Hidden Markov Modell

Werner Dichler

BACHELORARBEIT

Nr. 238-003-045-A

eingereicht am
Fachhochschul-Bachelorstudiengang

Hardware Software System Engineering

in Hagenberg

im Juli 2008

Diese Arbeit entstand im Rahmen des Gegenstands

Digitale Signalverarbeitung

im

Wintersemester 2007

Betreuer:

Dr.-Ing. habil. Hans-Georg Brachtendorf

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 3. Juni 2008

Werner Dichler

Inhaltsverzeichnis

Kurzfassung

Bei den Hidden Markov Modellen handelt es sich um stochastische Modelle zur Analyse von Signalquellen. Sie sind ein wichtiger Teil von Spracherkennungsalgorithmen. Ein Hidden Markov Modell besteht aus diversen Zuständen, welche Ausgabezeichen generieren können. Das Modell besitzt gewisse Start-, Übergangs- und Emissionswahrscheinlichkeiten. Da man in der Praxis das Modell nicht direkt vorliegen hat, sonder erst aufbauen muss, sind einige Algorithmen für das Training nötig (z.B. EM-Algorithmus). Für Analysezwecke muss das Modell bewertet und beurteilt werden. Welche inneren Zustände stellen sich ein? Mit welcher Wahrscheinlichkeit hat das Modell die Zeichenfolge generiert? Um diese Algorithmen vollständig verstehen zu können sind einige theoretische Grundlagen notwendig.

Abstract

The Hidden Markov models are stochastic models for the analysis of signal sources. They are important for speech recognition algorithms. The Hidden Markov model consists of various states which can generate output values. The model has certain start, transition and issue probabilities. The model does not exist in the practice directly. It must be build up first and for this, training algorithm are needed (e.g. EM algorithm). For analysis purposes the model must be judged. Which inner states did the model assume and how high is the probability for generating the output values. To be able to understand these algorithms some theoretical bases are necessarily.

Kapitel 1

Einleitung

1.1 Warum HMM

Da eine Sprachverarbeitung in einem Studien-Projekt erarbeitet wird und die Hidden Markov Modelle eine wichtige Rolle in der Sprachverarbeitung darstellen, sollte das Thema besser durchleuchtet werden. Die genauere Ausarbeitung während des Projektes ist aufgrund von Zeitmangel nicht möglich und wird deshalb gesondert als Bachelorarbeit behandelt.

1.2 Zielsetzung

Ziel dieses Dokumentes ist es die HMM theoretisch genau zu erfassen, damit sie des weiteren im Bezug zur Sprachverarbeitung betrachtet werden kann. Damit ein möglichst hoher Nutzen des Dokuments entsteht, sollte auch eine mögliche Anwendung beschrieben und erläutert werden.
Diese Arbeit ist in drei Hauptteile eingeteilt. Zu Beginn wird der theoretische Hintergrund beleuchtet, der im nächsten Kapitel für die Hidden Markov Modelle nötig ist. Abschließend wird noch eine Anwendung in der Sprachverarbeitung erarbeitet, die mit Simulationen für bessere Verständlichkeit erweitert ist.

Kapitel 2

Mathematische Hintergrund

2.1 Wahrscheinlichkeitsrechnung

Zu Beginn werden allgemeine Wahrscheinlichkeits-Grundlagen betrachtet, die im weiteren Verlauf dieses Dokuments benötigt werden. Für die Berechnung mit Wahrscheinlichkeiten sind 3 unterschiedliche Definitionen von Wahrscheinlichkeiten vorhanden.

klassische (a priori) Wahrscheinlichkeit:

$$P(A) = \frac{Zahl\ für\ A\ günstigen\ Fälle}{Zahl\ der\ möglichen\ Fälle}$$

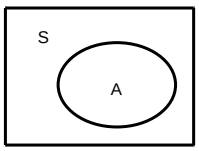
statistische (a posteriori) Wahrscheinlichkeit:

$$P(A) = \frac{k}{n}$$

$k ... absolute\ Häufigkeit\ des\ Ereignisses\ A$
$n ... Durchführhäufigkeit\ des\ Experiment$

geometrische Wahrscheinlichkeit:

$$P(A) = \frac{geometrisches\ Maß\ von\ A}{geometrisches\ Maß\ von\ S}$$



Für weitere Berechnungen sind nachfolgende Formeln äußerst hilfreich. Die bedingte Wahrscheinlichkeit $P(A|B)$ gibt die Wahrscheinlichkeit des Auftretens von A an, mit der Bedingung dass B bereits aufgetreten ist.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Mit der totalen Wahrscheinlichkeit erhält man die Wahrscheinlichkeit von B indem alle bedingten Wahrscheinlichkeiten $P(B|A_i)$ mit den Wahrscheinlichkeiten von A_i multipliziert und aufsummiert werden.

$$P(B) = P(A_1)P(B|A_1) + P(A_2)P(B|A_2) + ... + P(A_n)P(B|A_n)$$

Mit der Ausnutzung der totalen Wahrscheinlichkeit erhält man die Formel von Bayes.

$$P(A_k|B) = \frac{P(A_k)P(B|A_k)}{\sum_{i=1}^{n} P(A_i)P(B|A_i)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Sind Wahrscheinlichkeiten statisch voneinander unabhängig, so gelten folgende Gleichungen.

$$P(A \cap B) = P(A)P(B)$$

$$P(A|B) = \frac{P(A)P(B)}{P(B)} = P(A)$$

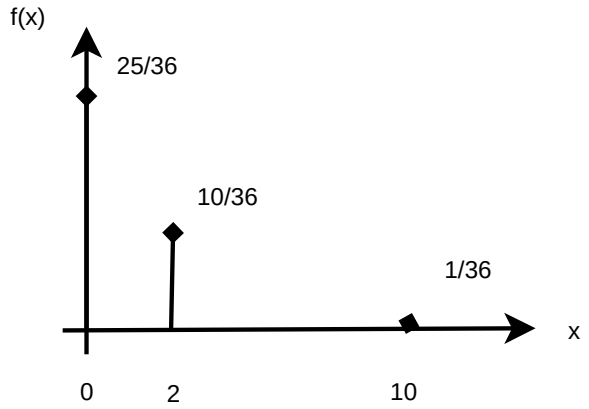


Abbildung 2.1: diskrete Wahrscheinlichkeitsfunktion

2.2 Zufallsvariablen

2.2.1 diskrete Zufallsvariablen

Eine Zufallszahl ist das Ergebnis eines durchgeführten Zufallsprozesses. Die Zufallsvariable stellt eine Funktion dar, welche dem Ergebnis des Zufallsprozesses einen Wert zuweist. Im Bezug auf diskrete Zufallsvariablen gibt es zwei unterschiedliche Wertevorrat Definitionen, endlich und abzählbar unendlich. Der endliche Wertevorrat hat eine bestimmte Anzahl von Werten. Beim abzählbar unendlichen Wertevorrat können die Werte zwar abgezählt werden, des weiteren sind aber unendlich viele Werte vorhanden.

$$W_x = \{x_1 ... x_n\} \ldots endlich$$

$$W_x = \{x_1, x_2, ...\} \ldots abzählbar\ unendlich$$

Die Wahrscheinlichkeitsfunktion $f(x)$ gibt an, mit welcher Wahrscheinlichkeit das Zufallsexperiment den Wert x liefert.

$$f(x) = P(X = x)$$

$$P(a \leq x \leq b) = \sum_{a \leq x_i \leq b} f(x_i)$$

$$P(-\infty < x < \infty) = \sum_{i} f(x_i) = 1$$

Die Verteilungsfunktion $F(x)$ gibt an mit welcher Wahrscheinlichkeit das Zufallsexperiment unter einer bestimmten Schranke x liegt.

$$F(x) = P(X \leq x) = \sum_{x_i \leq x} f(x_i)$$

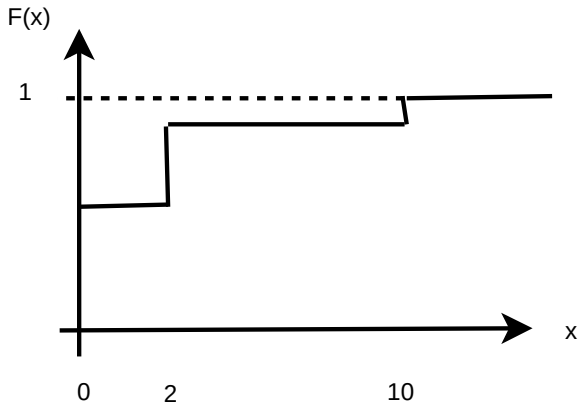


Abbildung 2.2: diskrete Verteilungsfunktion

$$0 \leq F(x) \leq 1$$

$$P(a < X \leq b) = F(b) - F(a)$$

$$f(x_n) = F(x_n) - F(x_n - 1)$$

Der Erwartungswert einer Zufallsvariable gibt den am häufigst auftretenden Wert eines Zufallsprozesses an. Er kann als der Mittelwert des Zufallsprozesses gesehen werden.

$$E(X) = \sum_{i=1}^{n} x_i f(x_i)$$

Die Varianz repräsentiert ein Streumaß, Abweichung der Zufallsvariable von dessen Erwartungswert. Quadratwurzel von der Varianz ergibt die Standardabweichung.

$$Var(X) = E\left[X - E(X)\right]^2 = \sum_i \left(x_i - E(X)\right)^2 f(x)$$

$$Var(X) = E\left[X^2\right] - \left[E(X)\right]^2$$

$$S = \sqrt{Var(X)}$$

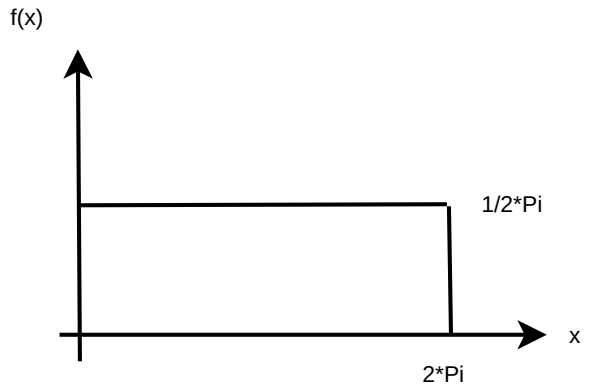


Abbildung 2.3: stetige Wahrscheinlichkeitsfunktion

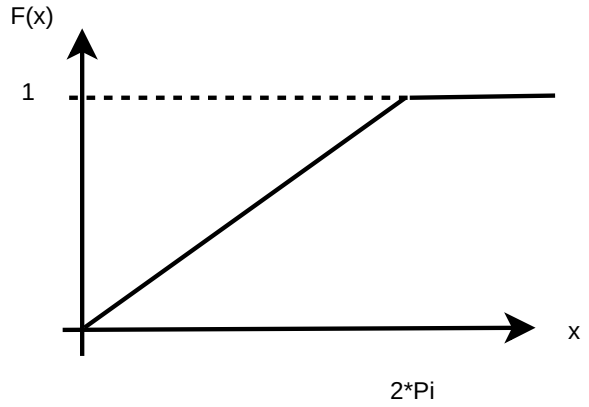


Abbildung 2.4: stetige Verteilungsfunktion

2.2.2 stetige Zufallsvariablen

Der Wertevorrat von stetigen Zufallsvariablen ist im Gegenzug zu den diskreten Zufallsvariablen nicht abzählbar.

$$W_x \ldots \textit{überabzählbar unendlich}$$

Die Wahrscheinlichkeits- und Verteilungsfunktion sind stetige Funktionen. Von einem bestimmten Wert x kann keine Wahrscheinlichkeit angegeben werden, stattdessen wird die Wahrscheinlichkeit von einem sehr kleinen Bereich angegeben.

$$F(x) = \int_{-\infty}^{x} f(u)du$$

$$\int_{-\infty}^{\infty} f(n)dn = 1$$

$$P(a < x \leq b) = \int_{a}^{b} f(n)dn = F(b) - F(a)$$

$$P(x > c) = \int_{c}^{\infty} f(n)dn = 1 - F(c)$$

Beim Erwartungswert und bei der Varianz von stetigen Zufallsvariablen wird die Summe durch ein Integral ersetzt.

