Der Baum-Welch Algorithmus für Hidden Markov Models, ein Spezialfall des EM-Algorithmus

Holger Wunsch

6. August 2001

Inhaltsverzeichnis

1 Einleitung

Der EM-Algorithmus (Expectation-Maximization Algorithmus) ist ein allgemeines Verfahren zur Bestimmung von Maximum-Likelihood Schätzwerten von probabilistischen Modellen. Die Bezeichnung "Algorithmus" ist eigentlich irreführend, da in der allgemeinen Definition des Verfahrens keine genauen Anweisungen für Rechenschritte gegeben werden, wie es der Begriff des Algorithmus fordert, sondern nur eine allgemeine Beschreibung des Verfahrens und seiner mathematischen Eigenschaften.

Für jedes Anwendungsgebiet muß daher ein EM-Algorithmus "gefunden" werden. Für Hidden Markov Models heißt dieser Algorithmus "Baum-Welch Algorithmus".

Diese Hausarbeit beschreibt zunächst die allgemeinen Ansätze der Maximum-Likelihood Methode und des EM-Algorithmus. Nach einer formalen Beschreibung von Hidden Markov Models wird dann der Baum-Welch Algorithmus vorgestellt, zunächst in seiner klassischen Version und dann seine Herleitung in der Begrifflichkeit des allgemeinen EM-Algorithmus. Die Darstellung der Maximum-Likelihood Methode richtet sich im Wesentlichen nach [3], ebenso wie die allgemeine Beschreibung des EM-Algorithmus. Die Darstellung von Hidden Markov Models ist sehr nah an am Kapitel über HMMs in [1]. Die Präsentation des Baum-Welch Algorithmus in seiner klassischen Form gründet sich auf die Darstellung in [4], die Herleitung aus den allgemeinen EM Formeln ist in dieser Form in [3] beschrieben.

Weiterhin wird eine Java-Implementation des Baum-Welch Algorithmus gezeigt, die im Wesentlichen die Version des Baum-Welch Algorithmus in [4] umsetzt. Dabei wird auf das Beispiel der "Crazy Soft Drink Machine" in [1] zurückgegriffen.

2 Maximum Likelihood Methode

Die Probleme, die mit statistischen Methoden in NLP gelöst werden, haben häufig die folgende Gestalt: Man nimmt an, daß ein bestimmtes linguistisches Phänomen mit einer bestimmten statistischen Verteilung ausreichend gut beschrieben werden kann. Es liegt eine Menge an Daten vor, die als Ergebnisse eines Zufallsexperiments interpretiert werden, nicht bekannt sind jedoch die Parameter der zugrundeliegenden, angenommenen Verteilung. Diese müssen mit geeigneten Methoden geschätzt werden.

Die Maximum Likelihood Methode ist ein solches Verfahren. Sie basiert darauf, daß für eine gegebene Menge beobachteter Daten (eine solche Menge wird **Realisierung** genannt) versucht wird, die unbekannten Parameter der zugrundeliegenden Verteilung so zu bestimmen, daß diese den beobachteten Daten maximale Wahrscheinlichkeiten zuordnet.

2.1 Formale Beschreibung der Maximum Likelihood Methode

Definition 2.1 *Sei X eine Zufallsvariable, und $\mathcal{X} = \{x_1, \ldots, x_n\}$ eine Realisierung. Sei weiterhin $p(x|\theta)$ die Dichtefunktion von X bezüglich einer bestimmten Belegung von Parametern $\theta \in \Theta$. (Θ ist die Menge der möglichen Parameterbelegungen für p).*

Die Funktion

$$\mathcal{L}(\theta|\mathcal{X}) := \prod_{x \in \mathcal{X}} p(x|\theta)$$

*heißt **Likelihood-Funktion** einer Menge von Parametern $\theta \in \Theta$ bei einer gegebenen Realisierung $\mathcal{X}$.*

Informell berechnet die Likelihood-Funktion also die Wahrscheinlichkeit, die einer Realisierung von der zugrundeliegenden Dichtefunktion zugewiesen wird, wenn die Parameter auf den Wert θ gesetzt sind.

Definition 2.2 *Ein Parameterwert $\hat{\theta} = \hat{\theta}(\mathcal{X})$ heißt **Maximum Likelihood***

***Schätzwert**, wenn für alle $\theta \in \Theta$ gilt:*

$$\mathcal{L}(\hat{\theta}|\mathcal{X}) \geq \mathcal{L}(\theta|\mathcal{X})$$

Definition 2.3 *Ein Schätzverfahren $T_n : \mathbb{X} \to \Theta$ das für alle Realisierungen $\mathcal{X} \in \mathbb{X}$, zu denen Maximum-Likelihood-Schätzwerte $\hat{\theta}(\mathcal{X})$ existieren, diese als Schätzwerte $T_n(\mathcal{X})$ liefert, also*

$$T_n(\mathcal{X}) = \hat{\theta}(\mathcal{X})$$

*gilt, heißt **Maximum-Likelihood Schätzer**.*

Ein allgemeiner Ansatz zur Ermittlung eines Maximum-Likelihood Schätzers ist, ein Maximum von $\mathcal{L}$ durch die Berechnung der Nullstellen der Ableitung $\frac{\partial}{\partial\theta}\mathcal{L}(\theta|\mathcal{X})$ zu bestimmen. Die Berechnung der Ableitung kann jedoch in vielen Fällen sehr kompliziert sein. In diesem Fall benötigt man Verfahren, die iterativ den gesuchten Schätzwert $\hat{\theta}$ bestimmen. Ein solches Verfahren ist der EM-Algorithmus.

3 Die allgemeine Form des EM-Algorithmus

Mit dem EM-Algorithmus kann iterativ ein Maximum-Likelihood Schätzwert ermittelt werden, wenn die vorhandenen Daten entweder unvollständig sind, oder wenn nicht alle Parameter der Daten in eine direkte analytische Berechnung des Schätzwertes eingehen können, weil ein Teil der Parameter sich dem Zugriff entzieht.

Ausgangspunkt ist also eine Realisation $\mathcal{X}$ genannt. Die Annahme ist, daß $\mathcal{X}$ von einer bestimmten Wahrscheinlichkeitsverteilung generiert wurde, deren Parameter jedoch unbekannt sind. Das Ziel ist, die Parameter so zu finden, daß die Verteilung die Daten optimal beschreibt.

$\mathcal{X}$ sind also die *unvollständigen Daten*. Wir nehmen aber an, daß eine vollständige Datenmenge existiert, die wir mit $\mathcal{Z}$ bezeichnen, wobei $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$ und $\mathcal{Y}$ eine Zufallsvariable mit der zugeörigen Verteilung $f(y|\mathcal{X}, \theta^{(i-1)})$ ist, die die zugrundeliegenden, unsichtbaren Ereignisse beschreibt. Die Dichtefunktion für $\mathcal{Z}$ ist:

$$\mathrm{p}(z \mid \theta) = \mathrm{p}(x, y \mid \theta)$$

Wir können nun eine neue Likelihood-Funktion für die vollständigen Daten einführen, $\mathcal{L}(\theta|\mathcal{Z}) = \mathcal{L}(\theta|\mathcal{X},\mathcal{Y}) = \mathrm{p}(\mathcal{X},\mathcal{Y}|\theta) = \prod_{y\in\mathcal{Y}} \mathrm{p}(y,\mathcal{X}|\theta)$.

