Nachkorrektur von Ergebnissen einer optischen Charaktererkennung

Klaus U. Schulz

10. Oktober 2006

Inhaltsverzeichnis

Kapitel 1

Einführung und Motivation

1.1 Motivation

Vorurteil 1: *Papierdokumente werden bedeutungslos wegen elektronischer Dokumente, Web-Technologie. Stichwort vom papierlosen Büro...*

Tatsächliches Bild: Eine Fülle von Büchern, Dokumenten und Textunterlagen bei Verlagen, Bibliotheken, Behörden und Unternehmen liegt in Papierform vor. Ursachen:

- Historische Gründe.
- Weder Hard- noch Software ist voll standardisiert. Elektronik kann ausfallen. Verfasse sensible Dokumente als Papierdokumente. Selbst wenn diese elektronische erstellt sind, ist elektronische Ursprungsdatei oft nicht verfügbar.
- ...

Vorurteil 2: *OCR gelöste Technologie. Jeder kann sich Scanner und Software kaufen...*

Tatsächliches Bild: Von vornherein zwischen OCR und Dokumentenanalyse/Erfassung mit Hilfe von OCR zu unterscheiden. Grenzen fließend. Ermittle in korrekter Weise

1. alle Symbole und Wörter des Dokuments,
2. wie (i), zusätzlich die intendierte (oder eine sinnvolle) Reihenfolge der Wörter,
3. wie (ii), zusätzlich die logische Gliederung des Dokuments,
4. wie (iii), zusätzlich soviel wie möglich inhaltliche Aspekte.

In allen vier Teilbereichen gibt es viele spezifische ungelöste Probleme.

1. Fehleranfälligkeit der Buchstabenerkennung vor allem bei Handschriften, Kursivschriften, kleinen Fontgroßen,... Erfassung von Nichtstandardsymbolen (math. Notation, Asiatische Schriftsymbole, Sonderzeichen)

2. Unzulänglichkeiten bei der Segmentierung bei kompliziertem Layout, bei gemischten Bild- und Textdokumenten.

3. Extreme Vielzahl von Dokumententypen. Selbst bei feststehendem und einfachem Dokumententyp (Karteikarte, Scheck) Erfassung der logischen Struktur schwierig, Gegenstand aktueller Forschung.

4. Was ist Inhalt eines Dokuments? Wie kann man ihn automatisch ermitteln, optimal repräsentieren, abfragen? Frage ungelöst. Parallel bei IR- und Webtechnologie untersucht.

Die genannte Schwachstellen führen dazu, daß in vielen Bereichen, in denen Systeme der OCR und Dokumentenanalyse im Prinzip sinnvoll eingesetzt werden könnten, noch andere Techniken eingesetzt werden, die mit sehr viel mehr manuellen Tätigkeiten verbunden sind.

- Bei der OCR-Erfassung von Dokumenten in Verlagen, Bibliotheken und Behörden bleibt oft eine mühsame manuelle oder interaktive Nachbearbeitung notwendig. Der sich letztendlich ergebende Zeit- und Personalaufwand führt oft zu einer Situation, wo aus finanziellen Gründen andere Methoden der digitalen Erfassung gewählt werden, obwohl die OCR-Konversion langfristig deutlich sinnvoller wäre.

- Systeme zur vollautomatische Analyse für Eingangskorrespondenz, Schecks, Steuerformular etc. werden erst in einigen wenigen Bereichen eingesetzt. Die Weiterentwicklung ist ein intensiv diskutiertes Forschungsthema.

1.2 OCR-Forschung: Informationsquellen

Im Anhang ist eine Liste der wichtigsten Zeitschriften, Tagungen, Forschungsgruppen und Webseiten zur Forschung im Bereich OCR beigefügt.

1.3 Benachbarte Disziplinen

Computerlinguistik, elektronische Lexikographie
Datenbanken, Workflow
Angewandte Stochastik (Klassifikationsverfahren, Neuronale Netze, Bilderkennung)
Algorithmen auf Strings (Bioinformatik)
Dokumentenmodellierung, XML, SGML
Information Retrieval

Kapitel 2

Optische Charaktererkennung und Dokumentenanalyse

Die Diskussion im vorausgegangenen Kapitel hat bereits klargestellt, daß es nicht einfach ist, die optische Charaktererkennung (OCR) als einen abgeschlossenen, klar abgegrenzten Vorgang zu definieren. In diesem Kapitel soll das Bild präzisiert werden, was denn überhaupt unter optischer Charaktererkennung und Dokumentenanalyse zu verstehen ist. Die nachfolgenden Teilkapitel geben eine partielle Antwort, wobei jeweils unterschiedliche Perspektiven eingenommen werden. Nach einer allgemeinen und vorläufigen Antwort im ersten Teil des Kapitels gehen wir kurz auf die Geschichte der OCR ein. Nachfolgend beschreiben wir die typischen Arbeitsschritte bei der OCR und skizzieren die dort verwendeten Hintergrundverfahren. Im Anschluß werden die heutigen Grenzen der so umrissenen Technologie umrissen. Das Kapitel schließt mit einer Auflistung von Anwendungsgebieten der OCR und Dokumentenanalyse.

2.1 Aufgabe und Ziel der optischen Charaktererkennung

Um den Begriff der optischen Charaktererkennung zu präzisieren, kann man zunächst die wissenschaftliche Fragestellung beschreiben, die den Kern der OCR-Forschung ausmacht. Hierbei ist eine engere oder eine weitere Perspektive möglich.

- **Eingeschränkte technische Antwort**: Mit dem Begriff OCR wird der Prozeß beschrieben, wo das digitalisierte Bild eines Textes in einem *off-line Prozeß* in eine *symbolische Repräsentation* übergeführt wird, in welcher Buchstaben und Wörter explizit repräsentiert sind. Der Text kann handgeschrieben, gedruckt oder in gemischter Form vorliegen. Hierbei werden *geeignete Erkennungsverfahren* eingesetzt, um Symbole oder Wörter zu ermitteln. Allgemeinere Formen setzen sich auch mit dem Problem der Erkennung von speziellen Symbolen (mathematische oder chemische Zeichen), Zeichnungen, Karten, u. ä. auseinander. Im Kern steht stets die Überführung von Bildinformation in symbolische Information.

- **Weitergehende technische Antwort**: Mit dem Begriff OCR wird — unscharf — ein allgemeinerer Prozeß beschrieben, dessen Ziel nicht nur die korrekte Erfassung von Buchstaben und Wörtern ist, sondern wo eine möglichst vollständige symbolische Repräsentation des Dokuments einschließlich seiner (Layoutform) logischen Gliederung und inhaltlichen Bedeutung erreicht werden soll. Man spricht dann besser von Dokumentenanalyse mit Hilfe von OCR.

Eine andere Perspektive ergibt sich, wenn man den *kontextuellen Rahmen* der typischen *Anwendungen der OCR* ins Auge fasst. Hier stellt sich OCR dar als eine von mehreren Möglichkeiten zur elektronischen Erfassung von Daten oder Textdokumenten. Anderere Formen der Datenerfassung, teils thematisch verwandt, sind

- die Spracherkennung,
- die online-Schrifterkennung (elektronischer Terminkalender),
- die Bilderfassung (Parkhäuser),
- Magnetstreifen (Scheckkarten etc.),
- Bar code (Supermarkt Produkte).

Alternativen zur OCR bei der Erfassung von Textdokumenten in Papierform sind

- die manuelle Neuerfassung,
- die Erfassung mittels digitaler Bilddateien und ggf. in Verbindung mit einfachen Formen der manuellen Verschlagwortung.

Der Vorteil der OCR-Erfassung im Vergleich zu den oben genannten Alternativen sollte offenkundig sein: da bei der OCR-Konversion — anders als bei der Erfassung mittels digitalisierter Bilddateien — eine vollständige symbolische Erfassung von Text und Struktur erreicht wird, bietet diese eine optimale Ausgangsbasis für nachfolgende Formen der Weiterverarbeitung und Modifikation. Dies wird verdeutlicht durch einige der später dargestellten Anwendungen.

2.2 Geschichte der OCR

Der nachfolgende kurze Abriß der Geschichte der OCR folgt L. Eikvil [Eik93].

1870
C.R. Carey (Boston Massachusetts) erfindet den Retina-Scanner. Bildtransmissions-System, das ein Mosaik von Photozellen verwendet.

ca. 1890
P. Nipkow erfindet den sequentiellen Scanner.

ca. 1900
Erste erfolgreiche Charaktererkennung durch russischen Wissenschaftler Tyurin, cf. [Man86].

Mitte der 40er Jahre
Erste „moderne“ Versionen von OCR durch Entwicklung des digitalen Computers.

Mitte der 50er Jahre
Erste kommerzielle OCR-Maschinen erhältlich. Erste Maschine bei Reader's Digest 1954 installiert, um maschinengeschriebene Verkaufsberichte in Lochkartenformat zu konvertieren.

Ca. 1960-1965
OCR-Systeme der ersten Generation. Lesen von Buchstaben eingeschränkter Form. Symbole zum Teil speziell entworfen, um Maschinenlesbarkeit zu erreichen. Später bis zu 10 Fonts unterscheidbar. Beschränkt durch das verwendetet Erkennungsverfahren: Template Matching (das Bild eines Charakters wird verglichen mit einer Bibliothek von prototypischen Bildern für jeden Charakter und jeden Font).

Ca. 1965-1975
OCR-Systeme der zweiten Generation. Einschränkungen bezüglich der Form der Symbole entfallen nun. Auch erste Schritte zur Handschrifterkennung (Ziffern und wenige weitere Buchstaben und Symbole). IBM 1287 als erstes System dieser Art, New York 1965 (Vgl. Foto in [Bal70]). Toshiba entwickelt die erste automatische Briefsortier-Maschine zur Erkennung von Postleitzahlen. Hitachi entwickelt die erste „High Performance“ OCR-Maschine mit „Low-Cost“. Entwicklung von Standardsets für OCR-lesbare Symbole. OCR-A amerikanischer Standard, OCR-B europäischer Standard (mehr Fonts). Heute noch auf manchen Formularen zu finden.

Ca. 1975- Ende 80er
OCR-Systeme der dritten Generation. Behandlung von Dokumenten schlechter Qualität. Bearbeitung einer großen Vielzahl von Symbolen und Fontfamilien („MultiFont-Systeme“) auch handgeschrieben. Durch Verbesserungen der Hardware immer mehr „High Performance“ bei „Low-Cost“. Schreibmaschinen mit mehreren Schriftfamilien wichtige Quelle der Textproduktion, Text-Editorsysteme plus Laserdruck noch sehr teuer. Daher einfache OCR-Systeme bedeutsam, mit denen billig produzierte Schreibmaschinentexte (Drafts) vermöge OCR zum Schlußeditieren in Wortprozessoren konvertiert wurden.

Ende 80er bis heute
Entwicklung von Systemen zur Erkennung beliebiger Fonts („OmniFont-Systeme“). Durch Reduktion der Kosten für Hardware und durch billige Software allgemeine Verbreitung von OCR bei Verlagen, Bibliotheken, Firmen, Behörden. Spezielle Anwendungen, wobei die Erkennung von Dokumentenstruktur und inhaltlicher Bedeutung als Bestandteil immer wichtiger wird (Buchkonversion, Scheckanalyse, Formularanalyse, Eingangspostanalyse). OCR+DA etabliert sich immer mehr als Forschungsfeld. Unterschied zwischen kommerziellen Systemen (notwendig hocheffizient) und Laborsystemen (möglichst weitgehende und fehlerfreie Erfassung). Erfassung vieler Schriftarten (Chinesische, Japanische, Koreanische, Arabische, Kyrillische Schriftsymbole, Mathematische Notation, Musiknotation,

2.3 Teilschritte und Verfahren bei der OCR und Dokumentenanalyse

Dieses Teilkapitel fasst im wesentlichen die Inhalte aus Kapitel 1 von [HB97] zusammen. Die optische Charaktererkennung gliedert sich in eine Reihe von Schritten und Teilverfahren. Üblicherweise werden folgende Teilverfahren unterschieden.

1. Bild-Akquisition
2. Vorbereitung und Bildtransformation
3. Layout-Segmentierung,
4. Merkmalsextraktion,
5. Symbol-Klassifikation,
6. Nachkorrektur,
7. gegebenenfalls strukturelle-inhaltliche Analyse des Dokuments "Document Understanding".

Wird ein System zur Bearbeitung unterschiedlicher Dokumententypen eingesetzt, so findet oft zusätzlich eine Prä-Klassifikation der Dokumente statt, welche dann je nach Typ verschiedenen Bearbeitungsschleifen zugeführt werden. Beispielweise stellt die Bearbeitung von Dokumenten mit freiem Layout andere Anforderungen als die Analyse von Formularen eingeschränkter oder bekannter Struktur.

2.3.1 Bild-Akquisition

Beim Scannen wird ein digitales Abbild des Originaldokuments erstellt. Die bei der OCR verwendeten optischen Scanner transferieren ein flächenhaftes Bild kontinuierlicher Helligkeitswerte in eine diskrete Mengen von Pixeln mit einer vordefinierten Menge von Graustufen. Das dergestalt vereinfache Bild wird noch in spezieller Weise kodiert, um den Speicherbedarf möglichst gering zu halten. Zu unterscheiden sind demgemäs drei Teilschritte:

1. Das "Sampeln" des Bilds, wo die Helligkeitswerte für die Punkte (Pixel) eines Rasters geeigneter Gitterbreite erfasst werden,
2. Die Quantisierung, d.h. die Approximation der Helligkeitswerte an den Gitterpunkten durch geeignete Graustufenwerte aus einem kleinen Spektrum möglicher Werte
3. Die Kodierung.

Sampeln

Wir nehmen an, daß sich die Helligkeitswerte des (schwarz-weiß) Bilds mittels einer reellwertigen Funktion $i(x, y)$ beschreiben läßst. Hierbei durchlaufen die Koordinaten x, y die Bildfläche. Beim Sampeln werden die Werte $i(x, y)$ an einer diskreten Punktemenge („grid“) erfasst. Wir erhalten die Wertemenge $i_s(k\Delta x, l\Delta y)$ $(k, l \in \mathbb{N})$ wobei Δx und Δy die räumlichen Sampling-Intervalle in x- und y-Richtung darstellen. Die zentrale Frage bei diesem Vorgang besteht darin, für welche Gittergröße sich die ursprüngliche Funktion i in genau oder in hinreichend guter Annäherung aus den Werten für i_s wieder rekonstruierten läßt. Diese Frage wird durch *Shannon's Sampling Theorem* beantwortet:

Theorem 2.3.1 *Es bezeichne $I(f_x, f_y)$ die Fourier-Transformierte der Helligkeitsfunktion $i(x,y)$. Gilt $I(f_x, f_y) = 0$ für $|f_x| > B_x$ und $|f_y| > B_y$ für geeignete "Bandbreiten" B_x und B_y, und wählt man Δx und Δy in der Weise, dass $1/(\Delta x) > 2B_x$ und $1/(\Delta y) > 2B_y$, so kann $i(x,y)$ in voller Genauigkeit aus den Werten $i_s(k\Delta x, l\Delta y)$ rekonstruieren.*

Hierbei werden $1/(\Delta x)$ und $1/(\Delta y)$ als Sampling-Frequenzen in x bzw. y-Richtung bezeichnet. Die Fouriertransformierte von $i_s(k\Delta x, l\Delta y)$ besteht aus einem Gitter (im Frequenzraum) von Punkten, an denen sich die Fouriertransformierte von $i(x,y)$ wiederholen. Die obige Bedingung garantiert, daß es zwischen unterchiedlichen Kopien keinen Überlapp gibt, so daß sich $I(f_x, f_y)$, damit $i(x,y)$ voll rekonstruieren läßt. Hierzu vergleiche man Abbildung 1 in [HB97].

In der Praxis ist $i(x,y)$ meist nicht im obigen Sinn bandlimitiert. Dann wird $i(x,y)$ zunächst durch ein bandlimitiertes Signal $i_1(x,y)$ approximiert, das dann gesampelt wird ("anti-aliasing"). In $i_1(x,y)$ werden die niedrig-frequenten Anteile von $i(x,y)$ beibehalten, die hoch-frequenten Anteile werden abgeschnitten ("low-pass filtering"). Für viele Anwendungen ergibt sich ein besseres Verhalten, wie wenn direkt das Signal $i(x,y)$ gesampelt wird. Hierzu vergleiche man Abbildung 2 in [HB97] und die Erläuterungen im Text.

In der Praxis stellt sich die Frage nach einer adäquaten räumlichen Auflösung Δx und Δy. Die Wahl der Gitterbreite, damit die Güte der Approximation, erfolgt anwendungspezifisch. Übliche Gitterbreiten liegen im Bereich von 50 dpi (dots per inch) bis 600 dpi. Eine sehr hohe Auflösung kann durchaus zu unerwünschten Nebeneffekten führen, indem zum Beispiel die Papierstruktur die Darstellung beeinflusst. Außerdem führt sie zu einer großen Menge von Daten. Eine zu geringe Auflösung andererseits kann die Erkennung von Symbolen eines kleinen Fonts unmöglich machen. Auch die Zahl der nachfolgend verwendeten Graustufen beeinflusst die notwendige Gitterbreite. Liegen keine Erfahrungswerte bei der Wahl der Auflösung vor, so ist es durch Anwendung des Shannonschen Sample Theorems möglich, die Gitterbreite so zu wählen, daß man die ursprüngliche Signalfunktion aus der Samplefunktion in guter Approximation rekonstruieren kann.

Quantisierung

Üblich sind Graustufenbilder, bei denen pro Pixel 256 Graustufen unterschieden werden. Da es zumindest bei einfachen OCR-Anwendungen im digitalen Bild nur darum geht, Pixel von Textsymbolen von Pixeln des Bildhintergrunds zu unterscheiden, geht man letztendlich oft in eine *binäre Bilddatei* mit nur zwei Graustufen (schwarz-weiß) über. Diese wird jedoch in der Regel erst aus einer zuvor berechneten und behandelten Darstellung mit mehreren Graustufen berechnet. Bei der Übersetzung in Graustufenwerte ist darauf zu achten, daß der Informationsverlust oder Gesamtfehler möglichst gering gehalten wird. Man wird also die Grenzwerte für die einzelnen Graustufen nicht a priori festlegen, sondern in Abhängigkeit des vorgefundenen Wertespektrums. Danach kann man den Grenzwert für den Bereich, der in der Binärdarstellung schwarz ist, lokal in Abhängigkeit von Kontrast und anderen Eigenschaften festsetzen. Dieses ,,adaptive thresholding" ist allerdings in Bezug auf Rechenzeit und Speicherbedarf aufwendiger das ,,uniform thresholding" (cf. Abbildung 9 [Bal70]).

Kodierung

In Abhängigkeit vom Umfang des so erhaltenen Datenmaterials werden im Anschluß noch Techniken der Bildkodierung angewandt, die helfen, den Datenumfang zu verkleinern.

Run-length coding. Das Bild wird Pixel-Zeile für Zeile durch Angabe der Startposition und Länge schwarzer Pixelfolgen kodiert. Vgl. hierzu S. 11 Kapitel 1 von [HB97].

Huffman Code. Basiert auf der ungleichen Häufigkeit verschiedener Symbole in zu komprimierenden Files. Man kodiert häufige Symbole mit kurzen Codes, seltene Symbole mit längeren. Insgesamt erhält man so eine komprimierte Darstellung. Kodierung: Man ordnet die Symbole nach absteigender Häufigkeit. Die zwei seltendsten Symbole werden zu einer Klasse (Metasymbol) zusammengefasst, deren Wahrscheinlichkeiten werden addiert. Alle (neuen und alten) Metasymbole werden wieder frequenzgeordnet. Dies iteriert man unter Bildung komplexerer Metasymbole, bis nur zwei Metasymbole uebrig sind. Das häufigere erhält Code 0, das seltenere Code 1. Metasymbole mit Code c werden nun wieder gesplittet, das häufigere Teilmetasymbol erhält Code c0, das andere Code c1. Am Ende erhält so jedes echte Symbol eine Bitfolge. Vgl. hierzu Fig. 6 in Kapitel 1 von [HB97].

2.3.2 Vorbereitung und Bildtransformation

Dieser Schritt beinhaltet eine Vielzahl spezieller Techniken, die dazu dienen, Verzerrungen des Bildes zu korrigieren, den Papierhintergrund vom Bereich des eigentlichen Textes oder Bildes zu trennen, Schmutzeffekte und andere Störungen zu beseitigen, Löcher zu flicken, Kanten zu verbessern, Linien in Bezug auf ihre Stärke zu normalisieren und ähnliche Verbesserungen durchzuführen. Ein Teil der unten erwähnten Segmentierungstechniken setzt nachfolgende Vorbereitungsschritte bereits voraus. So muß zum Beispiel vor der X-Y-Baum-Dekomposition (s.u.) eine Hintergrundtrennung stattgefunden haben und es müssen Verdrehungen des Bildes korrigiert sein. Andererseits werden aber manchmal auch unterschiedliche Vorbereitungsschritte für die verschiedenen Regionen einer Seite durchgeführt, was dann bedingt, daß eine Grobsegmentierung vorausgeht. Daher ist die Reihenfolge zwischen Vorbereitungs- und Segmentierungsschritten unterschiedlich. Auf zwei Gruppen von Methoden gehen wir nachfolgend etwas näher ein.

Geometrische Transformationen

Geometrische Transformationen werden eingesetzt, um Störungen zu beseitigen, die durch eine nichtoptimale Aufnahme des Bildes durch den Scanner resultieren. Beispielsweise kann das Dokument verdreht sein, oder verschiedene Formen einer Wölbung aufweisen. Typische Effekte entstehen beim Scannen von Büchern durch die Wölbung der aufgeschlagenen Seiten. Beim Normalisieren des Bildes werden diese Effekte korrigiert. Hierzu muß zunächst die Art der Störung ermittelt werden. Die Störung wird dann mathematisch durch eine Transformation beschreiben.

Beispiel affine Transformationen. Affine Transformationen sind durch Gleichungen

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

beschrieben. Hierunter fallen die Translation ($a, b, d, e = 0$), das Scaling ($a = e = 0$), die Rotation und die Schrgstellung (slant).

Andere Transformationen sind der Fasseffekt (Wölbung nach vorne) und der Nadelkisseneffekt (Wölbung nach hinten).

Die Parameter der Transformation sind abzuschätzen, um ein korrigiertes Bild zu erhalten, aus dem dann nach einem Resampling Prozess das Ausgabebild berechnet wird.

Neben Störungen, die das ganze Bild betreffen, können bei Formularen auch einzelne handschriftliche Symbole verdreht oder unterschiedlich in Bezug auf die Form und Größe sein. Geometrische Transformationen werden daher teilweise auch auf einzelne Symbole angewandt, um diese in eine standardisierte Form bzgl. Größe, Schräge etc. zu bringen. Natürlich setzt dies voraus, daß eine Segmentierung bis zum Symbollevel zuvor durchgeführt wurde.

Filterungstechniken

Filterungstechniken dienen dazu, kleine Brüche und Löcher auszugeschmieren („Filling“), Linien dünner zu machen („Thinning“) und ähnliche Verbesserungen der Darstellung zu erreichen. Der Wert des Bildes an einer Position (k, l) wird hierbei in Abhängigkeit der Werte des Bildes in der Umgebung angepaßt.

Beispiel 2.3.2 Bei der Verwendung *linearer Filter* ergibt sich der neue Bildwert am Punkt (k, l) durch eine Linearkombination der Werte in einer $n \times n$-Umgebung von (k, l), wobei ein Umgebungsquadrat mit der Kantenlänge $n = 3$ üblich ist. Lineare Filter werden — meist in Verbindung mit anderen Operationen — dazu verwendet, um Ecken, Linien und Kanten zu entdecken.

Beispiel 2.3.3 *Rangordnungsfilter*, die hauptsächlich für Bilder mit mehreren Graustufen angewandt werden, analysieren gleichfalls eine $n \times n$-Umgebung des fokussierten Punkts (k, l) und berechnen die Rangfolge aller dort auftretenden Grauwerte. Sie weisen dann dem Punkt (k, l) auf dieser Basis einen neuen Grauwert zu.

- Beim *Erosionsfilter* wird der kleinste, beim *Dilatationsfilter* der größte in der Umgebung auftretende Grauwert verwendet.
- Bei der *Konturen-Entdeckung* wird die Differenz von größtem und kleinstem Grauwert, beim *Medianfilter* der mittlere Grauwert verwendet.

Beispiel 2.3.4 Bei der Verwendung *morphologischer Filter* wird im einfachsten Fall („hit-or-miss“) über die Umgebung eines Bildpunkts (k, l) eine vordefinierte Maske mit Binär- oder Grauwerten gelegt. Ergibt sich ein perfekter Match zwischen Maske und den Graustufen der Umgebung, so erhält die Position (k, l) einen vordefinierten Wert. Dies ist in Abbildung 2.1 illustriert, wo sich durch die Anwendung der Maske aus dem linken Bild das Ausgabebild auf der rechten Seite ergibt. Bei komplizierteren Filtern dient die Maske lediglich dazu, eine Menge V von Translationen des Bildes zu definieren. Die in Abbildung 2.1 angegebene Maske beschreibt die Verschiebungen in acht Richtungen: um ein Pixel nach oben, unten, nach links und nach rechts, sowie in die gemittelten Richtungen. Aus der Menge der verschobenen Bilder wird durch eine definierte mathematische Vorschrift das neue Bild berechnet.

•	1	1	1	•	•						•	1	1	1	•	•
1	1	•	1	•	•		1	1	1		1	1	1	1	•	•
1	1	1	1	1	•		1	?	1		1	1	1	1	1	•
1	•	1	•	1	•		1	1	1		1	1	1	1	1	•
1	1	1	1	1	•			?->1			1	1	1	1	1	•
1	1	•	•	•	•						1	1	•	•	•	•

Abbildung 2.1: Anwendung einer Maske (Mitte) auf ein Binärbild.

•	•	1	1	1	•	•		1	1	1	1	1	1	•		1	1	1	1	1	•	•
•	1	1	•	1	•	•		1	1	1	1	1	1	1		1	1	1	1	1	•	•
1	1	1	1	1	1	•		1	1	1	1	1	1	1		1	1	1	1	1	1	1
•	•	•	1	•	1	•		1	1	1	1	1	1	1		•	•	1	1	1	1	1
•	•	•	•	1	1	•		•	•	1	1	1	1	1		•	•	•	•	1	1	1
•	•	•	•	•	•	•		•	•	•	1	1	1	1		•	•	•	•	1	•	•
•	•	•	•	1	•	•		•	•	•	1	1	1	•		•	•	•	•	1	•	•

Abbildung 2.2: ,,Closing“ eines Binärbilds durch Dilatation und anschließende Erosion.

- Bei der *Dilatation* verwendet man an jedem Punkt (k, l) das Maximum der Werte, das man bei (k, l) durch die verschobenen Bilder erhält. Aus mengentheoretischer Sicht ergibt sich das neue Bild als Vereinigung der verlagerten Bilder. Die Dilatation führt daher zu einem Aufblähen des Bildes.

- Bei der *Erosion* wird analog an jedem Punkt das Minimum der erhaltenen Werte verwendet, was mengentheoretisch auf eine Durchschnittsbildung hinausläuft.

- Beim Schließen (,,Closing“) wird eine Dilatation ergänzt um eine anschließende Erosion. Die Wirkung ist in Abbildung 2.2 illustriert.

- Beim Öffnen (,,Opening“) wird umgekehrt zunächst eine Erosion durchgeführt, im Anschluß eine Dilatation. Dies ist in Abbildung 2.3 illustriert.

Die Beispiele machen deutlich, daß eine enge Verwandtschaft zwischen Rangordnungsfiltern und morphologischen Filtern besteht.

•	•	1	1	1	•	•		•	•	•	1	•	•	•		•	•	1	1	1	•	•
•	1	1	1	1	•	•		•	•	•	1	•	•	•		•	•	1	1	1	•	•
1	1	1	1	1	1	1		•	•	•	1	•	•	•		•	•	1	1	1	1	•
•	•	1	1	1	1	•		•	•	•	•	1	•	•		•	•	1	1	1	1	•
•	•	•	1	1	1	•		•	•	•	•	1	•	•		•	•	•	1	1	1	•
•	•	•	1	1	1	•		•	•	•	•	•	•	•		•	•	•	1	1	1	•
•	•	•	•	1	•	•		•	•	•	•	•	•	•		•	•	•	•	•	•	•

Abbildung 2.3: ,,Opening“ eines Binärbilds durch Erosion und anschließende Dilatation.

2.3.3 Layout-Segmentierung

Bei diesem Prozess werden die Konstituenten des digitalen Bildes berechnet. Hierbei werden Dokumentenregionen mit Bildern und Graphiken getrennt von Regionen mit Text. Beim automatischen Sortieren von Briefumschlägen wird der Bereich der Empfängeradresse von anderen Bildanteilen (Briefmarke, Stempel, Logos, Absender) getrennt. Schwierigkeiten treten etwa auf, wenn Text auf Bilduntergrund zu finden ist.

Der eigentliche Textbereich wird dann weiter unterteilt in Zeilen (und Wörter[1]) und einzelne Symbole. In [HB97] sind vier Segmentierungsverfahren angegeben, von denen wir zwei kurz zusammenfassen.

Beispiel 2.3.5 Ein wichtiges Verfahren ist die Ermittlung der zusammenhängenden Komponenten (Flächen schwarzer Pixels). Hierbei wird das Binärbild in der üblichen Lesereihenfolge von links nach rechts und von oben nach unten Zeile für Zeile durchgegangen. In jeder Zeile werden die Punkte jeder ununterbrochenen Folge von schwarzen Pixeln mit einer Nummer gelabelt, wobei für neue Folgen, die auch nicht nach oben an eine schwarze Pixelfolge der Vorgängerzeile angrenzen, jeweils eine neue Nummer vergeben wird. Die zu einer gelabelten Folge benachbarten schwarzen Pixels der Nachfolgerzeile erhalten hierbei jeweils dieselbe Nummer. Wenn zwei schwarze Bereiche, die in Zeile i noch getrennt getrennt und mit verschiedenen Nummern versehen waren, in Zeile $i + 1$ zusammenwachsen, so werden die Labels vereinheitlicht. Am Ende stellen die Punkte desselben Labels genau die Zusammenhangskomponenten dar. Allerdings entsprechen diese nicht immer den einzelnen Symbolen. Verschiedene Symbole können — ausgelößt durch eine zu geringe Auflösung, durch einen zu niedrigen Grenzwert oder andere Phänomene (Kursivschrift) — zu einer Zusammenhangskomponente verschmelzen. Auch kann ein einzelnes Symbol — ausgelöst u.a. durch zu hohes Thresholding — in verschiedene Zusammenhangskomponenten fallen (m $\rightarrow$ in). Ein spezielles Problem stellen i-Punkte, Umlautzeichen und Akzente dar. Diese werden oft als Schmutzpunkte interpretiert und unterdrückt (s.u.).

Beispiel 2.3.6 Als weiteres Beispiel eines Segmentierungsverfahren soll die sogenannte X-Y-Baum Dekomposition beschreiben werden, die wir anhand Abbildung 2.4 erklären. Hierbei wird zunächst eine Projektion des Bereichs schwarzer Pixel in horizontaler Richtung berechnet. Beim linken Beispiel wird ersichtlich, daß man durch die leeren (weißen) Projektsbereiche unter geeigneten Voraussetzungen direkt die Zeilenaufteilung der Seite erhält. Für jede erhaltene Zeile wird im Anschluß eine entsprechende Projektion in vertikaler Richtung durchgeführt. Diese ergibt die Aufteilung der Zeile in Symbole oder Wörter (Kursivschrift). Die Projektion der letzten Zeile ist in der Abbildung angedeutet.

Das mittlere Beispiel verdeutlicht, daß zwei Projektionen im allgemeinen nicht ausreichend sind. Hier erhält man durch eine horizontale Projektion zunächst keine vollständige Zeilenaufteilung. Jedoch ergibt sich aus der anschließenden vertikalen Projektion die Aufteilung in linke beziehungsweise rechte Spalte. Eine spaltenweise horizontale Projektion führt dann auf die Zeilen, die abschließende vertikale Projektion jeder Zeile einer Spalte ergibt dann die Aufteilung in Symbole oder Wörter.

Beim rechten Beispiel schließlich wird klar, daß es bei der X-Y-Baum-Dekomposition im Zusammenhang mit sogenannten ,,Fließelementen“ zu unlösbaren Problemen kommen kann. Bei der dargestellten Konstellation ergibt sich weder durch eine horizontale noch durch eine vertikale Projektion eine Aufgliederung des

[1] Die Ermittlung zusammenhängender Symbolketten (,,Grouping“) erfolgt oft erst bei der Nachbearbeitung.