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx$$

$$Var(X) = \int_{-\infty}^{\infty} (x - E(X))^2 f(x)dx$$

2.3 Normalverteilung

Die Normalverteilung ist eine oft verwendete Wahrscheinlichkeitsverteilung, sie wird auch Gauß-Verteilung genannt.

$$f(x) = N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{1}{2}(\frac{u-\mu}{\sigma})^2} du$$

$$E(X) = \mu$$

$$Var(X) = \sigma^2$$

Bei Berechnungen mit Normalverteilungen wird oft die normierte Form verwendet. Für diese existieren bereits Tabellen mit den wichtigsten Werten.

$$\Phi(x) = N(x|0, 1)$$

$$F(x) = \Phi(\frac{x-\mu}{\sigma})$$

Nachfolgende Formeln sind oft hilfreich bei Umformungen für Berechnungen mit Normalverteilungen.

$$P(Z \leq z) = \Phi(z)$$

$$P(Z > z) = 1 - \Phi(z)$$

$$P(Z \leq -z) = \Phi(-z) = 1 - \Phi(z)$$

$$P(Z > -z) = \Phi(z)$$

$$P(z_1 < Z \leq z_2) = \Phi(z_2) - \Phi(z_1)$$

$$P(-z < Z < z) = 2\Phi(z) - 1$$

$$P(-z > Z > z) = 2(1 - \Phi(z))$$

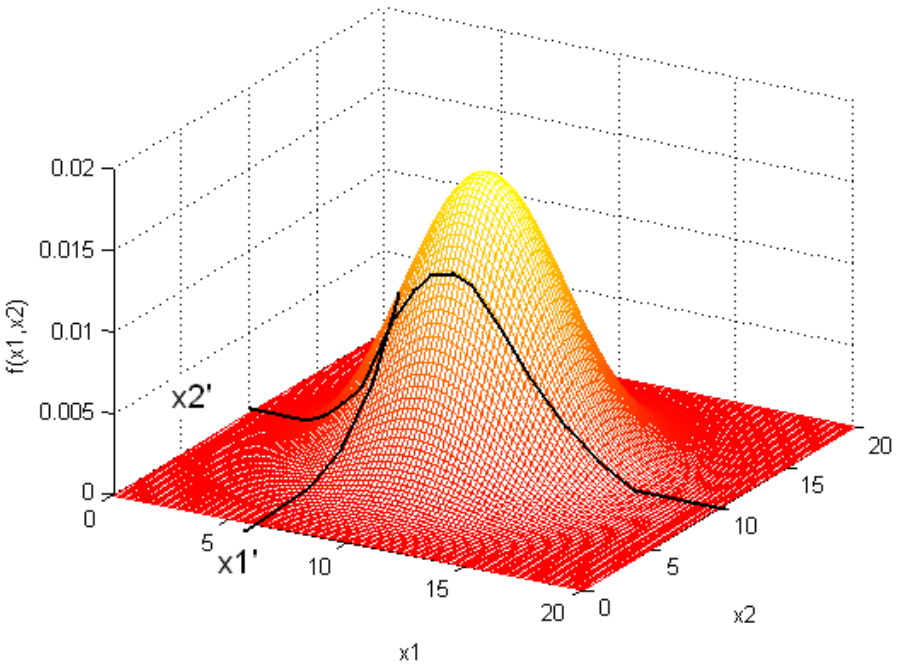


Abbildung 2.5: 2 dimensionale Wahrscheinlichkeitsfunktion

2.4 Vektorwertige Größen

Da viele Bereiche der Markov Modelle in der Praxis mit vektorwertigen Größen berechnet werden, werden diese hier kurz betrachtet.

Die vektorwertige Verteilungsfunktion F_x berechnet sich analog zur eindimensionalen Verteilungsfunktion, indem für jede Dimension ein Integral gebildet wird. Die Grenzen der Integrale werden durch den Eingabevektor $\vec{x}$ bestimmt.

$$F_x(\vec{x}) = P(x_1 \leq x_1', x_2 \leq x_2', \ldots x_n \leq x_n')$$
$$= \int_{-\infty}^{x_1'} \int_{-\infty}^{x_2'} \ldots \int_{-\infty}^{x_n'} f_x(x_1, x_2, \ldots x_n) dx_1 dx_2 \ldots dx_n$$

Vektor mit 2 Größen :

$$F(x_1', x_2') = \int_{-\infty}^{x_1'} \int_{-\infty}^{x_2'} f(x_1, x_2) dx_1 dx_2$$

Die Wahrscheinlichkeitsfunktion f_{xi} gibt die Wahrscheinlichkeit an, dass xi einen bestimmten Wert x annimmt, wobei die anderen Größen des Vektors

einen beliebigen Wert annehmen können.

$$f_{xi}(x) = \int\limits_{x_1,\ldots x_{i-1},x_{i+1},\ldots x_n} f_x(x_1, x_2, \ldots x_n) dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_n$$

Vektor mit 2 Größen :
$f_{x_1}(x_1') = \int\limits_{x_2} f(x_1', x_2) dx_2$
$f_{x_2}(x_2') = \int\limits_{x_1} f(x_1, x_2') dx_1$

Die Wahrscheinlichkeitsfunktion f_x gibt die Wahrscheinlichkeit an, dass x_1 den Wert x_1', x_2 den Wert x_2', ... x_n den Wert x_n' annimmt. Bei statistischer Unabhängigkeit kann einfach das Produkt über alle Teilwahrscheinlichkeitsfunktionen gebildet werden.

$$f_x(\vec{x}) = f_x(x_1', x_2', \ldots x_n') = \prod_{i=1}^{n} f_{xi}(x_i')$$

Vektor mit 2 Größen :
$f_x(\vec{x}) = f_x(x_1', x_2') = f_{x_1}(x_1') f_{x_2|x_1}(x_2'|x_1') = f_{x_2}(x_2') f_{x_1|x_2}(x_1'|x_2')$

Der Erwartungswert und die Varianz sind ebenfalls vektorwertige Größen. Die Formeln zur Berechnung ändern sich bis auf die vektorwertigen Variablen nicht. Ebenso kann die Normalverteilung vektormäßig angegeben werden.

$$E(\vec{X}) = \int \vec{X} f_x(\vec{x}) d\vec{x}$$

Vektor mit 2 Größen :
$$E(x_1, x_2) = \begin{pmatrix} \int\int x_1 f_x(x_1, x_2) dx_1 dx_2 \\ \int\int x_2 f_x(x_1, x_2) dx_1 dx_2 \end{pmatrix}$$

$$Var(\vec{X}) = E\left[\left(\vec{X} - E(\vec{X})\right)\left(\vec{X} - E(\vec{X})\right)^T\right]$$
$$= \int \left(\vec{x} - E(\vec{X})\right)\left(\vec{x} - E(\vec{X})\right)^T f_x(\vec{x}) dx$$

$$Var(\vec{X}) = E(\vec{X}\vec{X}^T) - E(\vec{X})E(\vec{X})^T$$

$$N(\vec{x}|\vec{\mu}, \vec{K}) = \frac{1}{\sqrt{|2\pi\vec{K}|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \vec{K}^{-1}(\vec{x}-\vec{\mu})}$$

Beliebige Verteilungsfunktionen werden oft durch Mischverteilungen dargestellt. Somit werden mehrere Normalverteilungen verwendet um eine spezielle Verteilung zu modellieren.

$$p(x) = \sum_{i=1}^{\infty} c_i N(\vec{x}|\vec{\mu_i}, \vec{K_i}) \quad mit \sum_{i=1}^{\infty} c_i = 1$$

$c_i \ldots Mischungsgewichte$

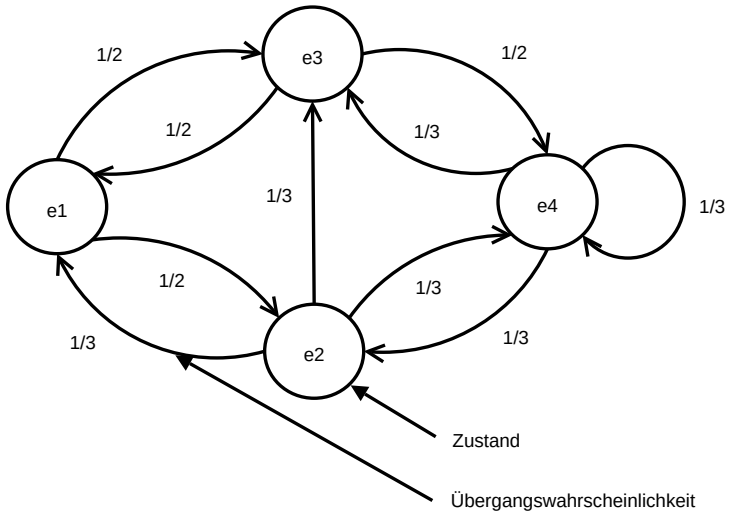


Abbildung 2.6: Zustandsgraph (Kreisdiagramm)

2.5 Markov Ketten

Mit stochastischen Modellen versucht man zukünftige Entwicklungen von Prozessen vorherzusagen. Häufig sind solche stochastischen Prozesse zyklisch. Mit den Markov Ketten wird ein stochastischer Prozess über einen längeren Zeitraum betrachtet. Die Markov Kette ist durch die Anfangsverteilung, den Übergangswahrscheinlichkeiten und den Zustandsraum definiert $(p(0), P, M)$. Das nachfolgende Beispiel verwendet vier Zustände.

$$M = \{e_1, e_2, e_3, e_4\}$$

$$P(e_i \rightarrow e_k) = P(e_k|e_i) = p_{ik}$$

$$\sum_{k=1}^{N} p_{ik} = 1$$

Die einzelnen Zustände müssen voneinander unabhängig sein. Ist eine Übergang von einem Zustand zu dem Nächsten nicht möglich so ist $p_{ik} = 0$. Die Summe aller Übergangswahrscheinlichkeiten je Knoten müssen immer 1 ergeben.

$e_i \rightarrow e_k$	e_1	e_2	e_3	e_4
e_1	0	1/2	1/2	0
e_2	1/3	0	1/3	1/3
e_3	1/2	0	0	1/2
e_4	0	1/3	1/3	1/3

Tabelle 2.1: Übergangswahrscheinlichkeiten

Die Übergangswahrscheinlichkeiten in Tabellenform kann man auch als stochastische Übergangsmatrix anschreiben. Die Matrix hat den Zeilenindex i und den Spaltenindex k. Ein Element p_{ik} gibt die Wahrscheinlichkeit an dass nach dem Zustand e_i der Zustand e_k folgt.

$$P = \begin{pmatrix} p_{ik} & \cdots \\ \vdots & \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/3 & 0 & 1/3 & 1/3 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 1/3 & 1/3 & 1/3 \end{pmatrix}$$

Bei mehrstufigen Übergängen müssen alle Wege, die vom Start-Zustand zum End-Zustand führen, berücksichtigt werden. Um dessen Wahrscheinlichkeit zu ermitteln werden die Übergangswahrscheinlichkeiten eines Pfades multipliziert und die Übergangswahrscheinlichkeiten aller Pfade summiert. Diese Wahrscheinlichkeiten können ebenfalls durch eine Matrix $P(n)$ angegeben werden.

$$e_i \rightarrow e_k \ldots p_{ik}(1) = p_{ik}$$
$$e_i \rightarrow e_j \rightarrow e_k \ldots p_{ik}(2)$$
$$\vdots$$

$$p_{14}(2) = 1/2 \cdot 1/2 + 1/2 \cdot 1/3 = 5/12$$

$$P(2) = \begin{pmatrix} p_{ik}(2) & \cdots \\ \vdots & \end{pmatrix}$$

$$p_{ik}(n+1) = \sum_{j=1}^{N} p_{ij}(n) \cdot p_{jk} = \sum_{j=1}^{N} p_{ij} \cdot p_{jk}(n)$$

$$P(n+1) = P(n) \cdot P = P \cdot P(n)$$

$$P(n) = P^n$$

Die Anfangsverteilung wird durch einen Vektor repräsentiert und gibt die Startwahrscheinlichkeiten an. Bei einem zufälligen Start haben alle möglichen Zustände die gleiche Startwahrscheinlichkeit, bei einem bestimmten Start bekommt ein Zustand die Wahrscheinlichkeit 1. Ebenso können spätere Wahrscheinlichkeiten der einzelnen Zustände angegeben werden. Die Summe aller Wahrscheinlichkeiten der Vektoren muss immer 1 ergeben.

$$\vec{p}(0) = (p_1(0), p_2(0), p_3(0), p_4(0))$$

$$\vec{p}(0) = (1/4, 1/4, 1/4, 1/4) \ldots zufälligerStart$$
$$\vec{p}(0) = (0, 0, 1, 0) \ldots bestimmterStart$$

$$\vec{p}(n) = (p_1(n), p_2(n), p_3(n), p_4(n))$$

$$p_i(n) = \sum_{j=1}^{N} p_j(0) \cdot p_{ji}(n)$$

$$\vec{p}(n) = \vec{p}(0) \cdot P^n$$

Betrachtet man spätere Wahrscheinlichkeiten, so stellt man fest dass das System gegen gewisse Grenzwerte strebt. Diese werden durch die Grenzmatrix P^∞ und die Grenzverteilung $\vec{p}^\infty$ veranschaulicht.