$\mathcal{L}$ ist als Funktion der Zufallsvariablen $\mathcal{Y}$ selbst eine Zufallsvariable, und kann auch als eine von $\mathcal{Y}$ abhängige Funktion bezüglich der zwei Konstanten $\mathcal{X}$ und θ betrachtet werden: $\mathcal{L}(\theta|\mathcal{X},\mathcal{Y}) = \hat{h}_{\mathcal{X},\theta}(\mathcal{Y})$.

Wir betrachten im Folgenden stets die "Log-Likelihood" (den Logarithmus der Likelihood-Funktion) und definieren $h_{\mathcal{X},\theta}(\mathcal{Y}) := \log \hat{h}_{\mathcal{X},\theta}(\mathcal{Y})$.

Der EM-Algorithmus geht in zwei Schritten vor. Im ersten Schritt wird der Erwartungswert der log-Likelihood der vollständigen Daten gegeben die Realisierung und die aktuellen Parameter-Schätzungen berechnet.

$$Q(\theta,\theta^{(i-1)}) = E(h_{\mathcal{X},\theta}(\mathcal{Y}) \mid \mathcal{X},\theta^{(i-1)}) = \int_{y\in\mathcal{Y}} h_{\mathcal{X},\theta}(y) f(y,\mathcal{X}|\theta^{(i-1)})\,dy$$

Dieser Schritt wird **Estimation-Step** genannt.

Im nächsten Schritt, dem **Maximization-Step** werden nun die Parameter θ neu berechnet, daß der eben ermittelte Erwartungswert maximiert wird:

$$\theta^{(i)} = \arg\max_{\theta} Q(\theta,\theta^{(i-1)})$$

Es wurde gezeigt, daß für jeden Berechnungsschritt gilt:

$$\mathcal{L}(\theta^{(i)}) \geq \mathcal{L}(\theta^{(i-1)})$$

d.h. der EM-Algorithmus konvergiert monoton gegen ein **lokales** Maximum. Der Beweis soll hier nicht beschrieben werden; nachzulesen ist er z.B. in [2]. Es wird nicht garantiert, daß der EM-Alorithmus gegen ein globales Maximum konvergiert. Um optimale Ergebnisse zu erzielen, ist daher eine gewisse Vorkenntnis über die Eigenschaften der Einsatzdomäne unerläßlich.

4 Hidden Markov Models

In der Computerlinguistik muß nach gangbaren Lösungen gesucht werden, um linguistische Phänomene zu modellieren und zumindest sinnvoll zu approximieren.

Als Beispiel sei hier die Aufgabe genannt, den Wörtern eines Textes ihre Wortarten zuzuweisen. Alles was vorliegt, ist ein Text als eine Folge von Wörtern. Einen solchen Text kann man in Annäherung als eine Folge von "zufälligen" Ereignissen interpretieren, und zwar in dem Sinne, daß auf eine Wortart mit einer bestimmten Wahrscheinlichkeit eine andere Wortart folgt, und für jede Wortart dann mit einer bestimmten Wahrscheinlichkeit ein konkretes Wort generiert wird. Betrachtet man den Text als Ergebnis des gesamten Zufallsexperiments, so ist die Abfolge der Wörter erkennbar, nicht mehr jedoch die Folge der Wortarten, die die Wörter generierte.

Formal gesehen kann ein Text also als ein Zufallsexperiment mit einer Menge von Zufallsvariablen $\{O_1, \ldots, O_T, X_1, \ldots, X_T\}$ gesehen werden. Die Werte der O_i ist die Folge der beobachtbaren Daten (die Realisierung), also hier die einzelnen Wörter. Die Werte der X_i sind die versteckten Daten, nämlich die zu den Wörtern gehörigen Wortarten.

Die zugrundeliegende Verteilung ist also eine gemeinsame Verteilung[1] der Menge von Zufallsvariablen $\{O_1, \ldots, O_T, X_1, \ldots, X_T\}$. Hidden Markov Models modellieren solche Verteilungen.

Definition 4.1 *Ein Hidden Markov Model ist ein Modell für eine gemeinsame Verteilung einer Menge von Zufallsvariablen $\{O_1, \ldots, O_T, X_1, \ldots, X_T\}$. Die O_i sind entweder stetig oder diskret verteilt und sichtbar; die X_i sind diskret verteilt und unsichtbar.*

Ein Hidden Markov Model ist ein 5-Tupel $\langle S, K, \Pi, A, B\rangle$, dabei ist

- $S = \{s_1, \ldots s_N\}$*: die Menge der Zustände*
- K*: das Ausgabealphabet*
- Π*: die Menge der Anfangswahrscheinlichkeiten für Startzustände*
- A*: die Menge der Übergangswahrscheinlichkeiten*
- B*: die Menge der Ausgabewahrscheinlichkeiten*

Die Zufallsvariablen können dabei wie folgt in das Schema eingegliedert werden: Die Werte der O_t sind die beobachteten Daten, d.h. die Werte der Zufallsvariablen

[1] In der englischen Literatur wird hierfür der Begriff *joint distribution* verwendet.

liegen in K, also $O_t : \Omega \to K$. Die Werte der X_t sind die durchlaufenen Zustände, also $X_t : \Omega \to S$.

Für die von Hidden Markov Models modellierten stochastischen Prozesse werden zwei Grundannahmen gemacht, genannt "Markov[2] -Annahmen":

- **"Begrenzter Horizont"**: Die Wahrscheinlichkeit, daß ein bestimmter Zustand s_i erreicht wird, ist nur vom vorherigen Zustand abhängig:

$$P(X_{t+1} = s_i \mid X_1, \ldots, X_t) = P(X_{t+1} = s_i \mid X_t)$$

- **Zeitinvariant**: Die Wahrscheinlichkeit, daß ein bestimmter Zustand s_i erreicht wird, ist immer gleich, egal zu welchem Zeitpunkt:

$$P(X_{t+1} = s_i \mid X_t) = P(X_2 = s_i \mid X_1); X_t = X_1$$

  Diese beiden Annahmen sind strenggenommen so stark, daß jede linguistische Analyse mit HMMs eigentlich Unsinn ist. Niemand wird zum Beispiel ernsthaft glauben, daß das Auftauchen einer Wortart nur von der Wortart davor abhängig ist (Begrenzter Horizont). Ebensowenig trifft die Zeitinvarianz auf Sprache zu: Zum Beispiel kann innerhalb einer NP auf ein Nomen mit recht hoher Wahrscheinlichkeit wieder ein Artikel folgen (z.B. "Der/Art Besitzer/N des/Art Fahrzeugs/N"), aber an der Grenze von NP und VP ist sicherlich ein Verb viel wahrscheinlicher. Das heißt, es kommt sicher sehr wohl darauf an, zu welcher Zeit ein Übergang stattfindet. Nichtsdestotrotz zeigt die Erfahrung, daß man mit Hidden Markov Models solche linguistischen Phänomene mit gutem Erfolg approximieren kann.

Im Folgenden soll näher auf die Modellparameter Π, A und B eingegangen werden.

- $\Pi = \{\pi_1, \ldots, \pi_N\}$ ist die Menge der Anfangswahrscheinlichkeiten für jeden Zustand, $\pi_i = P(X_1 = s_i)$. Es handelt sich hier um einen erweiterten Begriff des "Startzustandes" aus der Automatentheorie. Ein Zustand mit Anfangswahrscheinlichkeit $\pi_i > 0$ ist mit dieser Wahrscheinlichkeit ein "Startzustand".

[2]Nach A.A. Markoff, 1856-1922. Da "Hidden Markov Model" auch im Deutschen ein feststehender Begriff ist, und die englische Schreibweise "Markov" Teil dieses Begriffs ist, soll diese auch hier beibehalten werden.