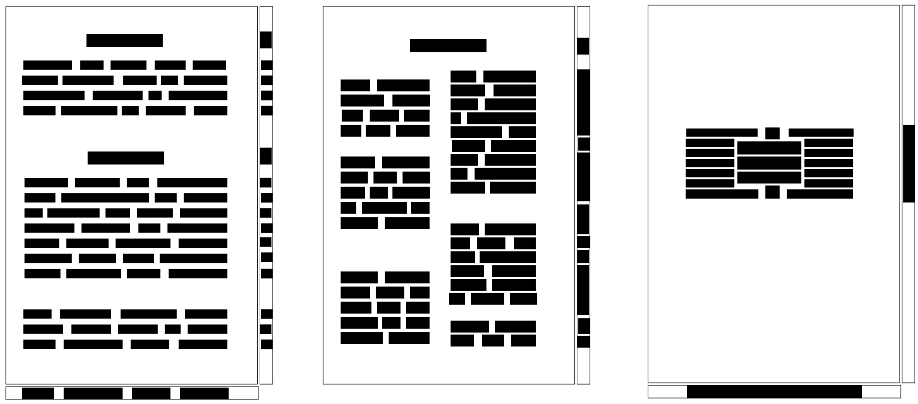


Abbildung 2.4: Zum Prinzip der X-Y-Baum Dekomposition.

Bildes.

Andere in [HB97] dargestellte Segmentierungstechniken beruhen auf dem sogenannten ,,run-length smearing“ und der Hough-Transformation.

2.3.4 Merkmalsextraktion

Fast alle Formen der Symbolklassifikation setzen die Extraktion geeigneter Features voraus. Die Grundidee beruht darauf, daß die Klassifikationsverfahren nicht die Bitmap-Beschreibung eines Symbols selbst als Eingabe verwenden, sondern das für das Symbol eine Reihe charakteristischer Eigenschaften ermittelt werden, die dann als Eingabe dienen. Die Vielzahl möglicher Feature-Systemen werden oft in drei Gruppen eingeteilt

1. Verteils- und Momentenverfahren. Hierbei basieren die Features auf der statistischen Verteilung von Punkten über den Bereich des Symbols. Beim *Zoning* wird dieses Rechteck in mehrere Teilbereiche unterteilt, in denen dann die Dichte der schwarzen Punkte gemessen wird. Bei den *Momentenverfahren* werden die sogenannte Momente, wie zum Beispiel die Lage des Gravitätszentrums, als Features verwendet. Bei der Verwendung von *Crossings und Entfernungen* wird gemessen, wie oft der Bereich des Symbols in bestimmten Richtungen betreten und verlassen wird. Diese Technik wird oft bei kommerziellen Systemen angewandt, da sie einfach implementiert werden kann und effizient ist.

2. Transformations- und Reihenentwicklungsverfahren. Mit diesen Techniken kann die Dimensionalität des vollen Symbolbildes (aufgefasst als Featureraum) reduziert werden und die extrahierten Features können invariant unter globalen Deformationen wie Translation und Rotation gemacht werden. Beispiel sind Fourier-Transformation, Karhunen-Loeve und Houghtransformation u.a.m.

3. Strukturelle Analyse. Hierbei werden Features, die die geometrische und topologische Gestalt eines Buchstabens beschreiben ermittelt. Verwendet werden charakteristische Linien (z.B. T und L haben zwei Linien), Bögen, Endpunkte ermittelt.

Alle Feature-Systeme haben Vor- und Nachteile. Qualitätskriterien, die je nach

Anwendungsbereich verschieden zu gewichten sind, sind die Erkennungsgeschwindigkeit (Effizienz) der Feature-Extraktion, die Komplexität der Implementierung, die Robustheit gegenüber Schmutzeffekten, Verzerrungen, Stilvariationen, Translationen, Rotationen u.ä.

2.3.5 Symbol-Klassifikation

Bei diesem Prozeß wird aufgrund der ermittelten Features ein Eingabepattern als ein Symbol eines vorgegebenen Alphabets klassifiziert. Es gibt eine Vielzahl möglicher Klassifikationsverfahren, auf die wir in Kapitel 3 etwas näher eingehen.

2.3.6 Nachbehandlung

Beim *Grouping* werden Folgen von Symbolen zusammengruppiert, um Wörter und ähnliche ununterbrochene Zeichenfolgen zu bestimmen. Im Prinzip werden einfach Symbole, die nah genug beieinander stehen, zusammengruppiert. Für Fonts mit fixiertem Buchstabenabstand (Schreibmaschine) ist das einfach. Für gedruckte Dokumente ist normalerweise der Abstand zwischen zwei Wörtern signifikant größer als der Abstand zweier aufeinanderfolgender Buchstaben eines Worts. Große, zum Teil unlösbare Probleme ergeben sich im Zusammenhang mit Schrägschriften und Handschriften.

Bei der *Nachkorrektur* wird versucht, fehlklassifizierte Symbole zu erkennen und zu korrigieren. Wir werden auf dieses Problem in nachfolgenden Kapiteln ausführlich eingehen.

2.3.7 Document Understanding

Bei anspruchsvolleren Anwendungen, wie sie in den zurückliegenden Dekade entstanden sind, schließt sich nach der Erkennung von Symbolen und Wörtern eine Analyse der logischen Struktur an. So möchte man etwa beim Erfassen von Zeitungen automatisch die logische Aufteilung des Textes in Artikel, Artikelüberschriften und ähnliches ermitteln. Beim Erfassen von wissenschaftlichen Artikeln möchte man Titel, Autoren, Adressangaben, Abstracts, Kapitel, Überschriften, bibliographische Angaben korrekt erfassen und markieren. Im Zusammenhang mit der Standardisierung von Dokumentenrepräsentationen durch Markup-Sprachen wie XML oder SGML besteht ein sinnvolles, aber im allgemeinen schwer zu erreichendes Ziel darin, eine reichhaltige Repräsentation der logischen Struktur zu erhalten. Weiter ist es interessant, Verfahren der Inhalterschließung von Texten, wie sie im Text Retrieval verwendet werden, einzusetzen, um eine möglichst präzise symbolische Repräsentation des Dokumenteninhalts zu erreichen.

2.4 Charakteristische Fehler, Grenzen

Bei allen oben genannten Arbeitsschritten treten Fehler auf. Die nachfolgende Liste ist keinesfalls erschöpfend und macht nur auf einige Probleme aufmerksam. Bei der Segmentierung wird oft ein Einzelsymbol getrennt (m $\rightarrow$ in) oder zwei Symbole verschmolzen (in $\rightarrow$ m). Einzelwörter werden manchmal fälschlich aufgeteilt, Wortpaare manchmal fälschlich verschmolzen. Text auf Hintergrund wird oft unzureichend getrennt, etwa bei farblich hinterlegten Schriftanteilen. Die Unterteilung des Textes in logische relevante Gruppen oder Textblöcke ist oft unzureichend. Bei kleinen Fontgrößen (ab 6pt) werden die Ergebnisse oft generell unbrauchbar. Das Problem der Handschriftenerkennung ist weitgehend ungelöst, insbesondere wenn Buchstaben verbunden sind, und Kursivschriften sind generell problematisch.

2.5 Anwendungen der OCR und Dokumentenanalyse

Die Liste der Gebiete, in denen Techniken der OCR und Dokumentenanalyse eingesetzt werden, umfaßt neben anderen Anwendungen die folgenden:

- Einsatz bei verschiedenen Lesehilfen für Blinde,
- Automatische Postadresserkennung und Postsortierung in zentralen Postämtern,
- Formularerkennung und -Analyse,
- Qualitätskontrolle beim Druck,
- Dokumentensortierung,
- Automatische Nummernschilderkennung, Verkehrskontrolle,
- High-Speed Datenerfassung,
- Erfassung von Musiknoten,
- Automatische Fingerabdruckerkennung,
- Scheck-Sortierung und -Analyse
- Akquisition von Gesundheitsdaten,
- Preis-Lesesysteme und Bar-Code Erfassung,
- Automatische Kartenlesesysteme,
- Erfassung von Ingenieurs-Zeichnungen.

Einige komplexe Anwendungen, wo OCR und Techniken der Dokumentenanalyse eingesetzt werden und Vorteile gegenüber anderen Methoden der digitalen Erfassung bieten, sind unter anderem

- die elektronische Aufbereitung von Archivmaterial zur Unterstützung unterschiedlicher Rechercheaufgaben,
- die Neuauflage von Büchern und Schriften bei Bibliotheken und Verlagen nach einer retrospektiven Konversion,
- die Überführung von gedruckten Büchern und Dokumenten in elektronisches Format, gegebenenfalls zur Internetpräsentation einschließlich der Inhaltserfassung durch Suchmachinen und Textretrievalsysteme,

- die selektive Auswahl und Integration von Dokumententeilen in thematisch fokussierende Informationssysteme wie Pressespiegel und Informationsdatenbanken und
- das effiziente Bearbeiten der Eingangspost bei Unternehmen durch automatische Analyse und Erledigung im Workflow.

Bei anspruchsvollen Anwendungen ist jedoch die Fehleranfälligkeit der Systeme auch heute noch ein kritischer Punkt, so daß teilweise aus reinen Kostengründen — die manuelle OCR-Nachkorrektur ist sehr teuer — andere Erfassungsmöglichkeiten bevorzugt werden.

2.6 Bibliographische Anmerkungen

Bücher und Buchbeiträge über OCR sind [Fu82, Nag82, Sri84]. Weitere Reports, die die „state of the art“ zu verschiedenen Zeitpunkten beschreiben, sind in [GS90] erwähnt. Einen Überblick liefert auch L. Eikvil [Eik93].

Kapitel 3

Symbol-Klassifikation

Als *Klassifikator* bezeichnen wir nachfolgend ein Verfahren, das eine geeignete Beschreibung eines Patterns als Eingabe nimmt und dem Pattern darauf eine Klasse unter einer vorgegebenen Menge möglicher Klassen zuweist. Bei der OCR-Erfassung einzelner Symbole sind die Pattern die digitalen Bilder von Vorkommen der Symbole in einem gescannten Text. Geeignete Beschreibungen werden durch die Extraktion von Features erreicht. Die Menge der möglichen Klassen ist durch das Alphabet der möglichen Symbole des Texts vorgegeben und umfasst anwendungsabhängig neben den üblichen Buchstaben und Ziffern gegebenenfalls noch zusätzliche Sonderzeichen.

Im Bereich der OCR-Erkennung sind zahlreiche Symbol-Klassifikatoren und darauf abgestimmte Arten der Patternbeschreibung bekannt. Es ist unmöglich, in diesem Rahmen einen auch nur annähernd vollständigen Überblick zu geben. Wir stellen zunächst exemplarisch die in [AB95] beschriebene Klassifikationsmethode vor und erwähnen im zweiten Teil kurz einige wichtige Alternativen.

3.1 Multifonterkennung und Symbol-Klassifikation mit Hidden Markov Modellen

Wir beginnen mit einer kurzen Darstellung der Hidden Markov Modelle (HMM). Eine ausführliche Beschreibung wird in [Rab89] gegeben.

Hidden Markov Modelle

Um HMM zu erklären, beginnen wir mit einem Beispiel. Stellen wir uns eine Reihe von N Urnen vor, in denen rote, grüne und gelbe Kugeln liegen. Wir wollen aus den Urnen Kugeln ziehen (mit Zurücklegen). Für jede Urne sei die jeweilige Zahl der Kugeln einer bestimmten Farbe bekannt. Damit ergibt sich für jede Urne eine charakteristische Wahrscheinlichkeit dafür, daß eine Kugel einer bestimmten Farbe gezogen wird. Weiter sei zwischen je zwei Urnen eine bestimmte Übergangswahrscheinlichkeit festgelegt. Die Übergangswahrscheinlichkeit zwischen den Urnen i und j gibt an, mit welcher Wahrscheinlichkeit Ziehung $k+1$ aus Urne j erfolgt, wenn Ziehung k aus Urne i erfolgt. Um die Beschreibung zu komplementieren, seien noch für jede der Urnen die Wahrscheinlichkeit festgelegt, daß sie bei der ersten Ziehung verwendet wird. Nun verkomplizieren wir den Prozess und stellen uns vor, wir können die Reihenfolge der Urnen nicht mehr sehen, sondern bekommen nur noch die Farben der jeweils gezogenen Kugeln gemeldet. Mit Hilfe des unten beschriebenen *Viterbi-Algorithmus* kann man bei einer gegebenen Folge von Kugelfarben auf die wahrscheinlichste Folge der Urnen schließen, aus denen nacheinander gezogen wurde. Die Gesamtwahrscheinlichkeit dafür, daß die zugehörige Urnenfolge eintritt und dabei die beobachteten Kugelfarben gezogen werden, wird manchmal als der sich ergebende „Viterbi-Score“ bezeichnet. HMMS, das Viterbi-Verfahren und der ermittelte Viterbi-Score einer Beobachtungsfolge können in unterschiedlicher Weise bei konkreten Klassifikationsaufgaben verwendet werden.

Beispiel 3.1.1 Wir stellen uns vor, daß Wörter in eindeutiger Weise aus einer endlichen Menge von (silbenähnlichen) charakteristischen Buchstabenfolgen zusammengesetzt sind. Jede einzelne Aussprache einer solchen Buchstabenfolge ist mit einem Laut verbunden ist, den wir messen können. Wir nehmen an, die Messung ist diskret und ergibt einen Wert aus einer endlichen Menge. Eine fest vorgegebene Buchstabenfolge kann jedoch bei mehrmaliger Aussprache zu verschiedenen Werten führen. Können wir nun aus einer gemessenen Folge von Messwerten auf die wahrscheinlichste Folge von charakteristischen Buchstabenfolgen schließen, die zu dieser Messung führt, so ermöglicht dies offenkundig eine primitive Form von Spracherkennung.

Wenn wir von dem Urnenbild etwas abstrahieren, kommen wir zu folgender Grundform eines Hidden Markov Modells.

Definition 3.1.2 Ein *Hidden Markov Modell* (HMM) ist ein Quintupel der Form $\lambda = \langle \mathcal{S}, \mathcal{V}, A, B, C \rangle$ wo gilt:

1. $\mathcal{S} = \{S_1, \ldots, S_N\}$ ist eine nichtleere Menge, genannt die Menge der *Zustände* des Modells,

2. $\mathcal{V} = \{V_1, \ldots, V_M\}$ ist eine nichtleere Menge, genannt das Alphabet der *observablen Symbole*,

3. A ist eine Matrix der Dimension $N \times N$, genannt *Matrix der Zustandsübergangswahrscheinlichkeiten.* Einträge sind nichtnegativ, die Summe der Einträge jeder Zeile muß 1 betragen. Das Element $A[i, j]$ heißt *Übergangswahrscheinlichkeit zwischen den Zuständen S_i und S_j.*

4. B ist eine Matrix der Dimension $N \times M$, genannt *Matrix der Emissionswahrscheinlichkeiten.* Einträge sind nichtnegativ, die Summe der Einträge jeder Zeile muß 1 betragen. Das Element $B[i, j]$ heißt *Emissionswahrscheinlichkeit des Symbols V_j im Zustand S_i*,

5. C ist ein Vektor der Länge N, genannt Vektor der *Anfangswahrscheinlichkeiten.* $C[i]$ heißt die *Anfangswahrscheinlichkeit des Zustands S_i.* Die Einträge sind nichtnegativ, die Summe der Einträge muß 1 betragen.

Ein HMM ist bis auf Isomorphie eindeutig gekennzeichnet durch die Zahl N der Zustände, die Zahl M der observablen Symbole und durch durch die Angabe aller Wahrscheinlichkeitsparameter A, B und C.

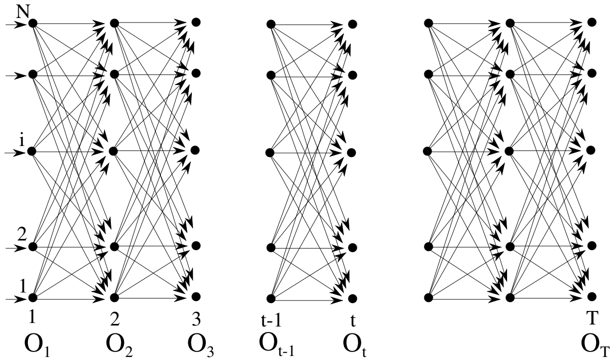


Abbildung 3.1: Zum Prinzip des Viterbi-Algorithmus.

Der Viterbi-Algorithmus

Das Prinzip des Viterbi-Algorithmus kann mit dem Gartenzaun-Bild (englisch: „trellis“) aus Abbildung 3.1 verdeutlicht werden. Die $N \times T$ Punkte des dargestellten Graphen markieren die N möglichen Zustände des vorgegebenen Modells λ für jede der T beobachteten Emmissionen $O_1, \ldots, O_T$. Die Pfeile ganz links stellen die mögliche Wahl eines Anfangszustands dar. Die anderen Pfeile stellen die möglichen Übergang von einem Zustand zu einem anderen dar. Um den Viterbi-Algorithmus zu erläutern, stellen wir uns nun jeden Knoten und jede Kante des Graphen als mit einem Label oder Gewicht versehen vor. Das Label von Punkt $[i, t]$ gibt die Emissionswahrscheinlichkeit des beobachteten Symbols O_t im Zustand S_i an. Die Labels der Pfeile am linken Rand stellen die Wahrscheinlichkeiten $B[i]$ dar, mit denen die betreffenden Zustände i als Anfangszustände angenommen werden. Die übrigen Kantenlabels repräsentieren die Übergangswahrscheinlichkeiten zwischen den beteiligten Zuständen. Ein *maximaler Pfad* des Graphen beginnt mit einem Pfeil der linken Spalte und folgt einer ununterbrochenen Kette von Pfeilen bis zu einem Punkt der Form $[i, T]$. Wenn wir die Gewichte aller Knoten und Kanten eines maximalen Pfads multiplizieren, erhalten wir die Gesamtwahrscheinlichkeit bei gegebenen Modell λ, daß unter Besuch der beteiligten Zustände des Pfades die Folge $O_1 \ldots, O_T$ emittiert wird. Der Viterbi-Algorithmus berechnet den in diesem Sinn wahrscheinlichsten Pfad, indem mittels dynamischer Programmierung der beste Pfad bis zum Gitterpunkt $[i, t]$ und der zugehörige bestmögliche Wahrscheinlichkeitswert $\delta[i, t]$ (der sich durch Multiplikation der bis dahin im Pfad aufgetretenen Gewichte) berechnet werden. Formal lassen sich drei Schritte unterscheiden:

1. *Initialisierung.* Sei $V_j = O_1$ das erste beobachtete Symbol. Wir setzen $\delta[i, 1] := C(i) \cdot B[i, j]$ $(1 \leq i \leq N)$.

2. *Rekursionschritt.* Es liegen die Werte $\delta[1, t-1], \delta[2, t-1], \ldots, \delta[N, t-1]$ vor. Es sei $V_k = O_t$ das t-te beobachtete Symbol. Dann ergibt sich $\delta[i, t]$ als *Maximum* der N Werte $\delta[i', t-1] \cdot A[i', i] \cdot B[i, k]$ die wir für die Alternativen $i' = 1, \ldots, N$ erhalten.

3. *Abschluß.* Der optimale Pfad wird letztendlich durch den besten Wert zum Zeitpunkt T, d.h. den maximalen Wert der Form $\delta[i, T]$ bestimmt, der dann manchmal als *Viterbi-Score* bezeichnet wird.

Wenn wir den Pfad explizit berechnen wollen, so muß man beim Rekursionschritt speichern, welcher Vorgängerzustand $[i', t-1]$ auf den Wert $\delta[i,t]$ führt. Offenkundig ist die Komplexität des Verfahrens quadratisch in der Höhe des Gartenzauns und linear in der Länge, sie beträgt daher $O(N^2 \cdot T)$.

Es ist zu beachten, daß der sich ergebende „Viterbi-Score“ *nicht* die Gesamtwahrscheinlichkeit angibt, bei gegebenem Modell λ die vorgegebene Folge von Observablen $O_1 \cdots O_T$ zu erhalten. Hierzu müßte im obigen Verfahren beim Rekursionsschritt nicht maximiert werden, sondern über alle Wahrscheinlichkeiten aufsummiert werden. Dieser nah verwandte, aber unterschiedliche Prozeß wird oft als „forward“-Algorithmus bezeichnet.

Optimierung der Parameter eines Modells

Die Qualität der Klassifikation mit HMM hängt entscheidend von der Wahl des Modells λ ab. In der Praxis werden eine ganze Reihe unterschiedlicher Spezialmodelle betrachtet, auf die wir hier nicht eingehen. Selbst wenn wir Modelle der oben beschriebenen Art berücksichtigen, bleibt die Aufgabe, die Zahl der Zustände und die Wahrscheinlichkeitsmatrizen A, B und C festzulegen. Für die optimale Wahl einer Zustandszahl sind keine allgemeinen Verfahren bekannt, sie ist daher in der Regel Gegenstand von Tests. Bei festgelegter Zustandszahl gibt es jedoch Verfahren, die aufgrund einer vorgegebenen Folge von Trainingsbeobachtungen die Parameter des Modells in der Weise verbessern, daß sie die vorgegebenen Beobachtungen im Trainingsset in möglichst guter Weise erklären. Hierzu wird üblicherweise das Verfahren von Baum-Welch verwendet. Es würde den Rahmen dieser Zusammenfassung sprengen, dieses Verfahren darzustellen, wir verweisen auf [Rab89].

HMM-Klassifikation von Symbolen bei der OCR

Bei der in [AB95] beschriebenen Klassifikation von gescannten Symbolen mit Hilfe von HMMs ist die Idee etwas anders als in Beispiel 3.1.1. Das digitale Bild eines Symbol-Patterns wird mittels einer fest vorgegebenen Folge von Beschreibungsmerkmalen repräsentiert. Die verwendeten Beschreibungsmerkmale werden unten näher angedeutet. Vermöge einer einfachen Kodierung wird der Wert eines jeden Merkmals als eine Ziffernfolge fester Länge über dem Alphabet $\mathcal{V} := \{0, 1, \ldots, 8\}$ beschrieben. Was wir demnach bei der „Beobachtung“ eines Symbol-Patterns messen, ist die Gesamtfolge $O_1, \ldots, O_T$ der Ziffern, die sich durch die Wertefolge aller Merkmale ergibt. Wir führen nun für jedes Symbol eines Fonts ein eigenes HMM ein![1] Die Vorstellung ist die, daß ein Zustand des einem Symbol zugeordneten HMM einen bestimmten Wert $V \in \mathcal{V}$ „emittiert“. Wenn wir nun eine Folge von Werten $O_1, \ldots, O_T$ aus $\mathcal{V}$ bei der Analyse des Digitalbilds eines Symbol-Patterns messen, so können wir dem HMM, damit dem zugehörigen Symbol, einen Viterbi-Score zuordnen. Dieser Score gibt einen Anhaltspunkt dafür, wie plausibel es ist anzunehmen, daß das betreffende Symbol zu der vorliegenden Ziffernfolge geführt hat. Wenn wir eine eindeutige Hypothese aufstellen müssen, welches Symbol das betreffende Digitalbild verursacht hat, so nehmen wir dieses, das den höchsten Viterbi-Score ergibt. Sind mehrere Hypothesen möglich, da die endgültige Desambiguierung erst später

[1] Die Zahl der Zustände ist nicht von vorneherein festgelegt, die Qualität der Symbolklassifikation kann von einer guten Wahl abhängen. In den nachfolgend geschilderten Experimenten wurden HMM einer bestimmten Kategorie, sogenannte eingeschränkte links-rechts-sequentielle Modelle, und Zustandszahlen von mininal 4 bis maximal 8 verwendet.

erfolgt, so können wir eine Liste der besten k Scores ausgeben, oder eine Liste aller Scores oberhalb eines bestimmten Minimalwerts.

Das angegebene Verfahren wurde in [AB95] zur Charakterklassifikation verwendet und mit anderen Erkennungsverfahren verglichen. In Abhängigkeit des Fonts waren zwischen $93,49\%$ und $99,83\%$ aller Scorelisten von der Form, daß das tatsächliche Symbol unter den ersten drei Kandidaten waren. Die Ergebnisse werden durch die Zahl der Zustände der Modelle beeinflußt, allerdings nicht im trivialen Sinn, daß eine höhere Zahl von Zuständen stets ein besseres Ergebnis garantiert. Die optimale Zahl der Zustände zeigte sich abhängig vom Font.

Zum Abschluß sei kurz die in [AB95] verwendete Folge von Beschreibungsmerkmalen angedeutet. Wir gehen exemplarisch auf drei der sechs dort verwendeten Merkmale und ihre Kodierungen ein.

1. *Zentroidales Profil.* Das Charakterbild wird in acht Oktanden aufgeteilt, wo die Trennlinien den ,,Himmelsrichtungen“ Nord, Nordost, Ost,..., Nordwest entsprechen. In jedem Oktanden werden nun zwei Kennlinien betrachtet. Die Linie, die den äußeren Umriss des Patterns wiedergibt, und die Linie, die den äußeren Umriss des kleinsten Rechtecks wiedergibt, das das Pattern umfasst. Für beide Kennlinien wird der durchschnittliche Abstand zum Bildmittelpunkt G berechnet. Aufgrund des Quotienten der beiden so erhaltenen Mittelwerte wird eine Kennziffer aus $\{0, 1, 2, 3\}$ vergeben. Zur Kennziffer wird noch eine Kennzahl 1, 2, 3 oder 4 in Abhängigkeit von der ,,Himmelsrichtung“ des Oktanden (Nord, Ost, Süd, West) hinzuaddiert. Dadurch ergeben sich Ziffernfolgen der Länge 8. Für das Symbol ,,l“, wo äußere Kontur und umgebendes Rechteck übereinstimmen, ergibt sich die Folge 44556677.

2. *Löcher.* Aufgrund der Zahl und Position der Löcher des Charakters wird eine Kennziffer zwischen 0 und 8 vergeben.

3. *vertikale Transition.* Durch den Bildmittelpunkt wird eine senkrechte Linie gezogen. Es wird berechnet, wie oft man auf dieser Linie einen Übergang von einem Bereich weißer Pixels in einen Bereich schwarzer Pixels hat. Dazu wird die Position der Übergänge mitberücksichtigt. Beide Informationen werden in einer Kennzahl zwischen 0 und 6 kodiert.

Erweiterung zur Multifont-Erkennung

Um das beschriebene System zur Multifont-Erkennung zu verwenden, wird für jedes Symbol jedes Fonts nach obigem Muster ein HMM eingeführt und trainiert. Die Multifonterkennung basiert auf der Hypothese, daß innerhalb eines Paragraphen — abgesehen von möglichen Fettdruck- und Kursivsymbolen — ein einheitlicher Font verwendet wird. Es wird daher paragraphenweise zunächst der (wahrscheinlichste) Font ermittelt, danach werden die zum Font gehörigen HMM wie oben beschrieben zur Klassifikation der Symbole des Paragraphen verwendet. Wir stellen nun noch die Prinzipien dar, mit denen der (wahrscheinlichste) Font eines Paragraphen ermittelt wird.

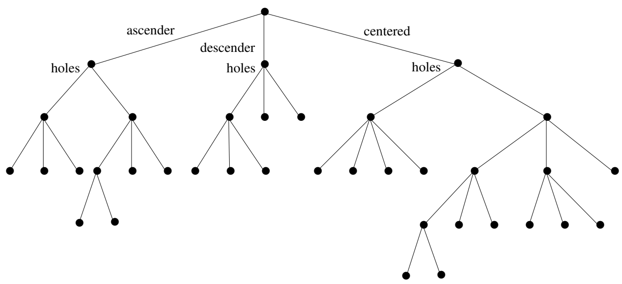


Abbildung 3.2: Generischer Font-Baum.

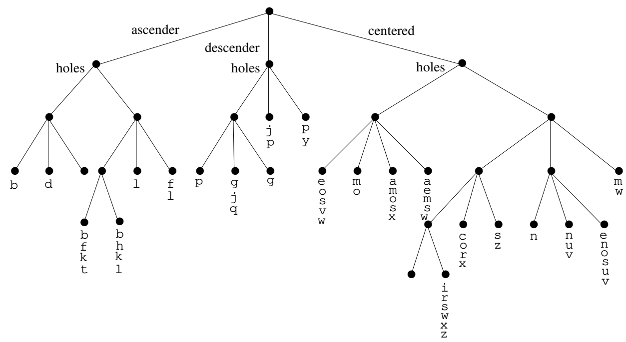


Abbildung 3.3: Instantiierung für Courier.

Konstruktion von Font-Bäumen

Ein Fontbaum ist ein Entscheidungsbaum, wo einige der bei der OCR-Erfassung verwendeten Features dazu verwendet werden, um das Binärbild des Vorkommens eines Symbols einem eindeutigen Blatt zuzuordnen. Symbol und Font sind bekannt und einem Vorrat an Trainingsdaten entnommen. Das Prinzip der Einteilung ist in Abbildung 3.2 dargestellt. Um das Symbol einem Blatt zuzuordnen, werden bei jedem Knoten bestimmte Features (wie das Vorkommen oder Nichtvorkommen eines Lochs) dazu verwendet, das eingehende Symbol einem Kinderknoten zuzuweisen.[2]

Bei der in [AB95] dargestellten Methode wird für jeden Font getrennt ein Baum berechnet, wo alle Symbolvorkommen des Fonts in den Trainingsdaten einsortiert werden. Jedes Vorkommen landet an einem Blatt des Baums. Es können verschiedene Vorkommen desselben Symbols an verschiedenen Blättern landen, da sich die Binärbilder und die Feature-Werte unterscheiden können. Bei jedem Blatt wird für jedes dort vorkommende Symbol ein Repräsentant berechnet, der alle bei dem Blatt eingeordneten Vorkommen dieses Symbols (dies können je nach Umfang der Trainingsdaten sehr viele sein) vertritt. Wenn man die einzelnen Vorkommen eines Symbols bei einem Blatt als reelle Vektoren repräsentiert, so kann man einfach den Schwerpunkt der Vektoren als Repräsentant nehmen. Auf diese Weise können wir beispielsweise für Courier den in Figur 3.3 dargestellten Baum erhalten, wo bei jedem Blatt jedem Symbol ein prototypischer Feature-Vektor zugeordnet ist.

[2] „Ascenders“ sind Buchstaben wie b, h oder k; descenders sind Buchstaben wie g und q, centered sind Buchstaben wie a,c,e.

Fonterkennung

Um den Font eines Paragraphen zu erkennen, werden die dort auftretenden Symbolvorkommen in denselben Entscheidungsbaum eingeordnet wie oben. Natürlich ist nun das Symbol selbst unbekannt, es führt aber zu einem eindeutig bestimmten Blatt des generischen Fontbaums. In den Bäumen der einzelnen Fonts können wir das entsprechende Blatt nehmen und das unbekannte Symbol mit allen dort auftretenden Prototypen vergleichen, indem wir die euklidische Distanz der Feature-Vektoren als Maßstab nehmen. Der kleinste so auftretende Distanzwert kann als ein Maß der Nähe des Symbols zum betreffenden Font (zu einem Symbol dieses Fonts) betrachtet werden. Wenn wir dieses Maß für alle Symbolvorkommen des Abschnitts addieren, erhalten wir ein Gesamtmaß für die Nähe des Fonts zu allen Symbolvorkommen des Paragraphen. Wir können den Font wählen, der in dieser Weise auf das kleinste Gesamtmaß führt.

Ergänzende Bemerkungen

Die obige Darstellung geht nicht auf alle Einzelheiten ein, die bei der Symbolklassifikation beachtenswert sind. Ergänzend sei die Möglichkeit erwähnt, die vorkommenden Symbole eines Dokuments zunächst mittels eines leicht zu berechnenden Ähnlichkeitsmaßes in Äquivalenzklassen einzuteilen, und danach nur noch für jede Äquivalenzklasse einen Prototypen zu berechnen und diesen zu klassifizieren. Mehr Details zu diesem und anderen Punkten sind in [AB95] zu finden.

3.2 Weitere Symbol-Klassifikationsverfahren

Grundform des Bayesschen Klassifikators. Es sollen $C_1, \ldots, C_M$ für die in Frage kommenden Klassen von Symbolen stehen, mit $\vec{X}$ sei ein Feature-Vektor bezeichnet. Der Bayessche Klassifikator in der einfachsten Grundform ignoriert bei Beobachtung von $\vec{X}$ alle kontextuelle Information (Klassen der vorausgegangenen Symbole etc.) und bildet $\vec{X}$ auf diejenige Klasse C_i ab, die die höchste „a posteriori" Wahrscheinlichkeit nach Beobachtung von $\vec{X}$ hat. Maximiert wird also die bedingte Wahrscheinlichkeit $P(C_i|\vec{X})$. Nach dem Bayesschen Gesetz gilt

$$P(C_i|\vec{X}) = \frac{P(\vec{X}|C_i)P(C_i)}{P(\vec{X})}$$

Da $P(\vec{X})$ für jeden Wert i gleich ist, ist also das Produkt $P(\vec{X}|C_i)P(C_i)$ zu maximieren. Abschätzungen für die Werte $P(\vec{X}|C_i)$ und $P(C_i)$ erhält man aus dem Trainingskorpus, wobei hier oft die Unabängigkeit der Komponenten des Feature-Vektors angenommen werden, um die Komplexität zu verringern. Mehr Informationen zu Bayeschen Klassifikatoren sind in [DH73] zu finden.