$$P^\infty = \lim_{n\to\infty} P^n = \begin{pmatrix} \lim\limits_{n\to\infty} p_{ik}(n) & \ldots \\ \vdots & \end{pmatrix}$$

$$\vec{p}^\infty = \lim_{n\to\infty} \vec{p}(n) = \lim_{n\to\infty} (p_1(n), \ldots)$$

2.6 Maximum Likelihood Methode

In den nächsten Kapiteln wurden die formalen Beschreibungen überwiegend aus dem Buch [4] entnommen. Bei der Maximum Likelihood Methode handelt es sich um ein Schätzverfahren, welches Parameter einer Verteilung berechnet. $f(x)$ ist die Dichtefunktion die vom Parameter q abhängig ist. Um die Parameterschätzung durchführen zu können werden Beispieldaten bzw. Stichproben der Zufallsvariable benötigt $(x_1, x_2, \ldots)$, diese müssen voneinander unabhängig sein. Die Likelihood Funktion $L(q|x_1, \ldots x_n)$ betrachtet die Dichte als Funktion von q mit festen Realisationen x_i.

$$f(x_1, \ldots x_n|q) = \prod_{i=1}^{n} f_{x_i}(x_i|q)$$

$$L(q|x_1, \ldots x_n) = \prod_{i=1}^{n} f_{x_i}(x_i|q)$$

Damit man den plausibelsten/günstigsten Parameterwert für q erhält wird die Funktion auf q maximiert (erste Ableitung nach q = 0). Für einfachere Berechnung wird der ln gebildet der das Maximum nicht beeinflusst.

$$L(q|x_1, \ldots x_n) = \ln \prod_{i=1}^{n} f_{x_i}(x_i|q) = \sum_{i=1}^{n} \ln f_{x_i}(x_i|q)$$

$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$f(x_1, \ldots x_n|\mu, \sigma) = \prod_{i=1}^{n} f(x_i|\mu, \sigma)$$

$$f(x_1, \ldots x_n|\mu, \sigma) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{\frac{-\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}$$

$$f(x_1, \ldots x_n|\mu, \sigma) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{\frac{-\sum(x_i-\bar{x})^2+n(\bar{x}-\mu)^2}{2\sigma^2}}$$

$\bar{x} \ldots Mittelwert$

nach μ und σ maximieren:

$$\frac{d}{d\mu}\ln(\ldots) = \frac{d}{d\mu}(\ln(\frac{1}{2\pi\sigma^2})^{\frac{n}{2}} - \frac{-\sum(x_i-\bar{x})^2+n(\bar{x}-\mu)^2}{2\sigma^2}) = 0 - \frac{-2n(\bar{x}-\mu)}{2\sigma^2}$$

$$\frac{d}{d\sigma}\ln(\ldots) = \frac{d}{d\sigma}(\frac{n}{2}\ln(\frac{1}{2\pi\sigma^2}) - \frac{\sum(x_i-\mu)^2}{2\sigma^2}) = -\frac{n}{\sigma} + \frac{\sum(x_i-\mu)^2}{\sigma^3}$$

Gleichungen 0 setzten:

$$\mu = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

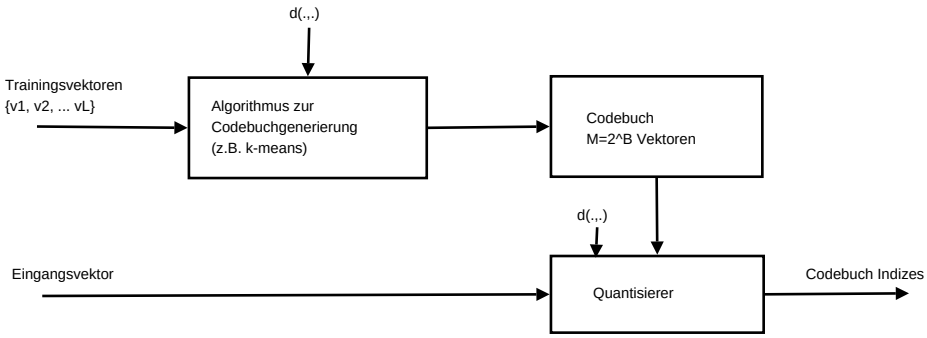


Abbildung 2.7: Ablaufschema

2.7 Vektorquantisierer

Zweck eines Vektorquantisierers ist es Eingabedaten möglichst gut und kompakt durch Repräsentanten abzubilden. Das Codebuch ist ein Teil des Vektorquantisierers, welches aus Prototypenvektoren besteht. Die, mittels Index durchnummerierten, Prototypenvektoren bilden die Repräsentanten für Merkmalsvektoren. Der Vektorquantisierer weist jedem Merkmalsvektor einen Eintrag des Codebuches zu und gibt dessen Index zurück. Die Herausforderung ist es möglichst gute Repräsentanten für die Merkmalsvektoren zu finden.

$$q : \left\{ \begin{array}{l} R^D \rightarrow Z = \{Z_1, Z_2, \ldots Z_n\} \\ x \rightarrow q(x) \end{array} \right\}$$

$q \ldots Quantisierer$
$R^D \ldots D\ dimensionaler\ Merkmalsraum$
$Z \ldots Codebuchvektor\ mit\ n\ Partitionen$

Durch genügend Trainingsvektoren (ca. 10M) wird mit einem geeigneten Algorithmus und einem gewählten Abstandsmaß $d(.,.)$ ein Codebuch der Größe M generiert. Der Quantisierer verwendet das generierte Codebuch und ermittelt mit dem gewählten Abstandsmaß für einen Eingangsvektor dessen zugehörige Codebuch Indizes.

Als Abstandsmaß wird oft die Minimumregel verwendet. Ist der Abstand eines Vektors c bei dem Prototypenvektor z_1 am kleinsten, so wird c durch z_1 abgebildet.

$$||\vec{c} - \vec{z_1}||^2 \leq ||\vec{c} - \vec{z_k}||^2 \quad für\ alle\ k \neq 1$$

$$d(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||^2 = \sum_{i=1}^{L} (x_i - y_i)^2$$
$$wobei\ \vec{x} = (x_1, x_2, \ldots x_L)\ und\ \vec{y} = (y_1, y_2, \ldots y_L)$$

2.7.1 Lloyd Algorithmus

Der Lloyd Algorithmus ist ein Algorithmus zur Codebuch-Generierung. Er verwendet das Prinzip des Vektorquantisierers und setzt es iterativ ein. In der Initialisierungsphase wird das Codebuch durch geeignete Wahl vorinitialisiert, z.B. durch zufälliges Auswählen von Vektoren aus den Trainingsvektoren. Des weiteren wird der Iterationszähler auf 0 gesetzt. Im zweiten Schritt wird die Partition R optimiert indem alle Trainingsvektoren einer Partition zugeordnet werden und danach für das aktuelle Codebuch Y^m die optimale Partition bestimmt wird. Dabei wird zugleich der mittlere Quantisierungsfehler $\bar{\epsilon}$ ermittelt. Die nächste Phase befasst sich mit der Aktualisierung des Codebuches, welches durch die Zentroiden der Partitionen gebildet wird. In der Abschlussphase wird geprüft ob sich eine relative Verbesserung ergeben hat und so von Neuem begonnen wird oder ob der Algorithmus terminiert wird.

$w = \{x_1, x_2, \dots x_T\}$ ... Beispielvektoren
$Y^m = \{y_1, y_2, \dots y_N\}$ ... Codebuch
N ... gewünschte Codebuchgröße
$\Delta\epsilon_{min}$ ... untere Schranke für relative Verbesserung des Quantisierungsfehlers
R_i^m ... Partitionen die den Merkmalsraum aufteilen

Initialisierung

initiale Codebuch Y^0 wählen
Iterationszähler m auf 0 setzen

Optimierung der Partition

Zuordnen aller Trainingsvektoren an eine Zelle der Partition
für aktuelle Codebuch Y^m optimale Partition bestimmen

$$R_i^m = \left\{ \vec{x} | \vec{y_i}^m = \arg \min_{\vec{y} \in Y^m} d(\vec{x}, \vec{y}) \right\}$$

mittleren Quantisierungsfehler bestimmen

$$\bar{\epsilon}(Y^m) = \frac{1}{T} \sum_{t=1}^{T} \min d(\vec{x_t}, \vec{y})$$

$$d(\vec{x}, \vec{y}) = |\vec{x} - \vec{y}|^2$$

Aktualisierung des Codebuches

aus Zellen R_i^m neues Codebuch erzeugen
$y_i^{m+1} = cent(R_i^m)$
$Y^{m+1} = \{y_i^{i+1} | 1 \leq i \leq N\}$
$cent(R) = \frac{1}{R} \sum x$

Terminierung

$\Delta\epsilon_m = \frac{\bar{\epsilon}(Y^{m-1}) - \bar{\epsilon}(Y^m)}{\bar{\epsilon}(Y^m)}$
$\Delta e_m > \Delta e_{min} \rightarrow m = m + 1$ sonst Terminierung

Der Lloyd Algorithmus ist zwar einfach zu implementieren und der Optimierungsgrad ist frei wählbar, dennoch ist er in der Praxis oft unbrauchbar da er sehr Rechenintensiv ist. Des weiteren hängt die Qualität des Ergebnises von der Wahl des Initial-Codebuches ab.

Beispiel Codebuchgenerierung

Für besseres Verständnis des Lloyd Algorithmus wird in diesem Abschnitt ein Codebuch, mit der Größe 2, aus 9 Trainingsdaten berechnet.

$w = \{1, 2, 1, 2, 2, 8, 9, 8, 8\}$
$Y^0 = \{1, 2\}$

1. *Iteration* :
$y_1^0 = 1 : 1, 1$
$y_2^0 = 2 : 2, 2, 2, 8, 9, 8, 8$
$R_1^0 = 2$
$R_2^0 = 7$
$y_1^1 = \frac{2}{2} = 1$
$y_2^1 = \frac{39}{7} = 5.57 \Rightarrow Y^1 = \{1, 5.57\}$
$\bar{\epsilon}(Y^0) = \frac{1}{9}(0 + 0 + 0 + 0 + 0 + 36 + 49 + 36 + 36) = 17.44$

2. *Iteration* :
$y_1^1 = 1 : 1, 2, 1, 2, 2$
$y_2^1 = 2 : 8, 9, 8, 8$
$R_1^1 = 5$
$R_2^1 = 4$
$y_1^2 = \frac{8}{5} = 1.6$
$y_2^2 = \frac{33}{4} = 8.25 \Rightarrow Y^2 = \{1.6, 8.25\}$
$\bar{\epsilon}(Y^1) = \frac{1}{9}(0 + 1 + 0 + 1 + 1 + 5.90 + 11.76 + 5.90 + 5.90) = 3.61$

...

2.7.2 LBG Algorithmus

Der LBG Algorithmus verbessert den zu Grunde liegenden Lloyd Algorithmus mit einem fest definierten Start-Codebuch. Das Codebuch besteht zu Beginn aus einem Vektor (z.B. Zentroid aus Trainingsvektoren) und wird im Laufe des Algorithmus zur Größe N aufgebaut. Bei jedem Durchlauf verdoppelt sich die Größe des Codebuches.

$$N^{m+1} = 2N^m \ldots \textit{Anzahl neuer Repräsentanten}$$

$$Y^{m+1} = \{y_1 + \epsilon, y_1 - \epsilon, y_2 + \epsilon, y_2 - \epsilon, \ldots y_{N^m} + \epsilon, y_{N^m} - \epsilon\}$$
$$\epsilon \ldots \textit{geeigneter, betragsmäßig kleiner Störvektor}$$

Das neue Codebuch wird mittels Lloyd Algorithmus optimiert. Terminiert wird der Algorithmus nach dem Erreichen der gewünschten Codebuch-Größe.

2.7.3 k-means Algorithmus

Die beiden vorherigen Algorithmen sind in der Implementierung äußerst ineffizient, da das gesamte Codebuch öfters bewertet werden muss. Der k-means Algorithmus ist ein einfacher aber dennoch guter Algorithmus zum Erstellen eines Codebuches. Das initiale Codebuch wird von den ersten N Trainingsvektoren gebildet und wird in T-N Schritten weiters optimiert. Jeder Durchlauf betrachtet nur einen Trainingsvektor, gestartet wird bei x_{N+1}. Der Optimierungsalgorithmus basiert auf dem vom Lloyd. Ein großer Vorteil entsteht durch die Zuordnung des einen Trainingsvektors, denn dadurch ändert sich auch nur die Partition und das Codebuch an einer Stelle.