- A ist eine Matrix, die die Wahrscheinlichkeiten für die Übergänge zwischen zwei Zuständen enthält:

$$a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i),\ a_{ij} \geq 0,\ \sum_{j=1}^{N} a_{ij} = 1\ \forall i$$

  Die Nebenbedingung $a_{ij} \geq 0$ besagt, daß es auch Zustände geben kann, zwischen denen keine Kante existiert (was äquivalent ist zu einer Kante mit Übergangswahrscheinlichkeit 0). Die Bedingung $\sum_{j=1}^{N} a_{ij} = 1\ \forall i$ stellt sicher, daß sich Übergangswahrscheinlichkeiten aller aus einem Zustand s_i ausgehenden Kanten zu 1 addieren (d.h. es gibt keinen Zustand, der nicht mehr verlassen werden kann).

- B ist die Menge der Wahrscheinlichkeiten für die Ausgabe eines bestimmten Symbols $k \in K$ in einem Zustand s_i; $b_i(k) = P(o_t = k \mid X_t = s_i)$.

Zurück zum Beispiel am Anfang dieses Abschnitts. Das Ziel ist, zu einem gegebenen Text automatisch alle Wörter mit den zugehörigen Wortarten zu annotieren. Wie beschrieben kann ein HMM diese Aufgabe lösen, wenn seine Parameter (Π, A, B) richtig gesetzt sind. Ein HMM muß also trainiert werden, indem man mithilfe eines vorgetaggten Textes die Parameter so anpaßt, daß das HMM für die Wortfolgen in diesem Text die richtigen Wortartfolgen mit möglichst geringer Fehlerrate ausgibt.

Es steht also eine Menge von sichtbaren Daten zur Verfügung, der Text. Diese Daten wurden durch Ereignisse generiert, nämlich die Folge der Wortarten, die nicht mehr sichtbar ist. Weiterhin nimmt man ein stochastisches Modell an, ein HMM, das in der Lage ist, solche Daten zu repräsentieren, dessen Parameter angepaßt werden sollen. Dies sind genau die Voraussetzungen für eine Parameter-Bestimmung mit der Maximum-Likelihood Methode, die in Abschnitt 2 beschrieben wurde. Der Algorithmus, der die Parameter eines HMMs ermittelt heißt Baum-Welch-Algorithmus und ist ein Spezialfall des EM-Algorithmus.

5 Der Baum-Welch Algorithmus

Der Baum-Welch Algorithmus berechnet iterativ einen Maximum-Likelihood-Schätzwert für ein Modell $\theta = (\Pi, A, B)$ eines HMM bei einer gegebenen Realisierung $\mathcal{O}$:

$$\arg\max_{\theta} P(\mathcal{O} \mid \theta)$$

Der Algorithmus dient dazu, ein Hidden Markov Model zu trainieren, was bedeutet, die Parameter des HMM so zu setzen, daß es ein bestimmtes Phänomen so gut wie möglich modelliert.

Der Algorithmus funktioniert so, daß basierend auf einem beliebigen anfänglichen Modell die Wahrscheinlichkeit der Realisation berechnet wird. Während der Berechnung wird festgehalten, wie oft Übergänge und Ausgabesymbole verwendet wurden. Im nächsten Schritt werden die Parameter so neu berechnet, daß häufiger benutzte Übergänge und Ausgabesymbole höhere Wahrscheinlichkeiten erhalten als weniger häufig benutzte. Nach der Anpassung wird das Modell für die Realisierung eine höhere Wahrscheinlichkeit errechnen, also besser angepaßt sein. Die zwei Schritte werden so oft wiederholt, bis sich die Modellparameter nur noch unwesentlich ändern.

5.1 Effiziente Berechnungsmethoden

Die Wahrscheinlichkeit $P(\mathcal{O} \mid \theta)$ der Realisierung $\mathcal{O} = (o_1, \ldots, o_T)$ kann folgendermaßen berechnet werden.

Sei zunächst $\mathcal{X} = (X_1, \ldots, X_T)$ eine Folge von T Zuständen.

$$\begin{aligned} P(\mathcal{O} \mid \theta) &= \sum_{\mathcal{X}} P(\mathcal{O}|\mathcal{X}, \theta) P(\mathcal{X}|\theta) \\ &= \sum_{\mathcal{X}} \pi_{X_1} \prod_{t=1}^{T} a_{X_t X_{t+1}} b_{X_t}(o_t) \end{aligned}$$

Die Formel besagt folgendes: Die Wahrscheinlichkeit von $\mathcal{O}$ gegeben zunächst eine einzige Zustandssequenz $(X_1, \ldots, X_T)$ ist das Produkt aus der Anfangswahrscheinlichkeit des Zustands X_1, der Übergangswahrscheinlichkeiten $a_{X_t X_{t+1}}$ von Zustand X_t nach X_{t+1} und den Ausgabewahrscheinlichkeiten $b_{X_t}(o_t)$ für ein Symbol o_t im Zustand X_t.

Die Wahrscheinlichkeit von $\mathcal{O}$ gegeben alle möglichen Zustandssequenzen $X \in \mathbb{X}$ ist die Summe der oben genannten Wahrscheinlichkeit bei einer Zustandssequenz über alle Zustandssequenzen.

Die Berechnung der obigen Formel erfordert im allgemeinen Fall $N^{T+1}(2T+1)$ Multiplikationen.

> Aus einer Menge von N Zuständen können N^{T+1} verschiedene Zustandfolgen der Länge $T+1$ erzeugt werden. Die Berechnung der Wahrscheinlichkeit für $\mathcal{O}$ gegeben eine Zustandsfolge benötigt $2T+1$ Multiplikationen (Pro Zeitschritt t 2 Multiplikationen: Übergangswahrscheinlichkeit mal Ausgabewahrscheinlichkeit mal die Wahrscheinlichkeiten aus den vorherigen Zeitschritten, das ganze T mal. Hinzu kommt noch eine Multiplikation für die Anfangswahrscheinlichkeit).

Es ist offensichtlich, daß die Berechnung dieser Formel zu aufwendig ist. Das Problem liegt darin, daß für jede Zustandssequenz alle Berechnungen komplett neu ausgeführt werden. Sinnvoller ist es, für jeden Zustand die Gesamtwahrscheinlichkeit zu speichern, daß man zu einem Zeitpunkt t bei bisher gesehener Realisierung $(o_1, \ldots, o_t)$ diesen Zustand erreicht. Auf diese Weise berechnet man jeden Schritt nur einmal.

5.1.1 "Forward-Procedure"

Die Forward-Procedure berechnet die Gesamtwahrscheinlichkeit, daß ein Zustand i zum Zeitpunkt t erreicht wird unter der Vorgabe, daß der bisher gesehene Teil der Realisierung $(o_1, \ldots, o_t)$ ist:

$$\alpha_t(i) = P(o_1 o_2 \ldots o_{t-1}, X_t = s_i \mid \theta)$$

$\alpha_t(i)$ kann rekursiv wie folgt berechnet werden:

- **Basisfall:**

  $$\alpha_1(i) = \pi_i b_i(o_1),\ 1 \leq i \leq N$$

  Die Gesamtwahrscheinlichkeit, Zustand s_i im Zeitpunkt 1 zu erreichen, ist die Anfangswahrscheinlichkeit für s_i multipliziert mit der Wahrscheinlichkeit, daß o_1 ausgegeben wird.

- **Induktion:**

  $$\alpha_{t+1}(i) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_i(o_{t+1}),\ 1 \leq t \leq T-1, 1 \leq j \leq N$$

Die Gesamtwahrscheinlichkeit, Zustand s_j im Zeitpunkt $t+1$ zu erreichen, ist die Summe der Wahrscheinlichkeiten von einem beliebigen Zustand s_i in Zustand s_j zu kommen unter Ausgabe des nächsten Symbols o_{t+1}.

