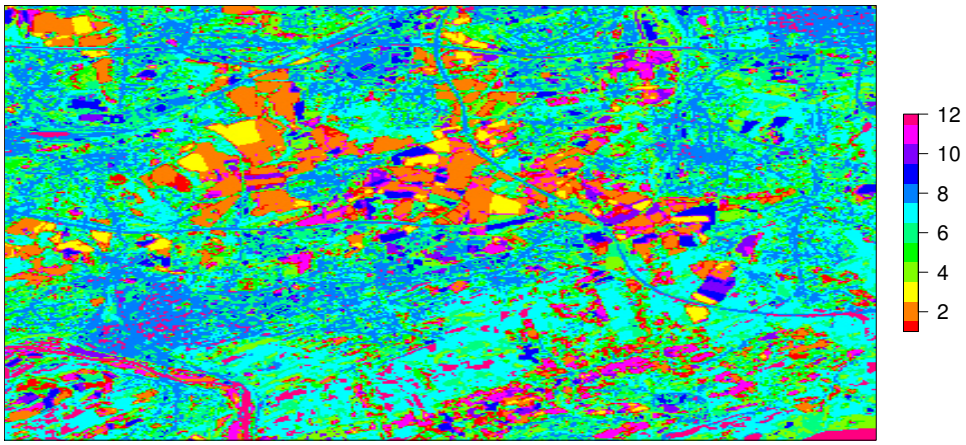


Abbildung 10: Gruppen anhand des clara-Algorithmus im unteren linken Teil des Satellitenbildes

genen Funktion `phclust()` durchgeführt. Der `R`-Code der Funktion ist in Kapitel B.1 des Anhangs ab Seite 102 zu finden. Wie im Kapitel 3.2.3 zur Theorie beschrieben basiert `phclust` auf der Funktion `hclust()` unter Verwendung des Parameters `method="ward.D2"`. Als Parameter der Funktion `phclust` wird die maximale Größe der Teildatensätze mit `partsize=10000` festgelegt. Die Wahl erfolgte dabei anhand der Kapazitäten des ausführenden Computers. Der Grenzwert zur Bestimmung der Gruppenanzahl wird als `gw=sqrt(7*0.25^2)` gewählt. Dies entspricht dem euklidischen Abstand zwischen zwei Objekten, die sich in jedem Kanal um exakt 0.25 unterscheiden. Zwei Objekte mit diesen Eigenschaften erscheinen zu unterschiedlich um sie der gleichen Gruppe zuzuordnen. Alle weiteren Parameter von `phclust` werden hier mit den Standardeinstellungen verwendet. Das Verfahren phclust ist das einzige hier verwendete, bei dem die Anzahl der Gruppen nicht vorher festgelegt, sondern während des Verfahrens bestimmt wird. Für die Viertel ol, or, ul und ur wurden dabei 15, 11, 11 und 17 Gruppen ermittelt. Die erzeugten Gruppen für das rechte obere Viertel des Satellitenbildes sind in Abbildung 11 zu sehen. Bei einigen Darstellungen der Gruppen sind besonders markante Gebäude oder Oberflächen sichtbar. Hier ist deutlich ein Sportstadion mit Laufbahn (in rot) und innen liegender Rasenfläche (in hellblau) am unteren Bildrand auszumachen. Die Darstellungen der Gruppen in den anderen Vierteln befinden sich in Kapitel A des Anhangs (Abbildungen 28 - 30).

Bagged Clustering (kurz: bagclust) wird mit der Funktion `bclust()` umgesetzt. Für die Gruppenbildung in den einzelnen im Verfahren verwendeten Teildatensätzen

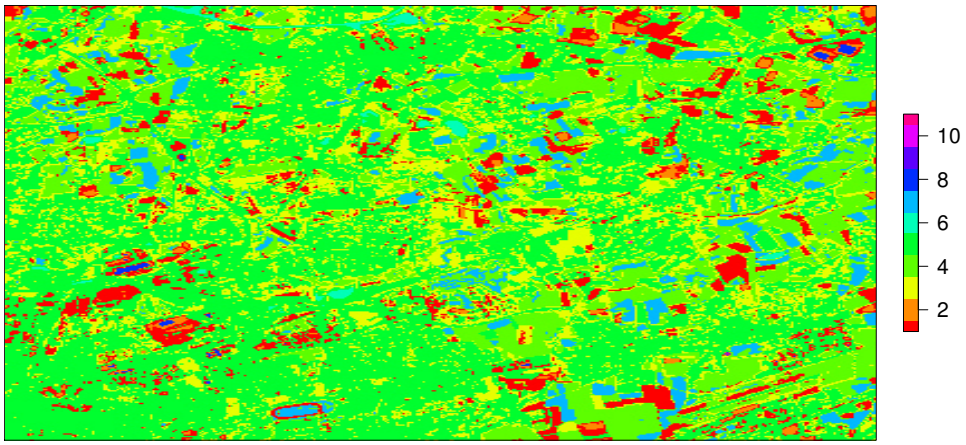


Abbildung 11: Gruppen anhand des phclust-Verfahrens im oberen rechten Teil des Satellitenbildes

wird jeweils der k-means-Algorithmus mit der Funktion `kmeans()` genutzt. Der eingestellte Standard von 20 Gruppen pro Teildatensatz wird hier übernommen. Ebenso wird die Standardanzahl an Teildatensätzen von 10 verwendet. Wie beim k-means-Algorithmus wird die maximale Iterationsanzahl auf `iter.max=200` gesetzt. Dieser Parameter wird von der Funktion `kmeans()` innerhalb von `bclust` aufgegriffen und verwendet. Da `kmeans` verwendet wird, kommt es auch hier zu Warnmeldungen, die angeben, dass die maximale Anzahl an QTRAN-Schritten erreicht ist. Zur Kombination der Gruppen aus den Teildatensätzen wird nun die Funktion `hclust()` benutzt. Da diese standardmäßig nicht den Ward-Algorithmus nutzt, wird dieser über den Parameter `hclust.method="ward.D2"` eingestellt. Die Anzahl der Gruppen, die von `bclust` erzeugt werden soll, wird über den Parameter `centers` auf 16, 13, 12 beziehungsweise 14 festgelegt. Die vom Verfahren Bagged Clustering erzeugten Gruppen für das untere rechte Viertel des Satellitenbildes sind in Abbildung 12 dargestellt. Auffällig ist hier die starke Vermischung der Gruppen 2 (orange) und 6 (mittleres Grün). Die Gruppen in den anderen Vierteln sind in den Abbildungen 31 - 33 in Kapitel A des Anhangs zu sehen. Das Teilziel 1a), die Gruppierung der Pixel mittels Clusterverfahren, ist damit erreicht.

Anhand der soeben bestimmten Gruppen werden nun Segmente gebildet. Horizontal und vertikal benachbarte Pixel mit der gleichen Gruppenzugehörigkeit bilden ein Segment. Dies wird veranschaulicht durch Abbildung 13 in der die Pixel in der oberen linken Ecke des Viertels ol dargestellt sind. Das verwendete Verfahren ist dabei phclust. Gäbe es nur die abgebildeten 25 Pixel, so würden die Segmente $S_1, \ldots, S_{14}$

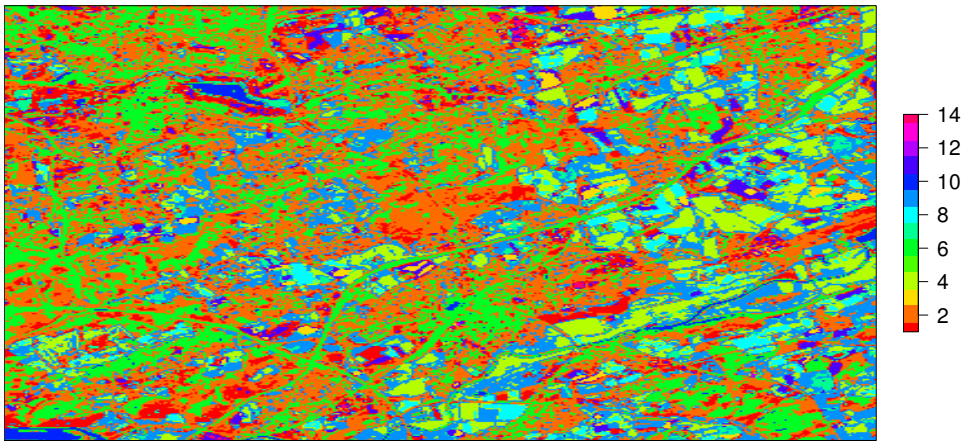


Abbildung 12: Gruppen anhand des Verfahrens Bagged Clustering im unteren rechten Teil des Satellitenbildes

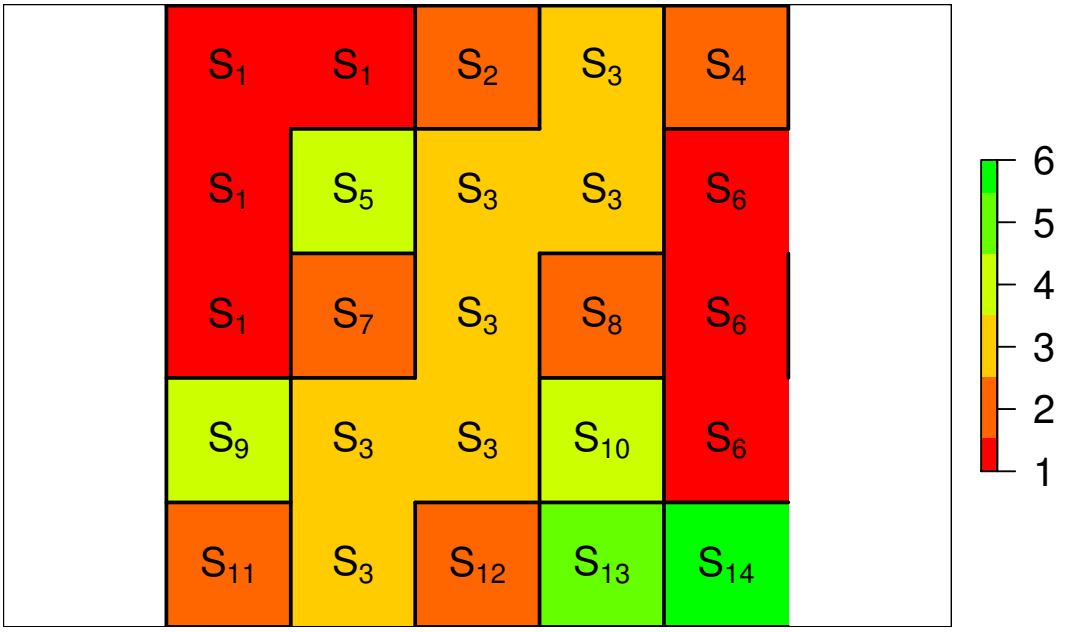


Abbildung 13: Bildung der Segmente im Detail

anhand der sechs vorhandenen Gruppen gebildet werden. Gut zu erkennen ist, das Pixel der gleichen Gruppe nicht unbedingt im gleichen Segment sind. So gibt es beispielsweise im dargestellten Ausschnitt sieben Pixel der Gruppe 1. Diese verteilen sich aber auf die Segmente S_1 und S_6. Die Suche nach einer Umsetzung der Segmentbildung in `R` verlief leider ergebnislos. Da im weiteren Verlauf mehrmals Segmente gebildet werden, wurde eine eigene Funktion `segmentsfromlayer()` erstellt. Die Details und der R-Code der Funktion sind in Kapitel B.1 des Anhangs ab Seite 105 zu finden. Wie der Funktionsname schon andeutet, werden anhand der Werte des Layers eines Bricks Segmente konstruiert. Daher wird nun für jedes Clusterverfahren und jedes Viertel ol, or, ul und ur ein eigener leerer Brick, mit den gleichen Eigenschaften wie beim Satellitenbild, erzeugt. Im ersten Layer jedes Bricks wird nun die passende Gruppenzugehörigkeit als numerischer Wert $1, \ldots$ abgelegt. Dies geschieht mit der Funktion `setValues()`, die wie die Funktion `brick()` zur Erzeugung der Bricks, aus dem Paket `raster` (siehe Hijmans (2015)) stammt. In der Funktion `segmentsfromlayer()` wird der jeweilige Brick und der zugehörige Layer angegeben anhand dessen die Segmente konstruiert werden sollen. Das Ergebnis sind zwei Listen. Die erste Liste mit dem Namen "Segmente" beinhaltet die gebildeten Segmente. Jedes Element der Liste entspricht einem Segment. Die Elemente bestehen aus Vektoren, die die Nummern der Pixel dieses Segments enthalten. Die Pixel des Satellitenbildes wurden zu diesem Zweck zeilenweise von 1 bis 535860 durchnummeriert. Die zweite Liste gibt für jedes Segment die Gruppe an, die bei allen betroffenen Pixeln gleich ist und zur Segmentbildung geführt hat. Die Angaben der zweiten Liste werden für spätere Berechnungen benötigt. Die beiden Listen sind in der Regel recht groß und unübersichtlich. Daher werden hier nur die ersten zehn Segmente des oberen linken Viertels bei Verwendung des k-means-Algorithmus gezeigt (siehe Tabelle 4). Die Segmente lassen sich generell besser in den Grafiken zu den Gruppen erkennen. Deshalb werden an dieser Stelle keine weiteren Ergebnisse zu den Segmenten aufgeführt. Teilziel 1b) (Bildung der Segmente anhand der Gruppen) ist mit den vorliegenden Listen der Segmente erfüllt.

Im Folgenden werden nun die mittels der Clusterverfahren erstellten Segmentierungen bewertet. Die Bewertung erfolgt im Hinblick auf die angestrebte Nachempfindung der Struktur der Bodenbedeckung, da im Weiteren nach dieser klassifiziert wird. Die Bodenbedeckung ist hier durch die CORINE-Daten gegeben. Anhand dieser Daten können Segmente für die Viertel des Satellitenbildes bestimmt werden,

Tabelle 4: Erste 10 Segmente der Funktion `segmentsfromlayer()` für das obere linke Viertel des Satellitenbildes bei Verwendung des k-means-Algorithmus

Segment	Pixelnummern
1	2
2	7, 8, 395, 396, 397, 398, 785
3	183, 184, 572, 573, 574, 963
4	223, 224, 225, 226, 613, 614, 616, 1003, 1005, 1006, 1007, 1393, 1395, 1396, 1782, 1783, 1784, 1785, 1786, 2175
5	270, 271, 660, 661, 1050, 1051, 1440
6	390
7	749, 1139
8	1417
9	1447
10	1589, 1590, 1591, 1980

die dann mit den Segmentierungen durch die Clusterverfahren verglichen werden können. Da die CORINE-Daten während der Vorverarbeitung bereits in Layer 10 des Satellitenbild-Bricks abgelegt und anschließend geviertelt wurden, können die zugehörigen Segmente nun relativ einfach mit der Funktion `segmentsfromlayer()` bestimmt werden. Die Bewertung kann auf unterschiedliche Arten vollzogen werden. Viele Quellen aus dem Bereich der Fernerkundung schlagen einen visuellen Vergleich der Segmentierungen durch den Anwender vor. Dabei können die Segmentierungen aus den Clusterverfahren sowohl untereinander als auch mit den Segmentierungen aus den CORINE-Daten verglichen werden. Die Segmente sind, wie bereits erwähnt, besser in Grafiken für die Gruppenzugehörigkeit zu erkennen. Der visuelle Vergleich erfolgt hier beispielhaft für das obere rechte Viertel des Satellitenbildes anhand der Abbildung 14. Zu vergleichen ist hier erneut nur die Struktur und nicht die Farbgebung der Grafiken. Beim Vergleich der Clusterverfahren untereinander fallen einige Unterschiede auf. Beim k-means-Algorithmus gibt es beispielsweise viele kleine Segmente, während beim partiellen hierarchischen agglomerativen Clustern das Gegenteil der Fall ist. Bei den beiden genannten Verfahren fällt zudem auf, dass beim Ersten alle Gruppen recht gleichmäßig verteilt sind beziehungsweise beim Zweiten einige Gruppen nur mit wenigen kleinen Segmenten vorhanden sind. Be-

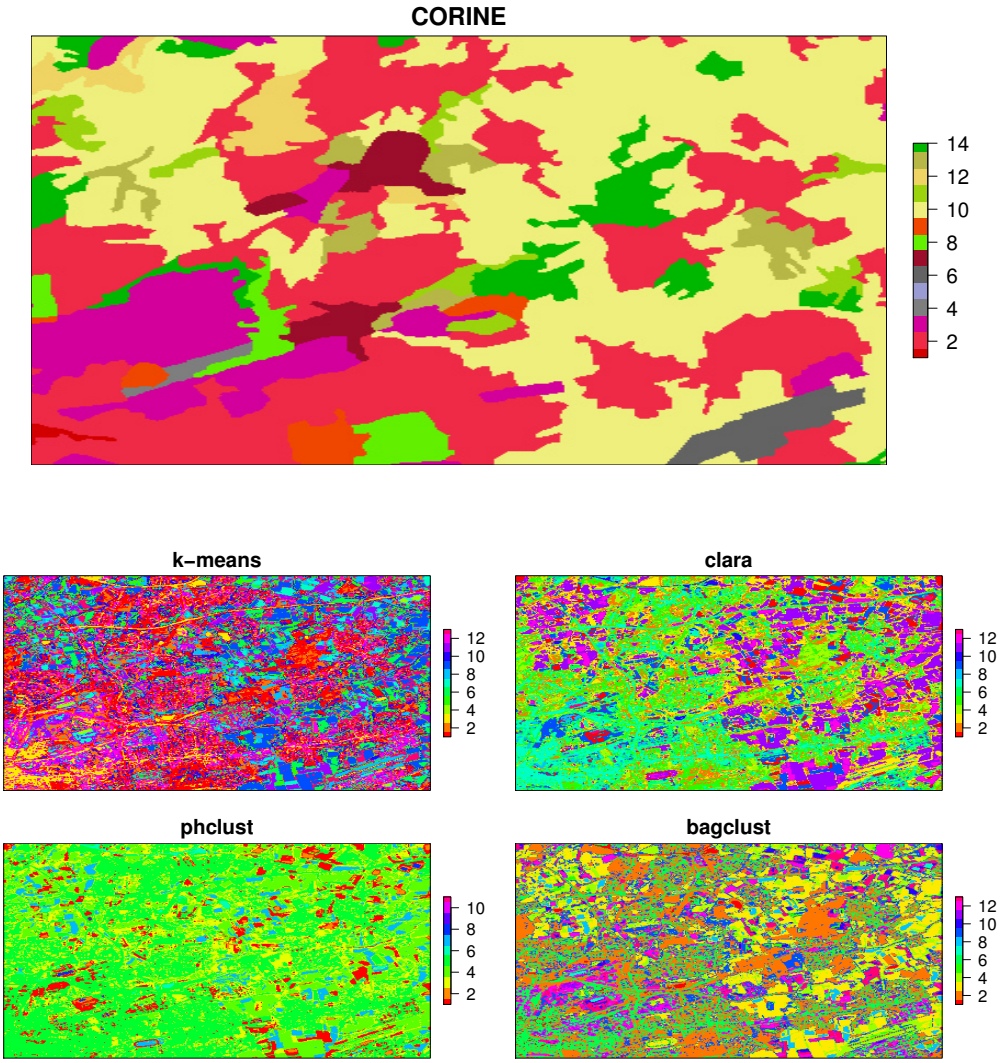


Abbildung 14: CORINE-Klassen und Gruppen anhand der vier Clusterverfahren im oberen rechten Teil des Satellitenbildes

sonders auffällige Bereiche, wie das schon erwähnte Sportstadion am unteren Bildrand, werden unterschiedlich gut erkannt. Am besten scheint hier phclust und am schlechtesten der k-means-Algorithmus zu funktionieren. Der clara-Algorithmus und Bagged Clustering befinden sich zwischen den "Extremen" der anderen beiden Verfahren. Beim Vergleich der Segmentierung aus den CORINE-Daten ist zu erkennen, dass keine der Segmentierungen aus den Clusterverfahren wirklich gut die Strukturen der Bodenbedeckungsklassen annähert. Dies liegt auch daran, dass die Segmente der Bodenbedeckungsklassen sehr groß sind. Sie beinhalten mitunter auch verschieden beschaffene Oberflächen (beispielsweise Garten und Hausdach in Klasse 1 (Flächen durchgängig städtischer Prägung)). Die beschriebenen Eigenschaften lassen sich auf die anderen Viertel ol, ul und ur übertragen. Eine visuelle Bewertung, welches Verfahren bezüglich der CORINE-Daten am besten zur Segmentierung geeignet ist, fällt daher schwer. Eine solche Bewertung wäre zudem auch immer subjektiv abhängig vom Anwender und dessen persönlicher Einschätzung. Eine objektivere Form der Bewertung soll im Folgenden durch die Berechnung der NIBS-Distanzen zwischen den Segmentierungen aus den CORINE-Daten und denjenigen aus den Clusterverfahren bestimmt werden. Dazu werden für jedes Viertel des Satellitenbildes die Segmente aus den CORINE-Daten und jeweils eines Clusterverfahrens verwendet. Pro Viertel werden also vier NIBS-Distanzen bestimmt. Da die NIBS-Distanz eine eigene Idee ist, die auf der LCS-Distanz beruht, gibt es nur Funktionen die Letztere berechnen. Ein paarweiser Vergleich von Segmenten wäre mit diesen Funktionen jedoch kompliziert und langwierig geworden. Daher wurde eine eigene Funktion `nibs()` erstellt, die direkt die NIBS-Distanz berechnet und auf Effizienz ausgerichtet ist. Der `R`-Code zur Funktion ist in Kapitel B.1 des Anhangs ab Seite 108 zu finden. Um eine Bewertung für das gesamte Bild zu erhalten, werden die Distanzen über die Clusterverfahren hinweg aufsummiert. Die einzelnen Distanzen sowie die Summen sind in Tabelle 5 zu finden. Die ermittelten Distanzen sind alle sehr groß. Die Verfahren sind also nicht gut geeignet um die Struktur von CORINE-Bodenbedeckungsklassen anzunähern. Die kleinste Summe und damit die beste Annäherung liefert noch das partielle hierarchische agglomerative Clustern. Ingesamt scheint die Nachbildung der Segmente aus den Bodenbedeckungsklassen sehr schwierig zu sein. Dies wurde sowohl visuell als auch bei den NIBS-Distanzen sichtbar. Eventuell funktioniert ein anderes Verfahren, welches nicht aus dem Bereich Clustern stammt, besser. Diese Arbeit ist jedoch auf Clusterverfahren beschränkt, sodass andere Verfahren gegebe-

Tabelle 5: NIBS-Distanzen

	ol	or	ul	ur	Summe
k-means	326 142 226	590 608 135	417 971 866	289 412 271	1 624 134 498
clara	276 524 036	507 991 168	425 229 250	299 062 777	1 508 807 231
phclust	97 824 524	245 816 132	153 054 913	93 079 813	589 775 382
bagclust	134 544 074	437 893 096	150 187 222	167 742 900	890 367 292

nenfalls in folgenden Untersuchungen durchgeführt werden könnten. Die Bewertung der Segmentierungen durch die Clusterverfahren, also Teilziel 1c), wird somit als abgeschlossen betrachtet.

Anhand der Segmentierungen aus den Clusterverfahren werden nun für die Klassifikation im nächsten Kapitel neue Datensätze konstruiert. Zwecks Übersichtlichkeit werden die Datensätze und deren Daten im Folgenden mit den Abkürzungen KM, CL, PH und BC gekennzeichnet. Die Abkürzungen signalisieren, anhand welches Clusterverfahrens (k-means, clara, phclust beziehungsweise bagclust) der Datensatz gebildet wurde. Die Segmente, aus denen die Datensätze entstehen, erhalten die gleiche Kennzeichnung. Im Folgenden wird die Konstruktion des k-means-Datensatzes erläutert. Die anderen Datensätze werden in analoger Weise erstellt. Da die Clusterverfahren in den Vierteln ol, or, ul und ur des Satellitenbildes separat ausgeführt wurden, wird zunächst für jedes Viertel ein Datensatz bestimmt. Die Konstruktion der Datensätze in den Vierteln erfolgt auf die gleiche Weise, weswegen im Folgenden lediglich die Entstehung des Datensatzes für das Viertel ol beschrieben wird. Jedes Segment des Viertels bildet ein neues Objekt / einen neuen Datenpunkt. Die Segmentierung mit k-means in ol ergab $n^{\mathrm{KM\,ol}} = 40966$ Segmente, womit nun ebenso viele Datenpunkte $1, \ldots, n^{\mathrm{KM\,ol}}$ vorliegen. Die Bezeichnung KM ol deutet dabei sowohl auf das Clusterverfahren KM als auch das betroffene Viertel ol hin. Die vorliegenden Segmente für das obere linke Viertel unter Verwendung des k-means-Algorithmus werden mit $S_1^{\mathrm{KM\,ol}}, \ldots, S_{40966}^{\mathrm{KM\,ol}}$ bezeichnet, wobei die ersten zehn Segmente in Tabelle 4 dargestellt wurden. Analog hierzu werden Bezeichnungen mit anderen Clusterverfahren und / oder anderen Vierteln verwendet.