Initialisierung

$Y^0 = \{x_1, x_2, \ldots x_N\}$
Iterationszähler m auf 0 setzen

Iteration

für restliche x_t wird der beste Repräsentant aus Codebuch gesucht
$y_i^m = \arg\min d(x_t, y)$

jeweilige Partition aktualisieren
$$R_j^{m+1} = \left\{ \begin{array}{l} R_j^m \cup \{x_t\} \ldots j = i \\ R_j^m \ldots sonst \end{array} \right\}$$

Codebuch an genau dieser Stelle aktualisieren
$$y_j^{m+1} = \left\{ \begin{array}{l} cent(R_j^{m+1} \ldots j = i \\ y_j^m \ldots sonst \end{array} \right\}$$

Terminierung

Terminierung wenn alle Vektoren abgearbeitet
$x_t \; mit \; N < t \leq T$

2.7.4 Mischverteilungen

Alle zuvor erläuterten Algorithmen verwenden ein Modell welches N Repräsentanten zur Darstellung der Merkmalsvektoren verwenden. Dieses Modell ist oft zu ungenau, deshalb werden stattdessen Mischverteilungen verwendet (z.B. Normalverteilungen). Die einfachste Variante um die Parameter der Mischverteilungen zu bestimmen erfolgt nach der Vektorquantisierung aus den letzten Punkten. Die Zentroiden y_i stellen zugleich die Mittelwertvektoren μ_i dar. Somit müssen nur die Kovarianzmatrizen K_i berechnet werden.

$$K_i = \sum_{x \in R_i} (x - \mu_i)(x - \mu_i)^T$$

Da dieses Verfahren zwar einfach aber unzureichend ist versucht man die Mischverteilungsberechnung bereits während der Quantisierung durchzuführen. Hierzu muss das Abstandsmaß angepasst und die Zuordnung der Trainingsvektoren zu den Partitionen angepasst werden.

$$d_{Mahalanobis} = (x - \mu)^T K^{-1} (x - \mu)$$

$$R_i = \left\{ x | i = \arg\max_j N(x|\mu_j, K_j) \right\}$$

Diese Anpassung wäre dennoch unbrauchbar, da die Algorithmen den mittleren Quantisierungsfehler zu minimieren versuchen. Dieses Optimierungskriterium ist für eine Mischverteilung aber unbrauchbar, da es zu keinen optimalen Ergebnissen führt.

2.7.5 EM Algorithmus

Die Problematik der Mischverteilung aus dem letzten Kapitel kann mittels EM Algorithmus gelöst werden. Zuvor wird der allgemeine Aufbau des Algorithmus erläutert, danach wird er speziell bei Quantisierern mit Mischverteilungen angewendet.

Der EM Algorithmus ist für Parameterschätzungen gedacht, wobei einige Daten nicht vorhanden sind. Somit müssen bei jedem Durchlauf diese fehlenden Daten geschätzt werden, des weiteren können die Parameter mittels Likelihood maximiert werden.
Die fehlenden Daten werden innerhalb der Q-Funktion mit den Endparametern θ aus der vorherigen Runde und den vorhandenen Daten y geschätzt. Für die Maximierung der Parameter mittels Likelihood Methode wird der Standard-Ansatz mit den Wahrscheinlichkeiten der fehlenden Daten gewichtet (diese Gewichtung hat keinen Einfluss auf das Maximum). Die Terminierung erfolgt nachdem bestimmte Parameter während zwei Durchläufen nicht mehr verändert wurden.

$$z \ldots fehlende\ Daten$$
$$y \ldots unvollständige\ Daten$$
$$z + y \ldots vollständige\ Daten$$
$$\theta \ldots Parameter\ der\ Verteilung$$

$$p(z|y,\theta) = \frac{p(y,z|\theta)}{p(y|\theta)} = \frac{p(y|z,\theta)p(z|\theta)}{\int p(y|\hat{z},\theta)p(\hat{z}|\theta)dz}$$

$$Q(\theta) = \sum_z p(z|y,\theta) \log p(y,z|\theta)$$

$$\theta_{n+1} = \arg\max_\theta Q(\theta)$$

genauere Herleitung [3]:

$$p(z|(y,\theta)) = \frac{p(z,(y,\theta))}{p(y,\theta)} = \frac{p((z,y),\theta)}{p(y|\theta)p(\theta)} = \frac{p((z,y)|\theta)p(\theta)}{p(y|\theta)p(\theta)} = \frac{p((z,y)|\theta)}{p(y|\theta)}$$

$$p(y,z|\theta) = \frac{p(y,z,\theta)}{p(\theta)} = \frac{p(y|z,\theta)p(z,\theta)}{p(\theta)} = \frac{p(y|z,\theta)p(z|\theta)p(\theta)}{p(\theta)} = p(y|z,\theta)p(z|\theta)$$

$$p(x) = \int_{-\infty}^{\infty} p(x,y)dy \ldots Randverteilung$$
$$\int_{-\infty}^{\infty} p(y,z,\theta)dz = p(y,\theta) = p(y|\theta)p(\theta)$$
$$\int_{-\infty}^{\infty} p(y,z,\theta)dz = \int_{-\infty}^{\infty} p(y|z,\theta)p(\theta)dz$$
$$somit:$$
$$p(y|\theta) = \int_{-\infty}^{\infty} p(y|z,\theta)p(z|\theta)dz$$

Der EM Algorithmus für die Berechnung der Parameter $\theta = (c_i, \mu_i, K_i)$ von N Mischverteilungen benötigt die Ergebnisse eines vorherigen Vektorquantisierers. Nach der Initialisierung wird für jedes x aus den Trainingsvektoren w die Wahrscheinlichkeit für die Zuordnung zu einer der Codebuchklassen w_i berechnet. Für die Beurteilung der Verbesserung durch eine Iteration wird zugleich die Likelihood L des aktuellen Modells θ^m berechnet. Im nächsten Schritt werden die Parameter θ^{m+1} mittels Maximierung aktualisiert. Terminiert wird der Algorithmus wenn die relative Verbesserung der Likelihood größer als ein definiertes Minimum ist.

Initialisierung

$\theta^0 = (c_i^0, \mu_i^0, K_i^0)$ bestimmen (z.B. durch k-means Alg.)
Iterationszähler m auf 0 setzen

Schätzung

für jedes x $\in$ w Wahrscheinlichkeit berechnen
$P(w_i|x, \theta^m) = \frac{c_i^m N(x|\mu_i^m, K_i^m)}{\sum_j c_j^m N(x|\mu_j^m, K_j^m)}$

Likelihood für Bewertung berechnen
$L(\theta^m|w) = \ln p(x_1, x_2, \ldots x_T|\theta^m) = \sum_{x \in w} ln \sum_j c_j^m N(x|\mu_j^m, K_j^m)$

Maximierung

neue Parameter berechnen θ^{m+1}
$c_i^{m+1} = \frac{\sum_{x \in w} P(w_i|x,\theta^m)}{|w|}$

$\mu_i^{m+1} = \frac{\sum_{x \in w} P(w_i|x,\theta^m)x}{\sum_{x \in w} P(w_i|x,\theta^m)}$

$K_i^{m+1} = \frac{\sum_{x \in w} P(w_i|x,\theta^m)xx^T}{\sum_{x \in w} P(w_i|x,\theta^m)} - \mu_i^{m+1}(\mu_i^{m+1})^T$

Terminierung

Terminierung wenn $\Delta L_m \leq \Delta L_{min}$
$\Delta L_m = \frac{L(\theta^m|w) - L(\theta^{m-1}|w)}{L(\theta^m|w)}$

Kapitel 3

Hidden Markov Modell

Bei den Hidden Markov Modellen handelt es sich um stochastische Modelle zur Analyse von Signalquellen und wird oft im Zusammenhang mit der Sprachverarbeitung verwendet. Das Modell besteht aus Zuständen die mit einer bestimmten Wahrscheinlichkeit zu einem anderen Zustand wechseln können. Innerhalb eines Zustandes gibt es Wahrscheinlichkeiten mit welche es definierte Zeichen ausgibt.

$T \ldots Länge\ der\ beobachteten\ Zeichenfolge$
$N \ldots Anzahl\ der\ Zustände\ im\ Modell$
$M \ldots Anzahl\ der\ verschiedenen\ Zeichen$
$Q = \{q_1, q_2, \ldots q_N\} \ldots Zustände$
$V = \{v_1, v_2, \ldots v_M\} \ldots diskrete\ Menge\ von\ möglichen\ Zeichen$
$A = \{a_{ij}\} \ldots Zustandsübergangswahrscheinlichkeiten$
$a_{ij} = P(i_{t+1} = j | i_t = i)$
$B = \{b_j(k)\} \ldots Wahrscheinlichkeitsverteilung\ der\ Zeichen\ im\ Zustand\ j$
$b_j(k) = P(O_t = v_k | i_t = j)$
$\pi = \{\pi_i\} \ldots Anfangsverteilung\ der\ Zustände$
$\pi_i = P(i_{t=1} = i)$
$O = O_1, O_2, \ldots O_T \ldots Beobachtete\ Zeichenfolge$
$I = i_1, i_2, \ldots i_T \ldots eingenommene\ Zustände$

Generation einer Zeichenfolge:

1. Anfangszustand durch Anfangsverteilung wählen
2. setze t=1
3. Zeichen durch Wahrscheinlichkeitsverteilung der Zeichen im Zustand wählen
4. Folgezustand durch Übergangswahrscheinlichkeiten wählen
5. setze t=t+1
6. solange t < T wiederhole bei Punkt 3

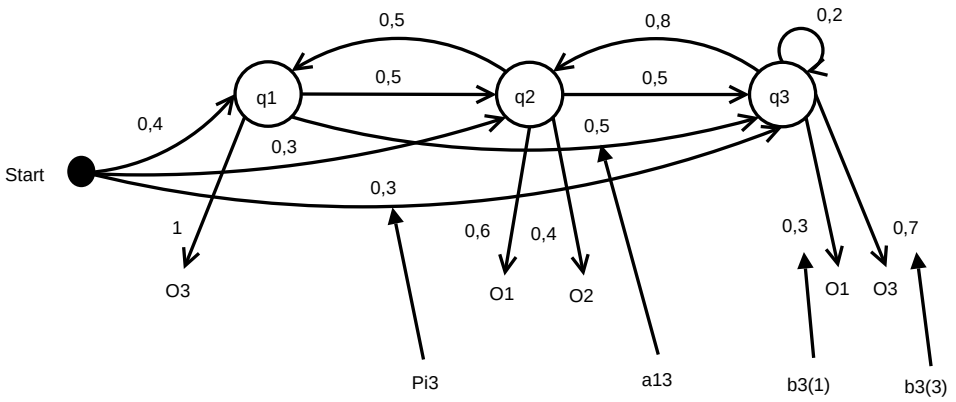


Abbildung 3.1: Hidden Markov Modell

Das Modell hat zwei Besonderheiten, denn der nächste Zustand hängt nur vom aktuellen Zustand ab und das beobachtete Zeichen hängt nur vom aktuellen Zustand ab.

$$P(I_{t+1}|I_t, I_{t-1}, \ldots I_1) = P(I_{t+1}|I_t)$$

$$P(O_{t+1}|I_{t+1}, I_t, \ldots I_1, O_t, O_{t-1}, \ldots O_1) = P(O_{t+1}|I_{t+1})$$

Betrachtet man das Modell so kommt man auf 3 zu lösende Probleme:

- Wahrscheinlichkeit dass eine Zeichenfolge O von einem Modell $\lambda = (A, B, \pi)$ generiert wurde → Bewertung
- optimale Folge von Zuständen um mit einem Modell λ die Zeichenfolge O zu generieren → Beurteilung des inneren Zustandes
- wählen der Modellparameter λ um Wahrscheinlichkeit $P(O|\lambda)$ zu maximieren → Training

3.1 HMM - Bewertung

Bei der Bewertung ist die zu bewertende Zeichenfolge O und das Modell $\lambda = (A, B, \pi)$ bekannt. Gesucht ist mit welcher Wahrscheinlichkeit die Zeichenfolge von dem vorliegenden Modell generiert wurde. Eine einfache Weise das Problem zu lösen besteht darin alle möglichen Zustandsfolgen, welche die Ausgangsfolge generieren, zu ermitteln und deren Wahrscheinlichkeit summieren. Diese Berechnung ist mit einem hohen Aufwand verbunden und daher ungeeignet für praktische Realisierungen.