- **Gesamt:**
$$P(\mathcal{O} \mid \theta) = \sum_{i=1}^{N} \alpha_T(i)$$
Die Gesamtwahrscheinlichkeit der Realisierung ist die Summe der Wahscheinlichkeiten, zum Zeitpunkt T in einen beliebigen Zustand zu kommen.

5.1.2 "Backward Procedure"

Die Backward Procedure funktioniert analog; nur wird hier die Gesamtwahrscheinlichkeit berechnet, daß sich das HMM im Zustand s_i zum Zeitpunkt t befindet, und ab hier noch die Sequenz $(o_t \dots o_T)$ gesehen wird.

$$\beta_t(i) = P(o_t \dots o_T \mid X_t = i, \theta)$$

Die Kombination beider ergibt:

$$P(\mathcal{O} \mid \theta) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i),\ 1 \le t \le T$$

Durch die Verwendung von α und β kann z.B. die Wahrscheinlichkeit daß zum Zeitpunkt t der Übergang von Zustand s_i nach s_j unter Ausgabe von o_t benutzt wird durch folgenden einfachen Ausdruck berechnet werden:

$$P(X_t = s_i, X_{t+1} = s_j \mid O, \theta) = \frac{\alpha_t(i)\ a_{ij} b_i(o_{t+1})\ \beta_{t+1}(i)}{\sum_{m=1}^{N} \alpha_t(m)\beta_t(m)}$$

5.2 Klassische Darstellung von Baum-Welch

Gegeben sei eine Realisierung $\mathcal{O} = (o_1, \dots, o_T)$ und ein (möglicherweise zufällig gewähltes) Modell $\theta = (\Pi, A, B)$.

Definition 5.1 *$p_t(i,j), 1 \le t \le T, 1 \le i,j \le N$ sei die Wahrscheinlichkeit, daß der Übergang zwischen Zustand s_i und Zustand s_j zur Zeit t bei einer gegebenen*

Realisation $\mathcal{O}$ genommen wird:

$$\begin{aligned} p_t(i,j) &:= P(X_t = s_i, X_{t+1} = s_j \mid \mathcal{O}, \theta) \\ &= \frac{\alpha_t(i)\ a_{ij} b_i(o_{t+1})\ \beta_{t+1}(j)}{\sum_{m=1}^{N} \alpha_t(m)\beta_t(m)} \end{aligned}$$

Weiterhin ist $\gamma_i(t) = \sum_{j=1}^{N} p_t(i,j)$ die Wahrscheinlichkeit, daß Zustand i zum Zeitpunkt t bezüglich $\mathcal{O}$ erreicht wird.

Im **Estimation-Step** werden nun die Erwartungswerte für die Anzahl der Übergänge pro Kante berechnet:

- $\sum_{t=1}^{T-1} \gamma_i(t)$ ist die erwartete Anzahl von Übergängen aus Zustand s_i bei Realisierung $\mathcal{O}$.
- $\sum_{t=1}^{T-1} p_t(i,j)$ ist die erwartete Anzahl von Übergängen von Zustand s_i nach s_j bei Realisierung $\mathcal{O}$.

Im **Maximization-Step** werden nun die Modellparameter neu berechnet anhand der Erwartungswerte:

- $\hat{\pi}_i = \gamma_i(1)$
  Die neuen Anfangswahrscheinlichkeiten sind die Wahrscheinlichkeiten, daß Zustand s_i zum Anfang der Zustandssequenz bezüglich $\mathcal{O}$ auftritt.
- $\hat{a}_{ij} = \dfrac{\sum_{t=1}^{T-1} p_t(i,j)}{\sum_{t=1}^{T-1} \gamma_i(t)}$
  Die neue Wahrscheinlichkeit für einen Übergang von Zustand s_i nach Zustand s_j ist die erwartete Anzahl der Übergänge von s_i nach s_j dividiert durch die erwartete Gesamtzahl der Übergänge aus s_i (durch die Normierung entsteht hier wieder eine Wahrscheinlichkeit $0 \leq a_{ij} \leq 1$).
- $\hat{b}_i(k) = \dfrac{\sum_{t=1}^{T} \gamma_i(t)\, \delta_{k,o_t}}{\sum_{t=1}^{T} \gamma_i(t)}$
  δ_{k,o_t} ist wie folgt definiert: $\delta_{k,o_t} := \begin{cases} 1; k = o_t \\ 0; sonst \end{cases}$

Die neue Wahrscheinlichkeit für die Ausgabe eines Symbols k im Zustand s_i ist die erwartete Anzahl im Zustand s_i zu sein und dabei das Symbol $k = o_t$ auszugeben, dividiert durch die erwartete Anzahl, überhaupt im Zustand s_i zu sein.

Damit wurde ein neues Modell $\hat{\theta} = (\hat{\Pi}, \hat{A}, \hat{B})$ berechnet. Wie für den EM-Algorithmus allgemein gezeigt, gilt $P(\mathcal{O} \mid \hat{\theta}) \geq P(\mathcal{O} \mid \theta)$.

Die beschriebene Version des Baum-Welch Algorithmus ist die originale Version. Sie ist nicht sehr "EM-typisch", weil der Baum-Welch Algorithmus vor der allgemeinen Formalisierung des EM-Algorithmus entwickelt wurde. In [3] wird eine typischere Herleitung der obigen Parameter gegeben. Diese wird im nächsten Abschnitt wiedergegeben.

5.3 Herleitung durch die Q-Funktion

Seien wieder $\mathcal{O} = (o_1, \ldots, o_T)$ eine Realisierung und $X : \Omega \to \mathbb{X}$ eine Zufallsvariable, deren Werte Zustandssequenzen $\mathcal{X} = (X_1, \ldots, X_T)$ sind.

In Abschnitt 3 wurden die Begriffe der Likelihood-Funktion für die unvollständigen Daten und die vollständigen Daten eingeführt.

Die unvollständigen Daten sind hier die beobachteten Symbole, also $\mathcal{O}$. Die vollständigen Daten bestehen aus $\mathcal{O}$ und einer zugrundeliegenden Zustandssequenz $\mathcal{X}$.

Also ist die Likelihood des Modells θ bezüglich der vollständigen Daten:

$$\mathcal{L}(\theta \mid \mathcal{O}, X) = P(\mathcal{O}, X \mid \theta)$$

Analog zu der Beschreibung in Abschnitt 3 ist

$$h_{\mathcal{O},\theta}(X) := \log \mathcal{L}(\theta \mid \mathcal{O}, X)$$

Das heißt, für die Funktion Q gilt[3]:

[3]Es ist zu beachten, daß die Zufallsvariable für die versteckte Zustandssequenz diskret verteilt ist. Daher muß hier nicht das Integral über eine Dichte berechnet werden, sondern die Summe über eine Wahrscheinlichkeit.

$$Q(\theta, \theta^{(i-1)}) = E(h_{\mathcal{O},\theta}(X \mid \mathcal{O}, \theta^{(i-1)}) = \sum_{\mathcal{X}} h_{\mathcal{O},\theta}(\mathcal{X}) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)})$$

oder, durch Ersetzen von h durch die Definition:

$$Q(\theta, \theta^{(i-1)}) = \sum_{\mathcal{X}} \log P(\mathcal{O}, \mathcal{X} \mid \theta) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)})$$

Für eine Zustandssequenz $\mathcal{X} = (X_1, \ldots, X_T)$ kann $P(\mathcal{O}, \mathcal{X} \mid \theta)$ so dargestellt werden[4]:

$$P(\mathcal{O}, \mathcal{X} \mid \theta) = \pi_{X_0} \prod_{t=1}^{T} a_{X_{t-1} X_t} b_{X_t}(o_t)$$

Also:

$$\begin{aligned} Q(\theta, \theta^{(i-1)}) &= \sum_{\mathcal{X}} \log(\pi_{X_0} \prod_{t=1}^{T} a_{X_{t-1} X_t} b_{X_t}(o_t)) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) \\ &= \sum_{\mathcal{X}} \log \pi_{X_0} P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) + \\ &\quad \sum_{\mathcal{X}} (\sum_{t=1}^{T} \log a_{X_{t-1} X_t}) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) + \\ &\quad \sum_{\mathcal{X}} (\sum_{t=1}^{T} \log b_{X_t}(o_t)) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) \end{aligned}$$

indem der log in das Produkt hineingezogen wird, d.h. Q zerfällt in drei Summanden, die einzeln optimiert werden.