Nächster-Nachbar-Klassifikatoren. Hier wird zunächst mit Hilfe des Trainingskorpus für jedes Symbol eine prototypische Feature-Beschreibung erstellt. Des weiteren muß ein geeignetes Abstandsmaß (wie Hamming-Abstand oder euklidischer Abstand) auf dem Raum der Feature-Beschreibungen vorliegen. Ist ein Feature-Vektor $\vec{X}$ gegeben, so wird der nächste Nachbarn unter den Prototypen ermittelt und die entsprechende Klasse als Klassifikationsresultat ausgegeben. Werden Pixel-werte direkt als Features verwendet, so spricht man bei Verwendung eines nächsten Nachbarn Klassifikators meist von *Template Matching.*

Polynom-Klassifikatoren. Hier wird jeder Klasse C_i bei Beobachten von $\vec{X}$ ein Wert zugeordnet, der sich durch Einsetzen der Komponenten von $\vec{X}$ in ein Polynom $p(i)$ ergibt. Die Zahl der Unbestimmten der Polynome p entspricht dabei der Zahl der Komponenten der Feature-Vektoren. Die X zugeordnete Klasse richtet sich nach dem höchsten Wert der Polynome. Abgesehen von den Koeffizienten ist die Form der Polynome durch die Auswahl des Klassifikators festgelegt. Geeignete Koeffizienten werden durch Auswerten des Trainingskorpus gefunden. Details werden beispielweise in [KS97] beschrieben.

Bayessche Klassifikatoren, Nächster-Nachbar-Klassifikatoren und Polynomklassifikatoren werden manchmal als *statistische Klassifikatoren* bezeichnet. Einen Überblick über statistische Klassifikatoren, die bei der Charaktererkennung nützlich sind, bietet [Nag82].

Neuronale Netze. Hier werden mehrfache Schichten von untereinander verbundenen Knoten verwendet, um die Organisation von Neuronen in biologischen Systemen zu imitieren. Die Werte eines Eingabe-Feature-Vektors $\vec{X}$ werden entlang der Netzstruktur bis zur Ausgangsebene weiterpropagiert, wobei sie nach bestimmten Schemas gewichtet und addiert werden. Die Knoten der Ausgabeschicht entsprechen den möglichen Klassen, gewählt wird diejenige Klasse, deren Knoten den höchsten Eingangswert erhalt.

Weiter Klassifikatoren umfassen Baumklassifikatoren [Mor82, SS82, SD84, Kur83], syntaktische Klassifikatoren [ANT79, AP77, DHTW80, Fu82, PA75, PA79, Str78], die SVM-Methode (Support Vector Machines) [Bur] und Fuzzy-Techniken [Fu74, ea93].

3.3 Bibliographische Anmerkungen

Einen Überblick über Verfahren zur Charaktererkennung liefert [GS90]. Viele Verfahren sind auch in [KS97] dargestellt. Die wichtigsten Feature-Extraktionsmethoden bei der Klassifikation isolierter Charaktere sind in [TJT96] beschrieben.

Kapitel 4

Kombination von Klassifikatoren

Oft stehen bei der OCR-Erfassung mehrere Klassifikatoren zur Verfügung. Es hat sich in vielen Untersuchungen gezeigt, daß man durch geeignete Formen der *Kombination* der Klassifikatoren zu einem wesentlich verbesserten Erkennungsverhalten kommen kann. In diesem Abschnitt geben wir einen kurzen Überblick über mögliche Formen der Kombination und gehen auf einige spezielle Kombinationsverfahren näher ein.

4.1 Typen von Klassifikatoren

Um die möglichen Formen der Kombination darzustellen, ist es zunächst notwendig, Klassifikatoren nach der Art ihres Outputs zu unterscheiden.

1. Bei Klassifikation auf *abstraktem Level* wird dem Input Pattern aufgrund der vorliegenden Beschreibung *genau eine* der möglichen Klassen zugewiesen.
2. Bei Klassifikation auf *Rangordnungs-Level* wird dem Input Pattern eine in *Rangfolge geordnete Liste* möglicher Klassen zugewiesen,
3. Bei Klassifikation auf *Maßordnungs-Level* wird einem Input Pattern eine Liste möglicher Klassen zugewiesen, wobei für jede Klasse ein *Konfidenzwert* angegeben wird.

Für jede der drei Klassen von Klassifikatoren stehen unterschiedliche Kombinationsmethoden zur Verfügung.

4.2 Kombination auf abstraktem Level

Zur Kombination von Klassifikatoren auf abstraktem Level werden unterschiedliche Formen der *Abstimmung* (engl. ,,voting") verwendet.

Im einfachsten Fall verwendet man schlicht einen *Mehrheitsentscheid* (engl. „majority vote“). Dem Eingabepattern wird diejenige Klasse zugewiesen, die das meistgenannte Ergebnis der Komponentenklassifikatoren darstellt.

Feinere Methoden ergeben sich, wenn man die Ergebnisse der einzelnen Klassifikatoren unterschiedlich gewichtet. Man unterscheidet

- die Form, wo das Ergebnis jedes Klassifikators mit einem universellen (d.h. pattern-unabhängigen) *Gewicht* versehen wird. Anstelle des „Mehrheitsentscheids“, wo jeder Klassifikator mit demselben Gewicht 1 berücksichtigt wird, gehen nun die Entscheidungen der einzelnen Klassifikatoren in gewichteter Form ein, es gewinnt die Klasse mit dem größten Gesamtgewicht.
- weitergehenden Methoden, wo das Verhalten der Klassifikatoren auf den einzelnen Symbolen mitberücksichtigt wird.

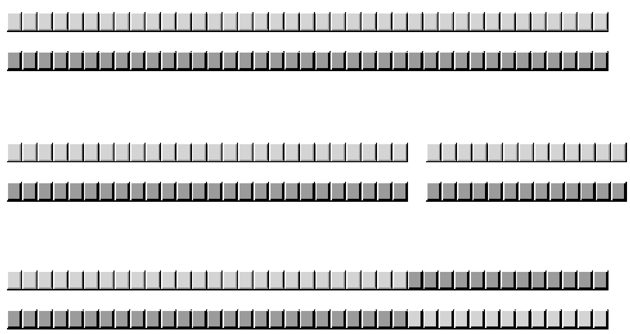
Nachfolgend drei Beispiele aus [LHS97], die die beiden Konzepte illustrieren.

Kombination mittels genetischer Algorithmen

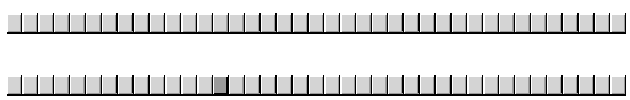
Wir beginnen mit einem Beispiel, wo jeder Klassifikator mit einem symbolunabhängigen Gewicht versehen wird.

Genetische Verfahren werden aus abstrakter Sicht in folgenden Situationen verwendet: Gegeben ist eine Menge von Objekten, für die ein Qualitätsmaß festgelegt und meßbar ist. Jedes Objekt ist in einer standardisierten Weise durch eine Zeichenfolge beschreibbar. Das Ziel ist es, durch die Simulation eines evolutionären Prozesses neue Objekte von höherer Qualität zu generieren, bis schließlich ein optimales oder fast optimales Objekt in der Population auftritt. Hierzu werden zwei biologische Prozesse simuliert.

1. Bei der „Kreuzung“ zweier Objekte werden die Zeichenfolgen beider Objekte an einer Stelle aufgetrennt, anschließend die zwei Endstücke vertauscht.



2. Bei der „Mutation“ wird eine Zeichenfolge an einer Stelle durch einen zufälligen Prozeß abgeändert.



Beim eigentlichen Erzeugungsprozeß beginnt man mit zunächst mit einer Anfangsmenge von Objekten, der ersten Generation. Durch Kreuzung und Mutationen der Elemente erhält man neue Elemente, aus denen man die Elemente der zweiten Generation auswählt. Qualitativ hochwertige Elemente der ersten Generation werden bevorzugt zum Kreuzen und bei der Mutation zugelassen. Auch bei der Auswahl der Elemente der zweiten Generation, die gegebenenfalls gemacht wird, um eine feste Populationsgröße zu erreichen, werden ,,gute“ Objekte bevorzugt.

Bei der Kombination von Klassifikatoren stellen die möglichen Gewichtungen der einzelnen Klassifikatoren die Objekte dar. Ziel ist es, eine möglichst gute Gewichtung zu finden. Um ein genetisches Verfahren anzuwenden, muß man zunächst das Qualitätsmaß präzisieren, dann eine geeignete Kodierung von Gewichtsverteilungen durch Zeichenketten. In [LHS97] werden ,,Auslasser“, in denen der Klassifikator keine Klasse bestimmt hat, durch einen Strafpunkt bewertet. OCR-Fehler, wo ein Symbolvorkommen falsch klassifiziert wurde, werden mit einer festen Zahl β von Strafpunkten belegt. Als mögliche Werte von β werden $10, 15, 20, 25$ und 30 betrachtet. Die Qualität einer Gewichtung wird durch Auswertung der Funktion

$$\begin{aligned} F &= \text{Anteil der korrekt erkannten Symbole} \\ &\quad -\beta \cdot \text{Anteil der falsch erkannten Symbole} \end{aligned}$$

auf einem fest vorgegebenen Trainingskorpus erfasst. Für $\beta = 10$ ist die Optimierung von F äquivalent dazu, die Funktion

$$\begin{aligned} &\text{Anteil der ausgelassenen Symbole} \\ &+11 \cdot \text{Anteil der falsch erkannten Symbole} \end{aligned}$$

zu minimieren.

Zur Kodierung einer Gewichtsverteilung wird jedem Klassifikator ein Gewicht zwischen 0 und 1023 zugewiesen, das durch eine Folge von 10 Bits dargestellt wird. Die Kodierung einer Gewichtung von K Klassifikatoren erfordert dann $K \cdot 10$ Bits.

Man beginnt mit einer Menge zufällig gewählter Gewichtungen. Wie oben dargestellt werden die betreffenden Strings nun einerseits gekreuzt, was darauf hinausläuft, die Gewichte für einen Teil der Klassifikatoren von einem ,,Chromosom“ zu übernehmen, die restlichen Gewichte vom anderen Chromosom. Zusätzlich werden noch Mutationen zugelassen, die spontan Gewichte verändern können. Im angegebenen Experiment umfassen die Populationen jeweils 50 Exemplare. Bei der Kreuzung werden Chromosomen, die auf der Testmenge zu einem guten kombinierten Erkennungsergebnis führen, bevorzugt: sie erhalten eine erhöhte Wahrscheinlichkeit bei dem Roulette-Prozeß, mit dem die Teilnehmer einer Kreuzungsrunde ausgesucht werden. Bei den ausgesuchten Teilnehmern ist die Wahrscheinlichkeit einer Kreuzung auf $0,9$ gesetzt, die Wahrscheinlichkeit einer Mutation liegt bei $0,05$. Bei dem in [LHS97] beschriebenen Experiment waren die Qualitätsunterschiede der einzelnen Chromosomen nach einigen Kreuzungsrunden sehr gering, so daß die Fitnesswerte, die sich durch die Funktion F ergeben, nach einem bestimmten Schema linear skaliert werden.

Bayessches Kombinationsverfahren

Das Bayessche Kombinationsverfahren erlaubt eine symbolabhängige Gewichtung der einzelnen Klassifikatoren. Nachfolgend seien $C_1, \ldots, C_M$ die möglichen Klassen, es seien K Klassifikatoren gegeben. Wir berechnen auf der Basis des Trainingskorpus für jeden Klassifikator seine Konfusions-Matrix KM. In $KM[i,j]$ ist gespeichert, wie oft der Experte bei Vorliegen eines Symbols der Klasse C_i die Klasse C_j zugewiesen hat. Da wir auch die Möglichkeit berücksichtigen, daß der Klassifikator die Eingabe verwirft, ihr also keine Klasse zuordnet, hat die Konfusionsmatrix die Dimension $M \times (M+1)$.

echte Klasse \ zugewiesene Klasse	C_1	...	C_j	...	C_{M+1}
C_1					
...					
C_i			KM[i,j]		
...					
C_M					

Die Gesamtzahl der Trainingsbeispiele aus Klasse C_i ist dann die Zeilensumme $\sum_{j=1}^{M+1} KM[i,j]$, während die Spaltensummen $\sum_{i=1}^{M} KM[i,j]$ die Gesamtzahl der Symbolvorkommen angibt, die der Klassifikator auf Klasse C_j abgebildet hat. Insgesamt haben wir K Konfusionsmatrizen $KM^{(k)}$ $(1 \leq k \leq K)$, die den K Klassifikatoren entsprechen. Die bedingte Wahrscheinlichkeit, daß ein Pattern zur Klasse C_i gehört, wenn Klassifikator k es auf die Klasse C_j abbildet, kann experimentell durch

$$P(x \in C_i | e_k(x) = j) = \frac{KM^{(k)}[i,j]}{\sum_{l=1}^{M} KM^{(k)}[l,j]}$$

abgeschätzt werden. Hierbei stellt $e_k(x) = j$ das Ereignis dar, daß Klassifikator k das Symbolvorkommen x der Klasse C_j zuordnet. Wir interessieren uns bei der aktuellen Klassifikation für die bedingte Wahrscheinlichkeit des Vorliegens der Klasse C_i, unter der Voraussetzung, daß Klassifikator k die Klasse C_{j_k} zuweist $(1 \leq k \leq K)$, also für

$$P(x \in C_i \mid e_1(x) = j_1, \ldots, e_K(x) = j_K).$$

Diese Formel wird in [LHS97] bzw. [XKS92] durch den Ausdruck

$$\frac{\prod_{k=1}^{K} P(x \in C_i | e_k(x) = j_k)}{\sum_{l=1}^{M} \prod_{k=1}^{K} P(x \in C_l | e_k(x) = j_k)}$$

approximiert, wobei vereinfachend Unabhängigkeit der Klassifikatorergebnisse angenommen wird. Der Nenner stellt hierbei einen Normierungsfaktor dar. Als Ergebnis der kombinierten Klassifikation erhält man diejenige Klasse C_i, für die sich der höchste Wert ergibt, wobei x auch verworfen werden kann, falls keiner der Werte einen kritischen Grenzwert α übersteigt. Die Wahl des Grenzwerts α legt einen Kompromiß zwischen einer möglichst hohen Erkennungsrate und einer möglichst hohen Fehlerfreiheit fest.

Die BKS-Methode

Die ,,Behaviour-Knowledge Space“ (BKS) Methode geht insofern über die vorausgegangenen Techniken hinaus, als daß sie die gegenseitige Abhängigkeit der Ergebnisse der einzelnen Klassifikatoren berücksichtigt. Hierzu wird eine spezielle Datenstruktur eingeführt. Im Fall von K Klassifikatoren und M Klassen ist dies eine K-dimensionale Array-Struktur *BKS* mit der Ausdehnung $M+1$ in jede Richtung. Ist x ein Pattern, und sind $e_1(x) = j_1, \ldots, e_K(x) = j_K$ die Klassifikationsresultate der K Einzelklassifikatoren, so wird in Zelle $BKS[j_1, \ldots, j_K]$ eine Information abgelegt. In jeder Zelle $BKS[j_1, \ldots, j_K]$ wird die Zahl $N_m[j_1, \ldots, j_K]$ der eingehenden Samples der Klasse C_m gespeichert, für alle $1 \leq m \leq M$. Hieraus ergibt sich die Gesamtzahl $T[j_1, \ldots, j_K]$ aller dort eingehenden Samples. Mit $opt[j_1, \ldots, j_K]$ bezeichen wir denjenigen Index m, für den $N_m[j_1, \ldots, j_K]$ den maximalen Wert einnimmt.

Beispiel 4.2.1 Bei der Kombination von zwei Klassifikatoren und 4 möglichen Klassen kann nach Eingabe einer Folge von Samples mit bekannter Klassenzugehörigkeit folgendes *BKS* entstanden sein ($\sim$ repräsentiert die Situation, wo der Klassifikator dem Pattern aufgrund mangelnder Sicherheit keine Klasse zuweist).

$e_1 \setminus e_2$	1	2	3	4	$\sim$
1	...	...	...	...	...
2	...	...	$\langle 2, 5, 19, 1 \rangle$	...	...
3	...	...	...	...	...
4	...	...	...	...	...
$\sim$	...	...	...	...	...

Der hervorgehobene Eintrag $BKS[2, 3]$ bedeutet, daß es bislang insgesamt $27 = 2+5+19+1$ Samples gab, die von Klassifikator 1 der Klasse $i = 2$, von Klassifikator 2 der Klasse $j = 3$ zugeordnet wurden. Unter diesen befinden sich 2 Samples der Klasse C_1, 5 Samples der Klasse C_2, 19 Samples der Klasse C_3 und 1 Sample der Klasse C_4. Das Trainingsergebnis deutet darauf hin, daß bei einer Klassifikation eines neuen Samples x mit dem Resultat $e_1(x) = 2, e_(x) = 3$ eine Wahrscheinlichkeit von $19/27 = 19/(2+5+19+1)$ besteht, daß x zur Klasse C_3 gehört. Für alle anderen Klassen ist die entsprechende Wahrscheinlichkeit niedriger. Es sollte daher C_3 als kombiniertes Entscheidungsresultat gewählt werden.

Die semantische Bedeutung der Einträge sollte aus dem Beispiel klar sein. In der Praxis arbeitet man bei der BKS-Methode mit zwei Schritten. Zunächst wird mit Hilfe der Trainingsdaten, wo eine größere Menge von Samples bekannter Klassenzugehörigkeit verwendet wird, ein BKS aufgebaut. Bei der Klassifikation eines neuen Samples x werden das Array *BKS* und die Ergebnisse $e_1(x) = j_1, \ldots, e_K(x) = j_K$ der Einzelklassifikatoren verwendet. Das Ergebnis der kombinierten Klassifikation ist dann $m := opt[j_1, \ldots, j_K]$, unter der Voraussetzung, daß der Anteil $N_m[j_1, \ldots, j_K]/T[j_1, \ldots, j_K]$ einen bestimmten Grenzwert α überschreitet. Andernfalls wird die Eingabe verworfen.

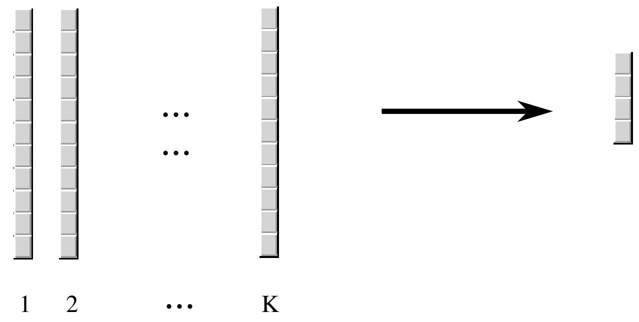
Das Entscheidungsverfahren ist verwandt zum Bayesschen Kombinationsprinzip, da die Klasse gewählt wird, die die höchste Wahrscheinlichkeit ergibt. Verwendet zur Beurteilung dieser Wahrscheinlichkeit wird allerdings nicht der vorliegende Feature-Vektor des Samples, sondern lediglich das Verhalten der Einzelklassifikatoren.

4.3 Kombination auf Rangfolgen-Level

Kombination auf Rangordnungslevel wird ausführlich in [Ho92] studiert. Die Einzelklassifikatoren liefern hier bei Eingabe eines Patterns Ranglisten der möglichen Klassen, ohne daß diese (notwendigerweise) mit zusätzlichen numerischen Werten versehen sind. Als Grundmethoden lassen sich die ,,Klassenmengen-Reduktion“ (engl. class set reduction) und die ,,Klassenmengen-Neuordnung“ unterscheiden.

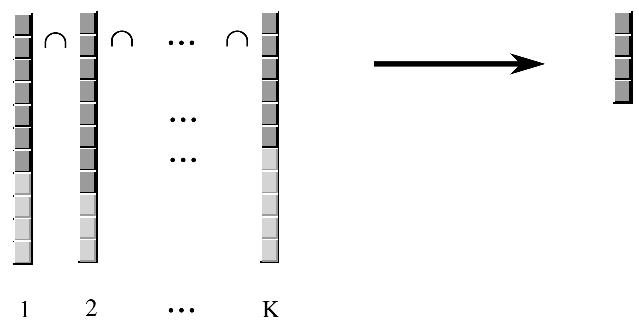
Klassenmengen-Reduktion

Hier dient die Kombination dazu, bei Eingabe eines Patterns mit Hilfe der Komponentenklassifikatoren eine möglichst kleine Menge von Kandidatenklassen auszugeben, die mit Sicherheit die richtige Klasse des Symbols enthält.



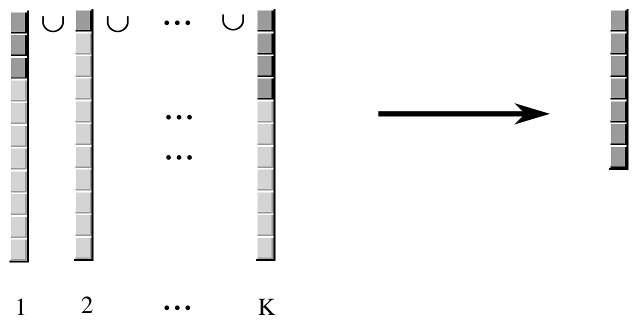
In [Ho92] werden zwei mögliche Vorgehensweisen genannt.

Durchschnittsverfahren. Man sortiert von den Ranglisten der einzelnen Klassifikatoren alle Klassen oberhalb einer gewissen Schranke aus. Sie bilden die Kandidatenliste des jeweiligen Einzelklassifikators. Die jeweiligen Schranken sollten so gewählt sein, daß die korrekte Klasse mit Sicherheit in der Kandidatenliste des Klassifikators ist. Die Ausgabeliste ergibt sich durch Durchschnittsbildung über die Kandidatenlisten der Einzelklassifikatoren.



Die Wahl der Schranken ergibt sich aus dem Trainingskorpus. Komponentenklassifikatoren, bei denen die Kandidatenliste jeweils die gesamte Menge der möglichen Klassen umfasst, sind redundant und werden weggelassen.

Vereinigungsverfahren. Man sortiert wieder von den Ranglisten der einzelnen Klassifikatoren alle Klassen oberhalb einer gewissen Schranke aus. Allerdings ist nun die Schranke so hoch, daß sich nur sehr kleine Kandidatenlisten ergeben. Die Ausgabeliste ergibt sich durch Vereinigung über die Kandidatenlisten der Einzelklassifikatoren.



Geeignete Schranken ergeben sich aus dem Trainingsset wie folgt. Es bezeichne $rg_j^i \in \{1, \ldots, M\}$ der Rang, den die wahre Klasse des i-ten Trainingssamples durch den Klassifikator j erhält.

Um das i-te Sample in geeigneter Weise bei der Rangfolgenvereinigung zu erfassen, gehen wir wie folgt vor. Da wir letztendlich die oberen Abschnitte der einzelnen Rangfolgen zusammenfassen, reicht es, einen einzigen Klassifikator auszusuchen, und hier den oberen Teil der Rangfolge des Samples i bis zur korrekten Klasse, also den durch rg_j^i beschriebenen Anfangsteil zu übernehmen. Es ist offenkundig geschickt, einen Klassifikator j zu nehmen, bei dem der entsprechende Anfangsabschnitt bis so klein wie möglich ist. Diesen nennen wir den *für Sample i günstigsten Klassifikator*. Ist $j(i)$ der für Sample i günstigste Klassifikator, so heißt $mrg_{j(i)}^i$ der *bei Sample i zu berücksichtigende Rang des Klassifikators* $j(i)$. Für alle anderen Klassifikatoren ist der bei Sample i zu berücksichtigende Rang gleich 0.

Wir gehen nun alle Patterns durch, bestimmen jeweils den günstigsten Klassifikator und den jeweils zu berücksichtigenden Rang. Tritt ein Klassifikator mehrfach als günstigster Klassifikator auf, so müssen wir im Endeffekt den *größten* dabei lokal zu berücksichtigenden Rang zur Bestimmung des Beitrags des Klassifikators bei der Vereinigung nehmen. Ihn nennen wir den *maximalen relevanten Rang* des Klassifikators j. Wir nehmen nun bei der Kombination von der Rangfolge des Klassifikators j stets alle Klassen mit Rang kleinergleich dem maximalen relevante Rang. Es ist einfach zu zeigen, daß auf diese Weise die Vereinigungsmethode zumindest auf dem Trainingsset stets die korrekte Klasse enthält.

Beispiel 4.3.1 Wir erläutern die Begriffe mit einem Beispiel aus [Ho92]. Durch Eingabe der Samples $S_1, \ldots S_6$ sollen sich für die beteiligten Klassifikatoren mit Nummern 1–4 folgende Ränge ergeben.

	1	2	3	4
S_1	3	12	1	24
S_2	1	5	29	12
S_2	34	3	4	6
S_4	9	7	6	7
S_5	4	36	5	5
S_6	16	2	3	4

Durch Berücksichtigung des kleinsten Rangs jeder Zeile ergeben sich dann die folgenden lokal zu berücksichtigenden Ränge.

	1	2	3	4
S_1	0	0	1	0
S_2	1	0	0	0
S_2	0	3	0	0
S_4	0	0	6	0
S_5	4	0	0	0
S_6	0	2	0	0

Die maximalen relevante Ränge der vier Klassifikatoren sind demnach 4, 3, 6 respektive 0.

Als Spezialfälle kann man die Situation betrachten, wo stets bei einem der Klassifikatoren die oberste Klasse die korrekte ist. In diesem Fall sind die lokal zu berücksichtigenden Ränge stets 0 oder 1 und man braucht nur die obersten Klassen zu vereinigen. Ein anderer Spezialfall liegt vor, wenn wir nur einen Klassifikator haben. In diesem Fall ist der maximale relevante Rang gerade der schlechteste Rang, der der wahren Klasse eines Samples zugeordnet wurde. Klassifikatoren, deren maximaler relevanter Rang 0 ist, sind offenkundig redundant. Im obigen Beispiel träfe dies auf Klassifikator 4 zu.

Klassenmengen-Neuordnung

Hier dient die Kombination dazu, bei Eingabe eines Patterns mit Hilfe der Rangordnungen der Komponentenklassifikatoren eine kombinierte Rangordnung auszugeben, wo die richtige Klasse des Symbols stets möglichst weit oben ist.

Methode des höchsten Rangs. Hierbei erhält bei der Kombination jede Klasse k den höchsten Rang, den sie unter den Rangordnungen der Einzelklassifikatoren erhält. Wenn durch diese Vorschrift zwei Klassen denselben Rang erhalten, werden sie in willkürlicher Weise untereinander angeordnet. Die Methode ist vor allem bei einer großen Zahl von Klassen und wenigen Klassifikatoren sinnvoll.

Die Borda Zählmethode. Hierbei wird bei der Klassifikation eines Patterns einer Klasse diejenige Zahl zugeordnet, die sich dadurch ergibt, daß man die Zahl der Klassen mit schlechterem Rang für jeden Einzelklassifikator ermittelt und diese Zahlen aufsummiert. Die betreffende Zahl wird der Borda-Count genannt [Bla58]. Zum Abschluß werden die Klassen nach absteigendem Borda-Rang geordnet.

In [Ho92] wird ein verfeinertes Verfahren angegeben, bei dem die Borda-Zählmethode durch Gewichtung der Einzelklassifikatoren ergänzt wird.

4.4 Kombination auf Maßordnungs-Level

Als ein Beispiel der Kombination auf Maßordnungs-Level beschreiben wir die Verwendung von neuronalen Netzen. Zunächst beschreiben wir kurz die Struktur und Arbeitsweise von neuronalen Netzen.

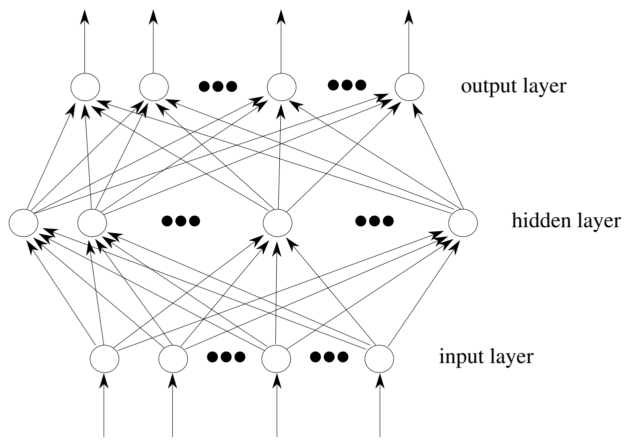


Abbildung 4.1: Neuronales Netz mit drei Schichten.

Neuronale Netze

Ein neuronales Netz — in der konventionellen Architektur — besteht aus einer Menge von Knoten, die in einer geordneten Folge von Schichten angeordnet sind, wobei die Knoten aufeinanderfolgender Schichten miteinander verbunden sind. Die erste Schicht wird als *Eingabeschicht* ausgezeichnet, die letzte Schicht als *Ausgabeschicht*. Jeder Knoten der Eingabeschicht erhält bei einem Klassifikationsvorgang ein Inputsignal einer bestimmten Höhe. Die Inputsignale der nachfolgenden Schichten ergeben sich als Summe der gewichteten Ausgaben der Knoten der Vorgängerschicht. Genauer hat das Eingabesignal eines Knotens i die Form $\sum_j w_{i,j}O(j)$ wo j über alle Knoten der Vorgängerschicht läuft, $w_{i,j}$ ein Gewicht ist, und wo $O(j)$ das Ausgabesignal des Vorgängerknotens j ist. Das Ausgabesignal eines Knotens wird jeweils aus den Eingabesignalen durch eine nichtlineare vorgegebene Aktivierungsfunktion f ermittelt. Die Zahl der Eingabeknoten entspricht der Zahl der gemessenen Features bei der Erkennung eines Patterns, die Zahl der Ausgabeknoten entspricht der Anzahl K der Klassen. Erhalten nun die Eingabeknoten bei der Verarbeitung eines Patters ein Signal, so wird dieses durch das Netz propagiert und führt zu Ausgabesignalen an allen Ausgabeknoten. Soll das Ergebnis der Klassifikation eine eindeutig bestimmte Klasse sein, so richtet sich die Entscheidung nach dem Ausgabeknoten mit dem höchsten Ausgabewert. Gegebenenfalls kann man auch eine Schranke α einführen, und nur dann eine Entscheidung treffen, wenn der Abstand des höchsten Ausgabewerts zum nächstniedrigeren Wert die Schranke α übersteigt. Alternativ kann man auch eine geordnete Rangfolge ausgeben, oder aber jeder Klasse aufgrund des Wertes des Ausgabeknotens einen Score zuordnen.

Beispiel 4.4.1 Abbildung 4.1 zeigt ein neuronales Netz mit drei Schichten.

Beim Trainieren eines Netzes werden die Trainingssamples dazu verwendet, um optimale Gewichte $w_{i,j}$ zu bestimmen. Hierzu existieren verschiedene Verfahren, auf die wir hier nicht näher eingehen.

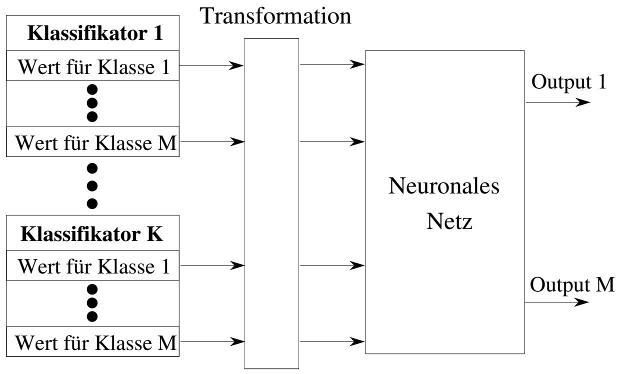


Abbildung 4.2: Kombination auf Maßordnungslevel mit neuronalen Netzen.