$$P(O|\lambda) = \sum_I P(O|I,\lambda)P(I|\lambda)$$

$P(O|I,\lambda) = b_{i1}(O_1)b_{i2}(O_2)\dots b_{iT}(O_T)$
$\dots Wahrscheinlichkeit\ für\ Zustandsfolge\ I = i_1, i_2, \dots i_T\ dass\ O\ beobachtet\ wird$

$P(I|\lambda) = \pi_{i1}a_{i1i2}a_{i2i3}\dots a_{i(T-1)iT}$
$\dots Wahrscheinlichkeit\ für\ Zustandsfolge\ I$

Eine effizientere Lösung des Problems bietet der Forward Algorithmus. Bei diesem wird eine Vorwärtsvariable $\alpha_t(q)$ verwendet. Diese gibt die Wahrscheinlichkeit an, dass mit einem Modell λ die Zeichenfolge O_t erzeugt und der Zustand q erreicht wird. Dieser Algorithmus dient also zur mathematisch exakten Ermittlung der Gesamtproduktionswahrscheinlichkeit von O.

$$\alpha_t(q) = P(O_1, O_2, \dots O_t, I_t = q|\lambda)$$

Initialisierung

$$\alpha_1(q) = \pi_q b_q(O_1)$$

Rekursion

$$\alpha_{t+1}(g) = \sum_q \{\alpha_t(q)a_q g\}\, b_g(O_{t+1})$$

Abschluss

$$P(O|\lambda) = \sum_{q=1}^{N} \alpha_T(q)$$

Die Berechnung der optimalen Produktionswahrscheinlichkeit ist wie der Forward Algorithmus aufgebaut, nur dass die optimale Wahrscheinlichkeit betrachtet wird. Dieser Algorithmus bildet ebenfalls ein Bewertungsmaß für die Modellierungsgüte.

$$P^*(O|\lambda) = P(O, I^*|\lambda) = \max_I P(O, I|\lambda)$$
$$\delta_t(q) = \max_{i_1, i_2, \ldots i_{t-1}} P(O_1, O_2, \ldots O_t, i_1, i_2, \ldots i_{t-1}, i_t = q|\lambda)$$

Initialisierung

$$\delta_1(q) = \pi_q b_q(O_1)$$

Rekursion

$$\delta_{t+1}(g) = \max_q \{\delta_t(q) a_q g\} \, b_g(O_{t+1})$$

Abschluss

$$P^*(O|\lambda) = \max_q \delta_T(q)$$

3.2 HMM - Beurteilung

Die inneren Zustände haben eine bestimmte Bedeutung und sind relevant für Analysen, daher sollte der Vorgang aufgedeckt werden. Prinzipiell sind alle Zustandsfolgen möglich, somit betrachtet man nur die Zustandsfolge I^* mit der höchsten Wahrscheinlichkeit.

$$I^* = \arg\max_I P(I|O,\lambda)$$

$$mittels\ Bayes - Regel:$$
$$P(I|O,\lambda) = \frac{P(O,I|\lambda)}{P(O|\lambda)}$$

$$für\ Maximierung\ von\ I\ ist\ P(O|\lambda)\ unwichtig:$$
$$I^* = \arg\max_I P(I|O,\lambda) = \arg\max_I P(O,I|\lambda)$$

Das Ergebnis dieser Berechnung liefert die optimale Produktionswahrscheinlichkeit aus dem letzten Kapitel. Daher ist es sinnvoll den Algorithmus für die optimale Produktionswahrscheinlichkeit zu erweitern. Eine effiziente Implementierung davon wird Viterbi Algorithmus genannt. Dabei werden nach der Berechnung der optimalen Produktionswahrscheinlichkeit die inneren Zustände ermittelt.

Initialisierung

$$I_T^* = \arg\max_g \delta_T(g)$$

Rekursion - Rückverfolgung des optimalen Pfades

$$I_t^* = \psi_{t+1}(I_{t+1}^*) \text{ mit } \psi_{t+1}(g) = \arg\max_q \{\delta_t(q)a_{qg}\}$$
... für alle t: T-1, ... 1

3.3 HMM - Training

Da die Parameter eines Markov Modells nur schwer direkt zu ermitteln sind versucht man die Parameter zu trainieren. Dabei werden die Parameter so geändert damit $P(\ldots|\hat{\lambda})$ mit neuem Modell $\hat{\lambda}$ das alte übertrifft. Dabei gibt es verschiedene Ansätze für die Bewertung der Modellierungsgüte (z.B. $P(O|\lambda), P(O, I^*|\lambda), \ldots$).
Das Training erfolgt durch Beobachtung der Erzeugung von Zeichenfolgen. Weiters werden die Übergangs- und Emissionswahrscheinlichkeiten durch relative Häufigkeit der entsprechenden Ereignisse ersetzt.

$$\hat{a}_{qg} = \frac{erwartete\ Anzahl\ der\ Übergänge\ von\ Zustand\ q\ nach\ g}{erwartete\ Anzahl\ der\ Übergänge\ von\ Zustand\ q\ aus}$$

$$\hat{b}_q(O_k) = \frac{erwartete\ Anzahl\ der\ Emissionen\ von\ O_k\ in\ Zustand\ q}{erwartete\ Gesamtanzahl\ der\ Emissionen\ in\ Zustand\ q}$$

Um die Anzahl zu ermitteln benötigt man die Wahrscheinlichkeit mit der ein Zustand, zu einem bestimmten Zeitpunkt t, eingenommen wird. Bei der Bewertung mit der optimalen Produktionswahrscheinlichkeit können diese Wahrscheinlichkeiten direkt angegeben werden.

$$P^*(I_t = q|O, \lambda) = \left\{ \begin{array}{l} 1\ falls\ I_t^* = q \\ 0\ sonst \end{array} \right\}$$

3.3.1 Forward-Backward Algorithmus

Um die Wahrscheinlichkeit eines Zustandes zu ermitteln welches die Gesamtproduktionswahrscheinlichkeit als Modellierungsgüte verwendet kann der Forward Algorithmus mit einer Rückwärtsvariable β erweitert werden. Diese gibt an mit welcher Wahrscheinlichkeit bei gegebenen Modell mit Parameter λ vom Zustand g aus die Zeichenfolge $O_{t+1}, O_{t+2}, \ldots O_T$ erzeugt wird.

$\alpha_t(q) = P(O_1, O_2, \ldots O_t, I_t = q|\lambda)$
$\beta_t(q) = P(O_{t+1}, O_{t+2}, \ldots O_T|I_t = q, \lambda)$
$\gamma_t(q) = P(I_t = q|O, \lambda) = \frac{\alpha_t(q)\beta_t(q)}{P(O|\lambda)}$
... Wahrscheinlichkeit dass zum Zeitpunkt t im Zustand q

Initialisierung

$\alpha_1(q) = \pi_q b_q(O_1)$
$\beta_T(i) = 1$

Rekursion

für $t = 1 \dots T-1$
$\alpha_{t+1}(g) = \sum_q \{\alpha_t(q) a_{qg}\} b_g(O_{t+1})$

für $t = T-1 \dots 1$
$\beta_t(q) = \sum_g a_{qg} b_g(Ot+q) \beta_{t+1}(g)$

Abschluss

$$P(O|\lambda) = \sum_{q=1}^{N} \alpha_T(q)$$
$$P(O|\lambda) = \sum_{q=1}^{N} \pi_q b_q(O_1) \beta_1(q)$$

3.3.2 Baum-Welch Algorithmus

Der Baum-Welch Algorithmus ist ein Optimierungsalgorithmus für die Parameter eines Modells. Optimierungskriterium ist die Gesamtproduktionswahrscheinlichkeit $P(O|\lambda)$, dadurch kann der Forward-Backward Algorithmus für die Berechnung herangezogen werden. Ziel ist es dass das optimierte Modell eine höhere oder gleiche Wahrscheinlichkeit hat die Zeichenfolge zu erzeugen wie das alte Modell $P(O|\hat{\lambda}) \geq P(O|\lambda)$.

$\gamma_t(q) = P(I_t = q|O, \lambda) = \frac{\alpha_t(q)\beta_t(q)}{P(O|\lambda)}$
$\dots Wahrscheinlichkeit,\ dass\ zum\ Zeitpunkt\ t\ der\ Zustand\ q\ eingetreten\ ist$

$\gamma_t(q,g) = P(I_t = q, I_{t+1} = g|O, \lambda) = \frac{P(I_t=q, I_{t+1}=g, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(q) a_{qg} b_g(O_{t+1}) \beta_{t+1}(g)}{P(O|\lambda)}$
$\dots Wahrscheinlichkeit\ eines\ \ddot{U}bergangs\ vom\ Zustand\ q\ in\ den\ Zustand\ g$

$$\hat{a}_{qg} = \frac{\sum_{t=1}^{T-1} \gamma_t(q,g)}{\sum_{t=1}^{T-1} \gamma_t(q)}$$

$$\hat{\pi}_q = \gamma_1(q)$$

$$\hat{b}_g(O_k) = \frac{\sum_{t=1}^{T} P(I_t=g, O_t=O_k|O,\lambda)}{\sum_{t=1}^{T} P(I_t=g|O,\lambda)} = \frac{\sum_{t:O_t=O_k} P(I_t=g|O,\lambda)}{\sum_{t=1}^{T} P(I_t=g|O,\lambda)} = \frac{\sum_{t:O_t=O_k} \gamma_t(g)}{\sum_{t=1}^{T} \gamma_t(g)}$$

3.3.3 Mischverteilungen

Bei der Emissionsmodellierung mittels Mischverteilungen müssen die Normalverteilungsdichten und die Mischungsgewicht optimiert werden.

$$\xi_t(g,k) = P(I_t = g, M_t = k|O,\lambda) = \frac{\sum_{q=1}^{N} \alpha_{t-1}(q) a_{qg} c_{gk} g_{gk}(O_t) \beta_t(g)}{P(O|\lambda)}$$

$\ldots Wahrscheinlichkeit\ dass\ zum\ Zeitpunkt\ t\ im\ Zustand\ g\ die$
$k-te\ Mischverteilungskomponente\ zur\ Erzeugung\ von\ O_t\ verwendet\ wird$

$$\hat{c}_{gk} = \frac{\sum_{t=1}^{T} P(I_t{=}g, M_t{=}k|O,\lambda)}{\sum_{t=1}^{T} P(I_t{=}g|O,\lambda)} = \frac{\sum_{t=1}^{T} \xi_t(g,k)}{\sum_{t=1}^{T} \gamma_t(g)}$$

$$\hat{\mu}_{gk} = \frac{\sum_{t=1}^{T} \xi_t(g,k) x_t}{\sum_{t=1}^{T} \xi_t(g,k)}$$

$$\hat{K}_{gk} = \frac{\sum_{t=1}^{T} \xi_t(g,k)(x_t - \hat{\mu}_{gk})(x_t - \hat{\mu}_{gk})^T}{\sum_{t=1}^{T} \xi_t(g,k)}$$

3.3.4 Viterbi Training

Mit dem Viterbi Training werden ebenfalls Parameter eines Modells optimiert, doch im Gegenzug zum Baum-Welch Algorithmus wird hier als Optimierungskriterium die optimale Produktionswahrscheinlichkeit $P^*(O|\lambda)$ verwendet. Mischverteilungen lassen sich mit dem Viterbi Training nur sehr schwer schätzen.

$$x_t(q) = \left\{ \begin{array}{l} 1\ falls\ I_t^* = q \\ 0\ sonst \end{array} \right\}$$
mit $I^* = \arg\max_I P(I, O|\lambda)$

Initialisierung

Startmodell $\lambda(\pi, A, B)$ wählen

Segmentierung

mittels Viterbi Algorithmus optimale Zustandsfolge I^* zur Erzeugung der Zeichenfolge O berechnen

Optimierung

$\hat{\pi}$ können nicht ermittelt werden

$$\hat{a}_{qg} = \frac{\sum_{t=1}^{T} x_t(q)x_{t+1}(g)}{\sum_{t=1}^{T} x_t(q)}$$

$$\hat{b}_g(O_k) = \frac{\sum_{t:O_t=O_k} x_t(g)}{\sum_{t=1}^{T} x_t(g)}$$

3.3.5 Segmental k-Means

Dieser Ansatz ist ähnlich dem Viterbi Training und dient für die Schätzung von Mischverteilungsparametern. Das Gütemaß ist ebenfalls die optimale Produktionswahrscheinlichkeit.