Als Erstes werden für die Erstellung eines Datensatzes die Messwerte der Pixel jedes Segments zusammengefasst. Dies betrifft nur die ersten sieben Messwerte, die die Reflexion in den Spektralkanälen darstellen. Hier wird innerhalb der Kanäle jeweils

das arithmetische Mittel, wie in (84) dargestellt, berechnet.

$$\bar{x}_j^{\mathrm{KM\,ol}} = \frac{1}{n_j^{\mathrm{KM\,ol}}} \sum_{i \in S_j^{\mathrm{KM\,ol}}} x_i \quad , j = 1, \ldots, n^{\mathrm{KM\,ol}} \tag{84}$$

$n_j^{\mathrm{KM\,ol}}$ steht für die Anzahl der Pixel in Segment j. Jeder Mittelwertvektor enthält die ersten sieben Messwerte des Segments im neuen Datensatz. Zusätzlich werden für jedes Segment weitere Informationen über dessen Struktur hinzugefügt. Als achte und neunte Angabe wird die Breite und Höhe des Segments (in Pixeln) vermerkt. Zur Berechnung der Breite und Höhe werden die Koordinaten der Pixel genutzt, die während der Datenvorverarbeitung in den Layern 8 und 9 des Satellitenbild-Bricks abgelegt wurden. In beiden Richtungen (horizontal und vertikal) wird jeweils die minimale von der maximalen Koordinate abgezogen und das Ergebnis durch die Auflösung von 30 Einheiten pro Pixel geteilt. Die zehnte Angabe zu jedem Segment ist die Anzahl der zugehörigen Pixel. Die Werte werden dabei einfach aus der Liste mit den Segmenten, wie beispielhaft in Tabelle 4 zu sehen, bestimmt. Zusammen sollen Breite, Höhe und Pixelzahl die Form des Segments wiedergeben. Ein lang gestrecktes Segment beispielsweise wird zwar recht breit und / oder hoch sein, aber verhältnismäßig wenig Pixel beinhalten. Ein kompaktes Segment dagegen wird in Bezug auf Breite und Höhe aus vielen Pixeln bestehen. Ein weiteres Indiz zur Einordnung der Segmente könnte die Ähnlichkeit der Pixel innerhalb des Segments sein. Dazu wird die empirische Standardabweichung in einem Segment wie in (85) zu sehen berechnet.

$$\hat{\sigma}_j^{\mathrm{KM\,ol}} = \sqrt{\frac{1}{n_j^{\mathrm{KM\,ol}} - 1} \sum_{i \in S_j^{\mathrm{KM\,ol}}} (x_i - \bar{x}_j^{\mathrm{KM\,ol}})^2} \quad , j = 1, \ldots, n^{\mathrm{KM\,ol}} \tag{85}$$

Die Standardabweichungen werden als elfter Wert zu jedem Segment festgehalten. Während der Klassifikation werden Vorhersagen für die Bodenbedeckungsklasse eines Segments gemacht. Um die Richtigkeit der Vorhersage zu prüfen, wird eine Angabe zur korrekten Klasse des Segments benötigt. Diese Angaben sollen aus den CORINE-Daten kommen. Diese liegen allerdings nur für die einzelnen Pixel vor und wurden bei der Bestimmung der Segmente auch nicht berücksichtigt. Es kann also vorkommen, dass den Pixeln eines Segments unterschiedliche Bodenbedeckungsklassen zugeordnet sind. Daraus könnte sich die Situation ergeben, dass die Vorhersage für einige Pixel des Segments richtig und für andere falsch ist. Um eine klare Aussage (richtig oder falsch) für das ganze Segment treffen zu können wird daher nach einer

einzigen Klasse gesucht, die das Segment ausreichend beschreibt. Hier wird die Klasse verwendet, die im Segment am häufigsten auftritt. Wird die maximale Anzahl von mehreren Klassen erreicht, wird eine von ihnen zufällig ausgewählt. Um eine Information darüber zu erhalten, wie gut die Klasse das Segment repräsentiert, wird der Anteil der Pixel mit dieser Klasse im Segment berechnet. Der Anteil und die Klasse selbst werden als zwölfte und dreizehnte Werte jedes Segments festgehalten. Die Anteile sollten nicht zur Klassifikation selbst genutzt werden, da die Informationen aus den CORINE-Daten und damit den echten Bodenbedeckungsklassen gewonnen wurden. Sie können aber zur Interpretation der Ergebnisse verwendet werden. Zum Beispiel könnte ein Segment mit vielen unterschiedlichen Klassen schwer vorhersagbar sein. Dies könnte mit dem Anteil recht einfach erkannt werden. Die 13 Messwerte aller Segmente bilden nun den neuen Datensatz. Auf obige Weise werden nun auch die Datensätze der anderen Viertel or, ul und ur bei Verwendung des k-means-Algorithmus erstellt. Da die Konstruktion der Datensätze immer auf die gleiche Weise geschieht, wurde für einen reibungslosen Ablauf und zur Vermeidung von Fehlern eine eigene Funktion `constructSegdata()` verwendet. Diese bildet unter Angabe eines Bricks und einer Segmentierung einen Datensatz wie hier beschrieben. Der R-Code der Funktion ist in Kapitel B.1 des Anhangs ab Seite 110 zu finden.
Während der Klassifikation sollen jeweils drei der vier Datensätze zum Trainieren des Algorithmus und der Vierte zum Testen genutzt werden. Um die Berechnungen in `R` zu vereinfachen werden die Objekte der vier Datensätze zu einem einzigen Datensatz kombiniert. Für die Kreuzvalidierung ist es unerheblich, ob einer oder vier Datensätze vorliegen. Es muss lediglich bekannt sein, welche Objekte zu welchem Viertel gehören, um diese dann jeweils als Trainings- beziehungsweise Testdaten zu verwenden. Der Einfachheit halber erscheinen die Segmente der Reihenfolge nach im Datensatz. Zuerst sind die 40966 Segmente des oberen linken Viertels aufgeführt. Es folgen die 30392 und 28587 Segmente der Viertel or und ul. Zuletzt kommen die 30580 Segmente des Viertels ur hinzu. Der gemeinsame Datensatz wird, wie oben bereits angedeutet, in der Folge durch das Kürzel KM ohne weiteren Zusatz für das Viertel angezeigt. Die durch die anderen Clusterverfahren erzeugten je vier Datensätze werden auf die gleiche Weise kombiniert und mit den Abkürzungen CL, PH und BC versehen. Letztendlich liegt nun also für jedes der vier Clusterverfahren ein Datensatz vor. Innerhalb jedes dieser Datensätze werden nun noch die Höhe, Breite, Pixelzahl und Standardabweichung auf das Intervall $[0, 1]$ skaliert. Dies verhindert

Probleme bei einigen Klassifikationsverfahren. Wird bei einem Verfahren beispielsweise die euklidische Distanz verwendet, würde diese ansonsten durch die großen Werte bestimmt werden und Unterschiede in den Kanälen nicht mehr erkennen. Durch die Kombination von jeweils vier Datensätzen der Viertel wird für die Einteilung der gemeinsamen Datensätze in Trainings- und Testdaten noch die Anzahl der Segmente pro Clusterverfahren und Viertel benötigt. Diese Informationen sind in Tabelle 6 einsehbar. Da die Segmente der Reihenfolge nach angeordnet wurden,

Tabelle 6: Anzahl Segmente nach Clusterverfahren und Viertel des Satellitenbildes getrennt

	ol	or	ul	ur	Gesamt
k-means	40966	30392	28587	30580	130525
clara	35989	29241	30041	30449	125720
phclust	13680	11082	11167	9477	45406
bagclust	19306	21779	11285	18597	70967

genügt diese Information für die Einteilung. Die Tabelle enthält eine weitere Spalte "Gesamt", in der die Segmentzahl in den gemeinsamen Datensätzen aufgeführt ist. Die Unterschiede zwischen den Clusterverfahren finden sich auch hier wieder. Ebenfalls erkennbar sind teils große Unterschiede zwischen den Vierteln. Die Aufteilung in die Viertel bereits ganz zu Beginn war also durchaus sinnvoll, da eine zufällige Auswahl von 25 % der Segmente (als Testdaten) zu sehr unterschiedlichen Anteilen repräsentierter Pixel geführt hätte. Mit der Erstellung der gemeinsamen Datensätze ist das letzte Teilziel 1d) dieses Kapitels erreicht worden. Damit wurden alle Voraussetzungen geschaffen, um die Segmente im nächsten Kapitel problemlos klassifizieren zu können.

4.3 Klassifikation

4.3.1 Segmentbasierte Klassifikation

Für die Klassifikation wird das `mlr`-Paket (siehe Bischl et al. (2016)) verwendet. Alle im Weiteren aufgeführten Funktion stammen, soforn nicht anders angegeben aus diesem Paket. Die Struktur von `mlr` unterscheidet es von vielen anderen Paketen.

Es ist nicht nur eine Sammlung von Funktionen für Klassifikationsverfahren, sondern steuert auch die Rahmenbedingungen für das gesamte Klassifikationsproblem. Die Vorgehensweise, von der Eingabe der Daten bis zum Parametertuning der Verfahren, soll damit vereinheitlicht werden. Dies erfordert am Anfang etwas Zeit um sich in die Vorgehensweise des Pakets einzuarbeiten, führt langfristig jedoch zu einer vereinfachten Bearbeitung von Klassifikationsproblemen. So muss beispielsweise bei den Verfahren nicht mehr überprüft werden, in welcher Struktur die Daten eingegeben werden müssen, oder wie die Ausgabe der Ergebnisse genau aussieht. Um die weitere Analyse verständlich zu machen, wird zunächst der Aufbau von `mlr` näher erklärt.

Wie bereits erwähnt, werden zunächst die Rahmenbedingungen des zu lösenden Klassifikationsproblems festgelegt. Als Erstes wird ein sogenannter "Task" definiert. Dieser beinhaltet die Daten selbst und zusätzliche Informationen über deren Struktur. Der Task wird mit der Funktion `makeClassifTask()` erzeugt. Der Mittelteil "Classif" deutet dabei an, dass es sich um die Beschreibung eines Klassifikationsproblems handelt, da mit dem Paket auch andere Arten der Analyse in analoger Weise durchgeführt werden können. Für die Erstellung des Tasks werden ein Datensatz in Form eines Data Frames und der Spaltenname der zu klassifizierenden Variable benötigt. Die zu klassifizierende Variable (`target`) muss hierbei im `factor`-Format vorliegen. Für jeden Task wird eine ID erstellt, die eine eindeutige Identifizierung ermöglicht. Sie kann durch den Parameter `id` vom Anwender bestimmt werden. Aus den vorigen Kapiteln liegen vier Datensätze KM, CL, PH und BC vor, für die nun jeweils ein Task erzeugt wird. Zur Erinnerung wird hier nochmals der aus dem k-means-Algorithmus resultierende Datensatz vorgestellt. Für die $n^{KM} = 130525$ Objekte (Segmente) liegen jeweils Messwerte $x_i = (x_{i1}^{KM}, \ldots, x_{iv}^{KM})^T$, $i = 1, \ldots, 130525$, von $v = 11$ Variablen vor. Die Variablen (auch Features genannt) stehen dabei für den Mittelwert der sieben Spektralkanäle, die Höhe, Breite und Pixelzahl, sowie die Standardabweichung in jedem Segment. Hinzu kommt für jedes Objekt i, $i = 1, \ldots, 130525$, die Bodenbedeckungsklasse aus den CORINE-Daten $y_i^{KM} \in \{C_1, \ldots, C_{14}, C_{16}, \ldots, C_{19}\}$. Die Nummerierung entspricht dabei den Bodenbedeckungsklassen $1, \ldots, 19$. Folglich wird die nicht im Satellitenbildausschnitt erscheinende Klasse 15 ausgelassen. Die anderen Datensätze sind in analoger Weise aufgebaut. Ein Aufruf eines Tasks zeigt eine kurze Zusammenfassung des Datensatzes an. Unter anderem werden die Anzahl der Objekte, der Name des targets

und die Task-ID angezeigt. Letztere ist nach den Clusterverfahren gewählt, anhand derer die Datensätze entstanden. Ebenfalls angegeben wird die Anzahl der Objekte in den Klassen des targets. Für die anderen Spalten des Datensatzes wird deren Format (`numerics`, `factors` oder `ordered`) zusammengefasst. Einige Klassifikationsverfahren können Werte bestimmter Formate nicht verwenden und würden daher zur Analyse des Problems bereits ausscheiden. Beispielsweise kann das Minimum-Distance-Verfahren wegen der euklidischen Distanz nur mit numerischen Werten durchgeführt werden. Die vorliegenden 11 Variablen / Features sind aber alle numerisch, sodass alle ausgewählten Klassifikationsverfahren problemlos durchgeführt werden können.

Die Rahmenbedingungen jedes Verfahrens werden vor dessen Durchführung ebenfalls in einer eigenen Umgebung festgelegt. Dazu wird die Funktion `makeLearner()` verwendet, die einen sogenannten "Learner" erzeugt. Es können sowohl eigene als auch bestehende Verfahren verwendet werden. Einige der bestehenden Verfahren sind bereits in `mlr` eingebunden, sodass lediglich der Name des Learners eingegeben werden muss. Eventuell benötigte Pakete werden automatisch aufgerufen. Drei der im Folgenden verwendeten Verfahren sind bereits eingebunden. Die lineare Diskriminanzanalyse wird mit der Funktion `lda()` aus dem Paket `MASS` (siehe Venables und Ripley(2002)) unter dem Namen "classif.lda" geführt. Der Random Forest (Funktion `randomForest()` aus dem gleichnamigen Paket von Liaw und Wiener (2002)) trägt die Bezeichnung "classif.randomForest". Die Support Vector Machine in Form der Funktion `svm()` aus dem Paket `e1071` (siehe Meyer et al. (2015)) ist mit "classif.svm" benannt. Soll das Verfahren mit den jeweiligen Standardeinstellungen der Parameter ausgeführt werden, so genügt bei der Duchführung die Angabe des Namens. Veränderungen der Parameter können durch die Funktion `makeLearner()` über den Parameter `par.vals` in Form einer Liste angegeben werden. Die Parameter werden bei Aufruf des so erzeugten Learners unter dem Eintrag "Hyperparameters" abgespeichert. Zusätzlich sind weitere Eigenschaften des Learners, wie beispielsweise der "Predict-Type", dargestellt. Der Predict-Type gibt die Form der Vorhersage an und kann als `response` oder `prob` gewählt werden. Bei erstgenannter Form wird die Klasse ganz normal in Form eines Levels des Factors target ausgegeben. Mit der Variante `prob` wird für jedes Level (also jede Klasse) eine Wahrscheinlichkeit ausgegeben. Standardmäßig wird die Klasse mit dem höchsten Wert vorhergesagt. Dies kann jedoch auch variiert werden. Die Berechnung der Wahrscheinlichkeiten

ist der Angabe der Klasse vorzuziehen, allerdings nicht immer möglich. Bei manchen Verfahren, wie Minimum-Distance oder Quader, können keine Wahscheinlichkeiten angegeben werden. Es wird daher bei allen Verfahren die Standardeinstellung `predict.type = "response"` verwendet.

Bisher wurde in Bezug auf `mlr` noch nicht auf das Minimum-Distance-, das Maximum-Likelihood- und das Quader-Verfahren eingegangen. Wie den jeweiligen Theoriekapiteln 3.3.5, 3.3.6 und 3.3.7 zu entnehmen ist, werden für alle drei Verfahren eigene Funktionen verwendet. Diese müssen nun in `mlr` eingebunden werden, damit die Vorgänge während der Klassifikation weiterhin einheitlich sind und weitergehende Funktionen des Paktes ebenfalls genutzt werden können. Um ein Verfahren einzubinden wird dreierlei benötigt. Zuerst wird der interne Name des Verfahrens und dessen Eigenschaften mit der Funktion `makeRLearnerClassif()` erstellt. Der interne Name wird mit dem Parameter `cl` angegeben, wobei ein Klassifikationsverfahren mit "classif." beginnen soll. Die Eigenschaften umfassen Angaben, die auch bei Aufruf der anderen Learner schon erschienen. Mit dem Parameter `properties` wird angegeben, bei welcher Art Klassifikationsproblem das Verfahren angewendet werden kann. Es wird unterschieden nach Zweiklassen- und Mehrklassenproblemen (`twoclass` sowie `multiclass`). Ebenso wird angegeben, ob die Verfahren mit numerischen (`numerics`), nominal (`factor`) beziehungsweise ordinal (`ordered`) skalierten Features ausgeführt werden können. Die Eigenschaft `prob` findet sich bei Verfahren, die Wahrscheinlichkeiten für die Klassen berechnen können. Die Verfahren können mehrere (oder auch alle) Eigenschaften gleichzeitig erfüllen. Die Parameter des einzubindenden Klassifikationsverfahrens werden durch ein Parameter-Set über `par.set` angegeben. Das Parameter-Set kann mit der Funktion `makeParamSet()` erstellt werden. Innerhalb des Sets werden die einzelnen Parameter definiert, indem ihnen ein Name, ein Standardwert und die Menge möglicher Werte zugewiesen werden. Je nachdem, ob ein numerischer, diskreter oder logischer Parameter erstellt werden soll, wird dazu die Funktion `makeNumericLearnerParam()`, `makeDiscreteLearnerParam()` oder `makeLogicalLearnerParam()` genutzt. Die Details zu den Parametern von Minimum-Distance-, Maximum-Likelihood- und Quader-Verfahren sind in der Erläuterung des `R`-Codes der erstellten Funktionen im Anhang (ab Seite 112, 115 beziehungsweise 119) zu finden. Die Verwendung der Parameter wird ebenfalls beim Parametertuning näher erklärt. Für die Funktion `makeRLearnerClassif()` werden weiterhin die Parameter `name`, `short.name` und `package` verwendet. `name` und

`short.name` besitzen für die Durchführung wenig Relevanz und geben einen offiziellen und einen Kurznamen für das Verfahren an. Mit `package` werden die benötigten `R`-Pakete angegeben, die dann vor der Durchführung des Verfahrens geladen werden. Für alle drei Verfahren wird das Paket `nnet` (siehe Venables und Ripley (2002)) angegeben, da die darin enthaltene Funktion `which.is.max()` verwendet wird. Für das Maximum-Likelihood- sowie das Quader-Verfahren wird außerdem zur Berechnung multivariater Normalverteilungen das Paket `mvtnorm` (vergleiche Genz et al. (2016)) genutzt.

Nachdem auf obige Weise eine Basis für die einzubindenden Verfahren geschaffen wurde, wird noch jeweils eine Modellfunktion und eine Vorhersagefunktion benötigt. Die Modellfunktion soll anhand der (Trainings-) Daten ein Modell beliebiger Form erstellen. Anhand des Modells und der Vorhersagefunktion sollen dann Vorhersagen für (Test-) Daten gemacht werden können. Als Zweites wird bei der Einbindung eines Verfahren in `mlr` also die zu verwendende Modellfunktion angegeben. Beim Minimum-Distance-Verfahren ist dies die eigene Funktion `mindist.mod()`, deren Details in Kapitel B.2 des Anhangs ab Seite 112 zu finden sind. Als Modell wird eine Liste erstellt, die unter Anderem die arithmetischen Mittelwerte (`means`) als Zentroide für die Gruppen enthält. Außerdem enthält die Liste nochmals die Daten und den Namen der Spalte des target. Die Modellfunktion `maxlike.mod()` (siehe Kapitel B.2 des Anhangs ab Seite 115) für das Maximum-Likelihood-Verfahren erzeugt als Modell ebenfalls eine Liste. In dieser finden sich alle Angaben, die auch durch `mindist.mod` erzeugt werden. Zusätzlich sind noch die Schätzungen für Kovarianzmatrizen (`covmats`) und a-priori-Wahrscheinlichkeiten (`apriori`) der Klassen in der Liste vorhanden. Das Modell des Quader-Verfahrens wird durch die eigene Funktion `quader.mod()` (siehe Kapitel B.2 des Anhangs ab Seite 119) erzeugt. Das erstellte Modell in Form eine Liste hat Ähnlichkeit mit dem Modell des Maximum-Likelihood-Verfahrens. Statt `apriori` gibt es hier einen Eintrag `boxes`, der die für jede Gruppe berechneten Quader enthält. Zusätzlich werden noch die Einstellungen der Parameter `tiemethod` und `cases` angegeben, da diese erst für die Vorhersage benötigt und dann aus dem Modell heraus ermittelt werden. Die Modelle von Maximum-Likelihood- und Quader-Verfahren beinhalten ebenfalls die Einstellung des Parameters `info`, mit dem während der Vorhersage Informationen über deren Fortschritt ausgegeben werden können. Die drei genannten Modellfunktionen werden nun mit der Funktion `trainLearner()` eingebunden. Durch diese werden bei

Verwendung des anzugebenden Learners verschiedene Vorgänge initiert. Zunächst wird angegeben, wie die Daten des verwendeten Tasks in die für die Modellfunktion passende Form umgewandelt werden. Mit der Funktion `getTaskData()` können dabei die Daten aus dem Task extrahiert werden. Hierbei kann mit dem Parameter `subset` auch angegeben werden, ob nur ein Teil der Daten (zum Beispiel als Trainingsdaten) im Folgenden zur Erstellung des Modells genutzt werden. Das Modell selbst wird dann mit dem Aufruf der Modellfunktion erzeugt. Dabei müssen alle Parameter angegeben werden, damit diese später innerhalb der Learner-Umgebung benutzt werden können. Die Einbindungen der Modelle für Minimum-Distance-, Maximum-Likelihood- und Quader-Verfahren sind in Kapitel B.2 des Anhangs ab den Seiten 114, 118 beziehungsweise 126 unter dem Begriff "Einbindung in `mlr`" zu finden. Dort ist ebenfalls zu sehen, wie die Basis der Learner erstellt wird und wie die Vorhersagefunktionen in den `mlr`-Kontext integriert werden. Die Angabe der Vorhersagefunktion ist der dritte und letzte Teil der Einbindung eines Verfahrens und wird mit der Funktion `predictLearner()` durchgeführt. Die Vorhersagefunktion des Verfahrens wird hier unter Angabe des Modells und der zu klassifizierenden Daten aufgerufen. Die (Test-) Daten können dabei aus einem neuen Data Frame `.newdata` stammen oder werden alternativ unter Angabe eines Vektors mit Objektnummern aus den Daten des Tasks gewonnen. Als Resultat wird eine Zusammenfassung der Ergebnisse ausgegeben. Diese enthält für die ersten sechs Objekte die Vorhersagen $\hat{y}_1, \ldots, \hat{y}_6$ sowie die Werte $y_1, \ldots, y_6$ der echten Klassen. Als zusätzliche Information werden die Anzahl der Testobjekte, der Vorhersagetyp und die zur Erstellung der Vorhersagen benötigte Zeit angezeigt. Die Einbindung der Vorhersagefunktionen findet für Minimum-Distance-, Maximum-Likelihood- und Quader-Verfahren in analoger oben dargestellter Form statt. Die drei Verfahren sind nun unter den Bezeichnungen "classif.mindist", "classif.maxlike" und "classif.quader" in `mlr` eingebunden. Sie können auf die gleiche Weise wie "classif.lda", "classif.randomForest" und "classif.svm" verwendet werden.

Um ein Klassifikationsverfahren in `mlr` durchzuführen, können nun die Funktionen `train()` und `predict()` genutzt werden. Mit `train` wird das Modell, also die Entscheidungsregel für die Vorhersage, erstellt. Es müssen ein Learner und ein Task angegeben werden. Mit dem Parameter `subset` können die zur Erstellung des Modells gewählten Objekte eingeschränkt werden. Ansonsten werden alle Datenpunkte des Tasks verwendet. Die Funktion `predict()` aus dem Paket `stats` (siehe R Core Team

(2016)) kann in `R` für viele verschiedene Methoden und Verfahren genutzt werden. Sie greift anhand des Eingabeformats auf diverse Vorhersagefunktionen auch anderer Pakete zu und führt diese aus. Hier wird die Funktion `predict.WrappedModel()` aus `mlr` aufgerufen. Sie benötigt als Eingabe ein durch `train` erzeugtes Modell sowie die Daten, für die Vorhersagen erstellt werden sollen. Entweder es wird ein Task angegeben um die darin enthaltenen Daten (oder einen Teil davon) vorherzusagen oder es werden neue Daten über den Parameter `newdata` eingegeben. Im Folgenden wird immer einer der Tasks KM, CL, PH oder BC angegeben und mit dem Parameter `subset` werden die Segmente eines Viertels des Satellitenbildes als Testdaten ausgewählt. Da die Klassenzugehörigkeit für die Testdaten vorliegt, kann nun mit den Vorhersagen der `predict`-Funktion die Fehlklassifikationsrate berechnet werden. Sie wird hierbei auch schon als mittlere Fehlklassifikationsrate (mmce) bezeichnet. Bei nur einer Rate kann diese auch als Mittelwert bezüglich sich selbst interpretiert werden. Die mmce kann mit der Funktion `performance()` bestimmt werden. Dazu wird einfach das Ergebnis der Funktion `predict()` von oben eingegeben, da die Vorhersagen und die echten Klassen hierin enthalten sind. Es können auch andere Maße für die Bewertung der Klassifikation gewählt werden, die mmce ist jedoch voreingestellt. Die wichtigsten Vorarbeiten zur Nutzung von `mlr` (Teilziel 2a)) sind damit abgeschlossen. Weitere Funktionen des Paketes werden an den entsprechenden Stellen erläutert.

Um mehrere Verfahren beziehungsweise die Klassifikation mehrerer Datensätze gleichzeitig durchzuführen kann die Funktion `benchmark()` verwendet werden. Die Verfahren werden in Form einer Liste mit den entsprechenden Learnern angegeben, während die Datensätze als Liste mit entsprechenden Tasks eingegeben werden. Der durch `benchmark` erzeugte "Benchmark" kombiniert jeden Learner mit jedem Task. Mit jeder Kombination werden dann die Funktionen `train()`, `predict()` und `performance()` ausgeführt. Es wird also für jedes Learner-Task-Paar eine mmce berechnet. Dies ermöglicht einen Vergleich sowohl der Verfahren als auch der Datensätze. Um die mmce bestimmen zu können, müssen Teile des Datensatzes als Trainingsdaten und anderen Teile als Testdaten gekennzeichnet werden. Im `benchmark` wird dazu über den Parameter `resamplings` eine "Resampling Description" angegeben. Normalerweise wird diese mit der Funktion `makeResampleDesc()` erzeugt. Mit der Parameterwahl `method="CV"` wird beispielsweise eine 10-fache Kreuzvalidierung durchgeführt. Der Datensatz wird dabei in zehn etwa gleich große Teile

aufgeteilt und jeweils einer der Teile wird als Testdaten genutzt, während aus den anderen Teilen das Modell erstellt wird. Jede Learner-Task-Kombination wird also insgesamt zehn mal verwendet. Die zehn errechneten mmce der Durchläufe werden gemittelt, sodass für jede Kombination letztlich ein Wert vorliegt. Die zehn Teile des Datensatzes werden ganz zu Beginn zufällig aus den Daten ermittelt und dann als "Resample Instance" gespeichert. Diese wird bei Verwendung des Datensatzes im Benchmark genutzt. Der Learner ist dabei beliebig, sodass alle Verfahren die gleichen Voraussetzungen haben. Die vier vorhandenen Datensätze sollen hier aber nicht zufällig aufgeteilt werden. Die vorliegenden Segmente würden dann wie bereits erwähnt unterschiedlich große Mengen von Pixeln repräsentieren. Dadurch könnte die Kreuzvalidierung ihre Wirkung verfehlen und es könnte zu einer Überanpassung der Klassifikationsregeln an die Daten kommen. Statt einer 10-fachen soll im Folgenden auch eine 4-fache Kreuzvalidierung durchgeführt werden. Die Vorbereitung hierzu wurden bereits getroffen, indem das Satellitenbild geviertelt wurde. Für jedes Viertel wurden die Segmente einzeln bestimmt und diese so in den Datensätzen angeordnet, dass deren Position bekannt ist. Die Segmente jedes Viertels werden nun einmal als Testdaten genutzt, während die Segmente der anderen drei Viertel zum Trainieren verwendet werden. Zur Festlegung konkreter Objekte als Trainings- und Testdaten wird daher die Funktion `makeFixedHoldoutInstance()` genutzt. Dazu werden die Nummern der Trainingsobjekte (`train.inds`), der Testobjekte (`test.inds`) und die Größe des Datensatzes (`size`) angegeben, womit dann eine entsprechende Resampling Instance erzeugt wird. Als Beispiel wird für den Datensatz KM und das Viertel ol `test.inds=1:40966` gewählt, da die ersten 40966 Segmente in KM aus dem Viertel ol stammen. Die Segmente der anderen drei Viertel werden mit `train.inds=40967:130525` als Trainingsdaten festgelegt. Die Größe des Datensatzes ist `size=130525`. Resampling Instances für die Viertel or, ul und ur ergeben sich in analoger Weise. Die Funktion `benchmark()` erlaubt leider nicht mehrere Resampling Instances bei Eingabe nur eines Tasks. Daher wird jeder Task viermal, jeweils mit der Resampling Instance zu einem Viertel, verwendet. Aus den resultierenden vier mmce wird dann per Hand das arithmetische Mittel bestimmt und dem verwendeten Learner-Task-Paar zugewiesen. Damit wurde die 4-fach Kreuzvalidierung bei diesem Paar abgeschlossen. Zur Berechnung der mmce aller Learner-Task-Paare wurden Benchmarks getrennt nach Vierteln berechnet und deren Ergebnisse kombiniert, da dies in `R` als einfachste Art der Durchführung erschien. Die mmce

der einzelnen Paare sind, auf vier Nachkommastellen gerundet, in Tabelle 7 dargestellt. Die verwendeten Klassifikationsverfahren sind hierbei die lineare Diskrimi-

Tabelle 7: Mittlere Fehlklassifikationsraten nach Art der Segmentierung und Klassifikationsverfahren

	KM	CL	PH	BC
LDA	0.4710	0.4753	0.5453	0.5421
Random Forest	0.4530	0.4604	0.5041	0.5080
SVM	0.4477	0.4527	0.5028	0.5036
MinDist	0.8523	0.8608	0.8844	0.8796
MaxLike	0.5559	0.7209	0.8972	0.7595
Quader	0.8449	0.8427	0.8763	0.8697

nanzanalyse (LDA), der Random Forest, die Support Vector Machine (SVM) sowie das Minimum-Distance- (MinDist), Maximum-Likelihood- (MaxLike) und Quader-Verfahren. Alle Verfahren werden zunächst mit ihren Standardeinstellungen verwendet. Die lineare Diskriminanzanalyse besitzt keine nennenswerten Parameter. Für den Random Forest werden `ntree=500` Bäume genutzt. Die Parameter der SVM sind als $C_{SVM} = 1$ und $\gamma = \frac{1}{v}$ vorgegeben, wobei hier $v = 11$ Variablen im Datensatz vorliegen. Beim Minimum-Distance-Verfahren gibt `empty.class="infty"` an, dass eine nicht in den Trainingsdaten vorhandene Klasse als unendlich weit entfernt angesehen wird. Eine solche Klasse kann also auch nicht vorhergesagt werden. In ähnlicher Weise setzt `empty.class="null"` im Maximum-Likelihood-Verfahren die Wahrscheinlichkeit nicht vorhandener Klassen auf 0. Diese Klassen können also ebenfalls praktisch nicht vorhergesagt werden. Die Voreinstellung `empty.class="none"` des Quader-Verfahrens vermeidet ebenfalls die Vorhersage nicht vorhandener Klassen. Zusätzlich werden mit `boxmethod="range"` die Grenzen der Quader als minimaler und maximaler Wert jeder Klasse in den einzelnen Dimensionen festgelegt. Für den Fall, dass ein Objekt nicht in genau einem Quader liegt, wird mit `tiemethod="mind"` angegeben, dass die Vorhersage der Klasse durch das Minimum-Distance-Verfahren bestimmt wird. Liegt das Objekt in mehreren Quadern, wird das Minimum-Distance-Verfahren nur mit diesen Klassen durchgeführt. Ist das Objekt hingegen in keinem Quader, werden alle Klassen, auch nicht vorhandene, verwendet.

Die mmce aus Tabelle 7 werden nun zum Vergleich und zur Bewertung der Verfahren genutzt. Der niedrigste Werte (0.4477) resultiert aus der Support Vector Machine bei Verwendung der Segmentierung anhand des k-means-Algorithmus. Die höchste mmce wurde mit 0.8972 für das Maximum-Likelihood-Verfahren und den mit partiellem hierarchischem agglomerativem Clustern erzeugten Datensatz berechnet. Die Unterschiede zwischen dem minimalen und maximalen Wert sind groß. Dies spiegelt die unterschiedliche Vorgehensweise der Verfahren wieder. Generell sind die Werte auf einem hohen Niveau. Die Vorhersage der Bodenbedeckungsklasse ist mit allen verwendeten Verfahren recht schwierig. Selbst bei der besten Verfahren-Datensatz-Kombination sind im Schnitt immer noch etwa 9 von 20 Vorhersagen falsch. Dies wird etwas relativiert durch die Tatsache, dass 18 mögliche Klassen zur Verfügung stehen. Zudem sind die Bodenbedeckungsklassen aus den CORINE-Daten und das Satellitenbild zu unterschiedlichen Zeitpunkten erstellt worden. Dennoch wären bessere Ergebnisse wünschenswert. Ein Vergleich der Verfahren über alle vier Datensätze hinweg zeigt, dass die Support Vector Machine insgesamt am besten klassifiziert. Bei der SVM werden jeweils die kleinsten mmce pro Datensatz berechnet, womit hier die wenigsten falschen Vorhersagen erzeugt wurden. Die Werte des Random Forest sind insgesamt nur wenig größer als bei der SVM. Zusammen mit der Tatsache, dass die Berechnung des Random Forest in der Regel schneller erfolgt, können beide Verfahren hier als ebenbürtig betrachtet werden. Mit kleinem aber deutlich erkennbarem Abstand sind die drittkleinsten mmce jeweils bei der linearen Diskriminanzanalyse zu finden. Da deren Durchführung in `R` nochmals weniger Zeit in Anspruch nimmt als die des Random Forest, wäre auch die LDA wegen des kleinen Abstands eine gute Alternative. Die drei am besten bewerteten Verfahren sind also die, die typischerweise in der Statistik verwendet werden. Die Reihenfolge der Verfahren ist dabei sehr eindeutig. Bei den drei typischen Verfahren der Fernerkundung ist dies nur begrenzt der Fall. Das Quader-Verfahren hat hier gegenüber dem Minimum-Distance-Verfahren die etwas kleineren Werte, sodass zwischen diesen Beiden das Quader-Verfahren vorzuziehen ist. Die Ähnlichkeit der beiden Verfahren lässt sich vermutlich über die Vorgehensweise des Quader-Verfahrens erklären. Liegt ein Objekt nicht in genau einem Quader, wird das Minimum-Distance-Verfahren zur Bestimmung der Vorhersage weiterverwendet. Beim Datensatz PH liegen beispielsweise in allen Vierteln ingesamt nur 45 von 45406 zu klassifizierenden Objekten in exakt einem Quader. Andererseits liegen nur 21 Objekte in keinem Quader, sodass

die verfügbaren Klassen bei Anwendung des Minimum-Distance-Verfahrens zumeist eingeschränkt sind. Dies könnte den kleinen Abstand von weniger als einem Prozentpunkt zwischen den beiden Verfahren erklären. Die Bewertung des Maximum-Likelihood-Verfahrens ist schwierig, da die mmce je nach Datensatz sehr unterschiedlich sind. Der insgesamt höchste Wert findet sich beim Datensatz PH. Bei den Datensätzen CL und BC sind die Werte auf einem mittleren Niveau, während die mmce bei KM, im Verhältnis zu den übrigen drei Werten, sehr niedrig ist. Falls die Segmente nicht mit partiellem hierarchischem agglomerativem Clustern erstellt werden, scheint das Maximum-Likelihood- also besser als das Quader-Verfahren zu sein. Insgesamt gesehen, sollte zur Klassifikation der Bodenbedeckungen aber eines der "Statistikverfahren" verwendet werden. Zuletzt lassen sich auch die Datensätze untereinander vergleichen. Für den Datensatz KM ergeben sich die jeweils kleinsten Werte. Eine Ausnahme bildet nur das Quader-Verfahren, bei dem der vorliegende zweitkleinste Wert aber nur geringfügig vom kleinsten Wert abweicht. Die durch den k-means-Algorithmus gebildeten Segmente scheinen also schlecht für die Nachbildung der Bodenbedeckungsstruktur (größte NIBS-Distanz), dafür aber bei der Klassifikation umso besser verwendbar zu sein. Die Werte der anderen Datensätze sind je nach Verfahren unterschiedlich. Bei den drei "Fernerkundungsverfahren" sind die mmce aus CL, BC und PH etwa auf gleichem Niveau. Davon weicht nur das Maximum-Likelihood-Verfahren ab, welches bei PH eine deutlich höhere mmce erzeugt. Die mittleren Fehlklassifikationsraten der drei Statistikverfahren wiederum sind bei BC und PH ähnlich. Die Werte für CL sind erkennbar tiefer und fast so gut wie bei KM. Für die Klassifikation ist insgesamt die Segmentierung mittels des k-means-Algorithmus am besten geeignet.

Details oder Besonderheiten, wie die Werte des Maximum-Likelihood-Verfahrens, können mit den bisher aufgeführten mmce nicht erklärt werden. Eine detailiertere Auswertung ist aber durch die mmce aus den einzelnen Vierteln vor deren Mittellung möglich. Die entsprechenden Werte, allerdings der Übersichtlichkeit halber nach Clusterverfahren sortiert, sind in Tabelle 8 zu finden. Dabei wurden die Segmente des angegebenen Viertels jeweils als Testdaten genutzt. Die Unterschiedlichkeit der mmce des Maximum-Likelihood-Verfahrens in Tabelle 7 fällt auch bei der Betrachtung der Werte in den einzelnen Vierteln auf. Die Werte innerhalb der Datensätze sind hier sehr verschieden. Beispielsweise findet sich für CL der kleinste Wert mit 0.56 in Viertel ul, während der größte Wert 0.7997 zum Viertel or gehört.

Tabelle 8: Mittlere Fehlklassifikationsraten nach Vierteln getrennt

k-means (KM)	ol	or	ul	ur
LDA	0.4741	0.4561	0.4799	0.4740
Random Forest	0.4537	0.4307	0.4626	0.4650
SVM	0.4483	0.4345	0.4607	0.4475
MinDist	0.8373	0.8725	0.8726	0.8267
MaxLike	0.6155	0.4873	0.5493	0.5713
Quader	0.8277	0.8649	0.8702	0.8168
clara (CL)	ol	or	ul	ur
LDA	0.4829	0.4665	0.4911	0.4608
Random Forest	0.4685	0.4352	0.4852	0.4528
SVM	0.4603	0.4386	0.4769	0.4349
MinDist	0.8641	0.8583	0.8623	0.8585
MaxLike	0.7661	0.7997	0.5600	0.7579
Quader	0.8286	0.8454	0.8500	0.8466
phclust (PH)	ol	or	ul	ur
LDA	0.5553	0.5436	0.5549	0.5274
Random Forest	0.5107	0.4892	0.5292	0.4873
SVM	0.5056	0.5003	0.5362	0.4692
MinDist	0.8208	0.9154	0.8942	0.9072
MaxLike	0.9198	0.8212	0.9458	0.9022
Quader	0.8108	0.9094	0.8910	0.8938
bagclust (BC)	ol	or	ul	ur
LDA	0.5433	0.4966	0.5882	0.5401
Random Forest	0.5062	0.4710	0.5466	0.5083
SVM	0.5113	0.4683	0.5462	0.4883
MinDist	0.8446	0.8360	0.9116	0.9263
MaxLike	0.7601	0.7527	0.8096	0.7157
Quader	0.8237	0.8297	0.9039	0.9215

Die Richtigkeit der Vorhersagen ist also mit den Testdaten aus ul um zirka 24 Prozentpunkte höher als bei Verwendung der Segmente aus or. Die Aussage, dass sich die Segmente in or mit denen der anderen Viertel schlecht erklären lassen, stimmt aber auch nicht. Das Viertel or erzielt pro Datensatz gesehen sogar in zwei Fällen (KM und PH) den besten Wert im Vergleich zu den anderen Vierteln. Darunter ist auch der beste Wert des Maximum-Likelihood-Verfahrens über alle Viertel und alle Datensätze. Ein gegenteiliges Phänomen taucht bei ul auf. Dort stehen dem für CL mit Abstand besten Wert von 0.56 die jeweils schlechtesten Werte bezüglich PH und BC gegenüber. Darunter ist mit 0.9458 auch der insgesamt schlechteste Wert. Auffällig ist weiterhin auch, dass sich die besten und schlechtesten Werte pro Datensatz auf jeweils drei Viertel aufteilen. Bei den anderen Verfahren sind meist nur ein bis zwei Viertel gut und dieselbe Menge schlecht klassifizierbar. Im Fall der Support Vector Machine sind die schlechtesten Werte sogar immer im gleichen Viertel ul zu finden, egal welcher Datensatz verwendet wurde. Auch insgesamt gesehen ist über die Verfahren hinweg eine Tendenz für die Viertel zu erkennen. Die Viertel or und ur lassen sich gut vorhersagen. Hier liegt über die Verfahren und Datensätze hinweg 11 beziehungsweise 8 mal die niedrigste mmce vor. Besonders schlecht lässt sich hingegen die Bodenbedeckung in ul anhand der drei anderen Viertel beschreiben. In ul wurde insgesamt 15 mal die höchste mmce erreicht. Die Datensätze und damit die Clusterverfahren mit denen die Segmente erzeugt wurden sind je nach Klassifikationsverfahren unterschiedlich wichtig. Beim Maximum-Likelihood-Verfahren ist die Relevanz für die Ergebnisse der einzelnen Viertel sehr groß. Dies trifft in kleinerem Ausmaß auch auf das Minimum-Distance-Verfahren zu. Vor allem bei der Support Vector Machine aber auch bei den anderen Verfahren ist die Relevanz hingegen eher gering. Teilziel 2b) ist nun erreicht.

Die oben aufgeführten Ergebnisse können unter Umständen noch verbessert werden, indem die Parameter der Verfahren variiert werden. Dieses Parametertuning ist nicht bei allen Verfahren möglich. Bei der linearen Diskriminanzanalyse liegt kein Parameter vor, der hier sinnvoll getuned werden kann. Die anderen Verfahren besitzen alle mindestens einen sinnvoll variierbaren Parameter. Mit der Funktion `tuneParams()` kann das Parametertuning dieser Verfahren in `R` ausgeführt werden. Es müssen ein Learner, ein Task und eine Resampling Instance eingegeben werden. Für jede Learner-Task-Kombination wird also für jedes einzelne Viertel wieder die mmce berechnet. Die vier Werte der Viertel, die aus den gleichen Verfahrenspara-

metern resultieren, werden dann gemittelt und ergeben eine mmce für die verwendete Parameterkombination. In `tuneParams` wird des Weiteren über den Parameter `control` die Art des Tunings angegeben. Hier wird mit einer Gittersuche nach einer optimalen Parameterkombination gesucht. Ein Gitter mit Parameterkombinationen wird mit der Funktion `makeTuneControlGrid()` erzeugt, indem diese bei `control` angegeben wird. Die Funktion benötigt keine Eingaben. Sie greift automatisch auf den Parameter `par.set` aus `tuneParams` zu. In diesem werden die verschiedenen Ausprägungen der Parameter des Verfahrens in Form eines Parameter-Set vorgegeben. Das Set kann, ähnlich wie bei der Einbindung neuer Verfahren, mit der Funktion `makeParamSet()` erstellt werden. Um ein Gitter herzustellen, werden nur diskrete Parameter mit `makeDiscreteParam()` erzeugt, selbst wenn der Parameter nicht diskret ist. Über `values` kann eine endliche Menge von Parameterwerten vorgegeben werden, mit denen dann getuned wird.

Beim Random Forest wird nur der Parameter `ntree`, welcher die Anzahl der verwendeten Bäume angibt, zum Tunen genutzt. Es entsteht also kein wirkliches Gitter, da keine weiteren Parameter verwendet werden. Das Verfahren wird schlicht mehrfach mit den gewählten Anzahlen an Bäumen durchgeführt. Als Anzahl werden hier 1, 2, 5, 10, 25, 50, 100, 250 und 500 Bäume gewählt. Das Parameter-Set hat entsprechend die Form `makeParamSet(makeDiscreteParam("ntree", values = c(1,2,5,10, 25,50,100,250,500,1000)))`. Das Tuning der Baumanzahl bei den vier Datensätzen führt zu den in Tabelle 9 dargestellten und auf vier Nachkommastellen gerundeten mittleren Fehlklassifikationsraten. Es ist erkennbar, dass die mmce mit steigender Baumzahl sinken. Die Verbesserungen sind ab zirka 100 Bäumen aber nur noch geringfügig. Die besten Ergebnisse werden bei allen Datensätzen durch die Standardwahl von 500 Bäumen erreicht. Ein Random Forest mit einer höheren Anzahl Bäumen war nicht durchführbar, da der Arbeitsspeicher des ausführenden Computers zu klein ist. Da die zu erwartende Verbesserung aber nur gering ist, scheinen 500 Bäume ausreichend zu sein. Insgesamt wurden also durch das Parametertuning beim Random Forest keine Verbesserungen erzielt.

Für das Tunen der Support Vector Machine werden die Parameter C_{SVM} und γ verändert. Der Parameter C_{SVM} reguliert die Schlupfvariablen, während γ die Form der Radial-Basis-Kernfunktion beeinflusst. Die Berechnung der SVM ist in `R` zeitintensiver als bei allen anderen Verfahren. Die Parameterwerte für das Tuning sollten also besonders sorgfältig gewählt werden. In der Literatur werden vielfach Poten-

Tabelle 9: Ergebnisse des Parametertunings beim Random Forest

`ntree`	KM	CL	PH	BC
1	0.6097	0.6151	0.6677	0.6571
2	0.6062	0.6142	0.6666	0.6586
5	0.5196	0.5270	0.5813	0.5773
10	0.4878	0.4931	0.5473	0.5419
25	0.4656	0.4735	0.5196	0.5225
50	0.4598	0.4664	0.5121	0.5142
100	0.4558	0.4635	0.5085	0.5098
250	0.4541	0.4614	0.5055	0.5082
500	0.4530	0.4604	0.5041	0.5080

zen mit der Basis 2 verwendet (vergleiche beispielsweise Hsu et al. (2003, S. 5)). Die Standardeinstellungen der ausführenden Funktion `svm()` sind $C_{SVM} = 1$ und $\gamma = \frac{1}{11}$. Daher werden für die Erstellung des Gitters die nächstgelegenen niedrigeren und höheren Zweierpotenzen zu jedem Parameter verwendet. Für C_{SVM} führt dies zu den Werten $\frac{1}{2}$, 1 und 2. Die Werte von γ werden als $2^{-4} = \frac{1}{16}$, $\frac{1}{11}$ und $2^{-3} = \frac{1}{8}$ gewählt. Die Ergebnisse für jede Parameterkombination in Form der mmce sind in Tabelle 10 zu finden. Die Entwicklung der mmce ähnelt sich bei den vier

Tabelle 10: Ergebnisse des Parametertunings bei der Support Vector Machine

	KM	C_{SVM}					C_{SVM}		
		$\frac{1}{2}$	1	2		**CL**	$\frac{1}{2}$	1	2
γ	$\frac{1}{16}$	0.4511	0.4489	0.4474	γ	$\frac{1}{16}$	0.4567	0.4537	0.4519
	$\frac{1}{11}$	0.4499	0.4477	0.4462		$\frac{1}{11}$	0.4550	0.4527	0.4512
	$\frac{1}{8}$	0.4489	0.4466	0.4456		$\frac{1}{8}$	0.4544	0.4523	0.4511
	PH	C_{SVM}					C_{SVM}		
		$\frac{1}{2}$	1	2		**BC**	$\frac{1}{2}$	1	2
γ	$\frac{1}{16}$	0.5114	0.5062	0.5014	γ	$\frac{1}{16}$	0.5104	0.5063	0.5026
	$\frac{1}{11}$	0.5075	0.5028	0.4996		$\frac{1}{11}$	0.5084	0.5036	0.4994
	$\frac{1}{8}$	0.5047	0.5016	0.4980		$\frac{1}{8}$	0.5059	0.5009	0.4979

Datensätzen. Steigen die Werte von C_{SVM} oder γ, sinkt die mittlere Fehlklassifikati-

onsrate. Diese Verbesserung ist besonders stark, wenn beide Parameter gleichzeitig erhöht werden. Der gegenteilige Effekt ist bei Absinken eines oder beider Parameter zu beobachten. Ein höherer Wert von C_{SVM} führt in der Regel zu einem kleineren Margin, da mehr Datenpunkte innerhalb oder auf der "falschen" Seite liegen dürfen. Durch die Erhöhung von γ werden die Entscheidungsgrenzen um die Datenpunkte einer Klasse herum unregelmäßiger und weniger glatt, dadurch aber auch flexibler. Beide Parameter sollten nicht gleichzeitig zu stark erhöht werden, da dies zu einer Überanpassung an die Daten ("Overfitting") führen kann. Hier kann aber wegen der ausgeführten Kreuzvalidierung davon ausgegangen werden, dass noch kein Overfitting vorliegt. Es wird daher zur Verwendung einer Support Vector Machine mit den Parametern $C_{SVM} = 2$ und $\gamma = \frac{1}{8}$ geraten. Bei zwei Parametern können die Tuningergebnisse auch gut in einer Contour-Grafik, wie Abbildung 15, visualisiert werden. Die mmce bei gewählten Parametereinstellungen werden hierbei durch Höhenlinien

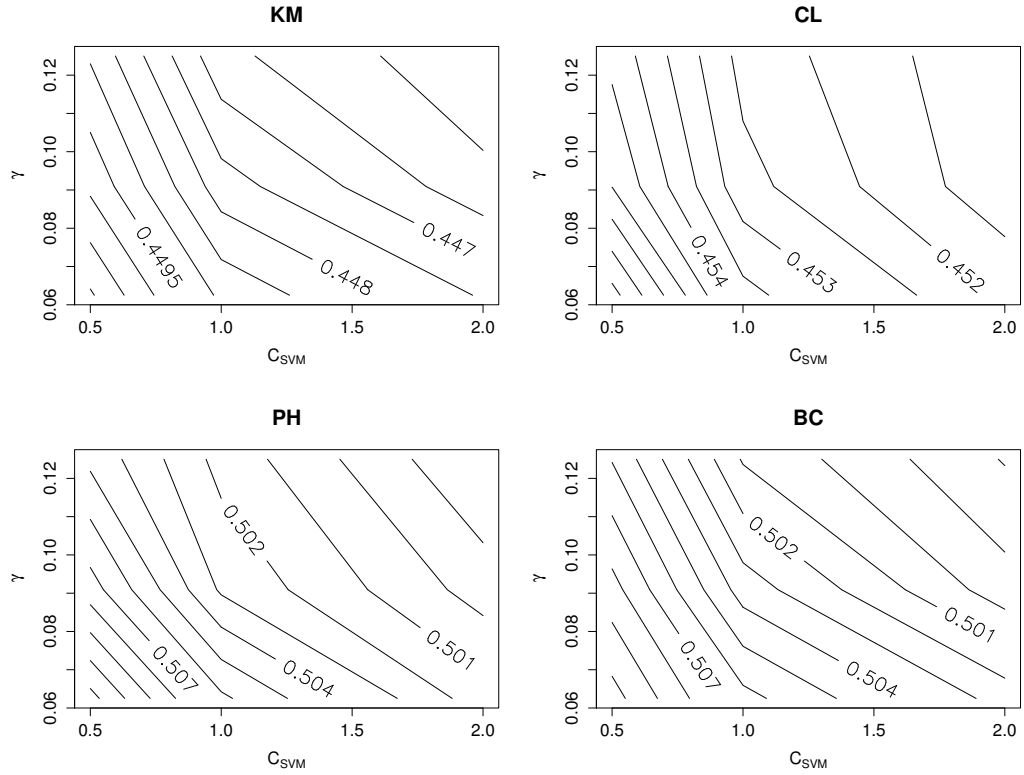


Abbildung 15: Contour-Grafik für die Ergebnisse des Parametertunings der Support Vector Machine

dargestellt. Die Zwischenräume werden approximiert. Anhand der Grafik wird deutlich, dass die mmce bei gleicher Variation der Parameter in den Datensätzen PH und BC schneller absinkt als bei KM und CL. Die Art der Segmentierung wird also für

die Klassifikation weniger relevant. Dieser interessante Effekt erscheint auch beim Parametertuning anderer Verfahren.
Die obige Darstellungsform wird hier auch zur detaillierten Analyse der Ergebnisse in den einzelnen Vierteln genutzt. In Abbildung 16 für den Datensatz KM ist die Entwicklung der mmce der einzelnen Viertel zu erkennen. Die Viertel ol, or und ur

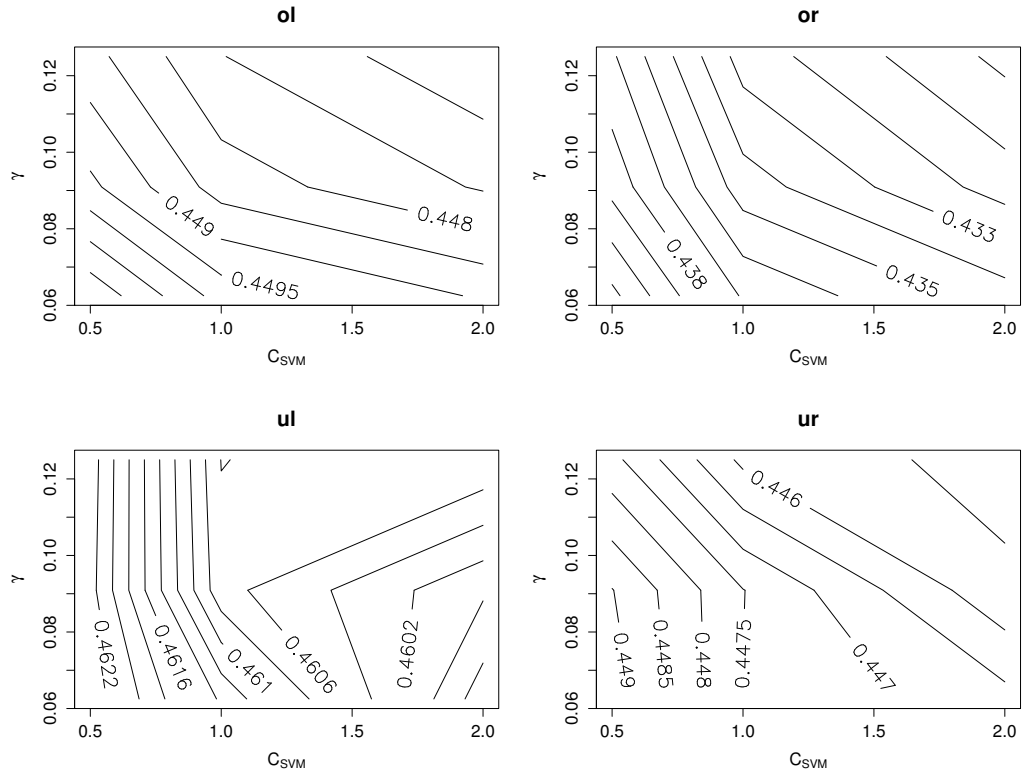


Abbildung 16: Contour-Grafik für die Ergebnisse des Parametertunings der Support Vector Machine bei Verwendung des Datensatzes KM

unterscheiden sich prinzipiell nicht wesentlich von den Grafiken für die gesamten Datensätze in Abbildung 15. Bei ul gibt es hingegen Abweichungen. Die mmce sinkt weiterhin bei steigendem C_{SVM}. Hat dieser Parameter den Wert 2 sinkt die mmce jedoch mit fallendem γ. Die beste Parameterkombination scheint hier $C_{SVM} = 2$ und $\gamma = \frac{1}{16}$ zu sein. Die Unterschiede der mmce in ul sind aber gegenüber den anderen Vierteln eher gering. Die beschriebene Tendenz wird daher durch die Bildung der Mittelwerte sehr stark abgeschwächt. Die Grafik der Viertel zum Datensatz CL (Abbildung 34 in Kapitel A des Anhangs) weist eine ähnliche Tendenz wie oben dargestellt auf. Die Klassifikation ist, wie bereits beschrieben, im Viertel ul am schwierigsten. Eine detailliertere Untersuchung könnte darüber Aufschluss geben, ob dies nicht in Form einer Gewichtung während des Tunings eingehen sollte. Aus bereits ge-

nannten zeitlichen Gründen wird hier aber darauf verzichtet. Die Contour-Grafiken der Viertel für die Datensätze PH und BC (Abbildungen 35 und 36 in Kapitel A des Anhangs) weisen überdies keine so starke Abweichung wie hier auf.
Das Minimum-Distance-Verfahren wird über den Parameter `empty.class` getuned. Dieser gibt die Vorgehensweise an, falls eine der Klassen nicht in den Trainingsdaten auftaucht. Die Standardwahl ist `"infty"`, wodurch die Distanz der nicht erscheinenden Klassen zu allen möglichen Objekten auf unendlich festgelegt wird. Die betroffenen Klassen können also nicht vorhergesagt werden, da eine der Distanzen zu den anderen Klassen immer kleiner ist. Als alternative Wahl des Parameters kann `"overall"` verwendet werden. Hier wird das arithmetische Mittel über alle Trainingsdatenpunkte als Zentroid für die nicht vorhandenen Klassen verwendet. Gibt es mehrere dieser "leeren" Klassen sind deren Zentroide natürlich identisch. Weist ein Objekt die kleinste Distanz zu diesem Gesamtmittelpunkt der Trainingsdaten auf, wird zufällig eine der leeren Klassen zugewiesen. Die Veränderung von `empty.class` hat nur Einfluss, falls die Segmente des oberen linken Viertels als Testdaten verwendet werden. Nur hier gibt es nämlich Segmente mit den Klassen C_5 und C_{18}. Folglich sind diese Klassen in den anderen drei zum Trainieren gewählten Vierteln nicht enthalten. Bei allen weiteren Konstellationen gibt es immer mindestens ein Segment jeder Klasse in den Trainingsdaten. Die berechneten mmce sind dort also identisch zu denen bei Verwendung von `"infty"`. Die gemittelten mmce für einen Datensatz ändern sich dadurch nur wenig. Die entsprechenden Werte sind in Tabelle 11 zu finden. Die mmce bei Verwendung von `"overall"` sind bei allen vier Datensätzen

Tabelle 11: Ergebnisse des Parametertunings beim Minimum-Distance-Verfahren

`empty.class`	KM	CL	PH	BC
`"infty"`	0.8523	0.8608	0.8844	0.8796
`"overall"`	0.8531	0.8614	0.8847	0.8798

geringfügig höher. Wird nur die mmce mit ol als Testdaten betrachtet, sind die Unterschiede etwas deutlicher. Die Werte von `"overall"` liegen in diesem Fall für die Datensätze KM, CL, PH und BC um zirka 0.0032, 0.0023, 0.0013 und 0.0005 höher als bei Wahl von `"infty"`. Die Unterschiede sind insgesamt aber sehr gering und das Tuning führt nur zu leicht schlechteren mmce als mit den Standardeinstellungen. Es sollte daher die Zielsetzung der Untersuchung genau beachtet werden. Geht

es speziell darum auch kleine wenig vorhandene Klassen in einem Satellitenbild zu finden, ist `"overall"` vorzuziehen. Wenn das Verfahren mit den wenigsten Fehlern gesucht wird, sollte die Wahl auf `"infty"` fallen.

Beim Maximum-Likelihood-Verfahren wird in ähnlicher Weise wie beim Minimum-Distance-Verfahren nur der Parameter `empty.class` variiert. Die Einstellung `"null"` setzt die Wahrscheinlichkeiten für nicht vorhandene (leere) Klassen auf 0. Bei der Alternative `"overall"` werden das arithmetische Mittel und die empirische Kovarianzmatrix aller Trainingsdaten berechnet. Mit diesen Schätzungen für den Mittelwert und die Kovarianzmatrix werden dann die Wahrscheinlichkeiten der multivariaten Normalverteilung für die leeren Klassen ermittelt. Bei mehreren leeren Klassen sind die Wahrscheinlichkeiten für diese dann natürlich identisch. Sollte eine dieser Wahscheinlichkeiten das Maximum für einen Datenpunkt bilden, wird zufällig eine der leeren Klassen vorhergesagt. Im Ergebnis sind die mmce beider Einstellungen gleich. Die Streuung der Pixel in den gesamten Trainingsdaten ist wesentlich höher als in nur einer Klasse. Bei Wahl von `"overall"` verteilt sich die Wahrscheinlichkeitsmasse in den leeren Klassen auf viele Punkte. Die einzelnen Wahrscheinlichkeiten sind entsprechend niedrig und und stellen selten oder nie das Maximum über alle Klassen dar. Insgesamt liegt also keine Veränderung und somit auch keine Verbesserung durch das Parametertuning vor.

Auch beim Quader-Verfahren existiert der Parameter `empty.class`. Er hat mit den Ausprägungen `"none"` und `"all"` die gleichen Auswirkungen wie schon beim Minimum-Distance- und Maximum-Likelihood-Verfahren. Mit `"all"` werden das arithmetische Mittel und die emprische Kovarianzmatrix der gesamten Trainingsdaten für die leeren Klassen im Modell hinterlegt. Ein Quader dieser Klassen wird als gesamter Messraum festgelegt, sodass jeder mögliche Datenpunkt innerhalb des Quaders ist. Die genauen Auswirkungen von `"none"` sind abhängig vom weiteren Parameter `tiemethod`. Dieser gibt an, welches Verfahren zur Vorhersage der Klasse genutzt werden soll, wenn ein Objekt nicht in genau einem Quader ist. Mit der Voreinstellung `tiemethod="mind"` wird das Minimum-Distance-Verfahren verwendet. `empty.class` funktioniert hier genau wie bei diesem Verfahren beschrieben. Bei Verwendung von `tiemethod="ml"` werden die leeren Klassen wie beim Maximum-Likelihood-Verfahren beschrieben gehandhabt. Die Wahlmöglichkeiten `"infty"`, `"null"` und `"none"` sowie `"overall"` und `"all"` entsprechen sich dabei jeweils. Der Parameter `tiemethod` mit seinen zwei Optionen wird ebenfalls zum Tunen verwen-

det. Als dritter Tuningparameter wird `boxmethod` genutzt. Mit dem voreingestellten Wert `"range"` werden die Grenzen der Quader als minimaler und maximaler Wert einer Klasse in den verschiedenen Dimensionen bestimmt. Mit der Wahl von `"sigma"` wird zum arithmetischen Mittel der Klasse ein Vielfaches der empirischen Standardabweichungen in den Dimensionen hinzu addiert und subtrahiert. Die resultierenden Werte sind dann die Ober- und Untergrenzen des Quaders dieser Klasse. Die Standardwahl für das Vielfache ist $\eta = 3$ und kann mit dem Parameter `sigmapar` verändert werden. Eine Variation wird für das Parametertuning sinnvollerweise aber nur durchgeführt, wenn `boxmethod="sigma"` gewählt wurde. Im Fall `boxmethod="range"` hätte die Veränderung von `sigmapar` keinen Einfluss auf das Ergebnis. Das Quader-Verfahren bietet insgesamt die meisten Tuningparameter aller Verfahren. Dadurch kann das Ergebnis unter Umständen weitreichend verbessert werden. Die Darstellung der Ergebnisse wird allerdings auch erschwert. Die mmce bezüglich aller vier Datensätze sind in Tabelle 12 zu sehen. Die linke Hälfte beinhaltet alle mmce die mit `empty.class="none"` berechnet wurden. Bei der rechten Hälfte wurde entsprechend `empty.class="all"` verwendet. Für die Einstellungen `"mind"` und `"ml"` des Parameters `tiemethod` wurde die Tabelle weiter in eine obere und untere Hälfte unterteilt. Zuletzt befinden sich in jeder Zeile eines Viertels der Tabelle die Einstellungen des Parameters `boxmethod`. Bei Verwendung von `"sigma"` ist die Wahl von `sigmapar` dahinter in Klammern angegeben. Die schon zuvor ermittelten Werte mit den Standardeinstellungen `"none"`, `"mind"` und `"range"` finden sich in der ersten Zeile des oberen linken Teils der Tabelle. Hier resultiert der beste Wert mit 0.8427 aus den mit dem clara-Algorithmus erzeugten Segmenten. Insgesamt gibt es drei Parameterkombinationen die niedrigere Werte aufweisen. Ebenfalls im oberen linken Tabellenteil sind bei der Wahl von `"sigma"` und $\eta = 1$ die Werte 0.8360 und 0.8385 zu sehen. Der größere Wert resultiert aus dem Datensatz CL, während der kleinere mittels KM berechnet wurde. Wird nun lediglich die Parameterwahl `"none"` durch `"all"` ersetzt, ist die dritte Parameterkombination mit einem Wert von 0.8409 im rechten oberen Teil der Tabelle bei KM zu finden. Es bestätigt sich also der Eindruck vorhergehender Berechnungen, dass die Segmente des k-means-Algorithmus am besten zur Klassifikation geeignet sind. Generell sind die mmce bei Verwendung von `"all"` statt `"none"` bis auf wenige Ausnahmen geringfügig höher. Diese Tendenz ist nicht überraschend, da sie bereits in den Ergebnissen des Minimum-Distance-Verfahrens zu erkennen ist. Nicht erwartet wurde

Tabelle 12: Ergebnisse des Parametertunings beim Quader-Verfahren

empty.class="none"					empty.class="all"				
tiemethod="mind"	KM	CL	PH	BC	tiemethod="mind"	KM	CL	PH	BC
"range"	0.8449	0.8427	0.8763	0.8697	"range"	0.8457	0.8433	0.8767	0.8699
"sigma" (0.25)	0.8523	0.8608	0.8844	0.8797	"sigma" (0.25)	0.8921	0.8940	0.9275	0.9177
"sigma" (0.5)	0.8538	0.8623	0.8819	0.8779	"sigma" (0.5)	0.8882	0.8920	0.9209	0.9138
"sigma" (1)	0.8360	0.8385	0.8536	0.8550	"sigma" (1)	0.8409	0.8442	0.8618	0.8641
"sigma" (2)	0.8443	0.8465	0.8687	0.8663	"sigma" (2)	0.8461	0.8490	0.8710	0.8683
"sigma" (3)	0.8489	0.8504	0.8790	0.8756	"sigma" (3)	0.8501	0.8517	0.8802	0.8764
"sigma" (4)	0.8506	0.8542	0.8827	0.8781	"sigma" (4)	0.8516	0.8551	0.8833	0.8785
"sigma" (5)	0.8518	0.8572	0.8837	0.8789	"sigma" (5)	0.8528	0.8579	0.8842	0.8792
tiemethod="ml"	KM	CL	PH	BC	tiemethod="ml"	KM	CL	PH	BC
"range"	0.9337	0.9468	0.9537	0.9250	"range"	0.9338	0.9469	0.9537	0.9251
"sigma" (0.25)	0.9353	0.9475	0.9552	0.9289	"sigma" (0.25)	0.9470	0.9584	0.9671	0.9438
"sigma" (0.5)	0.9269	0.9413	0.9481	0.9245	"sigma" (0.5)	0.9382	0.9514	0.9613	0.9392
"sigma" (1)	0.9117	0.9392	0.9390	0.9165	"sigma" (1)	0.9159	0.9445	0.9442	0.9266
"sigma" (2)	0.9382	0.9477	0.9540	0.9335	"sigma" (2)	0.9397	0.9501	0.9565	0.9372
"sigma" (3)	0.9384	0.9497	0.9581	0.9316	"sigma" (3)	0.9390	0.9505	0.9590	0.9324
"sigma" (4)	0.9375	0.9489	0.9567	0.9297	"sigma" (4)	0.9377	0.9492	0.9571	0.9301
"sigma" (5)	0.9367	0.9486	0.9562	0.9294	"sigma" (5)	0.9369	0.9488	0.9565	0.9296

hingegen die schlechten Werte bei Verwendung von `"ml"` an Stelle von `"mind"`. Bei Variation der `boxmethod` ist klar zu erkennen, dass die Verwendung von `"sigma"` mit $\eta = 1$ oder auch $\eta = 2$ vielfach zu niedrigeren Ergebnissen führt als `"range"`. Beachtenswert ist bei $\eta = 1$ zudem, dass die mmce der vier Datensätze jeweils näher beisammen liegen als mit anderen Einstellungen. Generell sind für `tiemethod="ml"` die besten Werte oft sogar bei BC zu finden, mit dem zuvor meist schlechte Werte ermittelt wurden. Ingesamt konnten die mmce beim Quader-Verfahren durch das Parametertuning leicht verbessert werden. Sie sind allerdings immer noch auf sehr hohem Niveau, sodass ein anderes Verfahren vorzuziehen wäre.
Die Ergebnisse des Parametertunings sind insgesamt durchwachsen. Ein Verfahren wurde wegen mangelnder Parameter nicht einbezogen. Ein Anderes wies keinerlei Veränderung bei Variation der Parameter auf. Bei zwei weiteren Verfahren wurden durch die Veränderung der Parameter nur schlechtere mmce erzielt. Lediglich bei der Support Vector Machine und dem Quader-Verfahren konnten leichte Verbesserungen erreicht werden. Der Support Vector Machine wird hierbei jedoch das größere Potenzial eingeräumt. Zum einen ergaben sich mit ihr schon die bisher niedrigsten mmce. Zum Anderen fand das Tuning aus zeitlichen Gründen nur mit einer eingeschränkten Auswahl von Parameterkombinationen statt. Die Support Vector Machine ist als Verfahren also weiterhin zu empfehlen. Die Parameter sollten jedoch zu $C_{SVM} = 2$ und $\gamma = \frac{1}{8}$ geändert werden. Teilziel 2c), das Parametertuning, ist damit ebenfalls abgeschlossen.

4.3.2 Pixelbasierte Klassifikation

Als letzter Teil der Auswertung erfolgt nun ein Vergleich zwischen segmentbasierter und pixelbasierter Klassifikation. Die segmentbasierte Klassifikation wurde bereits durchgeführt. Im Folgenden wird also zunächst die Ausführung der pixelbasierten Klassifikation beschrieben. Bei dieser werden die Pixel direkt anhand ihrer Messwerte in den sieben Spektralkanälen klassifiziert. Zusatzinformationen wie bei den Segmenten werden nicht verwendet. Es liegen also Objekte $1, \ldots, 535860$ mit Messwertvektoren $x_1, \ldots, x_{535860}$ für die $v = 7$ Variablen (in Form der Spektralkanäle) vor. Zusätzlich sind die echten Klassenzugehörigkeiten $y_1, \ldots, y_{535860}$ aus den CORINE-Daten gegeben. Die Klassen $C_1, \ldots, C_{14}, C_{16}, \ldots, C_{19}$ sind erneut analog zu den Bodenbedeckungsklassen aus Tabelle 2 (Kapitel 2.1, S. 6) nummeriert. Um

die Vergleichbarkeit der Klassifikationsarten zu gewährleisten werden wie zu Beginn der segmentbasierten Klassifikation die Standardparameter der Verfahren genutzt. Die lineare Diskriminanzanalyse hat keine erwähnenswerten Parameter. Die Support Vector Machine wird mit den Parametern $C_{SVM} = 1$ und $\gamma = \frac{1}{11}$ ausgeführt. Beim Minimum-Distance- und Maximum-Likelihood-Verfahren können nicht vorhandene Klassen mit `empty.class="infty"` und `empty.class="null"` nicht vorhergesagt werden. Dies trifft mit `empty.class="none"` auch auf das Quader-Verfahren zu. Falls ein Objekt nicht genau in einem Quader liegt, wird mit dem Minimum-Distance-Verfahren die Klasse bestimmt. Je nachdem ob das Objekt in mehreren oder keinem Quader liegt werden dazu nur die betroffenen oder alle Klassen verwendet. Beim Random Forest weicht die Anzahl der Bäume von der Standardwahl `ntree=500` ab. Bei der pixelbasierten Klassifikation sind wesentlich mehr Objekte vorhanden, die in die Berechnungen einfließen. Die Funktion `randomForest()` kann wegen mangelnder Arbeitsspeicherkapazität daher nur mit `ntree=100` ausgeführt werden. Da während des Parametertunings der segmentbasierten Klassifikation ebenfalls Random Forests mit 100 Bäumen berechnet wurden, können deren Ergebnisse dann mit den hier zu ermittelnden Werten verglichen werden. Bei besagtem Tuning unterscheiden sich die mmce bei 100 und 500 Bäumen nur geringfügig. Wird dieser Trend bei der Bewertung der Verfahren der pixelbasierten Klassifikation berücksichtigt, erscheint ein Vergleich vertretbar zu sein.

Die pixelbasierte Klassifikation mit dem Paket `mlr` verläuft analog zur segmentbasierten Klassifikation. Es wird lediglich ein neuer Task mit den oben beschriebenen Daten erzeugt. Die entsprechenden Learner können aus der segmentbasierten Klassifikation übernommen werden. Aus Gründen der Vergleichbarkeit wird bei jedem der Verfahren genau wie zuvor eine vierfache Kreuzvalidierung durchgeführt. Dazu wird das Satellitenbild wie bereits beschrieben in die Viertel ol, or, ul und ur unterteilt. Die Pixel aus drei der Viertel werden jeweils als Trainingsdaten genutzt, während die Pixel des letzten Viertels die Testdaten darstellen. Für die Kreuzvalidierung werden ebenfalls vier neue Resampling Instances benötigt, die festlegen, welche der Viertel jeweils Trainings- beziehungsweise Testdaten sind. Die Resampling Instances werden wieder über die Funktion `makeFixedHoldoutInstance()` erzeugt. Für jeden Durchlauf der Kreuzvalidierung, also für jede Resampling Instance wird die Fehlklassifikationsrate berechnet. Die vier Fehlklassifikationsraten für die einzelnen Verfahren sind in Tabelle 13 einsehbar. Die Angabe des Viertels in den Spalten deu-

Tabelle 13: Mittlere Fehlklassifikationsraten bei pixelbasierter Klassifikation

	ol	or	ul	ur	∅
LDA	0.4486	0.4191	0.4576	0.4360	0.4403
Random Forest (100)	0.4419	0.3861	0.4266	0.4257	0.4201
SVM	0.4226	0.3801	0.4161	0.4019	0.4052
MinDist	0.7580	0.7329	0.8459	0.7187	0.7639
MaxLike	0.4654	0.4478	0.4975	0.4795	0.4726
Quader	0.7506	0.7306	0.8443	0.7184	0.7610

tet darauf hin, dass die Pixel dieses Viertels dort zum Testen verwendet wurden. Zur zweckmäßigen Erzeugung der Werte in `R` wurde erneut die Funktion `benchmark()` genutzt. Dabei wurden alle Learner und jeweils eine Resampling Instance eingegeben. Aus den vier Raten jedes Verfahrens wird nun durch die Bildung des arithmetischen Mittels die mittlere Fehlklassifikationsrate (mmce) für jedes Verfahren bestimmt. Diese sechs Werte sind ebenfalls in Tabelle 13 in der Spalte "∅" zu finden. Die Support Vector Machine weist hier mit 0.4052 den niedrigsten und damit besten Wert auf. Da für den Random Forest nur 100 Bäume verwendet wurden, kann davon ausgegangen werden, dass das Ergebnis für 500 Bäume noch etwas niedriger wäre (siehe segmentbasierte Klassifikation). Die mmce des Random Forest ist also ebenfalls verhältnismäßig gut. Dies trifft auch auf die lineare Diskriminanzanalyse und das Maximum-Likelihood-Verfahren zu, deren mmce sich jeweils in kleinen Schritten erhöhen. Ein deutlicher Abstand besteht hingegen zu den mmce von Quader- und Minimum-Distance-Verfahren. Die Werte der beiden Verfahren sind sich sowohl im Durchschnitt als auch bei den einzelnen Vierteln sehr ähnlich. Falls ein Objekt nicht in exakt einem Quader ist, klassifiziert das Quader- anhand des Minimum-Distance-Verfahrens weiter. Daher liegt die Vermutung nahe, dass dieser Fall sehr oft eintrifft. Die pixelbasierte Klassifikation sowie der Vergleich der einzelnen Verfahren, wie in Teilziel 3a) angegeben, ist damit abgeschlossen.

Im folgenden Abschnitt werden nun noch die Ergebnisse der pixelbasierten und der segmentbasierten Klassifikation miteinander verglichen. Hierzu werden die Ergebnisse der Tabellen 7 und 13 nochmals übersichtlich in Tabelle 14 zusammengefasst. Die Werte der pixelbasierten Klassifikation befinden sich dabei in der Spalte "Pixel". Zunächst ist zu beobachten, dass die Ergebnisse der beiden Klassifikationsar-

Tabelle 14: Vergleich mittlerer Fehlklassifikationsraten für pixelbasierte und segmentbasierte Klassifikation

	Pixel	KM	CL	PH	BC
LDA	0.4403	0.4710	0.4753	0.5453	0.5421
Random Forest (100)	0.4201	0.4558	0.4635	0.5085	0.5098
SVM	0.4052	0.4477	0.4527	0.5028	0.5036
MinDist	0.7639	0.8523	0.8608	0.8844	0.8796
MaxLike	0.4726	0.5559	0.7209	0.8972	0.7595
Quader	0.7610	0.8449	0.8427	0.8763	0.8697

ten grundsätzlich übereinstimmen. Die Support Vector Machine weist bei beiden Ansätzen die niedrigste mittlere Fehlklassifikationsrate auf. Mit dem Minimum-Distance-Verfahren werden insgesamt die schlechtesten Werte ermittelt. Die Reihenfolge der anderen Verfahren dazwischen bleibt ebenfalls gleich. Die mmce der pixelbasierten Klassifikation sind dabei alle niedriger als bei den besten mmce der segmentbasierten Klassifikation, die alle mit dem Datensatz KM erzielt wurden. Je nach Verfahren sind die Differenzen unterschiedlich groß. Die mmce von linearer Diskriminanzanalyse, Random Forest und Support Vector Machine bei KM sind nur geringfügig (zwischen 0.03 und 0.05) höher als bei der pixelbasierten Klassifikation. Das Minimum-Distance-, das Maximum-Likelihood- und das Quader-Verfahren weisen hier mit 0.0884, 0.0833 und 0.0839 größere Unterschiede auf. Die Unterscheidung in typische Statistik- und Fernerkundungsverfahren fällt hier erneut ins Auge. Dieser Trend verringert sich, wenn statt KM einer der anderen Datensätze mit den pixelbasierten Ergebnissen verglichen wird. Ausgenommen hiervon ist das Maximum-Likelihood-Verfahren. Dessen Resultate mit den vier Datensätzen waren schon während der segmentbasierten Klassifikation sehr verschieden. Die Unterschiede zur pixelbasierten Klassifikation variieren dementsprechend stark. Der Unterschied zwischen pixelbasierter und segmentbasierter Klassifikation mit PH ist beispielsweise mit etwa 42.5 Prozentpunkten enorm hoch. Andererseits ist das Maximum-Likelihood-Verfahren bei pixelbasierter Klassifikation fast gleichwertig zu den drei Verfahren aus der Statistik. Neben den allgemein niedrigeren mmce ist dies der zweite Grund die pixelbasierte der segmentbasierten Klassifikation vorzuziehen. Statt drei gibt es hier nämlich nun vier akzeptable Klassifikationsverfahren, die ge-

nutzt werden können. Aus der Support Vector Machine ergeben sich zwar immer die besten Werte, sie hat allerdings auch einen großen Nachteil. Die Zeit für die Durchführung in `R` ist enorm. Betrug sie während der segmentbasierten Klassifikation schon je nach Datensatz 0.5 bis 2 Stunden, so werden in der pixelbasierten Variante mehr als 23 Stunden benötigt. Durch die Kreuzvalidierung wird die SVM ingesamt vier mal ausgeführt, womit sich auch die benötigte Zeit vervierfacht. Zum Vergleich benötigen alle vier Durchführungen der linearen Diskriminanzanalyse beziehungsweise des Maximum-Likelihood-Verfahrens weniger als 10 Minuten. Die großen Unterschiede bei der SVM resultieren aus der unterschiedlichen Menge an Objekten, die für das Training des Verfahrens genutzt werden. Bei segmentbasierter Klassifikation sind dies maximal 101938 Objekte (bei KM). Die pixelbasierte Klassifikation hingegen verwendet maximal 402090 Objekte zum Trainieren. Die knapp viertägige Berechnungszeit der SVM führt dazu, dass lineare Diskriminanzanalyse, Random Forest und Maximum-Likelihood-Verfahren in der pixelbasierten Klassifikation als nahezu gleichwertig zur SVM angesehen werden.

Ein Grund für die guten Ergebnisse der Support Vector Machine ist Abbildung 17 zu sehen. Dort ist der Ausschnitt eines Streudiagramms mit den Messwerten der

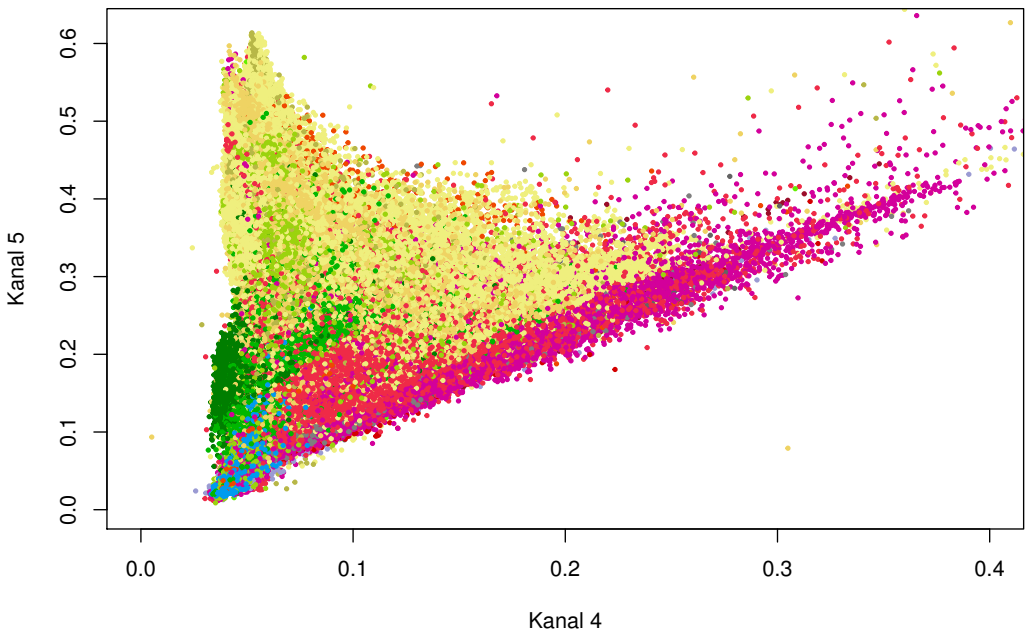


Abbildung 17: Streudiagramm für die Messwerte der Pixel aus den Spektralkanälen 4 und 5 (Ausschnitt)

Pixel aus den Kanälen 4 und 5 zu sehen. Einige Pixel wurden hierbei weggelassen um möglichst viele Details darstellen zu können. Die Pixel jeder Klasse wurden in der in Tabelle 2 (Kapitel 2.1, S. 6) angegebenen Farbe abgebildet. Es ist deutlich erkennbar, dass sich die meisten Klassen auf bestimmte Bereiche des Messraumes beschränken. Diese Bereiche überschneiden sich aber teilweise sehr stark. Die Support Vector Machine und auch der Random Forest, also die beiden Verfahren mit den besten Ergebnissen, haben hierdurch Vorteile. Sie bilden im Gegensatz zu den anderen Verfahren unregelmäßige Entscheidungsgrenzen für die Klassen und sind dadurch flexibler. Insbesondere gilt dies für die Support Vector Machine, bei der die Flexibilität durch den Parameter γ bestimmt wird. Durch dessen Erhöhung werden die Grenzen flexibler, wodurch im Parametertuning bereits bessere mmce erzielt wurden.

Für das jeweils beste pixelbasierte und segmentbasierte Klassifikationsverfahren werden deren Vorhersagen nun in den Abbildungen 18 und 20 dargestellt. Das beste

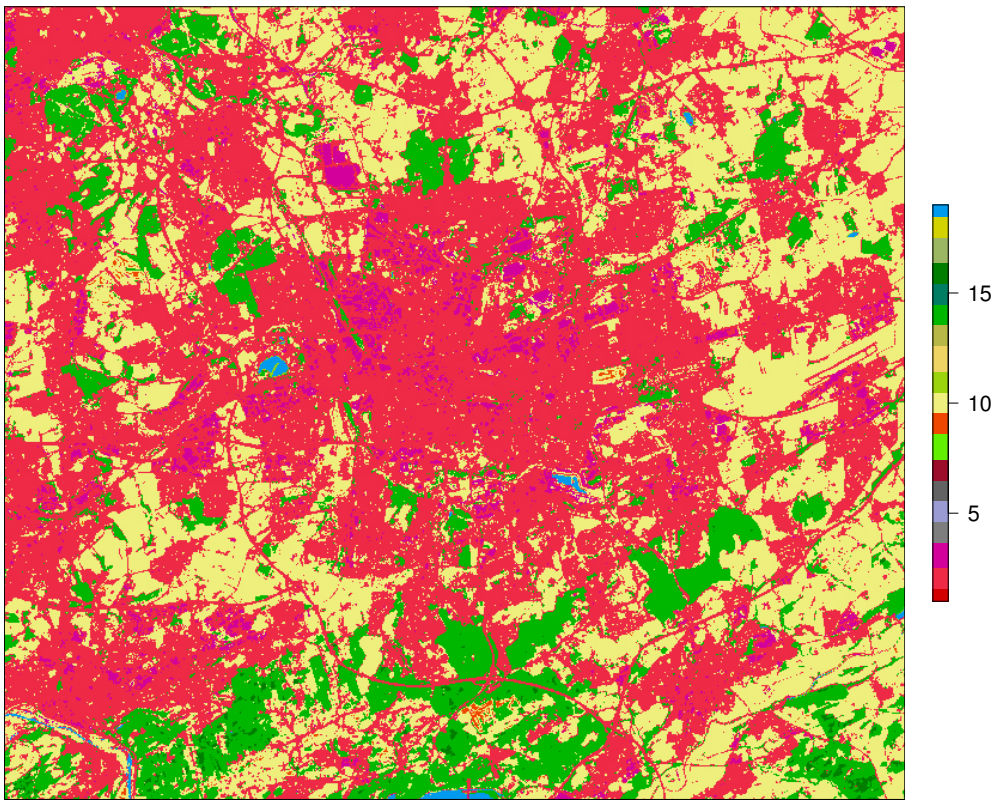


Abbildung 18: Vorhersagen der Bodenbedeckungsklassen anhand einer pixelbasierten Support Vector Machine mit den Parametern $C_{SVM} = 1$ und $\gamma = \frac{1}{11}$

Klassifikationsverfahren ist jeweils die Support Vector Machine mit den Parametern $C_{SVM} = 1$ und $\gamma = \frac{1}{11}$ (pixelbasiert) beziehungsweise $C_{SVM} = 2$ und $\gamma = \frac{1}{8}$ (seg-

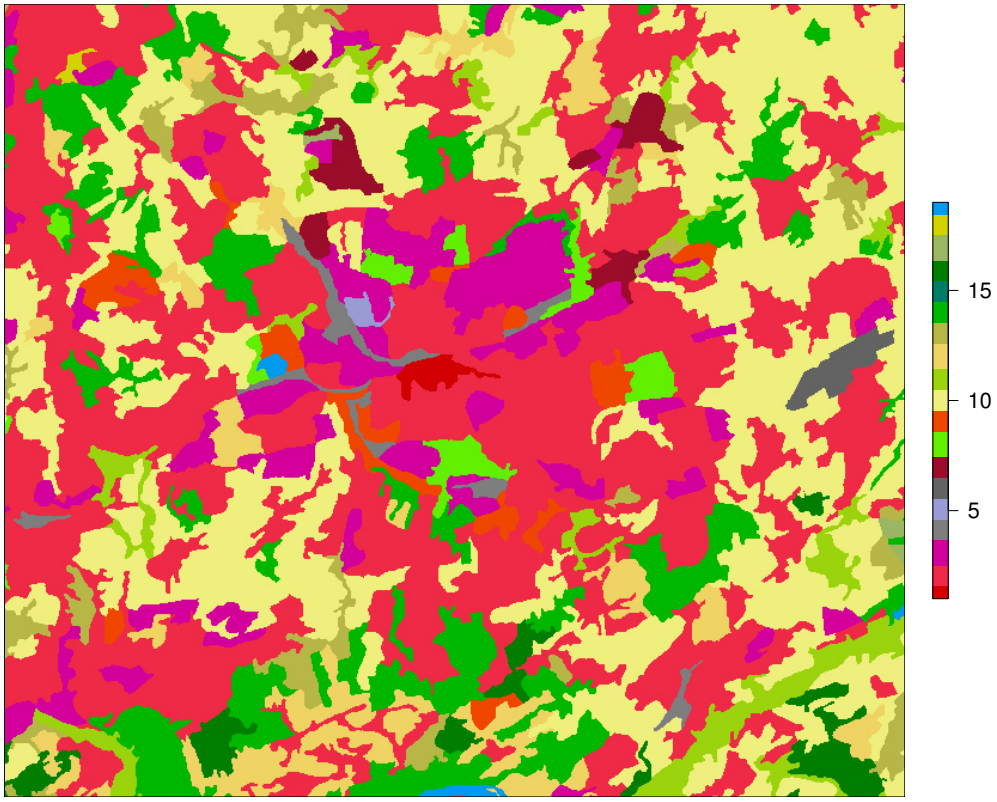


Abbildung 19: Echte Bodenbedeckungsklassen aus CORINE-Daten

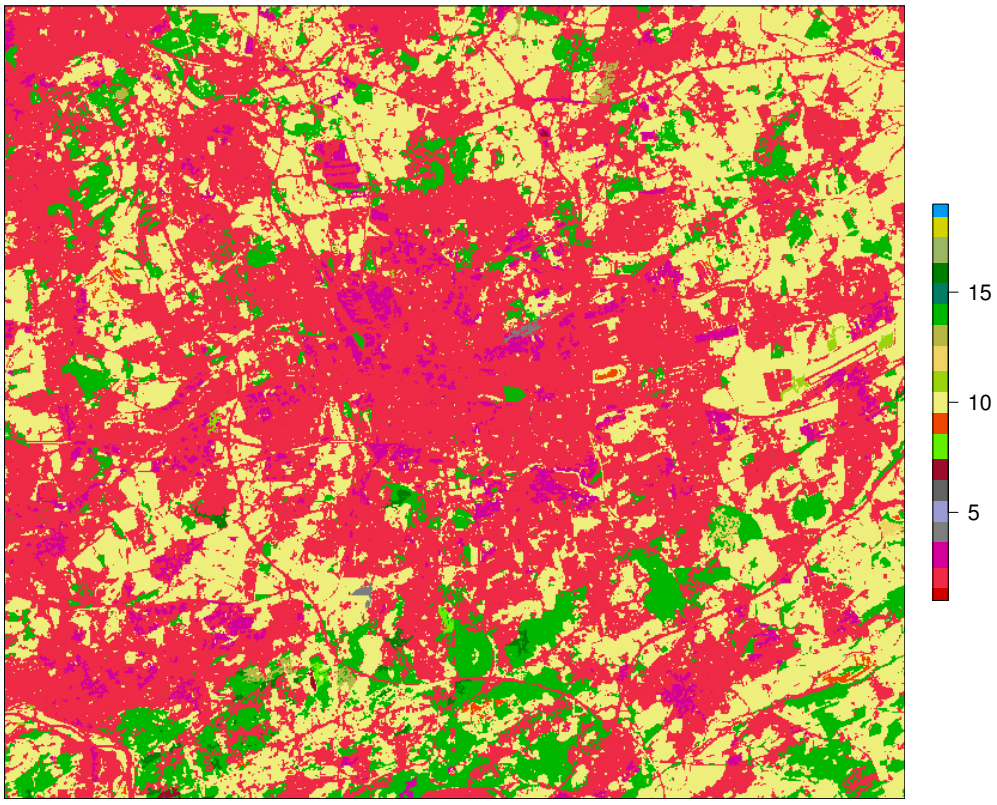


Abbildung 20: Vorhersagen der Bodenbedeckungsklassen anhand einer segmentbasierten Support Vector Machine mit den Parametern $C_{SVM} = 2$ und $\gamma = \frac{1}{8}$

mentbasiert). Die beiden Grafiken werden einer Karte mit den echten Bodenbedeckungsklassen (Abbildung 19) gegenübergestellt. Die häufig erscheinenden Klassen 2, 10 und 14 sind bei beiden Vorhersagen deutlich überproportional oft vorhergesagt worden. Kleine Klassen hingegen erscheinen in den Vorhersagen kaum. Für weitere Untersuchungen wäre es daher sinnvoll über eine Gewichtung der Klassen nach ihrer Größe nachzudenken. Die Strukturen der großen Klassen sind insgesamt gut erfasst worden, scheinen aber bei der pixelbasierten Klassifikation präziser zu sein. Das Ergebnis der mmce wird hier also bestätigt.

Die unterschiedlichen Ergebnisse bei den beiden Klassifikationsarten lassen sich durch die Bildung der Segmente erklären. Die Informationen eines Segmentes wurden jeweils aus den Messwerten der enthaltenen Pixel gemittelt. Je nach Größe des Segments kommt es dadurch zu einem erheblichen Informationsverlust. Dieser kann nur zum Teil durch die Zusatzinformationen zu jedem Segment wieder ausgeglichen werden. Bei Datensätzen mit kleinen Segmenten wie KM oder CL ist dieser Verlust nicht so hoch wie bei PH und BC, die größere Segmente beinhalten. Alles in allem ist die Entscheidung für eine der beiden Klassifikationsarten schwierig. Die Auswahl sollte nach den vorhandenen Rahmenbedingungen der Untersuchung getroffen werden. Die pixelbasierte Klassifikation ist von den Ergebnissen her vorzuziehen. Dies gilt allerdings nur, falls kein wesentlich größeres als das hier verwendete Satellitenbild genutzt wird. Bei einem solchen größeren Bild wäre die segmentbasierte Klassifikation die bessere Wahl, da es bei der pixelbasierten Klassifikation wohl zu zeitlichen Problemen kommen würde. Davon ausgenommen ist die reine Vorhersage von Klassen für andere Satellitenbilder anhand eines zuvor erstellten Modells. Die Erstellung der Vorhersagen erfolgt auch bei größeren Bildern in verhältnismäßig kurzer Zeit, sodass hierzu am Besten das Modell der pixelbasierten Support Vector Machine genutzt wird. Das letzte Teilziel 3b) wurde mit obigem Vergleich von pixelbasierter und segmentbasierter Klassifikation erreicht. Die wichtigsten Ergebnisse zu den einzelnen Zielen werden nun im folgenden Kapitel zusammengefasst.

5 Zusammenfassung

Die in den Kapiteln 1 - 4 beschriebene Untersuchung beinhaltet den Vergleich verschiedener Klassifikationsverfahren in der Fernerkundung. Die Fernerkundung befasst sich unter Anderem mit der Erstellung von thematischen Karten anhand von Satellitenbildern. Im konkreten Fall liegt ein Satellitenbild der Stadt Dortmund mit Umgebung vor, aus dem eine Karte mit Bodenbedeckungen produziert werden soll. Dazu ist es notwendig die Teile des Satellitenbildes in Form von Pixeln in Bodenbedeckungsklassen einzuordnen. Die Pixel der Klassen werden in unterschiedlichen Farben dargestellt und bilden so die Karte. Die Einordnung der Pixel in die Klassen geschieht mit den Informationen des Satellitenbildes. Diese bestehen aus den Messwerten von sieben Spektralkanälen, die die Reflexion elektromagnetischer Strahlung von der Erde über am Satellit befestigte Sensoren messen. Die Strahlung ist Licht in verschiedenen Spektralfarben. Die Erstellung der Karte sollte wenig Kosten verursachen und gleichzeitig schnell durchführbar sein, um die Aktualität möglichst hoch zu halten. Daher wird nach einem automatisierten Verfahren für dieses Klassifikationsproblem (Zuordnung von Klassen mittels Messwerten) gesucht. Die zu diesem Zweck verwendeten Klassifikationsverfahren stellen dabei die Verbindung zwischen Fernerkundung und Statistik dar. In beiden Bereichen gibt es unterschiedliche Verfahren von denen jeweils drei für den Vergleich genutzt werden. So kann sowohl ein Vergleich zwischen den Verfahren als auch zwischen den Bereichen gezogen werden. Neben dem Satellitenbild sind auch die Bodenbedeckungsklassen der einzelnen Pixel gegeben, damit die Klassifizierungen der Verfahren entsprechend bewertet werden können.

In der Fernerkundung wird zwischen pixelbasierter und segmentbasierter Klassifikation unterschieden. Der Fokus liegt hier auf der segmentbasierten Klassifikation. Bei dieser werden aus den Pixeln Segmente gebildet, die dann von den Verfahren klassifiziert werden. Die Bildung der Segmente (Segmentierung) kann auf unterschiedliche Arten vollzogen werden. In dieser Arbeit wurden vier Clusterverfahren aus dem Bereich der Statistik verwendet. Dies waren der k-means-Algorithmus von Hartigan und Wong, der clara-Algorithmus, partielles hierarchisches agglomeratives Clustern und Bagged Clustering. Die Clusterverfahren berücksichtigen die Position der Pixel innerhalb des Bildes nicht, sondern arbeiten nur mit den Messwerten der Spektralkanäle. Aus diesem Grund werden mit den Verfahren zunächst nur Gruppen erzeugt.

Im Anschluss werden aus benachbarten Pixeln mit der gleichen Gruppe Segmente gebildet. Mit dem k-means-Algorithmus entstanden dabei die kleinsten Segmente, während die mit partiellem hierarchischem agglomerativem Clustern erzeugten Segmente durchschnittlich am größten sind. Die Art und Weise der Segmentbildung mittels der Clusterverfahren wird nicht nur visuell sondern auch über die selbst definierte NIBS-Distanz bewertet. Die NIBS-Distanz vergleicht zwei Segmentierungen durch die paarweise Betrachtung deren Segmente. Anhand der gegebenen Bodenbedeckungsklassen der Pixel können ebenfalls Segmente bestimmt werden. Diese werden als zu erreichende optimale Segmentierung betrachtet und jeweils mit einer durch eines der Clusterverfahren erzeugten Segmentierung verglichen. Mit der Segmentierung des k-means-Algorithmus wird die höchste und damit schlechteste Distanz ermittelt. Am besten geeignet scheint die Segmentierung durch das partielle hierarchische agglomerative Clustern, die die kleinste Distanz besitzt. Insgesamt gesehen sind aber alle Distanzen sehr groß, sodass die Nachempfindung der Bodenbedeckungsstruktur als schwierig bewertet wird. Die Segmentierungen aus allen vier Clusterverfahren werden für die Klassifikation weiterverwendet. Dazu werden für jede Segmentierung Segmentinformationen aus den Messwerten der zugehörigen Pixel gemittelt. Die Segmente mit ihren Informationen stellen dann jeweils einen neuen Datensatz dar, mit dem in der Folge klassifiziert wird. Insgesamt wird die Klassifikation also mit vier verschiedenen Datensätzen durchgeführt. Dadurch kann auch die Auswirkung der Segmentbildungsart auf die Klassifikation analysiert werden. Zuvor werden allerdings noch zusätzliche Informationen zur Struktur und Homogenität der Segmente aus den Pixeln gewonnen und zum jeweiligen Datensatz hinzugefügt. Diese Zusatzinformationen sind der Hauptgrund für die Durchführung einer segmentbasierten Klassifikation und sollen zur Verbesserung der Verfahren führen.
Die Klassifizierung der Segmente wurde mit der Software `R` unter Verwendung des Paketes `mlr` durchgeführt. Das Paket ermöglicht eine einheitliche und damit effiziente Bearbeitung von Klassifikationsproblem. Der Aufbau des Paketes ist jedoch recht komplex. Die Nutzung und die einzelnen Funktionen werden daher vor der Klassifikation zunächst etwas allgemeiner erläutert. Zusätzlich wird gezeigt, wie die Verfahren der Fernerkundung, für die eigene Funktionen erstellt wurden, in das Paket integriert und somit nutzbar gemacht werden können. Anschließend wird die eigentliche Klassifikation beschrieben, die mit sechs Klassifikationsverfahren durchgeführt wird. Als Verfahren aus dem Bereich der Statistik werden eine lineare Diskriminanzanalyse,

ein Random Forest und eine Support Vector Machine verwendet. Aus der Fernerkundung werden das Minimum-Distance-, das Maximum-Likelihood- sowie das Quader-Verfahren genutzt. Für jedes der Verfahren wird eine vierfache Kreuzvalidierung ausgeführt, um eine Überanpassung an die vorhandenen Daten zu vermeiden. Die aus den Verfahren erstellten Modelle wären sonst zwar gut mit dem Satellitenbild von Dortmund, jedoch schlecht mit anders strukturierten Bildern nutzbar. Normalerweise werden bei einer vierfachen Kreuzvalidierung für jeweils zirka 25 % der Daten Vorhersagen für die Klasse erstellt, während die anderen 75 % zum Erstellen des Modells verwendet werden. Hier bestehen die Daten jedoch aus unterschiedlich großen Segmenten, weshalb das Satellitenbild geviertelt und jeweils die Segmente aus einem Viertel für die Vorhersagen verwendet werden. Somit wird die Nutzung etwa gleich vieler Informationen gewährleistet. Zu diesem Zweck wird bereits die Erzeugung der Segmente für jedes Viertel des Satellitenbildes seperat durchgeführt. Mit den Vorhersagen für die Segmente eines Viertels und den echten Klassenzugehörigkeiten können Fehlklassifikationsraten bestimmt werden. Diese geben an, welcher Anteil der Vorhersagen falsch ist und werden zur Bewertung der Klassifikationsverfahren genutzt. Um einen Gesamteindruck des Satellitenbildes zu erhalten, werden die Raten der Viertel zu einer mittleren Fehlklassifiaktionsrate (mmce) kombiniert. Diese mmce wird für jedes Verfahren und jeden der vier Datensätze bestimmt und für den Vergleich herangezogen. Insgesamt lässt sich sagen, dass die Raten mit Werten zwischen 40 und 90 % recht hoch sind. Dieses relativ hohe Niveau lässt sich zum Teil durch die unterschiedlichen Zeitpunkte der Erstellung des Satellitenbildes sowie der Bodenbedeckungsklassen erklären. Beim Vergleich der Verfahren untereinander ist dieser Unterschied aber nicht relevant. Hier ergeben sich mit der Support Vector Machine die niedrigsten und damit besten mmce, egal welcher Datensatz verwendet wurde. Lineare Diskriminanzanalyse und Random Forest weisen geringfügig schlechtere Werte auf. Die mmce von Minimum-Distance- und Quader-Verfahren haben ein erkennbar höheres Niveau. Die Raten des Maximum-Likelihood-Verfahrens schwanken bei Verwendung der vier unterschiedlichen Datensätze sehr stark. Ein Vergleich der Datensätze wiederum zeigt, dass die mit dem k-means-Algorithmus erzeugten Segmente am besten zur Klassifikation geeignet sind. Anhand der mittels partiellem hierarchischem agglomerativem Clustern erzeugten Segmente wurden hingegen die höchsten Ergebnisse erzielt. Bisher ist also eines der Klassifikationsverfahren aus dem Bereich Statistik, bevorzugt die Support Vector Machine, zu empfehlen. Die

Segmentierung sollte anhand des k-means-Algorithmus erfolgen.
Um die Ergebnisse der Verfahren zu verbessern wurde nun, sofern möglich, ein Parametertuning durchgeführt. Zu diesem Zweck werden von den Klassifikationsverfahren verwendete Parameter variiert. Für das Tuning wurde eine Gittersuche ausgeführt, bei der Werte für die einzelnen Parameter vorgegeben werden. Um die Vergleichbarkeit zu den zuvor mit den "Standardparametern" erzeugten mmce zu gewähren, wird für jede Verfahrensdurchführung mit veränderten Parametern ebenfalls eine vierfache Kreuzvalidierung ausgeführt. Die Kreuzvalidierung erfolgt dabei wie bereits beschrieben durch die Aufteilung des Satellitenbildes in Viertel. Für jeden Parametersatz werden dadurch mmce berechnet, die den Vergleich mit den Standardparametern ermöglichen. Bei der linearen Diskriminanzanalyse findet sich kein sinnvoll variierbarer Parameter. Die Wahl des Parameters beim Maximum-Likelihood-Verfahren hingegen hat keine Auswirkungen auf die mmce. Bei beiden Verfahren gibt es also keine Verbesserungen. Beim Random Forest und Minimum-Distance-Verfahren führt das Tuning nur zu schlechteren Werten. Die Standardparameter werden hier als beste Wahl eingestuft. Leichte Verbesserungen ergeben sich lediglich beim Tuning der Support Vector Machine und des Quader-Verfahrens. Die Eigenheiten der vier Datensätze bleiben beim Tuning erhalten. Vor allem bei der Support Vector Machine ist jedoch zu berücksichtigen, dass das Gitter nur sehr klein gewählt wurde. Dies geschah hauptsächlich aus zeitlichen Gründen, da die Durchführung des Verfahrens sehr langwierig ist. Diese zeitliche Komponente sollte bei der Wahl eines der Klassifikationsverfahrens ebenfalls beachtet werden. Die Verwendung der Support Vector Machine sowie des k-means-Algorithmus zur Segmentierung sind aber weiter empfehlenswert.
Da die Fernerkundung in pixelbasierte und segmentbasierte Klassifikation unterscheidet, wird nun abschließend ein Vergleich zwischen diesen beiden Klassifikationsarten gezogen. Hierzu wird zunächst mit dem gleichen Satellitenbild sowie den Informationen zur Bodenbedeckung wie zuvor eine pixelbasierte Klassifikation ausgeführt. Bei dieser werden die Pixel direkt mittels der Messwerte aus den sieben Spektralkanälen klassifiziert. Aus Gründen der Vergleichbarkeit wird bei jedem der sechs Verfahren unter Verwendung gleicher Parameter eine vierfache Kreuzvalidierung zur Bestimmung einer mittleren Fehlklassifikationsrate ausgeführt. Das Satellitenbild wird dazu auf die gleiche Weise geviertelt und für die Pixel jeweils eines Viertels werden Vorhersagen für die Bodenbedeckungsklassen erstellt. Die Resultate

der pixelbasierten Klassifikation bestätigen die bereits beobachtete Ergebnisstruktur. Wie bei der segmentbasierten Klassifikation weisen die Verfahren der Statistik die besten Werte auf, wobei die Support Vector Machine wiederum die kleinste mmce erzeugt. Auffällig ist allerdings, dass das Maximum-Likelihood-Verfahren fast vergleichbare Werte zu diesen drei Verfahren erzielt. Die mmce der pixelbasierten Klassifikation sind jeweils etwas kleiner und somit besser als die besten Werte aus der segmentbasierten Klassifikation. Dies lässt sich über den Informationsverlust erklären, der bei der Zusammenfassung von Pixel- zu Segmentinformationen entsteht. Es können zwar auch neue Informationen über die Segmente gewonnen werden, diese scheinen aber nicht ausreichend um den Unterschied auszugleichen. Die pixelbasierte ist also der segmentbasierten Klassifikation vorzuziehen. Ausgenommen hiervon ist die Klassifikation eines größeren als des hier verwendeten Satellitenbildes, bei dem es mit Verwendung der Support Vector Machine zu zeitlichen Problemen kommen kann. Insgesamt klassifiziert die Support Vector Machine also am besten. Dies kann auf die durch das Verfahren erstellten flexiblen Entscheidungsgrenzen zurückgeführt werden.

Abschließend sei angemerkt, dass nur die gebräuchlichsten Verfahren für den Vegleich genutzt wurden. In der Statistik gibt es zahlreiche weitere Klassifikationsverfahren, die hier aber aus zeitlichen Gründen nicht verwendet wurden. Sie könnten aber unter gleichen Voraussetzungen als Teil einer Folgeuntersuchung in den Vergleich einbezogen werden. Dasselbe gilt auch für die Clusterverfahren, für die in der Statistik ebenfalls weitere Ansätze zu finden sind. Interessant wäre in diesem Zusammenhang auch eine Form der Segmentierung, die, anders als die Clusterverfahren, die Nachbarschaftsbeziehungen der Pixel berücksichtigt. Zum Beispiel können sogenannte “Region Growing“-Verfahren, wie in Haralick und Shapiro (1985, S. 115ff) beschrieben, verwendet werden. Die Umsetzung der Verfahren in `R` müsste aber für mehr als einen Spektralkanal vermutlich selbst erstellt werden. Alternativ zu weiteren Verfahren können auch die bereits Verwendeten eventuell noch weiter verbessert werden. Die Gittersuche beim Parametertuning für die Klassifikationsverfahren ist sicher nicht erschöpfend. Vor allem bei dem aus zeitlichen Gründen recht kleinen Gitter der Support Vector Machine wird noch ein größeres Potenzial vermutet. Statt einer einfachen Gittersuche wären hierzu auch andere komplexere Suchstrategien denkbar. Die Clusterverfahren wurden nur mit ihren Standardparametern beziehungsweise einer vorgegebenen Anzahl Gruppen ausgeführt. Auch hier wäre

eine Variation möglich. Die Suche nach optimalen Einstellungen könnte dabei an der NIBS-Distanz oder den durch die Segmente resultierenden Klassifikationsergebnissen ausgerichtet werden. Eine andere Option für weitere Untersuchungen ist die Verwendung verschiedener Distanzmaße. Bei den Clusterverfahren und einigen Klassifikationsverfahren wurde hier ausschließlich die euklische Distanz verwendet, da diese in der gängigen Literatur fast immer exklusiv verwendet wird. Um Ähnlichkeit zwischen Pixeln zu messen wären aber auch anderen Distanzmaße, beispielsweise die Manhattan-Distanz, anwendbar. Eine Veränderung kann ebenfalls durch eine Gewichtung der Bodenbedeckungsklassen erfolgen. Hier wurden die Klassen alle auf die gleiche Weise verwendet. Ist die Erkennung von bestimmten Klassen aber wichtiger, so kann dies bei den meisten Verfahren angegeben werden. In der Analyse wurde festgestellt, das häufig auftretende Klassen überproportional vorhergesagt werden. Die Gewichtung könnte folglich an der Pixelanzahl in den Klassen ausgerichtet werden. Auch die mittlere Fehlklassifikationsrate kann hierzu angepasst oder alternativ ein gänzlich anderes Entscheidungskriterium genutzt werden.
Der hier ausgeführte Vergleich ist also nicht abschließend. Er ermöglicht aber eine erste Orientierung bezüglich in der Fernerkundung sinnvoll einsetzbarer Klassifikationsverfahren.

Anhang

A Weitere Grafiken

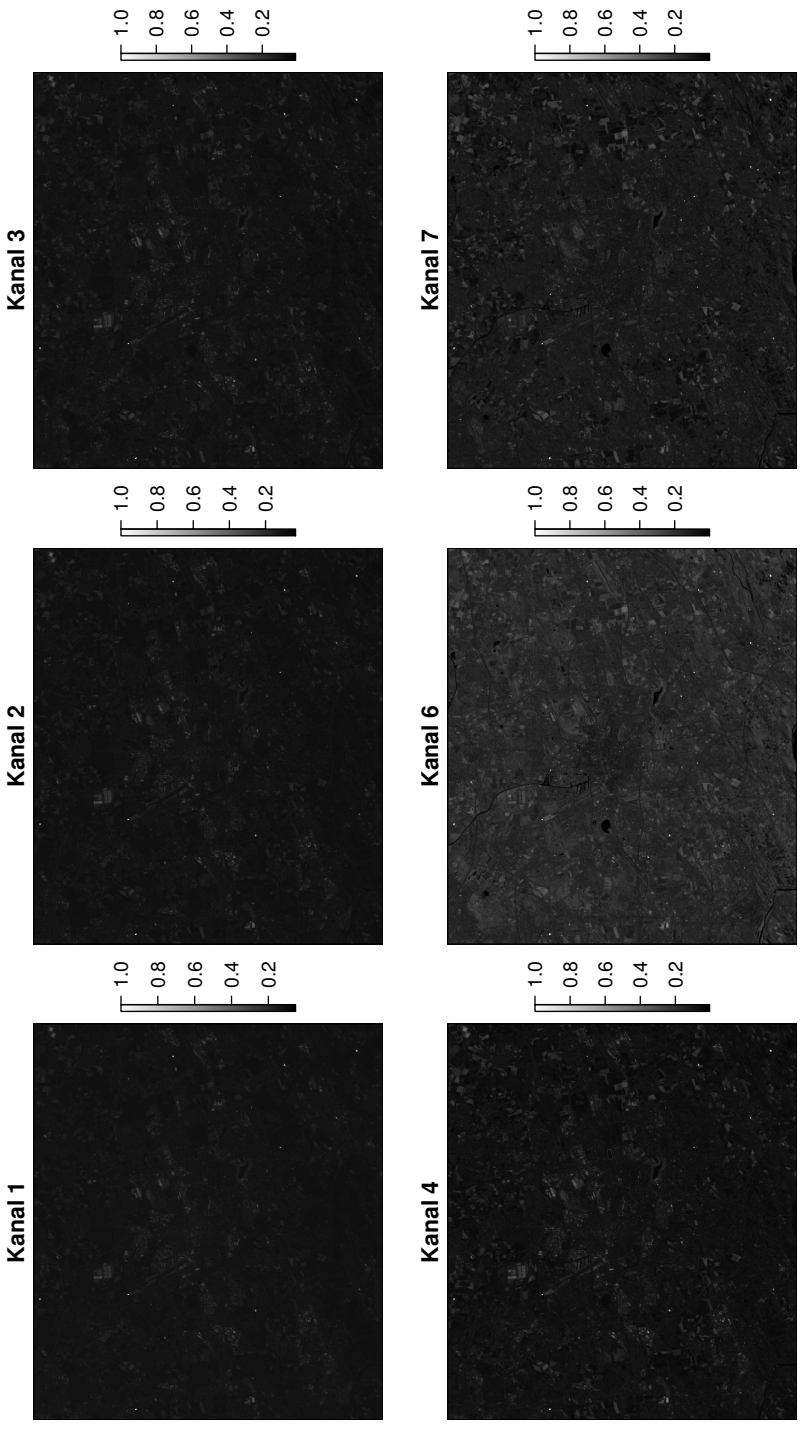


Abbildung 21: Grauwertdarstellung Kanäle 1, 2, 3, 4, 6 und 7

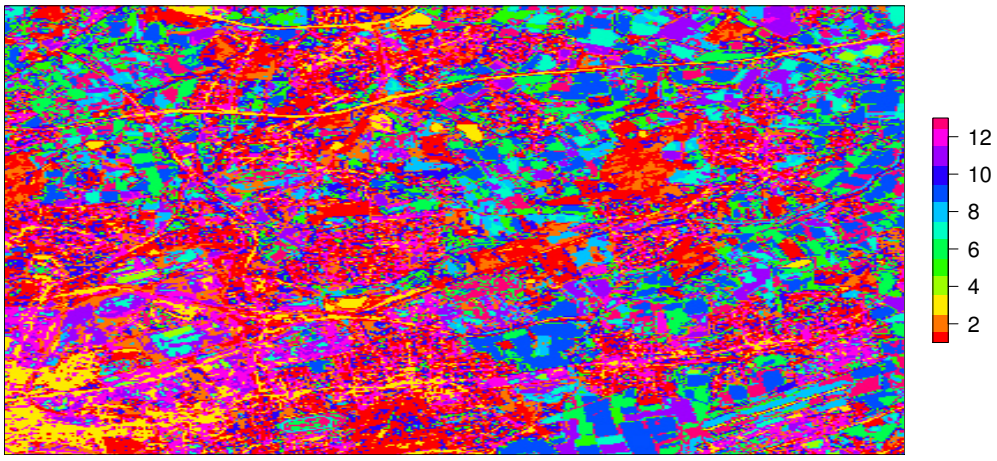


Abbildung 22: Gruppen anhand des k-means-Algorithmus im oberen rechten Teil des Satellitenbildes

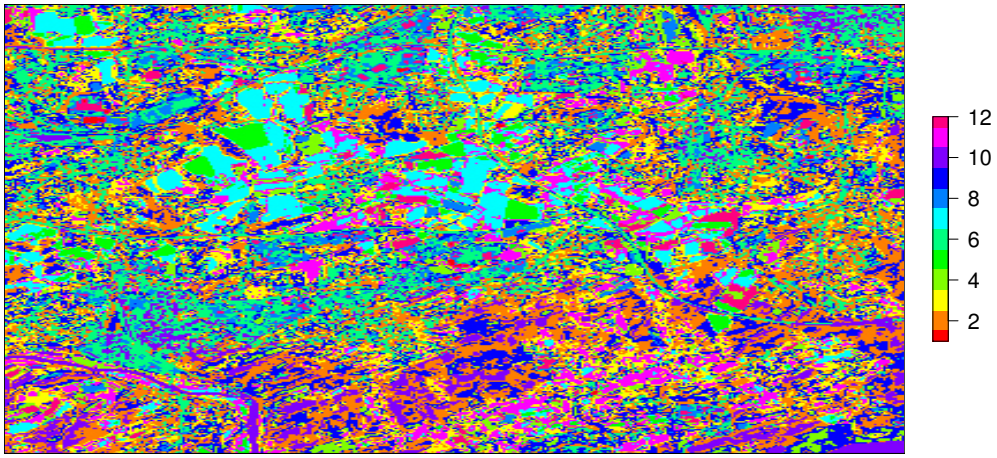


Abbildung 23: Gruppen anhand des k-means-Algorithmus im unteren linken Teil des Satellitenbildes

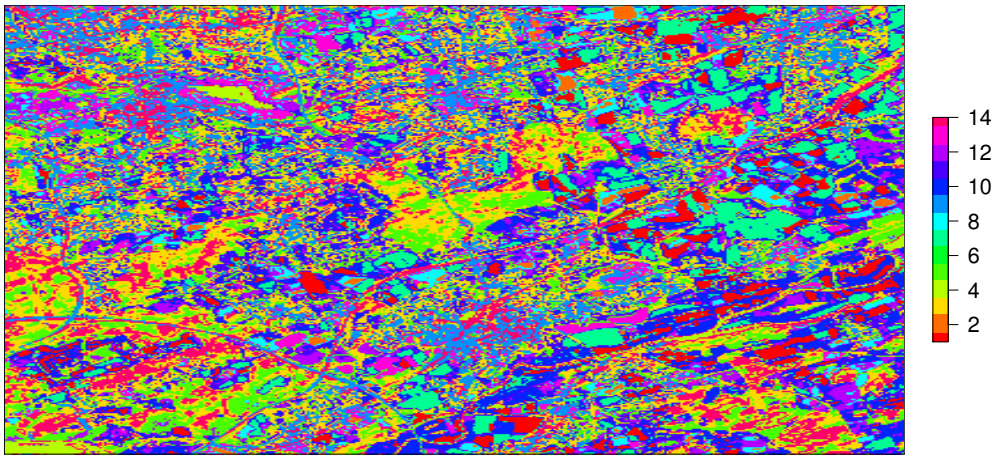


Abbildung 24: Gruppen anhand des k-means-Algorithmus im unteren rechten Teil des Satellitenbildes

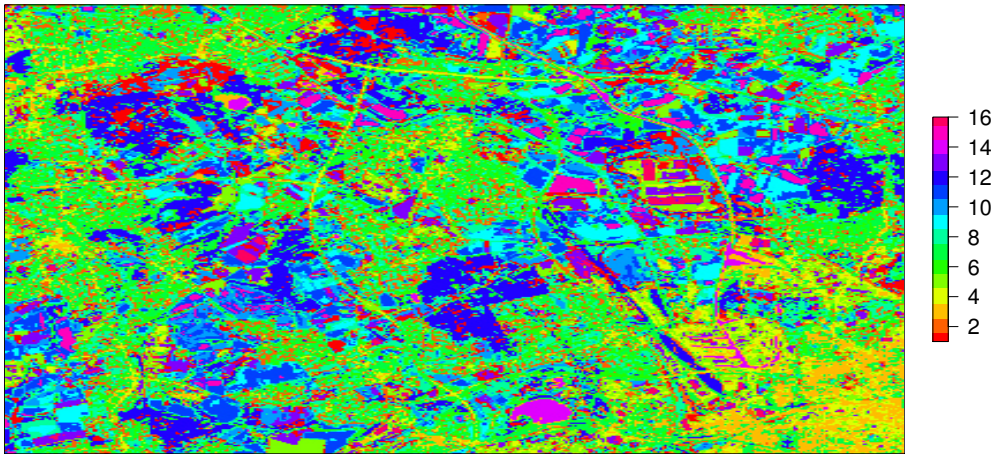


Abbildung 25: Gruppen anhand des clara-Algorithmus im oberen linken Teil des Satellitenbildes

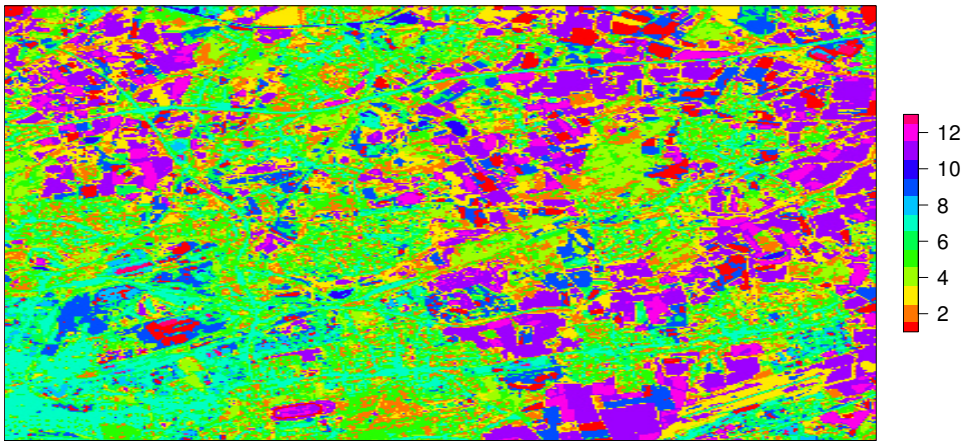


Abbildung 26: Gruppen anhand des clara-Algorithmus im oberen rechten Teil des Satellitenbildes

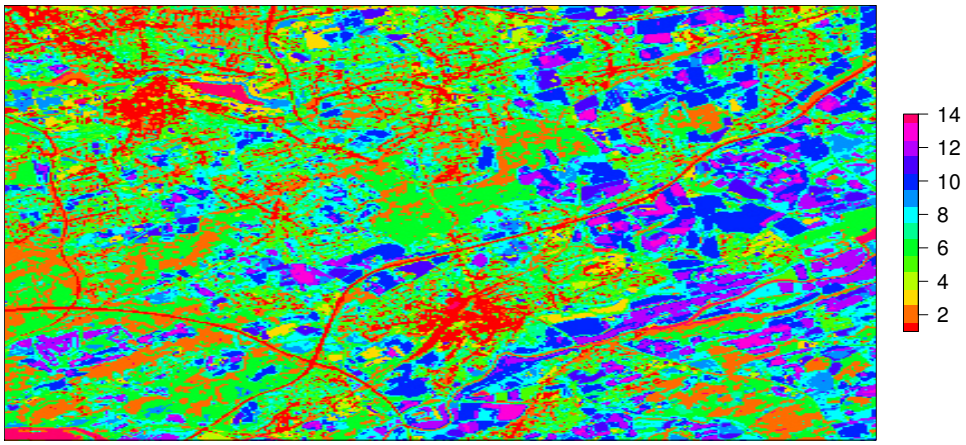


Abbildung 27: Gruppen anhand des clara-Algorithmus im unteren rechten Teil des Satellitenbildes

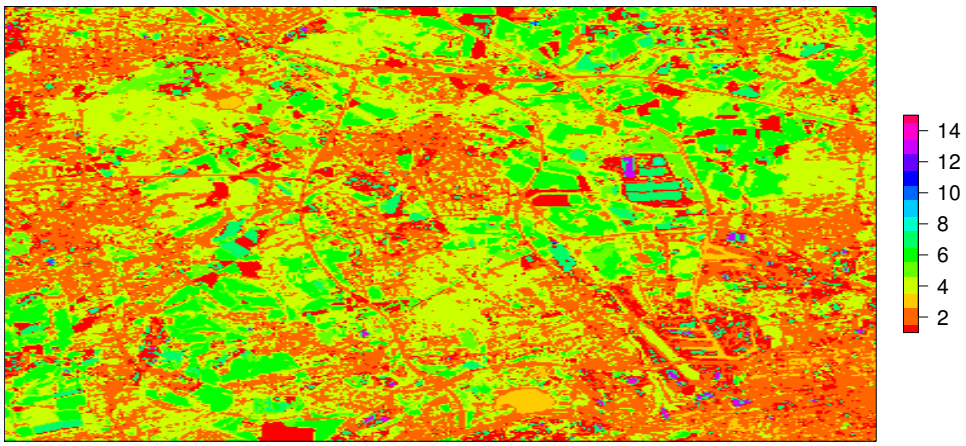


Abbildung 28: Gruppen anhand des phclust-Verfahrens im oberen linken Teil des Satellitenbildes

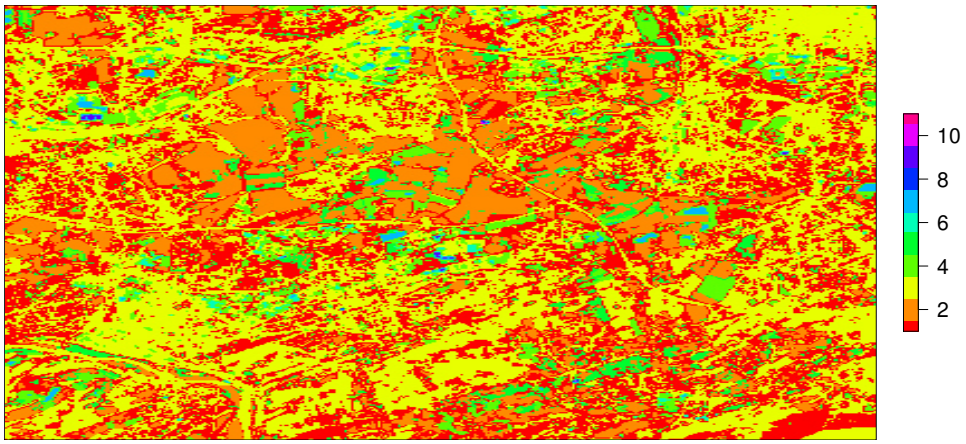


Abbildung 29: Gruppen anhand des phclust-Verfahrens im unteren linken Teil des Satellitenbildes

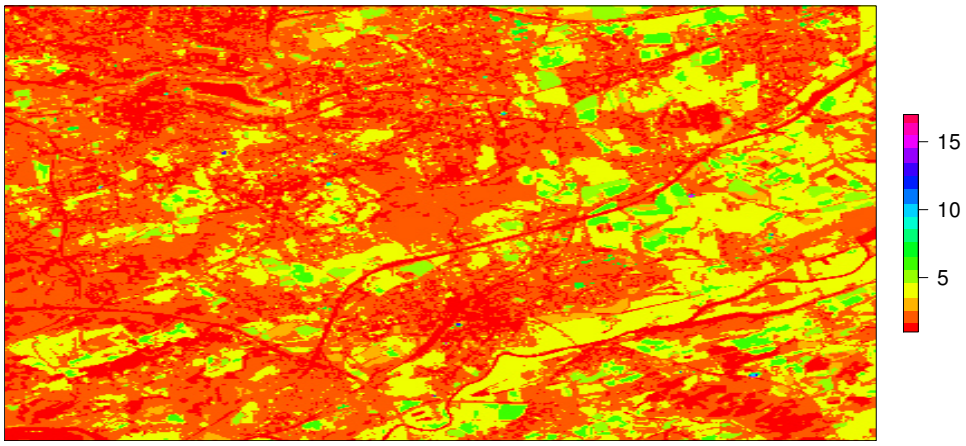


Abbildung 30: Gruppen anhand des phclust-Verfahrens im unteren rechten Teil des Satellitenbildes

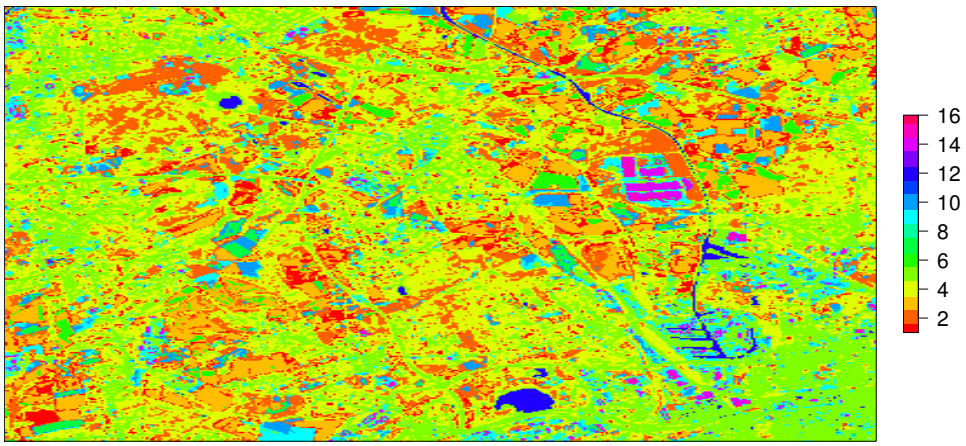


Abbildung 31: Gruppen anhand des Verfahrens Bagged Clustering im oberen linken Teil des Satellitenbildes

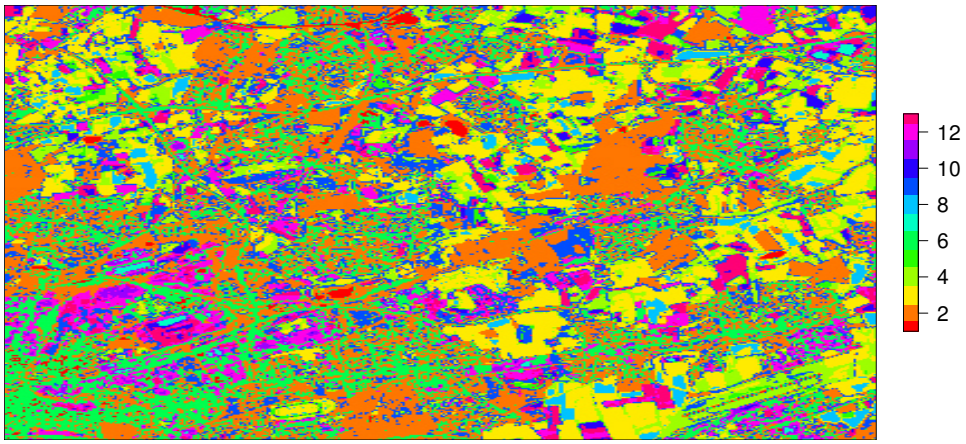


Abbildung 32: Gruppen anhand des Verfahrens Bagged Clustering im oberen rechten Teil des Satellitenbildes

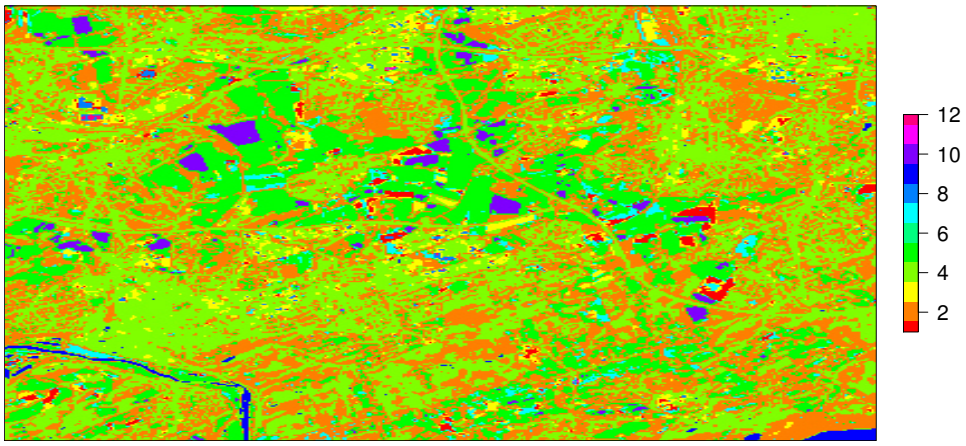


Abbildung 33: Gruppen anhand des Verfahrens Bagged Clustering im unteren linken Teil des Satellitenbildes

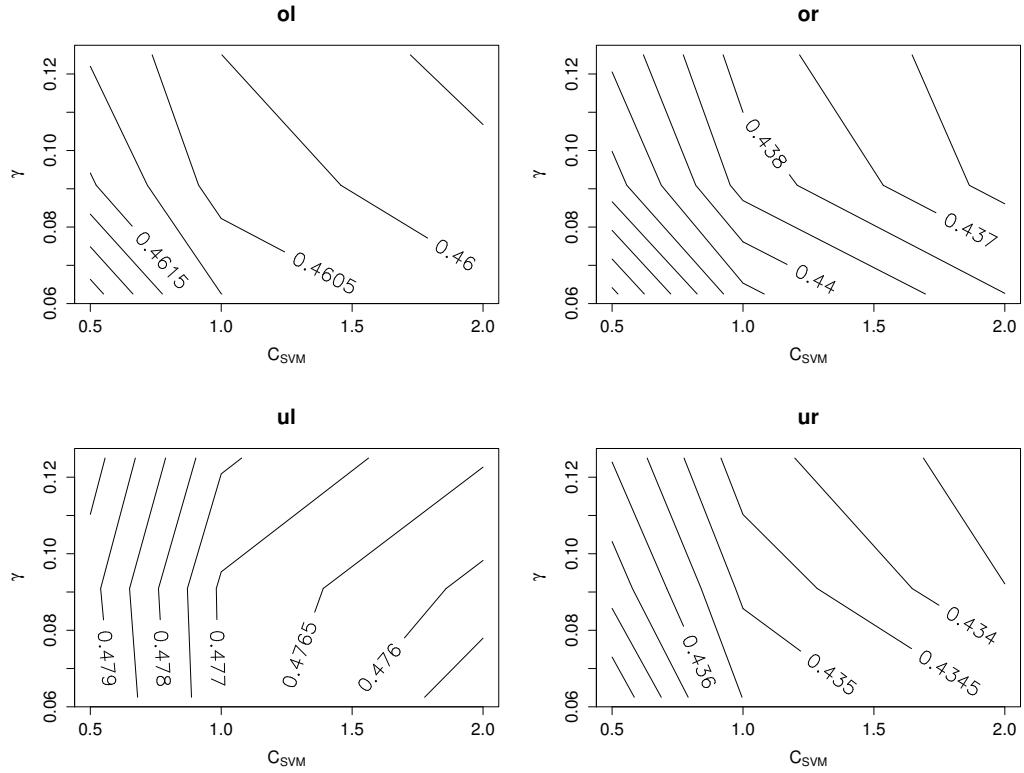


Abbildung 34: Contour-Grafik für die Ergebnisse des Parametertunings der Support Vector Machine bei Verwendung des Datensatzes CL

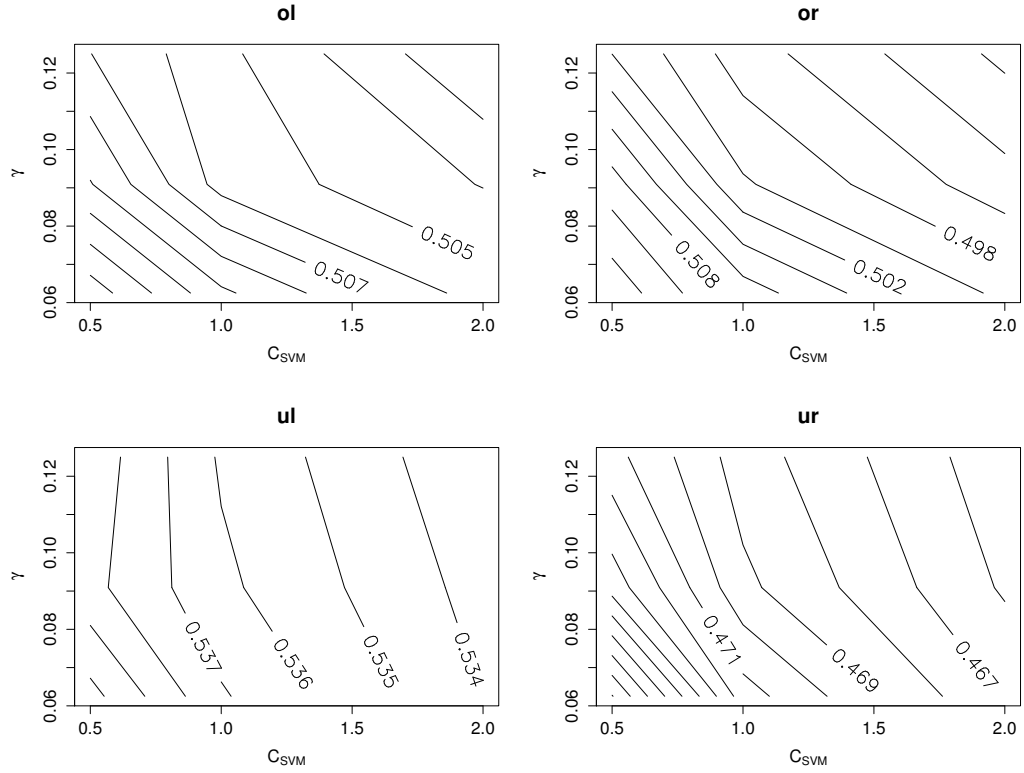


Abbildung 35: Contour-Grafik für die Ergebnisse des Parametertunings der Support Vector Machine bei Verwendung des Datensatzes PH

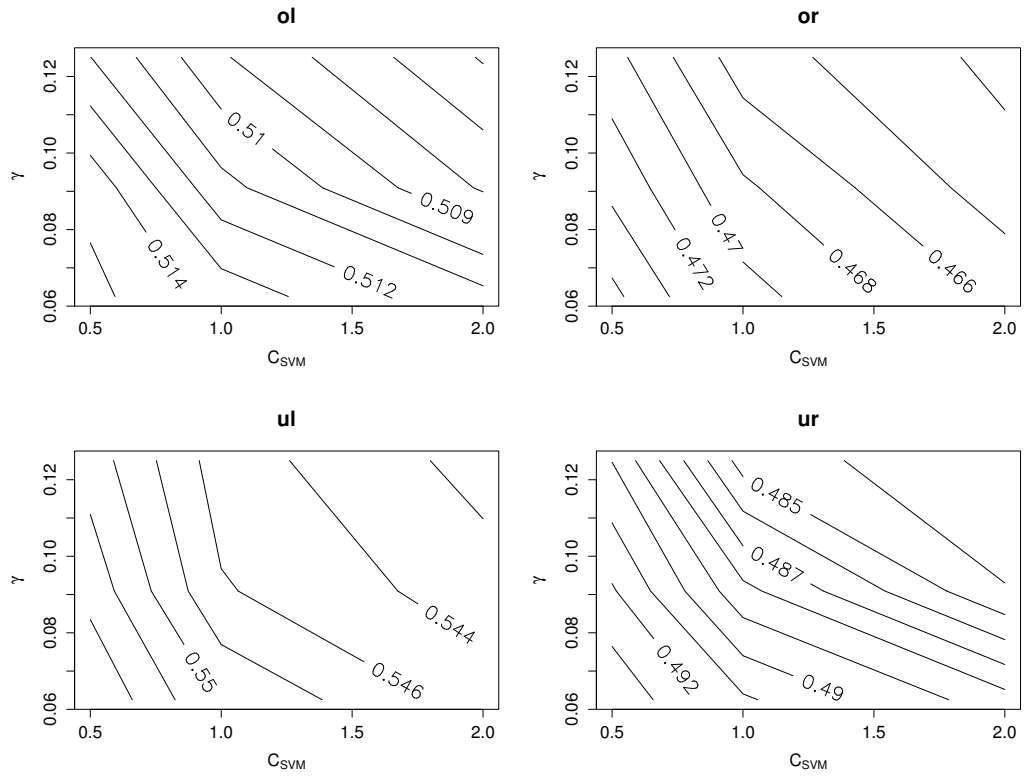


Abbildung 36: Contour-Grafik für die Ergebnisse des Parametertunings der Support Vector Machine bei Verwendung des Datensatzes BC

B R-Code eigene Funktionen

B.1 Segmentierung / Clustern

Partielles hierarchisches agglomeratives Clustern

```
##### Eigene Funktion für partielles hierarchisches agglomeratives
##### Clustern
### Input: ###
## data          Rasterdatensatz (brick)
## partsize      maximale Anzahl Pixel einer Subclusterung (numeric)
## sub           Auswahl der Subareale, "deter" oder "random"
## bord          Festlegungsart für die Gruppenzahl, "rel" oder "abs"
## rangemulti    relativer Grenzwert aus [0,1] für Einstellung
##               bord="rel" (numeric)
## gw            absoluter Grenzwert für bord="abs" (numeric)
## info          Angabe zum Fortschritt der Berechnung in Prozent
##               (logical)
### Output: ###
## clusters      Vektor mit Gruppenzugehörigkeit jedes Pixel (numeric)
## nsubgroups    Anzahl Gruppen in den Subarealen (numeric)
## nendgroup     Anzahl Gruppen insgesamt (numeric)
partial.hclust <-function(data,partsize,sub="deter",bord="rel"
,rangemulti,gw,info=FALSE){
# Berechnung der Anzahl einzeln zu betrachtender Subareale, deren
# Größe partsize nicht überschreitet
areanr <- ceiling(ncell(data)/partsize)
# Einteilung Pixelnummern in areanr Bereiche
border <- ceiling(seq(0,ncell(data),length=areanr+1))
subareas <- list()
if(sub=="deter"){
# Einteilung deterministisch von kleinen zu großen Pixelnummern
for(s in 1:areanr){
subareas[[s]] <- (border[s]+1):border[s+1]
}
```

```
}
if(sub=="random"){
# Einteilung per Zufall
samp <- sample(1:ncell(data),ncell(data))
for(s in 1:areanr){
subareas[[s]] <- samp[(border[s]+1):border[s+1]]
}
}
# Initialisierungen für unten {{
pixel.klumpen <- numeric(ncell(data))
zwischendatasammler <- matrix(ncol=nlayers(data),nrow=0)
pixelinfolist <- list()
ngrusammler <- numeric(areanr)
# }}
# in der folgenden Schleife wird für jedes area einzeln die Funktion
# hclust() ausgeführt
for(i in 1:areanr){
if(info){print(i/areanr)}
# Bestimmung Pixelnummern des Subareals
pixelnr <- subareas[[i]]
# Durchführung hclust und Bestimmung der Gruppenzahl{{
subcluster <- hclust(dist(data[pixelnr]),method="ward.D2")
if(bord=="abs"){
grenzwert <- gw
}
if(bord=="rel"){
grenzwert <-
(max(subcluster$height)-min(subcluster$height))*rangemulti
}
ngru <- length(which(subcluster$height>=grenzwert))+1
einteilung <- cutree(subcluster,ngru)
# }}
# Initialisierungen für unten{{
zwischendata <- matrix(NA,ncol=nlayers(data),nrow=ngru)
```

```
pixelinfo <- list()
# }}
# für jedes Cluster werden die Pixel einzeln bearbeitet
for(j in 1:ngru){
jpixel <- which(einteilung==j)
# Abspeicherung Pixelnummern der Gruppe
pixelinfo[[j]] <- pixelnr[jpixel]
# Mittelwertbildung der Pixel eines Clusters zur weiteren
# Verarbeitung
zwischendata[j,] <- apply(data[pixelnr[jpixel]],2,mean)
}
# Sammelbecken für die Infos aller Subareale
zwischendatasammler <- rbind(zwischendatasammler,zwischendata)
pixelinfolist <- c(pixelinfolist,pixelinfo)
ngrusammler[i] <- ngru
}
zwischendataframe <- as.data.frame(zwischendatasammler)
# erneute Clusterung der Gruppen aus den Subarealen anhand der
# gebildeten Mittelwerte der in den Gruppen befindlichen Pixel
endcluster <- hclust(dist(zwischendataframe),method="ward.D2")
if(bord=="abs"){
grenzwert <- gw
}
if(bord=="rel"){
grenzwert <-
(max(endcluster$height)-min(endcluster$height))*rangemulti
}
ngru.end <- length(which(endcluster$height>=grenzwert))+1
endcluster.klumpen <- cutree(endcluster,ngru.end)
# Gruppen der finalen Clusterung werden den Pixelnummern anhand der
# ersten Clusterung zugeordnet
for(j in 1:length(pixelinfolist)){
pixel.klumpen[pixelinfolist[[j]]] <- endcluster.klumpen[j]
}
```

```
return(list(clusters=pixel.klumpen,nsubgroups=ngrusammler
,nendgroup=ngru.end))
}
# Die Funktion bestimmt zunächst mit dem Parameter partsize aus data
# heraus die Anzahl der verwendeten Subareale. Auf diese wird dann
# jeweils die Funktion hclust() angewendet. Für die Gruppen aus jedem
# Subareal werden Mittelwerte der enthaltenen Datenpunkte gebildet.
# Auf diese Mittelwerte wird anschließend erneut die Funktion
# hclust() angewendet. Die daraus resultierenden Gruppen werden
# rückwirkend über die Mittelwerte den einzelnen Datenpunkten
# zugewiesen. Dadurch ist die finale Gruppenzugehörigkeit der Pixel
# ermittelt. Die Anzahl der Gruppen in jedem Schritt wird immer auf
# die gleiche Weise festgelegt. Die Art der Festlegung wird durch den
# Parameter bord bestimmt. Mit "abs" wird ein absoluter Grenzwert
# festgelegt. Bei dessen Überschreitung wird die Gruppierung des
# vorhergehenden Schrittes von hclust() verwendet. "rel"
# berücksichtigt die Distanzen in den einzelnen Schritten. Erreichen
# diese den Prozentsatz rangemulti der Spannweite der Distanzen, wird
# die vorhergehende Gruppierung verwendet. Der Parameter info ist für
# größere Datensätze gedacht und gibt an, wie viel Prozent der
# Subareale schon bearbeitet wurden.
```

Bildung von Segmenten anhand des Layers eines Bricks

```
##### Funktion zur Erstellung von Segmenten aus dem Layer eines
##### Bricks
### Input: ###
## data          Rasterdatensatz (brick)
## layernr       Nummer des Layers (numeric)
### Output: ###
## Segmente      Liste, deren Einträge jeweils die Pixelnummern
##               eines Segmentes bilden
## Segmentklasse Klassen der gebildeten Segmente
segmentsfromlayer <- function(data,layernr){
# Initialisierungen für unten {{
```

```
segmente <- list()
starter <- 1
segklasse <- numeric()
# }}
# Hier wird sichergestellt, dass die Klassenzugehörigkeit in
# numerischer Form vorliegt, da dies für die Folgeoperationen
# notwendig ist.
lev <- as.numeric(levels(as.factor(data[[layernr]][1:ncell(data)])))
# Bildung der Segmente für jede Klasse einzeln
for(m in lev){
# Erzeugung Vektor mit allen Pixelnummern aus Klasse m
alle <- which(data[[layernr]][1:ncell(data)]==m)
# Trennung zwischen nicht aufeinander folgenden Pixelnummern
cutpoint <- c(0,which(c(alle[-1],ncell(data)+2)-alle>1))
zeile <- list()
zeile.pos <- numeric()
# Abspeicherung der zusammengehörenden Pixel in zeile
for(i in 2:length(cutpoint)){
zeile[[i-1]] <- alle[(cutpoint[i-1]+1):cutpoint[i]]
zeile.pos[(cutpoint[i-1]+1):cutpoint[i]] <- i-1
}
anzeiger <- list()
for(i in 1:(length(cutpoint)-1)){
anzeiger[[i]] <- c(i)
}
for(j in 1:length(alle)){
# für jeden Pixel wird überprüft, ob der darunter liegende Pixel in
# der gleichen Klasse ist
drin <- alle[j]+ncol(data)==alle
if(any(drin)){
pos <- zeile.pos[which(drin)]
if(all(anzeiger[[zeile.pos[j]]]!=pos)){
# kombiniere Pixelnummern, falls Nummer des unteren Pixels noch nicht
# mit anderen zusammen ist
```

```
neu <- unique(c(zeile[[zeile.pos[j]]],zeile[[pos]]))
# Überschreibe alte Einträge der beteiligten Pixel {{
for(l in anzeiger[[pos]]){
zeile[[l]] <- neu
anzeiger[[l]] <- unique(c(anzeiger[[zeile.pos[j]]],anzeiger[[pos]]))
}
for(k in anzeiger[[zeile.pos[j]]]){
zeile[[k]] <- neu
anzeiger[[k]] <- unique(c(anzeiger[[zeile.pos[j]]],anzeiger[[pos]]))
}
# }}
}
}
}
# speichere zusammengehörige Pixel jeweils als Segment ab
segi <- unique(zeile)
for(p in 1:length(segi)){
print(p+starter-1)
segmente[[p+starter-1]] <- segi[[p]]
segklasse[p+starter-1] <- m
}
starter <- starter+length(segi)
}
return(list(Segmente=segmente,Segmentklasse=segklasse))
}
# Die Segmente werden aus dem Layer gebildet, indem zunächst Pixel
# mit hintereinander liegenden Nummern aus der gleichen Klasse
# identifiziert werden. Dies Pixel liegen nebeneinander und gehören
# zum gleichen Segment. Anschließend wird bei jedem Pixel einer
# Klasse überprüft, ob der darunter liegende Pixel zur gleichen
# Klasse gehört. Falls ja wird der untere mit dem oberen und
# eventuell zu diesem gehörigen weiteren Pixeln verbunden. Sind die
# Pixel bereits zu einem Segment verbunden geschieht nichts. Die
# beschriebene Prozedur wird getrennt nach Klassen durchgeführt.
```

NIBS-Distanz

```
##### Funktion zur Berechnung der NIBS-Distanz (not inside both
##### sequences)
### Input:  ###
## data          Rasterdatensatz (brick) als Ursprung der
##               Segmentierungen
## seg.t         Segmentierung 1, echte Segmente (list)
## seg.p         Segmentierung 2, vorhergesagte Segmente (list)
## error         zu zählender Fehler, "true" oder "predicted" oder
##               "both"
### Output: ###
## nibs          NIBS-Distanz
nibs <- function(data,seg.t,seg.p,error="both",info=FALSE){
# Initialisierung
score <- score.w <- 0
act.pixelnr <- 1:ncell(data)
# erzeuge Vektoren in denen das Segment jedes Pixels steht {{
pos.t <- pos.p <- numeric(ncell(data))
for(i in 1:length(seg.t)){
pos.t[seg.t[[i]]] <- i
}
for(i in 1:length(seg.p)){
pos.p[seg.p[[i]]] <- i
}
#}}
repeat{
pix <- act.pixelnr[1]
if(info){print(pix)}
# suche aus beiden Segmentlisten den Eintrag mit Pixel pix
tru.seg <- seg.t[[pos.t[pix]]]
pre.seg <- seg.p[[pos.p[pix]]]
# alle Pixelnummern die in beiden Segmenten vorkommen
schnitt <- intersect(tru.seg,pre.seg)
```

```
# bestimme und addiere Pixelanzahl je nach Wahl von error
if(error=="true"){
# zähle alle Elemente die im echten aber nicht im vorhergesagten
# Segment vorkommen
score <- score+length(tru.seg)-length(schnitt)
}
if(error=="predicted"){
# zähle alle Elemente die im vorhergesagten aber nicht im echten
# Segment vorkommen
score <- score+length(pre.seg)-length(schnitt)
}
if(error=="both"){
# zähle alle Elemente die nicht gleichzeitig im echten und
# vorhergesagten Segment vorkommen
score <- score+length(tru.seg)+length(pre.seg)-2*length(schnitt)
}
# Sortiere Pixel aus, die mit dem aktuellen Pixel zusammen in beiden
# Segmenten vorkommen. Keine mehrfachen Berechnungen für die gleichen
# Segmente.
get.out <- numeric()
for(s in schnitt){get.out <- c(get.out,which(act.pixelnr==s))}
act.pixelnr <- act.pixelnr[-get.out]
# nach Löschung der letzten Pixel wird die Schleife unterbrochen
if(length(act.pixelnr)==0) break
}
return(nibs=score)
}
# Zur Berechnung der NIBS-Distanz werden als Erstes die Sequenz-
# beziehungsweise die Segmentpaare gebildet. Dazu werden alle Pixel
# der Reihe nach durchgesehen und jeweils das Segment aus beiden
# Segmentierungen herausgesucht, welches diesen Pixel beinhaltet. Mit
# error="both" werden alle Pixel gezählt, die nur in genau einem
# Segment eines Segmentpaares erscheinen. Alternativ können die Pixel
# gezählt werden, die in einem aber nicht im anderen Segment zu
```

```
# finden sind. Dies sind die Optionen "true" und "predicted". Dabei
# werden Segmentierung 1 mit den echten Segmenten und Segmentierung 2
# mit den aus einem Clusterverfahren vorhergesagten Segmenten
# assoziert. Um mehrfache Berechnungen beim selben Segmentpaar zu
# vermeiden, werden aus der Menge der noch zu bearbeitenden Pixel
# diverse gelöscht. Dies umfasst alle Pixel die beim aktuellen Paar
# in beiden Segmenten vorhanden sind. Mit info=TRUE wird ausgegeben,
# bei welchem Pixel die Berechnung aktuell ist.
```

Erstellung Segmentdatensätze

```
##### Funktion zur Erstellung von Segmentdatensätzen
### Input: ###
## sat           Rasterdatensatz (brick)
## p.seg         Segmentierung mit gleicher Pixelzahl wie sat (list)
### Output: ###
## segdata       Datensatz mit Segmenten (data frame)
constructSegdata <- function(sat,p.seg){
# Initialisierung Datenmatrix
newdata <- matrix(NA,nrow=length(p.seg$Segmente),ncol=13)
# Bildung arithmetisches Mittel der Messwerte in 7 Spektralkanälen,
# Nutzung aller Pixel eines Segments,
# Abspeicherung als Segmentinfos
for(i in 1:length(p.seg$Segmente)){
akt.seg <- p.seg$Segmente[[i]]
if(length(akt.seg)>1){
newdata[i,1:7] <- apply(sat[akt.seg][,1:7],2,mean)
} else{newdata[i,1:7] <- sat[akt.seg][,1:7]}
# Berechnung Höhe und Breite eines Segmentes aus Koordinaten bei 30
# Einheiten Auflösung
xcodis <- sat[akt.seg][,8]
ycodis <- sat[akt.seg][,9]
newdata[i,8] <- (max(xcodis)-min(xcodis))/30+1
newdata[i,9] <- (max(ycodis)-min(ycodis))/30+1
# Anzahl Pixel im Segment
```

```
newdata[i,10] <- length(p.seg$Segmente[[i]])
# Standardabweichung Segmente
if(length(akt.seg)>1){
newdata[i,11] <- sum(apply(sat[akt.seg][,1:7],2,sd))
} else{newdata[i,11] <- 0}
# Mehrheit echte Klasse plus Wahrscheinlichkeit der Mehrheit
mehrheiten <- table(sat[akt.seg][,10])
newdata[i,13] <- as.numeric(names(mehrheiten)[which.max(mehrheiten)])
newdata[i,12] <-
as.numeric((mehrheiten/sum(mehrheiten))[which.max(mehrheiten)])
}
colnames(newdata) <-
c("Kanal_1","Kanal_2","Kanal_3","Kanal_4","Kanal_5","Kanal_6"
,"Kanal_7","Breite","Hoehe","Pixelzahl","Variabilität"
,"true_class.prob","true_class")
newdata <- as.data.frame(newdata)
return(segdata=newdata)
}
# Zur Bildung der Segmentdatensätze werden ein Satellitenbild in
# Form eines Bricks und eine Segmentierung in Form einer Liste
# benötigt. Beides muss die gleiche Anzahl Pixel beinhalten. Der
# Brick enthält in den ersten sieben Layern die Spektralkanäle. In
# den Layern 8 und 9 befinden sich die Koordinaten. Layer 10
# beinhaltet die echten Klassen. Die Liste mit den Segmenten besteht
# aus den Pixelnummern und kann mit der Funktion segmentsfromlayer()
# erstellt werden. Jedes Segment wird einzeln bearbeitet. Die
# Pixelwerte der Spektralkanäle eines Segments werden gemittelt und
# die Zusatzinformationen erhoben. Alle Daten werden in die Zeile
# einer Matrix geschrieben. Diese wird am Ende in einen Data Frame
# umgewandelt.
```

B.2 Klassifikation

Minimum-Distance-Verfahren (Modell)

```
##### Funktion zur Erstellung eines Modells für das
##### Minimum-Distance-Verfahren
### Input: ###
## data          Datensatz (data frame oder matrix)
## trainsubset   Vektor mit Pixelnummern der Trainingsdaten (numeric)
## empty.class   Handhabung "leere" Klassen, "infty" oder "overall"
### Output: ###
## means         Zentroide (list)
## data          Kopie des verwendeten Datensatzes (data frame)
## target        Name der Zielspalte (character)
mindist.mod <- function(data,trainsubset,empty.class="infty"){
# falls noch nicht geschehen, Umwandlung Daten in data frame
data <- as.data.frame(data)
# letzte Spalte wird als zu klassifizierende Variable festgelegt und
# ggf. in factor umgewandelt
target <- as.numeric(dim(data)[2])
data[,target] <- factor(data[,target],levels=c(1:14,16:19))
# Auswahl eines Teils der Daten zur Modellbildung
subdata <- data[trainsubset,]
# Berechnung Gesamtmittelwert für leere Klassen
overall.mean <- apply(subdata[,1:(target-1)],2,mean)
model.means <- list()
# Klassenweise Berechnung arith. Mittel
for(i in levels(data[,target])){
input <- which(subdata[,target]==i)
if(length(input)>0){
model.means[[i]]<- apply(subdata[input,1:(target-1)],2,mean)
}else{
# falls kein Objekt in der Klasse ist
if(empty.class=="overall"){model.means[[i]] <- overall.mean}
if(empty.class=="infty"){model.means[[i]] <- rep(Inf,dim(data)[2]-1)}
```

```
}
}
return(list(means=model.means,data=data
,target=colnames(data)[target]))
}
# Die Funktion erstellt ein Modell, mit dem anhand der zugehörigen
# Funktion mindist.predict() Vorhersagen der Klasse gemacht werden
# können. Zunächst werden aus dem Datensatz diejenigen Objekte
# herausgenommen, die zum Trainieren genutzt werden. Diese werden
# nach den Klassen der Zielvariable sortiert, die sich in der letzten
# Spalte befinden muss. Aus allen Objekten einer Klasse werden arith.
# Mittelwerte bestimmt und für diese Klasse abgespeichert. Für nicht
# vorhandene leere Klassen wird entweder der Mittelwert aller
# Trainingsdaten zugeordnet oder die Werte des Zentroiden werden auf
# Unendlich gesetzt. Für die Vorhersage werden zusätzlich zur Liste
# der Zentroide auch der komplette Datensatz und der Name der
# Zielspalte ausgegeben. Der gesamte Output bildet das Modell des
# Minimum-Distance-Verfahrens.
```

Minimum-Distance-Verfahren (Vorhersage)

```
##### Funktion zur Erstellung von Vorhersagen für das
##### Minimum-Distance-Verfahren
### Input: ###
## mindistmodel  Minimum-Distance-Modell (siehe mindist.mod())
## newdata       Daten, für die Vorhersagen erstellt werden sollen
### Output: ###
## class         Vorhersagen für Datenpunkte aus newdata (factor)
mindist.predict <- function(mindistmodel,newdata){
# Funktion zur Berechnung der Distanz eines Datenpunktes zu
# Zentroid j
mwbildung <- function(punkt){as.numeric(dist(rbind(newdata[punkt,]
,mindistmodel$means[[j]])))}
# Initialisierung einer Matrix mit Distanzen jedes Datenpunktes zu
# allen Zentroiden
```

```
pointdist <- matrix(nrow=dim(newdata)[1]
,ncol=length(mindistmodel$means))
# Zu jedem Zentroid j wird die Distanz zu allen Datenpunkten aus
# newdata bestimmt.
for(j in 1:length(mindistmodel$means)){
pointdist[,j] <- sapply(1:dim(newdata)[1],mwbildung)
}
# Bestimmung der minimalen Distanz für jeden Datenpunkt
# Zufallsauswahl bei Gleichstand
require(nnet)
predn r <- apply(-pointdist,1,which.is.max)
lev <- c(1:14,16:19)
pred <- lev[predn r]
return(class=factor(pred,levels=lev))
}
# Die euklidischen Distanzen aller zu klassifizierenden Datenpunkte
# zu jedem Zentroid werden berechnet. Die Zentroide werden dabei dem
# Eintrag means aus dem Modell entnommen, welches mit mindist.mod()
# erzeugt wurde. Als Datenpunkte können sowohl Teile des in der
# Modellbildung verwendeten Datensatzes genutzt oder gänzlich neue
# Daten verwendet werden. Die minimale Distanz eines Datenpunktes zu
# einem Zentroid führt zur Vorhersage der zugehörigen Klasse für
# diesen Punkt. Gibt es mehrere Minima wird zufällig zwischen den
# beteiligten Klassen gewählt. Für diese Zufallsauswahl wird die
# Funktion which.is.max() aus dem Paket nnet verwendet.
```

Minimum-Distance-Verfahren (Einbindung in `mlr`)

```
makeRLearner.classif.mindist = function() {
makeRLearnerClassif(
cl = "classif.mindist",
package=c("base","nnet"),
par.set = makeParamSet(
makeDiscreteLearnerParam(id = "empty.class", default = "infty"
, values = c("overall", "infty"))
```

```
),
properties = c("twoclass", "multiclass", "numerics"),
name = "Minimum-Distance-Classifier",
short.name = "mindist"
)
}

trainLearner.classif.mindist <- function(.learner, .task, .subset
, .weights = NULL,empty.class.par="infty", ...) {
mindist.mod(data = getTaskData(.task, .subset)
,empty.class=empty.class.par)
}

predictLearner.classif.mindist <- function(.learner, .model
, .newdata, ...) {
mindist.predict(.model$learner.model, newdata = .newdata ,...)
}
```

Maximum-Likelihood-Verfahren (Modell)

```
##### Funktion zur Erstellung eines Modells für das
##### Maximum-Likelihood-Verfahren
### Input: ###
## data          Datensatz (data frame oder matrix)
## trainsubset   Vektor mit Pixelnummern der Trainingsdaten (numeric)
## empty.class   Handhabung "leere" Klassen, "null" oder "overall"
## info          Informationsoption für spätere Vorhersage (logical)
### Output: ###
## apriori       empirische Häufigkeit der Klassen in den
##               Trainingsdaten (list)
## means         Zentroide (list)
## covmats       Kovarianzmatrizen (list)
## data          Kopie des verwendeten Datensatzes (data frame)
## target        Name der Zielspalte (character)
## info          Abspeicherung info-Einstellung im Modell (logical)
```

```
maxlike.mod <- function(data,trainsubset,empty.class="null"
,info=FALSE){
# falls noch nicht geschehen, Umwandlung Daten in data frame
data <- as.data.frame(data)
# letzte Spalte wird als zu klassifizierende Variable festgelegt und
# ggf. in factor umgewandelt
target <- as.numeric(dim(data)[2])
data[,target] <- factor(data[,target],levels=c(1:14,16:19))
# Auswahl eines Teils der Daten zur Modellbildung
subdata <- data[trainsubset,]
# Berechnung empirische Häufigkeiten der Klassen
apriori <- as.list(table(subdata[,target])/length(subdata[,target]))
# Berechnung Gesamtmittelwert und Gesamtkovarianz für leere Klassen
overall.mean <- apply(subdata[,1:(target-1)],2,mean)
overall.covmat <- cov(subdata[,-target])
model.means <- list()
model.covmat <- list()
for(i in levels(data[,target])){
# Klassenweise Berechnung Mittelwerte und Kovarianzmatrizen
input <- which(subdata[,target]==i)
inputdata <- subdata[input,-target]
if(length(input)>0){
model.means[[i]] <- apply(inputdata,2,mean)
model.covmat[[i]] <- cov(inputdata)
}else{
if(empty.class=="overall"){
model.means[[i]] <- overall.mean
model.covmat[[i]] <- overall.covmat
}
if(empty.class=="null"){
model.means[[i]] <- "leer"
model.covmat[[i]] <- "leer"
}
}
```

```
}
return(list(apriori=apriori,means=model.means,covmats=model.covmat
,data=data,target=colnames(data)[target],info=info))
}
# Obige Funktion erstellt ein Modell, durch das mit der Funktion
# maxlike.predict() Vorhersagen für die Klassen getätigt werden. Ein
# Teil der Daten werden dazu eventuell als Trainingsdaten ausgewählt.
# Mit diesen werden dann die Schätzer für a-priori-
# Wahrscheinlichkeiten, Zentroide und Kovarianzmatrizen der einzelnen
# Klassen bestimmt. Die anderen Ausgaben der Funktion sind der
# Datensatz, der Name der Zielvariable und die Einstellung der info-
# Option. Diese drei Angaben werden für die Vorhersage in das Modell
# aufgenommen.
```

Maximum-Likelihood-Verfahren (Vorhersage)

```
##### Funktion zur Erstellung von Vorhersagen für das
##### Maximum-Likelihood-Verfahren
### Input: ###
## maxlikemodel  Maximum-Likelihood-Modell (siehe maxlike.mod())
## newdata       Daten, für die Vorhersagen erstellt werden sollen
### Output: ###
## class         Vorhersagen für Datenpunkte aus newdata (factor)
maxlike.predict <- function(maxlikemodel,newdata){
# Für die Berechnung der Wahrscheinlichkeiten wird das Paket mvtnorm
# verwendet
require(mvtnorm)
# Berechnung Wahrscheinlichkeiten für einzelne Klassen
predwktmat <-
matrix(NA,ncol=length(maxlikemodel$means),nrow=dim(newdata)[1])
for(i in 1:length(maxlikemodel$means)){
if(maxlikemodel$info){print(i)}
# Leere Klassen erhalten Wahrscheinlichkeiten 0 (empty.class="null")
if(any(maxlikemodel$means[[i]]=="leer")){
predwktmat[,i] <- rep(0,dim(newdata)[1])
```

```
}else{
predwktmat[,i] <- maxlikemodel$apriori[[i]]*dmvnorm(newdata
,mean=maxlikemodel$means[[i]],sigma=maxlikemodel$covmats[[i]])
}
}
# Die Klasse mit der höchsten Wahrscheinlichkeit wird vorhergesagt
require(nnet)
pred.ohne <- apply(predwktmat,1,which.is.max)
lev <- c(1:14,16:19)
pred <- lev[pred.ohne]
return(class=factor(pred,levels=lev))
}
# Anhand eines durch die Funktion maxlike.mod() erzeugten Modells
# wird für jede Klasse die angenommene multivariate Normalverteilung
# bestimmt. Die Berechnung der Wahrscheinlichkeiten dieser
# Verteilungen erfolgt für alle Klassen einzeln und unter Nutzung des
# Paketes mvtnorm. Die Klasse mit dem höchsten Wert wird vorhergesagt.
# Bei mehreren Maxima wird zufällig eine der Klassen gewählt. Dazu
# wird which.is.max() aus dem Paket nnet verwendet.
```

Maximum-Likelihood-Verfahren (Einbindung in `mlr`)

```
makeRLearner.classif.maxlike = function() {
makeRLearnerClassif(
cl = "classif.maxlike",
package=c("base","mvtnorm","nnet"),
par.set = makeParamSet(
makeDiscreteLearnerParam(id = "empty.class", default = "null"
, values = c("null", "overall")),
makeLogicalLearnerParam(id = "info", default = FALSE
, tunable = FALSE)
),
properties = c("twoclass", "multiclass", "numerics"),
name = "Maximum-Likelihood-Classifier",
short.name = "ml"
```

```
)
}

trainLearner.classif.maxlike <- function(.learner, .task, .subset
, .weights = NULL,empty.class.par="null", info.dec=FALSE, ...) {
maxlike.mod(data = getTaskData(.task, .subset)
,empty.class=empty.class.par,info=info.dec)
}

predictLearner.classif.maxlike <- function(.learner, .model
, .newdata, ...) {
maxlike.predict(.model$learner.model, newdata = .newdata , ...)
}
```

Quader-Verfahren (Modell)

```
##### Funktion zur Erstellung eines Modells für das Quader-Verfahren
### Input: ###
## data          Datensatz (data frame oder matrix)
## trainsubset   Vektor mit Pixelnummern der Trainingsdaten (numeric)
## empty.class   Handhabung "leere" Klassen, "none" oder "all"
## boxmethod     Bestimmungsart für die Grenzen der Quader, "range"
##               oder "sigma"
## sigmapar      Vielfaches, bei Wahl von boxmethod="sigma" (numeric)
## tiemethod     Art der Vorhersage, falls Objekt nicht in genau
##               einem Quader, "mind" oder "ml"
## cases         Anzeige in wievielen Quadern die Punkte jeweils
##               liegen (logical)
## info          Informationsoption für spätere Vorhersage (logical)
### Output: ###
## boxes         Grenzen der Quader (list)
## means         Zentroide (list)
## covmats       Kovarianzmatrizen (list)
## data          Kopie des verwendeten Datensatzes
## target        Name der Zielspalte (character)
```

```
## tiemethod     Abspeicherung tiemethod-Einstellung im Modell (s.o.)
## cases         Abspeicherung cases-Einstellung im Modell (logical)
## info          Abspeicherung info-Einstellung im Modell (logical)
quader.mod <- function(data,trainsubset,empty.class="none"
,boxmethod="range",sigmapar=3,tiemethod="mind",cases=FALSE
,info=FALSE){
# falls noch nicht geschehen, Umwandlung Daten in data frame
data <- as.data.frame(data)
# letzte Spalte wird als zu klassifizierende Variable festgelegt und
# ggf. in factor umgewandelt
target <- as.numeric(dim(data)[2])
data[,target] <- factor(data[,target],levels=c(1:14,16:19))
# Auswahl eines Teils der Daten zur Modellbildung
subdata <- data[trainsubset,]
# Berechnung Gesamtmittelwert und Gesamtkovarianz für leere Klassen
overall.mean <- apply(subdata[,1:(target-1)],2,mean)
overall.covmat <- cov(subdata[,-target])
model.means <- list()
model.boxes <- list()
model.covmat <- list()
for(i in levels(data[,target])){
input <- which(subdata[,target]==i)
inputdata <- subdata[input,-target]
if(length(input)>0){
# Berechnung Mittelwerte und Kovarianzen für jede Klasse
model.means[[i]]<- apply(inputdata,2,mean)
model.covmat[[i]] <- cov(inputdata)
# Bestimmung der Ober- und Untergrenze des Quaders für jede Klasse
if(boxmethod=="range"){model.boxes[[i]] <- apply(inputdata,2,range)}
if(boxmethod=="sigma"){
model.boxes[[i]] <- rbind(
model.means[[i]]-sigmapar*sqrt(diag(model.covmat[[i]])),
model.means[[i]]+sigmapar*sqrt(diag(model.covmat[[i]])))
}
```

```
}else{
# Berechnungen für leere Klassen
if(empty.class=="all"){
model.boxes[[i]] <- matrix(c(rep(-Inf,dim(subdata)[2])
,rep(Inf,dim(subdata)[2])),nrow=2,byrow=T)
model.means[[i]] <- overall.mean
model.covmat[[i]] <- overall.covmat
}
if(empty.class=="none"){
model.boxes[[i]] <- matrix(c(rep(Inf,dim(subdata)[2])
,rep(-Inf,dim(subdata)[2])),nrow=2,byrow=T)
model.means[[i]] <- rep(Inf,dim(data)[2]-1)
model.covmat[[i]] <- "leer"
}
}
}
return(list(boxes=model.boxes,means=model.means,covmats=model.covmat
,data=data,target=colnames(data)[target],tiemethod=tiemethod
,cases=cases,info=info))
}
# Zunächst werden ggf. aus dem Datensatz Trainingsdaten ausgewählt.
# Für jede Klasse werden dann Mittelwerte, Kovarianzen und Grenzwerte
# der Quader ermittelt. Mit "range" werden der minmale und maximale
# Wert einer Klasse in jeder Dimension ermittelt. Mit "sigma" wird zu
# den berechneten Mittelwerten ein Vielfaches der Standardabweichung
# hinzu addiert oder subtrahiert. Das Vielfache kann mit sigmapar
# eingestellt werden. Falls empty.class="all" gewählt wird, bestehen
# die Quader der leeren Klassen aus dem gesamten Messraum. Jeder
# zulässige Datenpunkt liegt also in diesem Quader. Die zugehörige
# Klasse steht somit immer als Wahlmöglichkeit zur Verfügung. Für
# empty.class="none" werden Vorkehrungen getroffen um analog zum
# Minimum-Distance- bzw. Maximum-Likelihood-Verfahren vorgehen zu
# können. Die Einstellungen von tiemethod, cases und info werden für
# die Vorhersagefunktion mit in das Modell aufgenommen.
```

Quader-Verfahren (Vorhersage)

```
##### Funktion zur Erstellung von Vorhersagen für das
##### Quader-Verfahren
### Input: ###
## quadermodel   Quader-Modell (siehe quader.mod())
## newdata       Daten, für die Vorhersagen erstellt werden sollen
### Output: ###
## class         Vorhersagen für Datenpunkte aus newdata (factor)
quader.predict <- function(quadermodel,newdata){
fall <- character()
pred <- numeric()
# Überprüfe in welchen Quadern die neuen Datenpunkte liegen
boxmat <- matrix(NA,nrow=dim(newdata)[1]
,ncol=length(quadermodel$boxes))
for(i in 1:length(quadermodel$boxes)){
if(quadermodel$info){print(i)}
boxcheck <- function(b){all(newdata[b,]>=quadermodel$boxes[[i]][1,])&
all(newdata[b,]<=quadermodel$boxes[[i]][2,])}
boxmat[,i] <- sapply(1:dim(newdata)[1],boxcheck)
}
# Liste mit Quadernummern für jeden zu klassifizierenden Datenpunkt
boxlist <- apply(boxmat,1,which)
# Erstelle Vorhersage je nach Fall durch Funktion in Abhängigkeit
# vom Datenpunkt
fallfkt <- function(j){
# in welchen Quadern liegt j?
boxnr <- boxlist[[j]]
# Fall 1: Punkt liegt in genau einem Quader
if(length(boxnr)==1){
fall <- "1"
pred <- boxnr
}
require(nnet)
```

```
if(quadermodel$tiemethod=="mind"){
# Fall 2: Punkt liegt in mehreren Quadern
if(length(boxnr)>1){
# Bestimme Vorhersage mit mindist unter allen Quadern, in denen der
# Punkt liegt
boxdist <- numeric()
for(k in boxnr){
boxdist[k] <- as.numeric(dist(rbind(newdata[j,]
,quadermodel$means[[k]])))
}
bestbox <- which(boxdist==min(boxdist,na.rm=TRUE))
# falls mehrere Zentren gleich nah am Punkt (z.b. nicht vorhandene
# Klassen), ziehe zufällig eine davon, setze seed für
# Reproduzierbarkeit
if(length(bestbox)>1){
fall <- "2a"
pred <- sample(bestbox,1)
}else{
fall <- "2a"
pred <- bestbox}
}
# Fall 3: Punkt liegt in keinem Quader
if(length(boxnr)==0){
# Bestimme Vorhersage mit mindist aus allen Quadern
boxdist3 <- numeric()
for(l in 1:length(quadermodel$means)){
boxdist3[l] <- as.numeric(dist(rbind(newdata[j,]
,quadermodel$means[[l]])))
}
bestbox3 <- which(boxdist3==min(boxdist3,na.rm=TRUE))
# falls mehrere Zentren gleich nah am Punkt (z.b. nicht vorhandene
# Klassen), ziehe zufällig eine davon
if(length(bestbox3)>1){
fall <- "3a"
```

```
pred <- sample(bestbox3,1)
}else{
fall <- "3a"
pred <- bestbox3}
}
}
if(quadermodel$tiemethod=="ml"){
require(mvtnorm)
# Fall 2: Punkt liegt in mehreren Quadern
if(length(boxnr)>1){
# Bestimme Vorhersage mit maxlike unter allen Quadern, in denen der
# Punkt liegt
fall <- "2b"
predwkt <- rep(0,18)
for(i in boxnr){
if(any(quadermodel$covmats[[i]]=="leer")){
predwkt[i] <- 0
}else{
predwkt[i] <- dmvnorm(newdata[j,],mean=quadermodel$means[[i]]
,sigma=quadermodel$covmats[[i]])
}
}
pred <- which.is.max(predwkt)
}
# Fall 3: Punkt liegt in keinem Quader
if(length(boxnr)==0){
# Bestimme Vorhersage mit maxlike aus allen Quadern
fall <- "3b"
predwkt <- numeric()
for(i in 1:length(quadermodel$means)){
if(any(quadermodel$covmats[[i]]=="leer")){
predwkt[i] <- 0
}else{
predwkt[i] <- dmvnorm(newdata[j,],mean=quadermodel$means[[i]]
```

```
,sigma=quadermodel$covmats[[i]])
}
}
pred <- which.is.max(predwkt)
}
}
return(list(pred,fall))
}
# wende obige Funktion auf alle neuen Datenpunkte an
predi <- lapply(1:length(boxlist),fallfkt)
# Vorhersage und Fall werden jeweils als Elemente einer Liste
# abgespeichert und einzeln ausgelesen
for(i in 1:length(predi)){pred[i] <- predi[[i]][[1]]
fall[i] <- predi[[i]][[2]]}
lev <- c(1:14,16:19)
pred.char <- lev[pred]
# falls cases=TRUE, werden Fälle mit ausgegeben, sonst nur
# Vorhersagen
if(quadermodel$cases){return(list(class=factor(pred.char,levels=lev)
,cases=fall))
}else{return(class=factor(pred.char,levels=lev))}
}
# Zunächst wird geprüft, in welchen Quadern die zu klassifizierenden
# Objekte jeweils enthalten sind. Um eine Vorhersage zu erstellen,
# gilt es dann drei Fälle zu unterscheiden. Im einfachsten Fall, dass
# der Punkt in genau einem Quader liegt, wird einfach zu zugehörige
# Klasse als Vorhersage zugewiesen. Bei den anderen beiden Fällen
# wird jeweils nach dem Minimum-Distance oder Maximum-Likelihood-
# Verfahren entschieden. Liegt das Objekt in mehreren Quadern, werden
# nur diese für die weiteren Berechnungen genutzt (Fälle 2a und 2b).
# Ist das Objekt in keinem Quader (Fälle 3a und 3b) werden die
# Berechnungen mit allen Klassen fortgesetzt. Diese
# Fallunterscheidung wird für jeden Datenpunkt einzeln durchgeführt.
# Die Elemente der resultierenden Liste beinhalten die Angabe der
```

```
# Vorhersage und des Falles. Mit cases=TRUE wird beides ausgegeben.
# Die Voreinstellung ist jedoch cases=FALSE mit der nur die
# Vorhersagen ausgegeben werden. Die Funktion würde sonst in mlr
# nicht funktionieren, da vorgeschrieben ist, dass eine
# Vorhersagefunktion nur einen Vector des Formats factor ausgeben
# darf. Die Option cases kann jedoch bei einfacher Nutzung der
# Modell- und Vorhersagefunktion verwendet werden.
```

Quader-Verfahren (Einbindung in `mlr`)

```
makeRLearner.classif.quader = function() {
makeRLearnerClassif(
cl = "classif.quader",
package=c("base","mvtnorm","nnet"),
par.set = makeParamSet(
makeDiscreteLearnerParam(id = "boxmethod", default = "range"
, values = c("range", "sigma")),
makeNumericLearnerParam(id = "sigmapar", default = 3, lower = 0),
makeDiscreteLearnerParam(id = "empty.class", default = "none"
, values = c("all", "none")),
makeLogicalLearnerParam(id ="info",default=FALSE,tunable=FALSE),
makeLogicalLearnerParam(id="cases",default=FALSE,tunable=FALSE),
makeDiscreteLearnerParam(id = "tiemethod", default = "mind"
, values = c("mind", "ml"))
),
properties = c("twoclass", "multiclass", "numerics"),
name = "Quader-Classifier",
short.name = "quader"
)
}


trainLearner.classif.quader <- function(.learner, .task, .subset
, .weights = NULL,boxmethod.dec="range",sigmapar.dec=3
, empty.class.pro="none",info.dec=FALSE,tiemethod.dec="mind"
, cases.dec=FALSE) {
```

```
quader.mod(data = getTaskData(.task, .subset),boxmethod=boxmethod.dec
,sigmapar=sigmapar.dec,empty.class=empty.class.pro,info=info.dec
,tiemethod = tiemethod.dec,cases=cases.dec)
}

predictLearner.classif.quader <- function(.learner, .model
, .newdata, ...) {
quader.predict(.model$learner.model, newdata = .newdata , ...)
}
```

C Abkürzungen

Im Folgenden werden einige häufig verwendete Abkürzungen aufgelistet und näher erläutert.

bagclust	Bagged Clustering
bzw. BC	bagclust wird überwiegend im Fließtext verwendet und bezieht sich zumeist auf das Verfahren selbst sowie die daraus erzeugten Segmente und den Datensatz. Die Abkürzung BC wird analog verwendet, jedoch größtenteils in Abbildungen und Tabellen mit wenig Platz.
clara	clara-Algorithmus
bzw. CL	clara wird überwiegend im Fließtext verwendet und bezieht sich zumeist auf das Verfahren selbst sowie die daraus erzeugten Segmente und den Datensatz. Die Abkürzung CL wird analog verwendet, jedoch größtenteils in Abbildungen und Tabellen mit wenig Platz.
k-means	k-means-Algorithmus
bzw. KM	k-means wird überwiegend im Fließtext verwendet und bezieht sich zumeist auf das Verfahren selbst sowie die daraus erzeugten Segmente und den Datensatz. Die Abkürzung KM wird analog verwendet, jedoch größtenteils in Abbildungen und Tabellen mit wenig Platz.
LDA	Lineare Diskriminanzanalyse
MaxLike	Maximum-Likelihood-Verfahren
MinDist	Minimum-Distance-Verfahren
mmce	Mittlere Fehlklassifikationsrate
ol	oberes linkes Viertel des Satellitenbildes
or	oberes rechtes Viertel des Satellitenbildes
phclust	Partielles hierarchisches agglomeratives Clustern
bzw. PH	phclust wird überwiegend im Fließtext verwendet und bezieht sich zumeist auf das Verfahren selbst sowie die daraus erzeugten Segmente und den Datensatz. Die Abkürzung PH wird analog verwendet, jedoch größtenteils in Abbildungen und Tabellen mit wenig Platz.
SVM	Support Vector Machine
ul	unteres linkes Viertel des Satellitenbildes
ur	unteres rechtes Viertel des Satellitenbildes

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

[1] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Zachary Jones and Giuseppe Casalicchio (2016). mlr: Machine Learning in R. R package version 2.9. https://CRAN.R-project.org/package=mlr

[2] Czado, C., Schmidt, T. (2011). *Mathematische Statistik.* 1. Auflage, Springer, Heidelberg

[3] Deitmar, A. (2017). *Analysis.* 2. Auflage, Springer, Berlin

[4] Deutschen Zentrum für Luft- und Raumfahrt (DLR) (2007). *Klassifizierungsschlüssel und Lookup-Tabelle für CLC2000 Rasterdaten.* Köln http://corine.dfd.dlr.de/media/download/clc_lut_de.pdf (Abruf: 15.10.2016)

[5] Fahrmeir, L., Hamerle, A., Tutz, G. (1996). *Multivariate statistische Verfahren.* 2. überarbeitete Auflage, de Gruyter, Berlin

[6] Friedman, J.H., Tibshirani, R., Hastie, T. (2009). *The elements of statistical learning. Data mining, inference, and prediction.* 2. Auflage, Springer, New York

[7] Alan Genz, Frank Bretz, Tetsuhisa Miwa, Xuefei Mi, Friedrich Leisch, Fabian Scheipl, Torsten Hothorn (2016). mvtnorm: Multivariate Normal and t Distributions. R package version 1.0-5. URL http://CRAN.R-project.org/package=mvtnorm

[8] Haralick, R.M., Shapiro, L.G. (1985). Image Segmentation Techniques. *Computer Vision, Graphics and Image Processing* **29**(1), 100-132

[9] Hartigan, J.A., Wong, M.A. (1979). Algorithm AS 136. A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **28**(1), 100-108

[10] Hastie, T., Tibshirani, R. (1998). Classification by pairwise coupling. *The Annals of Statistics* **26**(2), 451-471

[11] Robert J. Hijmans (2015). raster: Geographic Data Analysis and Modeling. R package version 2.5-2. http://CRAN.R-project.org/package=raster

[12] Hsu, C.W., Chang, C.C., Lin, C.J. (2003). *A Practical Guide to Support Vector Classification.* Technical report, Department of Computer Science, National Taiwan University, Taipei, Taiwan, http://www.csie.ntu.edu.tw/∼cjlin/papers/guide/guide.pdf (Abruf: 16.11.2016)

[13] Kaufman, L., Rousseeuw, P.J. (2005). *Finding Groups in Data. An Introduction to Cluster Analysis.* 2. Auflage, Wiley, Hoboken

[14] Landsat 8 (L8) Data User Handbook (2016). U.S. Geological Survey (USGS) Landsat Project Science Office, Earth Resources Observation and Science (EROS) Center, Sioux Falls https://landsat.usgs.gov/sites/default/files/documents/Landsat8DataUsersHandbook.pdf (Abruf: 12.10.2016)

[15] Lange, N. (2013). *Geoinformatik in Theorie und Praxis.* 3. Auflage, Springer, Berlin

[16] Leisch, F. (1999) Bagged clustering. Working Paper 51, SFB "Adaptive Information Systems and Modeling in Economics and Management Science", Vienna University of Economics and Business Administration, Wien http://epub.wu.ac.at/1272/1/document.pdf

[17] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

[18] Lillesand, T.M., Kiefer, R.W., Chipman, J.W. (2008). *Remote sensing and image interpretation.* 6. Auflage, Wiley, Hoboken

[19] Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K. (2015). cluster: Cluster Analysis Basics and Extensions. R package version 2.0.3.

[20] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2015). e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.6-7. http://CRAN.R-project.org/package=e1071

[21] Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys* **33**(1), 31-88

[22] Nischwitz, A., Fischer, M., Haberäcker, P., Socher, G. (2011). *Computergrafik und Bildverarbeitung. Band II: Bildverarbeitung.*

3. Auflage, Vieweg + Teubner, Wiesbaden

[23] Papageorgiou, M. (1996). *Optimierung. Statische, dynamische, stochastische Verfahren für die Anwendung.* 2. Auflage, Oldenbourg, München

[24] R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[25] Ripley, B.D. (1996). *Pattern recognition and neural networks.* 1. Auflage, Cambridge University Press, Cambridge

[26] Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

[27] Witten, I.H., Frank, E., Hall, M.A. (2011). *Data mining. Practical machine learning tools and techniques.* 3. Auflage, Elsevier, Amsterdam

Danksagung

An dieser Stelle möchte ich mich bei Prof. Dr. Christine Müller für ihre Unterstützung in statistischer Hinsicht sowie zur Erstellung dieser Masterarbeit bedanken. Mein Dank gilt auch Prof. Dr. Nguyen Thinh, der mich als Raumplanungsprofessor im Bereich der Fernerkundung unterstützt hat. Ebenfalls bedanken möchte ich mich bei Frau Haniyeh Ebrahimi für die Bereitstellung der Daten und hilfreiche Tipps zu deren Einarbeitung in `R`.