Der erste Summand kann vereinfacht werden:

$$\sum_{\mathcal{X}} \log \pi_{X_0} P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) = \sum_{i=1}^{N} \log \pi_i P(\mathcal{O}, X_0 = s_i \mid \theta^{(i-1)})$$

Es wird ja über alle Zustandsfolgen $\mathcal{X}$ summiert, aber nur immer nur der Wert von X_0 betrachtet.

[4]Die hier aufgeführte Gleichung unterscheidet sich leicht von der ursprünglich genannten in der Weise, daß eine Ausgabe $b_i(k)$ schon mit dem Zustand s_{i-1} ausgegeben wird. Dies ändert jedoch nichts an der Aussage, da ja nur "ein Faktor in einem Produkt nach vorn geschoben wird".

Durch partielle Ableitung nach π_i dieses Ausdrucks und Bestimmung der Extremstellen unter der Einschränkung daß $\sum_i \pi_i = 1$ ergibt sich

$$\hat{\pi}_i = \frac{P(\mathcal{O}, X_0 = s_i \mid \theta^{(i-1)}}{P(\mathcal{O} \mid \theta^{(i-1)})}$$

Der zweite Term wird vereinfacht zu:

$$\sum_{\mathcal{X}} (\sum_{t=1}^{T} \log a_{X_{t-1}X_t}) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \log a_{ij} P(\mathcal{O}, X_{t-1} = s_i, X_t = s_j \mid \theta^{(i-1)})$$

Partielle Ableitung nach a_{ij} und Bestimmen der Maxima unter der Einschränkung $\sum_{j=1}^{N} a_{ij} = 1$ ergibt:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} P(\mathcal{O}, X_{t-1} = i, X_t = j \mid \theta^{(i-1)})}{\sum_{t=1}^{T} P(\mathcal{O}, X_{t-1} = s_i \mid \theta^{(i-1)})}$$

Der letzte Term ergibt vereinfacht

$$\sum_{\mathcal{X}} (\sum_{t=1}^{T} \log b_{X_t}(o_t)) P(\mathcal{O}, \mathcal{X} \mid \theta^{(i-1)}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{t=1}^{T} \log b_i(o_t) P(\mathcal{O}, X_t = s_i \mid \theta^{(i-1)})$$

Partielle Ableitung nach b_i und Bestimmung der Maxima unter der Einschränkung $\sum_{k=1}^{L} b_i(k) = 1$ ergibt:

$$\hat{b}_i(k) = \frac{\sum_{t=1}^{T} P(\mathcal{O}, X_t = s_i \mid \theta^{(i-1)}) \, \delta_{o_t,k}}{\sum_{t=1}^{T} P(\mathcal{O}, X_t = s_i \mid \theta^{(i-1)})}$$

Die Ausdrücke kann man auch mithilfe der Forward- und Backward-Variablen $\alpha_t(i)$ und $\beta_t(i)$ ausdrücken:

$$\hat{\pi}_i = \frac{P(\mathcal{O}, X_0 = s_i \mid \theta^{(i-1)}}{P(\mathcal{O} \mid \theta^{(i-1)})} = \gamma_i(0)$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} P(\mathcal{O}, X_{t-1} = s_i, X_t = s_j \mid \theta^{(i-1)})}{\sum_{t=1}^{T} P(\mathcal{O}, X_{t-1} = s_i \mid \theta^{(i-1)})} = \frac{\sum_{t=1}^{T} p_{t-1}(i,j)}{\sum_{t=1}^{T} \gamma_i(t-1)}$$

$$\hat{b}_i(k) = \frac{\sum_{t=1}^{T} P(\mathcal{O}, X_t = s_i \mid \theta^{(i-1)})\, \delta_{o_t,k}}{\sum_{t=1}^{T} P(\mathcal{O}, X_t = s_i \mid \theta^{(i-1)})} = \frac{\sum_{t=1}^{T} \gamma_i(t)\, \delta_{k,o_t}}{\sum_{t=1}^{T} \gamma_i(t)}$$

Man erkennt, daß als Ergebnis der Optimierung dieselben Terme entstehen wie im vorigen Abschnitt.

6 Implementation von Baum-Welch

Im Folgenden wird eine konkrete Implementation des Baum-Welch Algorithmus in Java beschrieben. Der vollständige Code befindet sich im Anhang; zudem ist der Code auch unter

`http://www.sfs.nphil.uni-tuebingen.de/~wunsch/bw/index.html`

elektronisch verfügbar.

Das Programm trainiert ein Hidden Markov Model mit dem in den vorigen Abschnitten beschriebenen Baum-Welch Algorithmus. Trainiert wird die "Crazy Soft Drink Machine" wie sie Manning & Schütze in ihrem Buch "Foundation of Natural Language Processing" ([1]) vorstellen. Es ist in der Tat eine Spielzeuganwendung; sie wurde gewählt, weil sowohl Daten als auch das statistische Modell noch relativ leicht überschaubar sind.

6.1 Gewinnung von Trainingdaten

Die Ausgabe-Sequenzen, die in [1] aufgeführt sind, sind viel zu kurz, um ein HMM zu trainieren. Die Trainingdaten werden daher künstlich generiert, um so

Trainingdaten mit größerer Länge (bis zu 650 Symbole) zu gewinnen. Das im Anhang A gezeigte Programm `soda.java` simuliert ein Hidden Markov Model, das exakt dem in [1] auf Seite 321 gezeigten entspricht.

Das durch Wahrscheinlichkeiten gesteuerte Verhalten des HMMs wird wie folgt realisiert: Bei jedem Übergang und bei jeder Ausgabe eines Symbols wird zunächst eine Pseudo-Zufallszahl zwischen 1 und 1000 ermittelt. Hierbei kommt der Java-Zufallszahlengenerator zum Einsatz, der nahezu gleichverteilte Zufallszahlen erzeugt. Der Bereich zwischen 1 und 1000 wird nun entsprechend der Wahrscheinlichkeiten in i Intervalle aufgeteilt,

$$I_1 = [i_1 = 1 \ldots i_2], I_2 = [i_2 + 1 \ldots, i_3], \ldots, I_n = [i_{n-1} + 1 \ldots i_n = 1000].$$

Dabei gilt für die Wahrscheinlichkeit p_i: $\frac{i_{n+1} - i_n + 1}{1000} = p_n$. Verlassen also z.B. drei Kanten einen Zustand mit Wahrscheinlichkeiten $p_1 = 0,5; p_2 = 0,3; p_3 = 0,2$, dann werden den Wahrscheinlichkeiten die Intervalle $I_1 = [1 \ldots 500], I_2 = [501 \ldots 800], I_3 = [801 \ldots 1000]$ zugewiesen. Da die vom Zufallsgenerator ermittelten Zufallszahlen etwa gleichverteilt sind, wird eine solche Zufallszahl mit Wahrscheinlichkeit p_i im Intervall I_i liegen.

Auf diese Art wird eine Ausgabesequenz generiert, wobei die drei verschiedenen Getränke durch Zahlen von 0 bis 2 dargestellt werden. Die Sequenz wird in einer Datei gespeichert.

6.2 Das Treiberprogramm

Das Treiberprogramm `bw.java`, Anhang B, lädt eine gespeicherte Ausgabesequenz und initialisert ein Hidden Markov Model mit zwei Zuständen und einer Alphabetgröße von drei Zeichen.

Das initiale Modell läßt alle Übergänge und Ausgaben mit gleicher Wahrscheinlichkeit zu.

Sodann wird der Baum-Welch Algorithmus (die Funktion `hmm.train()` aufgerufen.

6.3 Die Implementierung des HMM

Im Anhang C ist die Klasse `HMM` abgedruckt. Diese ist eine Implementation eines state-emitting Hidden Markov Models. Die Arrays `pi[i]` und `a[i][j]` entsprechen π_i und a_{ij}, `b[i][k]` entspricht $b_i(k)$. Die Indizes `k` sind dabei zugleich auch die eigentlichen Symbole des Alphabets.

Die zwei Methoden `forwardProc` und `backwardProc` berechnen die Forward- bzw. Backward-Variablen und legen diese jeweils in einem zweidimensionalen Array `fwd[][]` bzw. `bwd[][]` über die Zustände und die Zeit ab. Damit ist $\alpha_t(i) =$ `fwd[i][t]` und $\beta_t(i) =$ `bwd[i][t]`. Die Implementierung der zwei Methoden richtet sich ganz genau nach der Definition wie in den Abschnitten 5.1.1 und 5.1.2 gezeigt. Es sei angemerkt, daß im gesamten Programm alle Indizes stets bei 0 anfangen – im Text beginnen sie immer bei 1.

Die Funktionen `gamma` und `p` berechnen die Werte γ und p aus dem Text.

Die Funktion `train()` ist die Implementation des Baum-Welch Algorithmus. Auch hier konnte im Wesentlichen die Definition direkt implementiert werden. Der Estimation und der Maximization Step werden jedoch kollabiert. Weiterhin sei auf die Implementation von $\gamma_i(t)$ hingewiesen: $gamma_i(t)$ kann auf zwei verschiedene Arten definiert werden, die äquivalent sind:

$$\gamma_i(t) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

oder aber wie in Abschnitt 5.2:

$$\gamma_i(t) = \sum_{j=1}^{N} p_t(i,j) = \sum_{j=1}^{N} \frac{\alpha_t(i)\ a_{ij} b_i(o_{t+1})\ \beta_{t+1}(j)}{\sum_{m=1}^{N} \alpha_t(m)\beta_t(m)}$$

Beide drücken die Wahrscheinlichkeit aus, zum Zeitpunkt t in Zustand s_i zu sein bezüglich einer Realisation $\mathcal{O}$ und einem Modell θ. Für die Implementierung von γ wird die erste Version verwendet, da man sich hier einen Schleifendurchlauf spart, und auf die vorberechneten α und β zurückgegriffen werden kann.

Zur Implementierung von $p_t(i,j)$ (Funktion `p(...)`):

Für Zeit T ist $p_T(i,j)$ strenggenommen undefiniert, weil auf die undefinierten Variablen $\beta_{T+1}(j)$ und $b_i(o_{T+1})$ zugegriffen wird:

$$p_t(i,j) = \frac{\alpha_t(i)\ a_{ij} b_i(o_{t+1})\ \beta_{t+1}(j)}{\sum_{m=1}^{N} \alpha_t(m)\beta_t(m)}$$

Daher wird im Programm gesetzt:

$$p_T(i,j) = \frac{\alpha_T(i)\ a_{ij}}{\sum_{m=1}^{N} \alpha_T(m)\beta_T(m)}$$

d.h. Wahrscheinlichkeiten jenseits der eigentlichen Realisierung und der Zustandsfolge werden auf 1 gesetzt.

6.4 Beispieldurchläufe

Dieser Abschnitt zeigt einige Beispieldurchläufe mit dem Lerner.

6.4.1 Manning & Schütze's Beispiel

Manning und Schütze geben in ihrem Buch [1] als Beispiel für einen Baum-Welch Durchlauf die Schätzung der Modellparameter in einem Schritt für die Sequenz {lemonade, ice tea, cola}. Die initialen Parameter des HMM sind dabei die wie in [1] auf S. 321 gezeigt.

Die Datei `crazysoda.seq` hat folgenden Inhalt:

```
3
210
```

Der Aufruf des Hauptprogramms mit `java bw 1` (für einen Trainingsdurchlauf) ergibt folgende Ausgabe:

```
Initiale Parameter:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,70000  a(0,1) = 0,30000
a(1,0) = 0,50000  a(1,1) = 0,50000
```

```
b(0,0) = 0,60000  b(0,1) = 0,10000  b(0,2) = 0,30000
b(1,0) = 0,10000  b(1,1) = 0,70000  b(1,2) = 0,20000

Trainiertes Modell:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,54862  a(0,1) = 0,45138
a(1,0) = 0,80488  a(1,1) = 0,19512

b(0,0) = 0,40367  b(0,1) = 0,13761  b(0,2) = 0,45872
b(1,0) = 0,14634  b(1,1) = 0,85366  b(1,2) = 0,00000
```

Die Ergebnisse sind also die gleichen wie in [1] auf S. 336.

Nach 19 Durchläufen konvergiert das Modell auf folgende Parameter:

```
Initiale Parameter:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,70000  a(0,1) = 0,30000
a(1,0) = 0,50000  a(1,1) = 0,50000

b(0,0) = 0,60000  b(0,1) = 0,10000  b(0,2) = 0,30000
b(1,0) = 0,10000  b(1,1) = 0,70000  b(1,2) = 0,20000

Trainiertes Modell:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,00000  a(0,1) = 1,00000
a(1,0) = 1,00000  a(1,1) = 0,00000

b(0,0) = 0,50000  b(0,1) = 0,00000  b(0,2) = 0,50000
b(1,0) = 0,00000  b(1,1) = 1,00000  b(1,2) = 0,00000
```

Dies ist für die kurze Sequenz sicherlich ein optimales Modell (siehe [1], S. 336 Exercise 9.4).

6.4.2 Trainingssequenz von 650 Zeichen Länge

Als nächstes Beispiel ein Durchlauf einer mit `java soda 650` künstlich erzeugten Trainingssequenz. Die Wahrscheinlichkeiten sind dabei wie im Anhang gezeigten Programmcode gleichverteilt (die verwendete Datei `crazysoda.seq` ist auch auf der genannten Webpage vorhanden).

```
Initiale Parameter:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,50000  a(0,1) = 0,50000
a(1,0) = 0,50000  a(1,1) = 0,50000

b(0,0) = 0,33333  b(0,1) = 0,33333  b(0,2) = 0,33333
b(1,0) = 0,33333  b(1,1) = 0,33333  b(1,2) = 0,33333

Trainiertes Modell:
pi(0) = 1,00000
pi(1) = 0,00000

a(0,0) = 0,68662  a(0,1) = 0,31338
a(1,0) = 0,30340  a(1,1) = 0,69660

b(0,0) = 0,64099  b(0,1) = 0,11277  b(0,2) = 0,24625
b(1,0) = 0,11353  b(1,1) = 0,61610  b(1,2) = 0,27036
```

In diesem Beispiel wurden die "richtigen" Modellparameter (die, mit denen die Trainingssequenz generiert wurde) recht gut gelernt.