Initialisierung

Startmodell $\lambda(\pi, A, B)$ wählen

Segmentierung

mittels Viterbi Algorithmus optimale Zustandsfolge I^* zur Erzeugung der Zeichenfolge O berechnen
zugleich $\hat{a}_{qg}$ berechnen (gleich dem Viterbi Training)

Neuschätzung

für alle Zustände $g : 0 \dots N$

Clusteranalyse:

aus Teilstichprobe $\vec{X}(g)$ berechne Vektorquantisierungscodebuch $Y = \{y_1, y_2, \dots y_M\}$ und zugehörige Partitionen $\{R_1, R_2, \dots R_M\}$ mit Hilfe k-means Algorithmus

Berechnung der Modellparameter:

$\hat{c}_{gk} = \frac{|R_k|}{|\vec{X}(g)|}$

$\hat{\mu}_{gk} = y_k$

$\hat{K}_{gk} = \sum_{x \in R_k} xx^T - \hat{\mu}_{gk}\hat{\mu}_{gk}^T$

Terminierung

wenn Gütemaß nicht verbessert wurde

Kapitel 4

Spracherkennung - HMM Anwendung

4.1 Grundaufbau

Die automatische computergestützte Spracherkennung ist ein gut erforschtes Themengebiet und wird mit vielen unterschiedlichen Ansätzen realisiert. Der Grundaufbau bei den unterschiedlichen Systemen ist dennoch ziemlich identisch. Zu Beginn wird das Sprachsignal analysiert, wobei Merkmale extrahiert werden, welche das Signal optimal beschreiben. Im nächsten Schritt wird eine Subworterkennung durchgeführt, um abschließend die Worterkennung durchzuführen.

4.1.1 Analyse

Der Analyseschritt beginnt mit dem Digitalisieren des gesprochenen Wortes. Die Schallwellen des Sprechers werden mittels Mikrofon in Spannungspegel umgewandelt, welche mit einem analog-digital Konverter in Computerverständliche Daten konvertiert werden. Die Daten liegen ab diesen Zeitpunkt in einem zeit- und wertdiskreten Format vor. Um die Daten auch Vergleichen zu können müssen passende Merkmale berechnet werden. Diese Merkmale werden aus dem Frequenz- und Zeitbereich extrahiert, wobei es viele verschiedene Ansätze gibt.

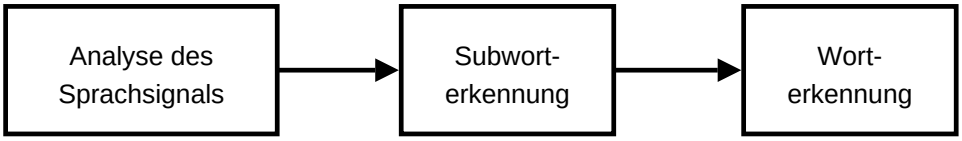


Abbildung 4.1: Grundaufbau einer Spracherkennung

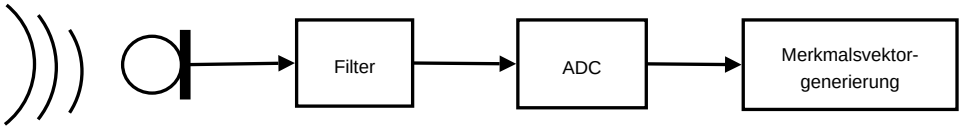


Abbildung 4.2: Sprachanalyse

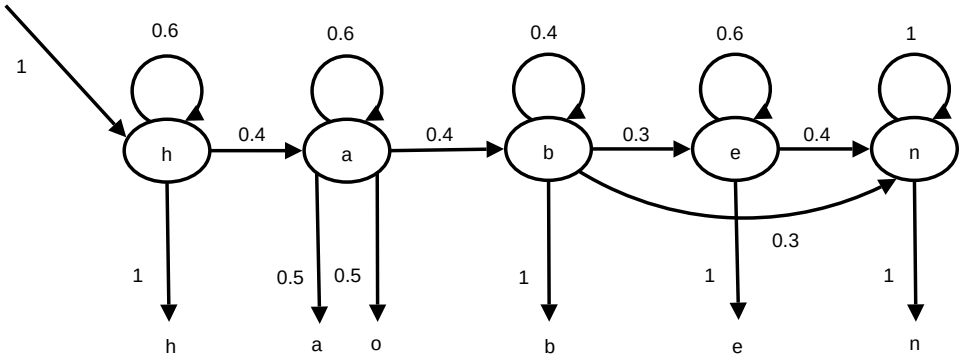


Abbildung 4.3: Worterkennung

4.1.2 Subworterkennung

Für Sprachanalysen wird die Sprache in kleinste bedeutungsunterscheidende Teilbereichen unterteilt. Um von den Merkmalsvektoren auf die Subwörter schließen zu können, werden Codebücher verwendet. Für die Erstellung eines Codebuches werden Vektorquantisierer eingesetzt. Das erstellte Codebuch ist also für die Zuordnung von Merkmalsvektoren zu den kleinsten Teilbereichen zuständig. Ein Beispiel für die kleinsten Teilbereiche könnte das Alphabet sein (a, b, c, ...).

4.1.3 Worterkennung

Die Worterkennung erfordert statistische Analysen der aufeinander folgenden Subwörter, denn die Wörter können unterschiedlich ausgesprochen werden. Um die Erkennung zu verbessern ist es üblich die statistischen Modelle mit Beispieldaten zu trainieren. Bei diesen Modellen handelt sich oft um die 'Hidden Markov Models', die zuvor im theoretischen Teil betrachtet wurden. Im speziellen werden left-to-right HMMs verwendet, wobei ein Zustand einem Subwort entspricht.

Ein Sprachsteuerungssystem verwendet bestimmte definierte Key-Wörter um gewünschte Befehle durchzuführen. Für jedes zu erkennende Wort wird ein HMM angelernt und bei der Sprachanalyse parallel ausgewertet. Das Modell mit der höchsten Produktionswahrscheinlichkeit gibt Auskunft über

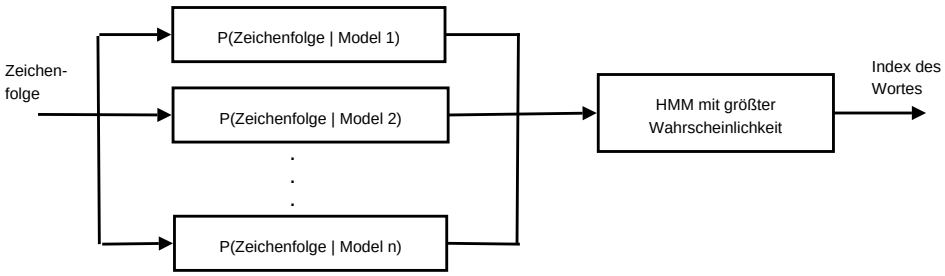


Abbildung 4.4: Bewertung

das gesprochene Wort.

4.2 Merkmalsvektoren

4.2.1 MFCC

Die Abkürzung MFCC steht für mel-frequency cepstral coefficients - melskalierte Cepstralkoeffizienten. Diese Koeffizienten werden häufig für die Merkmalsvektoren verwendet um das Sprachsignal interpretieren zu können. Es ist notwendig dass das Sprachsignal mit den gewählten Merkmalen hinreichend repräsentiert werden kann. Das Sprachsignal wird durch die Stimmbänder (periodisches Ausgangssignal) und durch die Veränderung des Ausgabekanales wie Mund und Zunge (Filter) erzeugt. Für die Analyse des Signals ist der Filter von hoher Relevanz, somit ist es sinnvoll das Ausgangssignal nicht weiter zu betrachten. Mit den MFCC wird genau dieser Filter vom Ausgangssignal getrennt.

- Sprachsignal in Zeitabschnitte unterteilen, welche sich überlappen
- diskrete Fourier-Transformation (DFT)
- Bildung des Betragsspektrums (Phase unwichtig)
- Logarithmisierung des Betragsspektrums (angepasst auf menschliche Ohr)
- mel-Skalierung: Zusammenfassen von Frequenzbändern durch Dreiecksfilter $f_{mel} \approx 2595 \cdot \log\left[1 + \frac{f_{Hz}}{700}\right]$
- Dekorrelation durch cosinus-Transformation

Die Zeitabschnitte, in welches das Signal aufgeteilt wird, sollte klein genug sein um gute zeitliche Auflösung zu gewährleisten und auch groß genug aufgrund der spektralen Auflösung. Durch die mel-Skalierung werden die unteren Frequenzbänder höher gewichtet, da diese eine höhere Bedeutung für die

Sprachanalyse haben. Für die Merkmale werden die mel-Cepstralkoeffizienten und deren erste und zweite Ableitung herangezogen.

4.2.2 lineare Vorhersage

Die lineare Vorhersage stellt eine einfache Methode dar mit der Merkmale des Sprachsignals gewonnen werden können. Die Grundidee besteht darin dass mit vorherigen Werten und sogenannter Vorhersagekoeffizienten der aktuelle Wert geschätzt wird. Die Koeffizienten bestimmen die Impulsantwort des linearen Systems (Mund, Zunge, ...) und somit den ausgegebenen Laut. Um die gewünschten Vorhersagekoeffizienten zu erhalten wird der Vorhersagefehler minimiert, basierend auf den Eingangsdaten.

$$\hat{f}_n = \sum_{\mu=1}^{m} a_\mu f_{n-\mu}$$
$$f_n \ldots aktuelle\ Wert$$
$$f_{n-1} \ldots vorherige\ Wert$$
$$a_\mu \ldots Vorhersagekoeffizienten$$

$$\epsilon = \sum_{n=n_0}^{n_1} (f_n - \hat{f}_n)^2$$

$$\frac{d\epsilon}{da_v} = 0 = \sum (f_n + \sum_{\mu} a_\mu f_{n-\mu}) 2 f_{n-v}$$

$$\sum_{\mu} a_\mu \sum_{n} f_{n-\mu} f_{n-v} = -\sum_{n} f_n f_{n-v}$$
$$v = 1 \ldots m$$

Man erhält m Gleichungen für die Berechnung der Koeffizienten die als Merkmale dienen. Des weiteren wird auch der Vorhersagefehler als Merkmal verwendet.

4.2.3 Lautheit

Um die Lautheit mitprotokollieren zu können wird der Schallpegel betrachtet. Mit Anlehnung an den Schallpegel kann die Lautheit im selben Schritt wie die Cepstrum-Koeffizienten berechnet werden.

$$L = 20 \log \left[\frac{p_s}{p_0}\right] = 10 \log \left[\frac{I_s}{I_0}\right]$$
$$p_s \ldots messbare\ Schalldruck$$
$$p_0 = 2 \cdot 10^{-5} Pa \ldots festgelegte\ Bezugsgröße$$
$$I_s \ldots Intensität$$
$$I_0 = 10^{-12} N m^{-2}$$

$$L_T = \sum_{j} 10 \log c_{Tj}$$
$$c_{Tj} \ldots mel\ transformierte\ Koeffizienten\ des\ Leistungsspektrums$$

4.2.4 Normierung

Damit externe Einflüsse, die durch das Mikrofon oder den Raum hervorgerufen werden, reduziert werden wird das Signal normiert. Die Normierung erfolgt beispielsweise durch cepstralen Mittelwertabzug.