Kombination von Klassifikatoren mit neuronalen Netzen

Bei der Kombination von Klassifikatoren des Maßordnungslevels mit Hilfe von neuronalen Netzen werden die Ausgabewerte $s(i,j)$, die Klassifikator i einem Pattern als Konfidenzwert der Zugehörigkeit zur Klasse j zuordnet, als Eingabefeatures betrachtet. Man verwendet also ein Netz mit $K \times M$ Eingabeknoten. Im Prinzip kann man die Werte $s(i,j)$ direkt als Eingabewerte für die Knoten des Input Layers verwenden. Allerdings erhält man hier in der Regel nur dürftige Resultate, da die Werte, die von unterschiedlichen Klassifikatoren geliefert werden, sehr unterschiedlich in Bezug auf ihre Bedeutung (z.B. Abstand, Konfidenz, Ähnlichkeit, Unähnlichkeit) und Größe (z.B. $0-1$, $100-10.000$) sein können. Daher führt man in der Regel mit den Ausgabewerten $s(i,j)$ zunächst eine Datentransformation durch, mit dem Ziel, die Werte dem Vorzeichen und der Größe nach vergleichbar zu machen (cf. [HLS95]). Üblich ist es, alle Werte in eine Form ,,Wahrscheinlichkeit" zu transformieren, deren Werte zwischen Null und Eins liegen. Die Gesamtarchitektur der Kombination ist in Abbildung 4.2 dargestellt.

4.5 Ergänzende Bemerkungen

Weitere Methoden der Kombination auf abstraktem Level oder auf Maßordnungslevel sind in [XKS92] zu finden. Weitere Arbeiten zur Kombination sind [HHS92, HHS94, MS88, HS93, LS95]. Weitere Literaturhinweise sind in [LHS97, DHH$^+$97] angegeben. Die oben dargestellten Methoden unterscheiden sich zum Teil wesentlich bezüglich des Trainingsaufwands und des Potentials zur Verbesserung der Ergebnisse der Einzelklassifikatoren. Hinweise hierzu finden sich in der angegebenen Literatur.

Kapitel 5

Fehlerarten und Modelle zur Nachkorrektur

In diesem Abschnitt geben wir zunächst einen kurzen Überblick über die Art und Häufigkeit der bei der OCR auftretenden Fehler. Nachfolgend beschreiben wir verschiedene Korrekturmodelle zur automatischen und interaktiven Nachkorrektur. Wir bestimmen verschiedene Parameter, mit der die Qualität von Nachkorrekturverfahren gemessen werden kann. Die in nachkorrigierten Texten auftretenden Fehler werden in systematischer Weise klassifiziert.

5.1 Fehlerverteilung bei der OCR-Erkennung

In [LLNT96] wird von einer Evaluierung berichtet, wo sich folgende Fehlerverteilung bei der OCR-Erkennung einer Times1 Datenmenge ergab. Die erste Tabelle gibt Aufschluss über die Gesamtzahl von Fehlern eines bestimmten Typs. Hierbei wurden beispielsweise auch $2 \rightarrow 4$ Operationen (wie "nn $\rightarrow$ IlIl") erfasst.

	$\rightarrow 0$	$\rightarrow 1$	$\rightarrow 2$	$\rightarrow 3$	$\rightarrow 4$
$0 \rightarrow$	–	619	1	1	8
$1 \rightarrow$	245	847	63	2	2
$2 \rightarrow$	0	724	270	25	7
$3 \rightarrow$	0	4	7	8	2
$4 \rightarrow$	1	0	0	3	2

Die zweite Tabelle gibt die Zahl der vorkommenden Fehlertypen an. Es fällt auf, dass die Zahl aufgetretener Fehlertypen teilweise sehr viel niedriger ist als die Geamtzahl der Fehlertoken. Dies macht deutlich, daß Fehler wie $2 \rightarrow 1$-merges nur für wenige Symbole vorkommen.

	$\rightarrow 0$	$\rightarrow 1$	$\rightarrow 2$	$\rightarrow 3$	$\rightarrow 4$
$0 \rightarrow$	–	6	1	1	3
$1 \rightarrow$	26	72	21	1	1
$2 \rightarrow$	0	33	44	15	4
$3 \rightarrow$	0	3	7	8	2
$4 \rightarrow$	1	0	0	2	1

5.2 Modelle und Qualitätsmaße zur Nachkorrektur

5.3 Fehlerarten bei nachkorrigierten Texten

Kapitel 6

Klassische Verfahren zur Korrektur im Wortkontext

Wenn man die von der Charakterklassifikation gelieferten Symbole ungeprüft übernimmt, so ergeben sich aus Fehlerraten von 4% bis 25% bei der Erkennung einzelner Symbole sofort signifikant erhöhte Fehlerraten von 21% bis 86% bei der Worterkennung. Ein genau umgekehrtes Verhältnis beobachtet man interessanterweise beim menschlichen Erkennungsprozeß. Auch bei Schrifttypen und -Größen, wo Menschen bei der Erkennung isolierter Symbole eine signifikante Fehlerrate aufweisen, werden ganze Wörter in aller Regel korrekt erkannt. Dies macht deutlich, daß sich gute Methoden zur Erfassung des Wortkontexts in einer dramatischen Verbesserung der Erkennungsrate niederschlagen. In diesem Abschnitt gehen wir auf einige klassische Verfahren ein, wie die Ergebnisse einer charakterbasierten Erkennung durch die Erfassung des Wortkontexts überprüft und gegebenenfalls korrigiert werden können. Der weitergehende Satz- oder Dokumentenkontext wird hierbei nicht berücksichtigt.

Korrekturverfahren, die den Buchstaben- und Wortkontext mitberücksichtigen, sind in moderne OCR-Systeme bereits integriert. Dadurch ergibt sich bezüglich der Symbol- bzw. Worterkennungsraten eine signifikate Verbesserung im Vergleich zu den oben genannten Zahlen. Die in diesem Kapitel dargestellten Verfahren, wurden vorwiegend in den 70er und 80er Jahren entwickelt. Durch die damalige mangelnde Effizienz beim Zugriff auf große Lexika ergab sich ein Konflikt zwischen lexikalischen Verfahren (langsam) und rein statistischen Verfahren (schnell aber unsicher). Mittels gemischter "Hybrid"-Verfahren versuchte man die Nachteile beider Ansätze auszugleichen

In späteren Arbeiten wurde der effiziente Zugriffs auf elektronische Lexika erheblich verbessert. Insbesondere wurden Methoden entwickelt, die Suche nach Korrekturvorschlägen nur auf Teile des Lexikons einzuschränken. Diese moderneren Verfahren werden in späteren Abschnitten dargestellt.

6.1 Lexikalische Korrekturverfahren

Bei lexikalischen Verfahren, die auch "Top-Down-Verfahren" genannt werden, geht man davon aus, daß ein vorhandenes Lexikon L die Menge aller zulässigen Wörter abdeckt. Ziel ist es, zu einem möglicherweise fehlerhaft erkannten Eingabewort das korrekte Ursprungswort im Text zu rekonstruieren, oder aber eine Liste guter Korrekturvorschläge zu machen.

6.1.1 Verwendung der Erkennungsscores der Klassifikation

Eines der ältesten Verfahren zum Abgleich eines OCR-Ergebnisses mit einem Wörterbuch geht auf Bledsoe und Browning [BB59] zurück. Wir erläutern die Methode anhand eines Beispiels. Angenommen, das tatsächlich eingegebene Wort sei HUT. Wir nehmen an, daß das Klassifikationsverfahren drei Feature-Vektoren X_1, X_2 und X_3 extrahiert und auf dieser Grundlage für jedes in Frage kommende Symbol des Alphabets drei korrespondierende Diskriminantenwerte (Erkennungsscores) ermittelt:

Eingabewort:	H	U	T
Feature-Vektoren:	X_1	X_2	X_3
Diskriminantwerte für A:	$g_A(X_1)$	$g_A(X_2)$	$g_A(X_3)$
Diskriminantwerte für B:	$g_B(X_1)$	$g_B(X_2)$	$g_B(X_3)$
...	...	...	...
Diskriminantwerte für Z:	$g_Z(X_1)$	$g_Z(X_2)$	$g_Z(X_3)$

Das Wörterbuch ist in Klassen von Wörtern gleicher Länge partitioniert. Für jedes Wort mit derselben Länge wie das Eingabewort wird nun ein Score berechnet:

Score von ,,AAL“:	$g_A(X_1)$	$+g_A(X_2)$	$+g_L(X_3)$
Score von ,,AAS“:	$g_A(X_1)$	$+g_A(X_2)$	$+g_S(X_3)$
...	...	...	...
Score von ,,ZUM“:	$g_Z(X_1)$	$+g_U(X_2)$	$+g_M(X_3)$

Als Resultat wird dann das Lexikonwort mit den höchsten Score gewählt. Man kann das Verfahren als ein ,,Maximum-Likelihood-Verfahren“ charakterisieren, bei dem als Nebenbedingung nur Lexikonwörter in Betracht gezogen werden. Allerdings wird hierbei — anders als bei den nachfolgend beschriebenen Verfahren — die Wahrscheinlichkeit der Lexikonwörter selbst nicht mitberücksichtigt.

Probleme. Die Qualität der so erzielten Korrekturergebnisse ist im allgemeinen gut, wenn auch aufgrund der *mangelnden Berücksichtigung von Worthäufigkeiten* nicht optimal. Ein weiterer Nachteil, den dieses Verfahren mit den nachfolgend beschriebenen teilt, besteht darin, dass mögliche *Einfügungen, Löschungen, Verschmelzungen und Splits* von Symbolen bei der OCR-Erkennung nicht berücksichtigt sind. Ein schwerwiegendes Problem bei diesem Verfahren besteht in der *schlechten Effizienz*, die sich dadurch ergibt, daß für alle Lexikonwörter einer festen Länge der Score berechnet wird.

6.1.2 Verwendung von Konfusionswahrscheinlichkeiten und Worthäufigkeiten

Das nun zu beschreibende Verfahren setzt nicht voraus, daß man als Klassifikationsergebnis eines Symbols einen Erkennungsscore für die in Frage kommenden Erkennungsresultate erhält. Die Klassifikation hat als Ergebnis lediglich ein Symbol.

Zur lexikalischen Korrektur verwendet werden statistisch ermittelte *Konfusionswahrscheinlichkeiten.* Diese geben die Wahrscheinlichkeit an, daß ein bestimmter Buchstabe mit einem anderen verwechselt wird. Sie hängen von der Art der Textquelle, von den verwendeten Fonts, von der OCR-Erfassung ab. Sie fallen unter den Begriff der „channel-characteristics". Um Konfusionswahrscheinlichkeiten zu ermitteln, kann man das Erfassungstool mit einer großen Textmenge füttern und eine Fehlerstatistik ermitteln.

Nachfolgend wird davon ausgegangen, daß man die Wahrscheinlichkeit des Auftretens eines Lexikonwortes Z durch die Ermittlung der Vorkommenhäufigkeit in einem ausreichend großen Korpus in akzeptabler Weise annähernd bestimmen kann. Die Berücksichtigung dieser Wahrscheinlichkeit stellt – neben der verbesserten Effizienz, die sich durch den fehlenden Lexikon-Abgleich ergibt — ein wesentliches Unterscheidungsmerkmal zur Methode von Bledsoe und Browning dar.

Sei $X = x_1 x_2 \dots x_n$ das von der OCR gelieferte Wort. Die Wahrscheinlichkeit, das eine Buchstabenfolge $Z = z_1 z_2 \dots z_n$ die Ausgabe X verursacht hat, kann mit dem Bayesschen Gesetz wie folgt abgeschätzt werden.

$$P(Z|X) = \frac{P(X|Z)P(Z)}{P(X)}.$$

Hierbei ist $P(X|Z)$ die bedingte Wahrscheinlichkeit, X als OCR-Ergebnis zu erhalten, wenn Z die wirkliche Eingabe ist, $P(Z)$ ist die a priori Wahrscheinlichkeit für Z und $P(X)$ ist die Wahrscheinlichkeit von X. Da X feststeht, muß

$$P(X|Z)P(Z)$$

maximiert werden. Um alle Wahrscheinlichkeiten $P(X|Z)$ abzuschätzen und zu speichern, wäre ein astronomischer Aufwand notwendig. Wenn man annimmt, daß die Wahrscheinlichkeit der Erkennung eines Symbols x bei tatsächlichem Vorliegen eines Symbols z nicht von den Symbolen der Umgebung beeinflußt wird — was insbesondere bei Kursivschriften durchaus eine stark vereinfachende Annahme darstellt — so gilt

$$\begin{aligned} P(X|Z) &= P(x_1 \cdots x_n | z_1 \cdots z_n) \\ &= \Pi_{i=1}^{n} P(x_i|z_i) \end{aligned}$$

Hieraus ergibt sich als zu maximierender Ausdruck nun

$$P(Z) \ \cdot \ \Pi_{i=1}^{n} P(x_i|z_i).$$

Wir können den Ausdruck maximieren, wenn wir die Wahrscheinlichkeit des Wortes Z, die vom Erkenner abhängigen Konfusionswahrscheinlichkeiten $P(x_i|z_i)$ kennen. Alternativ kann man durch die Unformung $P(x_i|z_i) = P(z_i|x_i)\frac{P(x_i)}{P(z_i)}$ und Weglassen der fixierten Faktoren $P(x_i)$ auch

$$P(Z) \ \cdot \ \Pi_{i=1}^{n} \frac{P(z_i|x_i)}{P(z_i)}$$

optimieren, was geschickter sein kann, wenn Abschätzungen für die Faktoren $P(z_i|x_i)$ direkt vom OCR-Erkennungsprozeß mitgeliefert werden. Zur Verwendung des Verfahrens vergleiche man [DH68, SH74].

Probleme/Vergleich. Das Verfahren berücksichtigt Worthäufigkeiten bei der Korrektur. Im Hinblick auf die *Effizienz* besteht das Problem, dass alle Wörter des Lexikons einer festen Länge zu durchlaufen sind. Das Verfahren berücksichtigt nicht mögliche *Einfügungen, Löschungen, Verschmelzungen und Splits* von Symbolen bei der OCR-Erkennung.

6.1.3 Generelle Schwierigkeiten einer lexikalischen Korrektur

Beim Einsatz von Lexika zur Nachkorrektur von OCR-Ergebnissen ergeben sich neben den oben erwähnten Problemen eine Reihe weiterer Schwierigkeiten. Einige zentrale Probleme werden nachfolgend aufgelistet. Vorschläge zu deren Lösung werden in den nachfolgenden Kapiteln erörtert.

Problem kleiner Wörterbücher. Werden kleine Lexika verwendet, so ergeben sich offenkundige Nachteile: es werden viele Wörter fälschlich als Schreibfehler betrachtet, da sie nicht im Lexikon auftreten. Die "Nachkorrektur" kann damit das OCR-Ergebnis verschlechtern.

Problem großer Wörterbücher. Verwendet man sehr große Wörterbücher, so führt dies zu zwei verwandten Problemen. Es kann eine hohe Zahl sogenannter "false friends" auftreten. Hierunter versteht falsch erkannte Strings, die nur durch Zufall wieder korrekte Lexikoneinträge darstellen. False friends treten sehr oft bei kurzen Wörtern auf, vor allem, wenn auch Abkürzungen im Lexikon stehen. Ein anderer Nachteil großer Lexika besteht darin, daß die Listen möglicher Korrekturvorschläge u.U. sehr groß werden und mit seltenen Wörtern überschwemmt sind. Bei der Verwendung großer Wörterbücher ist es daher notwendig, die Einträge mit Informationen zu Vorkommenhäufigkeiten anzureichern, die bei der Korrektur und beim Ranking berücksichtigt werden. Festzuhalten ist auch, daß selbst die größten Lexika nie alle Wörter enthalten, da in jeder Sprache ständig neue Wörter hinzukommen (ISDN, Applet, downloaden).

Mangelnde Domänenadaptivität von Wörterbüchern. Bei der Korrektur von Texten spezieller thematischer Bereiche stößt man sehr oft auf ein ganz spezielles Vokabular, das auch nicht in extrem großen allgemeinen Wörterbüchern zu finden ist.

Wortgrenzen-Problem. Ein weiteres Problem für die lexikalische Korrektur (weitergehend für alle Korrekturverfahren) besteht in falsch erkannten Wortgrenzen. Bei der OCR werden machmal zwei Wörter zu einem zusammengezogen (ofthe) oder — seltener — ein Wort in zwei Wörter unterteilt (mehr ere). Um das erste Phänomen zu erfassen, müsste man alle möglichen Zerlegungen des Eingabewortes in zwei (oder mehr!) Teile untersuchen, die man dann beide mit dem Lexikon vergleicht. Dies resultiert in einer kombinatorischen Explosion.

6.2 Statistische Korrekturprinzipien — Bottom-Up Methoden

Als Reaktion auf die Effizienzprobleme bei lexikalischen Verfahren wurden zu Beginn der 70er Jahre rein statistische Verfahren untersucht, bei denen auf einen Wörterbuchabgleich verzichtet wird. Die Grundidee dieser Verfahren ist es, mit Hilfe von statistischen Informationen unmögliche oder unwahrscheinliche Buchstabenfolgen in einer OCR-Ausgabe aufzuspüren und für diese entsprechende Korrekturvorschläge zu machen. Neben den oben erwähnten Konfusionswahrscheinlichkeiten werden hierbei auch sogenannte *Übergangswahrscheinlichkeiten* verwendet. Diese geben die Wahrscheinlichkeit an, daß ein bestimmter Buchstabe (oder ein Buchstabenpaar) von einem anderen Buchstaben gefolgt wird. Sie sind sprachabhängig. Man kann sie ermitteln, indem man die Vorkommenshäufigkeiten von Bigrammen oder Trigrammen statistisch erfaßt. Sie werden manchmal als *Markov-Wahrscheinlichkeiten* bezeichnet.

Sei wieder $X = x_1x_2 \dots x_n$ das von der OCR gelieferte Wort. Bei der statistischen Korrektur ist das Ziel, diejenige Zeichenfolge Z zu ermitteln, für die sich die größte *a posteriori* Wahrscheinlichkeit $P(Z|X)$ ergibt. Hierbei ist $P(Z|X)$ die Wahrscheinlichkeit, daß Z die korrekte Zeichenfolge ist, wenn von der OCR die Folge X ausgegeben wurde. Es wird davon ausgegangen, daß die Länge von X der Länge des korrekten Worts Z entspricht.

Die Einschränkung auf ein vorgegebenes Wörterbuch mit Vorkommenshäufigkeiten kann ausser acht gelassen werden, wenn man annimmt, daß die Wörter von einem Markovprozeß n-ter Ordnung erzeugt werden (cf. [Rav67, Neu75, SHC83]). Dies bedeutet, daß die Wahrscheinlichkeit des m-ten Buchstabens nur von den vorausgegangenen n Buchstaben abhängt.

$$P(Z) = P(z_m|z_{m-1} \dots z_{m-n})P(z_{m-1}|z_{m-2} \dots z_{m-n-1}) \dots P(z_2|z_1)P(z_1|\epsilon).$$

Es steht hier $P(z|V)$ (anders als oben in $P(Z|X)$!) für die Wahrscheinlichkeit, daß ein Wortanfang V durch den Buchstaben z fortgesetzt wird. Für $n = 1$, was wir nachfolgend annehmen, erhält man gerade die Transitionswahrscheinlichkeiten, es ergibt sich

$$P(Z) = P(z_m|z_{m-1})P(z_{m-1}|z_{m-2}) \dots P(z_2|z_1)P(z_1|\epsilon).$$

Der zu maximierende Ausdruck erhält nun die Form

$$P(x_1|z_1)P(z_1|\epsilon) \times \prod_{i=2}^{m} P(x_i|z_i)P(z_i|z_{i-1}),$$

wobei wie oben die Faktoren $P(x_i|z_i)$ umgeschrieben werden können, wenn die umgekehrten Konfusionswahrscheinlichkeiten direkt geliefert werden.

Um den letztgenannten Ausdruck zu maximieren, wird der in Teil 3.1 bereits eingeführte *Viterbi-Algorithmus* verwendet. Die möglichen Symbole bei der OCR-Erkennung $a_1, \dots, a_N$ werden hierbei gewissermaßen als die Zustände eines HMM betrachtet. Die Folge der von der OCR gelieferten Symbole $x_1, \dots, x_n$ wird als Folge von Emissionen aufgefaßt. Die Konfusionswahrscheinlichkeit $P(a_k \mid a_l)$ wird als Emissionswahrscheinlichkeit des Symbols a_k im „Zustand“ a_l interpretiert, wobei nur die Daten der Form $P(x_i \mid a_l)$ relevant sind. Die Anfangswahrscheinlichkeiten für die einzelnen „Zustände“ ergeben sich als $C[j] := P(a_j \mid \varepsilon)$ $(1 \leq j \leq N)$.

1. *Initialisierung.* Wir setzen $\delta[j, 1] := P(a_j \mid \varepsilon) \cdot P(x_1 \mid a_j)$ $(1 \leq j \leq N)$.
2. *Rekursionsschritt.* Liegen die Werte $\delta[1, i-1], \delta[2, i-1], \dots, \delta[N, i-1]$ vor, so ergibt sich $\delta[j, i]$ als Maximum der N Werte $\delta[j', i-1] \cdot P(a_j \mid a_{j'}) \cdot P(x_i \mid a_j)$, die wir für die Alternativen $j' = 1, \dots, N$ erhalten.
3. *Abschluß.* Der optimale Pfad wird letztendlich durch den besten Wert zum Zeitpunkt $T = n$, also den maximalen Wert der Form $\delta[j, n]$ bestimmt.

Die Zustände des optimalen Pfads $z_1, \dots, z_n$ ergeben das gesuchte Ausgabewort Z, das $P(X \mid Z)$ optimiert. Um den Pfad explizit zu berechnen, muß man natürlich beim Rekursionsschritt speichern, welcher Vorgängerzustand $[j', i-1]$ auf den Wert $\delta[j, i]$ führt.

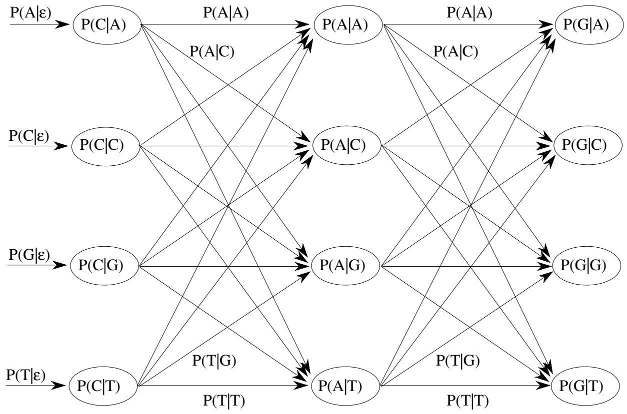


Abbildung 6.1: Viterbi-Algorithmus bei der Wortkorrektur.

Beispiel 6.2.1 Wir erläutern das Verfahren mit einem Beispiel in Abbildung 6.1. Es sei $\mathcal{A} := \{A, C, G, T\}$ das Symbolalphabet. Wir betrachten das von der OCR gelieferte Eingabewort *CAG*. In Abbildung 6.1 stellen die Knotenlabels die Konfusionswahrscheinlichkeiten dar, die Kantenlabels die Übergangswahrscheinlichkeiten. Wir haben den Ausdruck

$$P(z_1 \mid \varepsilon) \cdot P(C \mid z_1) \cdot P(z_2 \mid z_1) \cdot P(A \mid z_2) \cdot P(z_3 \mid z_2) \cdot P(G \mid z_3)$$

zu maximieren. Die möglichen Pfade durch den ,,Gartenzaun" korrespondieren zu den möglichen Wörtern $z_1z_2z_3$. Das zu maximierende Produkt ergibt sich durch Multiplikation der bei den Kanten und Knoten angegebenen Wahrscheinlichkeiten entlang des Pfads $z_1z_2z_3$. Um das maximale Produkt zu finden, wird für jeden Knoten der sich bis dahin maximal ergebende Wert rekursiv berechnet.

Verschiedene Techniken wurden betrachtet, um die Effizienz des Viterbi-Abgleichs zu steigern. Man kann die logarithmischen Wahrscheinlichkeiten bilden. Diese können dann summiert statt wie oben multipliziert werden, was schneller geht. Beim sogenannten ,,modifizierten Viterbi-Algorithmus" (MVA) wird jeweils nur eine kleine, fest vorgegebene Zahl von Alternativen zur Korrektur der einzelnen Buchstaben in Betracht gezogen. Dies führt zu einer verkleinerten Höhe des ,,Gartenzauns". Da der Viterbi-Algorithmus quadratisch in dieser Dimension ist, kann man eine erheblich Effizienzverbesserung erreichen.

Probleme/Vergleich. Das Verfahren arbeitet ohne Wörterbuch. Worthäufigkeiten werden bei der Korrektur nicht mitberücksichtigt, stattdessen wird die Häufigkeit (bzw. Wahrscheinlichkeit) der auftretenden Symbol-Bigramme (Symbolübergänge) zur Gewichtung verwendet. Das Verfahren geht nicht adäquat mit Einfügungen, Löschungen, Verschmelzungen und Splits bei der OCR-Erkennung um.

Rein statistische Verfahren der obigen Art wurden vor allem von E. M. Riseman in Zusammenarbeit mit anderen Forschern untersucht (cf. [RE71, RH74, HRF75]). Es hat sich allerdings gezeigt, daß statistische Korrekturprinzipien alleine keine ausreichende Treffsicherheit bei der Fehlerkorrektur ergeben. Teilweise ergab sich durch die Verwendung der Verfahren sogar eine Verschlechterung der Ergebnisse. Die Ursache liegt darin, daß eigentlich korrekt erkannte Wörter mit seltenen Buchstabenfolgen ungewollt ,,korrigiert" werden, indem plausiblere, aber falsche Buchstabenfolgen eingesetzt werden. Zudem kommt es allzuoft vor, daß eine ermittelte plausible Buchstabenfolge schlicht kein korrektes Wort darstellt.

Das Grundprinzip des Viterbi-Algorithmus spielt allerdings auch heute noch bei der OCR-Korrektur eine Rolle, allerdings auf einer anderen Ebene: wenn sich nach der Wortkorrektur für die einzelnen Wortpositionen in einem Satz jeweils mehrere Kandidaten ergeben, so kann man den Varianten des Viterbi-Algorithmus zur Desambiguierung einsetzen. ,,Lexikalische" Verfahren würden hier voraussetzen, daß eine Liste aller Sätze zur Verfügung steht, was unrealistisch ist. Auf diese Thematik werden wir allerdings erst später eingehen.

6.3 Hybrid-Verfahren

Aufgrund der qualitativen Probleme, die sich bei einer rein statistischen Wortkorrektur ergeben, wurden in den 70er Jahren Möglichkeiten untersucht, statistische Verfahren und lexikalische Verfahren zu kombinieren. Dies führte zur Entwicklung einer Reihe sogenannter ,,Hybridverfahren". Die Verfahren sind auch unter der Bezeichnung ,,gemischte Top-Down Bottom-Up Ansätze" bekannt.

Verwendung positioneller n-Gramme. Eine Technik zur Kombination von lexikalischen und statistischen Techniken wurde bereits in den oben genannten Arbeiten von E. M. Riseman et al. [RE71, RH74, HRF75] in verschiedenen Varianten beschrieben. Es wird wieder einschränkend davon ausgegangen, daß bei der OCR-Erkennung lediglich Substitutionsfehler auftreten. Nachfolgend beschreiben wir die Korrektur eines Wortes der festen Länge n. Ein *Positionstripel* ist ein Tripel der Form (i, j, k) wo $1 \leq i < j < k \leq n$. Für jedes Positionstripel (i, j, k) wird — ausgehend von einem Symbolalphabet mit den 26 Buchstaben $a_1 = A, \ldots, a_n = Z$ — ein dreidimensionales Array $D(i, j, k)$ der Größe $26 \times 26 \times 26$ berechnet. Der Eintrag $[l, m, n]$ von $D(i, j, k)$ wird auf 1 gesetzt genau dann, wenn es ein Lexikonwort der Länge n gibt, das die Buchstaben a_l, a_m, a_n an den Positionen i, j, k besitzt, andernfalls ist der Eintrag 0. Das Array $D(i, j, k)$ gibt also einen vollständigen Überblick darüber, welche Buchstaben-Trigramme in Lexikonwörtern an den Positionen i, j, k vorkommen. Man kann die einzelnen Einträge als binarisierte statistische Information ansehen. Die von der OCR gelieferte Buchstabenfolge $b_1 \ldots b_n$ wird nun dahingehend überprüft, ob für jedes positionelle Trigramm $b_i = a_l, b_j = a_m, b_k = a_n$ der Eintrag an der Stelle $[l, m, n]$ von $D(i, j, k)$ gleich 1 ist. Es wird also festgestellt, ob jedes positionelle Trigramm des Eingabewortes ein mögliches positionelles Trigramm des Lexikons ist. Wenn in $b_1 \ldots b_n$ verbotene positionelle Trigramme auftreten, so werden die Arrays der betreffenden Positionstripel dazu verwendet, um Korrekturvorschläge zu machen. Im einfachsten Fall läßt sich durch die Abänderung eines einzelnen Symbols eine Folge $b_1 \cdots b'_i \cdots b_n$ erreichen, wo alle positionellen Trigramme erlaubt sind. In schwierigeren Fällen müssen die Symbole mehrerer Positionen abgeändert werden.

Auch wenn das genannte Verfahren keine absolute Garantie liefert, daß eine unlexikalische Symbolfolge entdeckt wird, ist die Fehlererkennungsrate sehr hoch. Bei der Korrektur ergeben sich unter Umständen mehrere Alternativen. Man kann dann ergänzend das Lexikon einsetzen, um die Korrekturalternativen auf ihre Lexikalität zu validieren. Darüberhinaus können Informationen zu den Konfusionswahrscheinlichkeiten des Klassifikators dazu benutzt werden, um die Qualität verschiedener Korrekturalternativen zu messen.

Probleme/Vergleich. Das Verfahren setzt das Wörterbuch als eine Art Indexstruktur ein. Die Verwendung positioneller Trigramme kann zu verfälschten Ergebnissen führen, wenn bei der OCR-Erkennung Löschungen, Einfügungen, Verschmelzungen oder Splits von Symbolen auftreten.

Das „Predictor-Corrector-Verfahren". Das predictor-corrector Verfahren [Tou78, ST79, BT80] stellt eine echte Verbindung des Viterbi-Algorithmus mit einem Wörterbuchabgleich dar. Das Lexikon ist in Klassen von Wörtern derselben Länge partitioniert. Es wird zunächst das oben erwähnte modifizierte Viterbi-Verfahren (MVA) angewandt, um die Zeichenfolge Z mit dem höchsten Viterbi-Score zu ermitteln. Im zweiten Schritt wird überprüft, ob Z ein Lexikonwort ist. Verschiedene Techniken (Hash-Funktionen, binäre Suche auf alphabetisch geordnetem Lexikon,...) können eingesetzt werden, um dieses Nachschlagen effizient zu organisieren. Ist Z lexikalisch, so wird es als Korrekturergebnis ausgegeben. Andernfalls wird für jedes Lexikonwort der Länge n der Viterbi-Score (d.h. die zugehörigen Pfadtraversionskosten, die sich beim Durchlaufen der Gartenzaunstruktur ergeben) berechnet. Das Wort mit dem höchsten Score stellt das Korrekturergebnis dar. Durch verschiedene Verfeinerungen kann vermieden werden, daß der Viterbi-Score für alle Lexikonwörter der betreffenden Länge berechnet werden muß. Das Verfahren reduziert die Kosten des Lexikonabgleichs erheblich, kann aber zu einer leichten Zunahme der Fehlerrate führen.

Das „Wörterbuch-Viterbi-Verfahren". Ein häufig zitierter Hybrid-Ansatz ist in der Arbeit [SHC83] beschrieben. Da mir die Arbeit allerdings fehlerhaft erscheint, werde ich nur kurz auf die zugrundeliegende Idee eingehen. Das Verfahren kann als eine Weiterentwicklung des Predictor-Corrector-Verfahrens betrachtet werden. Anders als bei diesem wird nicht erst nach dem Ablauf des Viterbi-Algorithmus getestet, ob das erhaltene Wort lexikalisch ist. Stattdessen wird bereits beim Durchlaufen der Gartenzaunstruktur getestet, ob die erhaltenen optimalen Pfade zu den einzelnen Knoten zulässigen Präfixen von Lexikonwörtern der korrekten Länge korrespondieren. Bei jedem Schritt wird unter allen Anfangspfaden, die zu einem Knoten in der Gartenzaunstruktur führen, der Pfad mit dem bis dahin höchsten Score ausgewählt, wobei nur Pfade in die Wahl kommen, wo die „Emissionsfolge" ein Präfix eines Lexikonwortes darstellt. Der Fehler beruht darauf, daß die Beurteilung der Lexikalität der Präfixe im allgemeinen zu falschen Ablehnungen führen kann, da aus Effizienzgründen als Vergleichsgrundlage nicht alle möglichen lexikalischen Pfade (dies können exponentiell viele sein) zu den Vorgängerknoten, sondern lexikalische Pfade mit lokal optimalem Score betrachtet werden.