Es sei allerdings darauf hingewiesen, daß die Qualität der Ergebnisse sehr stark von der zufällig generierten Trainingssequenz abhängt.

6.5 Rechengenauigkeit

Wie in [1] in Abschnitt 9.4.1 erwähnt, werden die Wahrscheinlichkeiten für lange Sequenzen sehr klein. Das Programm verwendet den Datentyp `double`, dessen Genauigkeit gerade noch ausreicht, um auf Trainingssequenzen mit Länge 650 zu trainieren. Auf dem verwendeten Testrechner (PC mit Pentium-III Prozessor)

können Sequenzen mit einer Länge von 700 Zeichen nicht mehr bearbeitet werden; schon nach dem ersten Schritt gehen sämtliche Werte auf 0.

A Trainingdaten-Generator: `soda.java`

```
import java.util.*;
import java.io.*;

/** Simuliert Manning & Schützes "Crazy Soft Drink Machine" -
    Generiert Ausgabesequenzen der Länge t */
public class soda {
  /** Anfangswahrscheinlichkeiten */
  private int pi_min[];
  private int pi_max[];
  /** Übergangswahrscheinlichkeiten */
  private int a_min[][];
  private int a_max[][];

  /** Ausgabewahrscheinlichkeiten */
  private int b_min[][];
  private int b_max[][];

  /** Ausgabesymbole */
  private static final int cola = 0;
  private static final int ice_t = 1;
  private static final int lem = 2;

  /** Zustände */
  private static final int cola_pref = 0;
  private static final int ice_t_pref = 1;

  /** Initialisierung des HMM wie in M&S auf Se. 321 */
  private void init() {
    pi_min = new int[2];
    pi_max = new int[2];

    pi_min[cola_pref] = 1;
    pi_max[cola_pref] = 1000;
    pi_min[ice_t_pref] = 0;
    pi_max[ice_t_pref] = 0;

    a_min = new int[2][2];
    a_max = new int[2][2];

    a_min[cola_pref][cola_pref] = 1;
    a_max[cola_pref][cola_pref] = 700;
    a_min[cola_pref][ice_t_pref] = 701;
    a_max[cola_pref][ice_t_pref] = 1000;
```

```
  a_min[ice_t_pref][cola_pref] = 1;
  a_max[ice_t_pref][cola_pref] = 500;
  a_min[ice_t_pref][ice_t_pref] = 501;
  a_max[ice_t_pref][ice_t_pref] = 500;

  b_min = new int[2][3];
  b_max = new int[2][3];

  b_min[cola_pref][cola] = 1;
  b_max[cola_pref][cola] = 600;

  b_min[cola_pref][ice_t] = 601;
  b_max[cola_pref][ice_t] = 700;

  b_min[cola_pref][lem] = 701;
  b_max[cola_pref][lem] = 1000;

  b_min[ice_t_pref][cola] = 1;
  b_max[ice_t_pref][cola] = 100;

  b_min[ice_t_pref][ice_t] = 101;
  b_max[ice_t_pref][ice_t] = 800;

  b_min[ice_t_pref][lem] = 801;
  b_max[ice_t_pref][lem] = 1000;
}

/** Generierung von Ausgabesequenzen */
private void gen_output(int t) {
  try {
    PrintWriter pw = new PrintWriter(new FileWriter("crazysoda.seq"));

    pw.println(t);

    int rnd_number = ((int) (Math.random() * 1000)) + 1;
    int state;

    for (state = 0; state < 2; state++) {
      if ((pi_min[state] <= rnd_number) && (pi_max[state] >= rnd_number))
        break;
    }

    for (int i = 0; i < t; i++) {
      rnd_number = ((int) (Math.random() * 1000)) + 1;
      for (int symb = 0; symb < 3; symb++) {
```

```
          if ((b_min[state][symb] <= rnd_number) && (b_max[state][symb]) >=
              rnd_number) {
            printSymbol(symb, pw);
            break;
          }
        }

        rnd_number = ((int) (Math.random() * 1000)) + 1;
        for (int newState = 0; newState < 2; newState++) {
          if ((a_min[state][newState] <= rnd_number) &&
              (a_max[state][newState] >= rnd_number)) {
            state = newState;
            break;
          }
        }
      }
      pw.println();
      pw.close();
    }
    catch (IOException e) {
      System.out.println("crazysoda.seq kann nicht angelegt werden.");
      System.exit(0);
    }
  }

  /** Ausgeben eines Symbols */
  private void printSymbol(int s, PrintWriter pw) {
    pw.print(s);
  }

  /** Hauptprogramm. Aufruf mit java soda <t>, wobei t die Länge der
      generierten Sequenz ist. */
  public static void main(String argv[]) {
    soda s = new soda();
    s.init();
    s.gen_output(Integer.parseInt(argv[0]));
  }
}
```

B HMM-Trainer Hauptprogramm: `bw.java`

```
import java.io.*;

/** Trainiert ein HMM so, daß es sich verhält wie
    Manning & Schütze's "Crazy Soft Drink Machine", unter Einsatz
    des Baum-Welch Algorithmus
    (Foundations of Natural Language Processing, S. 321)

    @author Holger Wunsch (wunsch@sfs.nphil.uni-tuebingen.de)
*/

public class bw {
  public static void main(String argv[]) {
    HMM hmm = new HMM(2, 3);

    hmm.pi[0] = 1.0;
    hmm.pi[1] = 0.0;

    hmm.a[0][0] = 0.5;
    hmm.a[0][1] = 0.5;
    hmm.a[1][1] = 0.5;
    hmm.a[1][0] = 0.5;

    hmm.b[0][0] = 1.0/3.0;
    hmm.b[0][1] = 1.0/3.0;
    hmm.b[0][2] = 1.0/3.0;
    hmm.b[1][0] = 1.0/3.0;
    hmm.b[1][1] = 1.0/3.0;
    hmm.b[1][2] = 1.0/3.0;

    try {
      BufferedReader br = new BufferedReader(new FileReader("crazysoda.seq"));
      int olen = Integer.parseInt(br.readLine());
      int[] o = new int[olen];
      String os = br.readLine();

      for (int i = 0; i < olen; i++)
        o[i] = Integer.parseInt(os.substring(i, i + 1));

      System.out.println("Initiale Parameter:");
      hmm.print();

      hmm.train(o, Integer.parseInt(argv[0]));
```

```
      System.out.println();

      System.out.println("Trainiertes Modell:");
      hmm.print();
    }
    catch (FileNotFoundException e) {
      System.out.println("crazysoda.seq Datei fehlt. Erzeugen mit 'java soda'");
      System.exit(0);
    }
    catch (IOException e) {
      System.out.println("Probleme beim Lesen von crazysoda.seq");
      System.exit(0);
    }
  }
}
```

C Hidden Markov Model: `HMM.java`

```
import java.text.*;

/** Diese Klasse implementiert ein Hidden Markov Model, sowie
    den Baum-Welch Algorithmus zum Training von HMMs.
    @author Holger Wunsch (wunsch@sfs.nphil.uni-tuebingen.de)
*/
public class HMM {
  /** Anzahl der Zustände */
  public int numStates;

  /** Größe des ausgabealphabets */
  public int sigmaSize;

  /** Anfangswahrscheinlichkeiten */
  public double pi[];

  /** Übergangswahrscheinlichkeiten */
  public double a[][];

  /** Ausgabewahrscheinlichkeiten */
  public double b[][];

  /** Initialisiert ein HMM.
      @param numStates Anzahl der Zustände
      @param sigmaSize Größe aus Ausgabealphabets
  */
  public HMM(int numStates, int sigmaSize) {
    this.numStates = numStates;
    this.sigmaSize = sigmaSize;

    pi = new double[numStates];
    a = new double[numStates][numStates];
    b = new double[numStates][sigmaSize];
  }

  /** Implementierung des Baum-Welch Algorithmus für HMMs.
      @param o das Trainingset
      @param steps die Anzahl der Schritte
  */
  public void train(int[] o, int steps) {
    int T = o.length;
    double[][] fwd;
    double[][] bwd;
```

```
double pi1[] = new double[numStates];
double a1[][] = new double[numStates][numStates];
double b1[][] = new double[numStates][sigmaSize];

for (int s = 0; s < steps; s++) {
  /* Berechnen der Forward- und Backward Variablen bzgl. des
     aktuellen Modells */
  fwd = forwardProc(o);
  bwd = backwardProc(o);

  /* Neuschätzung der Anfangswahrscheinlichkeiten */
  for (int i = 0; i < numStates; i++)
    pi1[i] = gamma(i, 0, o, fwd, bwd);

  /* Neuschätzung der Übergangswahrscheinlichkeiten */
  for (int i = 0; i < numStates; i++) {
    for (int j = 0; j < numStates; j++) {
      double num = 0;
      double denom = 0;
      for (int t = 0; t <= T - 1; t++) {
        num += p(t, i, j, o, fwd, bwd);
        denom += gamma(i, t, o, fwd, bwd);
      }
      a1[i][j] = divide(num, denom);
    }
  }

  /* Neuschätzen der Ausgabewahrscheinlichkeiten */
  for (int i = 0; i < numStates; i++) {
    for (int k = 0; k < sigmaSize; k++) {
      double num = 0;
      double denom = 0;

      for (int t = 0; t <= T - 1; t++) {
        double g = gamma(i, t, o, fwd, bwd);
        num += g * (k == o[t] ? 1 : 0);
        denom += g;
      }
      b1[i][k] = divide(num, denom);
    }
  }
  pi = pi1;
  a = a1;
  b = b1;
```

```
  }
}


/** Berechnet die Forward-Variablen f(i,t) für Zustände i zum Zeitpunkt
    t unter einer Realisation O und der aktuellen Parameterbelegung
    @param o die Realisation O
    @return ein Verbund f(i,t) über die Zustände und die Zeit, der die
            Forward-variablen enthält.
*/
public double[][] forwardProc(int[] o) {
  int T = o.length;
  double[][] fwd = new double[numStates][T];

  /* Basisfall */
  for (int i = 0; i < numStates; i++)
    fwd[i][0] = pi[i] * b[i][o[0]];

  /* Induktion */
  for (int t = 0; t <= T-2; t++) {
    for (int j = 0; j < numStates; j++) {
      fwd[j][t+1] = 0;
      for (int i = 0; i < numStates; i++)
        fwd[j][t+1] += (fwd[i][t] * a[i][j]);
      fwd[j][t+1] *= b[j][o[t+1]];
    }
  }

  return fwd;
}

/** Berechnet die Backward-Variablen b(i,t) für Zustände i zum Zeitpunkt
    t unter einer Realisation O und der aktuellen Parameterbelegung
    @param o die Realisation O
    @return ein Verbund b(i,t) über die Zustände und die Zeit, der die
            Backward-Variablen enthält.
*/
public double[][] backwardProc(int[] o) {
  int T = o.length;
  double[][] bwd = new double[numStates][T];

  /* Basisfall */
  for (int i = 0; i < numStates; i++)
    bwd[i][T-1] = 1;
```

```
  /* Induktion */
  for (int t = T - 2; t >= 0; t--) {
    for (int i = 0; i < numStates; i++) {
      bwd[i][t] = 0;
      for (int j = 0; j < numStates; j++)
        bwd[i][t] += (bwd[j][t+1] * a[i][j] * b[j][o[t+1]]);
    }
  }

  return bwd;
}

/** Berechnet die Wahrscheinlichkeit P(X_t = s_i, X_t+1 = s_j | O, m).
    @param t der Zeitpunkt t
    @param i die Nummer des Zustandes s_i
    @param j die Nummer des Zustandes s_j
    @param o eine Realisation o
    @param fwd die Forward-Variablen für o
    @param bwd die Backward-Variablen für o
    @return P
*/
public double p(int t, int i, int j, int[] o, double[][] fwd, double[][] bwd) {
  double num;
  if (t == o.length - 1)
    num = fwd[i][t] * a[i][j];
  else
    num = fwd[i][t] * a[i][j] * b[j][o[t+1]] * bwd[j][t+1];

  double denom = 0;

  for (int k = 0; k < numStates; k++)
    denom += (fwd[k][t] * bwd[k][t]);

  return divide(num, denom);
}

/** Berechnet gamma(i, t) */
public double gamma(int i, int t, int[] o, double[][] fwd, double[][] bwd) {
  double num = fwd[i][t] * bwd[i][t];
  double denom = 0;

  for (int j = 0; j < numStates; j++)
    denom += fwd[j][t] * bwd[j][t];

  return divide(num, denom);
```

```
  }

  /** Druckt alle Parameter eines HMM */
  public void print() {
    DecimalFormat fmt = new DecimalFormat();
    fmt.setMinimumFractionDigits(5);
    fmt.setMaximumFractionDigits(5);

    for (int i = 0; i < numStates; i++)
      System.out.println("pi(" + i + ") = " + fmt.format(pi[i]));
    System.out.println();

    for (int i = 0; i < numStates; i++) {
      for (int j = 0; j < numStates; j++)
        System.out.print("a(" + i + "," + j + ") = " +
                         fmt.format(a[i][j]) + "  ");
      System.out.println();
    }

    System.out.println();
    for (int i = 0; i < numStates; i++) {
      for (int k = 0; k < sigmaSize; k++)
        System.out.print("b(" + i + "," + k + ") = " +
                         fmt.format(b[i][k]) + "  ");
      System.out.println();
    }
  }

  /** Dividiert zwei Doubles. 0 / 0 = 0! */
  public double divide(double n, double d) {
    if (n == 0)
      return 0;
    else
      return n / d;
  }
}
```

Literatur

[1] Christopher D. Manning und Hinrich Schütze *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge MA, 1999

[2] Geoffrey McLachlan und Thriyambakam Krishnan *The EM Algorithm and Extensions*. John Wiley & Sons, New York, 1997

[3] Jeff A. Bilmes *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. International Computer Science Institute, Berkeley CA, 1998

[4] Lawrence R. Rabiner und Biing-Hwang Juang *Fundamentals of Speech Recognition*. Prentice Hall Signal Processing Series, 1993

[5] Lawrence R. Rabiner und Biing-Hwang Juang *An Introduction to Hidden Markov Models*. IEEE ASSP Magazine, January 1986

[6] Lawrence R. Rabiner *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proceedings of the IEEE, Vol. 77, No. 2, February 1989

[7] Jürgen Lehn und Helmut Wegmann *Einführung in die Statistik*. Teubner, Stuttgart, 2000

[8] Ulrich Krengel *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg, Braunschweig, 2000

[9] Seymour Lipschutz und John Schiller *Introduction to Probability and Statistics*. Schaum's Outline Series, McGraw-Hill, New York, 1998

[10] *Duden - Rechnen und Mathematik* Dudenverlag, Mannheim, 1994