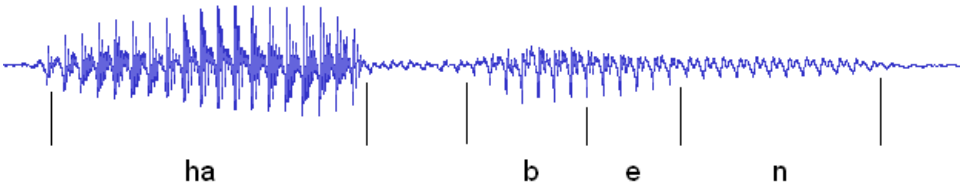


Abbildung 4.5: aufgenommene Sprachsignal

4.3 Simulation

4.3.1 MFCC

Zur Generierung der melskalierten Cepstralkoeffizienten wird eine Matlab Bibliothek von [1] verwendet. Diese Bibliothek unterstützt verschiedene Verfahren um Merkmale zu generieren. Das zu analysierende Signal wird mit einem Mikrophon und dem frei verfügbaren Audacity aufgenommen und als WAV-Datei abgespeichert. Bereits mit dieser Aufnahme ist ersichtlich welche Zeitabschnitte welche Laute hervorrufen. Für die die Veranschaulichung der Koeffizienten wird ein Matlab-Script benötigt, welches die Datei einliest und daraus die MFCCs berechnet. Da die MFCCs eine bessere Darstellung des Sprachsignals erlauben, als orginale Daten aus dem Zeit- bzw. Frequenzbereich, werden diese für weitere Analyse- / Verarbeitungszwecke verwendet.

```
1  % MFCC Analyse
2
3  % WAV Datei laden
4  file = 'haben_1';
5  [d_stereo, fsample, nbits] = wavread([file,'.wav']);
6  d_mono = d_stereo(:,1);
7
8  % original WAV anhören
9  soundsc(d_mono, fsample);
10
11 % MFCC Berechnung
12 [cepstra,aspectrum,pspectrum] = melfcc(d_mono, fsample);
13
14 % inverse MFCC Berechnung
15 [x,aspc,spec] = invmelfcc(cepstra, fsample);
16
17 % WAV anhören
18 soundsc(x, fsample);
19
20 % WAV Datei schreiben
21 wavwrite(x, fsample, [file,'_after.wav']);
```

```
22
23 subplot(511)
24 time = [1:length(d_mono)]./fsample;
25 plot(time, d_mono(1:40960));
26 xlabel('time');
27 ylabel('amplitude');
28
29 subplot(512)
30 time = [1:length(x)]./fsample;
31 plot(time, x(1:41738,1));
32 xlabel('time');
33 ylabel('amplitude');
34
35 subplot(513)
36 imagesc(aspectrum(1:40,1:91));
37 axis off
38 xlabel('time');
39 ylabel('frequency');
40
41 subplot(514)
42 imagesc(aspc(1:40,1:91));
43 axis off
44 xlabel('time');
45 ylabel('frequency');
46
47 subplot(515)
48 imagesc(cepstra(1:13,1:91));
49 axis off
50 xlabel('time');
51 ylabel('frequency');
```

.

Im ersten Plot ist das originale Sprachsignal im Zeitbereich dargestellt, das zweite ist das Signal welches durch die inverse MFCC generiert wurde. Die nächsten beiden Plots stellen die jeweiligen Spektren dar und der unterste der ersten fünf stellt die Cepstralkoeffizienten dar. Das zurückgewonnene Signal sieht dem originalen Sprachsignal sehr ähnlich, im Zeit- sowie im Frequenzbereich.
Die nächsten Plots sind ebenfalls mit dem Script erzeugt worden, aber mit einem wiederholt aufgenommenen Signal. Hier sieht man gut dass trotz der wiederholten Aufnahme die Wörter fast gleich abgebildet werden.

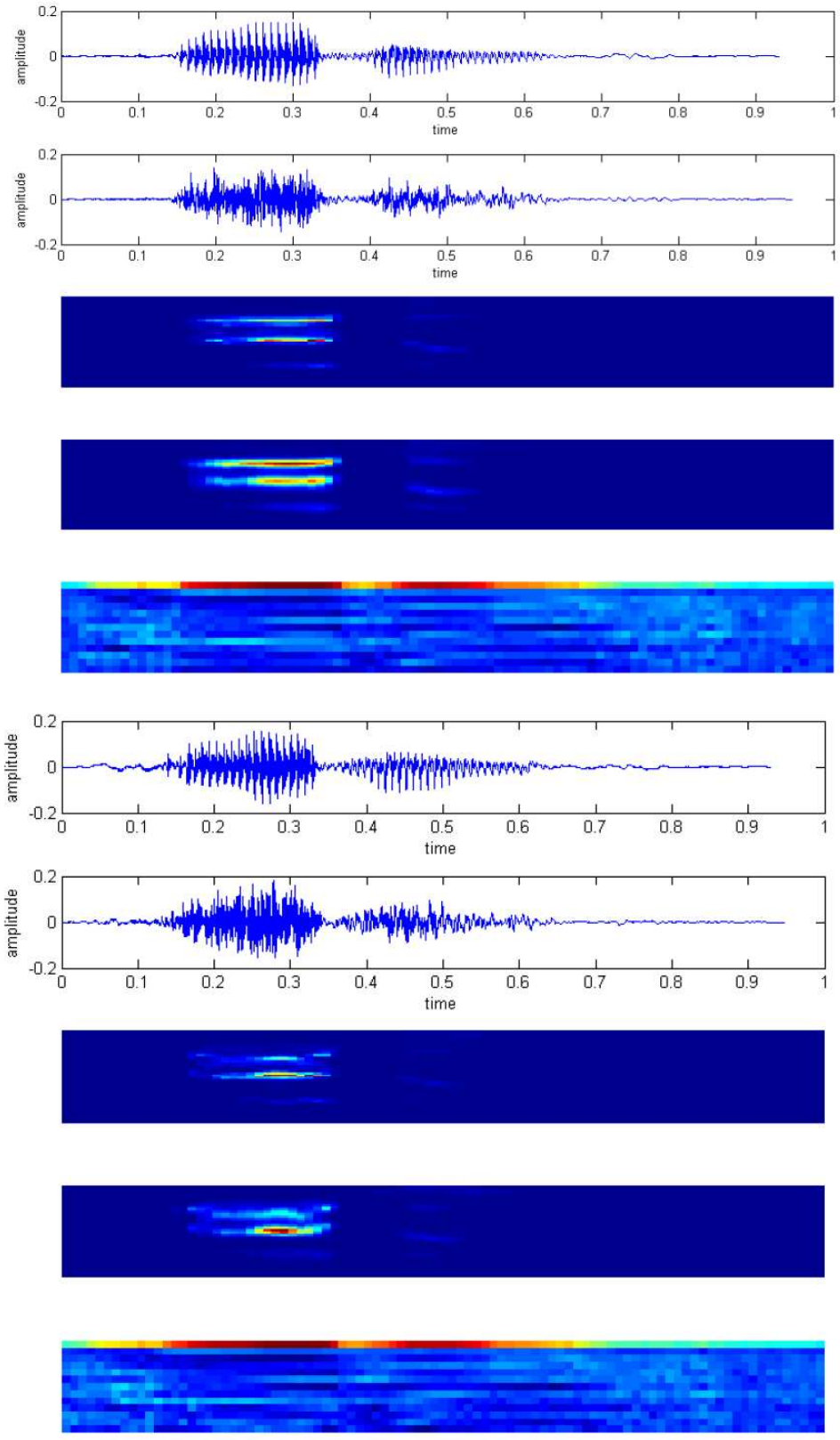


Abbildung 4.6: Matlab Simulationsergebnisse

4.3.2 Codebuchgenerierung

Das HMM benötigt als Eingangsfolge Indizes des zugehörigen Subwortes. Die Umsetzung von den Merkmalsvektoren zu den Indizes übernimmt ein Quantisierer aufgrund eines Codebuches. Damit die unterschiedlich ausgesprochenen gleichen Subwörter durch das Codebuch optimal repräsentiert werden, muss es zuvor trainiert werden. Nach dem Training gibt der Quantisierer, zu jedem Merkmalsvektor, den richtigen Index aus. Damit das funktioniert wird der Abstand des aktuellen Merkmalsvektors zu den Codebucheinträgen berechnet und das Minimum gebildet (Index des am nähesten liegenden Codebucheintrages).
Zur Simulation mit dem Matlab Script von [2] wird der Merkmalsvektor aus zwei Werten gebildet. Für alle 6 Buchstaben (h,a,o,b,e,n) werden Grundwerte definiert, die bei den Trainingsvektoren leicht abgeändert werden. Das initiale Codebuch besteht aus den ersten 6 Trainingsvektoren und wird mittels LBG-Algorithmus trainiert.

```
% Simulation eines Codebuchgenerators

% Initialisierung
% Trainingsdaten
T = [[100, 50];
     [10, 10];
     [40, 50];
     [200, 100];
     [40, 150];
     [50, 100];
     [103, 53];
     [13, 10];
     [43, 50];
...
     [58, 104]];
% initiale Codebuch
C = [[100, 50];
     [10, 10];
     [40, 50];
     [200, 100];
     [40, 150];
     [50, 100]];
% initiale Quantisierungsfehler
qestart = 1000000;
% minimal erreichbare Verbesserung
Qmin = 0.0001;
% maximal durchgeführte Iterationen
maxit = 50;

% Codebuch-Training
% C ... generierte Codebuch
% qe ... Quantisierungsfehler des generierten Codebuches
% nit ... Anzahl an durchgeführten Iterationen
```

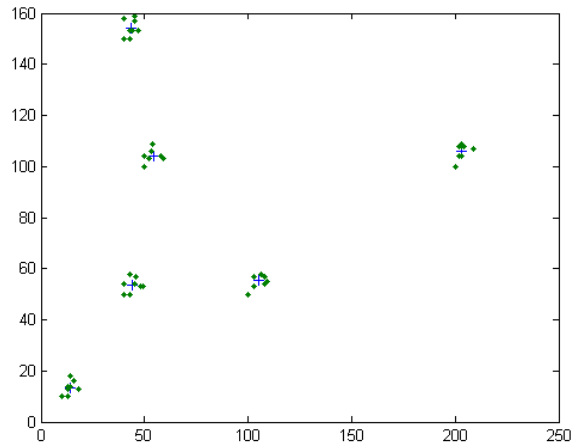


Abbildung 4.7: Codebuch und Trainingsdaten

```
34 % Q ... Verbesserungsgrad
35 [C, qe, nit, Q] = lbg(T', C', qestart, Qmin, maxit);
36 C = C';
37
38 % Indizes der Trainingsdaten berechnen
39 % Anzahl Trainingsvektoren
40 T_len = size(T, 1);
41 C_len = size(C, 1);
42
43 % für jeden Trainingssatz
44 d = [];
45 T_cbindex = [];
46 for (i = 1:T_len)
47     for (j = 1:C_len)                    % Abstand berechnen
48         d(j) = (T(i,1) - C(j,1))^2 + (T(i,2) - C(j,2))^2;
49     end
50     [tmp, T_cbindex(i)] = min(d);        % min. Abstand gewinnt
51 end
52
53 % graphische Ausgabe
54 plot(C(:,1),C(:,2),'+',T(:,1),T(:,2),'.')
```

.

Als Ergebnis der Simulation erhält man ein optimal angepasstes Codebuch. Die '+' Markierungen im Plot stellen die 6 Codebucheinträge dar und die '.' Markierungen die Trainingsdaten.

