Übung Automatische Spracherkennung

Kapitel 1

Einführung – Hidden Markov Tool Kit (HTK)

Dieser Teil des Kurses gibt einen allgemeinen Überblick über HTK und Literaturhinweise.

1.1 Spracherkennung und HTK

Bei der automatischen Spracherkennung (ASR) geht es um die Umsetzung des gemessenen akustischen Sprachsignals in eine symbolische Form, die mehr oder weniger den übermittelten Informationsgehalt des Sprachsignals repräsentiert. Im einfachsten Falle ist dies die Kette der tatsächlich gesprochenen Wörter.

Zur wissenschaftlichen Beschreibung der üblichsten Techniken sei hier auf die Veranstaltung 'P6.1 Einführung in die automatische Spracherkennung' und die Literaturhinweise am Ende dieses Abschnitts verwiesen.

Spracherkennung als Eingabe-Technik von Mensch zu Maschine ist noch nicht soweit etabliert, dass man von einer Standard-Technik (wie beispielsweise der Maus) sprechen kann. Mehrere wissenschaftliche Ansätze werden parallel von verschiedenen Forschergruppen auf der ganzen Welt und in einer Vielzahl von Sprachen verfolgt. Dennoch hat sich innerhalb der letzten 15 Jahre herauskristallisiert, daß sich zumindest deutlich artikulierte und wenig störgräuschbehaftete Sprache am besten mit sogenannten statistischen Verfahren erkennen lässt. Die bekannteste statistische Methode zur Erkennung von Sprache sind die sog. 'Hidden Markov Modelle', bei denen der Sprach-Prozess als Quelle mit bestimmten statistisch modellierbaren Eigenschaften dargestellt wird.

An der Universität Cambridge, UK, wurde schon seit längeren zu dieser Technik ein sogenannter 'tool kit' entwickelt und kostenlos über das Internet verbreitet. Mit Hilfe dieses 'tool kits' war es möglich, mit relative geringem Programmieraufwand einen einfachen Spracherkenner zu betreiben, der mit den damals etablierten Standard-Techniken ausgerüstet war. Eine Zeit lang hat die amerikanische Firma 'Entropic' die Rechte an diesem 'tool kit' erworben und vertrieben, bis sie 1999 von Microsoft aufgekauft wurde und den Vertrieb des 'Hidden Markov Tool Kits' (HTK) einstellen mußte. Daraufhin hat die Universität Cambridge (UK) die Pflege, Weiterentwicklung und Bereitstellung von HTK wieder aufgenommen.

HTK hat sich mittlererweile innerhalb der Wissenschaftsgemeinde als Standard durchgesetzt. Zahlreiche Institute und auch Firmen verwenden HTK als Referenz-System. In wissenschaftlichen Veröffentlichungen ist es mittlererweile möglich, einfach auf HTK zu verweisen, um sich die

umständliche Beschreibung des in einem Experiment verwendeten Spracherkenners zu ersparen. Zunehmend kommt HTK auch in phonetischen und linguistischen Arbeiten zum Einsatz.

Da HTK jetzt wieder als public domain Software von der University of Cambridge angeboten wird, kann jeder das System kostenfrei benutzen. (http://htk.eng.cam.ac.uk/)

HTK ist kein fertiger Erkennungsalgorithmus, etwa vergleichbar den derzeit auf dem Markt erhältlichen Diktiersystemen. Es ist vielmehr eine Sammlung von Programmen ('tools'), die alle der Modellierung und Erkennung von Sprache dienen, und letztendlich den Aufbau einer quasi unendlichen Vielfalt von verschiedenen Spracherkennungsarchitekturen ermöglicht. Der intendierte Benutzer ist nicht der Endbenutzer, der sich einen Diktier-Apparat oder eine sprachgesteuerte Oberfläche bauen will, sondern der Wissenschaftler bzw. System-Ingenieur, der eigene Methoden mit Hilfe von HTK exemplifizieren möchte. Dementsprechend ist eine gewisse Einarbeitung in die HTK-Technik erforderlich, welche aber andererseits eine ausgezeichnete didaktische Möglichkeit bietet, die Spracherkennungstechnik mit Hilfe von HMM von Grund auf zu verstehen und zu beherrschen.

1.2 Der Tool Kit

Der 'tool kit' besteht wie bereits gesagt aus einzelnen Programmen, welche jeweils verschiedene Aufgaben im Zusammenhang mit dem Aufbau eines Spracherkenners erledigen. HTK Programme sind UNIX-orientiert und werden deshalb von der Kommandozeile aus aufgerufen:

```
% Programm <Optionen> <Argumente>
```

z.B.

```
% HCopy -C preprocess.conf
```

Genau wie andere UNIX-Kommandos besteht ein Kommandozeilen-Aufruf aus dem Programmname, Optionen (normalerweise mit '-' eingeleitet) und Argumenten, die verarbeitet werden sollen (meistens Dateien). Sämtliche HTK-Programme beginnen mit einem 'H' und sind in einem sog. Referenz-Manual ([2]) detailiert beschrieben. Ausserdem befindet sich dort in einem vorangestellten Kapitel eine sehr ausführliche Beschreibung der Implementierung, so dass der Anwender von HTK genau erfährt, wie das Programm mit den jeweiligen Einstellungen verfährt. In diesem Kurs wird die Verwendung dieses Manuals mit geübt; daher sollte jede Gruppe mindestens über eine Kopie des Manuals verfügen . Im Link `HTK_Book_3.2` ist eine PDF-Version des Referenz-Manuals; bitte diese NICHT ausdrucken! Kopien können Sie im Sekretariat Phonetik oder beim Dozenten erwerben.

HTK hat einen großen Vorteil: Fast sämtliche Daten, die von HTK-Programmen verarbeitet werden, sind als Text lesbar. Dadurch ist es für den wissenschaftlichen Benutzer sehr einfach, seine Arbeitsweise zu überprüfen. Der zweite grosse Vorteil liegt darin, daß HTK analog zur UNIX-Philosophie ein sehr komplexes Problem - wie es die Spracherkennung darstellt - in viele kleine besser überschaubare Teilprobleme zerlegt. Dadurch ist es möglich, die notwendigen Einzelteile für einen Spracherkenner schrittweise zu erarbeiten.

1.3 Kommandozeilenbefehle mit UNIX

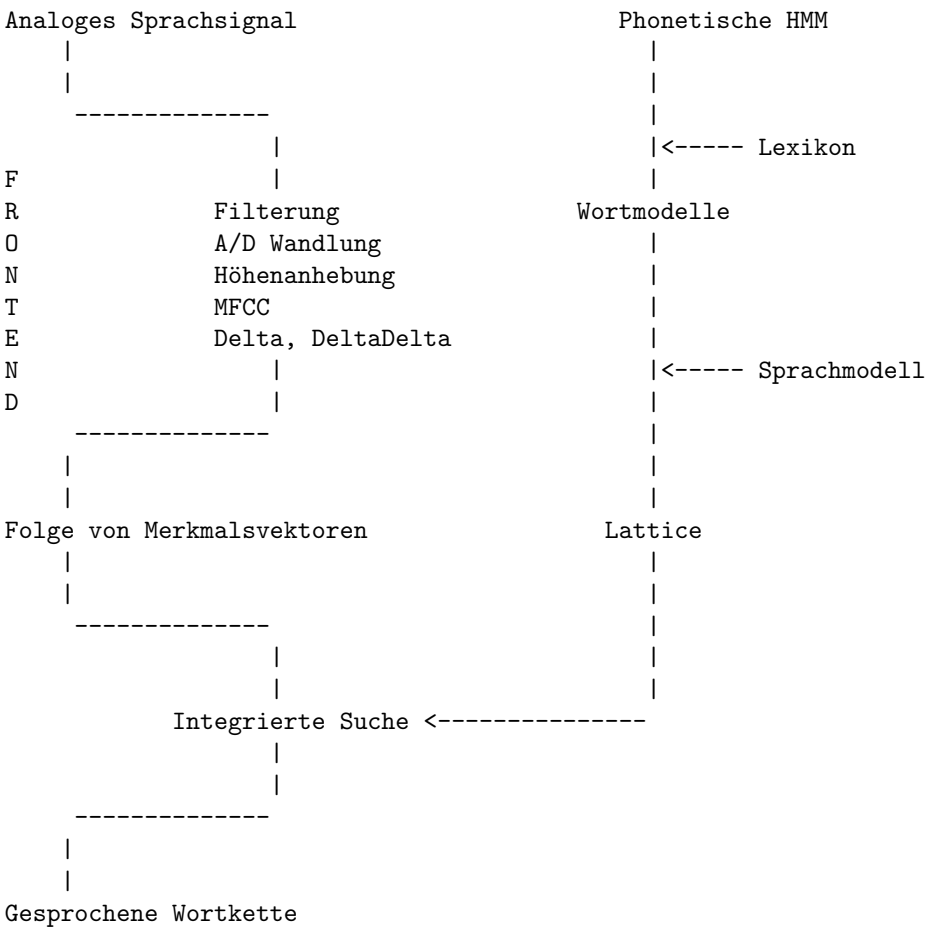
Da bei Teilnehmern, die nicht dem Bachelor-Programm angehören, oft keine UNIX-Kenntnisse vorhanden sind, werden wir zunächst die wichtigsten UNIX-Kommandos anhand des Miniskripts

`UNIX Kurzanleitung` besprechen und üben. Interessierte können sich anhand des längeren Skripts `UNIX Skript` weiter einarbeiten.

Eine didaktisch ausgewogene Einführung in UNIX ist im Kurs P4.2 'Werkzeuge der Sprachverarbeitung' enthalten.

1.4 Automatische Spracherkennung mit HMMs

Ein System zur Erkennung von Sprache läßt sich grob als stufenförmiges, signalverarbeitendes System betrachten, das die Information von Sprachsignal durch verschiedene Repräsentationen und Verarbeitungsstufen bis hin zum erkannten Satz verarbeitet (linker Zweig im nachstehenden Schema):



Das akustische Signal wird von einem geeigneten Sensor (Mikrophon) analog aufgezeichnet, gefiltert (Shannon-Theorem!), in ein digitales Signal gewandelt, ein weiteres Mal digital gefiltert (Höhenanhebung) und in eine Folge von Merkmalsvektoren umgewandelt. Diesen Schritt nennt man 'Frontend'. Die Merkmalsvektorfolge wird einem Suchalgorithmus übergeben, der die beste Wortkette nach dem Prinzip der Maximierung der Maximum-Likelihood berechnet, und zwar unter folgenden Einschränkungen ('integrierte Suche'):

1. Die möglichen Wörter bestehen aus statistischen Subwort-Modellen (i.a. Phonemen), die durch ein festes Set von HMM modelliert werden. 2. Ein Aussprache-Modell (Lexikon) regelt die erlaubten Phonemfolgen für jedes Wort. 3. Ein Sprachmodell gewichtet statistisch die erlaubten Wortkombinationen.

Daher verläuft parallel ein weiterer Strang von Verarbeitungen, in denen die erforderliche Wissensquelle für die Erkennung, nämlich die sogenannte 'lattice' ('Gitter') aufbereitet wird (rechter Zweig). Teilweise können diese Schritte - wie auch in unserem Praktikum - off-line, d.h. vor der eigentlichen Erkennung durchgeführt werden (das ist aber nicht immer der Fall!).

1.5 Das Grundprinzip von HTK

Kern eines jeden HTK-Spracherkenners ist die integrierte Suche. Um zu verstehen, was prinzipiell (und ohne den mathematischen Hintergrund) bei der integrierten Suche passiert, wollen wir in einem Gedankenexperiment den Erkenner Stück für Stück zusammenbauen.

In einer ersten Stufe habe der Erkenner überhaupt keine Wissensquelle. Der Input - also das vorverarbeitete Sprachsignal - stellt in diesem Falle nichts als eine Folge von sinnlosen Vektoren dar, die sich in mehr oder weniger ausgeprägten Clustern im Vektorraum herumtreiben.

(Skizze 2-dimensionaler Vektorraum mit Clustern)

Nun 'lernt' unser Phantasie-Erkenner die phonetischen HMM kennen. Damit ist er schon - zumindest theoretisch - in der Lage, aus dem sinnlosen Strom von Vektoren phonetische Segmente auszumachen. Z.B. wird er nun 'wissen', dass Vektoren, die in den Kern eines dicken Clusters mit dem Label /a:/ fallen, mit hoher Wahrscheinlichkeit dem phonetischen /a:/-Laut zuzuordnen sind.

(Skizze Clusterbildung und HMM)

Im Prinzip haben wir jetzt eine universelle Transkriptionsmaschine für deutsche Sprachlaute - der Wunschtraum eines jeden Phonetikers. Jedes Segment aus unserer Liste der erlaubten HMMs kann auf jedes andere folgen. Ein möglicher Output könnte z.B. so aussehen:

```
/QaIt@i:SOnEsfRi:lINv@t6/
```

Was passiert jetzt in unserem Erkenner auf den 'rechten Zweig'? HTK liest die Liste der erlaubten Sprachlaute (der phonetischen HMM) und bildet eine primitive 'loop lattice', eine vielfache Schleife, die alle Phonem-Label parallel leitet.

(Skizze Loop Lattice mit Labeln)

Dann liest HTK die HMM-Definitions-Dateien für jeden Sprachlaut und kopiert die jeweilige HMM-Struktur an die Stelle der Labels.

(Skizze Loop Lattice mit HMM)

Die 'integrierte Suche' macht nun folgendes: Noch bevor der erste Merkmalsvektor des Satzes gelesen wird, erzeugt sie ein sogenanntes 'token', das in die Lattice eingespeist wird. Es kommt zunächst in einen sogenannten 'Nullknoten', der keinen Vektor 'konsumieren' darf. Da sich die Lattice von diesem Nullknoten aus in die verschiedenen Phoneme verzweigt, wird das Token entsprechend oft ge-clont und weitergeschickt. Betrachten wir nur das Token, das nach dem /a:/-Laut geschickt wurde, dann landet dieses nun im ersten wirklichen Knoten oder 'state' des /a:/-HMM. Der erste Merkmalsvektor wird mit dem hier gespeicherten Wahrscheinlichkeitsmodell verglichen und eine Wahrscheinlichkeit berechnet, dass dieser Vektor von diesem State 'stammt'. Die Wahrscheinlichkeit wird im Token gespeichert und das Token an die nächsten Knoten weitergereicht. Da auch hier wieder eine Verzweigung stattfindet, wird das Token wieder ge-clont. Im nächsten State wird wieder eine Wahrscheinlichkeit berechnet und mit der bereits gespeicherten multipliziert, und so

weiter und so fort.

(Skizze Token-Verzweigung im /a:/-Modell)

Es entstehen explosionsartig Millionen von Tokens die durch die Lattice gereicht werden. Mit jeden neuen Merkmalsvektor, der vom Frontend gelesen wird, werden alle Tokens ge-clont und weitergereicht. Alle akkumulieren fleissig ihre Wahrscheinlichkeiten. Irgendwann ist der Satz zu Ende und der letzte Merkmalsvektor wird konsumiert. Von den vielen Tokens sind ein kleiner Bruchteil zufällig gerade in der Lage, auf den Nullknoten überzugehen, der den Ausgang aus der Lattice markiert. Diese werden bewertet und das 'beste', d.h. das Token mit der höchsten aufakkummulierten Wahrscheinlichkeit wird zum Sieger erklärt. Nun 'merken' sich die Tokens schlauerweise genau den Weg, den sie durch die Lattice durchlaufen haben, um zum Ziel zu kommen. Eine Auswertung dieses Pfades ergibt die Kette der erkannten Phoneme.

In der Praxis funktioniert diese Technik nur durch Ausnutzung des Viterbi-Prinzips, dass nämlich von allen im einem Knoten parallel ankommenden Token nur das mit dem bisher besten Pfad weiterverfolgt werden muss (Einzelheiten zu Viterbi und DP siehe [1]) und durch massives Ausdünnen ('pruning') unwahrscheinlicher Token-Kandidaten.

Die Sache hat nur einen Haken: wie man oben an unserem Beispiel sieht, funktioniert sie leider nicht sehr gut. Der Grund hierfür ist die starke Variabilität der Sprachlaute, wenn sie von verschiedenen Sprechern, in verschiedenen Situationen und in verschiedenem Kontext artikuliert werden. Außerdem möchte man normalerweise kein phonetisches Transkript sondern lieber die gesprochenen Wörter der gesprochenen Äußerung.

Als nächstes 'lernt' unser Phantasie-Erkenner daher etwas über deutsche Phonotaktik. Und damit es einfacher wird, überspringen wir die Phonotaktik-Stufe, definieren gleich ein endliches Set von Wörtern, welches der Erkenner verstehen soll, und nennen dieses Set 'Lexikon'. Im Lexikon wird für jedes Wort festgelegt, wie seine Orthographie aussieht und wie es ausgesprochen wird. Natürlich müssen wir darauf achten, dass wir nur phonetische Symbole im Lexikon verwenden, die auch in der Liste der phonetischen HMM vorkommen. Die Lattice verändert sich dadurch erheblich: wo

vorher Phoneme parallel geführt wurden, stehen nun alle Wörter des Lexikons.

(Skizze loop lattice mit Wort-Labeln)

Jedes Wort-Label wird ersetzt durch die Kette der Phonem-Label, wie sie im Lexikon definiert ist.

(Ergänzung der Skizze loop lattice mit Phonem-Labeln)

Und wie vorher werden die Phonem-Label ersetzt durch die echten HMM-Strukturen und der Token-Prozess startet.

(Ergänzug der Skizze Loop Lattice mit HMM)

Dieser Erkenner ist in der Lage, alle Wörter des Lexikons in beliebiger ('loop') Reihenfolge zu erkennen. Ein Output könnte z.B. so aussehen:

```
'Euter nicht schönes üppig Vetter'
```

Was unserem Erkenner noch fehlt, ist Wissen über die deutsche Syntax, semantische Beziehungen zwischen Wörtern und Wissen über den Kontext des Gesprächs ('Pragmatik'). Wir gehen in diesem Gedankenexperiment nur noch einen kleinen Schritt in dieses weite Feld der Linguistik, indem wir unserem Phantasie-Erkenner jetzt noch Wissen über Wort-Paare vermitteln. Dazu dient das Sprachmodell ('language model'), welches nichts anderes ist als eine Statistik darüber, mit welcher Wahrscheinlichkeit welches Wort auf ein anderes folgt (sog. Bigram-Statistik). (Zu Details über Sprachmodelle und deren mathematischen Zusammenhang mit dem Problem der statistischen Spracherkennung siehe in [1].)

Unsere einfache Loop-Lattice wird nun in eine sehr vernetzte Struktur aufgespalten, in der jedes Wort mit jedem verbunden wird. Auf den Verbindungen (den Uebergängen) werden die aus

dem Sprachmodell gelesenen Bigramm-Wahrscheinlichkeiten eingetragen und bei jedem Token-Uebergang aufmultipliziert.

(Skizze allgemeine Lattice)

Damit ist unser Phantasie-Erkenner fertig und das Gedankenexperiment beendet. Der Output sollte jetzt lauten:

```
'heute ist schönes Frühlingswetter'
```

1.6 Was ist zu tun?

Die vom Erkenner verwendeten Wissensquellen, also die phonetischen HMM, das Lexikon und das Sprachmodell sind nicht von vorneherein gegeben. Vielmehr müssen diese Daten für jede Spracherkennungsaufgabe erneut produziert werden. Dazu benötigt man unfangreiche (Beipiel-)Daten aus der Domäne des Spracherkenners, das sog. Trainingsmaterial. Ein erheblicher Anteil am Entwurf und Bau eines Spracherkenners liegt in der richtigen Aufbereitung und Verwendung dieser Trainings-Daten.

Des weiteren müssen wir in der Lage sein, den gebauten Erkenner in seiner Leistung zu beurteilen (evaluieren). Dazu dient ein definiertes Test-Set oder Test-Korpus von Daten, die der Erkenner nicht zum Training verwendet hat. Auch dieses Set muss zuerst definiert und erzeugt werden.

Als drittes muss der fertige Algorithmus noch optimiert werden ('tuning'). Zur Optimierung dient eine Vielzahl von Parametern, die den Erkennungsprozess direkt beeinflussen. Ein gutes Beispiel ist die Gewichtung von alternativen Aussprache im Lexikon: man kann solchen Alternativen nur wenig Gewicht geben; dann wird der Erkenner meistens versuchen, die kanonische Aussprache zu suchen. Man kann diesen Alternativen mehr Gewicht geben; dann kann es passieren, dass der Erkenner in Entscheidungskonflikte zwischen sehr ähnlich lautenden Wörtern gerät. Um die Wirkung der verschiedenen Parameter beurteilen zu können, müssen ebenfalls Tests durchgeführt werden. Damit sich der Erkenner bei diesem Prozess nicht an das Test-Set anpasst ('adaptiert'), verwendet man für den Tuning-Process ein weiteres unabhängiges Set von Daten, welches i.a. 'development test set ' genannt wird.

Die zu erledigenden Aufgaben für den Aufbau eines Spracherkenners lauten also:

1. Definition der Aufgabe
2. Aufnahme von Trainings- und Test-Sprache aus der Domäne
3. Aufbereitung von Trainings-, Development- und Test-Set ('labeling')

4. Training der phonetischen HMMs
5. Training des Aussprache-Modells
6. Training des Sprachmodells ('language model')
7. Aufbau des entgültigen Erkenners
8. Tuning des Erkenners
9. Test des Erkenners im Labor
10. (Test des Erkenners im tatsächlichen Einsatz)

Literatur

[1] Schiel F (1998): P6.1 Einführung in die Spracherkennung, Skriptum zur Vorlesung.

[2] Young St (1995): The HTK Book. Revised 1999, University of Cambridge.

[3] Euler St (2006): Grundkurs Spracherkennung. Vieweg Wiesbaden.

[4] Pfister B, Kaufmann T (2008): Sprachverarbeitung - Grundlagen und Methoden der Sprachsynthese und Spracherkennung. Springer-Verlag Berlion Heidelberg.

[5] Jurafsky D, Martin J (2000): Speech and Language Processing. Pearson International, 2nd Edition, 2009, Kapitel 9 und 10.