6.4 Probleme, Zusammenfassung und Ausblick

Alle in diesem Kapitel beschriebenen Verfahren sind insofern sehr eingeschränkt, als daß nur Substitutionen als OCR-Fehler erfaßt werden. Später entwickelte Verfahren behandeln auch Einfügungen, Löschungen, teilweise auch Verschmelzung und Zerteilung von Buchstaben. Die beschriebenen Hybrid-Verfahren stellten den Ausgangpunkt für eine Reihe neuerer Verfahren dar, die wir in den nun folgenden Kapiteln darstellen. Der Viterbi-Score wird bei diesem Verfahren in der Regel durch ein allgemeines Ähnlichkeitsmaß ersetzt, mit dem Eingabewort und Lexikonwort verglichen werden. Zusätzlich wird in der Regel die Worthäufigkeit mitberücksichtigt.

In diesem Kontext ergibt sich gleich eine doppelte Rolle von Wortabstandsmaßen: sie können einerseits eingesetzt werden, um eine Menge sinnvoller Korrekturvorschläge aus dem Lexikon auszusondern. Das Hauptaugenmerk liegt hier auf Techniken, bei denen zur Kandidatenauswahl nur ein möglichst kleiner Teil des Lexikons inspiziert werden muss. Sie dienen weiter als ein wichtiger Bestandteil von Ranking-Funktionen, die auf einer gegebenen Menge von Korrekturvorschlägen eine Präferenz definieren.

Wir gehen daher im folgenden zunächst auf Wortabstandsmaße und ihre effiziente Berechnung ein. Mit der Größe der verwendeten Wörterbücher rücken bei neueren Verfahren vor allem auch die Möglichkeiten des effizienten Zugriffs auf das Lexikon in das Zentrum des Interesses. Auch dieser Punkt wird uns in den nachfolgenden Kapiteln beschäftigen.

6.5 Bibliographische Anmerkungen

[DHH$^+$97] bietet einen kompakten Überblick über die wichtigsten Methoden zur OCR-Korrektur. Ausführlicher ist der Überblicksartikel von Karen Kukich [Kuk92]. Gute Beschreibungen statistischer und gemischter Korrekturverfahren liefern auch [Tou78] und [HS82].

Kapitel 7

Das Levenshtein-Abstandsmaß und seine Berechnung

Das Levenshtein-Abstandsmaß ([Lev66]) ist wohl das meistverwendetste Maß zur Beurteilung der Ähnlichkeit zweier Wörter. In diesem Abschnitt führen wir zunächst das Maß und einige seiner Varianten vor. Danach beschreiben wir, wie der Levenshteinabstand zweier Wörter berechnet werden kann. Zunächst erinnern wir an den allgemeinen Begriff des Abstandsmaßes.

Definition 7.0.1 Sei M eine Menge. Eine *Metrik* oder ein *Abstandsmaß* auf M ist eine reellwertige Funktion d auf M für die gilt:

$$\begin{array}{rll}
d(x,y) \geq 0 & \forall x,y \in M & \text{Nicht-Negativität} \\
d(x,y) = 0 \Leftrightarrow x = y & \forall x,y \in M & \text{Null-Eigenschaft} \\
d(x,y) = d(y,x) & \forall x,y \in M & \text{Symmetrie} \\
d(x,z) \leq d(x,y) + d(y,z) & \forall x,y,z \in M & \text{Dreiecks-Ungleichung}
\end{array}$$

Beispiel 7.0.2 In zahlreichen Kontexten werden die folgenden Abstandsmaße für Vektoren gleicher Länge verwendet:

$$\begin{array}{rl}
\textit{Schachbrett-Entferung} & \max_i |x_i - y_i|; \\
\textit{City-Block-Entfernung} & \sum_i |x_i - y_i|; \\
\textit{Euklidischer Abstand} & [\sum_i (x_i - y_i)^2]^{1/2}.
\end{array}$$

Bei nichtnegativen Komponenten, wie sie etwa im Bereich des Information Retrieval vorkommen, werden oft auch folgende Maße verwendet.

$$\begin{array}{rl}
\textit{Skalarprodukt} & \sum_i x_i \cdot y_i; \\
\textit{Cosinus} & \dfrac{\sum_i x_i \cdot y_i}{\sqrt{\sum_i x_i^2 \cdot \sum_i y_i^2}};
\end{array}$$

Der Nenner des Cosinus-Maßes stellt das Produkt der euklidischen Längen der Vektoren dar. Das Maß hängt von der Richtung, aber (bei gleicher Richtung) nicht von der Länge der Vektoren ab.

Beispiel 7.0.3 Für Wörter gleicher Länge wird manchmal die *Hamming-Entfernung* betrachtet, die die Zahl der Positionen angibt, wo sich die beiden Wörter unterscheiden.

Bei Wörtern beruhen die wichtigsten Abstandsmaße auf sogenannten *Editieroperationen*. Die wichtigsten Editieroperationen sind die *Einfügung* (insertion) eines Symbols, die *Löschung* (deletion) eines Symbols. Bereits durch Verkettung dieser zwei Operationen kann offenkundig jedes gegebene Wort in jedes andere transformiert werden. In der Regel wird auch die *Substitution* eines Symbols durch ein anderes als eine elementare Editieroperation betrachtet, weiterhin manchmal auch die *Transposition* (Vertauschung zweier aufeianderfolgender Symbole). Das nachfolgende Abstandsmaß wird sehr oft bei Stringkorrekturaufgaben verwendet.

Definition 7.0.4 Der *Editier-Abstand*[1] (oder *Levenshtein-Abstand* oder auch *Damerau-Levenshtein-Abstand*) $d_{Lev}(V, W)$ zweier Wörter V und W ist die minimale Zahl von Einfügungen, Löschungen und Substitution, die man braucht, um V in W überzuführen.

Das Levenshtein-Abstandsmaß wurde in der Literatur in zwei orthogonalen Richtungen verallgemeinert. In verschiedenen Arbeiten [LW75, OL97] werden Varianten betrachtet, wo neben den oben genannten noch weitere Editieroperationen zugelassen sind. Hierunter fallen u.a. Transpositionen zweier Symbole, Verschmelzung zweier Symbole zu einem, Splitting eines Symbols in zwei, sowie zwei-Buchstaben-Substitutionen. Darüberhinaus ist es oft sinnvoll, den Editieroperationen *in Abhängigkeit der beteiligten Symbole* einen spezifischen Kostenfaktor zuzuordnen (cf. etwa [HD80]). Unwahrscheinliche Operationen werden hierbei mit höheren Kosten belegt als wahrscheinliche. Nicht in jedem Fall erhält man allerdings auf diese Weise eine Metrik, beispielsweise kann die Symmetrie verletzt sein. Man spricht in diesem Kontext oft etwas unpräzise vom „Abstand“ von V *nach* W. Betrachtet man die Variante des Levenshtein-Abstands, wo man die minimale Summe der symbolabhängigen Editierkosten beim Überführen von V nach W als Unähnlichkeitsmaß nimmt, spricht man häufig vom *verallgemeinerten Levenshtein-Abstand*.

Beispiel 7.0.5 In Seni et al. [SKS96] wird motiviert durch das Problem der Erkennung handschriftlichen Textes eine Variante des Editierabstands behandelt, wo die Verschmelzung zweier Symbole, das Splitten eines Symbols und zwei-Buchstaben-Substitutionen neben den Standardoperationen zugelassen sind. Die Operationen sind symbolabhängig gewichtet, wobei die Deformierung der Wortgestalt, die „strichbasiert“ erfaßt wird, bei der Kostenfunktion mitberücksichtigt wird. Die in der Arbeit angegebenen Tests belegen, daß das so erweiterte Verfahren zu einer verbesserten Korrekturfähigkeit führt. Verschmelzen und Splitting von Charakteren als mögliche Editieroperationen werden etwa auch in [MG] betrachtet.

Bemerkung 7.0.6 Die Definition des Editierabstands ist geprägt von der Vorstellung, daß Wörter bei der Erstellung editiert werden, und daß bei dieser Erstellung

[1] Vorsicht: die Bezeichung *Editierabstand* wird manchmal auch verwendet für die minimale Zahl von Einfügungen und Löschungen, die man braucht, um V in W überzuführen. Beide Maße wurden in [Lev66] vorgeschlagen.

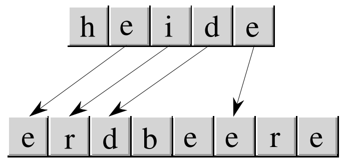


Abbildung 7.1: Traces bei der Überführung von *heide* in *erdbeere*.

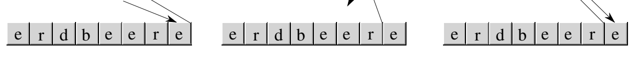
Fehler auftreten. Man kann nicht ohne weiteres davon ausgehen, daß der Begriff des (Standard-) Editierabstands für Anwendungen bei der OCR-Korrektur optimal ist. Hier treten neben den Standardeditieroperationen eine Reihe weiterer Fehler teils sogar häufig auf. Hierzu vergleiche man Kapitel 5. Es kann im Kontext der OCR-Korrektur daher sinnvoll sein, weitere Editieroperationen zuzulassen.

7.1 Berechnung des Levenshtein-Abstands nach Wagner und Fisher

Zur Berechnung des Editierabstands gibt es ein Standardverfahren, das auf dynamischer Programmierung beruht. Es wurde in Wagner und Fisher [WF74] eingeführt, aber unabhängig auch an anderen Stellen in der Literatur vorgeschlagen (cf. [OTK76]).

Traces

Wenn wir uns vorstellen, daß das Wort *heide* durch die Anwendung von Löschungen, Einfügungen oder Substitutionen in *erdbeere* übergeführt wird, so können wir –solange wir uns für die kleinste Zahl notwendiger Operationen interessieren — annehmen, daß ein einmal eingefügter Buchstabe nicht selbst wieder substituiert oder gelöscht wird. Sinngemäß dasselbe gilt für substituierte Buchstaben. Unter dieser Annahme können wir die angewandten Editieroperationen in eindeutiger Weise geometrisch dadurch kennzeichnen, daß wir jeden substituierten Buchstaben des Wortes *heide* mit seinem Substitutionsresultat in *erdbeere* durch einen Strich verbinden. Auch Symbole, die einfach übernommen werden, werden durch einen Strich verbunden, wir werden hier nachfolgend von kostenfreien ,,Substitutionen" reden. Alle Buchstaben von *heide* (resp. *erdbeere*), die nicht durch einen Strich verbunden sind, stellen dann Löschungen (resp. Einfügungen) dar. Dies ist in Abbildung 7.1 angedeutet. Es ist klar, daß sich zwei Substitutionslinien nicht überschneiden können, da dies eine Art von Vertauschungs- oder Transpositionsregel voraussetzen würde. Wenn wir die möglichen Traces beim Transformieren von *heide* in *erdbeere* betrachten, können wir drei Fälle unterscheiden:



Die Situation, wo keines der beiden Schlußsymbole Ausgangs- oder Endpunkt eines Strichs ist, kann ignoriert werden: hier würde das letzte Symbol von *heide* gelöscht werden und das letzte Symbol von *erdbeere* eingefügt werden. Es gibt aber eine kostengünstigere Lösung, wo wir die ,,kostenfreie Substitution" von ,,e" durch ,,e" durchführen. Nehmen wir nun einmal an, wir würden den Levensthein-Abstand zwischen folgenden Wortpaaren bereits kënnen:

Erstes Wort	*heid*	*heide*	*heide*
Zweites Wort	*erdbeere*	*erdbeer*	*erdbeere*
Editierabstand	c_1	c_2	c_3

Die obige Fallunterscheidung zeigt, daß wir die Möglichkeit haben,

1. das Umeditieren von *heid* in *erdbeere* durch eine Löschung des letzten Symbols von *heide* zu ergänzen (Gesamtkosten $c_1 + 1$),
2. das Umeditieren von *heide* in *erdbeer* durch eine Einfügung des letzten Symbols von *erdbeere* zu ergänzen (Gesamtkosten $c_2 + 1$),
3. das Umeditieren von *heid* in *erdbeer* durch die kostenfreie ,,Substitution" von ,,e" durch ,,e" zu ergänzen (Gesamtkosten c_3).

Das Minimum der drei möglichen Gesamtkosten ergeben damit den Levenshtein-Abstand. Natürlich ist der dritte Fall ein Sonderfall: Hätten wir *heidi* in *erdbeere* zu transformieren, so wären hier die Gesamtkosten $c_3 + 1$ zu veranschlagen.

Die Überlegung zeigt, daß man den Levenshtein-Abstand zweier Wörter in systematischer Weise aus den Levenshtein-Abständen derjenigen Präfixe erhält, wo man eines oder beide der Schlußsymbole wegläßt. Dies führt in natürlicher Weise auf eine rekursive Berechnung mittels dynamischer Programmierung.

Ist V ein Wort der Länge $n \geq 0$, so bezeichnen wir mit V_i das i-te Symbol von V $(1 \leq i \leq n)$. Es steht weiter $V_{1,i}$ für das Präfix mit den Symbolen $V_1, \ldots, V_i$ von V $(0 \leq i \leq n)$. Für $i = 0$ ist also $V_{1,i}$ gerade das leere Wort ,,ϵ". Bei der nachfolgende Darstellung des Berechnungsverfahrens verwenden wir eine Notation, die sich sofort verallgemeinern läßt. Wir setzen

$$
\begin{aligned}
c(a, \epsilon) &= 1 && \text{die Kosten für das Löschen des Symbols } a,\\
c(\epsilon, a) &= 1 && \text{die Kosten für das Einfügen des Symbols } a,\\
c(a, b) &= 1 && \text{die Kosten für das Substitieren von } b \text{ für } a\ (a \neq b),\\
c(a, a) &= 0. &&
\end{aligned}
$$

Es seien nun zwei zu vergleichende Wörter V und W der Länge m bzw. n gegeben. Wir definieren ein zweidimensionales Array $C[i, j]$ der Größe $(m + 1) \times (n + 1)$, wo $0 \leq i \leq m$ und $0 \leq j \leq n$. Die Idee ist, daß im Feld $C[i, j]$ der Editierabstand von $V_{1,i}$ nach $W_{1,j}$ notiert wird. Um die Array-Einträge zu berechnen, verwenden wir die Idee der dynamischen Programmierung.

Zur *Initialisierung* berechnen wir zunächst die Einträge der ersten Spalte $C[\_, 0]$. Es ist $C[i, 0]$ der Editierabstand von $V_{1,i}$ und $W_{1,0} = \epsilon$. Er ergibt sich offenkundig durch die Summe der Kosten der Löschung der Symbole $V_1, \ldots, V_i$ $(1 \leq i \leq m)$. Wir erhalten $C[0, 0] := 0$ und $C[i, 0] = C[i - 1, 0] + c(V_i, \epsilon)$ für $1 \leq i \leq m$. Analog erhalten wir für die verbleibenden Einträge der ersten Zeile die Formel $C[0, j] = C[0, j - 1] + c(\epsilon, W_j)$ $(1 \leq j \leq n)$.

Bei der Berechnung der verbleibenden Einträge $C[i, j]$ setzen wir induktiv voraus, daß die Werte $C[i - 1, j - 1]$, $C[i - 1, j]$ und $C[i, j - 1]$ bereits vorliegen und

Initialisierung
$C[0,0] := 0$
for $i = 1$ to m
$C[i,0] := C[i-1,0] + c(V_i, \epsilon)$
for $j = 1$ to n
$C(0,j) := C(0,j-1) + c(\epsilon, W_j)$

Berechnung der inneren Array-Werte
for $i = 0$ to $m-1$
for $j = 0$ to $n-1$
$C[i,j] := min\{C[i-1,j-1] + c(V_i, W_j), C[i,j-1] + c(\epsilon, W_j), C[i-1,j] + c(V_i, \epsilon)\}$

Abbildung 7.2: Der Algorithmus von Wagner und Fisher

im obigen Sinn korrekt sind. Wie oben dargestellt sind drei Möglichkeiten denkbar, wie $W_{1,j}$ aus $V_{1,i}$ hervorgeht.

1. indem wir zunächst $V_{1,i-1}$ in $W_{1,j-1}$ umeditieren und die Substitution von V_i durch W_j anschließen, was Gesamtkosten von $C[i-1,j-1] + c(V_i, W_j)$ ergibt,
2. indem wir zunächst $V_{1,i}$ zu $W_{1,j-1}$ umeditieren und die Einfügung von W_j anschließen, was Gesamtkosten von $C[i,j-1] + c(\epsilon, W_j)$ ergibt, oder
3. indem wir zunächst $V_{1,i-1}$ zu $W_{1,j}$ umeditieren und die Löschung von V_i ergänzen, was auf Gesamtkosten von $C[i-1,j] + c(V_i, \epsilon)$ führt.

Der Eintrag $C[i,j]$, also der Editierabstand von $V_{1,i}$ nach $W_{1,j}$ ergibt sich als Minimum dieser drei Alternativen. Hieraus erhält man die Beschreibung des Algorithmus von Wagner und Fisher in Abbildung 7.2.

Beispiel 7.1.1 Als Beispiel berechnen wir den Levenshtein-Abstand der Wörter „heide“ und „erdbeere“

 i	j	0	1 e	2 r	3 d	4 b	5 e	6 e	7 r	8 e
0		**0**	1	2	3	4	5	6	7	8
1	h	**1**	1	2	3	4	5	6	7	8
2	e	2	**1**	2	3	4	4	5	6	7
3	i	3	2	**2**	3	4	5	5	6	7
4	d	4	3	3	**2**	**3**	**4**	5	6	7
5	e	5	4	4	3	3	3	**4**	**5**	**6**

Die Wörter haben also den Abstand 6.

Offenkundig entspricht die Platz- und Zeitkomplexität des Algorithmus gerade der Dimension des Arrays und ist von der Ordnung $O(m, n)$.

Um eine mögliche Editierfolge mit minimalen Kosten zu erhalten, muß man sich nur beim Algorithmus merken, durch welche(s) Vorgängerfeld(er) ($C[i-1,j-1]$, $C[i,j-1]$ oder $C[i,j-1]$) man den Eintrag von $C[i,j]$ erhalten hat. Im obigen Beispiel ist ein solcher Pfad in Fettdruck hervorgehoben. Horizontale Schritte markieren Einfügungen, vertikale Schritte Löschungen. Die Zielfelder der Diagonalschritte entsprechen den Strichen der Traces. Im gegebenen Fall haben wir Striche

- von Symbol 2 aus *heide* zu Symbol 1 aus *erdbeere* (kostenfreie Substitution),

- von Symbol 3 aus *heide* zu Symbol 2 aus *erdbeere*,
- von Symbol 4 aus *heide* zu Symbol 3 aus *erdbeere*,
- von Symbol 5 aus *heide* zu Symbol 6 aus *erdbeere*.

Wir erhalten also den Trace aus Abbildung 7.1.

7.2 Verallgemeinerungen

Hängen die Kosten von Substitutionen, Einfügungen und Löschungen von den beteiligten Symbolen ab, so kann man eine Variante des Grundverfahren verwenden, um die minimalen Kosten zu berechnen, die man braucht, um V in W überzuführen.[2] Voraussetzung ist allerdings, daß es bei gegebenem V und W stets eine Editierfolge mit minimalen Kosten gibt, wo ein Symbol, das einmal gelöscht, eingeführt oder substituiert wurde, danach nicht mehr verändert wird. Dies ist stets der Fall, solange die Kostenfunktion die Dreiecksungleichung

$$c(a, c) \leq c(a, b) + c(b, c)$$

erfüllt für alle Symbole a, b, c das Alphabets. Das veränderte Verfahren mit symbolabhängigen Kosten folgt wieder genau der obigen Beschreibung in Abbildung 7.2.

In [Ukk85] wird das Verfahren (und eine unten betrachtete Verbesserung) auf beliebige Editieroperationen mit symbolabhängigen Kosten verallgemeinert. Eine allgemeine Editieroperation hat die Form $A \to B$, wo A und B Wörter über dem gegebenen Alphabet sind. Sie wird angewendet, indem man ein Vorkommen von A in einem Wort V durch B ersetzt. Ist E eine vorgegebene Menge von Editieroperationen, die die obige Dreiecksungleichung (entsprechend verallgemeinert) erfüllt, so kann man folgendes Schema der Berechnung der Werte $C[i, j]$ verwenden.

1. Wir setzen $C[0, 0] := 0$.
2. Ist $C[i', j']$ bereits für alle Paare $[i', j']$ berechnet, wo $i' < i$ oder $j' < j$ gilt, so setzt man $C'[i, j]$ als die kleinste aller Zahlen $C[i - k, j - r] + c(V_{i-k+1} \cdots V_i, W_{j-r+1} \cdots W_j)$, wo $V_{i-k+1} \cdots V_i \to W_{j-r+1} \cdots W_j \in E$. Es ist dann $C[i, j] = C'[i, j]$ falls $V_i \neq W_j$ und $C[i, j] = \min\{C'[i, j], C[i - 1, j - 1]\}$ falls $V_i = W_j$.

Beispiel 7.2.1 Möchte man auch Transpositionen als Editieroperationen erlauben, so kann man das Verfahren von Wagner und Fisher verwenden, die Minimumsbildung bei der Berechnung von $C[i, j]$ hat dann gegebenenfalls den vierten Term $C[i-2, j-2]+c(V_{i-1}V_i, V_iV_{i-1})$ zu berücksichtigen, vorausgesetzt, $V_{i-1}V_i \to V_iV_{i-1}$ ist eine erlaubte Transposition.

Beispiel 7.2.2 In [MG] werden — neben Einfügungen, Löschungen und Substitutionen — auch Verschmelzungen zweier Buchstaben zu einem und das Splitten eines Buchstabens zu zweien als Editieroperationen betrachtet. Es ergibt sich die

[2] Vorsicht: die Kosten, um V nqach W umzueditieren, können verschieden sein von den Kosten der Überführung von W in V. Ist etwa $c(a, \epsilon) = 0,5$ und $c(\epsilon, a) = 0,4$) und wählen wir $V := a$ und $W := \epsilon$, so erhalten wir die zwei Werte $0,5$ und $0,4$.

Vorschrift

$$
\begin{aligned}
C[0,0] &:= 0 \\
C[i,j] &= min \begin{cases} C[i-1,j] + c(V_i,\epsilon) \\ C[i,j-1] + c(\epsilon,W_j) \\ C[i-1,j-1] + c(V_i,W_j) \\ C[i-1,j-2] + c(V_i,W_{j-1}W_j) \\ C[i-2,j-1] + c(V_{i-1}V_i,W_j) \end{cases}
\end{aligned}
$$

7.3 Ukkonen's Verbesserung

Verfahren zur Stringabstandsberechung, die eine bessere Komplexität als das Verfahren von Wagner und Fisher besitzen, wurden eine zeitlang intensiv untersucht. Bekannt ist u.a. der Algorithmus von Masek und Paterson [MP80], der die Zeitkomplexität $O(n^2/log\ n)$ besitzt. Allerdings ist dieses Verfahren aufgrund der notwendigen Hilfsberechnungen nur asymptotisch für sehr lange Strings (z.B. Files) wirklich effizienter und daher für Anwendungen bei der OCR-Korrektur relativ uninteressant. Von Ukkonen [Ukk85] wurde eine Modifikation gefunden, die die Komplexität $O(md)$ besitzt, wo m die Länge des kürzeren Worts ist und $d = d_{Lev}(V_{1,m}, W_{1,n})$ der Editierabstand (wir nehmen an, daß V nicht länger als W ist). Der Algorithmus eignet sich daher gut in Fällen, wo die zu vergleichenden Wörter einen geringen Editierabstand haben. Die Kosten der Editieroperationen können nachfolgend von den an der Operation beteiligten Symbolen abhängen.

Der Algorithmus baut auf einem Test auf, wo für einen vorgegebenen Grenzwert h getestet wird, ob der Editierabstand der Eingabewörter V und W kleinergleich h ist. Trifft dies zu, so erhält man als Ergebnis des Tests automatisch den tatsächlichen Abstand mitgeliefert. Im negativen Fall wird der Test mit dem doppelten Grenzwert $h' := 2h$ wiederholt, solange bis man zu einem positiven Ergebnis kommt. Wir stellen nachfolgend zunächst den Test vor und berechnen seine Komplexität. Nachfolgend zeigen wir, daß das Gesamtverfahren die Zeitkomplexität $O(md)$ besitzt.

Das Testverfahren

Um das Testverfahren zu beschreiben, führt man zunächst den folgenden *Editiergraphen* ein: die Knoten sind die Koordinaten $[i, j]$ der Tabelle das Verfahrens von Wagner und Fisher, wo $0 \leq i \leq m$ und $0 \leq j \leq n$ (wir setzen $m \leq n$ voraus!). Die Kanten haben die Form $[i-1, j] \rightarrow [i, j]$ (vertikale Kante, entspricht Löschung von V_i), $[i, j-1] \rightarrow [i, j]$ (horizontale Kante, entspricht Einfügung von W_j) und $[i-1, j-1] \rightarrow [i, j]$ (Diagonalkante, entspricht Substitution von V_i durch W_j). Wir können jede Kante mit dem Gewicht der zugehörigen Operation versehen. Dann ergibt sich der Abstand d von V und W gerade als die minimale Summe der Kantengewichte auf einem Pfad von $[0, 0]$ zu $[m, n]$. Ein Pfad, wo sich diese Summe der Kantengewichte ergibt, wird nachfolgend *optimaler Pfad* genannt. Allgemeiner ist $d(V_{1,i}, W_{1,j})$ gerade die minimale Summe der Kantengewichte auf einem Pfad von $[0, 0]$ zu $[i, j]$. Da das Gewicht jeder Kante nichtnegativ ist, steigt die Summe der Kantengewichte entlang eines Pfads monoton an. Um die Notation zu vereinfachen, schreiben wir nachfolgend $d_{i,j}$ für die Editierkosten $d(V_{1,i}, W_{1,j})$.

Sei nun ein Grenzwert h vorgegeben. Wenn tatsächlich $d(V, W) \leq h$ gilt, dann kann ein optimaler Pfad von $[0, 0]$ zu $[m, n]$ mit minimalem Gewicht keinen Knoten

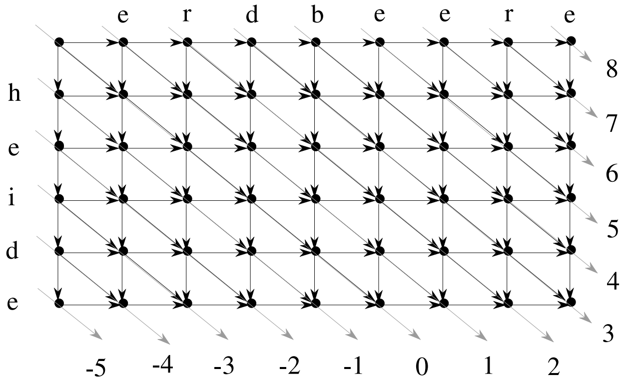


Abbildung 7.3: Editiergraph für ,,heide“ und ,,erdbeere“ mit Diagonalnummern

$[i, j]$ enthalten wo $d(V_{1,i}, W_{1,j}) > h$ gilt.

Es bezeichne nun c_{min} die minimalen Kosten einer Einfügungs- oder Löschoperation. Wir setzen voraus, daß $c_{min} > 0$. Wir numerieren nun die Diagonalen des Editiergraphen durch wie folgt: Punkt $[i, j]$ gehört zur Diagonalen mit Nummer $j - i$. Das Ziel ist, zu zeigen, daß sich die Punkte eines optimalen Pfads von $[0, 0]$ zu $[m, n]$ auf einem kleinen Band von Diagonalen befinden müssen.

Beispiel 7.3.1 Der Editiergraph und die zugehörigen Diagonalnummern, der sich beim Vergleich von $V = heide$ mit $W = erdbeere$ ergibt, ist in Abbildung 7.3 dargestellt.

Seien nun zwei Punkte $[i, j]$ und $[i', j']$ mit Diagonalnummern p und p' gegeben. Gilt $p' \geq p$, so enhält jeder Pfad von $[i, j]$ zu $[i', j']$ zumindest $p' - p$ Einfügungen oder horizontale Kanten. Gilt $p > p'$, so enthält jeder Pfad von $[i, j]$ zu $[i', j']$ zumindest $|p' - p|$ Löschungen oder vertikale Kanten. Daraus erhalten wir

$$d_{i',j'} - d_{i,j} \geq |p' - p| \cdot c_{min} = |(j' - i') - (j - i)| \cdot c_{min}.$$

Liegt $[i, j]$ auf einem optimalen Pfad von $[0, 0]$ zu $[m, n]$, so gilt

$$\begin{aligned} d_{m,n} &= (d_{i,j} - d_{0,0}) + (d_{m,n} - d_{i,j}) \\ &\geq |j - i| \cdot c_{min} + |(n - m) - (j - i)| \cdot c_{min}. \end{aligned}$$

Um diese Ungleichung weiter zu analysieren, unterscheiden wir die Fälle $j \leq i$ und $j > i$.

Fall 1: $j \leq i$. Hier ergibt sich $d_{m,n} \geq (i - j + n - m - j + i) \cdot c_{min} = (2(i - j) + n - m) \cdot c_{min}$ und damit $d_{m,n}/c_{min} - (n - m) \geq -2(j - i)$. Da $i - j \in \mathbb{N}$ folgt

$$-\lceil (\frac{d_{m,n}}{c_{min}} - (n - m))/2 \rceil \leq j - i \leq 0.$$

wo $\lceil x \rceil$ für den größten ganzzahligen Anteil der Zahl $x \in \mathbb{R}$ steht.

Fall 2: $j > i$. Wir unterscheiden genauer die Fälle, wo $n - m \geq j - i$ und wo $n - m < j - i$. Im ersten Teilfall ergibt sich aus der obigen Ungleichung $d_{m,n} \geq (j - i + n - m - (j - i)) \cdot c_{min}$ und damit

$$d_{m,n}/c_{min} \geq n - m,$$

im zweiten Fall ergibt sich $d_{m,n} \geq (j - i + j - i - (n - m)) \cdot c_{min}$ und damit $d_{m,n}|c_{min} - (n - m) \geq 2(j - i) - 2(n - m)$. Da $j - i \in \mathbb{N}$ folgt

$$\lceil (\frac{d_{m,n}}{c_{min}} - (n - m))/2 \rceil + (n - m) \geq j - i \geq 0.$$

Wenn wir die zwei Teilfälle, wo sich eine Bedingung an $[i, j]$ ergibt, zusammenfassen, so zeigt dies, daß die Punkte $[i, j]$ auf optimalen Pfaden auf Diagonalen mit Nummern zwischen $-k$ und $n - m + k$ liegen, wo $k = \frac{d_{m,n}}{c_{min}} - (n - m))/2$. Gilt nun $d_{m,n} \leq h$, so liegen die relevanten Punkte auf Diagonalen mit Nummern $-k^*$ und $n - m + k^*$ liegen, wo $k^* = \frac{h}{c_{min}} - (n - m))/2$.

Um $d_{m,n} \leq h$ zu überprüfen, berechnen wir die Matrixelemente von $C[i, j]$ im Bereich der Diagonalen zwischen $-k^*$ und $n - m + k^*$. Ergibt sich hierdurch ein Wert von $C[m, n]$ kleinergleich h, so ist der Test positiv. Umgekehrt zeigt ein Wert $C[m, n]$ oberhalb h aufgrund der obigen Überlegungen, daß die Hypothese $d_{m,n} \leq h$ falsch sein muß.

Für jede Zeile müssen höchstens $n - m + 2k^* + 1$ Matrixelemente berechnet werden. Aus $k^* = \lceil (h/c_{min} - (n-m))/2 \rceil$ folgt leicht $n-m+2k^*+1 \leq 1+h/c_{min} = O(h)$. Damit beträgt die Zeitkomplexität des Tests $O(hm)$.

Die Berechnung des Editierabstands

Aus dem ersten Teilfall von Fall 2 erhalten wir, daß stets $d_{m,n}/c_{min} \geq n - m$ gilt. Das Gesamtverfahren beginnt daher den Test mit dem Grenzwert $h_0 := (n - m + 1) \cdot c_{min}$ in Runde 0. Wie oben geschildert wird der Test solange wiederholt, bis das Ergebnis positiv ist, wobei der Grenzwert h bei jeder neuen Runde verdoppelt wird. Bezeichnet h_l den Grenzwert von Runde l, so gilt also $h_l = 2^l h_0$. Werden $r + 1$ Runden benötigt, um $d_{m,n}$ zu berechnen, so ist die Gesamtkomplexität von der Ordnung $O(m \sum_{l=0}^{r} h_l) = O(m(2h_r - h_0)) = O(mh_r)$. Da $d_{m,n} > h_r/2$ ist die Gesamtkomplexität $O(md)$.

7.4 Bibligraphische Anmerkungen

In [RY97] wird die Frage diskutiert, wie man ausgehend von einer Reihe von Trainingsbeispielen symbolabhängige Gewichte für die Editieroperationen lernen kann, die die Wahrscheinlichkeiten der Editieroperationen wiedergeben indem sie häufige Editieroperationen mit geringem Gewicht versehen, seltene mit höherem Gewicht. Dieselbe Problematik wird auch in [WBR95] behandelt.

Kapitel 8

Trigramm-Ähnlichkeitsmaße und weitere Abstandsmaße

Neben dem Levenshtein-Abstand existieren eine Reihe weiterer Maße, die bei der OCR-Korrektur nützlich sind. Zunächst können Trigramme — oder allgemeiner n-Gramme — über Buchstabenalphabeten in unterschiedlicher Weise zur Definition von Wortabstandsmaßen beziehungsweise Wortähnlichkeitsmaßen verwendet werden. In der Regel werden dabei die Trigramme aufeinanderfolgender Buchstaben der zwei zu vergleichenden Wörter betrachtet, anders als bei den im vorigen Kapitel erwähnten „positionellen“ Trigrammen. Neben den Trigrammabständen gibt es noch eine Reihe weiterer Wortabstandsmaße, die von geringerer Bedeutung sind.

8.1 Trigramm-Ähnlichkeitsmaße

Nachfolgend werden einige Trigrammähnlichkeitsmaße aus der Literatur aufgeführt. Ähnlichkeitsmaße liefern umso größere Werte, desto ähnlicher die zu vergleichenden Objekte sind. Durch Kehrwertbildung ergibt sich grob ein Verhalten, wie man es von Abstandsmaßen kennt, wobei oft die Nulleigenschaft verletzt ist. Mit Σ bezeichnen wir das zugrundegelegte Alphabet. Bei OCR-Anwendungen stellt dies die Menge aller zu unterscheidenden Symbole dar.

Definition 8.1.1 Eine Zeichenfolge der Länge $n \geq 1$ über Σ wird als *n-Gramm* über Σ bezeichnet. Im Fall $n = 1, 2, 3$ spricht man von *Unigrammen*, *Bigrammen* respektive *Trigrammen.*

Definition 8.1.2 Sei $W = W_1 \cdots W_n$ ein Wort über Σ, sei $1 \leq i \leq n-2$. Das Trigramm $v = v_1 v_2 v_3$ über Σ *kommt in W an Stelle i vor* genau dann, wenn $v = W_i W_{i+1} W_{i+2}$ gilt.

Mit $Trg(W)$ bezeichnen wir die Menge aller in W vorkommenden Trigramme.

Beispiel 8.1.3 In [AFW83] wird das Ähnlichkeitsmaß

$$d(V, W) := \frac{2c}{n+m}$$

vorgeschlagen, wo $c := |Trg(V) \cap Trg(W)|$ die Zahl der gemeinsamen Trigramme von V und W ist, und wo n (m) die Länge des Worts V (W) angibt. Das Maß bildet die Grundlage für ein Verfahren zur Wortkorrektur. Laut [AFW83] ist die Methode sehr gut für Auslassung und Einfügen, adäquat für Substitutionen, aber schlecht für Transpositionen. Offenkundig ist das Abstandsmaß bei kleiner Wortlänge (≤ 4) sehr problematisch, da bereits ein Editierfehler fast alle Trigramme mutiert. Ein anderes Problem tritt auf, wenn ein falsch geschriebenes Wort ganz in einem Lexikonwort enthalten ist oder umgekehrt. So erhält bei Eingabe von „*concider*" der Korrekturvorschlag „*cider*" den Wert $6/13 \approx 0,462$ und der offenkundig bessere Vorschlag „*consider*" lediglich den Wert $6/16 \approx 0,375$. Diese Fehlerquelle wird durch das Ähnlichkeitsmaß

$$\frac{c}{\max(n, m)}$$

entschärft. Hier ergibt sich für beide Korrekturvorschläge der Wert $3/8 = 0,375$.

Beispiel 8.1.4 In Ukkonen [Ukk92] wird

$$d(V, W) := \sum_{v \in \Sigma^3} |\sharp v(V) - \sharp v(W)|$$

als Abstand der Wörter V und W vorgeschlagen. Hierbei bezeichnet $\sharp v(U)$ die Zahl der Vorkommen des Trigramms v in U, die Summierung läuft über alle möglichen Trigramme, oder über alle in V und W vorkommenden. Wir erhalten $d(concider, cider) = 3$ und $d(concider, consider) = 6$.

In der Praxis kommt es sehr selten vor, daß ein Wort mehrere Vorkommen eines Trigramms hat. Wenn man mehrfache Vorkommen ignoriert, kann die obige Formel zu

$$d(V, W) = |Trg(U)| + |Trg(V)| - 2|Trg(U) \cap Trg(V)|$$

vereinfachen (vgl. [ZD95, HCW]).

Beispiel 8.1.5 In [MG] werden

$$d(V, W) := \frac{|Big(V) \cap Big(W)|}{|Big(W)|}$$

und

$$d(V, W) := \frac{|Big(V) \cap Big(W)|}{|Big(V) \cup Big(W)|}$$

als Abstandsmaße verwendet, wo $Big(U)$ die Menge der Bigramme von U ist.

8.2 Berechnung von Trigramm-Abstandsmaßen

Trigrammabstandsmaße können in der Regel sehr einfach berechnet werden. Exemplarisch stellen wir die in [Ukk92] angegebene Berechnung des Abstands $d(V, W)$ aus Beispiel 8.1.4 dar. Hierbei werden die Symbole des Alphabets mit Zahlen von 0 bis $c - 1$ durchnummeriert, es ist also $\Sigma = \{A_0, A_1 \ldots, A_{c-1}\}$. Einem Trigramm $v = A_i A_j A_k$ wird der Wert

$$\tilde{v} := ic^2 + jc + k$$

zugewiesen. Der Wert jedes in W vorkommenden Trigramms kann in konstanter Zeit berechnet werden. Anstelle einer naiven Berechnung kann man folgende Beobachtung verwenden. Sind

$$\begin{aligned} v_i &= W_s W_{s+1} W_{s+2} = A_i A_j A_k \\ v_{i+1} &= W_{s+1} W_{s+2} W_{s+3} = A_j A_k A_l \end{aligned}$$

zwei aufeinanderfolgende Trigrammvorkommen des Worts $W = W_1 \cdots W_n$, so gilt

$$\tilde{v}_{i+1} = (\tilde{v}_i - ic^2) \cdot c + l.$$

Ist nun ein Wort W gegeben, so gehen wir es von links nach rechts durch und berechnen die Werte aller auftretenden Trigramme. In einem Array $A(W)$ der Größe c^3, wo die Positionen den möglichen Trigrammwerten entsprechen, legen wir die Zahl der Vorkommen eines bestimmten Trigrammwerts ab. Gleichzeitig legen wir eine Liste $L(W)$ aller Werte an, die im Wort W vorkommen. Ein Wert, der bereits aufgetreten ist, wird nicht nochmals in der Liste abgelegt. Die Überprüfung erfolgt beim Hochzählen im Array. Offenkundig können $A(W)$ und $L(W)$ in linearer Zeit in der Länge von W angelegt werden. Um den Abstand $d(V, W)$ zweier Wörter V und W der Länge m und n zu berechnen, werden $A(V), A(W), L(V)$ und $L(W)$ in Zeit $O(n + m)$ berechnet. Es ist damit

$$d(V, W) := \sum_{\tilde{v} \in L(V) \cup L(W)} |A(V)[\tilde{v}] - A(W)[\tilde{v}]|$$

in linearer Zeit in der Summe der Wortlängen zu berechnen. Hieraus ergibt sich das folgende Lemma aus [Ukk92].

Lemma 8.2.1 *Der Abstand $d(V, W) := \sum_{v \in \Sigma^3} |\sharp v(V) - \sharp v(W)|$ zweier Wörter V und W der Länge m und n kann in Zeit $O(m + n)$ und Platz $O(|\Sigma|^3 + m + n)$ berechnet werden.*

Man beachte, daß wir hier eine lineare Zeitkomplexität im Vergleich zu der quadratischen (worst-case) Komplexität bei der Berechnung des Levensthein-Abstands erhalten. In [Ukk92] wird noch ein weiteres Verfahren zur Berechnung des obigen Abstands angegeben, das komplizierter ist, aber einen geringeren Platzbedarf hat.

8.3 Weitere Wortabstandsmaße

Wortabstand als Abstand von Vektoren. Alternativ können Wörter als n-Gramm-Vektoren dargestellt werden. Im Prinzip werden damit die üblichen Abstandsmaße (Skalarprodukt, Cosinus etc.) für Dokumente in Vektorendarstellung aus dem Information Retrieval (IR) verwendbar. Wie in [ZD95] bemerkt wird, bestehen allerdings prinzipielle Unterschiede zwischen beiden Problemen. Im IR werden gute Abstandsmaße stets die Länge des Textes ausfaktorisieren, um Texte sehr unterschiedlicher Länge (aber einer vergleichbaren Richtung) als ähnlich einzustufen. Dieses Verhalten ist beim Vergleich von Wörtern unerwünscht. Prinzipiell kann man Wörter auf viele Arten durch Features als Vektoren in einem Vektorraum beschreiben und dann Vektorabstandsmaße verwenden. Vgl. Hierzu [Kuk92]. In vielen Fällen werden aber obige Einschränkungen in ähnlicher Form zutreffen.

Längstes gemeinsames Präfix oder Suffix, maximale gemeinsame Teilstrings. Die Länge des größten gemeinsamen Präfixes oder Suffixes von Eingabewort und Korrekturwort wird manchmal auch als Abstandsmaß verwendet. Andere Abstandsmaße (cf. [Ukk92]) basieren auf der Länge maximaler gemeinsamer Teilstrings.

Kombinierte Ähnlichkeitsmaße. Mehrere Ähnlichkeitsmaße können in gewichteter Weise zu einem neuen Ähnlichkeitsmaß kombiniert werden. Führt ein Ähnlichkeitsmaß zu häufig auf denselben Wert, kann es durch ein zweites Maß verfeinert werden.

Kapitel 9

Ähnlichkeitsschlüssel

Ähnlichkeitsschlüssel — manchmal auch Codes genannt — sind Abbildungen, die zu einer Partitionierung des Lexikons in Klassen ähnlicher Wörter eingesetzt werden. Die Äquivalenzklassen bestehen hierbei aus den Wörtern mit demselben Funktionswert oder Code. Ähnlichkeitsschlüssel können als verallgemeinerte Hashfunktionen aufgefaßt werden. Während jedoch bei Hash-Funktionen identische Hash-Werte für unterschiedliche Wörter in der Regel unerwünscht sind, wird es bei der Verwendung von Ähnlichkeitsschlüsseln angestrebt, daß ähnliche Wörter zur selben Partitionsklasse gehören. Ähnlichkeitsschlüssel werden bei der lexikalischen Wortkorrektur in zwei Weisen angewandt. Indem man bei Eingabe eines fehlerhaften Wortes die Lexikonwörter mit demselben (oder einem fast identischen) Ähnlichkeitsschlüssel ausgibt, kann man unter Umständen bereits eine brauchbare Kandidatenliste zur Korrektur erhalten. Notwendige Voraussetzung ist, daß die betreffende Partitionsklasse nicht zu groß ist. Meist jedoch stellt die Partitionsklasse nur eine Vorselektion dar, man ordnet dann die Kandidaten, indem man ihren Abstand zum Eingabewort bezüglich eines geeigneten Abstandsmaßes (z.B. Levenshteinmaß) berechnet. Auf diese Weise wird die Notwendigkeit vermieden, den Abstand zu jedem Wort des Lexikons zu berechnen, was zu zeitaufwendig wäre.

9.1 Beispiele von Ähnlichkeitsschlüsseln

Wir geben zunächst eine Reihe von Ähnlichkeitsschlüsseln an, die in der Literatur erwähnt sind. Da es viele potentiell interessante Ähnlichkeitsschlüssel gibt, ist die nachfolgende Liste in keiner Weise erschöpfend.

Beispiel 9.1.1 Die Wortlänge stellt einen primitiven Ähnlichkeitsschlüssel dar. Dieser liefert bei konventionellen Wörterbüchern zumindest im Bereich kleiner Wortlängen sehr grobe (damit uninteressante) Partitionsklassen. Da bei der OCR-Erfassung Verschmelzungen und Splits von Symbolen auftreten können, ist für ein mißgestaltetes Eingabewort die Menge aller Lexikonwörter derselben Länge in der Regel keine ausreichende Kandidatenliste zur Korrektur. Man wird nur Wörter einer stark divergierenden Länge ausschließen.

Beispiel 9.1.2 Bei manchen OCR-Anwendungen, hauptsächlich aber bei der Rechtschreibkorrektur in getipptem Text, treten beim ersten Buchstabens eines Wortes nur sehr wenige Fehler auf. Man kann dann den ersten Buchstaben eines

Wortes als Code verwenden.

Beispiel 9.1.3 Häufig zitiert ist das „SOUNDEX“ System, das bereits 1918 im Rahmen phonetischer Anwendungen eingeführt wurde. Teilzielsetzung hierbei war, Wörter ähnlicher Aussprache auf denselben Code abzubilden. Bei SOUNDEX wird ein Wort auf eine Symbolfolge, die aus dem ersten Buchstaben und einer Ziffernfolge besteht, abgebildet. Buchstaben werden wie folgt auf Ziffern abgebildet: A,E,I,O,U,H,W,Y auf 0, weiter B,F,P,V auf 1, weiter C,G,J,K,Q,S,X,Z auf 2, weiter D,T auf 3, weiter L auf 4, M und N auf 5 und R auf 6. Nullen werden dann eliminiert und wiederholte Ziffern zusammengezogen. Aus „*Bush*“ wird „B020“ und schließlich „B2“. Dies ergibt einen relativ groben Code.

Beispiel 9.1.4 Das SIMILAREX System ([MG]) ist eine OCR-motivierte Variante von SOUNDEX mit 4 Regeln.

1. Indem man von links nach rechts durch den String geht, bildet man folgende Charakterfolgen der Länge drei wie angegeben ab: iii, iin, nii → M
2. Analog für Länge 2: in,iu,ri,rI,rn,ni,ui,tn → M,
3. Analog für einzelne Charaktere:

$$\begin{array}{rcl} \text{u,h,b} & \rightarrow & \text{n} \\ \text{m,N,H} & \rightarrow & \text{M} \\ \text{1,l,i,r} & \rightarrow & \text{I} \\ \text{o,0,D,C} & \rightarrow & \text{O} \\ \text{v,V,y} & \rightarrow & \text{Y} \\ \text{t} & \rightarrow & \text{f} \\ \text{5} & \rightarrow & \text{S} \\ \text{q} & \rightarrow & \text{g} \\ \text{E} & \rightarrow & \text{F} \\ \text{k} & \rightarrow & \text{K} \end{array}$$

4. Lösche alle verbleibenden Vorkommen von e, c, a, s.

Beispiel 9.1.5 Einen sehr feinen Code erhält man, indem man jedes Wort auf die Menge der in ihm enthaltenen Buchstaben (oder Bigramme bzw. Trigramme) abbildet. Vorkommenshäufigkeiten können hierbei ggf. auch mitberücksichtigt werden. Trigramme oder allgemeiner n-Gramme können in vielerlei Varianten zur Definition von Ähnlichkeitsschlüsseln verwendet werden.

Beispiel 9.1.6 In manchen OCR-Systemen (z.B. [Sin90] oder [AB95]) werden die Buchstaben in (solche mit) Ascenders (b,d,f,h,k,l,t,i), in (solche mit) Descenders (g,p,q,y,j) und in solche ohne Ascenders oder Descenders (a,c,e,m,n,o,r,s,u,v,w,x,z) eingeteilt. Man beachte, daß dieser Typ viel über die visuelle Form des Buchstabens verrät. Nun wird eine vereinfachte Wortrepräsentation (die „Umschlagsrepräsentation“) gebildet, indem man jeder Klasse eine Ziffer zuweist und die Buchstabenfolge durch die Folge der Ziffern ersetzt, wobei gegebenenfalls mehrere gleiche Ziffern zu einer zusammengezogen werden. Wenn wir Ascenders mit Code 2 bezeichnen, Descenders mit Code 3 und die übrigen mit Code 1, so erhalten wir als „Umschlagsrepräsentation“ des Wortes „*example*“ die Folge „1321“. Man kann diese Repräsentation noch durch die Wortlänge ergänzen was dann auf „132107“ führt, mit Null als Abgrenzungssymbol.

9.2 Sinhas Test von Ähnlichkeitsschlüsseln

Ziel der Verwendung eines Schlüssels ist es, zu einer Wörterbuchpartitionierung zu kommen, wo die Partitionsklassen einerseits möglichst fein sind, andererseits die Lexikonwörter, die zu einem Eingabewort einen kleinen Abstand im Sinn der verwendeten Metrik haben, in einer (kleinen Menge von) Partitionsklasse(n) liegen. R.M.K. Sinha in [Sin90] hat für die 10.000 häufigsten Wörter des Brown Korpus [KF67] eine Reihe von — teilweise kombinierten — Ähnlichkeitsschlüsseln in Hinblick auf ihre Eignung zur Wörterbuchpartitionierung empirisch getestet. Die Schlüssel und die damit erzielten Partitionsergebnisse sind die folgenden[1]:

1. *Längenbasiert.* Das Wörterbuch wurde in 12 Klassen eingeteilt, für jede mögliche Wortlänge von 1 bis 11 und für Wörter der Länge ≥ 12. Die größte Klasse enthält 1613 Wörter, die durchschnittliche Klassenstärke ist $820,42$.
2. *Umschlagsbasiert.* Hier wurde das in Beispiel 9.1.6 dargestellte Zerlegungsschema ohne Mitkodierung der Wortlänge verwendet. Es wurden 322 Partitionsklassen erhalten, mit maximal 922 Wörtern und einer Durchschnittsgröße von 30,57 Wörtern.
3. *Umschlags- und längenbasiert.* Kombination der vorausgegangenen Schlüssel, liefert den Durchschnitt der Äquivalenzklassen. Dies ergab 3184 Klassen, die maximale Klassengröße betrug 129, die Durchschnittsgröße lag bei 3,09 Wörtern.
4. 3-*Charakter-Kombinationen-basiert.* Ausgehend von einer durchschnittlichen Wortlänge von 4 und einer 80 prozentigen Korrektheitsrate bei der Buchstabenerkennung wird angenommen, daß in der Regel zumindest drei Buchstaben korrekt erkannt sind. Jedem Wort werden nun als mögliche Codes alle positionellen Trigramme zugewiesen. Zum Beispiel erhält „*dear*“ die vier Codes „*dea*“, „*der*“, „*dar*“, „*ear*“ und erscheint demnach in vier Teilklassen. Die Klasseneinteilung ist keine echte Partition, da unterschiedliche Klassen Wörter gemeinsam haben können. Dies führt als Nebeneffekt zu einer Vergrößerung des Lexikons. Die Zahl der Klassen lag bei 4157, mit maximaler (durchschnittlicher) Größe von 670 $(11,94)$ Wörtern.
5. 3-*Charakter- und längenbasiert.* Es wurden wieder die beiden genannten Codes per Konkatenation kombiniert. Es ergaben sich 16.198 Klassen mit einer maximalen (durchschnittlichen) Größe von 181 $(3,06)$ Wörtern.
6. 3-*Charakter-Kombinationen und umschlagsbasiert.* Hierbei wurden die beiden genannten Codes einfach konkateniert. Es ergaben sich 24.274 Klassen mit einer maximalen (durchschnittlichen) Größe von 125 $(2,04)$ Wörtern.
7. 3-*Charakter-, umschlags- und längenbasiert.* Es wurden die drei genannten Codes per Konkatenation kombiniert. Analog ergaben sich hier 42.787 Klassen mit einer maximalen (durchschnittlichen) Größe von 19 $(1,16)$ Wörtern.
8. 2-*Charakter-Kombinationen-basiert.* Weniger selektiv ist die Methode aus 4. bei der Verwendung von positionellen Bigrammen. Man erhält 688 Klassen mit einer maximalen (durchschnittlichen) Größe von 1414 $(86,43)$ Wörtern.

[1] Bemerkung: die oben genannten Zahlen vertauschen die Angaben aus [Sin90] insofern, als daß dort für Code „3-Charakter-Kombination und umschlagsbasiert“ eine größere Zahl von Partitionsklassen als für den feineren Code „3-Charakter-Kombinationen, umschlagsbasiert und längenbasiert“ angegeben ist, was offenkundig nicht stimmig ist. Daher wurden die beiden Angaben vertauscht. Dasselbe gilt für 2 statt 3.

9. 2-*Charakter- und längenbasiert.* Analog ergaben sich hier 4473 Klassen mit einer maximalen (durchschnittlichen) Größe von 299 $(13,29)$ Wörtern.
10. 2-*Charakter-Kombinationen und umschlagsbasiert.* Analog ergaben sich 13.657 Klassen mit einer maximalen (durchschnittlichen) Größe von 217 $(4,35)$ Wörtern.
11. 2-*Charakter-, umschlags- und längenbasiert.* Analog ergaben sich hier 43.405 Klassen mit einer maximalen (durchschnittlichen) Größe von 24 $(1,37)$ Wörtern.

Eine naive Verwendung von Codes mit Längeninformation macht nur Sinn, wenn die OCR die Zahl der Buchstaben zuverlässig erkennt, was in der Regel nicht der Fall ist. Daher sind die Schlüssel 6 und 10 vorzuziehen. Die Klassengrößen können allerdings insofern irreführend sein, als daß man bei der praktischen Anwendung unter Umständen mehrere Klassen zurückgeben muß, um sicher zu sein, daß alle interessanten Korrekturvorschläge in der Kandidatenliste enthalten sind. Dies wird im nächsten Abschnitt deutlicher.

9.3 Hierarchische Kodierung von Lexika mit Varianten

Die naive Verwendung von Ähnlichkeitsschlüsseln, die wir in der Einleitung skizzierten, ist bei genauerem Hinsehen mit verschiedenen Problemen behaftet, auf die wir im nachfolgenden Anschnitt eingehen. Im Anschluß stellen wir die Verwendung hierarchischer Codes vor, die diese Probleme teileliminiert. Der Inhalt dieses Teilkapitels folgt [dBdBT95].

Probleme der einfachen Kodierung

Wir wollen zunächst genauer betrachten, welche Zielsetzungen mit der Wörterbuchpartitionierung verbunden sind, und welche prinzipiellen Schwierigkeiten dabei auftreten. Als Ausgangspunkt nehmen wir an, daß wir eine symmetrische *„ist-Nachbar-von“-Relation* auf dem Wörterbuch definiert haben, die für beliebige Wortpaare definiert ist. Jedes Wortabstandsmaß definiert beispielsweise eine solche Nachbarschaftsrelation, wenn wir Worte bis zu einer vorgegebenen Abstandsschranke als „benachbart“ behandeln.

Aus abstrakter Sicht kann man für die Partitionierung des Wörterbuchs mit Hilfe eines Codes zwei Qualitätsbedingungen ausmachen.

1. Für jeden Eingabestring sollten *alle* benachbarten Wörter des Lexikons in der Rückgabemenge sein, um eine falsche Korrektur auszuschließen.
2. Für einen gegebenen Eingabestring sollten möglichst wenige Nicht-Nachbarn in der Rückgabemenge sein.

Ist nun ϕ ein Code, so besteht die *naive Idee* darin, bei Eingabe des Wortes W alle Wörter W' auszugegeben mit $\phi(W') = \phi(W)$. Hierbei stellt sich allerdings ein grundlegendes Problem: es gibt in der Regel gar keinen geeigneten Code, der bei

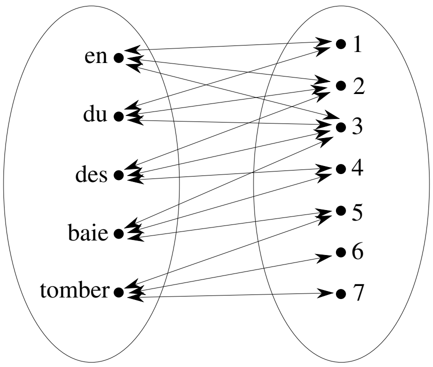
Verwendung der naiven Idee zu guten Ergebnissen im Sinn der obigen zwei Qualitätsmerkmale führt. Der Grund: die transitive Hülle der Nachbarschaftsrelation ist in aller Regel die totale Relation, da wir je zwei Wörter durch eine Kette benachbarter Wörter verbinden können. Gemäß der naiven Idee sollten nun wegen Anforderung 1 je zwei Nachbarn immer denselben Code haben. Damit müssten aber alle Lexikonwörter denselben Code haben. Der Code wäre trivial und nutzlos.

Was wir mit der naiven Idee in der Regel nur erreichen können, ist die Rückgabe einer unvollständigen Liste, bei der viele mögliche Korrekturkandidaten fehlen. Die *verbesserte Idee* ist recht naheliegend: wir verwenden zwar einen Code, setzen aber nicht voraus, daß Nachbarn stets denselben Code haben. Um Anforderung 1 zu erfüllen, geben wir nicht nur die Äquivalenzklasse von W zurück, sondern die aller lexikalischen Nachbarn von W. Wir erhalten zumindest alle lexikalischen Nachbarn des Eingabewortes. Eine on-line Berechnung der lexikalischen Nachbarn wäre natürlich zu zeitaufwendig. Stattdessen führen wir ein Preprocessing auf der Lexikonseite durch. Jedem Lexikoneintrag W werden die Codes aller (beliebigen, also nicht notwendig lexikalischen) Nachbarn von W zugewiesen. Voraussetzung ist natürlich, daß der Code eines jeden Wortes in der Nachbarschaft eines lexikalischen Wortes in der Liste der Codewerte enthalten ist.

Definition 9.3.1 Ein Lexikoneintrag W *steht in Beziehung* zu Code c (und umgekehrt steht c in Beziehung zu W) genau dann, wenn es einen Nachbarn W' von W mit Code c gibt.

Vorsicht: man sollte diese Relation nicht mit „W hat Code c“ verwechseln! Ist nun ein Eingabewort W gegeben, so wird der Code c von W berechnet und alle Lexikoneinträge, die in Beziehung zu c stehen, bilden die Rückgabemenge $R(W)$. Es bleibt dem Leser als Aufgabe überlassen, zu zeigen, daß $R(W)$ stets alle lexikalischen Nachbarn von W umfaßt!

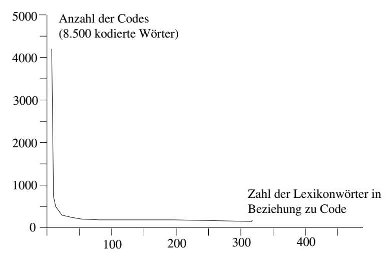
Beispiel 9.3.2 Zur Illustration betrachten wir hier und in den nachfolgenden Beispielen ein Wort V als benachbart zu einem Wort W genau dann, wenn W aus V vermöge einer Substitution, Löschung, Einfügung oder Transposition zu erhalten ist. Als Code verwenden wir die Wortlänge. Hat der Eingabestring W die Länge n, so besteht die Rückgabemenge $R(W)$ — der verbesserten Idee folgend — aus allen Lexikoneinträgen der Lange $n-1$, n oder $n+1$. Das triviale Beispiellexikon umfasse die Wörter „*en*“, „*du*“, „*des*“, „*baie*“ und „*tomber*“. In der nachfolgenden Abbildung ist die „steht in Beziehung zu Code“-Relation durch Pfeile angedeutet:



Bei Eingabewort „*de*“ werden alle Wörter mit Nachbarn des Codes 2 zurückgegeben, also „*du*“, „*en*“, „*des*“.

Leider hat auch diese Methode mit einem Problem zu kämpfen, das in [dBdBT95] durch ein kleines Experiment verdeutlicht wird. Hierbei wurde eine Variante des Soundex Codes verwendet, um ein Wörterbuch mit 10.000 Wörtern aus wissenschaftlichen Artikeln zu partitionieren. Die wichtigsten Ergebnisse aus negativer Sicht:

- Unter den auftretenden Codes sind zwar 80% in Beziehung zu weniger als 25 Wörtern (gute Größe bei gegebenem Lexikon). Knapp 6% der Codes stehen aber zu mehr als 100 (bis zu über 300) Wörtern in Beziehung. Für diese Codes ergibt sich ein schlechtes Verhalten der Partitionierung bzw. Rückgabemengen. Die nachfolgende Abbildung deutet den Zusammenhang an.



- Das tatsächlich in der Praxis auftretende Problem ist noch etwas verschärft: gibt man ein zufälliges Wort (aus dem Lexikon, beliebig, oder aus dem Lexikon mit Fehler) ein und fragt nach der durchschnittlichen Größe der erhaltenen Rückgabemenge, so ist diese mit 210 Wörtern deutlich größer als die durchschnittliche Zahl der Wörter, die zu einem Code in Beziehung stehen, da häufige Codes überwiegen.

Natürlich kann man versuchen, durch Verwendung eines feineren Codes ein besseres Resultat zu erzielen. Dies kann jedoch zu sehr großen Bildmengen der Codes führen. Ein Beispiel soll dies illustrieren: Im Extremfall könnte man ja die Identität als Code verwenden. Dann müßte man aber alle Wörter als Codewerte speichern, die einen (nicht notwendig lexikalischen!) Nachbarn eines Lexikonwortes darstellen. Die Zahl dieser Wörter kann offenkundig extrem groß werden. Ein Ausweg besteht darin, die am häufigsten verwendeten Codes hierarchisch weiterzuunterteilen.

Hierarchisch kodierte Lexika mit Varianten

Das Ziel bei der hierarchischen Kodierung besteht darin, Rückgabemengen zu konstruieren, die zu einem beliebigen Eingabewort stets alle seine lexikalischen Nachbarn (und ggf. weitere) Wörter enthalten, gleichzeitig aber eine kleine durchschnittliche Größe aufweisen.

Definition 9.3.3 Seien ϕ_1 und ϕ_2 Codes des Lexikons L, seien $\sim_1^L$ und $\sim_2^L$ die zugehörigen Äquivalenzrelationen auf L. Dann heißt ϕ_2 *feiner* als ϕ_1 bezüglich L (bzw. ϕ_1 *gröber* als ϕ_2) gdw. jede $\sim_2^L$-Äquivalenzklasse enthalten ist in einer $\sim_1^L$-Äquivalenzklasse.

Wir schreiben $\phi_1 \leq_L \phi_2$ gdw. ϕ_2 feiner ist als ϕ_1 bezüglich L.

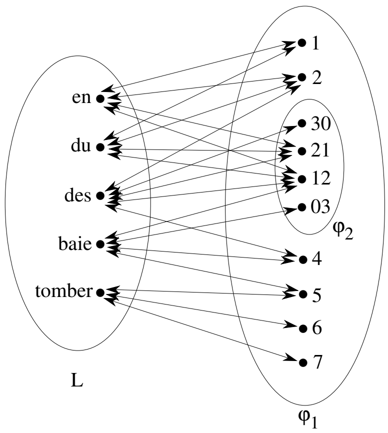
Beispiel 9.3.4 Seien ϕ_1 die Abbildung, die einem Wort $w \in L$ die Wortlänge n zuordnet. Seien ϕ_2 die Abbildung, die einem Wort $W \in L$ die das Tupel $\langle v(W), k(W)\rangle$

zuordnet, wo $v(W)$ (resp. $(k(W))$ die Zahl der Vorkommen von Vokalen (resp. Konsonanten) in W angibt. Dann ist ϕ_2 feiner als ϕ_1.

Definition 9.3.5 Seien $\phi_1 \leq_L \cdots \leq_L \phi_n$ Codes des Lexikons L, seien $C_1, \ldots, C_n$ die Bildmengen von $\phi_1, \ldots, \phi_n$. Eine *hierarchisch kodierte Variante von L zu den Codes $\phi_1, \ldots \phi_n$ mit Grenzwerten* $k_1, \ldots, k_{n-1}$ speichert die Mengen $L, C_1, \ldots, C_n$. Jeder Lexikoneintrag enthält zusätzlich eine Menge von bidirektionalen Pointern zu verschiedenen Codes. Für $i < n$ darf die Zahl der mit einem Code $c_i \in C_i$ verpointerten Wörter (die Rückgabemenge von c_i) den Grenzwert k_i nicht überschreiten. Folgende Bedingung müssen gelten: Ist W mit c_i verpointert, so gibt es einen Nachbarn W' von W mit ϕ_i-Code c_i. Ist W' ein Nachbar des Lexikonworts W, so existiert genau ein i wo W mit $\phi_i(W')$ verpointert ist.

Die Idee hinter der Definition ist einfach: wenn für $i < n$ mehr als k_i Wörter zu einem Code $c_i \in C_i$ in Beziehung stehen, wird c_i vermöge ϕ_{i+1} rekodiert. Die Pointer auf c_i entfallen dann. Dies endet, wenn man den feinsten Code C_n erreicht hat. Es gibt kein Rekodieren der Codes von C_n.

Beispiel 9.3.6 Als Fortsetzung von Beispiel 9.3.4, wo wir eine Code-Hierarchie der Tiefe 2 eingeführt haben, betrachten wir die hierarchische Kodierung des Lexikons aus Beispiel 9.3.2, wo wir den Grenzwert k_1 auf 3 setzen. Damit ist jeder Code im Bild von ϕ_1, der zu mehr als drei Wörtern in Beziehung steht, weiter zu unterteilen, indem man das Bild durch die möglichen Bildwerte unter ϕ_2 ersetzt. Es wird daher „3“ wie folgt ersetzt.



Wir setzen nachfolgend voraus, daß die Bildmengen verschiedener Codes disjunkt sind. Für ein Bildelement c steht $|c|$ für die Zahl der Lexikonwörter, die zu Code c in Beziehung stehen, und L_c für die Menge dieser Wörter.

Die **Berechnung der präselektierten Wörter zum Eingabewort** W mit einem hierarchisch kodierten Lexikon L zu den Codes $\phi_1, \ldots, \phi_n$ ist wie folgt: Mit $i = 1$ beginnend, solange $c_i := \phi_i(W)$ rekodiert ist (da $|c_i| > k_i$) gehe zu nächst feinerem Code ($i := i + 1$). Ist schließlich $\phi_i(W)$ nicht rekodiert (da $|c_i| \leq k_i$), so ist L_{c_i} die Menge der präselektierten Wörter zur Eingabe W.

Beispiel 9.3.7 Als Fortsetzung von Beispiel 9.3.6 betrachten wir das Eingabewort „*de*“. Es ist $\phi_1(de) = 2$ nicht rekodiert, daher erhalten wir die Rückgabemenge mit den Wörtern „*en*“, „*du*“ und „*des*“. Für die Eingabe „*oui*“ ist $\phi_1(oui) = 3$ rekodiert. Es ist $\phi_2(oui) = 03$ nur zu „*baie*“ in Beziehung, daher ist dies das einzige Wort in der Rückgabemenge.

Eine einfache Überlegung zeigt, daß die Antwortmenge bei der so beschriebenen hierarchischen Kodierung stets tatsächlich alle lexikalischen Nachbarn des Eingabeworts W umfaßt. Sei V ein lexikalischer Nachbar von W. Dann ist umgekehrt auch W ein lexikalischer Nachbar von V. Es ist damit der Code von W in Beziehung zu V. Nach Voraussetzung führt damit ein Pointer von V zum Code von W. Es liegt damit V in der Rückgabemenge (des Codes von) W.

Grenzen der Methode und Abschlußbemerkungen

In [dBdBT95] wird ein Verfahren zur Berechnung hierarchisch kodierter Lexika angegeben. Es wird anhand von Experimenten gezeigt, wie man mit hierarchisch kodierten Lexika die durchschnittliche Größe der bei einem Eingabewort zurückgegebenen Wortmenge deutlich reduzieren kann.

Für 8.500 kodierte Wörter des Beispiellexikons erhält man mit dem Soundex-Code alleine durchschnittlich 216 Wörter in der Rückgabemenge. Mit verschiedenen Code-Hierarchien ergeben sich Durchschnittswerte von $22 - 36$ Wörtern.

Ein Nachteil ist der vergrößerte Speicherbedarf, der durch die Code-Bildmengen und die Pointer resultiert. Ein Nachteil entsteht insbesondere, wenn die Code-Bereiche aufgrund ihrer Größe im Sekundärspeicher gehalten werden. Bei Eingabe eines Wortes erfolgen dann unter Umständen soviele Zugriffe auf die Festplatte, wie es Codes in der Hierarchie gibt. Dies kann zu einem Effizienzverlust führen.

Leider bleibt in der Arbeit unklar, wie groß der durchschnittliche Anteil von Nichtnachbarn in der Antwortmenge ist, und wie ,,weit entfernt“ im Sinn der Nachbarschaftsrelation die Wörter der Rückgabemenge sein können.

Bei wirklich großen Lexika sind der Methode natürliche Grenzen gesetzt. Da es dort zu einzelnen Wörtern unter Umständen sehr viele lexikalische Nachbarn gibt, kann man die Größe der Rückgabemengen — die ja immer alle Nachbarn enthalten sollen — nicht beliebig beschränken. Generell macht es wenig Sinn, Grenzwerte k_i zu verwenden, wo häufig mehr als k_i Lexikonwörter zu einem potentiellen Eingabewort benachbart sind. Dies würde lediglich zu einer großen Zahl von Wörtern führen, deren Code durch den feinsten möglichen Code gegeben wird, wo wir die Größe der Rückgabemengen nicht mehr weiterreduzieren können.

Kapitel 10

Verwendung von Abstandsmaßen und Ähnlichkeitsschlüsseln bei der lexikalischen Wortkorrektur

Wenn bei der OCR-Erkennung ein elektronisches Lexikon adäquaten Unfangs zur Verfügung steht, kann man die fraglichen Wörter, die von der OCR Erkennung geliefert werden, im Lexikon nachschlagen und auf diese Weise gegebenenfalls verifizieren. Tritt das Wort nicht im Lexikon auf, so kann man mittels des Lexikons mögliche Korrekturvorschläge ermitteln. Hierzu werden die Lexikonwörter mit dem fraglichen Wort verglichen, indem man ihren Abstand in Bezug auf eine geeignete Metrik berechnet. Die Lexikoneinträge mit dem geringsten Abstand zum Eingabewort stellen dann die besten Korrekturvorschläge dar. Schon bei Lexika einer vergleichsweise geringen Größe ist es allerdings bei den meisten Anwendungen aus Effizienzgründen nicht möglich, für das fragliche Eingabewort explizit den Abstand zu jedem Lexikonwort zu berechnen.

Ein erster, naheliegender Ausweg besteht darin, zunächst mit einem geeigneten Vorfilterungsschritt durch Anwenden eines Ähnlichkeitsschlüssels eine kleine „Kandidatenliste“ aus dem Lexikon auszusondern, in der jedoch alle interessanten Korrekturvorschläge enthalten sind. Ein geeignetes Abstandsmaß wird dann verwendet, um die Ähnlichkeit der Wörter der Kandidatenliste mit dem Eingabewort zu ermitteln. Diesem Muster folgt die unten beschriebene Methode von Takahashi, Itoh, Amano und Yamashita [TIAY90].

Eine Variante dieses Vorgehens besteht darin, zur Vorauswahl der Kandidatenliste ein „grobes“ Wortabstandsmaß zu nehmen, wo mit Hilfe spezieller Indexierungstechniken die zum Eingabewort nächsten lexikalischen Nachbarn sehr effizient ermittelt werden können. Danach wird — wie oben — ein feineres Maß zur Auswahl um zum Ranking der Kandidaten verwendet. Beispiele dieser Vorgehensweise liefern das unten beschriebene Verfahren von Owolabi und McGregor [OM88] und die von Kim und Shawe-Taylor [KST92] entwickelte Variante.

Eine dritte Möglichkeit besteht darin, ausgehend vom mißgestalteten Eingabewort eine Menge in Frage kommender „Rückkorrekturen“ zu berechnen, und die Wörter der so erhaltenen Kandidatenliste auf ihre Lexikalität zu überprüfen. Dies ist insbesondere interessant, wenn die OCR als Ausgabe nicht nur das „erkannte“ Wort, sondern auch die möglichen Alternativen der einzelnen Buchstaben oder Buchstabenpaare liefert. Bei der Berechnung sinnvoller Rückkorrekturen können als Filter statistische Beobachtungen über mögliche Fehler bei der Symbolerfassung und über wahrscheinliche Buchstabenabfolgen eingesetzt werden. Allerdings ist es im allgemeinen nicht leicht, zuverlässige Informationen über die möglichen Rückkorrekturen und ihre Wahrscheinlichkeiten bereitzustellen. Bei der Ermittlung der besten Korrekturvorschläge in der Kandidatenliste kann das vorgegebene universelle Wortabstandsmaß mit Hilfe der Korrekturwahrscheinlichkeiten relativiert oder ergänzt werden. Beispiele für dieses Vorgehen liefern [Gor] sowie das Verfahren von Shi und Pavlidis [SP].

Eine weitere, nur sporadisch diskutierte mögliche Vorgehensweise besteht darin, das Wörterbuch in geeigneter Weise durch Varianten der echten Wörter zu vergrößern, so daß es möglich wird, auch fehlerhafte Wörter direkt nachzuschlagen. Ist das Wörterbuch der Varianten in geeigneter Weise geordnet, so kann man zunächst ein fehlerhaftes Wort nachschlagen und die benachbarten Wörter als Kandidatenliste oder als fertige Korrekturvorschläge ausgeben. Eine Technik dieser Art wurde in [ZD95] betrachtet.

Die vier Varianten der Verwendung von Lexika, Ähnlichkeitsmaßen und Ähnlichkeitsschlüsseln sind im Überblick in Abbildung 10.1 dargestellt. Eine weitere attraktive Möglichkeit, die auf der Repräsentation von Wörterbüchern in Form von deterministischen endlichen Automaten beruht, werden wir in einem späteren Abschnitt angeben.

10.1 Verfahren von Takahashi, Itoh, Amano und Yamashita

Beim Verfahren von Takahashi, Itoh, Amano und Yamashita [TIAY90] wird davon ausgegangen, daß für die auftretenden Symbole eine lineare Ordnung bekannt ist, die die Rangfolge der Häufigkeiten der Symbole beschreibt. Ein beliebiges Wort wird durch die vier seltensten in ihm enthaltenen Symbole charakterisiert. Treten weniger Symbole auf, so werden gegebenenfalls mehrfach Blanks ergänzt. Allerdings wird nicht die so resultierende Vierermenge direkt als Schlüssel verwendet. Da im Rahmen der OCR-Erfassung Fehler auftreten können, wird jede mögliche dreielementige Teilmenge als eine Charakterisierung oder als Attribut des Wortes verwendet.

Beispiel 10.1.1 Die vier seltensten Buchstaben, die im Wort *example* auftreten, seien *x, p, m, l*. Dann erhält *example* die Attribute $\{x, p, m\}$, $\{x, p, l\}$, $\{x, m, l\}$ und $\{p, m, l\}$.

Ein von der OCR geliefertes Eingabewort führt auf diese Weise zu vier Schlüsseln bzw. Attributen. Für jedes Attribut werden alle Lexikonwörter mit demselben Attribut zurückgegeben, wobei Wörter ausgefiltert werden, deren Länge von der Länge des Eingabeworts über ein vorgegebenes Toleranzmaß hinaus abweicht. Die so erhaltenen vier Teilmengen des Lexikons werden —nach Elimination von mehrfach genannten Wörtern — als Kandidatenliste verwendet.

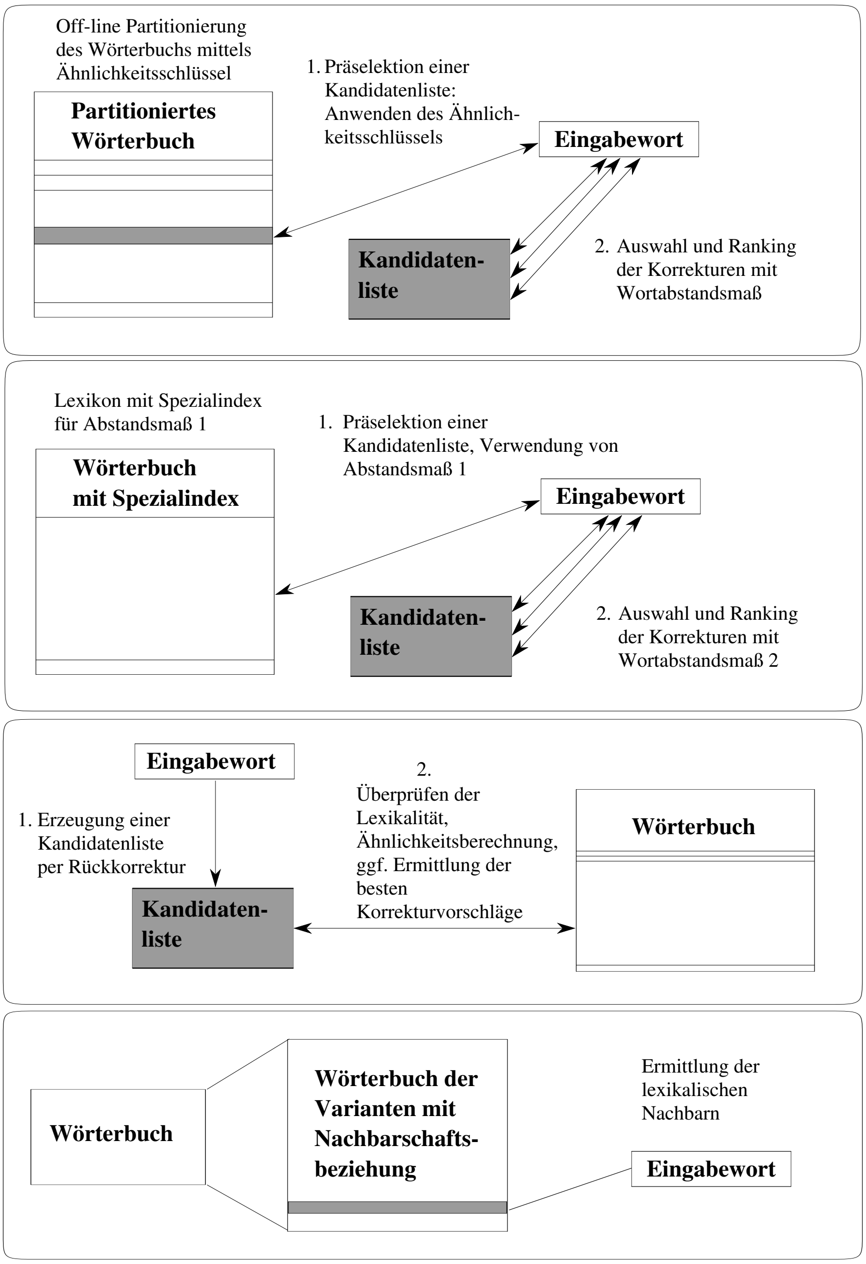


Abbildung 10.1: Vier Varianten der lexikalischen Wortkorrektur.

Beispiel 10.1.2 Beträgt die tolerierte Abweichung der Wortlänge 1, und liefert die OCR das Eingabewort *exanple* der Länge 7, so sind alle Lexikonwörter mit den Attributen $\{x, p, n\}$, $\{x, p, l\}$, $\{x, n, l\}$ und $\{p, n, l\}$ und einer Länge zwischen 6 und 8 in die Kandidatenliste aufzunehmen.

Natürlich kann man anstelle der vier seltensten Symbole eines Worts auch die k seltensten Symbole verwenden, und dann anstelle der dreielementigen Teilmengen die Teilmengen der Kardinalität $l < k$ verwenden. Wenn man von der Möglichkeit ausgeht, daß mehr als einer der seltenen Buchstaben verstümmelt sein kann, so ist von dieser Möglichkeit Gebrauch zu machen. Allerdings scheiden in der Praxis viele Paare (l, k) aus, da sie den einzelnen Wörtern zuviele Attribute zuordnen.

In [TIAY90] werden zur weiteren Reduktion der Kandidatenmengen auch Alternativen betrachtet, wo zusätzlich das erste (oder letzte) Symbol des von der OCR gelieferten Eingabewortes als korrekt betrachtet wird.

Die angegebenen Test beziehen sich auf englische Wörterbücher mit 11.000 bzw. 50.000 Wörtern. Für das erstgenannte kleine Wörterbuch enthalten die Kandidatenlisten der angegebenen Beispiele einige zig Wörter. Beim größeren Wörterbuch wird die durchschnittliche Zahl der Kandidaten pro Attributsklasse mit $134, 4$ angegeben. Die Kandidatenlisten sind demnach etwa um den Faktor vier größer.

Zur Feinsuche und zum Ranking der Kandidaten werden in der zweiten Phase das Levenshtein-Maß und eine Variante, die einfacher zu berechnen ist, betrachtet. Die Arbeit schließt mit einigen Betrachtungen über das Design von interaktiven Systemen zur Fehlerkorrektur.

10.2 Verfahren von Owolabi und McGregor

O. Owolabi und D.R. McGregor [OM88] gaben ein Wortkorrekturverfahren an, bei ein Trigrammabstandsmaß zur Präselektion einer Kandidatenliste aus dem Lexikon verwendet wird. Im Anschluß wird für jeden Korrekturkandidaten der Levenshtein-Abstand zum Eingabewort berechnet. Wir beschreiben zunächst eine Korrekturmethode, die lediglich als Ausgangspunkt für das Verfahren von Owolabi und McGregor dient.

Korrektur mit invertierten n-Gramm-Matrizen

Bei dieser Methode werden für ein Eingabewort W diejenigen Lexikonwörter ausgegeben, die zu W den kleinsten Trigrammabstand haben. Das Abstandsmaß

$$d(V, W) := \frac{2c}{k + l}$$

ist im Prinzip das in Beispiel 8.1.3 angegebene, wobei k (l) hier die Zahl der Trigramme von V (W) darstellt. Um das Auffinden der nächsten Nachbarn von W zu beschleunigen, werden die Lexikonwörter zunächst in alphabetischer Reihenfolge mit Zahlen $1, \ldots, n$ durchnummeriert. Nun wird für jedes der 26^3 möglichen Trigramme v ein Bitarray $A(v)$ der Länge n angelegt. Der Eintrag $A(v)[i]$ ist 1 genau dann, wenn das Trigramm v im i-ten Lexikonwort auftritt, 0 sonst. Die Arrays werden vorab beim Einlesen des Lexikons berechnet. Durch eine einfache Adressenkalkulation

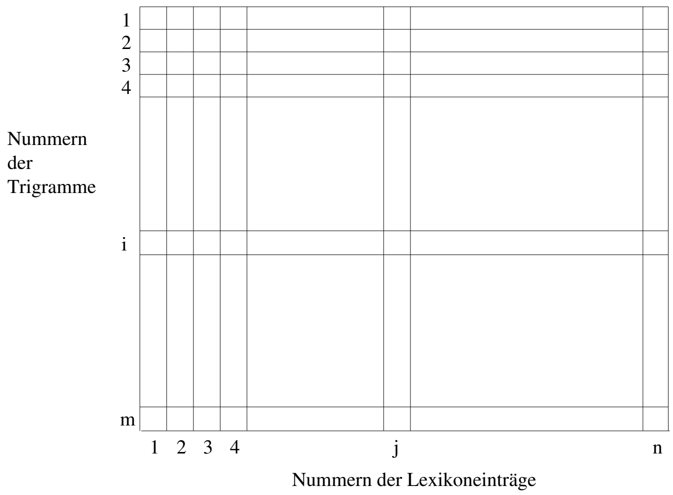


Abbildung 10.2: Matrix der n-Gramm-Vorkommen.

kann man jedem Trigramm eine Zeilennummer zuordnen und die vertikal angeordnete Folge aller Arrays als eine Matrix lesen, wo die Spalten den Lexikoneinträgen, die Zeilen den Trigrammen entsprechen. Dies ist in Abbildung 10.2 angedeutet, wo der Eintrag des Feldes i, j anzeigt, ob Trigramm i in Wort j vorkommt. Bei Eingabe eines Worts W werden nun die Arrays $A(v)$ für alle in W auftretenden Trigramme komponentenweise addiert. Als Ergebnis erhält man einen Vektor $A(W)$, wo die i-te Komponente $A(W)[i]$ angibt, wieviel Trigramme aus W im i-ten Lexikonwort auftreten. Nun wird das oben angegebene Abstandsmaß verwendet, um ein Ranking der Korrekturvorschläge zu erreichen.

Der Hauptvorteil der Methode besteht in der schnellen Selektion vernünftiger Korrekturkandidaten. Allerdings führt das Ranking mit dem Trigrammabstandsmaß zu schlechteren Erbenissen als ein Ranking mit dem Levenshtein-Abstand. Auch ist der Speicherbedarf für die Matrix aller Arrays $A(v)$ recht groß.

Zwei-Phasen Methode

In [OM88] werden die oben erwähnten Defizite durch ein abgeändertes Verfahren mit zwei Phasen eliminiert. In der ersten Phase wird mittels einer Matrix einer verringerten Dimension eine Teilmenge des Lexikons ausgewählt, die neben den interessanten Korrekturvorschlägen auch noch eine Reihe ungeeigneter Wörter enthalten kann. In der zweiten Phase wird eine längensensitive Abwandlung des Levenshtein-Maßes dazu verwendet, um mittels des Wagner-Fisher Verfahrens eine kleine Menge qualitativ hochwertiger Korrekturvorschläge auszusondern.

Phase 1. Präselektion einer Kandidatenliste mit n-Gramm Filter. Die Matrix aller Trigrammvorkommen (cf. Abbildung 10.2) wird zunächst in horizontaler Richtung dadurch komprimiert, das nun einer Gruppe von 16[1] aufeinanderfolgenden Wörtern des Lexikons eine Spalte zugewiesen wird. Ein Eintrag 1 in Zeile i besagt nun, daß Trigramm i in *zumindest einem* der betreffenden 16 Wörtern auftritt. In einem zweiten Schritt wird die Matrix auch noch in vertikaler Richtung kom-

[1] Dieser Wert wurde heuristisch ermittelt, bei anderen Lexikagrößen könnten andere Werte vorzuziehen sein.

primiert: hierzu werden die Zeilen —unter Verwendung einer Art Hash-Funktion — in 511 Partitionsklassen eingeteilt, Es gibt dann lediglich 511 Zeilen anstatt $26 \cdot 26 \cdot 26 = 17.576$. Ein Eintrag an der Stelle i, j besagt nun, daß zumindest eines der Trigramme aus Klasse i in einem Wort der Klasse j vorkommt.

Die Präselektion einer Kandidatenliste für das Eingabewort W erfolgt ähnlich wie vorher. Zunächst werden die Trigrammklassen der in W vorkommenden Trigramme berechnet. Diese definieren eine Menge von Zeilen der komprimierten Matrix. Durch Addition der Zeilen ergibt sich ein Vektor, wo der j-te Eintrag angibt, wieviele der Trigramme aus den selektierten Trigrammklassen in Partitionsklasse j des Lexikons auftreten. Für jede Partitionsklassen j, wo die Zahl

$$\frac{\text{Zahl der gemeinsamen Trigramme}}{\text{Zahl der Trigramme in } W}$$

eine heuristisch zu bestimmende anwendungsabhängige Schranke überschreitet, werden die entsprechenden Wörter zur Kandidatenliste hinzugefügt. Das tatsächlich verwendete Verfahren fügte allerdings sowohl bei den Lexikonwörtern wie auch beim Eingabewort zwei Platzhaltersymbole am Anfang hinzu. Aus anderer Darstellung bedeutet dies, daß auch der erste Buchstabe, und die ersten zwei Buchstaben, als ,,Trigramm“ behandelt werden. Dadurch ergibt sich insbesondere für kurze Wörter ein besseres Maß an Übereinstimmung.

Die so berechnete Kandidatenliste kann natürlich Wörter enthalten, die nur sehr wenige Trigramme (oder gar keines) mit W teilen. Da die Lexikonwörter in einer Klasse mit 16 Wörtern aufgrund der lexikographischen Ordnung des Lexikons zumeist ähnlich sind, sind die Unterschiede in der Trigrammstruktur für die Klassenmitglieder in der Regel nicht zu dramatisch. Daher enthält die Rückgabemenge nicht allzuviel nutzlose Wörter. Umgekehrt ist jedes zu W hinreichend ähnliche Wort mit Sicherheit in der Kandidatenliste enthalten. Die Auslese der wirklich geeigneten Korrekturkandidaten für W und deren Ranking erfolgt im zweiten Schritt.

Phase 2. Auslese und Ranking der Korrekturkandidaten. Sind $V_1, \ldots, V_k$ die Elemente der Rückgabemenge aus Phase 1, so wird nun der Levenshtein-Abstand d_i von W zu jedem der Wörter V_i berechnet. Es wird nun der Wert

$$s_i := \frac{(100 \cdot (a + b_i - 2 \cdot d_i))}{a + b_i}$$

berechnet, wo $a + b_i$ die Summe der Wortlängen von W und V_i ist. Es ist s_i eine Variante des Levenshtein-Abstands, die Wortlängen mit einbezieht. Sofern nun s_i eine vorbestimmte Schranke übersteigt, so wird V_i als Korrekturvorschlag ausgegeben. Gegebenenfalls kann man die Korrekturvorschläge mittels des s_i-Wertes ordnen.

Ergebnisse

Durch die Kompression der Trigramm-Vorkommensmatrix ergab sich eine Speicherplatzreduktion von 45 Mbytes auf 80 Kbytes. Die 2-Phasenmethode ist aber auch effizient: bei einem Test mit einer Gruppe von Eingabewörtern (Lexikongröße 25.000 Wörter) betrug die durchschnittliche Rechenzeit, um die besten Korrekturvorschläge für ein Eingabewort allein mittels des Wagner-Fisher Verfahrens zu berechnen, 1min 37s. Bei der n-Gramm Methode ergab sich ein Wert von 1,5s, bei der Zwei-Phasen-Methode ein Wert von 0,5s.

10.3 Verfahren von Kim und Shawe-Taylor

Eine Variante des Verfahrens wurde von J.Y. Kim und J. Shawe-Taylor in Bezug auf seine Komplexität untersucht [KST92]. Das Verfahren ist sehr ähnlich, verwendet lediglich eine etwas modifizierte Datenstruktur anstelle der Matrix der n-Gramm-Vorkommen. Wie bei Owolabi und McGregor wird das Lexikon in Klassen von Wörtern, sogenannte „Sektionen“ partitioniert. Die Gesamtzahl von Symbolen ist in jeder Sektion (annähernd) dieselbe.

Die Menge aller im Lexikon auftretenden Trigramme wird in einem Trie organisiert. Die Pfade des Tries entsprechen also genau den im Lexikon auftretenden Trigrammen. Jedes Blatt des Tries ist mit einer sortierten Liste versehen, die Pointer zu all denjenigen Lexikon-Sektionen enthält, in denen das Trigramm auftritt. Die Liste der Pointer ist nach absteigender Sektionennummer sortiert.

Beim Matching-Prozess werden zunächst die Trigramme des Eingabewortes berechnet, die Trigramme werden im Trie nachgeschlagen. Die zugehörigen Pointerlisten werden nun im Sinn einer Multimengenvereinigung in sortierter Weise aufaddiert. Man erhält auf diese Weise eine nach Nummern sortierte Folge von Sektionen, wobei für jede auftretende Sektion angegeben ist, wiewiele der Trigramme des Eingabeworts dort auftreten. Da die Eingabelisten dieses Additionsvorgangs sortiert sind, erfolgt die Addition mit linearem Aufwand. Jede Sektion des Lexikons, die mehr als eine vorzugebende Anzahl von Trigrammen enthält, wird präselektiert. Im Anschluß wird wieder eine Variante des Wagner-Fisher-Algorithmus verwendet, um die Korrekturvorschläge zu generieren. Kim und Shawe-Taylor zeigen, daß die erwartete Zeitkomplexität für das Berechnen einer Korrekturliste bei $O(m \cdot log_2 N)$ liegt, wo m die Länge des Eingabeworts und N die Zahl der Symbole im Lexikon ist.

10.4 Verfahren von Shi und Pavlidis

Beim Verfahren von Shi und Pavlidis [SP] wird davon ausgegangen, daß die OCR bei der Erkennung eines Worts für einzelne Buchstaben oder Buchstabenkombinationen verschiedene Alternativen liefert, die mit einem Konfidenzwert gewichtet sind. Die Alternativen ergeben sich daraus, daß der korrekte Buchstabe substituiert wurde (z.B. „t“ statt „l“), daß der korrekte Buchstabe zerteilt wurde („li“ statt „h“), oder daß ein Buchstabenpaar zusammengezogen wurde („m“ statt „rn“). Durch Kombination der jeweiligen Alternativen ergibt sich eine zunächst oft eine Vielzahl möglicher „Vorkandidaten“, die als Ausgaberesultat in Frage kommen.

Beispiel 10.4.1 Die nachfolgende Tabelle stellt ein erkanntes Wort „*inslilulion*“ und die für einzelne Teilbuchstaben oder Ketten angebotenen Alternativen (hier ohne Konfidenzwert) dar.

Erkanntes Wort	*in*	*s*	*li*	*l*	*u*	*li*	*on*
Alternativen	*m*		*ti*, *h*	*t*		*ti*, *h*	

Wir erhalten daraus $2 \cdot 3 \cdot 2 \cdot 3 = 36$ Vorkandidaten.

Das Lexikon wird zunächst einmal indirekt eingesetzt: aus dem Lexikon wird ein Filter entwickelt, mit dem diese Menge der Vorkandidaten verkleinert wird. Wir erhalten die kleine Menge der eigentlichen Kandidaten. In konkreten Fall werden

anti-	*geo-*	*meta-*	*out-*	*semi-*
auto-	*hyper-*	*micro-*	*over-*	*stereo-*
bio-	*intra-*	*mid-*	*photo-*	*sub-*
counter-	*inter-*	*mis-*	*poly-*	*super-*
dis-	*iso-*	*mono-*	*pre-*	*thermo-*
electro-	*kilo-*	*multi-*	*pseudo-*	*ultra-*
fore-	*magneto-*	*non-*	*psycho*	*under-*
-able	*-ful*	*-ists*	*-led*	*-ness*
-bility	*-hood*	*-ities*	*-less*	*-tion*
-ed	*-ist*	*-ity*	*-like*	
-es	*-istic*	*-ize*	*-ment*	

Abbildung 10.3: Häufige Affixe der englischen Sprache.

aus dem Lexikon für jede mögliche Anfangsposition in einem Lexikonwort die dort auftretenden Trigramme berechnet.

Beispiel 10.4.2 Enthält das Lexikon die Wörter *the*, *first*, *assumed*, *circumstance*. Die legalen Trigramme der möglichen Wortpositionen sind dann

1	*the, fir, ass, cir*
2	*irs, ssu, irc*
3	*rst, sum, rcu*
4	*ume, cum*
5	*med, ums*
6	*mst*
7	*sta*
8	*tan*
9	*anc*
≥ 10	*nce*

Zusätzlich verwenden Shi und Pavlidis bei der Aussonderung der Kandidaten eine Liste häufiger Präfixe und Suffixe. Bei der nachfolgenden Illustration nehmen wir an, daß diese die in Abbildung 10.3 aufgeführten Affixe umfaßt. Bei der Fehlerkorrektur wird nun wie folgt vorgegangen.

Schritt 1, Kandidatenerzeugung. Aus der Menge der Alternativen werden mittels eines Backtracking-Verfahres die Kandidaten erzeugt. Hierbei ersetzt man, vom Wortende beginnend, die erkannten Anteile durch die jeweiligen Alternativen. Bei jeder Ersetzung wird überprüft, ob sich legale positionelle Trigramme ergeben. Andernfalls wird der String gestrichen.

Beispiel 10.4.3 Durch Ersetzung des erkannten Wortendes *-lion* durch *-tion* und *-hon* erhalten wir die Wörter *inslilulion*, *inslilution* und *insliluhon*. Alle drei Wörter bestehen aus legalen Trigrammen.

Bei Wörtern, die eine bestimmte Länge überschreiten, wird nachgeschaut, ob man durch eine Alternative ein häufiges Suffix erhalten kann. In diesem Fall werden die anderen Alternativen verworfen. Im obigen Beispiel bleibt nur das Wort *inslilution*. Die verbleibende Alternativenmenge hat die Form

Erkanntes Wort	in	s	li	l	u	ti	on
Alternativen	m		ti, h	t			

und führt nach zwei Schritten auf die sechs Strings *inslilution*, *inslitution*, *instilution*, *institution*, *inshlution* und *inshtution*. Durch den Trigrammtest werden die letzten zwei Alternativen eliminiert. Bei den verbleibenden vier Strings wird noch die letzte Alternative probiert. Wortanfänge mit *ms-* fallen aber durch den Trigrammtest. Wir erhalten also am Ende von Schritt 1 die vier möglichen Kandidaten *inslilution*, *inslitution*, *instilution*, *institution*.

Schritt 2, Erstellung der Rangliste. Jeder Kandidat wird mit drei Kosten versehen. An erster Stelle wird geprüft, ob das Wort im Lexikon vorkommt oder nicht. An zweiter Stelle steht der gesamte Konfidenzwert, der sich durch die Kondidenzwerte der einzelnen Alternativen zusammensetzt. An dritter Stelle wird eine Distanz zwischen dem ursprünglich erkannten Wort und dem jeweiligen Kandidaten gewertet. Gemäß der drei Kriterien, die eine lexikographische Ordnung definieren, wird nun die Kandidatenliste sortiert. Ist eine eindeutige Ausgabe notwendig, so wird der ranghöchste Kandidat gewählt.

Kritikpunkte

Das Verfahren kann aufgrund der Bevorzugung häufiger Affixe nicht wirklich garantieren, daß das korrekte Wort unter den Korrekturvorschlägen ist. Darüberhinaus muß die oben angegebene Verwendung des Trigrammfilters abgeändert werden, da sich in der gegenwärtigen Form folgendes Problem ergibt: Ist bei dem von der OCR gelieferten Wort bereits beim Wortanfang ein Fehler aufgetreten, der sich auf die Wortlänge, insbesondere damit auf die Positionen der Trigramme auswirkt, so können aufgrund der Verschiebung und der Vorgehensweise von rechts nach links Trigramme als illegal eingestuft werden, die völlig legal (aber momentan noch verschoben) sind. Dies kann dazu führen, daß gute Korrekturkandidaten eliminiert werden.

10.5 Tests von Ähnlichkeitsschlüsseln und Abstandsmaßen von Zobel und Dart

Zum Abschluß des Kapitels gehen wir auf eine Arbeit ein, wo verschiedene der vorher erwähnten Techniken verglichen werden. Justin Zobel und Philip Dart [ZD95] untersuchten, wie man am besten Techniken der Präselektion zur Ermittlung einer Kandidatenmenge und im Anschluß Ähnlichkeitsmaße zur Feinsuche und zum Ranking miteinander kombinieren kann, um ähnliche Wörter in großen Lexika zu suchen. Diskutiert werden damit die ersten zwei der vier Vorgehenweisen aus Abbildung 10.1. Darüberhinaus wird eine Spezialform der vierten Vorgehensweise betrachtet und mit den Alternativen verglichen. Ein wesentlicher Teil der Zielsetzung war, die Qualität und Natürlichkeit der erhaltenen Korrekturvorschläge und des Rankings zu beurteilen. Als Testdaten wurden drei Wörterbücher verwendet:

1. „DICT" ist das englische Wörterbuch, das mit dem *ispell* spell checker mitgeliefert wird. Es enthält 113.212 Wörter mit einer durchschnittlichen Länge von $9,1$ Buchstaben pro Wort. DICT ist im Internet offen verfügbar,

2. Das zweite Lexikon, das „NAMES" genannt wird, enthält 31.763 Wörter, mit einer Durchschnittslänge von $6,8$. Es wurde vor allem in die Untersuchung

mitaufgenommen, um die Wirksamkeit von phonetischen Filtern wie SOUNDEX (cf. Beispiel 9.1.3) zu überprüfen.

3. Das dritte Lexikon, ,,TREC“ genannt, hat seine Bezeichnung von der Text Retrieval Conference. Es enthält 1.073.727 Wörter aus einer 3 GB Sammlung von Texten der Konferenz. Die durchschnittliche Wortlänge hier beträgt $9,3$.

Um die Qualität und Effizienz der Verfahren zu bestimmen, wurden für die Wörterbücher jeweils feste Mengen von Abfragen verwendet und dann Beurteilungen der Relevanz der zurückerhaltenen Wörter und der anderen Lexikonwörter (matched, matched nicht) durch Testpersonen ausgewertet. Ein Anliegen beim Matchen von Eigennamen war insbesondere, daß auch Schreibvarianten (wie sie etwa beim Zitieren russischer Namen in der angelsächsischen Literatur häufig auftreten) gut miterfaßt werden. Auf dieser Grundlage wurden ,,Präzision“ und ,,Recall“ der Verfahren gemessen. Für ,,TREC“ war eine Bestimmung des Recalls aus offenkundigen Gründen nicht möglich. Als kombiniertes ,,Effektivitätsmaß“ wurde die durchschnittliche Präzision gemessen, die sich bei 11 vordefinierten Recallwerten (0%, 10%,. . ., 100%) ergab.

Effektivität der Einzelverfahren

Zunächst wurden folgende Einzelverfahren getestet.

1. Die Verwendung des Levenshtein-Abstands (,,edit”),
2. die Verwendung des modifizierten Levenshtein-Abstands (,,modified edit“), wo auch Transpositionen als primitive Operationen mit Kosten 1 zugelassen sind (vgl. Abschnitt 7.2),
3. die Verwendung des Trigramm-Abstands, wo man nur die Zahl der gemeinsamen Trigramme von Eingabewort und Lexikonwort als Ähnlichkeitsmaß nimmt (,,gram-count“),
4. Ukkonens Trigrammabstand (,,gram-dist“) aus Beispiel 8.1.4,
5. Die Verwendung von *agrep*, bei der ohne Ranking alle Lexikonwörter zurückgegeben werden, die vom Eingabewort einen maximalen Editierabstand K (vom Benutzer vorzugeben) haben,
6. das SOUNDEX System (cf. Beispiel 9.1.3)

sowie drei weitere Methoden, auf die wir nicht eingehen. Die Ergebnisse (Effektivität im oben definierten Sinn) sind in Tabelle 10.4 aufgeführt und zeigen, daß die alleinige Verwendung des SOUNDEX Ähnlichkeitsschlüssels zu sehr schlechten Ergebnissen auch bei dem Namenslexikon (wo Schreibvarianten mitzuerfassen sind) führt. Bei den Angaben, insbesondere in der letzten Spalte, ist allerdings die Art der Eingabewörter des Tests mitzuberücksichtigen. Bei DICT bestand die Eingabe aus 170 menschlich verursachten Schreibfehlern. Bei menschlich verursachten Schreibfehlern, anders als bei der OCR-Erkennung, spielen Transpositionsfehler eine deutliche Rolle. Dies schlägt sich in der Überlegenheit der ,,modified edit“ Methode gegenüber der ,,edit“ Methode nieder. Die nachfolgend beschriebenen Ergebnisse betreffen kombinierte Techniken. Zunächst werden Techniken zur Präselektion einer Kandidatenliste untersucht.

Matching-Technik	NAMES	DICT
edit	$63,7\%$	$39,8\%$
modified edit	$61,2\%$	$45,5\%$
gram-dist	$61,5\%$	$45,5\%$
gram-count	$55,9\%$	$21,7\%$
agrep	$32,8\%$	$21,5\%$
SOUNDEX	$27,4\%$	$4,5\%$

Abbildung 10.4: Durchschnittliche Präzision verschiedener Matching Techniken

n	Index NAMES	CPU (ms) NAMES	Index DICT	CPU (ms) DICT
2	$94,3\%$	$57,5$	$72,2\%$	$174,2$
3	$137,6\%$	$21,5$	$92,5\%$	$79,4$
4	$242,2\%$	$16,9$	$127,2\%$	$66,1$

Abbildung 10.5: Platzbedarf und Effizienz der n-Gramm Indexierung

n-Gramm Indexierung und Effizienz der Präselektion

Zobel und Dart geben in ihrer Arbeit eine Variante der n-Gramm Indexierung von Owolabi und McGregor an. Diese wird mit Hinblick auf Speicherbedarf und Effizienz bei der Verwendung zur Präselektion einer Kandidatenliste mit 50 Kandidaten getestet. Für n werden die Alternativen $1, 2, 3, 4$ untersucht. Wie bei Owolabi und McGregor ist jeder Lexikoneintrag mit einer Nummer versehen. Mit jedem n-Gramm, das in einem Lexikonwort vorkommt, wird eine „Positionsliste“ assoziiert, die die Nummer aller Lexikonwörter enthält, in denen das n-Gramm auftritt. Zobel und Dart schlagen vor, bei den Lexikonwörtern wie beim Eingabewort ein Dummy-Symbol am Start und am Ende hinzuzufügen, um die Präzision zu erhöhen. Für Werte $n = 2, 3, 4$ werden die n-Gramme in einer dynamischen Datenstruktur (Hash-Tabelle) verwaltet. Die Verwendung des n-Gramm Indexes ist nun ähnlich wie bei Owolabi und McGregor beziehungsweise bei Kim und Shawe-Taylor. Die Positionslisten, die mit den n-Grammen des Eingabewortes assoziiert sind, werden vereinigt, wobei die Zahl der Vorkommen jeder Wortnummer ermittelt wird. Es wird dann eine Heap-Struktur verwendet, um die 50 Wörter mit dem höchsten Count zu ermitteln. Die Ergebnisse in Tabelle 10.5 zeigen, wie sich die Verwendung eines höheren Werts für n in einem höheren Platzbedarf für den Index und in einer geringeren Rechenzeit niederschlägt. Die Indexgrößen nehmen die Größe des Wörterbuchs als Vergleich. Die CPU Zeiten sind Durchschnittswerte für eine Abfrage bei Rückgabe von 50 Kandidaten.

Permutierte Lexika zur Präselektion

Zobel und Dart diskutieren auch die Verwendung sogenannter „permutierter Lexika“ zur Vorauswahl einer Kandidatenliste. Hierbei wird jedes Lexikonwort mit einem Endmarker versehen, danach werden alle Rotationen des Eintrags mit in das Lexikon aufgenommen. Aus dem Eintrag „*wine*” wird also „*wine*|” mit den zusätzlichen Rotationen „*ine*|*w*”, „*ne*|*wi*”, „e|*win*” und „|*wine*”. Das erweiterte Lexikon besteht aus allen Rotationen, die alphabetisch sortiert sind. Zur Vorauswahl einer Kandidatenliste für das Eingabewort W werden auch für $W|$ alle Rotationen berechnet. Mittels einer binären Suche werden die Positionen der rotierten Formen von W im Lexikon der Rotationen berechnet. Eine solche Position kann zwischen zwei Lexikoneinträgen liegen, wenn W nicht im ursprünglichen Lexikon vorkommt. Die Umgebungen der so ermittelten Positionen der Rotationen werden in die Korrekturliste mitaufgenommen. Zu den unmittelbaren Rotationsnachbarn von *wine* gehören beispielsweise (Rotationen von) *waistline* und von *swine*.

Dank spezieller Techniken zur Repräsentation permutierter Lexika ergibt sich eine sehr schnelle Auswahl einer Kandidatenliste, allerdings ist die Qualität (gemittelte Präzision) der Kandidatenliste nicht sehr hoch.

Kombinierte Vorauswahl und Feinsuche

Zur Beurteilung von Methoden der Feinsuche wurde die Präselektion einer Kandidatenliste mit Hilfe der ,,gram-count" Methode verwendet, die zu Experimentierzwecken auch durch die zusätzliche Verwendung von SOUNDEX-artigen Ähnlichkeitsschlüsseln verfeinert wurden. Die Experimente zeigten, daß bei der Endauswahl der Korrekturvorschläge sowohl die ,,gram-dist"-Methode (Ukkonens Abstandsmaß) als auch die Verwendung des Editierabstands zu einer guten Gesamtkomplexität und Effektivität des kombinierten Verfahrens führt. Etwas überraschend war, daß sich bei der Verwendung von Ukkonens Abstandsmaß sogar (unwesentlich) bessere Ergebnisse (bei DICT sowohl in Bezug auf CPU-Zeiten als auch auf Effektivität) ergaben.

Durch die zusätzliche Verwendung von SOUNDEX-artigen Ähnlichkeitsschlüsseln bei der Präselektion ergab sich eher eine Verschlechterung der Ergebnisse bezüglich der Effektivität. Zusammenfassend wird daher in [ZD95] eine Kombination von n-Gramm-basierter Präselektion und Feinselektion mittels Ukkonens Abstandsmaß oder mittels des Editierabstands als geeignetstes Verfahren betrachtet. Es wird angedeutet, das sich bei Verwendung des größeren TREC-Lexikons dieselbe Situation ergibt.

Kapitel 11

Tokenisierung, Erstellen und Auswerten von Trainingskorpora

Kapitel 12

Schnelle approximative Suche im Wörterbuch

Der Inhalt dieses Kapitels ist die getrennt beigelegte Arbeit “Fast Approximate Search in Large Dictionaries”.

Kapitel 13

Das Web als dynamisches Wörterbuch

Der Inhalt dieses Kapitels ist die getrennt beigelegte Arbeit “Lexical Postcorrection of OCR-Results - The Web as a Dynamic Secondary Dictionary?”.

Kapitel 14

Das Nachkorrektursystem des CIS

Der Inhalt dieses Kapitels ist die getrennt beigelegte Arbeit "A Visual and Interactive Tool for Lexical Postcorrection of OCR-Results".

Kapitel 15

Korrektur im Satz- und Dokumentenkontext

15.1 Verwendung von HMM

Nach Hull [Hul92] und Keenan et. al. [KEW91].

15.2 Verwendung von Wort-Assozationswerten

Nach Hong und Hull [HHb].

Grundidee

Worterkennungsverfahren versuchen anders als übliche Verfahren der OCR, bei der Analyse eines Dokumentenbilds direkt ganz Worte aus einem Lexikon zu erkennen. Das Problem gilt als schwierig, besonders bei Schmutzeffekten. Das Paper stellt ein Nachkorrekturverfahren vor, das als Eingabe eine Liste von Alternativwörtern für jedes Wort des Eingabetexts hat. Die Grundidee ist, Informationen über die Assoziiertheit von Wörtern zu verwenden.

Beispiel 15.2.1 Der Eingabesatz sei „The river boat race was held yesterday“. Das Worterkennungsverfahren könnte als zweites Wort die Alternativen {rover 0.5, river 0.5} vorschlagen. Wenn eine Wissensquelle vorhanden ist, die sagt, daß „river“ und „boat“ sehr viel stärker assoziiert sind als „rover“ und „boat“, so kann dies dazu ausgenützt werden, die Alternative „river“ zu finden.

Es sind in der Computerlinguistik-Literatur einige Assoziiertheitsmaße für Wörter bekannt. Der Artikel verwendet die sogenannte *Mutual Information.* Die Assoziiertheit zweier Wörter x und y ist hier

$$MI(x, y) = log_2 \frac{P(x, y)}{P(x)P(y)}$$

wo $P(x)$ die Wahrscheinlichkeiten ist, das Wort x in einem Textkorpus (was heißt das???) zu beobachten und $P(x, y)$ die die Wahrscheinlichkeit, x und y zusammen in einem Fenster fester Größe (also unter einem vorgegebenen Maximalabstand) zu beobachten. [1]. Die Besonderheit des hier vorliegenden Ansatzes ist, daß zwar *direkt* nur die Assoziiertheit von je zwei aufeinanderfolgenden Wörtern im Text in das Verfahren eingeht. Durch die Verwendung eines sogenannten Relaxationsalgorithmus wird der Effekt jedoch weitergegeben und vermittelt so einen möglichst guten Kompromiß zwischen allen Wortwahlen innerhalb eines Satzes.

Algorithmus: Hintergrund und Beschreibung

Zu ergänzen.

Experiment und Analyse

Zu ergänzen.

Probleme des Ansatzes

Aufgrund der großen Zahl sehr seltener Wörter ist es ein großes Problem, aus einem Korpus zuverlässige Zahlen über die Assoziiertheit zweier beliebiger Wörter zu erhalten. Die meisten Statistiken liefern für seltene Wörter keine verwendbaren Werte.

Um die Zahl der zu betrachtenden Wörter nicht explodieren zu lassen, wird die Trainingsstatistik nicht für beliebige Paare von Wörtern aus dem Trainingskorpus erstellt, sondern nur für die Paare aufeinanderfolgender Worte. Hier gibt es 1,2 Millionen solcher Paare. Überlicherweise arbeitet man bei der Ermittlung der Assoziiertheit aber mit größeren Fenstern.

---

[1]Bem.: das hier verwendete Assoziationsmaß ist nicht optimal, es gibt präzisere Modelle, die log-likelihood Methode

Kapitel 16

Ergänzende Themen

Zu Kombination(?)

-
- Zu Voting als Kurzzusammenfassung die Stringvergleich-Methode aus Rice et al. [RKN94].
- Verwendung visueller Zwischen-Wort Constraints nach Hong und Hull [HHa].

9. Lexikalische Korrektur I: Korrektur mit binären positionellen Trigrammen nach Hanson et. al. [HRF75].

12. Dokumentenkontext, Datenbankabgleich und unscharfe Suche
    - Systembeschreibung: Datenbankabgleich bei Steuerformularen nach [SSRL]
    - Approximative Stringsuche
    - Approximative Suche in Datenbanken
    - Ermittlung der logischen Struktur von wissenschaftlichen Bibliographischen Referenzen nach Parmentier und Belaid [PB]

13. Dokumentenanalyse I
    - Virtuelles Partitionieren von Wörterbüchern nach ([HK96])
    - Papier-SGML/XML

14. Dokumentenanalyse II:
    - Papier-SGML/XML
    - Briefanalysetool Daniel

Kapitel 17

Anhang: Informationsquellen zur OCR-Forschung

Die wichtigsten Zeitschriften

International Journal of Document Analysis and Recognition (neu)
Pattern Recognition
Image and Vision Computing
Machine Vision and Image Analysis
Pattern Recognition Letters
Int. Journal of Pattern Recognition and Artificial Intelligence (PRAI)
Machine Vision and Applications (Spec. Issue 92, No 5)
IEEE Trans. on Computers (C-20, No. 12, 1971)
IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI (No. 15, 1993)

Die wichtigsten Konferenzen

Proc. International Conference on Document Analysis and Recognition (ICDAR)
Proc. International Conference on Pattern Recognition
IEEE Proceedings (Special issue OCR, July 92)
Proc. ACM Conf. Document Processing Systems (1988)

Einige wichtige Web-Adressen

Links zu den nachfolgenden Forschungsgruppen finden sich unter http://hera.itc.it:3003/ messelod/OCR/RelatedSites.html

- CEDAR Home-page. Document Recognition Group at Buffalo University. http://www.cedar.buffalo.edu
- CENPARMI: Centre for Pattern Recognition and Machine Intelligence at Concordia University, Montreal, Kanada http://www.cenparmi.concordia.ca/

- DANTE: Document ANalysis and TEchnologies research group. University of Firenze and University of Siena (Italy). http://www-dii.ing.unisi.it/research/CS/docproc/home.html
- Document Analysis and Office Automation at DFKI. http://www.dfki.uni-kl.de/da/da.html
- Document Analysis and Understanding at LACAM, Uni Bari - Italy. http://lacam.di.uniba.it:8000/
- Document Image Analysis at BILD at the Norwegian Computing Center. http://www.nr.no/bild/DocumentAnalysis.html
- EECS, Electrical Engineering and Computer Science. http://www.cs.tufts.edu/ - The DART project (understanding of printed media)
- ECSE Department of Electrical, Computer, and System Engineering, Rensselaer Polytechnic Institute, Troy NY. http://www.ecse.rpi.edu/
- EPCC, Edinburgh Parallel Computing Centre
- Elsag Bailey, Genova, Italy. Visit also Elsag Bailey.
- ERIM The Environmental Research Institute of Michigan, Ann Arbor, Michigan.
- Handwriting Recognition Home Page, Essex University. - OSCAR an Off-line Script and Character Recognition Toolset.
- Handwritten Character Recognition TCTS Lab Faculté Polytechnique de Mons (Belgium)
- IDIAP Institut Dalle Molle dÍntelligence Artificelle Perceptive, in Switzerland.
- Imaging Systems Lab (ISL) at Carnegie Mellon University.
- ISRI Information Science Research Institute at University of Nevada, Las Vegas.
- ITC-irst Istituto per la Ricerca Scientifica e Tecnologica at ITC, Trento (Italy).
- Language And Media Processing (LAMP) Lab at the University of Maryland at College Park.
- NICI Handwriting Group at Nijmegen University, The Netherlands.
- NIST Visual Image Processing Group
- Praktische Informatik IV, University of Mannheim. MoCA Text Recognition.
- Pattern Recognition Team, Los Alamos National Laboratory.
- PRIP Pattern Recognition and Image Processing Lab, Michigan State University.
- Research Group ZAIK, at Cologne University,
- The Script and Pattern Recognition Group at The Nottingham Trent University,

- Software & Document Engineering Group at the Institute of Informatics, University of Fribourg, Switzerland:
- SRI International's Advanced Automation Technology Center. IDU and OCR
- U Bern Research Group on Computer Vision and AI
- University of Oulu (Finland).
- University of Washington Intelligent Systems lab.

Literaturverzeichnis

[AB95] J.C. Anigbogu and A. Belaid. Hidden markov models in text recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(6):925–958, 1995.

[AFW83] R.C. Angell, G.E. Freund, and P. Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing and Management*, 19:255–261, 1983.

[ANT79] T. Agui, M. Nakajima, and E.T. Takahashi. A method of recognition and representation of korean characters by tree grammars. *IEEE Trans. Patt. Anal. Machine Intell. PAMI*, 1(3):245–251, 1979.

[AP77] F. Ali and T. Pavlidis. Syntactic recognition of handwritten numerals. *IEEE Trans. on Systems, Man, and Cybernetics*, 7(7):537–541, 1977.

[Bal70] G.J. Balm. An introduction to optical character reader considerations. *Pattern Recognition*, 2:151–166, 1970.

[BB59] W.W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proc. Eastern Joint Computer Conference* **16**, pages 225–232, 1959.

[Bla58] D. Black. *The Theory of Committees and Elections.* Cambridge University Press, 1958.

[BT80] D.C. Bouchard and G.T. Toussaint. Heuristic search methods for efficient use of dictionary information in text recognition. Technical Report Technical Report SOCS 80.5, School of Computer Science, McGill University, 1980.

[Bur] Christopher J.C. Burges. A tutotial on support vector machines for pattern recognition. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[dBdBT95] Francois de Bertrand de Beuvron and Philippe Trigano. Hierarchically coded lexicon with variants. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(1):145–165, 1995.

[DH68] R.O. Duda and P.E. Hart. Experiments in the recognition of hand-printed text: part II - context analysis. In *Proc. AFIPS Fall Joint Comput. Conf. bf 33*, pages 1139–1149, 1968.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Addision-Wesley, New York, 1973.

[DHH+97] A. Dengel, R. Hoch, F. Hönes, T. Jäger, M. Malburg, and A. Weigel. Techniques for improving OCR results. In H. Bunke and P.S.P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[DHTW80] B. Duerr, W. Haettich, H. Tropf, and G. Winkler. A combination of statistical and syntactical pattern recognition applied to classification of unconstrained handwritten numerals. *Pattern Recognition*, 12(3):189–199, 1980.

[ea93] L. Breiman et. al. *Classification and Regression Trees*. Chapman and Hall, New York, 1993.

[Eik93] Line Eikvil. OCR Optical Character Recognition, 1993. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[Fu74] K. Fu. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.

[Fu82] K.S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Engelwood Cliffs, New York, 1982.

[Gor] R.E. Gorin. SPELL: Spelling check and correction program. Documentation Online in File DOC: SPELL DOC at CMU-CS-A. ARPA; 1971.

[GS90] V.K. Govindan and A.P. Shivaprased. Character recognition – a review. *Pattern Recognition*, 23:671–683, 1990.

[HB97] T.M. Ha and H. Bunke. Image processing methods for document image analysis. In H. Bunke and P.S.P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 49–78. World Scientific, 1997.

[HCW] S.M. Harding, W.B. Croft, and C. Weir. Probabilistic retrieval of ocr degraded text using n-grams. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[HD80] P.A.V. Hall and G.R. Dowling. Approximate string matching. *Computing Surveys*, 12(4):381–402, 1980.

[HHa] Tao Hong and Jonathan J. Hull. Algorithms for postprocessing ocr-results with visual inter-word constraints. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[HHb] Tao Hong and Jonathan J. Hull. A relaxation algorithm that uses word collocation to improve text recognition performance. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[HHS92] T.K. Ho, J.J. Hull, and S.N. Srihari. A word shape analysis approach to lexicon based word recognition. *Pattern Recognition Letters*, 1992.

[HHS94] T.K. Ho, J.J. Hull, and S.N. Srihari. On multiple classifier systems for pattern recognition. *IEEE Trans. Patt. Anal. Machine Intell. PAMI*, 16(1):66–75, 1994.

[HK96] Rainer Hoch and Thomas Kieninger. On virtual partitioning of large dictionaries for contextual post-processing to improve character recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(4):273–289, 1996.

[HLS95] Y.S. Huang, K. Liu, and C.Y. Suen. The combination of multiple classifiers by a neural network approach. *Int. J. of Pattern Recognition and Artificial Intelligence*, 9:579–597, 1995.

[Ho92] T.K. Ho. A theory of multiple classifier systems and its application to visual word recognition. Technical Report Technical Report 92-12, State University of New York at Buffalo, 1992.

[HRF75] A.R. Hanson, E.M. Riseman, and E. Fisher. Context in word recognition. *Pattern Recognition*, 1975.

[HS82] J.J. Hull and S.N. Srihari. Experiments in text recognition with binary n-gram and Viterbi algorithms. *IEEE Trans. Patt. Anal. Machine Intell. PAMI-4*, 5:520–530, 1982.

[HS93] Y.S. Huang and C.Y. Suen. Combination of multiple classifiers with measurement values. In *Proc. of the Second International Conference on Document Analysis and Recognition (ICDAR 93)*, pages 598–601, 1993.

[Hul92] J.J. Hull. A hidden markov model for language syntax in text recognition. In *Proc. of the 11th IAPR Int. Conf. on Pattern Recognition*, pages 416–423, The Hague, The Netherlands, 1992. IEEE Computer Society Press.

[KEW91] F.G. Keenan, L.J. Evett, and R.J. Withrow. A large vocabulary stochastic analyser for handwriting recognition. In *Proc. of the First International Conference on Document Analysis and Recognition (ICDAR 91)*, pages 794–802, 1991.

[KF67] H. Kucera and W.N. Francis, editors. *Computational Aspects of Present-Day American English*. Brown University Press, 1967.

[KS97] Ulrich Kressel and Jürgen Schürmann. Pattern classification techniques based on function approximation. In H. Bunke and P.S.P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 49–78. World Scientific, 1997.

[KST92] J.Y. Kim and J. Shawe-Taylor. An approximate string-matching algorithm. *Theoretical Computer Science*, 92:107–117, 1992.

[Kuk92] Karen Kukich. Techniques for automatically correcting words in texts. *ACM Computing Surveys*, pages 377–439, 1992.

[Kur83] M.W. Kurzynski. The optimal strategy of a tree classifier. *Pattern Recognition*, 22(1):81–87, 1983.

[Lev66] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 1966.

[LHS97] L. Lam, Y.-S. Huang, and C.Y. Suen. Combination of multiple classifier decisions for optical character recognition. In H. Bunke and P.S.P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[LLNT96] Yanhong Li, Daniel P. Lopresti, George Nagy, and Andrew Tomkins. Validation of image defect models for optical character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18 (2):99–108, 1996.

[LS95] D. Lee and S.N. Srihari. A theory of classifier combination: The neural network approach. In *Proc. of the Third International Conference on Document Analysis and Recognition (ICDAR 95)*, pages 42–45, 1995.

[LW75] R. Lowrance and R.A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 23(2):177–183, 1975.

[Man86] J. Mantas. An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430, 1986.

[MG] Andreas Myka and Ulrich Güntzer. Fuzzy full-text searches in ocr databases. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[Mor82] B.M.E. Moret. Decision trees and diagrams. *ACM Computing Surveys*, pages 593–623, 1982.

[MP80] W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *J. Computer Syst. Sci.*, 1980.

[MS88] E. Mandler and J. Schürmann. *Combining the Classification Results of Independent Classifiers Based on the Dempster/Shafer Theory of Evidence*, pages 381–393. Elsevier Science Publishers B.V., North Holland, 1988.

[Nag82] G. Nagy. Optical character recognition, theory and practice. In P.R. Krishnaiah and L.N. Kanal, editors, *Handbook of Statistics*, pages 621–649. North Holland, 1982.

[Neu75] D.L. Neuhoff. The viterbi-algorithm as an aid in text recognition. *IEEE Trans. Inform. Theory*, 11:222–226, 1975.

[OL97] B.J. Oommen and R.K.S. Loke. Pattern recognition of strings with substitutions, insertions, deletions, and generalized transpositions. *Pattern Recognition*, 30(5):789–800, 1997.

[OM88] O. Owolabi and D.R. McGregor. Fast approximate string matching. *Software - Practice and Experience*, 18(4):387–393, 1988.

[OTK76] T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the levenshtein metric. *IEEE Transactions on Computers*, C-25(2):172–178, February 1976.

[PA75] T. Pavlidis and F. Ali. Computer recognition of handwritten numerals. *IEEE Trans. on Systems, Man, and Cybernetics*, 5(6):610–614, 1975.

[PA79] T. Pavlidis and F. Ali. A hierarchical syntactic shape analyzer. *IEEE Trans. Patt. Anal. Machine Intell. PAMI*, 1(1):2–9, 1979.

[PB] F. Parmentier and A. Belaid. Logical structure recognition of scientific bibliographic references. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77, 1989.

[Rav67] J. Raviv. Decision making in markov chains applied to the problem of pattern recognition. *IEEE Trans. Inform. Theory*, IT-3:536–551, 1967.

[RE71] E.M. Riseman and R.W. Ehrich. Contextual word recognition using binary digrams. *IEEE Transactions on Computers*, C-20, 1971.

[RH74] E.M. Riseman and A.R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Transactions on Computers*, C-23, 1974.

[RKN94] Stephen V. Rice, Junichi Kanai, and Thomas A. Narkter. An algorithm for matching ocr-generated text strings. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(5):1259–1268, 1994.

[RY97] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. In *Proc. 14th International Conference on Machine Learning*, pages 287–295. Morgan Kaufmann, 1997.

[SD84] J. Schuermann and W. Doster. A decision theoretic approach to hierarchical classifier design. *Pattern Recognition*, 17(3):359–369, 1984.

[SH74] A.B. Shahidul-Hussain. Compound sequential probability ratio test for the classification of statistical dependent characters. *IEEE Trans. Comput.*, C-23:398–410, 1974.

[SHC83] Sargur N. Srihari, Jonathan J. Hull, and Ramesh Choudhari. Integrating diverse knowledge sources in text recognition. *ACM Transactions on Office Information Systems*, 1983.

[Sin90] R.M.K. Sinha. On partitioning a dictionary for visual text recognition. *Pattern Recognition*, 23(5):497–500, 1990.

[SKS96] Giovanni Seni, V. Kripasundar, and Rohini K. Srihari. Generalizing edit distance to incorporate domain information: Handwritten text recognition as a case study. *Pattern Recognition*, 29(3), 1996.

[SP] Hongwei Shi and Theo Pavlidis. Font recognition and contextual processing for more accurate text recognition. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[Sri84] S.N. Srihari. *Computer Text Recognition and Error Correction*. IEEE Computer Society Press, 1984.

[SS82] I.K. Sethi and G.P.R. Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Trans. Patt. Anal. Machine Intell. PAMI*, 4(4):441–445, 1982.

[SSRL] Sargur N. Srihari, Yong-Chul Shin, Vemulapati Ramanaprasad, and Dar-Shyang Lee. A system to read names and adresses on tax forms. Aus http://citeseer.nj.nec.com/cs ResearchIndex.

[ST79] R. Shinghal and G.T. Toussaint. A bottom-up and top-down approach to uding context in text recognition. *Int. J. Man-Mach. Stud.*, 11:201–212, 1979.

[Str78] L. Stringa. LCD: a formal language for constraint-free hand-printed character recognition. In *Proc. of the Fourth International Conference on Pattern Recognition*, pages 354–358, 1978.

[TIAY90] H. Takahashi, N. Itoh, T. Amano, and A. Yamashita. A spelling correction method and its application to an OCR system. *Pattern Recognition*, 23(3/4):363–377, 1990.

[TJT96] O. Due Trier, A.K. Jain, and T. Taxt. Feature extraction methods for character recognition — a survey. *Pattern Recognition*, 29:641–662, 1996.

[Tou78] G.T. Toussaint. The use of context in pattern recognition. *Pattern Recognition*, 1978.

[Ukk85] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[Ukk92] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92:191–211, 1992.

[WBR95] F. Weigel, S. Baumann, and J. Rohrschneider. Lexical postprocessing by heuristic search and automatic determination of the edit costs. In *Proc. of the Third International Conference on Document Analysis and Recognition (ICDAR 95)*, pages 857–860, 1995.

[WF74] R.A. Wagner and M. Fisher. The string-to-string correction problem. *Journal of the ACM*, 1974.

[XKS92] L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their application to handwritten numeral recognition. *IEEE Trans. on Systems, Man, and Cybernetics*, 22:418–435, 1992.

[ZD95] Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Software–Practice and Experience*, 25(3):331–345, 1995.