4.3.3 Worterkennung

Für die Worterkennung wird eine HMM Bibliothek von [5] verwendet. Die Bibliothek erlaubt es Modelle zu erstellen, trainieren und den optimalen Pfad zu berechnen. Zur Simulation wird das Wort 'haben' verwendet, das als HMM abgebildet wird. Um Aussprachefehler zu ignorieren werden auch die Wörter 'hoben', 'hobn' und 'habn' erlaubt. Während der Initialisierung müssen die Anfangswahrscheinlichkeiten, die Übergangswahrscheinlichkeiten und die Emissionswahrscheinlichkeiten angegeben werden. Diese können mit halbwegs brauchbaren oder mit zufälligen Werten definiert werden. Für das Training des Modells werden weiters auch Trainingsdaten benötigt, diese setzen sich aus mehreren Zeichenfolgen zusammen.

```
% O = [1, 2, 3, 4, 5, 6]
% h a o b e n
%
% Q = [1, 2, 3, 4, 5]
% h a b e n

% Startwerte
O = 6;
Q = 5;
% - Anfangswahrscheinlichkeiten
prior00 = [1,0,0,0,0];
prior01 = normalise(rand(Q,1));
% - Anfangsübergangswahrscheinlichkeiten
transmat00 = [[0.6, 0.4, 0 , 0 , 0 ];
              [0  , 0.6, 0.4, 0 , 0 ];
              [0  , 0  , 0.4, 0.3, 0.3];
              [0  , 0  , 0  , 0.6, 0.4];
              [0  , 0  , 0  , 0  , 1  ]];
transmat01 = mk_stochastic(rand(Q,Q));
% - Anfangsobservationswahrscheinlichkeiten
obsmat00 = [[1, 0 , 0 , 0, 0, 0];
            [0, 0.5, 0.5, 0, 0, 0];
            [0, 0  , 0  , 1, 0, 0];
            [0, 0  , 0  , 0, 1, 0];
            [0, 0  , 0  , 0, 0, 1]];
obsmat01 = mk_stochastic(rand(Q,O));

% - Trainingsdaten
traindata00 = [[1,1,1,2,2,2,4,4,4,5,5,5,6,6,6];
               [1,1,1,3,3,4,4,5,5,5,5,6,6,6,6];
               [1,1,1,1,2,2,2,2,2,4,4,4,4,6,6];
               [1,1,2,2,2,2,4,4,4,4,4,4,4,6,6]];
traindata01 = [[1,1,1,2,2,2,4,4,4,5,5,5,6,6,6];
               [1,1,1,3,3,4,4,5,5,5,5,6,6,6,6];
               [1,1,1,1,2,2,2,2,2,4,4,4,4,6,6];
               [1,1,2,2,2,2,4,4,4,4,4,4,4,6,6];
               [1,1,1,2,2,2,4,4,4,5,5,5,6,6,6];
```

```
38              [1,1,1,3,3,4,4,5,5,5,5,6,6,6,6];
39              [1,1,1,1,2,2,2,2,2,4,4,4,4,6,6];
40              [1,1,2,2,2,2,4,4,4,4,4,4,4,6,6]];
41
42 % Parameterschätzung mittels EM-Algorithmus
43 disp('gute Startwerte:')
44 [LL, prior10, transmat10, obsmat10] = dhmm_em(traindata00, prior00,
       transmat00, obsmat00, 'max_iter', 5);
45 disp('zufällige Startwerte:')
46 [LL, prior11, transmat11, obsmat11] = dhmm_em(traindata01, prior01,
       transmat01, obsmat01, 'max_iter', 100);
47
48 % Berechnung des inneren Zustandes - Viterbi
49     % optimierte Anfangswahrscheinlichkeiten
50     data = [1, 1, 2, 2, 4, 4, 5, 5, 6, 6]
51     B0 = multinomial_prob(data, obsmat10);
52     path0 = viterbi_path(prior10, transmat10, B0)
53
54     % zufällige Anfangswahrscheinlichkeiten
55     data = [1, 1, 2, 2, 4, 4, 5, 5, 6, 6]
56     B0 = multinomial_prob(data, obsmat11);
57     path0 = viterbi_path(prior11, transmat11, B0)
58
59 % Berechnung der Produktionswahrscheinlichkeit
60     % optimierte Anfangswahrscheinlichkeiten
61     data = [1, 1, 2, 2, 4, 4, 5, 5, 6, 6]
62     P = forwAlg(data, prior10, transmat10, obsmat10, O, Q)
63
64     data = [2, 2, 1, 1, 6, 6, 4, 5, 5, 5]
65     P = forwAlg(data, prior10, transmat10, obsmat10, O, Q)
66
67     % zufällige Anfangswahrscheinlichkeiten
68     data = [1, 1, 2, 2, 4, 4, 5, 5, 6, 6]
69     P = forwAlg(data, prior11, transmat11, obsmat11, O, Q)
70
71     data = [2, 2, 1, 1, 6, 6, 4, 5, 5, 5]
72     P = forwAlg(data, prior11, transmat11, obsmat11, O, Q)
```

.

```
1 function P = forwAlg(data, prior, transmat, obsmat, O, Q)
2 % Forward-Algorithmus für Berechnung der Produktions-Wahrscheinlichkeit
3 %
4 % data ..... beobachtete Zeichenfolge
5 % prior ..... Anfangswahrscheinlichkeiten
6 % transmat ... Übergangswahrscheinlichkeiten
7 % obsmat ..... Observationswahrscheinlichkeiten
8 % O ..... Anzahl der Ausgabezeichen
9 % Q ..... Anzahl der Zustände
10
11 L = [[]];
12 for q = 1:Q,
13     L(1, q) = prior(q) * obsmat(q, data(1));
```

```
end
for index = 2:length(data),
    for g = 1:Q,
        tmp_sum = 0;
        for q = 1:Q,
            tmp_sum = tmp_sum + L(index-1, q) * transmat(q, g);
        end
        L(index, g) = tmp_sum * obsmat(g, data(index));
    end
end
P = 0;
for q = 1:Q,
    P = P + L(length(data), q);
end
```

.

Wird das Matlab-Script ausgeführt so ist der Verbesserungswert, der optimale Pfad (innere Zustände) und die Produktionswahrscheinlichkeit ersichtlich. Weiters stellt man fest das die Wahrscheinlichkeiten der einzelnen Matrizen verändert bzw. mit Trainingsdaten verbessert wurden. Vergleicht man die Ergebnisse beider Startwerte, somit wird ersichtlich wie wichtig eine gute Vorinitialisierung ist. Bei einer guten Initialisierung verbessern sich die Ergebnisse ab dem 3 Durchlauf nicht mehr, wurden aber zufällige Startwerte verwendet benötigt der EM Algorithmus bereits 37 Durchläufe. Weiters sind die Ergebnisse stark unterschiedlich. Kann man im Voraus Übergänge zwischen zwei Zuständen ausschließen, so sollten diese mit 0 initialisiert werden. Um von einem gewünschten Zustand zu starten, sollten auch die Startwahrscheinlichkeiten manuell eingetragen werden.

Matlab-Ausgaben:

```
>> haben_hmm
gute Startwerte:
iteration 1, loglik = -46.427337
iteration 2, loglik = -37.713411
iteration 3, loglik = -37.713411
zufällige Startwerte:
iteration 1, loglik = -224.015001
iteration 2, loglik = -196.437154
iteration 3, loglik = -189.150900
iteration 4, loglik = -179.850824
iteration 5, loglik = -167.169847
iteration 6, loglik = -147.747140
iteration 7, loglik = -135.350885
iteration 8, loglik = -132.431172
iteration 9, loglik = -130.674447
iteration 10, loglik = -128.718028
iteration 11, loglik = -126.578402
iteration 12, loglik = -124.288369
iteration 13, loglik = -121.343634
iteration 14, loglik = -113.834639
```

```
iteration 15, loglik = -93.527860
iteration 16, loglik = -82.863778
iteration 17, loglik = -82.301783
iteration 18, loglik = -81.792004
iteration 19, loglik = -80.951681
iteration 20, loglik = -79.732686
iteration 21, loglik = -78.655264
iteration 22, loglik = -78.230884
iteration 23, loglik = -78.133896
iteration 24, loglik = -78.107878
iteration 25, loglik = -78.099492
iteration 26, loglik = -78.096711

data =

     1     1     2     2     4     4     5     5     6     6


path0 =

     1     1     2     2     3     3     4     4     5     5


data =

     1     1     2     2     4     4     5     5     6     6


path0 =

     1     1     2     2     5     5     4     4     4     4


data =

     1     1     2     2     4     4     5     5     6     6


P =

  6.3749e-004


data =

     2     2     1     1     6     6     4     5     5     5


P =

     0


data =
```

```
75
76     1     1     2     2     4     4     5     5     6     6
77
78
79 P =
80
81   3.3094e-004
82
83
84 data =
85
86     2     2     1     1     6     6     4     5     5     5
87
88
89 P =
90
91   1.9025e-071
92
93 >>
```

.

Bei den Produktionswahrscheinlichkeiten verschiedener Zeichenfolgen, mit unterschiedlich gut vorinitialisierten Hidden Markov Modellen, kann man den unterschied zwischen guter und schlechter Repräsentanz erkennen. Da aber die Wahrscheinlichkeiten, je nach Modell und Länge der Zeichenfolge, ziemlich klein werden können, wird oft eine Skalierung durchgeführt. Diese Skalierung kann mittels multiplikativem Faktor bzw. mittels logarithmischer Rechnung erfolgen.

Die Wahrscheinlichkeits-Tabellen stellen oberhalb die initialen Werte und unterhalb die Werte nach dem Training dar. Wurden sie mit passablen Werten vorinitialisiert, so werden diese Werte nur noch den Trainingsdaten angepasst. Bei zufälligen Startwerten kann nach dem Training kein Übergang ausgeschlossen werden und des weiteren ist der HMM Einsprungpunkt auch an anderen Stellen als beim Zustand 'h' möglich.

Emissionswahrscheinlichkeiten

	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	0.5	0.5	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1
	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	0.8571	0.1429	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1

Transmissionswahrscheinlichkeiten

	1	2	3	4	5
1	0.6	0.4	0	0	0
2	0	0.6	0.4	0	0
3	0	0	0.4	0.3	0.3
4	0	0	0	0.6	0.4
5	0	0	0	0	1
	1	2	3	4	5
1	0.6667	0.3333	0	0	0
2	0	0.7143	0.2857	0	0
3	0	0	0.75	0.125	0.125
4	0	0	0	0.7143	0.2857
5	0	0	0	0	1

Abbildung 4.8: Wahrscheinlichkeits Matrizen - durchdachte Startwerte

Emissionswahrscheinlichkeiten

	1	2	3	4	5	6
1	0.1728	0.0859	0.3268	0.0018	0.3272	0.0855
2	0.1088	0.2933	0.316	0.2558	0.0177	0.0084
3	0.2348	0.2269	0.1865	0.0608	0.0417	0.2493
4	0.0949	0.2554	0.1385	0.1186	0.2423	0.1503
5	0.0475	0.0773	0.2339	0.2915	0.232	0.1178

	1	2	3	4	5	6
1	0.9905	0	0.0095	0	0	0
2	4.3505e-023	0.8606	0.1394	2.446e-019	0	0
3	1	6.8895e-111	2.0903e-153	0	0	0
4	0	0	0	7.649e-020	0.3889	0.6111
5	0	1.9769e-024	7.6159e-019	1	7.1327e-019	8.6247e-027

Transmissionswahrscheinlichkeiten

	1	2	3	4	5
1	0.0765	0.2615	0.1208	0.3936	0.1476
2	0.2997	0.2178	0.1481	0.1946	0.1399
3	0.3077	0.2896	0.0943	0.2633	0.0451
4	0.1646	0.1894	0.0641	0.3624	0.2194
5	0.1719	0.2307	0.0741	0.1746	0.3487

	1	2	3	4	5
1	0.3145	0.6855	3.5876e-014	0	1.0612e-059
2	4.072e-095	0.7131	0	0	0.2869
3	0.643	3.6912e-011	0.357	0	0
4	0	0	0	1	1.7038e-280
5	0	0	0	0.25	0.75

Anfangswahrscheinlichkeiten

	1
1	0.1566
2	0.1108
3	0.2957
4	0.2466
5	0.1903

	1
1	1.4068e-040
2	0
3	1
4	0
5	0

Abbildung 4.9: Wahrscheinlichkeits Matrizen - zufällige Startwerte

Anhang A

Inhalt der CD-ROM/DVD

File System: Joliet[1]

Mode: Single-Session (CD-ROM)[2]

A.1 Diplomarbeit

Pfad: /

DaBA.dvi	Bachelorarbeit (DVI-File, ohne Grafiken)
DaBA.pdf	Bachelorarbeit (PDF-File)
DaBA.ps	Bachelorarbeit (PostScript-File)
haben_mfcc.m	MFCC Simulation (Matlab-Script)
haben_lbg.m	LBG Simulation (Matlab-Script)
forwAlg.m	Implementation des Forward-Algorithmus (Matlab-Script)
haben_hmm.m	HMM Simulation (Matlab-Script)

[1] *oder* ISO9660 – bei DVDs entsprechend andere Spezifikationen.
[2] *oder* Multi-Session (CD-ROM XA), DVD etc.

Literaturverzeichnis

[1] Daniel P. W. Ellis: *PLP and RASTA (and MFCC, and inversion) in Matlab*, 2005. online Web Quelle.

[2] Dr.-Ing. Armin Günter: *Signal and Image Processing*, 1999. online Web Quelle.

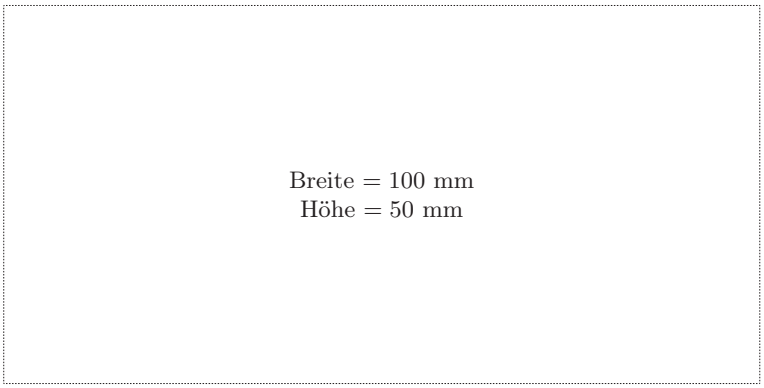
[3] Dr.-Ing. habil. Hans-Georg Brachtendorf. private Mitteilung.

[4] Gernot A. Fink: *Mustererkennung mit Markov-Modellen*. Teubner, 2003.

[5] Kevin Murphy: *Hidden Markov Model (HMM) Toolbox for Matlab*, 1998. online Web Quelle.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —