Institut für Informatik
AG Künstliche Intelligenz

„Entwicklung starker Klassifikatoren zur Zeichenerkennung“

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

eingereicht von

Marcus Lindner
(Matrikelnummer: 3897648)

bei
Prof. Dr. Raúl Rojas
Prof. Dr. Marco Block-Berlitz

Abgabedatum: 16.06.2011

Eidesstattliche Erklärung

Ich versichere, die Masterarbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die diesen Quellen und Hilfsmitteln wörtlich oder sinngemäß entnommenen Ausführungen als solche kenntlich gemacht habe.

Berlin, den 16.06.2011 Unterschrift:

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das menschliche Gehirn lernt die Welt mit allen fünf Sinnen kennen [1]. Es nimmt die Eigenschaften verschiedener Dinge wahr, sortiert nach Beispielen gleicher Klassen und abstrahiert anschließend. Dieses Prinzip lässt sich mit folgendem Satz zusammenfassen: „Ein Objekt ist die Summe seiner Eigenschaften“. Man kann das als eine Art natürliches Grundprinzip der Wahrnehmung bezeichnen, was sich in allen Bereichen des Lebens wiederfindet. Letztlich lassen sich nur die Eigenschaften eines Objektes wahrnehmen, nicht aber das Objekt selbst und anhand der erlernten Kombinationen von Eigenschaften das Objekt identifizieren. So werden beispielsweise in den Wissenschaften Modelle durch Definitionen aufgestellt, die alle Eigenschaften der Modellobjekte aufzählen und deren Zusammenhang beschreiben.

In dieser Masterarbeit wird ein Verfahren zur Zeichenerkennung gedruckter lateinischer Buchstaben und arabischer Ziffern vorgestellt, das jede Zeichenklasse anhand ihrer individuellen Eigenschaften erlernt. Dafür wird eine Menge von Merkmalen definiert, die die Eigenschaften von Zeichen im Pixelbild beschreiben. Nach geeigneter Vorverarbeitung werden alle Merkmale extrahiert und in einem spezialisierten Entscheidungsbaum, dem Bewertungsbaum, zur Identifikation der Klassen kombiniert. Dieser Baum trifft keine Entscheidung, sondern gibt eine Wertung des Ergebnisses, die zur Entscheidung herangezogen wird. Auf diese Weise wird jede Zeichenklasse als Summe ihrer Merkmale abgebildet.

1.1 Training ohne Gegenbeispiele

Ziel dieser Arbeit ist es ein Verfahren zu entwickeln, mit dem sich Klassen von Objekten nur durch die Beschreibung der Objekte lernen lassen. Beschrieben werden die Objekte durch problemorientierte Merkmale und es sind verschiedene Merkmale mit unterschiedlicher Häufigkeit je Eingabebild möglich. Jeder Merkmalstyp muss sich durch

einen Merkmalsvektor im entsprechenden Merkmalsraum angeben lassen. Jede Objektklasse ist dann gekennzeichnet durch eine individuelle Zusammensetzung von Merkmalen verschiedenen Typs und unterschiedlicher Anzahl. Das Verfahren berücksichtigt, dass bei Exemplaren einer Objektklasse nicht immer alle Merkmale mit einer genau bestimmten Häufigkeit auftreten. Dafür ist eine gewisse Redundanz in den Merkmalen notwendig. Würde man beispielsweise eine 8 nur durch zwei Kreise, die sich jeweils in der Mitte der oberen und unteren Bildhälfte befinden beschreiben, dann wäre es nicht möglich Achten zu erkennen, deren Kreise eingerissen sind.

Ein großer Vorteil der Vorgehensweise die Beschreibung der Objekte zu lernen ist, dass dabei nicht nur die notwendigsten Unterschiede zwischen den Zeichen gelernt werden, um einen hohen Erkennungsgrad zu erreichen. Verfahren wie [2], die Objekterkennung durch Erlernen von Unterschieden in den Trainingsbeispielen bewerkstelligen, sind darauf angewiesen eine große Auswahl ein Negativ-Beispielen präsentiert zu bekommen. Das in dieser Arbeit vorgestellte Verfahren benötigt keine Negativ-Beispiele, um Zeichen von Nicht-Zeichen zu unterscheiden. Das Verfahren macht sich zu nutze, dass allein durch die Beschreibung einer Objektklasse der Rest der Welt implizit bekannt ist: „Alles, was nicht wie ein bekanntes Objekt aussieht, ist ein unbekanntes Objekt“. Um ein Objekt zu erkennen, muss man es also kennen. Der Ansatz ist dadurch motiviert, dass wir Menschen in der Lage sind ein neues Objekt kennenzulernen, ohne das man uns dafür eine Reihe falscher Muster zeigt. Speziell bei Schrift ist das eben auch der Fall.

Anders als bei Verfahren mit universellen Merkmalen [3], erfordert die hier vorgestellte Methode problemorientierte Merkmale. Es ist nicht möglich ein universelles Merkmal zu verwenden, mit dem beliebige Objektklassen erlernt werden können. Aber es ist durchaus möglich, dass die Merkmalstypen für Schrift auch andere Muster beschreiben können oder zumindest eine Teilmenge dieser Merkmale auch in der Beschreibung anderer Muster Verwendung finden kann.

1.2 Bezug zu verwandten Arbeiten

In [4] wird eine Methode vorgeschlagen, bei der mit einem rekursiven k-Means Algorithmus ein Vokabelbaum erstellt wird. Auf diese Weise werden Merkmalsvektoren, die in unterschiedlicher Anzahl je Bild vorkommen dürfen, in einen einheitlichen Bewertungsvektor transformiert, der für den Ähnlichkeitsvergleich und damit die Erkennungsentscheidung herangezogen wird. In der Arbeit wurde gezeigt, dass für gute Ergebnisse sehr viele Merkmalsvektoren von jedem Bild benötigt werden. Da diese Merkmalsvektoren aber vom gleichen Typ sein müssen ist das in der genannten Größenordnung auf Binär-

bildern mit sehr beschränkter Größe, wie sie hier zum Einsatz kommen, nicht möglich. Für den hier verwendeten Ansatz von Multi-Merkmalen ist der einfache Vokabelbaum außerdem nicht geeignet. In [5, 6] wird dafür eine Erweiterung mit einem Vokabelbaum für jedes Merkmal vorgeschlagen. In [7, 8] werden Methoden vorgestellt, die verschiedene Merkmalstypen für die Klassifikation heranziehen, aber nicht erlauben, dass Vektoren eines Typs in unterschiedlicher Anzahl je Bild vorkommen.

k-NN Algorithmen sind eine Standardklasse zum Klassifizieren von Objekten [9, 10] und finden auch häufig Anwendung in der Zeichenerkennung [11, 12]. Neben den Nachteilen von Geschwindigkeit und Speicher, die in einigen Arbeiten deutlich verbessert wurden, bietet dieser Ansatz nicht die Möglichkeit die Behauptung der Objektspezifikation durch Eigenschaftskomposition zu überprüfen.

In [13] wird eine Methode zum Binarisieren und Segmentieren von Text in Bildern vorgestellt. Das Ergebnis ist nahezu frei von false-negatives[1], während noch einige false-positives[2] zu finden sind. Um den Anteil falscher Zusammenhangskomponenten im Hinblick auf Texterkennung zu minimieren, lassen sich mit dem vorgestellten Verfahren dieser Masterarbeit die false-positives minimieren.

1.3 Aufbau der Arbeit

Ausgehend von bereits digitalisierten und binarisierten Druckzeichen (Buchstaben und Ziffern), die als einzelne Zusammenhangskomponenten bereitgestellt sind, werden für diese Klasse von Objekten in Kapitel 2 Merkmale definiert. Die Merkmale sind problemorientiert und speziell auf die Eigenschaften von Druckschrift ausgerichtet. Weil die Ausgangsdaten für die Extraktion der Merkmale nicht entsprechend vorbereitet sind, werden passende Vorverarbeitungsschritte besprochen und die Extraktionsalgorithmen entwickelt. Kapitel 3 erklärt das Verfahren zum Erzeugen des Bewertungsbaums, eines speziellen Entscheidungsbaums, der die Beschreibung der Objektklassen anhand der gegebenen Merkmale widerspiegelt. Ein trainierter Bewertungsbaum wird zur späteren Erkennung von Schriftzeichen mit Hilfe des in Kapitel 4 erläuterten Algorithmus genutzt.

[1] False-negatives sind fälschlicherweise abgelehnte Schriftzeichen (als Nicht-Zeichen klassifiziert).
[2] False-positives sind fälschlicherweise akzeptierte Nicht-Zeichen (als Schriftzeichen klassifiziert).

Kapitel 2

Merkmalsextraktion

Jede Objektklasse wird durch eine Menge von Merkmalen spezifiziert mit denen jedes Objekt vollständig beschrieben werden kann. Ein einzelnes Objekt besitzt dann eine Reihe von Eigenschaften, wobei jede Eigenschaft eine konkrete Ausprägung eines bestimmten Merkmals darstellt. Selbst das Ausbleiben eines Merkmals kann eine Eigenschaft sein. Merkmale können zum Beispiel Kreisstrukturen oder Einbrüche in der konvexen Hülle sein. Während eine *8* gleich zwei Kreisstrukturen und zwei Einbrüche in der konvexen Hülle als Eigenschaften aufweist, so hat ein *I* keines der beiden Merkmale.

In dieser Arbeit werden gedruckte lateinische Buchstaben und arabische Ziffern behandelt. Als erster Schritt ist es notwendig eine Menge von Merkmalen zu definieren, die hinreichend die zu erkennenden Zeichen charakterisieren. Die Menge muss groß genug gewählt werden, damit nicht nur die einzelnen Zeichen gegeneinander, sondern auch gegen alle Nicht-Zeichen, abgegrenzt werden können.

Ausgehend von den binarisierten Pixelbildern der Zeichen kann es Sinn machen für verschiedene Merkmale eine individuelle Vorverarbeitungskette durchzuführen. Genau wie die Merkmale, die problemorientiert definiert werden müssen, wird auch die Vorverarbeitungskette den Anforderungen einzelner Merkmale entsprechend festgelegt. Im Anschluss an die Vorverarbeitung werden die einzelnen Eigenschaften eines Objektes durch Extraktion der Merkmale bestimmt.

2.1 Buchstaben- und Ziffernmerkmale

Das in Kapitel 3 und 4 entwickelte Verfahren zum Erkennen von Objekten ist nicht speziell auf Buchstaben und Ziffern ausgelegt, sondern kann für beliebige Objekttypen verwendet werden, sofern eine geeignete Menge von Merkmalen und dazugehörige Extraktoren für die Objekte zur Verfügung stehen. Diese Arbeit beschäftigt sich mit der Erkennung von Schriftzeichen und erfordert deshalb eine geeignete Merkmalsmenge für diese Klasse von Objekten. Es ist wichtig, dass die Merkmale einerseits hinreichend sind

zur Trennung der Zeichen voneinander, aber andererseits auch die Zeichen so genau beschreiben, dass sie von Nicht-Zeichen unterschieden werden können. Es wird sich zeigen, dass dieser Anforderung für beliebige Objektklassenmengen gerecht zu werden teilweise sehr schwer bis gar nicht möglich ist, da bestimmte einzelne Buchstaben und Ziffern ohne Kontext nicht unterschieden werden können.

Definition *Eine Merkmalsmenge heißt vollständig, wenn bei der Beschreibung der Objektklassen unter Verwendung aller angegebenen Merkmale gilt: Ein unbekanntes Bild Q gehört zu Klasse K, wenn alle Merkmale in Anzahl und Zusammensetzung aus der Beschreibung von Objektklasse K in Q gefunden werden.*

Diese Definition bedeutet insbesondere, dass eine vollständige Merkmalsmenge Merkmale enthalten muss, die eine erfolgreiche Klassifikation verhindern, wenn in einem Abfragebild zusätzliche Informationen zu sehen sind. Wird eine *8* beispielsweise wieder durch zwei Kreise beschrieben, dann würden zwei schematisch dargestellte nebeneinander liegende Neuronen (kreisförmige Objekte mit vielen abstehenden Verästelungen) auch als *8* erkannt werden. Es reicht das Merkmal relative Fläche hinzuzunehmen, um diesen Effekt zu verhindern, obwohl nach wie vor die Merkmalsmenge die Eigenschaft *Verästelungen* nicht berücksichtigt.

In der Typografie lassen sich viele Hinweise und Ideen finden, aus welchen Merkmalen Schrift aufgebaut ist [14]. Sie beschäftigt sich insbesondere mit den Eigenschaften von Schrift, die man im Pixelbild, also dem Ortsraum, eines digitalisierten Zeichens direkt wiederfindet. Dadurch wird die Beschreibung der zu erkennenden Objekte durch die Summe ihrer Eigenschaften auf natürliche Weise motiviert. Obwohl beliebige Merkmale denkbar sind, wird eine Menge angegeben, die der menschlichen Wahrnehmung entspricht.

Linienmuster In vielen Buchstaben und Ziffern lassen sich Linienmuster finden, die außerdem auf einige wenige diskrete Ausrichtungen beschränkt sind. Innerhalb aller Zeichen variieren darüber hinaus Anzahl und Winkel der Linien. Gefundene Linien, als Geraden und Strecken berücksichtigt, stellen ein Merkmal dar.

Merkmal	Beispiele
schräge Linien	A, K, M, N, Q, R, V, W, X, Y, Z, 1, 4, 5, 7
waagerechte Linien	A, E, F, G, H, L, T, Z, 2, 4, 5, 7
senkrechte Linien	B, D, E, F, H, I, K, L, 1, 4

Dreieckmuster Kein weiteres Zeichen außer dem Buchstaben *A* und der Ziffer *4* hat ein vollständiges Dreieck, aber viele enthalten Dreieckmuster derart, dass man ein oder mehrere Dreiecke so über das Zeichen legen kann, dass sich die Eckpunkte innerhalb der konvexen Hülle befinden und nur eine Dreieckseite fehlt.

Merkmal	Beispiele
Dreiecke	A, E, F, H, K, L, M, N, T, V, W, X, Y, Z, 4

Rechteckmuster Rechteckmuster verhalten sich ähnlich wie die Dreieckmuster. Man findet zwar kein Zeichen mit Linienzügen, die ein vollständiges Rechteck beschreiben, aber es gibt einige, die von einem Rechteck umschlossen werden können, so dass Linienzüge auf mindestens zwei der vier Rechteckseiten liegen und sich die restlichen Linienzüge innerhalb des Rechtecks befinden.

Merkmal	Beispiele
Rechtecke	E, F, H

Kreismuster Neben einigen Zeichen, die vollständige Kreis- oder wenigstens Ellipsenmuster aufweisen, gibt es auch einige deren Linienzüge unvollständige Kreise und Halbkreise beschreiben. Der Grat zwischen Kreis- und Halbkreismustern ist sehr schmal. Es ist trotzdem sinnvoll beide Merkmale zu berücksichtigen, da es kein Schaden ist, wenn beim Training die Buchstabenklasse *C* mit beiden Merkmalen klassifiziert wird. Man gewinnt aber mehr nützliche Information, wenn man zusätzlich vollständige Kreismuster differenzieren kann.

Merkmal	Beispiele
Kreise	C, G, O, Q, 3, 5, 6, 8, 9, 0
Halbkreise	B, D, P, R, U, 2

Innenraum Linien von Zeichen mit diesem Merkmal schließen mindestens einen zusammenhängenden Hintergrundbereich vollständig ein.

Merkmal	Beispiele
geschlossener Innenraum	A, B, D, O, P, Q, R, 4, 6, 8, 9, 0

Öffnungen Betrachtet man die verschiedenen Zeichen aus je einer der vier Himmelsrichtungen, so sind bei den meisten von ihnen Einbrüche in der konvexen Hülle relativ zu ihrer Kontur zu sehen. Die auftretenden Merkmale sind vielfältig. Ein X hat beispielsweise in jeder Richtung genau eine Öffnung, aber ein E hat gleich zwei Öffnungen aus nur einer Richtung.

Merkmal	Beispiele
Nordöffnung	H, J, K, M, N, U, V, W, X, Y
Ostöffnung	C, E, F, G, K, L, S, T, X, Z, 2, 5, 6, 8
Südöffnung	A, H, K, M, N, R, W, X
Westöffnung	J, S, T, X, Z, 2, 3, 5, 7, 8, 9

Skelettmerkmale Betrachtet man das Skelett eines Zeichens als ungerichteten Graph, so finden sich die Strukturmerkmale Enden, Kreuzungen und Kreise in diesem Graph.

Merkmal	Beispiele
Enden	A, C, E, F, G, H, I, ...
Kreuzungen	A, B, E, F, H, K, P, ...
Kreise	A, B, D, O, P, Q, R, ...

Einzellinie Zeichen mit diesem Merkmal sind ohne abzusetzen am Stück schreibbar. Das heißt sie bestehen aus nur einem einzigen Linienzug. Im Skelett ist das daran zu erkennen, dass es genau zwei Enden, keine Kreuzung und keinen Kreis gibt.

Merkmal	Beispiele
Einzellinie	C, G, I, J, L, M, N, O, S, U, V, W, Z

Symmetrie Der Symmetriewert eines Zeichens gibt das Verhältnis gespiegelter Pixel an der Mittelachse zu allen Vordergrundpixeln an. Es handelt sich um ein rein quantitatives Merkmal, welches bei jedem Zeichen genau einmal vertreten ist.

Merkmal	Beispiele
perfekt symmetrisch	A, H, I, M, O, T, U, W, V, X, Y

Relative Fläche Die relative Fläche gibt das Verhältnis von Vordergrundpixeln zu Hintergrundpixeln an.

Schwerpunkt Dieses Merkmal bezeichnet den Ortsvektor des Schwerpunktes aller Vordergrundpixel.

Für die Definition und Auswahl der Merkmale gibt es zwei Ziele: Auf der einen Seite soll die Erkennungsleistung für die einzelnen Buchstaben und Ziffern erhöht werden, aber auf der anderen Seite soll auch sicher ein Zeichen von einem Nicht-Zeichen unterschieden werden können. Beide Ziele stehen bei dem vorgestellten Verfahren in keinem direkten Zusammenhang. Durch eine erhöhte Abgrenzungsleistung zur Welt entsteht keine bessere Erkennungsleistung der Zeichen und durch eine gute Zeichenerkennung muss nicht automatisch die Isolierung der Welt gut sein. Es ist aber möglich durch geschickte Auswahl der Merkmale beides zu verbinden.

Wird eine große Menge an Merkmalen ausgewählt, von denen keines in den zu erkennenden Zeichen zu finden sind, dann findet keine Zeichenerkennung statt, aber es kann die binäre Klassifikation für Nicht-Zeichen erreicht werden. Voraussetzung dafür ist aber eine Merkmalsmenge, die die Welt beschreibt, was im Allgemeinen sehr schwer ist, weil keine Kenntnis über die Welt im Vorfeld bekannt ist. Andererseits führt eine kompakte Merkmalsmenge, die eine gute Erkennung der Zeichen erlaubt, nicht zwingend zu einer guten Isolation der Welt, weil damit eventuell nur eine gute Differenzierung der Zeichen untereinander erreicht wird. Deswegen ist es optimal, wenn die ausgewählte Merkmalsmenge die zu erkennenden Objekte vollständig beschreiben kann. Das bedeutet, sie enthält nicht nur die Merkmale, die für eine Unterscheidung der Zeichen ausreichen. Ist eine vollständige Beschreibung der Zeichen möglich, so kann diese erlernt werden und folglich ist alles Unbekannte kein Zeichen.

Alle hier angegebenen Merkmale beschreiben geometrische Basismuster aus denen sich gedruckte Buchstaben und Ziffern zusammensetzen lassen. Jedes Zeichen ist eine eigene individuelle Kombination aus den Mustern, wenn jedes Muster berücksichtigt wird, auch abwesende!

2.2 Vorverarbeitung

Viele Merkmale können nicht direkt aus dem Ausgangsbild extrahiert werden. Es ist dann notwendig alle Bilder für die Extraktion vorzubereiten. Ein Vorteil des hier vorgestellten Verfahrens ist, dass für jedes Merkmal eine individuelle Vorverarbeitung stattfinden kann, die auf die Bedürfnisse des jeweiligen Merkmals eingeht.

Die Natur der Schrift ist, dass sie aus Linienzügen besteht. Die meisten Merkmale legen daher diese Struktur zu Grunde. Es gibt zwei Möglichkeiten diese Anforderung zu erfüllen, entweder durch Skelettierung des Pixelbildes oder Vektorisierung. Da die

Vektorisierung im Allgemeinen einen erheblichen Mehraufwand erfordert, der für die zu gewinnenden Informationen unnötig ist, und in vielen Fällen außerdem eine Skelettierung voraussetzt werden die Ausgangsbilder nur skelettiert. Zuvor erfolgt eine Reinigung des Pixelbildes mit morphologischen Operatoren, um durch die Digitalisierung verklebte oder aufgerissene Stellen im Zeichen zu korrigieren. Außerdem müssen die Ausgangsbilder in Größe und Form normalisiert werden.

2.2.1 Morphologische Operatoren

In dieser Arbeit werden nur morphologische Operatoren auf Binärbildern betrachtet, da die Ausgangsbilder bereits in binarisierter Form vorliegen. Ist das nicht der Fall, so kann problemlos eine Schwellenwertbinarisierung oder ähnliches als zusätzlicher Vorverarbeitungsschritt durchgeführt werden.

Morphologische Operatoren sind Faltungsoperationen[1] auf Bildern mit Kernen, die Strukturelemente genannt werden. Die Strukturelemente können verschiedene Größe und Form haben, um so unterschiedliche Wirkungen zu erzielen. Bei einer morphologischen Operation wird der Ankerpunkt des Strukturelementes über jeden Pixel des Ausgangsbildes gelegt und abhängig von seiner Nachbarschaft, die durch das Strukturelement beschrieben wird, der Pixel in das Ergebnisbild mit übernommen oder nicht.

Es ist zu beachten, dass vor der Ausführung einer morphologischen Operation im Allgemeinen noch ein leerer Rand um das Bild hinzugefügt werden muss, damit es nicht zu Nebeneffekten an den Rändern kommt. Die Breite des leeren Randes hängt von der Größe des Strukturelements ab.

Erosion

Die Erosion ist eine mathematische Minimumfunktion auf einem Ausgangsbild I und einem Strukturelement K (siehe Gleichung 2.1). Sie frisst alle Pixel vom Ausgangsbild weg, deren Nachbarschaft nicht identisch ist mit der, die das Strukturelement beschreibt.

$$I \ominus K = \min_{(x',y') \epsilon K} I(x + x', y + y') \tag{2.1}$$

K ist eine Menge von Koordinaten relativ zum Ankerpunkt, die die Nachbarschaft beschreiben. In Abbildung 2.1(a) ist eine Erosion an einem Beispielbild mit kreuzförmigen Strukturelement zu sehen. Man kann sehr gut erkennen, wie die Ränder der Objekte entfernt werden und dabei die linke Struktur in zwei Zusammenhangskomponenten aufreißt.

[1] Faltung ist ein mathematischer Operator auf zwei Funktionen f und g, dessen Ergebnis Funktion f mit durch g veränderten Wertebereich ist. Hier sind f und g Binärbilder.

Dilatation

Ähnlich der Erosion ist die Dilatation eine mathematische Maximumfunktion auf I und K mit K als Koordinatenmenge des Strukturelements (siehe Gleichung 2.2). Sie fügt dem Ausgangsbild an den Konturrändern überall dort Pixel hinzu, wo mindestens einer der Nachbarschaftspixel bereits ein Vordergrundpixel ist.

$$I \oplus K = \max_{(x',y')\epsilon K} I(x + x', y + y') \tag{2.2}$$

In Abbildung 2.1(b) ist gut zu erkennen, wie die Dilatation mit geeignetem Strukturelement getrennte Zusammenhangskomponenten zusammenwachsen lässt.

Opening

Sind dünn verklebte Stellen in einem Pixelbild zu öffnen, so bietet sich dafür die ausschließliche Verwendung der Erosion nicht an. Neben dem Trennen von Zusammenhangskomponenten entfernt sie überall Randpixel und verkleinert somit die Objekte, was häufig auch mit Informationsverlust verbunden ist. Abbildung 2.1(a) zeigt, dass aus dem ursprünglich kreisähnlichen Objekt rechts unten nach der Erosion ein rechteckiges Objekt geworden ist.

Diesem Problem entgegnet die morphologische Öffnungsoperation. Sie kombiniert eine Reihe von Erosionsschritten mit anschließenden Dilatationsschritten. Das für dazu, dass nach dem Öffnen der Zusammenhangskomponenten, die vormals entfernten Randpixel durch Dilatation wieder hinzugefügt werden, ohne erneutes Zusammenwachsen der Objekte.

Abbildung 2.1(c) zeigt die Öffnungsoperation mit einem Erosions- und einem anschließenden Dilatationsschritt. Die linke Struktur bleibt jetzt getrennt und trotzdem ist das Objekt rechts unten immer noch kreisähnlich.

Closing

Vergleichbar der morphologischen Öffnungsoperation begegnet die Schließungsoperation dem Problem des künstlichen Verdickens und der Informationsverfälschung durch alleinige Dilatationsoperation, wenn es darum geht durch Ausreißer getrennte Objekte wieder zu verbinden. Dafür werden Erosionsoperationen nach Dilatationsoperationen ausgeführt, um unnötig hinzugefügte Randpixel wieder zu entfernen. Abbildung 2.1(d) zeigt einen Dilatations- und Erosionsschritt.

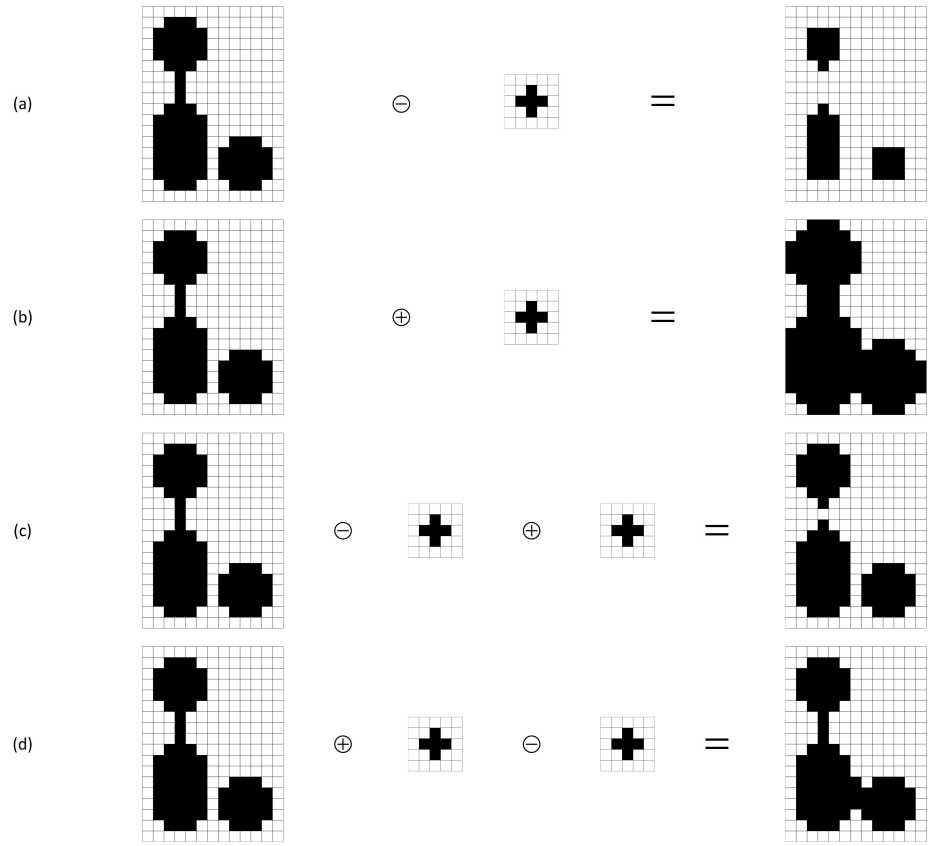


Abbildung 2.1: Morphologische Operationen mit kreuzförmigem Strukturelement und zentrischem Ankerpunkt: (a) Erosion, (b) Dilatation, (c) Opening, (d) Closing.

2.2.2 Normalisierung

Viele der angegebenen Merkmale verwenden Ortsinformationen und einige Extraktoren benutzen Pixelhäufigkeiten als Schwellenwerte. Damit diese auch vergleichbar sind ist es erforderlich alle Eingabebilder auf ein einheitliches Maß zu bringen. Alle Ausgangsbilder haben nach der Normalisierung eine Größe von 25×25 Pixel. Diese Größe ist klein genug, um den Rechenaufwand in Grenzen zu halten, aber groß genug, um nicht zu viel Information zu verlieren. Einige Merkmale erfordern zusätzlich, dass das Seitenverhältnis der Zeichen bei der Normalisierung erhalten bleibt. Andernfalls würde es zu Verzerrungseffekten und damit fehlerhafter Merkmalsextraktion kommen. Kreise würden beispielsweise Ellipsen. Da jeder Merkmalsextraktor seine eigene Vorverarbeitungskette haben kann gibt es die Normalisierung mit und ohne Erhaltung des Seitenverhältnisses.

Von jedem Bild wird zuerst der Rand so weit weggeschnitten, dass das dargestellte Objekt in jeder Richtung am Bildrand anstößt. In den äußersten Zeilen und Spalten des Bildes befindet sich dann jeweils mindestens 1 Vordergrundpixel. Anschließend wird das Bild auf 25×25 Pixel skaliert, wenn das Seitenverhältnis nicht erhalten bleiben muss. Soll das Seitenverhältnis aber erhalten bleiben, dann wird das Bild in x- und y-Richtung gleichmäßig skaliert, so dass die längere der beiden Seiten eine Ausdehnung von 25 Pixel hat. Am Ende wird das Bild in seiner kürzeren Ausdehnung auf beiden Seiten gleichmäßig mit Hintergrund aufgefüllt, damit es eine quadratische Form von 25×25 Pixel bekommt. Damit ist gewährleistet, dass ein Objekt ohne Verlust seines Seitenverhältnisses mit größtmöglicher Ausdehnung horizontal und vertikal zentriert in einer 25×25 Pixel Box sitzt. Bei der Skalierung von Bildern müssen aus den Originalpixeln neue berechnet werden. Dafür kommt eins der Standardinterpolationsverfahren bilinear, bikubisch, Pixelflächendurchschnitt oder nächster-Nachbar zur Anwendung [15]. Nach der Interpolation liegt ein Grauwertbild vor, das mit einer einfachen Schwellenwertbinarisierung wieder in ein Binärbild konvertiert wird. Beste Ergebnisse haben sich bei der bilinearen Interpolation gezeigt. Die Skalierung kommt auch bei der Verwendung von morphologischen Operatoren zur Anwendung. Abhängig von der Größe des Strukturelements ist es erforderlich das Ausgangsbild in eine entsprechende Größe zu bringen, weil die Wirkung des morphologischen Operators im Bild von der Größe des Strukturelements abhängt.

Abbildung 2.2 zeigt die Normalisierung ohne Erhaltung des Seitenverhältnisses. Für senkrechte und waagerechte Linienmerkmale ist diese Normalisierung notwendig, damit die entsprechenden Linien immer im selben Bereich des normalisierten Bildes liegen. Je nachdem, ob beispielsweise ein „H“ breit oder schmal ist haben die senkrechten Linien einen unterschiedlichen Abstand zueinander. Nach der Normalisierung ist der Abstand etwa gleich und die senkrechten Linien im gleichen Bildbereich zu finden.

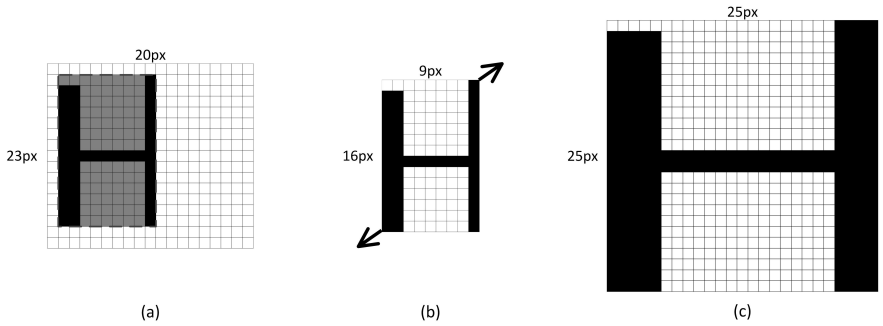


Abbildung 2.2: Normalisierung ohne Erhaltung des Seitenverhältnisses: (a) Ausgangsbild, (b) randlos ausgeschnittene Zusammenhangskomponente, (c) skaliertes Bild (25×25 Pixel)

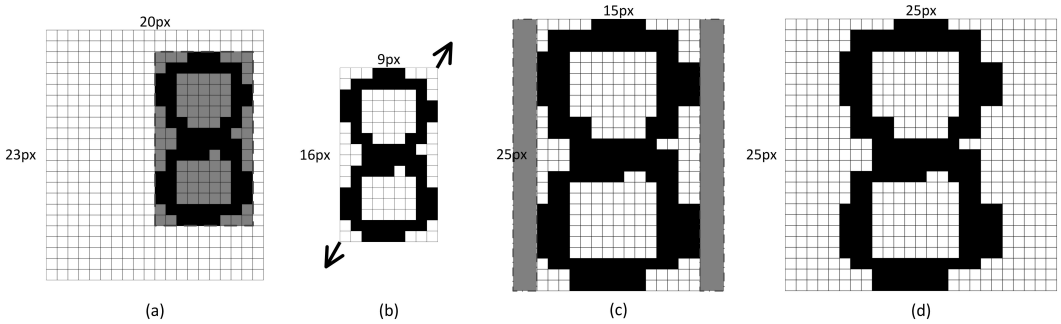


Abbildung 2.3: Normalisierung mit Erhaltung des Seitenverhältnisses: (a) Ausgangsbild, (b) randlos ausgeschnittene Zusammenhangskomponente, (c) gleichmäßige Skalierung beider Seiten auf 25 Pixel Ausdehnung bei der längeren Seite, (d) auf 25×25 Pixel aufgefülltes Bild

Abbildung 2.3 zeigt die Normalisierung mit Erhaltung des Seitenverhältnisses. Merkmale wie das Kreismuster erfordern diese Normalisierung, damit Kreise nicht verzerrt werden. Unterscheidungen zwischen *O* (Otto) und *0* (Null) sind ohne eine seitenverhältniserhaltende Normalisierung unmöglich.

2.2.3 Skelettierung

Der wichtigste Vorverarbeitungsschritt ist die Skelettierung. Er wird von den meisten Merkmalen benötigt und stellt eine Art Normierung auf Pixelstrukturebene dar. Bestimmte Muster, wie Kreise und Linien, haben in normalisierten und skelettierten Bildern ähnliche Pixelhäufungen, was für die Extraktoren wichtig ist, da einige mit festen

Schwellenwerten arbeiten.

Für das Skelett einer Zusammenhangskomponente gibt es keine eindeutige Definition, nur verschiedene Forderungen und die können von Anwendungsgebiet zu Anwendungsgebiet unterschiedlich sein. Typischerweise werden an die Skelettierungsverfahren folgende Forderungen gestellt, die auch für die in dieser Arbeit festgelegten Merkmale hinreichend sind:

- Linien des Skeletts sind 1 Pixel breit
- Linien des Skeletts sollten etwa in der Mitte der Flächen des Ausgangsbildes liegen
- Zusammenhangskomponenten sind im Skelett nach wie vor zusammenhängend
- Verzweigungen und Endpunkte werden im Skelett weder hinzugefügt noch entfernt (Topologieerhaltung)

Es gibt sehr viele verschiedene Verfahren zur Skelettierung von Binärbildern [16, 17], alle mit unterschiedlichen Stärken und Schwächen, sowie Zusicherungen an die Forderungen. Eine häufige Klasse sind die morphologischen Skelettierungsverfahren. Diese führen eine iterative Erosion mit verschiedenen Strukturelementen bis zur Konvergenz durch. Die Strukturelemente beschreiben dabei die verschiedenen möglichen Pixeltypen (Kantenpixel, Randpixel, innenliegender Pixel und so weiter), damit der Algorithmus entscheiden kann, ob ein Pixel zum Skelett gehört oder gelöscht werden muss. Allerdings handelt es sich nicht um die einfache Erosion, wie in Abschnitt 2.2.1 angegeben, sondern um aufwendigere Funktionen.

Im Rahmen dieser Arbeit wurden der MB Thinning Algorithmus [18] und der Zhang-Suen Algorithmus [19] für die angegebenen Merkmale untersucht. Der MB Thinning Algorithmus ist ein einfacher morphologischer Skelettieralgorithmus, ähnlich dem Hit-and-Miss Algorithmus [16], der optimiert wurde für Effizienz und Parallelisierung. Er hat entsprechend vergleichbare Eigenschaften, garantiert aber nicht Linien von einer Pixelbreite. Der Algorithmus von Zhang und Suen ist ebenfalls sehr effizient und garantiert die maximale Linienbreite von einem Pixel, kann allerdings zu Verkürzungen bei den Linien führen. Es hat sich gezeigt, dass der Zhang-Suen Algorithmus für die Merkmale bessere Ergebnisse erzielt.

Wie bei den Morphologischen Operatoren ist auch hier zu beachten, dass vor der Skelettierung ein leerer Rand dem Bild hinzugefügt werden muss, um Nebeneffekte zu vermeiden. Abbildung 2.4 zeigt die Eigenschaften der beiden Skelettierungsalgorithmen anhand einfacher geometrischer Grundformen. Es ist zu sehen, dass beim MB Thinning die Linien nicht immer eine Strichstärke von 1 aufweisen und Zhang-Suen die Linien

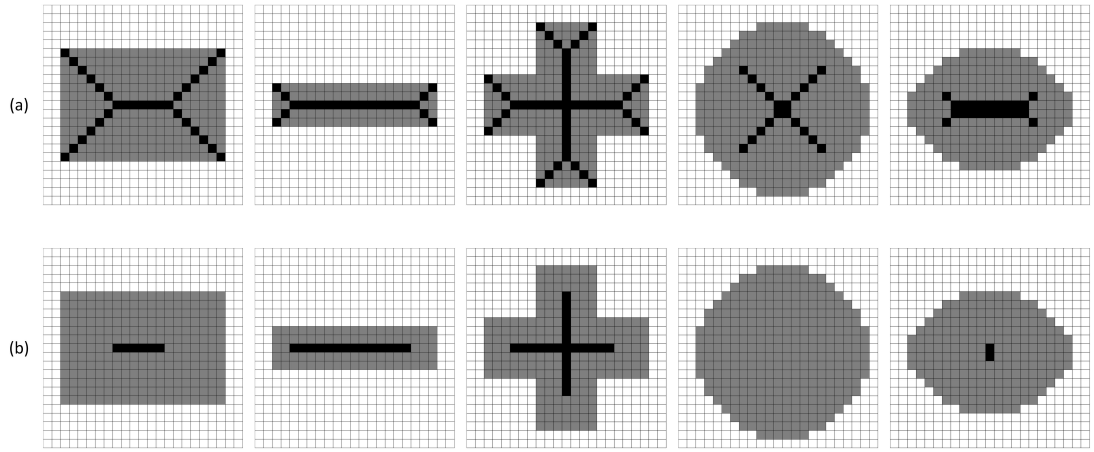


Abbildung 2.4: Skelettierung (schwarze Linien) einfacher geometrischer Formen (grau dargestellt): (a) MB Thinning Algorithmus, (b) Zhang-Suen Algorithmus

mitunter stark verkürzt. Bei quadratischen und symmetrischen Figuren, wie dem Kreis, liefert der Algorithmus sogar kein Skelett mehr. Hauptproblem bei MB Thinning sind die Verzweigungen, die an den Enden der Skelettlinien entstehen. Erwartet man für breite Pixelbereiche einzelne Linien, wie das bei Schriftzeichen der Fall ist, dann wird mit diesem Verfahren neue Information hinzugefügt. Solche Effekte treten bei Zhang-Suen nicht auf, was aber den Preis der verkürzten Linien hat, wie in den Beispielen deutlich zu sehen ist.

Abbildung 2.5 zeigt die Anwendung beider Skelettierungsverfahren auf einigen Beispielbildern. Die mit MB Thinning skelettierten Zeichen weisen viele Stellen auf, die breiter als 1 Pixel sind und haben die typischen Verästelungen am ganzen Skelett, wo im Ausgangsbild große schwarze Pixelbereiche sind. Zhang-Suen erzeugt einfache Linienzüge von einer Pixelbreite, die dem natürlichen Grundmuster der Zeichen entsprechen. Die Verkürzung der Linien bei Zhang-Suen ist deutlich an den Beispielen *3* und *f* zu sehen. Hier gehen bei der *3* die mittlere Einbuchtung nach links und bei dem *f* der Querbalken verloren.

Aufgrund der Größenvariabilität unter den zur Verfügung stehenden Beispielbildern hat sich gezeigt, dass bessere Ergebnisse erzielt werden, wenn eine bildgrößenabhängige Vorverarbeitung durchgeführt wird. Bei besonders kleinen Bildern ($5px \times 9px$) wurden

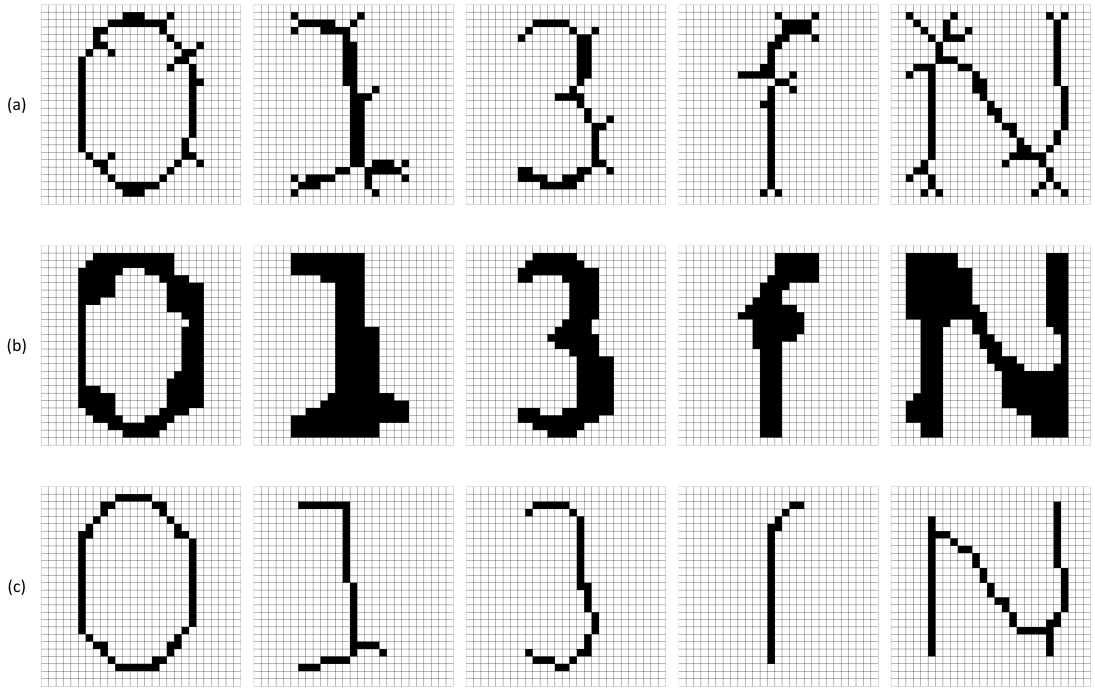


Abbildung 2.5: Skelettierung ausgewählter Zeichen (von links nach rechts 0, 1, 3, f, N): (a) Zhang-Suen, (b) Ausgangsbild, (c) MB Thinning

offene Löcher nur zuverlässig geschlossen, wenn die morphologische Schließungsoperation vor der Normalisierung ausgeführt wurde. Das liegt daran, dass wegen der Skalierung bei der Normalisierung die Löcher so sehr vergrößert werden, das ein größeres Strukturelement nötig ist. Das kann aber nicht bei großen Bildern verwendet werden, denn dann würden auch reguläre Lücken geschlossen.

2.3 Extraktoren

Einige der in diesem Abschnitt vorgestellten Algorithmen benutzen mathematische Operationen, die es erfordern, dass Hintergrundpixel den Wert 0 und Vordergrundpixel den Wert 1 haben (beispielsweise bei der Verwendung der bitweisen *OR* Funktion). Deswegen gilt im Folgenden die Konvention, dass der Pixelwert 0 gemeint ist, wenn von Hintergrund gesprochen wird und Pixelwert 1 gemeint ist, wenn von Vordergrund gesprochen wird. Liegen Bilder in einem anderen Format vor, sind sie hinsichtlich dieser Vereinbarung zu konvertieren.

2.3.1 Linien, Kreise

Eine Standardmethode zum Erkennen von Linien und Kreisen ist die Hough-Transformation [20]. Sie verwendet einen brute-force[2] Ansatz und ist daher sehr rechenintensiv. Eine andere Möglichkeit beruht auf dem RANSAC Algorithmus [21]. Hierbei wird versucht mit deinem randomisierten Verfahren Linien in einem Bild zu finden. Anders als bei der Hough-Transformation ist allerdings nicht sichergestellt, dass immer alle Linien gefunden werden. Die Untersuchungen haben jedoch gezeigt, dass RANSAC schon mit wenigen Wiederholungen sehr gute Ergebnisse liefert.

Bei dem RANSAC Algorithmus wird davon ausgegangen, dass Messwerte eines zu bestimmenden Modells fehlerbehaftet sind und vor der eigentlichen Schätzung des Modells möglichst die Messwerte mit großem Fehler verworfen werden sollen, damit ein geeignetes Schätzverfahren ein genaueres Ergebnis liefern kann. Hat man beispielsweise Messwerte, von denen man erwartet, dass sie auf einer Geraden liegen, so ist ein Standardverfahren zum Ausgleichen einer Geraden die Methode der kleinsten Quadrate [22]. Die Qualität der Ausgleichsgerade hängt von der Qualität der Messwerte ab. Man kann das Ergebnis erheblich verbessern, wenn man Messwerte mit großem Fehler von der Modellschätzung ausschließt. RANSAC bestimmt dafür aus einer zufälligen und ausreichend großen Teilmenge der Messpunkte ein Modell. Im Fall von Linien ist das

[2] Lösungsmethode, die im Wesentlichen auf naivem Ausprobieren aller Fälle basiert.

Modell eine Gerade, die durch zwei Punkte eindeutig bestimmt ist. Für dieses Modell werden dann die Messpunkte der Konsensmenge ermittelt. Das sind alle Datenpunkte, die einen vorgegebenen maximalen Abstand vom Modell nicht überschreiten. Dieser Vorgang wird mehrfach wiederholt und am Ende die größte Konsensmenge ausgewählt, um auf ihr das Schätzverfahren anzuwenden. Dadurch wird erreicht, dass alle Messpunkte, die einen gewissen Abstand vom Modell überschreiten, nicht berücksichtigt werden. Ausreißer werden so nicht in die Schätzung mit einbezogen. Besonders hilfreich ist RANSAC, wenn in der Menge der Messwerte auch Datenpunkte enthalten sind, die nicht dem zu schätzenden Modell angehören. Genau diese Situation ist in einem Bild gegeben, wenn bestimmte Formen gesucht werden. Einige Pixel gehören zum Modell und andere nicht. Ohne Ausschluss der fremden Pixel kann das Modell nicht bestimmt werden.

Bei der Liniensuche in Schriftzeichen besteht die Schwierigkeit darin, dass das Modell mehr als einmal vertreten sein kann. Der RANSAC Algorithmus muss angepasst werden, damit alle Linien gefunden werden können. Anschließend werden aus den Linien Strecken extrahiert und jeweils als Merkmalsvektor gespeichert. Algorithmus 2.1 ist die veränderte RANSAC Version für multiple Linien- und Streckensuche mit folgenden Parametern:

u Maximale Linienanzahl: Pro Durchlauf wird die längste, falls vorhanden, der noch verbleibenden Linien gefunden. Kleinere Werte von höchstens 5 werden hier angegeben.

v RANSAC Iterationen: Pro Durchlauf wird ein zufälliges Linienmodell bestimmt und evaluiert. Werte nicht größer als 20 sind hier sinnvoll.

I Punktmenge der Vordergrundpixel im Ausgangsbild.

$\theta_{maxDist}$ Maximaler Abstand eines Punktes von der Modellgeraden (Lotabstand), um noch zur Konsensmenge dazu gezählt zu werden.

$\theta_{minSize}$ Mindestgröße der Konsensmenge, um als gültige Linie akzeptiert zu werden.

θ_{maxGap} Maximal zulässiger Lückenabstand innerhalb einer Strecke. Ist ein Abstand zwischen zwei Punkten auf der Linie größer als dieser Schwellenwert, dann gehören die beiden Punkte nicht zu ein und derselben Strecke.

θ_{minSeg} Minimale Länge eines Liniensegments, um dieses als Strecke zu akzeptieren. Liegen die zwei Endpunkte eines Liniensegments näher beieinander, dann wird es verworfen.

$d(p_1, p_2)$ Distanzfunktion, die den euklidischen Abstand zwischen den Punkten p_1 und p_2 berechnet.

Die Modifikation des Algorithmus besteht darin, dass er noch nicht fertig ist, nachdem die beste Teilmenge der Datenpunkte für das gegebene Modell bestimmt wurde. Im Anschluss daran wird die Punktmenge, die für die zufällige Modellbestimmung verwendet wird, um die die Datenpunkte der gefundenen Konsensmenge verringert und von vorne begonnen. Im nächsten Durchlauf kann der Algorithmus so nur noch auf den verbliebenen Punkten das nun beste Modell bestimmen. Für die Berechnung der Konsensmengen und Regressionsgeraden stehen aber während der gesamten Laufzeit immer alle Datenpunkte zur Verfügung. Das ist nötig, weil Linien sich schneiden und berühren können, wie beispielsweise bei einem X oder E. Mit der Verringerung der Zufallsmenge soll nur verhindert werden, dass eine bereits gefundene Linie erneut gefunden wird. Damit aber nicht beliebig kleine Linien gefunden werden ist es nötig einen Schwellenwert für die Mindestgröße einer Konsensmenge einzuführen. Auf den normierten Bildern haben Schriftzeichen mit der Linieneigenschaft eine Untergrenze von etwa 10 Pixel pro Linie. Der Schwellenwert wird entsprechend konfiguriert. Regressionsgeraden werden mit der Methode der kleinsten Quadrate ermittelt und alle Strecken werden aus den Projektionen der Bildpunkte auf die Gerade und suche nach genügend großen zusammenhängenden Segmenten errechnet. Abschließend müssen Linien und Strecken in einem Merkmalsvektor gespeichert werden, der Ähnlichkeiten zwischen verschiedenen Vektoren mit einem geeigneten Abstandsmaß abbildet. Dafür werden Geraden als Vektor 2.3 mit ihren Polarkoordinaten d und α gespeichert. Zur allgemeinen Vergleichbarkeit muss α so umgerechnet werden, dass d immer positiv ist, was durch die Betragsstriche um $|d|$ gekennzeichnet ist. Für den Winkel α werden die Winkelfunktionswerte gespeichert, wodurch eine Standard L-Norm als Abstandsmaß möglich wird. Bei einer einfachen Speicherung des Winkels ist das nicht möglich, weil 0° und 360° einen erheblichen Abstand ergeben, obwohl die Geraden bei gleichem d identisch sind.

$$\overrightarrow{f_L} = [|d|, \sin\alpha, \cos\alpha] \tag{2.3}$$

Strecken werden auf ähnliche Weise als Vektor 2.4 gespeichert. Zusätzlich wird hier noch der Mittelpunkt (x, y) und die Länge der Strecke gespeichert.

$$\overrightarrow{f_S} = [|d|, \sin\alpha, \cos\alpha, x, y, l] \tag{2.4}$$

In Abbildung 2.6 sind einige Ergebnisse des modifizierten RANSAC Algorithmus für Linienerkennung zu sehen. Bei niedrigem Schwellenwert $\theta_{minSize}$ werden neben den Linien mit vielen Pixeln teilweise auch Linien quer durch das Bild erkannt. Ein hoher Schwellenwert reduziert die erkannten Linien deutlich auf die erwartete Menge. Zur Streckenerkennung wird trotzdem ein niedriger Schwellenwert $\theta_{minSize}$ verwendet, damit auch

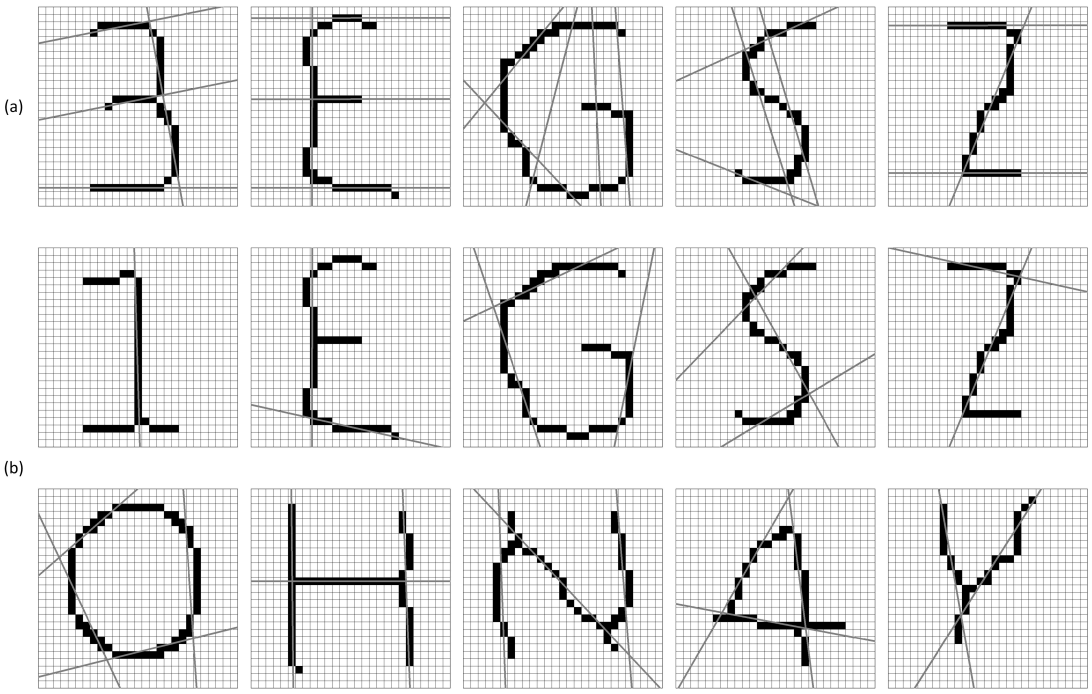


Abbildung 2.6: Beispiele Linienextraktion mit $\theta_{maxDist} = 1,5$: (a) Schwellenwert $\theta_{minSize} = 4$, (b) Schwellenwert $\theta_{minSize} = 9$

kurze Strecken gefunden werden, wie Abbildung 2.7 zeigt. Falsche Ergebnisse, wie bei der Linienerkennung, fallen durch die Streckenschwellenwerte raus.

Kreismerkmale werden analog zu den Linienmerkmalen mit Hilfe des modifizierten RANSAC Algorithmus extrahiert. Das Kreismodell wird durch drei Datenpunkte beschrieben, da ein Kreis durch drei Punkte eindeutig bestimmt ist, sofern sie nicht auf einer Geraden liegen. Denkbar wäre auch ein Modell, bei dem ein Kreis durch die zwei Endpunkte auf einer seiner Diagonalen beschrieben wird. Der Rechenaufwand wäre zwar geringer, aber zwei solche Punkte zufällig zu treffen ist sehr unwahrscheinlich und würde entsprechend zu schlechten Ergebnissen führen. Anders als bei der Liniensuche kann bei der Kreissuche der Schwellenwert $\theta_{minSize}$ nicht absolut vorgegeben werden. Die erwartete Anzahl von Pixeln in einem Kreis ist abhängig von der Größe des Kreises. Kleine Kreise enthalten wenige Pixel und große Kreise enthalten mehr. Ein kleiner absoluter Schwellenwert führt dazu, dass auch langgezogene leicht gebogene Linien als sehr große

Algorithmus 2.1 modifizierter RANSAC Algorithmus zur Streckenextraktion in Binärbildern

```
1  // Initialisierung
2  L = {}                  // Menge der gefundenen Geraden
3  S = {}                  // Menge der Strecken
4  R = I                   // Menge für zufällige Punktauswahl
5
6  // multiple RANSAC Linienberechnung
7  u Wiederholungen:
8          L_temp = {}
9          v Wiederholungen:
10                 wähle zufällig zwei Punkte p_1, p_2 ∈ R aus
11                 bestimme Modell m (Gerade durch p_1 und p_2)
12                 bestimme Konsensmenge C ⊆ I
13                         C = {p ∈ I | d(p, m_p) ≤ θ_maxDist,
14                                      m_p ist Lotpunkt von p auf m}
15                 |C| ≥ θ_minSize ?
16                         bestimme Regressionsgerade g mit C
17                         L_temp = L_temp ∪ {(g, C)}
18
19         wähle l = (g_l, C_l) ∈ L_temp , mit |C_l| = max_{l=(g,C)∈L_temp} |C|
20         R = R \ C_l
21         L = L ∪ {l}
22
23
24 // Streckenberechnung auf L
25 für jedes l ∈ L
26         bestimme Lotpunkte p aller c ∈ C von l = (g, C) auf g
27         sortiere Lotpunkte p aufsteigend
28         S = S ∪ {(p_i, p_j) | p_i < p_j;
29                             d(p_{i-1}, p_i) > θ_maxGap;
30                             d(p_j, p_{j+1}) > θ_maxGap;
31                             d(p_i, p_{i+1}), d(p_{i+1}, p_{i+2}), ..., d(p_{j-1}, p_j) ≤ θ_maxGap;
32                             d(p_i, p_j) ≥ θ_minSeg}
```

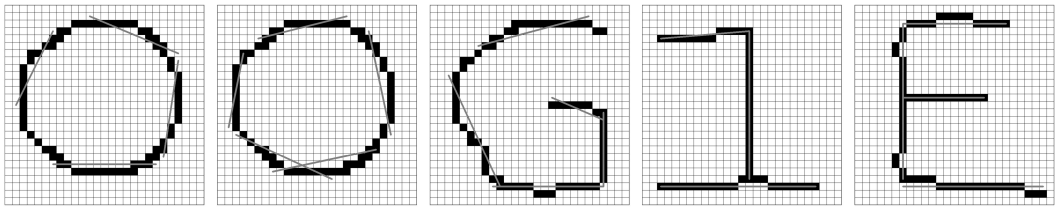


Abbildung 2.7: Beispiele Streckenextraktion mit Schwellenwerten $\theta_{maxDist} = 1,5$, $\theta_{minSize} = 4$, $\theta_{maxGap} = 2$ und $\theta_{minSeg} = 4$

Kreise erkannt werden. $\theta_{minSize}$ wird deswegen als relativer Wert $\theta_{minSize}/r$ vorgegeben und zur Laufzeit in den passenden Wert $\theta_{minSize}$, abhängig vom Radius des Kreiskandidaten, umgerechnet.

Ist (x_m, y_m) der Mittelpunkt eines Kreises mit Radius r, dann lässt sich aus der allgemeinen Kreisgleichung 2.5 für einen beliebigen Kreispunkt (x, y) ein lineares Gleichungssystem 2.6 für drei Kreispunkte (x_1, y_1), (x_2, y_2), (x_3, y_3) aufstellen.

$$\begin{aligned} (x - x_m)^2 + (y - y_m)^2 &= r^2 \\ x^2 - 2 \cdot x \cdot x_m + x_m^2 + y^2 - 2 \cdot y \cdot y_m + y_m^2 &= r^2 \end{aligned} \tag{2.5}$$

$$\begin{pmatrix} -1 & x_1 & y_1 \\ -1 & x_2 & y_2 \\ -1 & x_3 & y_3 \end{pmatrix} \cdot \begin{pmatrix} x_m^2 + y_m^2 - r^2 \\ 2 \cdot x_m \\ 2 \cdot y_m \end{pmatrix} = \begin{pmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ x_3^2 + y_3^2 \end{pmatrix} \tag{2.6}$$

Der Kreismerkmalextraktor bestimmt zufällig drei Punkte im Bild und löst mit ihnen das Gleichungssystem, um einen Kreiskandidaten zu erhalten. Anschließend wird die Konsensmenge in einem Kreisring mit vorgegebener Breite $2 \cdot \theta_{maxDist}$ und Radius des Kandidaten ermittelt. Nach mehrfacher Wiederholung dieses Vorgangs wird der Kandidat mit maximaler Konsensmenge ausgewählt und auf seiner Konsensmenge ein Kreis mit der Methode der kleinsten Quadrate ausgeglichen, vorausgesetzt die Konsensmenge war mindestens $\theta_{minSize}$ groß. Dieser Kreis entspricht dem ersten gefundenen Kreis. Die Punkte seiner Konsensmenge werden aus der Menge zufällig wählbarer Punkte entfernt, damit bei der nächsten Ausführung ein neuer Kreis gefunden werden kann. Zur Bestimmung von Halbkreisen wird der Kreis-RANSAC mit herabgesetztem Schwellenwert $\theta_{minSize}$ ausgeführt und anschließend ähnlich der Streckenextraktion ein zusam-

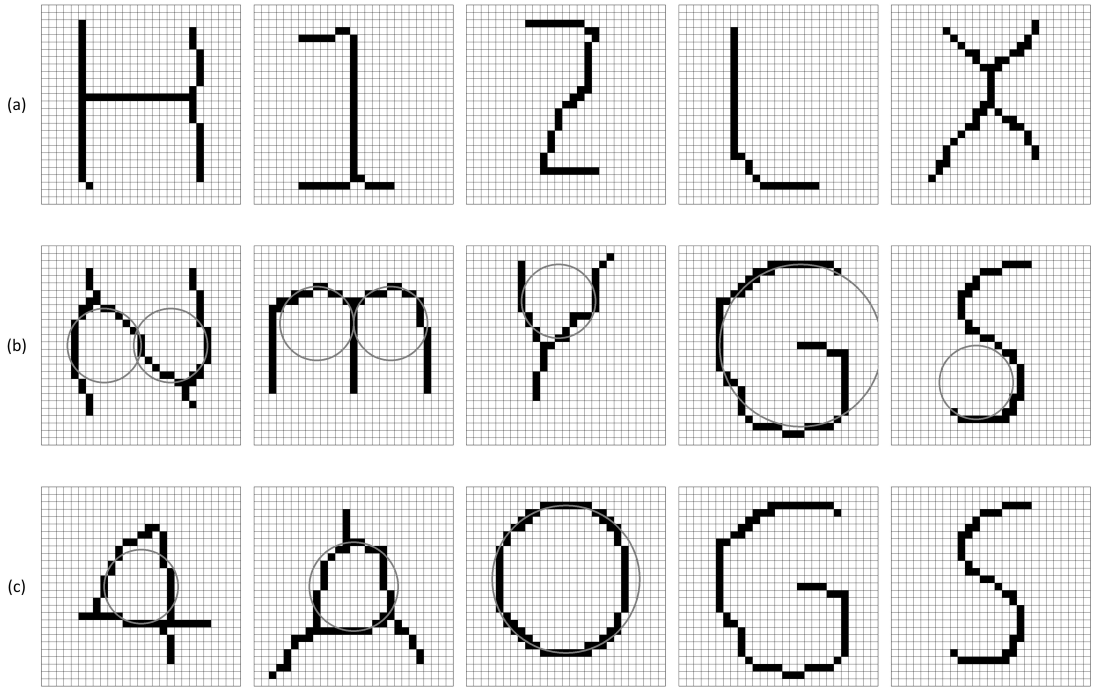


Abbildung 2.8: Beispiele Voll- und Halbkreisextraktion: (a) keine Kreiserkennung bei hohem und niedrigem Schwellenwert, (b) relativer Schwellenwert $\theta_{minSize} = {}^{23}/_{5}$ (Halbkreise), (c) relativer Schwellenwert $\theta_{minSize} = {}^{28}/_{5}$ (Vollkreise)

menhängendes Segment auf der Kreislinie von etwa einem halben Kreisumfang gesucht. Die Speicherung der Merkmale erfolgt durch einen einfachen Vektor 2.7 mit Kreismittelpunkt und Radius. Halbkreise (Vektor 2.8) werden zusätzlich mit dem Mittelpunkt und der Länge des Streckensegments gespeichert.

$$\overrightarrow{f_C} = [x_m, y_m, r] \tag{2.7}$$

$$\overrightarrow{f_{HC}} = [x_m, y_m, r, x_s, y_s, l_s] \tag{2.8}$$

Wie zuverlässig der RANSAC Algorithmus für Kreise und Halbkreise funktioniert ist in Abbildung 2.8 zu sehen, obwohl sie auch in Zeichen erkannt werden, von deren Grundstruktur her es nicht zu erwarten ist, wie bei einer *4* oder einem *A*.

2.3.2 Dreiecke, Rechtecke

Die in dieser Arbeit definierten Merkmale sind speziell auf die Charakteristiken lateinischer Schriftzeichen zugeschnitten. Im Hinblick auf Dreieck- und Rechteckmuster fällt auf, dass diese nicht in allgemeiner Form vorkommen, sondern mit spezielleren Eigenschaften. Deshalb werden die allgemeinen Definitionen von Dreiecken und Rechtecken angepasst und im Folgenden gelten diese Festlegungen:

Definition *Ein Dreieck ist eine Struktur aus drei Strecken, wobei jeder Endpunkt einer Strecke mit jeweils genau einem Endpunkt der beiden anderen Strecken verbunden ist. Mindestens eine der drei Strecken ist waagerecht oder senkrecht und höchstens eine Strecke ist eine Phantomstrecke sein. Das bedeutet diese Strecke hat im Bild keine Repräsentanten, es gibt also keine Vordergrundpixel auf dieser Strecke.*

Definition *Ein Rechteck ist eine Struktur aus vier Strecken, wobei jeder Endpunkt einer Strecke mit genau einem Endpunkt von zwei der drei anderen Strecken verbunden ist. Genau eine der vier Strecken ist eine Phantomstrecke, was bedeutet diese Strecke hat im Bild keine Repräsentanten, es gibt also keine Vordergrundpixel auf dieser Strecke. Pixelrepräsentierte Strecken sind waagerecht oder senkrecht und gibt es unter ihnen nur eine waagerechte, dann ist die Phantomstrecke auch waagerecht, andernfalls ist die Ausrichtung der Phantomstrecke nicht spezifiziert. Keine der Linien schneiden sich.*

Da die Definition der Rechtecke zulässt, dass unter bestimmten Umständen die Phantomstrecke nicht rechtwinklig zu ihren Nachbarstrecken stehen muss, ist die Bezeichnung Rechteck nicht ganz korrekt und müsste Viereck lauten. Die Bezeichnung Rechteck wird aber beibehalten um den Sinn dieses Musters deutlich zu machen. Man könnte außerdem die Definition der Dreiecke noch enger fassen und nur rechtwinklige oder gleichschenklige Dreiecke zulassen, der Einfachheit halber wird darauf aber verzichtet.

Dreiecke werden aus einem Binärbild mit Algorithmus 2.2 extrahiert, nachdem zuvor alle Strecken mit dem Linienextraktor gesucht wurden. Der Algorithmus verwendet die folgenden Parameter und Hilfsfunktionen:

S Menge aller Strecken im Ausgangsbild. Die Menge wird vor Ausführung des Algorithmus mit den gefundenen Strecken des Linienextraktors initialisiert.

$\theta_{maxDist}$ Maximaler Abstand zwischen zwei verbundenen Streckenendpunkten. Dieser Schwellenwert gibt den Toleranzradius von Streckenendpunkten an. Punkte

innerhalb dieses Radius von einem Punkt p sind mit p verbunden, andernfalls nicht verbunden.

$halfs(S)$ Funktion, die auf einer gegebenen Streckenmenge S alle Streckenhälften S_h berechnet und zurückgibt. Streckenhälften sind:

$$\begin{aligned} S_h \;=\; & \{s' = (p'_b, p'_e), s'' = (p''_b, p''_e) \mid \exists s = (p_b, p_e) \in S : \\ & d(p'_b, p'_e) = d(p''_b, p''_e) = \frac{d(p_b, p_e)}{2} \;\wedge \\ & p_b = p'_b \;\wedge\; p'_e = p''_b \;\wedge\; p''_e = p_e\} \end{aligned} \tag{2.9}$$

$thirds(S)$ Funktion, die auf einer gegebenen Streckenmenge S alle Streckendrittel S_t berechnet und zurückgibt. Streckendrittel sind:

$$\begin{aligned} S_t \;=\; & \{s' = (p'_b, p'_e), s'' = (p''_b, p''_e), s''' = (p'''_b, p'''_e) \mid \exists s = (p_b, p_e) \in S : \\ & d(p'_b, p'_e) = d(p''_b, p''_e) = d(p'''_b, p'''_e) = \frac{d(p_b, p_e)}{3} \;\wedge \\ & p_b = p'_b \;\wedge\; p'_e = p''_b \;\wedge\; p''_e = p'''_b \;\wedge\; p'''_e = p_e\} \end{aligned} \tag{2.10}$$

$d(p_1, p_2)$ Distanzfunktion, die den euklidischen Abstand zwischen den Punkten p_1 und p_2 berechnet.

$angleCriterion(s_1, s_2, s_3)$ Boole'sche Funktion, die das Ausrichtungskriterium für Dreieckseiten auf drei Strecken überprüft. Rückgabe ist *wahr*, wenn mindestens eine der drei Strecken waagerecht oder senkrecht ist, andernfalls *falsch*. Eine Strecke ist waagerecht oder senkrecht, wenn ihr Winkel $0° \pm \theta$ oder $90° \pm \theta$, mit vorgegebenem θ, beträgt.

$createPhantom(s_1, s_2)$ Funktion, die eine Phantomstrecke zwischen den gegebenen Strecken s_1 und s_2 erzeugt und zurückgibt. Anfangspunkt der Phantomstrecke ist der nicht mit s_2 verbundene Punkt von s_1 und Endpunkt ist der nicht mit s_1 verbundene Punkt von s_2.

Nach der Initialisierung der Streckenmenge mit dem Linienextraktor wird die Menge als erstes um alle Streckenhälften (vgl. Menge 2.9) und Streckendrittel (vgl. Menge 2.10) erweitert. Das ist notwendig, weil viele Dreiecke einen Endpunkt in der Mitte von den Linienzügen der Schriftzeichen haben, wie beispielsweise beim X der Fall. Es werden dafür künstliche Strecken eingefügt, die die Originalstrecken in zwei oder drei gleichlange Segmente aufteilen. Bei Betrachtung der Schriftzeichen mit vorkommendem Dreieckmerkmal fällt auf, dass Dreieckendpunkte entweder auf der Hälfte oder einem Drittel

zwischen den Endpunkten der Strecken liegen, weswegen diese Vereinfachung möglich ist. Korrekter wäre es die Schnittpunkte der Originalstrecken zu berechnen. Anschließend sucht der Algorithmus für jede Strecke in der Menge aller Strecken ihre Nachbarn und prüft, ob sich aus diesen ein gültiges Dreieck zusammensetzen lässt, notfalls auch mit einer Phantomstrecke. Wird ein solches Dreieck gefunden, das keine bereits besuchte Seite enthält, dann wird es der Ergebnismenge hinzugefügt. Jedes Dreieck wird in einem Merkmalsvektor 2.11 gespeichert, der den Mittelpunkt (x, y) und Radius r des umschließenden Kreises, sowie zwei Winkelverhältnisse v_1, v_2 und den Rotationswinkel δ des Dreiecks enthält. Abbildung 2.9 zeigt den Zusammenhang der Vektorwerte. Werden die Schnittwinkel der Winkelhalbierenden absteigend sortiert ($\alpha < \beta < \gamma$), dann sind die Verhältnisse v_1 und v_2 ein Ähnlichkeitsmaß für Dreiecke, bis auf Skalierung, Rotation und Ort. Sie geben die Verhältnisse der Seitenlängen zueinander an. Ähnlichkeit der Skalierung und des Ortes wird über den Umkreis mit (x, y) und r bestimmt. Die Rotation des Dreiecks gibt der Winkel δ an und muss wie bei den Linienvektoren über seine Winkelfunktionswerte gespeichert werden. Diese Darstellung des Dreieckmerkmals erlaubt Ähnlichkeit über die Standardabstandsfunktion zu berechnen, wenn zwei zu vergleichende Dreiecke nicht gleichseitig sind. In diesem Spezialfall sind Mittelpunkt des In- und Umkreises dieselben, was die Bestimmung der Rotation nicht mehr möglich macht. Da in Schriftzeichen keine gleichseitigen Dreiecke vorkommen, ist der Umstand vernachlässigbar, dass alle gleichseitigen Dreiecke, die sich am selben Ort befinden und dieselbe Größe haben mit beliebigem Rotationswinkel als identisch betrachtet werden.

$$\overrightarrow{f_T} = [x, y, r, v_1, v_2, \sin\delta, \cos\delta] \tag{2.11}$$

$$\begin{aligned} v_1 &= \alpha \\ v_2 &= \frac{\beta}{\beta + \gamma} \end{aligned}$$

Rechtecke werden auf vergleichbare Weise gesucht. Unterschiede bestehen hier in der Tiefe der Nachbarschaftssuche, sowie dem Phantomstrecken- und Ausrichtungskriterium. Jedes erfolgreich ermittelte Rechteck enthält genau eine Phantomstrecke und alle Realstrecken sind waagerecht oder senkrecht. Lediglich die Phantomstrecke muss nicht waagerecht sein, wenn das Rechteck bereits zwei waagerechte Strecken hat. Rechtecke

Algorithmus 2.2 Dreieckextraktion anhand vorberechneter Strecken

```
1  // Initialisierung
2  T = {}                 // Menge der gefundenen Dreiecke
3  V = {}                 // Menge bereits besuchter Strecken
4
5  // S wird initialisiert mit Strecken von Linienextraktor
6  S = S ∪ halfs(S) ∪ thirds(S)
7
8  für jedes s = (p_b, p_e) ∈ S
9          S = S ∪ {s}
10         S_b = {s' = (p'_b, p'_e) ∈ S | s ≠ s',
11                                  d(p_b, p'_b) ≤ θ_maxDist ∨ d(p_b, p'_e) ≤ θ_maxDist}
12         S_e = {s'' = (p''_b, p''_e) ∈ S | s ≠ s'',
13                                  d(p_e, p''_b) ≤ θ_maxDist ∨ d(p_e, p''_e) ≤ θ_maxDist}
14
15         für jedes s' ∈ S_b
16                 s' ∈ V ?
17                         continue
18
19                 ∃p' ∈ s' : ∃p'' ∈ s'' ∈ S_e : d(p', p'') ≤ θ_maxDist ?
20                         S_e = S_e \ {s''}
21
22                         s'' ∈ V ?
23                                 continue
24
25                         angleCriterion(s, s', s'') ?
26                                 T = T ∪ {(s, s', s'')}
27                 sonst
28                         s_p = createPhantom(s, s')
29                         angleCriterion(s, s', s_p) ?
30                                 T = T ∪ {(s, s', s_p)}
31
32         für jedes s'' ∈ S_e
33                 s'' ∈ V ?
34                         continue
35                 s_p = createPhantom(s, s'')
36                 angleCriterion(s, s_p, s'') ?
37                         T = T ∪ {(s, s_p, s'')}
```

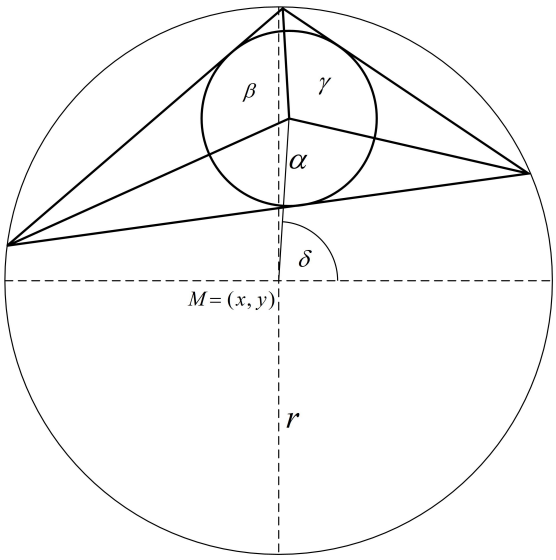


Abbildung 2.9: Dreieckparameter

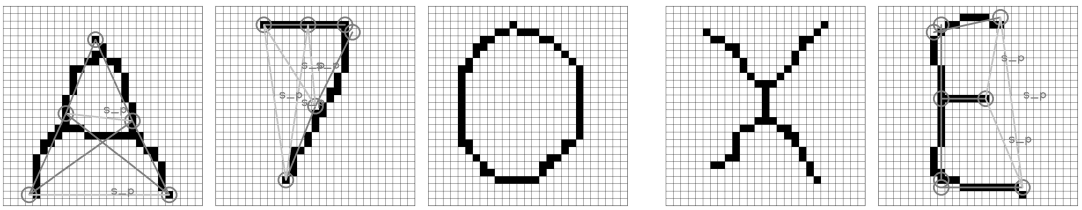


Abbildung 2.10: Beispiele Dreieck- und Rechteckextraktion mit Markierung der Phantomstrecke: (a) Dreiecke, (b) Rechtecke

werden als Merkmalsvektor 2.12 mit ihrem Mittelpunkt und den beiden Ausdehnungen gespeichert.

$$\overrightarrow{f_R} = [x, y, width, heigth] \tag{2.12}$$

Durch die Teilung der erkannten Strecken in halbe und drittel Abschnitte werden mitunter sehr viele Dreiecke oder Rechtecke erkannt. Abbildung 2.10 zeigt einige Beispiele, bei denen nicht alle gefundenen Strukturen eingeblendet sind.

2.3.3 Geschlossene Innenräume

Definition *Ein geschlossener Innenraum ist eine Menge aneinander angrenzender Hintergrundpixel, die komplett von Vordergrundpixel umschlossen sind. Keiner der Hintergrundpixel grenzt an den Bildrand.*

Zur Identifikation aller geschlossenen Innenräume muss es einen eindeutigen Rand aus Hintergrundpixel geben. Deswegen wird jedes Bild zur Vorbereitung auf die Extraktion mit einem ein Pixel breiten Rand aus weißen Pixel vergrößert. In [23] wird ein Verfahren für die Umwandlung von Binärbildern in eine Repräsentation ihrer Grenzlinien[3] vorgestellt. Jedes Binärbild enthält genau zwei Klassen von Zusammenhangskomponenten, Vordergrundkomponenten (1-Pixel) und Hintergrundkomponenten (0-Pixel), wobei die Zusammenhangskomponenten jeweils von genau einer Außen- und genau einer Innengrenze umschlossen sind. Dadurch entsteht eine 1:1 Beziehung zwischen den Zusammenhangskomponenten und den Grenzen im Bild. Ergebnis des Algorithmus ist ein Strukturbaum benachbarter Vorder- und Hintergrundkomponenten beziehungsweise benachbarter Außen- und Innengrenzen. Über diesem Baum sind topologische Strukturanalysen wie das Auffinden geschlossener Innenräume möglich. Im Grenzbaum enthält jeder Knoten die Liste der ihn repräsentierenden Grenzpixel. Dieser Baum wird mit Tiefensuche bis zu allen ersten Innengrenzen traversiert, die jeweils einen geschlossenen Innenraum darstellen. Es wird nicht tiefer gesucht, weil Schriftzeichen keine verschachtelten geschlossenen Innenräume enthalten. Das heißt es gibt in Buchstaben und Ziffern keine geschlossenen Innenräume, die selbst wieder eine Vordergrundkomponente mit geschlossenem Innenraum einschließen. Abschließend wird der Schwerpunkt, die einnehmende Fläche und die größte Ausdehnung in x- und y-Richtung für jeden Innenraum einzeln als Merkmalsvektor 2.13 gespeichert.

$$\overrightarrow{f_{Hole}} = [x, y, a, width, heigth] \tag{2.13}$$

Mit einer passenden Vorverarbeitungskette, die eine morphologische Schließungsoperation einschließt, lassen sich geschlossene Innenräume sehr zuverlässig finden. In Abbildung 2.11 ist zu sehen, dass die verschiedenen Zeichen deutliche Unterschiede in den Eigenschaften der geschlossenen Innenräume aufweisen, was für die Clusterbildung im Entscheidungsbaum notwendig ist. Obwohl die Innenräume der *0* und *4* denselben Schwerpunkt haben, unterscheiden sich die eingeschlossenen Flächen erheblich.

[3] Kette zusammenhängender Vordergrundpixel, die jeweils mindestens einen Hintergrundpixel als Nachbar haben oder umgekehrt.

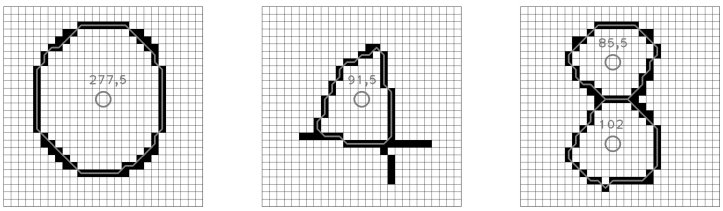


Abbildung 2.11: Beispiele geschlossener Innenräume mit Schwerpunkt und Flächenangabe.

2.3.4 Öffnungen (N, O, S, W)

Jede Zusammenhangskomponente, deren Objektrand[4] nicht vollständig auf ihrer konvexen Hülle liegt, hat mindestens eine Öffnung. Solche Öffnungen sind auch dadurch gekennzeichnet, dass beim Ablaufen des Objektrandes sowohl positive als auch negative Richtungsänderungen auftreten, das heißt Richtungsänderungen nach links und rechts im Kettencode der Kontur [24]. Mit *Öffnung* sind genau diese Einbrüche im Objektrand bezüglich der konvexen Hülle gemeint. Außer Schriftzeichen wie dem *O* oder der *0*, weisen alle solche Öffnungen auf. Im Sinne des Öffnungsmerkmals wird nur eine Teilmenge der Öffnungen mit einer einschränkenden Definition betrachtet.

Definition *Betrachtet man alle aus einer Himmelsrichtung H sichtbaren Pixel des Objektrandes, so begrenzt jedes Pixelpaar p_1, p_2 eine Öffnung, für das die folgenden Eigenschaften gilt:*

- *p_1 und p_2 haben den gleichen Abstand d vom Bildrand H*
- *alle zwischen p_1 und p_2 liegenden Pixel haben einen größeren Abstand als d von H*
- *mindestens einer der beiden Pixel, die an p_1 und p_2 grenzen aber nicht zwischen diesen liegen, hat einen größeren Abstand als d von H*

Abbildung 2.12 ist ein Beispiel für Öffnungen aus Richtung Westen. Es ist zu sehen, dass nach dieser Definition nicht jeder Einbruch in der konvexen Hülle eine Öffnung ist. Die oberste Einbuchtung des Linienzuges im Beispiel ist keine Öffnung. Solche schrägen Einbuchtungen kommen in Schriftzeichen nicht vor und erschweren die Erkennung,

[4] Der Objektrand einer Zusammenhangskomponente ist die oberste beziehungsweise erste Außengrenze im topologischen Strukturbaum (vgl. Abschnitt 2.3.3). Sie beschreibt den Außenrand der Zusammenhangskomponente.

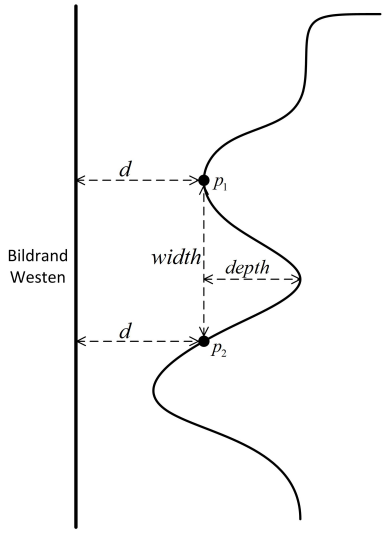


Abbildung 2.12: Öffnungsmerkmal aus Richtung Westen.

weswegen die Beschränkung in der Definition auf die vier Himmelsrichtungen gemacht wird.

Zur Vorbereitung auf die Öffnungssuche wird jedes Bild mit einem Rand aus Hintergrundpixel erweitert. Der Algorithmus 2.3 verwendet eine Sweep-Line[5] Technik und setzt voraus, dass die Sweep-Line an ihren Rändern immer 0-Pixel hat. Um Öffnungen aus allen vier Himmelsrichtungen zu finden, wird der Algorithmus für jede um 90° gedrehte Version des Ausgangsbildes mit folgenden Parametern ausgeführt:

$I(x, y)$ Bildfunktion, die für zwei Ortskoordinaten den Pixelwert liefert.

$\theta_{minDepth}$ Mindesttiefe, die eine Öffnung haben muss, um als solche akzeptiert zu werden.

Zu Beginn wird ein Sweep-Line Array von der Größe $Bildhöhe \times 1$ mit 0 initialisiert. Grundidee ist die Sweep-Line von links nach rechts über das Bild zu schieben und dabei alle Vordergrundpixel mit einer ODER Operation aufzusammeln. Durch Analyse zusammenwachsender 0-1 und 1-0 Übergänge in der Sweep-Line werden alle Öffnungen gefunden. Überschreitet eine Öffnungen den Schwellenwert $\theta_{minDepth}$ in seiner horizontalen Ausdehnung nicht, dann wird sie abgelehnt. Jede Öffnung wird anschließend in einem

[5] Bei einer Sweep-Line handelt es sich um eine gedachte Gerade, die in einer festen Richtung über das Bild geschoben wird. Ähnlich einem Seitenscanner werden in jedem Iterationsschritt nur die Pixel berücksichtigt, die sich gerade unter der Sweep-Line befinden.

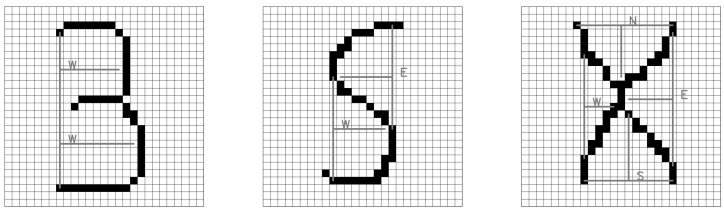


Abbildung 2.13: Beispiele verschiedener Öffnungen.

Merkmalsvektor 2.14 mit ihrem Schwerpunkt (x, y), ihrer Breite *width* (vertikale Ausdehnung) und Tiefe *depth* (horizontale Ausdehnung) gespeichert und je Merkmalsklasse Norden, Osten, Süden und Westen zurückgegeben.

$$\overrightarrow{f_{Opening}} = [x, y, width, depth] \tag{2.14}$$

Abbildung 2.13 zeigt einige typische Zeichen mit den erkannten Öffnungen.

2.3.5 Merkmale im Skelettgraphen

Skelettmerkmale lassen sich direkt aus einem Skelettgraphen ablesen. Deshalb wird für die Extraktion der hier definierten Merkmale *Ende*, *Kreuzung*, *Kreis* und *Einzellinie* zuvor der Skelettgraph auf dem Ausgangsbild berechnet. Der hier vorgestellte Algorithmus setzt voraus, dass das Binärbild in skelettierter Form mit Linienbreiten nicht dicker als 1 Pixel vorliegt. Findet sich im Bild ein 2×2 großer Pixelbereich mit Vordergrund, dann bricht der Algorithmus ab. Eine Vorverarbeitung mit Skelettierungsalgorithmen wie MB Thinning ist nicht möglich. Der Zhang-Suen Algorithmus garantiert hingegen diese Eigenschaft und ist deswegen geeignet zur Vorverarbeitung für die Erstellung des Skelettgraphen. Alternativ könnte das Pixelbild auch vektorisiert [25] und daraus der Skelettgraph berechnet werden. Da aber die Skelettierung ohnehin für andere Merkmalsextraktoren erstellt werden muss, die auf einem Vektorbild nicht arbeiten können, ist der hier verwendete Algorithmus deutlich einfacher und schneller.

Algorithmus 2.4 ist ein rekursiver Algorithmus, der an einem beliebigen Vordergrundpixel startet. Von diesem ausgehend wird das gesamte Pixelbild über seine Nachbarschaft erschlossen, indem Kanten zu Nachbarn hinzugefügt werden und dann mit den Nachbarn genauso verfahren wird. Die Knotenmenge des Graphen wird mit allen Vordergrundpixel initialisiert und die Kantenmenge ist leer. In Abbildung 2.14 sind alle 8 Fälle der rechten oberen Ecke einer 3×3 Nachbarschaft zu sehen. Es werden nicht in jedem Fall alle Nachbarn besucht. Der diagonale Nachbar wird nur bei der Nachbarschaft 2.14(c) direkt

Algorithmus 2.3 Öffnungssuche aus Richtung Westen

```
1  // Initialisierung
2  H = {}                    // Menge der gefundenen Öffnungen
3
4  // h–Tupel enthält die sukzessive veroderten Vordergrundpixel
5  // (ODER Sweep–Line); h ist Bildhöhe
6  S = (s_1, ..., s_h) = (0, ..., 0)
7
8  // ablaufen der Bildspalten
9  für x = 1...w
10         für y = 1...h
11                 s_y = s_y ∨ I(x, y)
12
13         // alle Übergänge finden (I muss leeren Rand haben);
14         // T ist sortierte Liste mit Indizes (y–Koordindate)
15         // der 0–1 oder 1–0 Übergänge, es gilt: |T| mod 2 = 0
16         T = [t ∈ {1...h − 1} | s_t ≠ s_{t+1}]
17         T = sort(T)
18         n_holes = |T| ÷ 2 − 1
19
20         für i = 1...n_holes
21                 t_1 = T(2i)           // Start aktuelle Öffnung
22                 t_2 = T(2i + 1)       // Ende aktuelle Öffnung
23
24                 x_lasth = −1
25                 ∃(x_h, y_h, width, depth) ∈ H : t_1 > y, t_2 < y + width ?
26                         depth = depth + 1
27
28                         // Aufspaltung von Öffnungen
29                         x_lasth = x_h ?
30                                 H = H ∪ {(x_h, t_1, y_h + width − t_1, depth)}
31                                 width = T(2i − 1) − y_h
32
33                         x_lasth = x_h
34                 sonst
35                         H = H ∪ {(x, t_1, t_2 − t_1, 1)}
36
37 // H bereinigen von Öffnungen mit zu geringer Eindringtiefe,
38 // sowie von ungeschlossenen Öffnungen
39 H = H \ {(x, y, width, depth) ∈ H | depth < θ_minDepth ∨
40                                 ∃i ∈ {y...y + width} : s_i = 0}
```

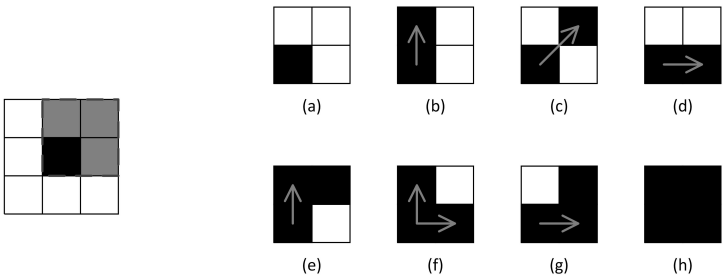


Abbildung 2.14: Erster Teil der Einfügeoperationen für Kanten in der 3×3 Nachbarschaft: (a) nichts tun, (b,d,f) alle Nachbarn besuchen, (c) diagonalen Nachbar besuchen, (e,g) waagerechten und senkrechten Nachbar besuchen, diagonalen Nachbar nicht besuchen, (h) Fehler, Abbruch des Algorithmus

besucht. In den beiden anderen Fällen 2.14(e) und 2.14(g), wo es neben dem diagonalen Nachbar auch noch einen waagerechten oder senkrechten gibt, wird dieser nicht besucht. Das geschieht erst beim rekursiven Aufruf der Funktion an dem waagerechten oder senkrechten Nachbar. Auf diese Weise ist sichergestellt, dass es keine zwei Pfade zu einem diagonalen Nachbarn gibt, wodurch Minikreise entstehen würden. Fälle 2.14(b), 2.14(d) und 2.14(f) werden vollständig abgearbeitet, das heißt jeder Nachbar wird besucht, in Fall 2.14(a) ist nichts zu tun und Fall 2.14(h) führt zum Fehler. Mit den 3 verbliebenen Ecken der 3×3 Nachbarschaft wird nach 90° Drehungen genauso verfahren. Dadurch werden eventuell mehrfach Einfügeoperationen für bereits vorhandene Kanten ausgeführt, was auf einer Menge aber kein Problem darstellt.

Nachdem alle Knoten besucht wurden liegt ein Graph vor, in dem jeder Bildpunkt als Knoten vertreten ist und die entsprechenden Kanten dazwischen. Abbildung 2.15(b) veranschaulicht diesen Graphen. Er wird mit Hilfe der $prune(E, I)$ Funktion noch vereinfacht (vgl. Abbildung 2.15(c)), um einerseits alle uninformativen Knoten zu entfernen und andererseits Kreuzungsfehler zu beheben. Da nur Knoten n mit $Grad(n) = 1$ (Enden) und $Grad(n) > 2$ (Kreuzungen) gesucht werden, können alle Knoten n mit $Grad(n) = 2$ entfernt und die Nachbarn direkt miteinander verbunden werden. An Kreuzungspunkten im Pixelbild kann im Graphen ein Verzerrungsfehler auftreten, wenn die Kreuzung nicht ausschließlich aus waagerechten und senkrechten oder nicht ausschließlich aus diagonalen Pixeln in der 3×3 Nachbarschaft besteht, wie in Bild 2.15(a,b) zu sehen ist. Aus diesem Grund wird jeder Pfad p zwischen zwei Kreuzungen gelöscht, der nur aus Knoten vom Grad 2 besteht und dessen Länge $l \leq \theta_{maxDist}$ ist. Die Kreuzungsknoten werden zu einem Knoten verschmolzen, der alle verbliebenen ausgehenden Kanten der beiden

Algorithmus 2.4 Erstellung Skelettgraph

```
1  // Initialisierung
2  // Knotenmenge ist die Menge der Vordergrundpixel p = (x, y) ∈ I
3  E = {}    // Kantenmenge
4
5  ∃p ∈ I ?
6          visit(p, E, I)
7
8  prune(E, I)
9
10 ---------------------------------------------------------
11 // Definition rekursive Funktion
12 funktion: visit(p = (x, y), E, I)
13         V_pending = {}
14
15         für alle (x_dir, y_dir) ∈ {(+1, +1), (−1, +1), (−1, −1), (+1, −1)}
16                 p_1 = (x + x_dir, y)
17                 p_2 = (x, y + y_dir)
18                 p_3 = (x + x_dir, y + y_dir)
19
20                 p_1 ∈ I ∧ p_2 ∈ I ∧ p_3 ∈ I ?
21                         error!
22
23                 p_3 ∈ I ∧ p_1 ∉ I ∧ p_2 ∉ I ?
24                         ∄e ∈ E : p_3 ∈ e ?
25                                 V_pending = V_pending ∪ {p_3}
26                         E = E ∪ {{p, p_3}}
27                 sonst
28                         p_1 ∈ I ?
29                                 ∄e ∈ E : p_1 ∈ e ?
30                                         V_pending = V_pending ∪ {p_1}
31                                 E = E ∪ {{p, p_1}}
32                         p_2 ∈ I ?
33                                 ∄e ∈ E : p_2 ∈ e ?
34                                         V_pending = V_pending ∪ {p_2}
35                                 E = E ∪ {{p, p_2}}
36
37         für alle p_pending ∈ V_pending
38                         visit(p_pending, E, I)
```

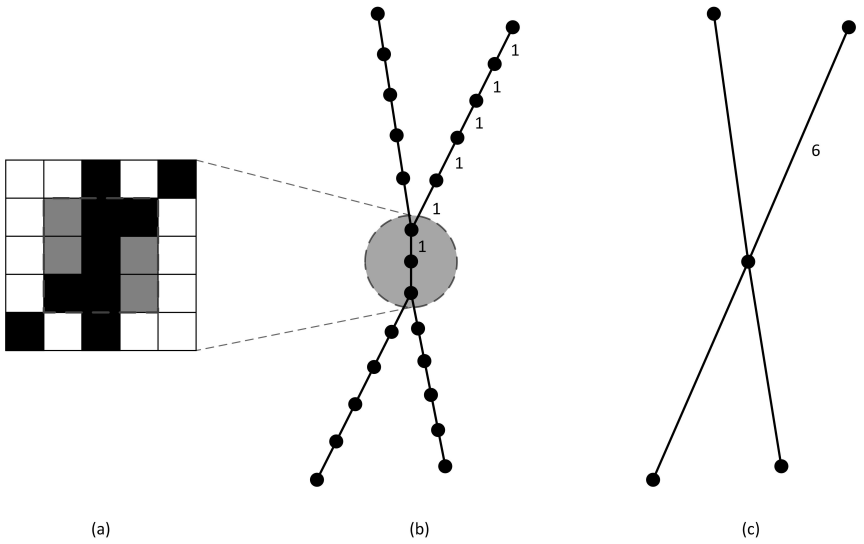


Abbildung 2.15: Beispiel Grapherstellung: (a) Kreuzung im Pixelbild, (b) vollständiger Graph mit Verzerrung der Kreuzung, (c) vereinfachter Graph

Kreuzungen verbindet. Der Schwellenwert $\theta_{maxDist}$ ist streng genommen 2 für das Verzerrungsproblem, kann aber auch toleranter eingestellt werden, um auseinander gezogene Kreuzungen im Pixelbild zusammenwachsen zu lassen. Bei Ausgangsbildern mit dicken Pixellinien, zeigen die Skelettierungsalgorithmen dieses Problem. Ein Schwellenwert von $\theta_{maxDist} = 6$ hat sich als ziemlich zuverlässig herausgestellt. Zeichen wie das X und die *8* erhalten zuverlässig eine Kreuzung im Graphen, während die beiden Kreuzungen vom Buchstaben B trotzdem erhalten bleiben. Die Verzerrungsentfernung muss vor der Knotenentfernung durchgeführt werden, weil sonst alle Kreuzungen zusammenwachsen. Nach Entfernung aller Knoten vom Grad 2 ist nämlich jede Kreuzung mit genau einer Kante verbunden und der Pfad zwischen diesen mit Länge 1 wäre kürzer als der Schwellenwert $\theta_{maxDist}$.

In dem vereinfachten Graph lassen sich die gesuchten Merkmale nun einfach ablesen. Jeder Knoten ist entweder ein Ende oder eine Kreuzung und wird in Vektor 2.15 gespeichert. Enden sind Kreuzungen mit Grad 1.

$$\overrightarrow{f_{Intersection}} = [x, y, degree] \tag{2.15}$$

Kreise werden mit Knotenmarkierung und Tiefensuche beginnend an einem beliebigen Ende gefunden. Hat ein Graph genau 2 Knoten und eine Kante zwischen diesen, dann war

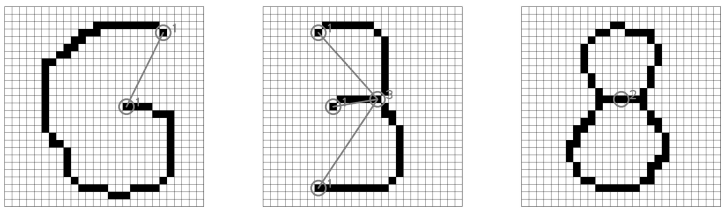


Abbildung 2.16: Beispiele vereinfachter Skelettgraphen mit Gradzahlbeschriftung der Knoten.

das Pixelbild eine Einzellinie. Kreise und Einzellinienmerkmal werden als quantitative Merkmale mit diskreten Werten in einem 1-dimensionalen Vektor gespeichert, Kreise mit der Anzahl auf ihnen liegender Knoten (vgl. Vektor 2.16) und Einzellinien als einfacher boole'scher Vektor 2.17 repräsentiert mit den natürlichen Zahlen 0 und 1.

$$\overrightarrow{f_{GraphCircle}} = [nodes] \tag{2.16}$$

$$\overrightarrow{f_{SingleLine}} = [0 \mid 1] \tag{2.17}$$

Das Merkmal Einzellinie ist ein redundantes Merkmal, da die Information auch im Ende- und Kreuzungsmerkmal steckt. Sie wird aber in diesem Vektor vom Ort abstrahiert und bietet damit gewisse Vorteile im Entscheidungsbaum, wie sich später zeigen wird. Abbildung 2.16 zeigt neben einer Einzellinie auch Beispiele mit Kreisen und Kreuzung.

2.3.6 Symmetrie

Um ein Maß für die Symmetrie eines Zeichens anzugeben muss zunächst die Frage nach der Symmetrieachse gestellt werden. Bei Betrachtung symmetrischer lateinischer Buchstaben und arabischer Zahlen fällt auf, dass diese entweder symmetrisch in horizontaler oder vertikaler Ausrichtung sind und die Symmetrieachse in der Mitte liegt. Da keine verzerrten oder geneigten Zeichen erwartet werden und alle nach der Normierung zentriert sind, wird die Symmetrieachse in der Bildmitte angenommen, um so einen vergleichbaren Symmetriewert zu berechnen. Eine einfache Partnerpixelsuche um die Symmetrieachse herum führt bei digitalisierten und skelettierten Binärbildern zu sehr schlechten Ergebnissen, da selten Partnerpixel sich noch genau gegenüber liegen. Deswegen werden zwei Toleranzen eingeführt. Algorithmus 2.5 wird zum einen mehrfach mit Symmetrieachsen im Intervall $x_{sym} = I_{width} \pm \delta_{width}$ ausgeführt und der höchste Symmetriewert ausgewählt. Zum anderen werden Pixelpartnerschaften nicht streng, sondern in einer kleinen

Algorithmus 2.5 Berechnung der horizontalen Symmetrie

```
1  // Initialisierung
2  v = 0
3  n = 0
4
5  für  y = 1 ... height(I)
6           P = {x ∈ {1 ... width(I)} | I(x, y) = 1}
7           n = n + |P|
8
9           für jedes x ∈ P
10                  x_corr = x_sym + (x_sym − x)
11
12                  x_corr = x  ∨  I(x_corr, y) = 1  ?
13                          v = v + 1
14                          continue
15
16                  I(x_corr − 1, y) = 1  ∨  I(x_corr + 1, y) = 1  ?
17                          v = v + v_tolerance
18
19 // Symmetrie ist durchschnittlicher Vote
20 s = v/n
```

Nachbarschaft gesucht und bewertet. Abbildung 2.17 zeigt, dass der Algorithmus für jeden Pixel ein Symmetriemaß ermittelt. Befindet sich auf der gegenüberliegenden Seite eines Pixels p auch ein Pixel, so ist die Symmetriestärke v von p gleich 1. Existiert dort kein Pixel, aber in der direkten horizontalen Nachbarschaft mindestens einer, dann ist die Symmetriestärke v von p gleich $v_{tolerance}$. Mit diesen beiden Änderungen wird der Algorithmus unempfindlicher gegenüber Rauschen. Werte für horizontale (s_h) und vertikale (s_v) Symmetrie werden durch Drehung des Ausgangsbildes errechnet und in Merkmalsvektor 2.18 gespeichert.

$$\overrightarrow{f_{Sym}} = [s_h, s_v] \tag{2.18}$$

Werte von etwa 55% bis 65% für symmetrische Zeichen und 15% bis 25% für unsymmetrische Zeichen haben sich bei Versuchen gezeigt, wie in Abbildung 2.18 beispielhaft zu sehen ist.

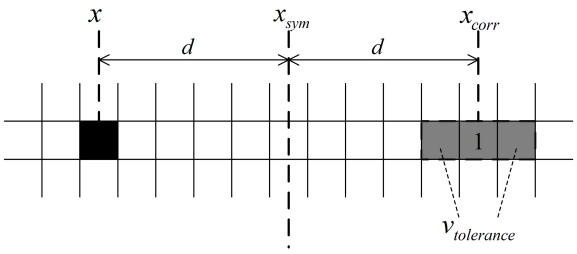


Abbildung 2.17: Ermittlung des Symmetriemaß für Pixel

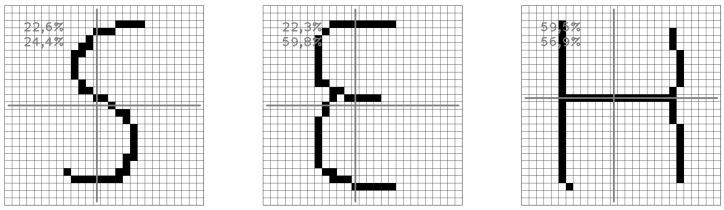


Abbildung 2.18: Symmetriebeispiele: (a) schlechte Symmetrie in beiden Richtungen, (b) gute Symmetrie nur in vertikaler Richtung, (c) gute beidseitige Symmetrie

2.3.7 Relative Fläche

Um vergleichbare Verhältnisse zu schaffen wird jedes Ausgangsbild vor der Berechnung der relativen Fläche skelettiert und ohne Erhaltung der Seitenverhältnisse normalisiert. Algorithmus 2.6 berechnet auf dem normalisierten Bild die relative Fläche a durch das Verhältnis von Vordergrundpixel zur Gesamtfläche und speichert sie in Merkmalsvektor 2.19.

$$\overrightarrow{f_A} = [a] \tag{2.19}$$

Abbildung 2.19 zeigt, dass Unterschiede in den Flächenwerten von bis zu 50% für verschiedene Zeichen existieren.

2.3.8 Schwerpunkt

Analog zur relativen Fläche wird vor der Berechnung des Zeichenschwerpunkts jedes Bild skelettiert und normiert. Algorithmus 2.7 berechnet dann auf dem Bild das arithmetische Mittel in x- und y-Richtung und speichert es in Merkmalsvektor 2.20.

$$\overrightarrow{f_{Centroid}} = [x, y] \tag{2.20}$$

Algorithmus 2.6 Berechnung der relativen Fläche

```
1  // Initialisierung
2  a = 0                    // Wert der relativen Fläche
3
4  // Aufsummieren aller Helligkeitswerte
5  für x = 1...width(I)
6          für y = 1...height(I)
7                  a = a + I(x, y)
8
9  // über vollständig ausgefülltes Bild normieren
10 a = a ÷ (width(I) · height(I))
```

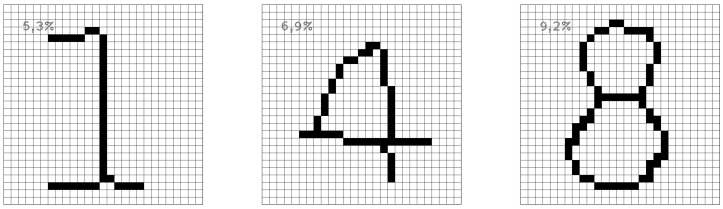


Abbildung 2.19: Typische Beispiele relativer Flächenwerte.

Algorithmus 2.7 Berechnung des Zeichenschwerpunktes

```
1  // Initialisierung
2  C = (c_x, c_y) = (0,0)              // Schwerpunkt
3  n_pixel = 0                         // Zähler Vordergrundpixel
4
5  // alle Pixelkoordinaten werden anhand ihres
6  // Helligkeitswertes zum Schwerpunkt addiert
7  für x = 1...width(I)
8          für y = 1...height(I)
9                  c_x = c_x + (x * I(x,y))
10                 c_y = c_y + (y * I(x,y))
11                 n_pixel = n_pixel + I(x,y)
12
13 // über vollständig ausgefülltes Bild normieren
14 c_x = c_x ÷ n_pixel
15 c_y = c_y ÷ n_pixel
```

Kapitel 3

Bewertungsbaum

Klassische Entscheidungsbäume [26, 27] haben das Problem starre Entscheidungen auf Grundlage eines Merkmals zu treffen. Diese Eigenschaft kann nicht die Anforderung der Fehlertoleranz zwischen den Merkmalen erfüllen, weswegen das Modell zu einem Bewertungsbaum modifiziert wird. Ein Entscheidungsbaum trifft auf Grundlage des Merkmals am Knoten eine Entscheidung und bleibt dabei. Insbesondere ändert sich diese Entscheidung auch nicht bei mehrfachem Durchlaufen des Baumes. Der Bewertungsbaum trifft aufgrund einer kindabhängigen Wahrscheinlichkeitsverteilung eine Pfadwahl. Dadurch ist es möglich, das bei mehrfachem Durchlaufen des Baumes nicht immer der gleiche Pfad gewählt wird. Dieser Effekt ist beabsichtigt.

Das Prinzip der Objektdefinition über seine Eigenschaften erfordert, dass Merkmale erstens aus unterschiedlichen Merkmalsräumen und zweitens in beliebiger Anzahl als Objektbeschreibung zugelassen werden. Gerade bei Buchstaben und Ziffern in Pixelbildern ist es wenig wahrscheinlich, dass ein universelles Merkmal existiert, auf das man die Bildinformation reduzieren kann, um anhand dessen eine Klassifikation durchzuführen. Da der Bewertungsbaum auch die *Welt* berücksichtigt, ohne jemals Beispiele dafür präsentiert zu bekommen, muss die Merkmalsmenge so gewählt werden, dass alle zu erkennenden Objekte vollständig spezifiziert werden können (vgl. Abschnitt 2.1).

3.1 Agglomerative Clusterbildung

Bei der Erstellung des Bewertungsbaumes wird für jeden Knoten eine Clusteranalyse auf allen vorkommenden Merkmalen eines bestimmten Typs in der Trainingsmenge ausgeführt. Es sollen dabei keine Cluster abhängig von den Klassen der einzelnen Merkmale gefunden werden, sondern lediglich Häufungen der Merkmale im entsprechenden Merkmalsraum. Um diese versteckten Strukturen aufzudecken wird ein Verfahren mit

unsupervised learning[1] benötigt. Hierarchische Clusteranalyseverfahren [28, 29] erfüllen diese Anforderung. Sie werden unterteilt in teilende und anhäufende Verfahren. Während Verfahren mit teilendem (*divisive*) Ansatz von einem großen Cluster ausgehen, was nach bestimmten Kriterien immer weiter zerlegt wird, beginnen anhäufende (*agglomerative*) Verfahren mit einem Cluster pro Datenpunkt und lassen diese nach und nach zusammenwachsen. Für die Knoten des Bewertungsbaumes wird die einfache agglomerative Clusterbildung verwendet. Sie bietet den Vorteil über verschiedene Parameter die Clusterbildung beeinflussen zu lassen und sehr einfach zu sein. Geschwindigkeitsnachteile durch die Laufzeit $O(n^2)$ können vernachlässigt werden, weil die Datenmengen pro Knoten nicht allzu groß sind und die Clusteranalyse nur beim Training des Baumes nicht aber bei seiner Verwendung zur Zeichenerkennung durchgeführt werden muss. Partitionierende Clusteranalyseverfahren [30] sind ungeeignet, weil die Anzahl der Cluster zuvor bekannt sein muss.

Die agglomerative Clusteranalyse beginnt mit der Zuweisung jedes Datenpunktes zu jeweils einem eigenen Cluster. Iterativ wachsen nun nach und nach die Cluster zusammen, bis alle Datenpunkte einem großen Cluster angehören oder ein entsprechendes Abbruchkriterium eintritt. Für die Entscheidung welche zwei Cluster C_1 und C_2 aus der Clustermenge C im aktuellen Iterationsschritt zu dem neuen Cluster C_{union} zusammenwachsen gibt es verschiedene Strategien. Üblich sind die beiden Distanzfunktionen single-linkage (vgl. Menge 3.1) und complete-linkage (vgl. Menge 3.2) mit einer Abstandsfunktion $d(c_1, c_2)$, für die hier eine L-Norm verwendet wird.

$$C_{union} = \left\{ c \in C_1 \cup C_2 \mid C_1 \cap C_2 = \emptyset,\ \min_{\substack{C_1 \subset C \\ C_2 \subset C}} \left\{ \min_{\substack{c_1 \in C_1 \\ c_2 \in C_2}} \{d(c_1, c_2\} \right\} \right\} \tag{3.1}$$

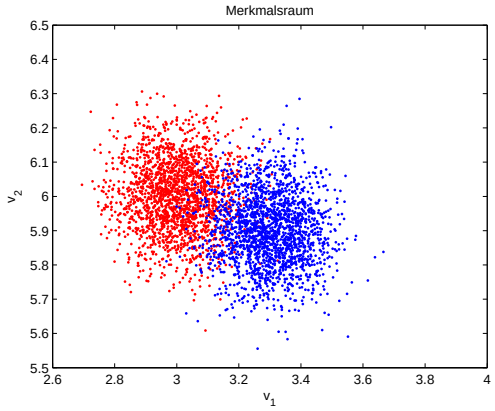
$$C_{union} = \left\{ c \in C_1 \cup C_2 \mid C_1 \cap C_2 = \emptyset,\ \min_{\substack{C_1 \subset C \\ C_2 \subset C}} \left\{ \max_{\substack{c_1 \in C_1 \\ c_2 \in C_2}} \{d(c_1, c_2\} \right\} \right\} \tag{3.2}$$

Die single-linkage Methode lässt die beiden Cluster zusammenwachsen die am nächsten beieinander liegen. Das heißt, das Clusterpaar deren minimaler Punktabstand kleiner ist als von allen anderen Clusterpaaren wird zu einem neuen Cluster verschmolzen. Dagegen verschmilzt die complete-linkage Methode das Clusterpaar, deren maximaler Punktabstand kleiner ist als von allen anderen Clusterpaaren. Der Prozess kann in einer Art Verschmelzungsbaum, der Dendrogramm genannt wird, visualisiert werden. Es handelt

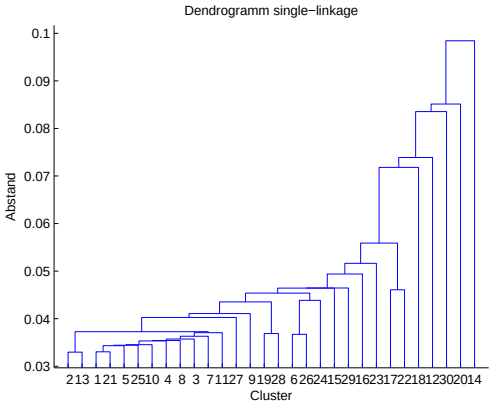
[1] Unsupervised learning, auch unüberwachtes Lernen, bezeichnet Techniken, die versuchen Strukturen oder Zusammenhänge in unklassifizierten Daten aufzudecken.

sich dabei um ein Diagramm, dass über den Abstand zeigt, wann zwei Cluster sich zu einem verbinden. Da es wenig sinnvoll ist alle Cluster bis zu einem einzigen Cluster zusammenwachsen zu lassen, muss der Prozess vorzeitig beendet werden. Das kann über den Abstand geschehen, indem keine Cluster mehr verbunden werden, wenn eine Abstandsschwelle θ überschritten wird. Im Dendrogramm stellt sich das dadurch dar, dass der Baum in einer bestimmten Tiefe durchtrennt wird. Alle dabei abgeschnittenen Teilbäume stellen ein Cluster dar. Unterschiede in der single-linkage und complete-linkage Methode zeigen sich bei der Bedeutung des Schwellenwertes θ. Er gibt den Mindestabstand zwischen zwei Clustern bei der single-linkage Methode an, aber die maximale Ausdehnung eines Clusters bei der complete-linkage Methode. In Abbildung 3.1 und 3.2 sind die Unterschiede deutlich gemacht. Beide zeigen die agglomerative Clusteranalyse auf einer Datenmenge, die aus zwei Normalverteilungen besteht. In Abbildung 3.1 sind die beiden Punktmengen überlagert, haben aber die gleiche Standardabweichung. Die Punktmengen in Abbildung 3.2 liegen auffällig auseinander, haben aber eine um Faktor 3 unterschiedliche Standardabweichung. Angenommen die Klassen der beiden Punktmengen in den Datenmengen sind nicht bekannt und mit Hilfe der agglomerativen Clusteranalyse sollen sie gefunden werden, dann hat die single-linkage Methode in Abbildung 3.1 keinen Erfolg. Es kann kein Mindestabstand angegeben werden, so dass die Cluster getrennt würden ohne dass sehr viele kleine Cluster am Rand der Punktmengen zurück bleiben. Mit Hilfe der complete-linkage Methode kann man ein recht gutes Ergebnis erzielen, wenn man die maximale Ausdehnung der Cluster kennt. Für Datenmengen wie in Abbildung 3.2 ist das Problem ein anderes. Verwendet man hier die single-linkage Methode ist ein sehr gutes Ergebnis zu erzielen, weil die Datenmengen gut getrennt sind. Durch die unterschiedliche Ausdehnung versagt aber die complete-linkage Methode. Gibt man für sie ein θ in der Größenordnung der Ausdehnung der rechten Punktmenge an, dann werden drei Cluster gefunden. Gibt man ein θ für die Ausdehnung der linken Punktmenge an, dann entstehen Cluster, von denen einer Teile beider Punktmengen enthält. Das geschieht durch die wenigen Punkte zwischen den beiden Punktmengen.

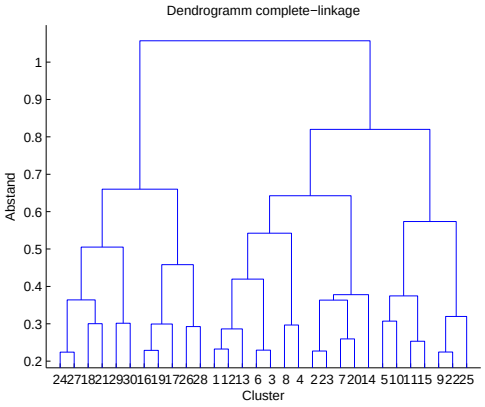
Im Fall des Bewertungsbaumes ist die Ausdehnung potentieller Merkmalscluster nicht bekannt. Allerdings sollen die Cluster an jedem Knoten die Merkmalsmenge deutlich separieren, wodurch die Angabe eines Mindestabstandes möglich wird und die single-linkage Methode zur Anwendung kommt.



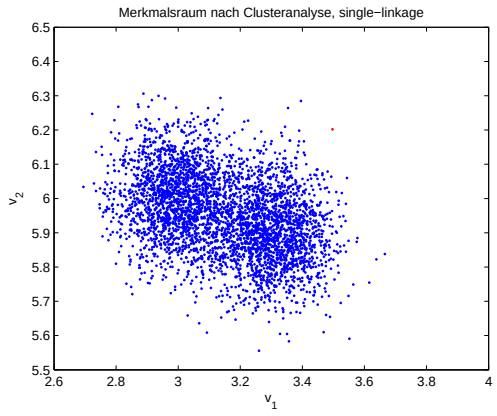
(a)



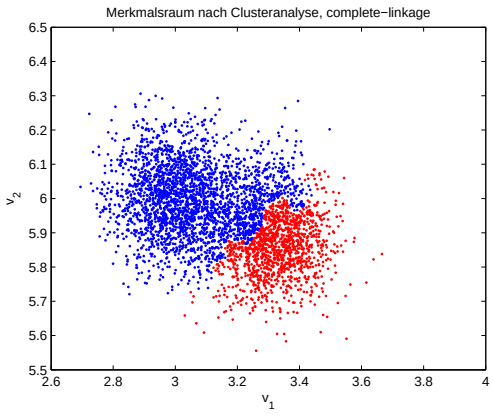
(b)



(d)



(c)



(e)

Abbildung 3.1: Agglomerative Clusterbildung: (a) 2 leicht überschneidende Normalverteilungen mit gekennzeichneten Zugehörigkeiten, (b) Dendrogramm für single-linkage Distanz, (c) Ergebnis der Clusteranalyse für $\theta = 0,09$, (d) Dendrogramm für complete-linkage Distanz, (e) Ergebnis für $\theta = 1$

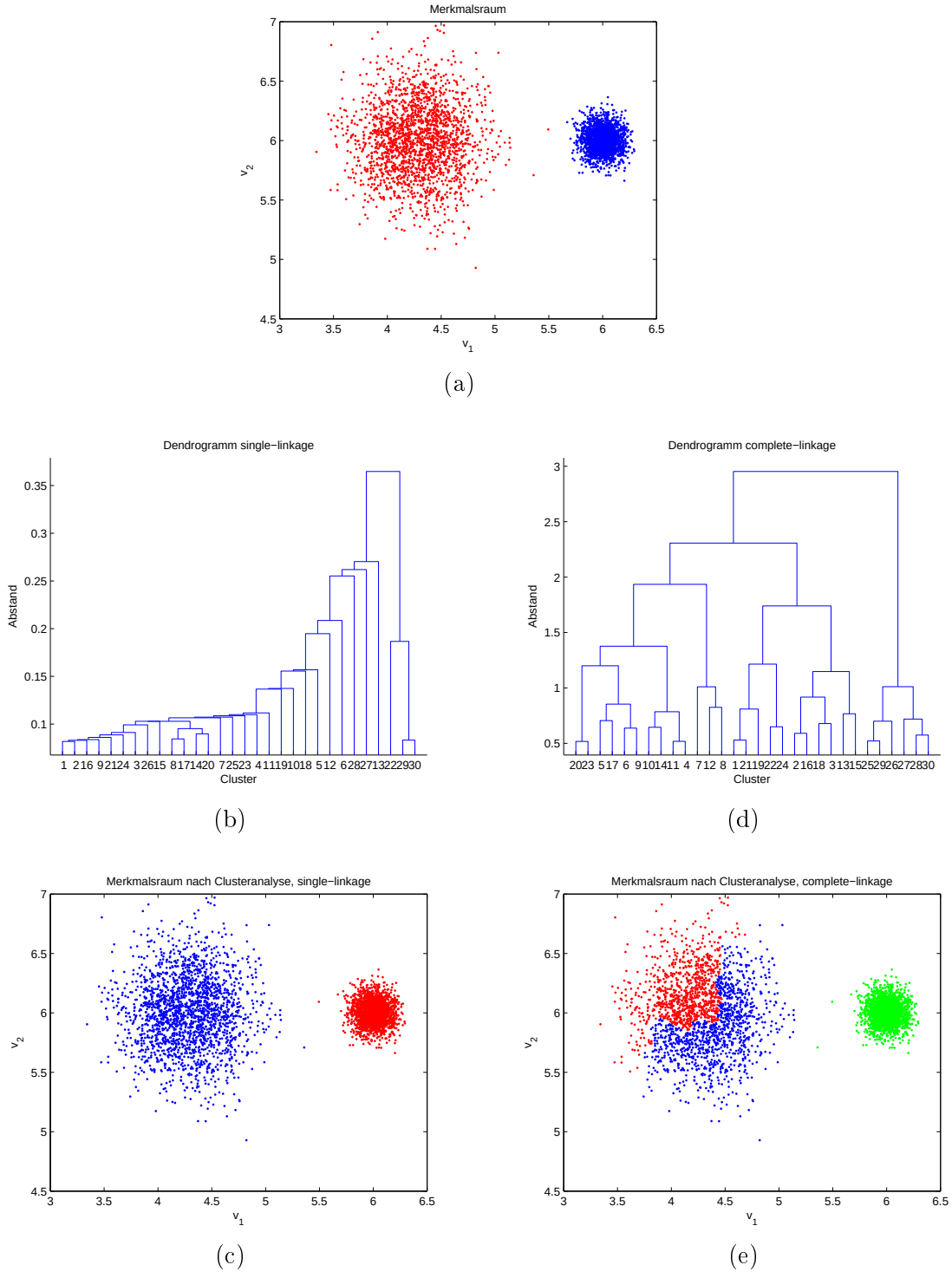


Abbildung 3.2: Agglomerative Clusterbildung: (a) 2 getrennte Normalverteilungen mit unterschiedlicher Größe und gekennzeichneten Zugehörigkeiten, (b) Dendrogramm für single-linkage Distanz, (c) Ergebnis der Clusteranalyse für $\theta = 0,3$, (d) Dendrogramm für complete-linkage Distanz, (e) Ergebnis für $\theta = 2$

3.2 Mehrdimensionale Normalverteilung

Ein Knoten des Bewertungsbaumes ist ein schwacher Klassifikator für das durch ihn repräsentierte Merkmal. Dazu speichert er Informationen für jedes Cluster aus der agglomerativen Clusteranalyse. Merkmale sind bis auf einige quantitative Ausnahmen im Allgemeinen mehrdimensional und für jedes Cluster im Merkmalsraum wird eine mehrdimensionale Normalverteilung angenommen. Das bedeutet es wird davon ausgegangen, dass sich häufende Merkmale um einen Mittelwertvektor mit Standardabweichung und Varianz normalverteilt streuen. Zum Beispiel wird erwartet, dass es für den oberen Kreis einer *8* einen Mittelpunkt gibt, um den alle Exemplare normalverteilt sind. Ist das nicht der Fall, weil es zum Beispiel zwei verschiedene Typen von Achten gibt und jede einen anderen Mittelwert für den oberen Kreis hat, dann wird das im Baum dadurch repräsentiert, dass zwei verschiedene Pfade für die beiden Typen entstehen und jeder für sich folgt wieder einer Normalverteilung. Liegt hingegen in einem Cluster wirklich keine Normalverteilung vor, so ist die Annahme der Normalverteilung unproblematisch, so lange die Cluster weit genug voneinander entfernt sind. Die Darstellung eines Clusters als mehrdimensionale Normalverteilung folgt nur der Idee ein Cluster als ein Ellipsoid darzustellen und mit einer entsprechenden Dichtefunktion unbekannte Merkmalsvektoren einem der Cluster oder dem Nichts des Merkmalsraumes zuzuordnen. Ist die Punktmenge eines Cluster nicht normalverteilt, so kann der Ellipsoid sehr große Ausmaße annehmen und viel mehr Raum abdecken als die Messpunkte beschreiben. Das kann durch verschiedene Möglichkeiten, wie dem Zurückstellen des Merkmals oder der Anpassung von θ bei der agglomerativen Clusterbildung, verhindert werden.

Formel 3.3 ist die allgemeine Dichtefunktion für mehrdimensionale Normalverteilungen. Sie gibt für einen Mittelwertvektor μ und Kovarianzmatrix Σ die Wahrscheinlichkeit des Auftretens eines d-dimensionalen Vektor x an.

$$f(x) = \frac{1}{\sqrt{|\Sigma|\,(2\pi)^d}} e^{1\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^T} \tag{3.3}$$

$$|\Sigma| : \textit{Determinante von } \Sigma$$

Die Parameter dieser Dichtefunktion für ein Cluster sind nicht bekannt und müssen anhand der Stichproben (Trainingsmenge), die einem Cluster zugeordnet sind, geschätzt werden. Enthält ein Cluster n Merkmalsvektoren der Länge d (vgl. Matrix 3.4), so lassen

sich Mittelwertvektor μ und Kovarianzmatrix Σ leicht mit Formel 3.5 und 3.6 schätzen.

$$V = (v_{ij})_{i \in \{1...n\}, j \in \{1...d\}} \tag{3.4}$$

$$\mu = \left(\frac{1}{n} \sum_{k=1}^{n} v_{kj} \right)_{j \in \{1...d\}} \tag{3.5}$$

$$\Sigma = \left(\frac{1}{n-1} \sum_{k=1}^{n} (v_{ki} - \mu_i) \cdot (v_{kj} - \mu_j) \right)_{i,j \in \{1...d\}} \tag{3.6}$$

Abbildung 3.3 veranschaulicht die Idee. Zu sehen sind Isoflächen[2] der Dichtfunktionen von zwei Clustern. Der Isowert der Isoflächen ist der Dichtewert des dreifachen Standardabweichungsvektors. Die Idee ist, alle Merkmalsvektoren, die innerhalb eines bestimmten Intervalls um einen Mittelwertvektor liegen, dem jeweiligen Cluster zuzuordnen. Alle Merkmalsvektoren, die keinem Cluster zugeordnet werden, also außerhalb aller Clusterintervalle liegen, werden der Welt zugeordnet. In der Abbildung veranschaulichen die Isoflächen diese Clustergrenzen. Jeder Ellipsoid ist ein Cluster. Es werden die Dichtefunktionen nicht genutzt, um die Wahrscheinlichkeit für eine Clusterzugehörigkeit, also den wahrscheinlichsten Cluster, zu bestimmen. Der Wahrscheinlichkeitswert eines Merkmalsvektors wird nur genutzt, um einen Vergleich mit den Clustergrenzen zu machen. Sind zwei Cluster so nah beieinander, dass die Ellipsoide sich schneiden, dann ist die Grenzfläche dort der Dichtewert der Schnittlinie. Das bedeutet wenn ein Merkmalsvektor in den Intervallgrenzen von mehr als einem Cluster liegt, dann wird der Cluster mit dem größten Wahrscheinlichkeitswert ausgewählt, aber nur dann.

3.3 Merkmalsabhängigkeit

Für jeden Knoten im Bewertungsbaum muss der Trainingsalgorithmus ein Merkmal aus der Menge der noch zur Verfügung stehenden Merkmale auswählen. Diese Auswahl kann nicht beliebig erfolgen. Manche Merkmale bilden nur auf einer Teilmenge der zu erkennenden Zeichen gut separierte Cluster aus. Führt man die Clusteranalyse auf Merkmalsmengen solcher Merkmale aus, wenn noch viele Merkmale von Zeichen enthalten sind, die weit über den Merkmalsraum streuen, ist es möglich, dass diese den Merkmalsraum

[2] In Bezug auf Funktionen sind Isoflächen alle Punkte, die den gleichen Funktionswert haben. Es lassen sich so 4-dimensionale Funktionen (Funktionen mit 3 Variablen) eingeschränkt 3-dimensional darstellen, indem alle Punkte im Raum eingefärbt werden, die einen angegebenen Funktionswert haben. Hier wird der Verlauf eines konkreten Dichtewertes für zwei Dichtefunktionen visualisiert.

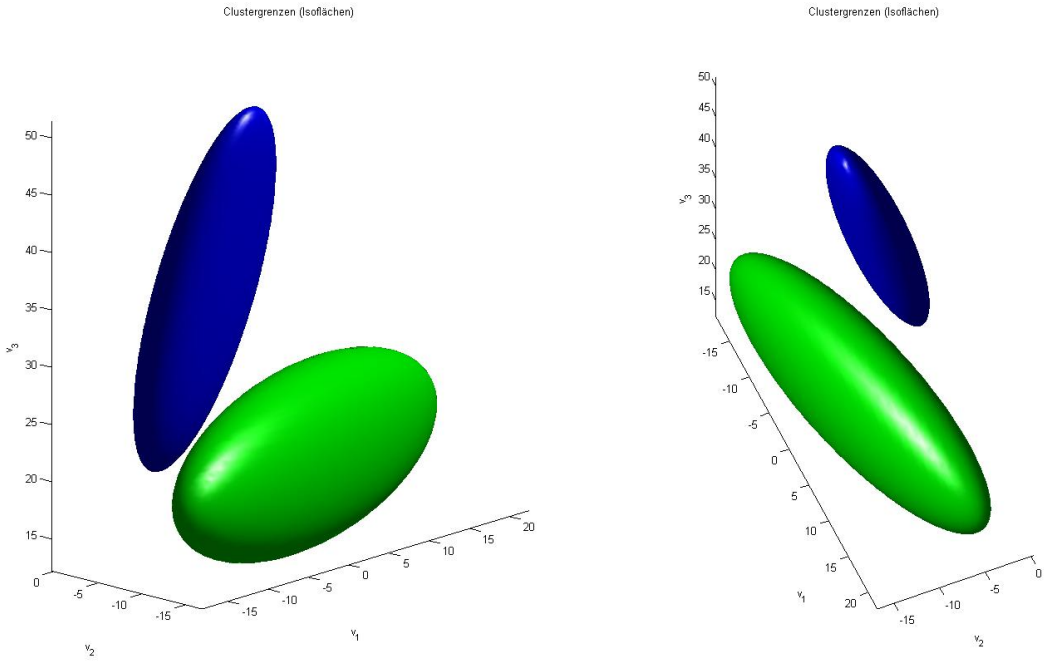


Abbildung 3.3: Visualisierung von zwei 3-dimensionalen Normalverteilungen. Dargestellt sind die Isoflächen des Dichtewertes der dreifachen Standardabweichung. Sie repräsentieren die Grenzflächen von zwei Clustern im Merkmalsraum.

so stark verwaschen, dass sich wenige oder im Extremfall sogar nur ein einziges Cluster ausbildet. Dann führt dieses Merkmal am Knoten zu einem Kind und hat gar keine Klassifikationsfunktion. Wurde ein Merkmal einmal verwendet, steht es nicht mehr zur Verfügung. Das kann soweit gehen, dass ein verfrüht verwendetes Merkmal eine komplette Klassifikation von allen Zeichen unmöglich macht, weil seine Klassifikationsleistung auf einer geringeren Merkmalsmenge gebraucht wird.

In Abbildung 3.4 sind drei verschiedene Merkmalsräume zu sehen und die Verteilung der Merkmale von vier Objektklassen in den Räumen. Eine agglomerative Clusteranalyse kann im Merkmalsraum M_1 keine Cluster separieren, weil alle Merkmale sich überlagern. Es wird nur ein großer Cluster gefunden, wenn alle Merkmale an den Clusteralgorithmus übergeben werden. In Merkmalsraum M_2und M_3überlagern oder vermischen sich andere Objektklassen. Wählt der Trainingsalgorithmus des Bewertungsbaumes als ersten Merkmalstyp M_1 aus, dann findet keine Unterteilung der Merkmalsmenge statt (vgl. Abbildung 3.5) Welches Merkmal nun im darauffolgenden Iterationsschritt ausgewählt wird spielt keine Rolle mehr. Typ M_2 kann die Merkmalsmenge dann zwar in Objekte der Klassen A, B und C, D aufteilen, aber Merkmalstyp M_3 schafft anschließend nur noch eine Trennung von C und D, aber nicht mehr von A und B. Im Fall der Wahl von M_3 als zweiten Merkmalstyp sieht es ähnlich aus. Abbildung 3.6 zeigt, dass es eine Kombination der Merkmalstypen gibt, die die notwendige Trennungsleistung erreicht. Zwischen den Merkmalstypen kann solch eine Abhängigkeit bestehen, die nicht direkt erkennbar ist. Für die Merkmalsauswahl muss der Trainingsalgorithmus im naiven Verfahren alle klassenbasierten Teilmengen aller Merkmalsmengen bestimmen, die Klassifikationsleistung nach der Clusterbildung für jede Teilmenge analysieren und darin eine funktionierende Kombination suchen. Dieser Ansatz zur Auflösung von Merkmalsabhängigkeiten ist zu aufwendig. Die in dieser Arbeit gemachten Versuche beschränken sich auf die manuelle Auswahl von Merkmalen. Um den Prozess während des Trainings zu automatisieren ist eine geeignete Methode erforderlich, die die Merkmalsabhängigkeiten effizient auflösen kann.

3.4 Vorbereitung

In diesem Abschnitt werden alle Vereinbarungen und Festlegungen getroffen, um den Algorithmus zum Training des Bewertungsbaumes im nächsten Abschnitt zu formalisieren. Bei dem Bewertungsbaum handelt es sich um eine vektorbasierte Klassifikationsmethode. Jedes verwendete Merkmal ist als Vektor darzustellen und alle Merkmale eines beliebigen Merkmalstypen müssen mit einer L-Norm auf Ähnlichkeit vergleichbar sein, wegen

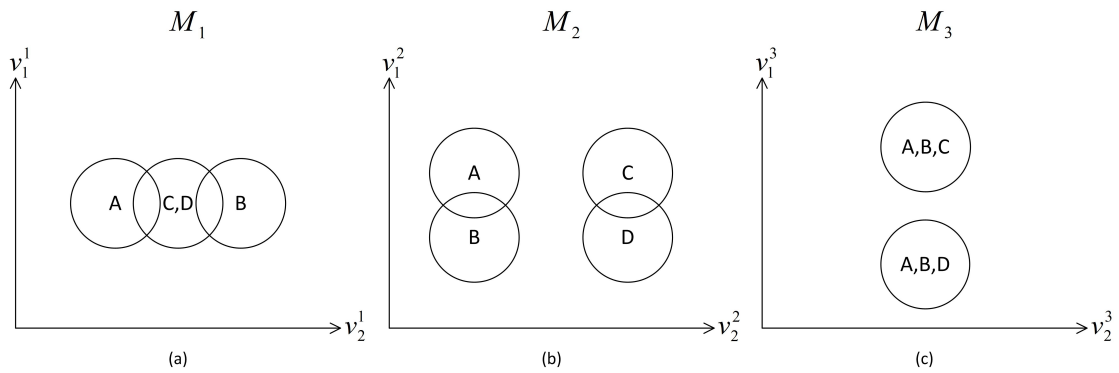


Abbildung 3.4: Beispiele für 3 Merkmalsräume M_1, M_2, M_3 und Merkmalsverteilung von 4 Klassen: (a) Merkmale aller 4 Klassen überlagern sich, (b) Merkmale von A und B, sowie von C und D überlagern sich, (c) Merkmale von A und B sind jeweils mit Merkmalen von C und D vermischt

der Repräsentation der Cluster mit der Dichtefunktion der Normalverteilung. In Rahmen dieser Arbeit wurden die L1- und L2-Norm als Abstandsmaß für die Clusterbildung verwendet. Ein signifikanter Unterschied war nicht zu erkennen.

Zu Beginn des Algorithmus steht eine Menge C mit n Beispielbildern bereit (vgl. Menge 3.7), die je ein Zeichen als einzelne Zusammenhangskomponente darstellen. Außerdem ist Funktion 3.8 gegeben, die jeder Zusammenhangskomponente c seine Klasse l zuweist.

$$C = \{c_i \mid i = 1 \ldots n\} \tag{3.7}$$

$$\begin{aligned} L : C &\rightarrow \{1 \ldots K\} \\ L(c) &= l \end{aligned} \tag{3.8}$$

Dem Trainingsalgorithmus werden t Merkmalsextraktoren bereitgestellt. Jeder von ihnen erzeugt Merkmalsvektoren für einem bestimmten Merkmalstyp m_i, die in Menge 3.9 zusammengefasst sind.

$$M = \{m_i \mid i = 1 \ldots t\} \tag{3.9}$$

Merkmalsvektoren f^i eines bestimmten Merkmalstyps m_i können je Zusammenhangskomponente c in unterschiedlicher Anzahl vorliegen. Deswegen gibt es für jede Zusammenhangskomponente c in Bezug auf ein Merkmal m_i ein spezifisches k (vgl. Aussage 3.10).

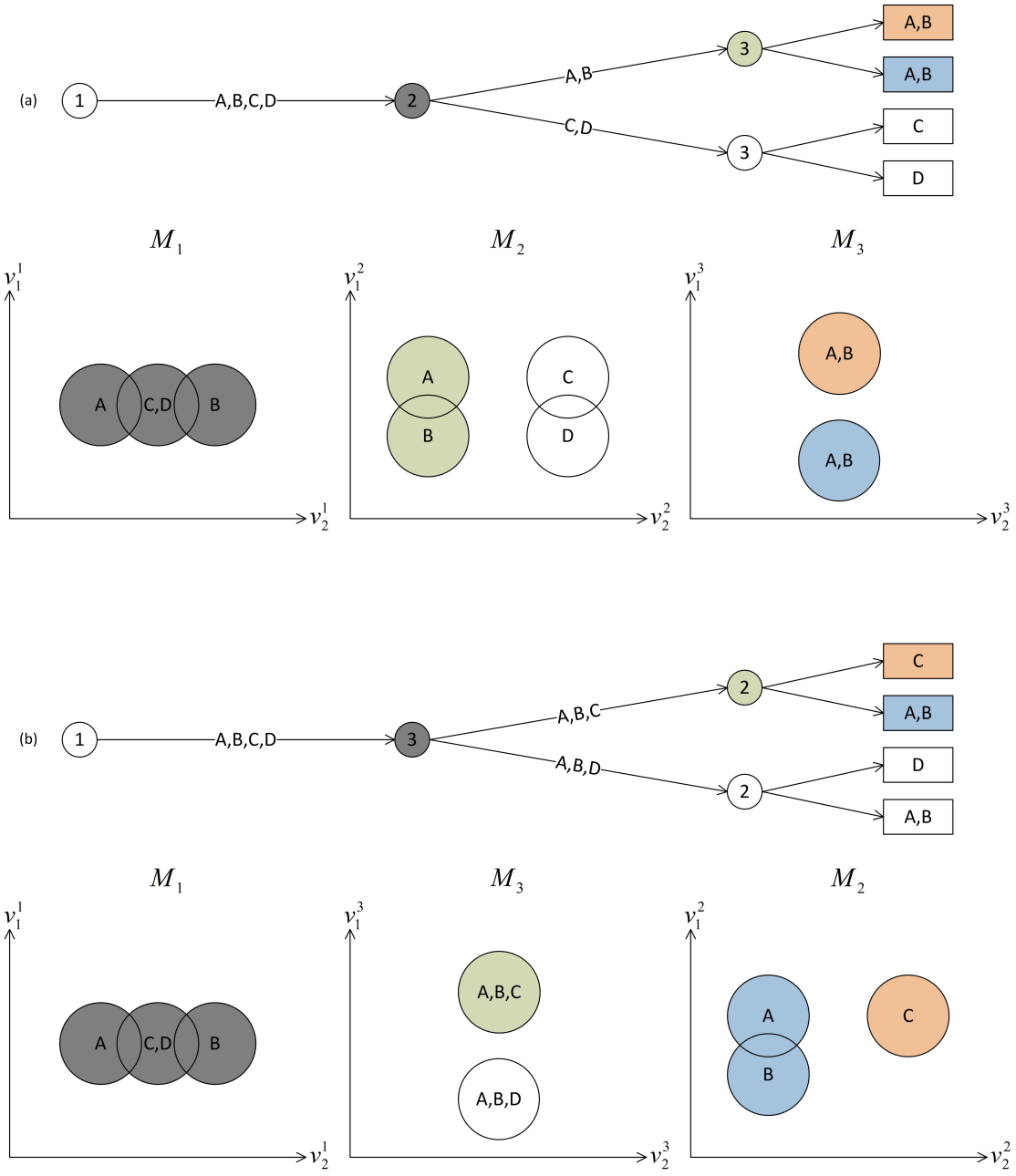


Abbildung 3.5: 2 verschiedene Bewertungsbäume mit Auswahl von Merkmalstyp M_1 an der Wurzel und Teilung der Objektklassen. Es ist keine vollständige Teilung der Objektklassen mehr möglich: (a) Wahl von M_2 als zweiten Merkmalstyp, (b) Wahl von M_3 als zweiten Merkmalstyp

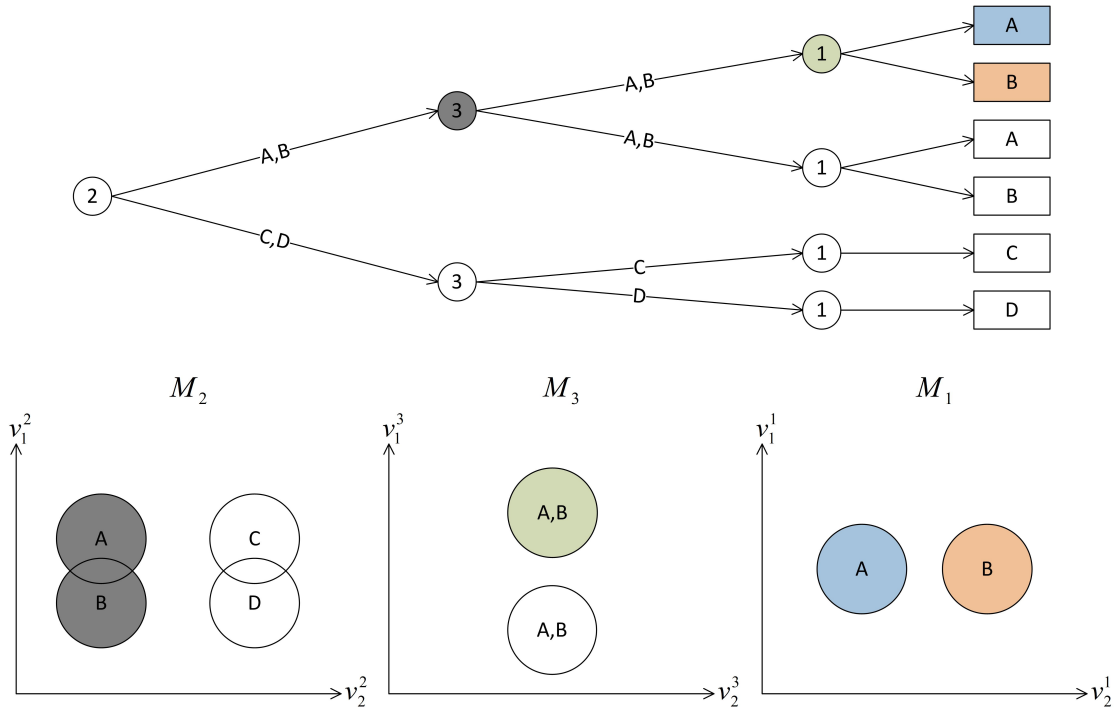


Abbildung 3.6: Bewertungsbaum mit vollständiger Klassenteilung bei günstiger Reihenfolge der Merkmalstypen

$$\forall_{m_i \in M} : \; F_i(c) = \left\{ f_1^i, \ldots, f_{k_{c,i}}^i \right\} \tag{3.10}$$

Alle Merkmalsvektoren f^i eines Merkmalstyps m_i befinden sich im gleichen Merkmalsraum und haben entsprechend die gleiche Länge $dim_i = \left| f^i \right|$. Es gilt Aussage 3.11.

$$\forall_{m_i \in M} : \; \exists_{dim_i} : \; \forall_c : \; dim_i = \left| f_1^i \right| = \ldots = \left| f_{k_{c,i}}^i \right| \tag{3.11}$$

Wie bereits erwähnt gibt es für jede Zusammenhangskomponente unterschiedlich viele Merkmalsvektoren abhängig vom Typ. Die Funktion 3.12 beschreibt diesen Zusammenhang und weist einer Zusammenhangskomponente c alle Merkmalsvektoren f^i von Typ m_i zu. Funktion 3.13 ist die Verallgemeinerung davon und beschreibt die Abbildung einer Zusammenhangskomponente c auf all ihre Merkmalsvektoren für jeden der t Merkmalstypen. Die Funktion $\mathfrak{P}(\bullet)$ bezeichnet die Potenzmenge und jeder Merkmalsvektor f^i ist ein Element des Merkmalsraumes $\mathbb{M}_i$, der hier festgelegt auf $\mathbb{R}^{dim_i}$ ist.

$$\begin{aligned} F_i : C &\rightarrow \mathfrak{P}\left(\mathbb{M}_i\right) \\ F_i(c) &= \left\{ f_1^i, \ldots, f_{k_{c,i}}^i \right\} \end{aligned} \tag{3.12}$$

$$\begin{aligned} F : C &\rightarrow \left\{ U \mid U \subseteq \mathfrak{P}\left(\mathbb{M}_i\right), \; |U| = t \right\} \\ F(c) &= \left\{ F_i(c) \mid i = 1 \ldots t \right\} \\ &= \left\{ \left\{ f_1^1, \ldots, f_{k_{c,1}}^1 \right\}, \ldots, \left\{ f_1^t, \ldots, f_{k_{c,t}}^t \right\} \right\} \end{aligned} \tag{3.13}$$

$$\mathbb{M}_i = \mathbb{R}^{dim_i}$$

Im letzten Schritt der Vorbereitung auf das Training wird die Menge der Bilder C in disjunkte Teilmengen T und V aufgeteilt (vgl. Menge 3.14). T dient als Trainingsmenge und V als Validierungsmenge für die Berechnung des Fehlers der erlernten Klassifikation an einem Knoten.

$$T \subset C, \; V \subset C, \; T \cap V = \emptyset \tag{3.14}$$

3.5 Training

Ausgangspunkt des Trainings ist die rekursive Funktion $createTree(N, M, T, V, F)$. Sie erzeugt einen Bewertungsbaum an Knoten N mit gegebener Merkmalsmenge M, Trainingsmenge T, Validierungsmenge V und Merkmalsvektorenmenge F. Algorithmus 3.1 erzeugt einen Wurzelknoten und ruft die Funktion $createTree()$ an diesem mit geeigneter Merkmals-, Trainings-, Validierungs- und Merkmalsvektorenmenge auf. Die Merkmalsvektorenmenge F wird initialisiert mit allen extrahierten Merkmalsvektoren jeden Typs von allen Trainings- und Validierungsbildern. Ist beim Aufruf der rekursiven Funktion $createTree()$ eine der Mengen M, T oder V leer, so ist der aktuelle Knoten N ein Blatt und $createTree()$ fertig (Rekursionsanker).

Für jeden Knoten N im Bewertungsbaum wird ein Merkmal m_i aus den zur Verfügung stehenden Merkmalen M ausgewählt (vgl. Abschnitt 3.3). Anschließend wird der Knoten mit diesem Merkmal markiert und auf allen Merkmalsvektoren in F mit Typ m_i eine agglomerative Clusteranalyse ausgeführt. Für jedes der r gefundenen Cluster C_d, werden auf den Merkmalsvektoren die Mittelwertvektoren $\mu(C_d)$ und Kovarianzmatrizen $\Sigma(C_d)$ zur Repräsentation der Cluster geschätzt. Danach wird für jeden Cluster und die Welt ein Kind am Knoten N erzeugt. Die Welt steht für alle Nicht-Zeichen. Sie bezeichnet hier im Kontext der Objekterkennung alles, was unbekannt ist. Sind alle Kinder erzeugt, werden die Kanten zu den Kindern beschriftet, Wahrscheinlichkeitsverteilungen berechnet und die rekursiven Aufrufe von $createTree()$ für jedes Kind außer der Welt vorbereitet und ausgeführt. In Abbildung 3.7 ist das Ergebnis für einen Knoten N schematisch dargestellt.

Algorithmus 3.2 zeigt die Kantenbeschriftung für jeden Kindknoten. Das Welt-Kind ist ein Blatt und erhält nur die Beschriftung, dass es sich um kein bekanntes Zeichen handelt. Alle anderen Kanten werden mit einer Liste beschriftet, die eine Menge von Klassenbezeichnungen, den Mittelwertvektor und die Kovarianzmatrix des Clusters, den das Kind repräsentiert, enthält. Die Menge von Klassenbezeichnungen enthält alle Klassen, für die es mindestens ein Trainingsbild gibt, das einen Merkmalsvektor im Cluster hat. Hier kann auch ein Schwellenwert eingestellt werden, so dass es größere Mindestanzahl von Trainingsbildern einer Klasse geben muss, bevor die Kante mit der Klasse beschriftet wird.

Algorithmus 3.3 führt die Berechnung von Wahrscheinlichkeitsverteilungen für jeden Kindknoten aus. Jeder Kindknoten speichert eine Wahrscheinlichkeitsverteilung. Das ist eine Funktion, die den Fehler der Klassifikationsleistung des Vaterknoten angibt. Anhand dieser Funktion wird später bei der Erkennung ermittelt mit welcher Wahrscheinlichkeit ein Pfad im Baum genommen wird und wie stark die Endaussage einer Entscheidung

ist. Weist der Klassifikator des Vaterknoten einen Merkmalsvektor f einem bestimmten Cluster und damit einem bestimmten Kindknoten zu, dann gibt die Wahrscheinlichkeitsverteilung dieses Kindknotens an, wie wahrscheinlich dieser oder ein anderer Kindknoten richtig ist. Dafür wird nach dem Training der Cluster auf der Validierungsmenge diese Wahrscheinlichkeitsverteilung ermittelt. Zu jedem Validierungsbild $v \in V$, mit $L(v) = l$ werden die Merkmalsvektoren f extrahiert und den Clustern zugewiesen. In der Wahrscheinlichkeitsverteilung p des Kindknoten k, dem ein f zugewiesen wird, werden die Werte $p(vater, k, j)$ um einen Zähler erhöht, wenn l an der Kante von Kindknoten j steht. Anschließend werden die Verteilungsfunktionen aller Kinder v von Vater u normiert, so dass $p(u, v, w)$ auf Wahrscheinlichkeitswerte zwischen 0 und 1 für jedes $w = 1 \ldots r$ abbildet und $\sum_{w=1\ldots r} p(u, v, w) = 1$, wenn r die Anzahl der Kinder von u ist. Abbildung 3.8 zeigt ein Beispiel für die Berechnung einer Wahrscheinlichkeitsverteilung vor der Normierung mit einem Validierungsbild v und zwei Merkmalsvektoren f_1^i und f_2^i, wobei f_1^i dem Weltknoten N_0 und f_2^i dem Knoten N_2 zugewiesen wird. Weil die Klasse von v an den Kanten der Knoten N_1 und N_2 steht, wird in der Wahrscheinlichkeitsverteilung für diese beiden Knoten der Wert erhöht.

Algorithmus 3.4 zeigt die Vorbereitung der rekursiven Aufrufe von *createTree*() an den Kindknoten durch Reduktion der Merkmals-, Beispiel- und Vektormengen. Die neuen Trainings- und Validierungsmengen und für einen Kindknoten enthalten nur die Bilder, deren Klasse an der Kante zu dem Knoten steht und die mindestens einen Merkmalsvektor haben, der in dem Cluster des Knoten liegt. So wird erreicht, dass in dem Teilbaum das Training auf die Zeichen spezialisiert wird, die eine bestimmte Eigenschaft haben. Anschließend wird aus der Menge der Merkmalsvektoren eines Bildes ein Vektor entfernt, der im Cluster des Knotens liegt. Das ist nötig, damit bereits verwendete Merkmalsvektoren nicht an einem weiteren Trainingsschritt teilnehmen. Sofern unter den Bildern der reduzierten Bildermengen, die für einen Knoten mitgenommen werden, kein Merkmalsvektor eines bestimmten Merkmalstyps zu finden ist, wird dieser Typ aus der Merkmalsmenge entfernt. Es ist für diesen Teilbaum nicht mehr notwendig das Merkmal an einem Knoten auszuwählen, denn es gibt keine Bilder mehr, die noch ein solches Merkmal enthalten. Nachdem die Mengen reduziert wurden, wird die Funktion *createTree*() für den entsprechenden Kindknoten aufgerufen, um dort das Training fortzusetzen.

Jeder Knoten hat neben dem Welt-Kind noch einen weiteren obligatorischen Kindknoten und zwar den Knoten der die Abwesenheit des Merkmals m_i repräsentiert und alle Trainings- und Validierungsbilder berücksichtigt unter denen kein einziger Merkmalsvektor f^i gefunden wurde. Dieser Knoten ist sehr wichtig, da er nicht nur die Unterscheidung der einzelnen Zeichen untereinander unterstützt, sondern auch die Abgrenzungsleistung

Algorithmus 3.1 Erstellung eines Bewertungsbaumes mit rekursiver Funktion *createTree*()

```
1  erzeuge Wurzelknoten R
2  createTree(R, M, T, V, F)
3
4
5  ------------------------------------------------------
6  // Definition rekursive Funktion
7  funktion:createTree(N, M, T, V, F)
8
9          // Rekursionsanker:
10         // wenn eine der benötigten Mengen erschöpft
11         M = ∅ ∨ T = ∅ ∨ V = ∅ ?
12                 N ist ein Blatt
13                 return
14
15         wähle ein m_i ∈ M aus
16         markiere Knoten N mit dem gewählten Merkmalstyp m_i
17
18         // Cluster werden auf allen Merkmalsvektoren
19         // des gewählten Merkmalstyps m_i bestimmt
20         FT = ⋃_{t∈T} F_i(t)
21         // agglomeratives Clustering → Anzahl r und Cluster C_d
22         bestimme die Cluster C_d mit d = 1...r in FT
23
24         // Repräsentation aller Cluster als
25         // multivariate Normalverteilung
26         für d = 1...r
27                 bestimme Mittelwertvektor μ(C_d)
28                 bestimme Kovarianzmatrix Σ(C_d)
29
30         // Anzahl an Kindknoten ist von Clusterzahl abhängig
31         für d = 0...r
32                 erzeuge Kindknoten N_d von Knoten N
33
34         // Kanten mit Clusterrepräsentation
35         // und Labels beschriften
36         makro:decorateEdges
37
38         // Wahrscheinlichkeitsverteilungen bestimmen
39         makro:calcDistributionFunctions
40
41         // Reduktion der Beispiele und rekursiver Aufruf
42         makro:processChildNodes
```

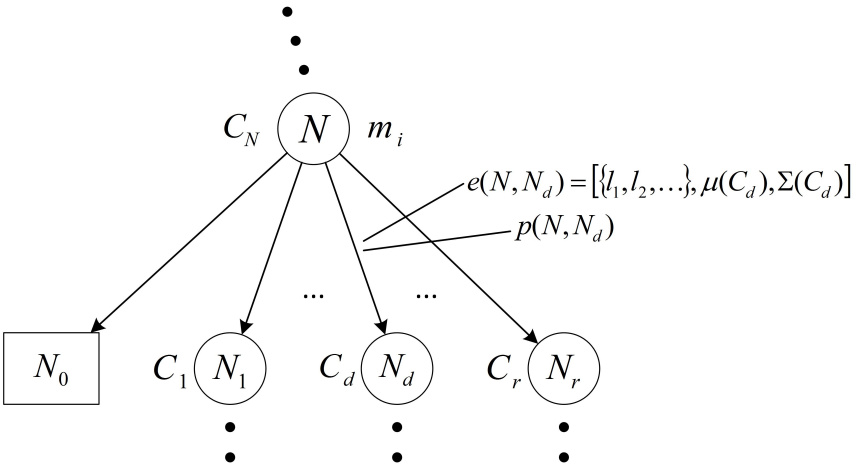


Abbildung 3.7: Ein Knoten des Bewertungsbaumes mit seinen Kindern und den gespeicherten Informationen.

Algorithmus 3.2 Kantenbeschriftung

```
1  // Dekoration der Kanten (Clusterrepräsentation und Klassen)
2  makro:decorateEdges
3          für d = 1...r
4                  // Mittelwert, Kovarianz und Klassen
5                  e(N, N_d) = [ {L(t ∈ T) | ∃_{f∈F_i(t)} : f ∈ C_d}, μ(C_d), Σ(C_d) ]
6
7          // Weltkante enthält keine Clusterrepräsentation
8          e(N, N_0) = ['no character']
```

Algorithmus 3.3 Berechnung der Wahrscheinlichkeitsverteilungen an den Kindknoten

```
1  // Wahrscheinlichkeitsverteilungen für Kindknoten N_0 ... N_r
2  makro: calcDistributionFunctions
3
4          // Menge aller Merkmalsvektoren f ∈ F_i(v ∈ V)
5          // des gewählten Merkmalstyps m_i
6          FV = ⋃_{v∈V} {(f,l) ∈ (F_i(v) × {1,...,K}) | L(v) = l}
7
8          // Summe aller Verteilungsvektoren (für Weltverteilung)
9          n_D = 0
10
11         // Wahrscheinlichkeitsverteilungen für Kinder
12         für d = 1...r
13                 // Merkmalsvektorenteilmenge, die in C_d liegt
14                 FV' = {(f,l) ∈ FV | f ∈ C_d}
15
16                 // Verteilungsmenge für j definiert
17                 D(j) ↦ {f ∈ M_i | (f,l) ∈ FV', l ∈ e(N, N_j)}
18                 n'_D = Σ_{j=1}^{r} D(j)
19                 n_D = n_D + n'_D
20
21                 // Ausnahme: keine Beispiele vorhanden (n'_D = 0)
22                 // beste Annahme: Gleichverteilung aller Pfade
23
24                 // Wahrscheinlichkeitsverteilung
25                 // für Beispiele mit C_d Merkmal
26                 p(N, N_d, 0) = 0
27                 p(N, N_d, j) = |D(j)| ÷ n'_D
28
29         // Wahrscheinlichkeitsverteilung für Weltknoten N_0
30         FV' = {(f,l) ∈ FV | ∀_{d∈{1,...,r}} : f ∉ C_d}
31         D(j) ↦ {f ∈ M_i | (f,l) ∈ FV', l ∈ e(N, N_j)}
32         n'_D = Σ_{j=1}^{r} D(j)
33         n_D = n_D + n'_D
34
35         // für Ausnahme n_D = 0, hier ebenfalls Gleichverteilung
36
37         // Wahrscheinlichkeitsverteilung für
38         //Beispiele mit Merkmal außerhalb aller C_d
39         p(N, N_0, 0) = 1 − (n'_D ÷ n_D)
40         p(N, N_0, j) = |D(j)| ÷ n_D
```

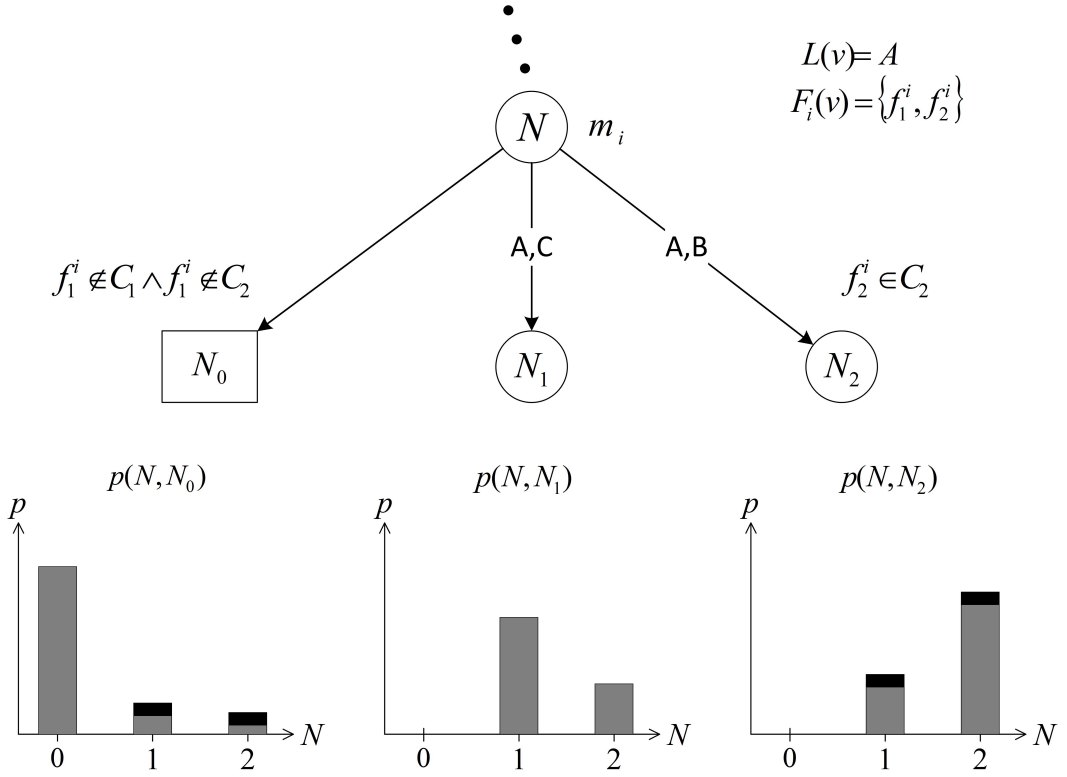


Abbildung 3.8: Beispiel eines Berechnungsschritts einer Wahrscheinlichkeitsverteilung. Validierungsbild v hat einen Merkmalsvektor außerhalb aller Cluster und einen in Cluster C_2. Wegen der Zugehörigkeit von v zu Klasse A werden die Wahrscheinlichkeitswerte von Knoten 1 und 2 in den Wahrscheinlichkeitsverteilungen von Knoten N_0 und N_2 erhöht.

Algorithmus 3.4 Reduktion der Trainingsmengen und rekursive Aufrufe

```
1  // Reduktion von Beispiel- und Merkmalsmengen
2  makro: processChildNodes
3          // Reduktion und rekursiver Aufruf für Kinder
4          für d = 1...r
5                  // neue Trainings- und Validierungsmenge
6                  T' = {t ∈ T | L(t) ∈ e(N, N_d) ∧ ∃_{f∈F_i(t)} : f ∈ C_d}
7                  V' = {v ∈ V | L(v) ∈ e(N, N_d) ∧ ∃_{f∈F_i(v)} : f ∈ C_d}
8
9                  // Herausnehmen eines Merkmals pro Beispiel
10                 für alle c ∈ (T ∪ V)
11                         wähle f ∈ {f ∈ F_i(c) | f ∈ C_d} zufällig
12                         F'_i(c) = F_i(c) \ {f}
13                         F'(c) = {F_j(c) ∈ F(c) | j ≠ i} ∪ {F'_i(c)}
14
15                 // Neubestimmung der verfügbaren Merkmale
16                 M' = {m_j ∈ M | ∀_{t∈T} F_j(t) ≠ ∅}
17
18                 createTree(N_d, M', T', V', F')
```

zur Welt verstärkt. Wegen der Übersichtlichkeit wurde auf die Darstellung verzichtet. Er wird bis auf wenige Einschränkungen und Bedingungen analog zu den normalen Kindknoten erstellt.

Der Bewertungsbaum ist ein spezieller Entscheidungsbaum. Jeder Pfad in dem Baum charakterisiert ein individuelles Zeichen. Werden von Zeichen einer Klasse Exemplare präsentiert, die zwei oder mehr deutlich verschiedene Ausprägungen zeigen (beispielsweise offene und geschlossene Nullen), dann entsteht im Bewertungsbaum für jede Ausprägung ein eigener Pfad. Um eine Überanpassung zu verhindern kann ein Schwellenwert für die Klassenbeschriftungen an einer Kante eingestellt werden. Nur wenn genug Trainingsbilder einen Merkmalsvektor in ein und demselben Cluster haben, dann wird die gesamte Klasse auch diesem Cluster zugeschrieben. Neben der manuellen Merkmalsauswahl am Knoten wird in den praktischen Versuchen dieser Arbeit mit dem Bewertungsbaum auch der Schwellenwert für die agglomerative Clusterbildung manuell ausgewählt. Je nach Merkmal und Bildermenge ein einem Knoten können die Cluster sehr unterschiedliche Abstände haben. Ein allgemein gültiger Schwellenwert θ kann daher nicht vorgegeben

werden. Er ist immer abhängig von der aktuellen Situation. Eine Möglichkeit den Schwellenwert automatisch zu bestimmen ist die Cluster so lange zusammenwachsen zu lassen, bis alle Clusterellipsoiden eine Mindestgröße haben, sich aber nicht schneiden. Die triviale Lösung jeder Cluster besteht aus einem Datenpunkt wäre sonst immer gegeben.

Kapitel 4

Zeichenerkennung

Nachdem im vorigen Kapitel der Bewertungsbaum erzeugt wurde, kann er jetzt genutzt werden, um für gegebene unbekannte Bilder zu entscheiden ob ein Zeichen zu sehen ist und wenn möglich, um welches Zeichen es sich dabei handelt. Wird eine einfache Tiefensuche auf dem Bewertungsbaum mit den Merkmalsvektoren eines Abfragebildes durchgeführt, dann muss das Ergebnis nicht richtig sein. Es ist möglich, dass durch einen einzigen Vektor, der außerhalb eines Clusters lag, ein Weg eingeschlagen wird, der zu einem falschen Ergebnis führt. Das Problem kann nur vollständig gelöst werden, indem für jeden Pfad die Wahrscheinlichkeit bestimmt wird, mit der das Abfragebild zu diesem Pfad gehört. Eine solche Suche ist wenig effizient. Deswegen wird der Baum mit Hilfe einer Monte Carlo Traversierung[1] durchsucht. Algorithmus 4.1 beschreibt das Verfahren.

Vor dem Start wird der Iterationsparameter n vorgegeben. Er bestimmt, wie oft der Baum von der Wurzel bis zu einem Blatt durchlaufen wird. Bei jedem Durchlauf wird die Länge l und Wertung v des Pfades bestimmt. An jedem Knoten N einer Iteration wird das Merkmal m_i bestimmt und die Zugehörigkeit der Merkmalsvektoren f^i von c zu den Kindknoten berechnet. Unter diesen wird zufällig ein Kindknoten N_d ausgewählt. Anhand der Wahrscheinlichkeitsverteilung $p(N, N_d)$ wird nun zufällig der Kindknoten N_e bestimmt und an diesem die Tiefensuche tatsächlich fortgesetzt. Aus der Menge der Merkmalsvektoren von c wird ein Vektor f^iheraus genommen, der im Cluster C_e liegt, falls vorhanden. So wird sichergestellt, dass die Merkmalsvektorenmenge von c so aussieht, als ob N_e wirklich ausgewählt worden wäre. Der aktuelle Pfadabschnitt geht in die Gesamtwertung mit der Wahrscheinlichkeit $p(N, N_d, e)$ ein. Das bedeutet, wird der Weg über N_e gegangen, obwohl N_d der treffende Cluster war, dann ist die Richtigkeit dieses alternativen Wegstückes so groß, wie beim Training die Validierungsbilder auch falsch in N_d einsortiert wurden, obwohl sie zu N_e gehört haben. Dieses Verfahren folgt der Idee, dass Fehler die beim Training gesehen wurden, jetzt vielleicht auch gerade gemacht

[1] Die Monte Carlo Methode bezeichnet eine Klasse von randomisierten Algorithmen, deren Ergebnis falsch sein darf, aber durch wiederholtes Ausführen immer weiter verbessert werden kann [31, 32, 33].

werden und so versucht wird diese zu korrigieren. War es einer dieser Trainingsfehler, dann wird der Restpfad ein sehr hohes Gewicht haben. War es aber keiner dieser Fehler, dann wird der Restpfad mit hoher Wahrscheinlichkeit zu keinem Erfolg führen. Am Ende eines Durchlaufs, wenn der Algorithmus an einem Blatt angekommen ist, werden alle Kantengewichte über l normiert, so dass kürzere Pfade keine schwächere Aussage als längere haben. Der Klassenname an der letzten Kante gibt das erkannte Zeichen an. Stehen mehr als ein Klassenname an der Kante, dann kann nur gesagt werden, dass es sich um ein Zeichen handelt, aber nicht um welches. Es werden n Durchläufe gemacht und der Durchlauf mit dem maximalem Pfadgewicht für die Entscheidung ausgewählt.

Wird der Fokus bei der Nutzung des Bewertungsbaumes auf die binäre Entscheidung gelegt, ob es sich um ein Zeichen handelt oder nicht, anstatt der Frage nachzugehen um welches Zeichen es sich handelt, dann können nach dem Training alle Wahrscheinlichkeitswerte des Welt-Knotens in den Wahrscheinlichkeitsverteilungen des Welt-Knotens abgesenkt werden. Dadurch wird die Strategie „im Zweifel für den Angeklagten“ verfolgt. Die Wahrscheinlichkeit, dass es sich um kein Zeichen handelt, wenn für ein Abfragebild ein Welt-Blatt gefunden wird, wird abgesenkt und verstärkt nach einer anderen passenden Möglichkeit gesucht. Wird der Wert zu stark abgesenkt, dann wird der Baum zu tolerant und sucht nur das „kleinste Übel“. Das ist das Zeichen was noch am besten passt.

Algorithmus 4.1 Monte Carlo Traversierung des Bewertungsbaumes zur Zeichenerkennung

```
1  // R - Wurzel eines trainierten Baums
2  // c - Zusammenhangskomponente zur Abfrage
3
4  for j = 1...n
5          // Merkmalsmenge durch Merkmalsextraktion bestimmen
6          bestimme F(c)
7
8          v = 0
9          l = 0
10
11         // gehe von der Wurzel bis zu einem Blatt
12         N = R
13
14         solange N ≠ leaf
15                 bestimme Merkmal m_i von Knoten N
16
17                 F_opt = {f ∈ F_i(c) | f ∈ C_d}
18                 F_opt ≠ ∅ ?
19                         wähle aus F_opt zufällig ein f
20                         bestimme d für das gilt f ∈ C_d
21                 sonst
22                         d = 0
23
24                 wähle anhand von p(N, N_d)
25                         zufällig einen Kindknoten N_e
26
27                 e ≠ 0 ?
28                         F_pot = {f ∈ F_i(c) | f ∈ C_e}
29
30                         // entferne ein f ∈ F_pot aus F_i(c)
31                         F_pot ≠ ∅ ?
32                                 wähle aus F_pot zufällig ein f
33
34                                 F_i(c) = F_i(c) \ {f}
35
36                 v = v + p(N, N_d, e)
37                 l = l + 1
38                 N = N_e
39
40         // wenn die letzte Kante mehr als eine Klasse enthält,
41         // ist Ergebnis zwar nicht eindeutig, aber kein Zeichen
42         LC_j = {l ∈ e(N_p, N)}
43
44         // Wertung für Durchlauf j
45         v_j = v ÷ l
46
47 v_j = max(v_1, ..., v_n)
48 LC = LC_j
```

Kapitel 5

Experimentelle Resultate

Im Rahmen dieser Masterarbeit wurde zum Testen und Überprüfen der entwickelten Algorithmen ein Framework in der Programmiersprache $C\#$[1] unter Zuhilfenahme der *OpenCV*[2] Bibliothek entwickelt. Das Programm implementiert ein Prototyping Konzept. Es können für die verschiedenen Vorverarbeitungs- und Extraktionsschritte Module bereitgestellt und visuell zu einer Kette verknüpft werden. Das Benutzerinterface ermöglicht beliebige Parameter der Module zu konfigurieren, sowie die einzelnen Ergebnisse einer Ausführung am betreffenden Bild direkt darzustellen. Beispielbilder in dieser Arbeit wurden auf diese Weise erzeugt.

Wegen der noch fehlenden automatisierten Möglichkeit die *informativsten*, *unabhängigen* Merkmale aus der Merkmalsmenge auszuwählen konnte bisher kein vollständiger Baum für den kompletten Zeichenvorrat lateinischer Buchstaben und arabischer Ziffern erlernt werden. Die zur Verfügung stehende Trainingsmenge ist außerdem zu klein, um für die verwendete Merkmalsmenge hinreichend viele Trainingsbeispiele bereitzustellen. Das stellt aber insoweit kein Problem dar, da ein kleiner Baum für wenige Zeichenklassen manuell erstellt werden kann. Es gilt außerdem, dass eine gut funktionierende Merkmalsmenge für eine Auswahl der Zeichenklassen nicht alle hier angegebenen Merkmale enthalten muss. Schriftzeichen mit runden Strukturen (*0*, *8*, *Q*, *O*) müssen nicht mit Linien- oder Dreieckmerkmalen beschrieben werden, um schon einen hohen Erkennungsgrad zu erreichen. Insbesondere ist es sogar möglich, für jedes Zeichen einzeln einen eigenen Bewertungsbaum zu trainieren. Das ist eine sehr interessante Möglichkeit, wenn man betrachtet, das sich dadurch sehr kleine und effiziente Bäume erstellen lassen, die nur die notwendigsten Merkmale für das jeweilige Zeichen berücksichtigen. Beim Erkennen von Zeichen wird dann jeder Baum gefragt und die Antwort mit der größten Bewertung

[1] Objektorientierte Programmiersprache der Firma Microsoft. (http://msdn.microsoft.com/en-us/library/kx37x362.aspx)

[2] Ursprünglich von Intel entwickelte Bibliothek mit Algorithmen für maschinelles Sehen. (http://opencv.willowgarage.com/)

unter allen Bäumen ausgewählt. Selbst bei großen Mengen von Zeichenklassen kann diese Möglichkeit leistungsfähig sein. Es hat sich bei den Versuchen gezeigt, dass mit sehr kleinen Bäumen schon Erkennungsraten von bis zu 95% erreicht werden. Im Hinblick auf die binäre Klassifikation von Zeichen und Nicht-Zeichen lag die Ablehnung von Zeichen als Welt (false-negative) deutlich darunter. Der Bewertungsbaum kann noch sehr viel toleranter eingestellt werden, wenn die Wahrscheinlichkeitsverteilungen der Welt-Kinder im Baum zugunsten der anderen Kinder abgesenkt werden.

Ein deutliches Problem hat sich bei der Unterscheidung von sehr ähnlichen Zeichen, wie dem *O* und der *0* gezeigt. Die vorgestellten Merkmalsextraktoren sind für diese Unterscheidung schlecht geeignet. Denkbar wäre ein Merkmal *Kreisähnlichkeit* oder *Ellipse* einzuführen, um die Unterschiede dieser Zeichen zu charakterisieren. Für die binäre Entscheidung nach einem Zeichen hat sich das Problem nicht ausgewirkt, da beide Zeichen nahezu immer auch als Zeichen erkannt wurden.

Benötigt man die Abgrenzung zur Welt nicht und erwartet nur Zeichen aus den vorgegebenen Zeichenklassen, dann kann die Merkmalsmenge reduziert werden, so dass sie für eine Separation der Zeichen ausreicht. Außerdem kann man das Training beenden, wenn an einem Knoten nur noch Beispielbilder einer einzigen Klasse zur Verfügung stehen. Bei der Auswertung des Ergebnisses wird dann die Antwort mit der höchsten Bewertung ausgewählt, die eine Zeichenklasse beinhaltet. Sie gibt ein Wahrscheinlichkeitsmaß für das gefundene Zeichen an. Es gibt bei dieser Art der Nutzung des Bewertungsbaumes aufgrund seiner Konstruktion häufig auch noch eine Klassifikation als Nicht-Zeichen, die ausgeschlossen werden muss.

Bei den Versuchen für die binäre Entscheidung die false-positive Rate zu senken, ohne die false-negative Rate zu erhöhen, hat sich eine Erkenntnis gezeigt. In jedem Teilbaum während des Baumtrainings, der nur noch eine Klasse enthält, stellt sich die Frage nach der Weiterführung des Trainings in dem Teilbaum. Es reicht aufzuhören, wenn nur Zeichenseparation gewünscht ist (siehe oben), aber es muss fortgefahren werden, wenn die false-positive Rate gesenkt werden soll. Fügt man von hier an immer weitere neue Merkmale ein, so kann die Trennung zur Welt nur noch weiter verbessert werden. Um die Qualität dieser Trennung zu bewerten sind ein Maß und Negativbeispiele nötig. Es ist nicht möglich ohne Negativbeispiele aus der Anwendungsdomäne eine gültige Aussage zu treffen.

Der Bewertungsbaum lernt für die gegebene Merkmalsmenge eine Beschreibung jeder Zeichenklasse. Solch eine Beschreibung stellt sich im Baum allerdings nicht zwingend durch einen einzigen Pfad dar. Man kann das so Interpretieren, dass ein Pfad eine *UND*-Verknüpfung aller Merkmale an den Knoten ist und alle Pfade einer Zeichenklasse

ODER-verknüpft sind. Betrachtet man beispielsweise nur das Streckenmerkmal bei dem Buchstaben *H*, dann ergeben sich 3 verschiedene Strecken. Eine waagerechte in der Mitte des Bildes und jeweils eine senkrechte rechts und links. Es ist unmöglich eine Beschreibung nur durch UND-Verknüpfungen dieses Buchstabens im Baum zu geben. Während des Trainings prägen sich 2 Pfade aus, die jeweils am Knoten der Trennung die linke oder rechte Strecke beschreiben (*ODER*). In jedem dieser beiden Pfade findet sich tiefer die jeweils andere Strecke. Dadurch entscheidet sich beim Abfragen mit einem *H* an dem Trennungsknoten das wählende Kind durch die zufällige Wahl des linken oder rechten Streckenmerkmals. Die zweite Strecke wird dann entsprechend auf dem verbleibenden Pfad zum Blatt noch abgefragt.

Das Testprogramm hat eine Eingabemöglichkeit für handgeschriebene Einzelzeichen, um die Qualität des Algorithmus auch hierauf zu testen. Dabei hat sich ergeben, dass das Merkmal Symmetrie eine Erkennung deutlich erschwert, weil handgeschriebene Zeichen selten symmetrisch sind. Zumindest nicht den hohen Grad der Symmetrie aufweisen, wie bei den Druckzeichen während des Trainings gelernt wurde. Durch Weglassen dieses Merkmals im Bewertungsbaum wurde die Erkennung robuster für Handschrift, sofern die Zeichen in Druckschrift geschrieben werden. Die Erkennungsraten waren ähnlich hoch, wie bei den Druckzeichen.

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit ist von dem Prinzip ausgegangen, nach welchem ein Objekt die Summe seiner Eigenschaften ist. Anstelle eines klassischen Ansatzes in der Mustererkennung, wurde auf diesem Prinzip aufbauend versucht die Beschreibung von Objekten zu lernen und sie anhand dieser wiederzuerkennen. Dafür wurde das Problem der Schrifterkennung als Beispiel herangezogen. Dafür muss zuerst die Frage beantwortet werden, welche Eigenschaften für diese Problemklasse geeignet sind. Aus der Betrachtung der Schrift haben sich geometrische Merkmale als naheliegend dargestellt und es wurde eine Auswahl für die Evaluation der aufgestellten Hypothese getroffen. Zur Extraktion dieser Merkmale wurden geeignete Vorverarbeitungsschritte festgelegt und passende Algorithmen entwickelt. Zur Beschreibung jeder Objektklasse mit Hilfe dieser vorgegebenen Merkmale wurde anschließend ein modifizierter Entscheidungsbaum konstruiert. Mit diesem sogenannten Bewertungsbaum wurde abschließend die Grundaussage dieser Masterarbeit überprüft.

Der vorgestellte Bewertungsbaum benötigt keine Negativbeispiele zum Trainieren. Er lernt jede Objektklasse in Form der Kombinationen ihrer zugrundeliegenden Eigenschaften auswendig. Die verwendete Vorverarbeitung und Merkmalsmenge ist maßgeblich für die Qualität der Leistung, durch Abwägung zwischen Genauigkeit und Größe des Baumes. Am Beispiel von Schrift konnte die Theorie der Objekte als Summe aus ihren Eigenschaften nicht widerlegt werden und nährt die Hoffnung, dass es stimmt.

6.1 Ausblick

Trotz der guten Ergebnisse mit dem vorgestellten Bewertungsbaum bleiben viele Fragen offen. Eine der schwierigsten Fragen ist die nach einer minimalen vollständigen Merkmalsmenge. Es ist zwar zu sehen, dass die Menge zu klein ist, wenn es noch mindestens ein Blatt im Baum gibt, das mehr als eine Klassenbeschriftung hat, aber dieses Kriterium gibt keinen Aufschluss über die Qualität der Merkmalsmenge bezüglich einer Abgrenzung zur Welt. Ein mathematisches Maß, mit dem eine Überdeckung der Objekte durch

die Merkmalsmenge berechnet werden kann, ist wünschenswert. Ebenso könnte eine gute Merkmalsmenge aus Merkmalen bestehen, die nicht direkt an die Wahrnehmung des Menschen angelehnt sind, wie die geometrischen Strukturen in Schriftzeichen.

Bei der Wahl verschiedener Parameter gibt es viele Möglichkeiten der Automation. Neben der Merkmalsauswahl am Knoten und der Schwellenwertbestimmung für die agglomerative Clusterbildung, können viele Parameter der Extraktoren maschinell ermittelt werden. Die Schwellenwerte der RANSAC Algorithmen sind ein Beispiel hierfür. Durch die Probleme ähnlicher Zeichen hat sich gezeigt, dass es außerdem sinnvoll ist die Merkmalsmenge um einige zusätzliche Merkmale wie *Ellipse* oder *Punktsymmetrie* zu erweitern. Obwohl der Bewertungsbaum als Konstrukt für die Speicherung einer ganzen Menge von Objektklassen entwickelt wurde, ist ein vielversprechender Ansatz die Objektklassen in mehrere kleine Gruppen einzuteilen (bis hin zu Einzelklassen) und auf diesen spezialisiertere Bäume trainieren zu lassen.

Alles in allem ist der Bewertungsbaum eine interessante neue Möglichkeit im Feld der Entscheidungsbaumalgorithmen, der noch viel Potential zur Verbesserung bietet.

Literaturverzeichnis

[1] R. Teicher and M. Griesbeck, *Pimp your Brain.* Campus-Verl., 2008.

[2] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vision*, vol. 57, pp. 137–154, May 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=966432.966458

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, November 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=993451.996342

[4] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, ser. CVPR '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2161–2168. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2006.264

[5] Z. Wu, Q. Ke, J. Sun, and H.-Y. Shum, "A multi-sample, multi-tree approach to bag-of-words image representation for image retrieval," in *ICCV*, 2009, pp. 1992–1999.

[6] T. Yeh, J. J. Lee, and T. Darrell, "Adaptive vocabulary forests br dynamic indexing and category learning," in *ICCV*, 2007, pp. 1–8.

[7] D. He, S. Liang, and Y. Fang, "A multi-descriptor, multi-nearest neighbor approach for image classification," in *Advanced Intelligent Computing Theories and Applications*, ser. Lecture Notes in Computer Science, D.-S. Huang, Z. Zhao, V. Bevilacqua, and J. Figueroa, Eds. Springer Berlin / Heidelberg, 2010, vol. 6215, pp. 515–523. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14922-1_64

[8] Q. Zhang and E. Izquierdo, "A multi-feature optimization approach to object-based image classification," in *Image and Video Retrieval*, ser. Lecture Notes in Computer Science, H. Sundaram, M. Naphade, J. Smith, and Y. Rui, Eds.

Springer Berlin / Heidelberg, 2006, vol. 4071, pp. 310–319. [Online]. Available: http://dx.doi.org/10.1007/11788034_32

[9] F. Bajramovic, F. Mattern, N. Butko, and J. Denzler, "A comparison of nearest neighbor search algorithms for generic object recognition," in *Advanced Concepts for Intelligent Vision Systems*, ser. Lecture Notes in Computer Science, J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, Eds. Springer Berlin / Heidelberg, 2006, vol. 4179, pp. 1186–1197. [Online]. Available: http://dx.doi.org/10.1007/11864349_108

[10] O. Boiman, E. Shechtman, and M. Irani, "In defense of nearest-neighbor based image classification," *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, vol. 2, no. i, pp. 1–8, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587598

[11] T. Hong, S. W. Lam, J. J. Hull, and S. N. Srihari, "The design of a nearest-neighbor classifier and its use for japanese character recognition," in *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ser. ICDAR '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 270–. [Online]. Available: http://portal.acm.org/citation.cfm?id=844379.844680

[12] F. Lebourgeois and J. Henry, "An evolutive ocr system based on continuous learning," *Applications of Computer Vision, IEEE Workshop on*, vol. 0, p. 272, 1996.

[13] M. Block and R. Rojas, "Local contrast segmentation to binarize images," in *Proceedings of the 2009 Third International Conference on Digital Society*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 294–299. [Online]. Available: http://portal.acm.org/citation.cfm?id=1510529.1511505

[14] Tex von tex-block.de, "Buchstaben und ihre typischen merkmale," Internet, 2010. [Online]. Available: http://tex-block.de/typografie/buchstaben-und-ihre-typischen-merkmale/842/

[15] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Cambridge, MA: O'Reilly, 2008.

[16] B. Naegel, N. Passat, and C. Ronse, "Grey-level hit-or-miss transforms-part i: Unified theory," *Pattern Recogn.*, vol. 40, pp. 635–647, February 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1220966.1221222

[17] H. Blum, "A Transformation for Extracting New Descriptors of Shape," in *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed. Cambridge: MIT Press, 1967, pp. 362–380.

[18] A. Manzanera, T. M. Bernard, F. J. Prêteux, and B. Longuet, "Ultra-fast skeleton based on an isotropic fully parallel algorithm," in *Proceedings of the 8th International Conference on Discrete Geometry for Computer Imagery*, ser. DCGI '99. London, UK: Springer-Verlag, 1999, pp. 313–324. [Online]. Available: http://portal.acm.org/citation.cfm?id=648318.761632

[19] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Commun. ACM*, vol. 27, pp. 236–239, March 1984. [Online]. Available: http://doi.acm.org/10.1145/357994.358023

[20] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, pp. 11–15, January 1972. [Online]. Available: http://doi.acm.org/10.1145/361237.361242

[21] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[22] P. J. Rousseeuw, "Least median of squares regression," *Journal of the American Statistical Association*, vol. 79, no. 388, pp. 871–880, 1984. [Online]. Available: http://www.jstor.org/stable/2288718

[23] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0734189X85900167

[24] S. Madhvanath, G. Kim, and V. Govindaraju, "Chaincode contour processing for handwritten word recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, pp. 928–932, September 1999. [Online]. Available: http://dx.doi.org/10.1109/34.790433

[25] D. Dori and W. Liu, "Sparse pixel vectorization: An algorithm and its performance evaluation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, pp. 202–215, March 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=299447.299452

[26] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, 2008, 10.1007/s10115-007-0114-2. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0114-2

[27] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, March 1986. [Online]. Available: http://portal.acm.org/citation.cfm?id=637962.637969

[28] S. Y. Jung and T.-S. Kim, "An agglomerative hierarchical clustering using partial maximum array and incremental similarity computation method," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ser. ICDM '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 265–272. [Online]. Available: http://portal.acm.org/citation.cfm?id=645496.658023

[29] B. Walter, K. Bala, M. Kulkarni, and K. Pingali, "Fast agglomerative clustering for rendering," *2008 IEEE Symposium on Interactive Ray Tracing*, pp. 81–86, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4634626

[30] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 881–892, July 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=628329.628801

[31] R. M. Neal, "Probabilistic inference using markov chain monte carlo methods," *Intelligence*, vol. 62, no. September, p. 144, 1993.

[32] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*. Wiley-VCH, 2008.

[33] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan, "An Introduction to MCMC for Machine Learning," *Machine Learning*, vol. 50, no. 1, pp. 5–43, Jan. 2003.