Bachelorarbeit

Zum Thema:

„State-Splitting für Hidden-Markov-Modelle“

Christof Leonhardt
s0157476@mail.zih.tu-dresden.de

Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Dipl.-Inf. Toni Dietze
Beginn:	10.04.2013
Abgabedatum:	03.07.2013

Technische Universität Dresden
Fakultät Informatik
Institut für Theoretische Informatik
Grundlagen der Programmierung

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit wird es immer wichtiger, dass Computer Dinge wie Gesten, Sprache, Handgeschriebenes und ähnliches erkennen können, um die Interaktion des Menschen mit ihnen zu erleichtern. Auch das maschinelle Übersetzen natürlicher Sprache rückt immer weiter in den Fokus der Forschung. Um die oben genannten Aufgaben bewältigen zu können, werden meist statistische Modelle verwendet. Ein Modell, welches dabei sehr häufig verwendet wird, ist das *Hidden-Markov-Modell*. Hierbei wird davon ausgegangen, dass hinter bestimmten Beobachtungen oder Beobachtungsfolgen die eigentlichen Informationen *versteckt* sind [JM09, S. 210, 211].

Grundsätzlich kann ein Hidden-Markov-Modell als probabilistischer, endlicher Automat aufgefasst werden, an dessen Zuständen jeweils eine Wahrscheinlichkeitsverteilung über Beobachtungen notiert ist. Damit ein Hidden-Markov-Modell eingesetzt werden kann, muss es jedoch zunächst lernen, wie sich die Wahrscheinlichkeitsverteilungen an den Zuständen zusammensetzen. Auch die Zustandsübergänge selbst müssen *trainiert* werden. Das Problem hierbei besteht darin, dass zunächst nicht klar ist, wie viele Zustände für das Modell benötigt werden. Eine mögliche Lösung ist, zu das Training mit genau einem Zustand zu beginnen. Danach kann der Zustand in zwei neue Zustände *geteilt* und ein erneutes Training angeschlossen werden. Eine weitere Überlegung, damit das Modell nicht zu groß wird, wäre anschließend, Zustände, die das Modell nur geringfügig verbessern, wieder *zusammenzuführen*. Dieses Verfahren wurde bereits bei *probabilistischen kontextfreien Grammatiken (PCFG)* unter dem Begriff *State-Splitting* angewandt [Pet+06]. Auch ein State-Splitting für Hidden-Markov-Modelle wurde bereits in Petrov, Pauls und Klein [PPK07] vorgestellt. Das dort vorgestellte Verfahren ist aber weit weniger formal beschrieben, als dies in dieser Arbeit geschehen soll.

In dieser Bachelorarbeit wird demnach versucht, das State-Splitting für Hiddden-Markov-Modelle zu formalisieren und zu implementieren. Dafür werden zunächst Hidden-Markov-Modelle, formal beschrieben. Danach werden sowohl das *Splitting*, als auch das *Merging* dieser Modelle formalisiert. Im vierten Abschnitt wird der Trainingsalgorithmus vorgestellt. Dabei wird zunächst ein Pseudocode entworfen, der dann implementiert wird. Im darauf folgenden Abschnitt werden Hidden-Markov-Modelle, die mit dem neuen State-Splitting Verfahren trainiert wurden, mit solchen verglichen, die auf herkömmliche Weise trainiert wurden. Schließlich wird ein Resümee gezogen und ein weiterer Ausblick gegeben.

2. Hidden-Markov-Modelle

Bei der maschinellen Verarbeitung natürlicher Sprache (*natural language processing*, kurz: NLP) können Hidden-Markov-Modelle oft als Sprachmodelle eingesetzt werden. Dabei besteht die Beobachtungsmenge des Modells aus dem Vokabular einer Sprache. Die Zustände können als Wortarten[1](*part of speech tags*) verstanden werden.

In diesem Abschnitt sollen Hidden-Markov-Modelle formalisiert werden. Dabei werden zunächst einige grundlegende Definitionen aufgestellt. Anschließend werden Hidden-Markov-Modelle formal beschrieben. Nach dieser Beschreibung wird dann der Formalismus durch ein Beispiel abgerundet. Zum Schluss wird das Training und der Einsatz des State-Splittings motiviert.

2.1. Wahrscheinlichkeiten

Als Einstieg werden zunächst Wahrscheinlichkeitsverteilungen definiert werden, damit klar ist, wie diese in der Arbeit verwendet werden.

Sei X eine nichtleere, abzählbare Menge. $p\colon X \to [0,1]$ ist genau dann eine *Wahrscheinlichkeitsverteilung über* X, wenn gilt:

$$\sum_{x \in X} p(x) = 1.$$

Die *Menge aller Wahrscheinlichkeitsverteilungen über* X wird mit $\mathcal{M}(X)$ bezeichnet.

Seien X und Y nichtleere, abzählbare Mengen. $p\colon (X \times Y) \to [0,1]$ ist genau dann eine *bedingte Wahrscheinlichkeitsverteilung über* Y *gegeben* X, wenn gilt:

$$\forall x \in X : \sum_{y \in Y} p(y \mid x) = 1.$$

Die *Menge aller bedingten Wahrscheinlichkeitsverteilungen über* Y *gegeben* X wird mit $\mathcal{M}(Y \mid X)$ bezeichnet.

Seien X eine endliche, abzählbare Menge und p eine Wahrscheinlichkeitsverteilung über X. Sei außerdem $C \subseteq X$. Die *Likelihood von* X *über* C ist dann definiert durch:

$$\mathrm{le}(C, p) = \prod_{c \in C} p(c)$$

[1] z.B. *Adj* für *Adjektiv*, *N* für *Nomen*, *V* für *Verb*, usw.

2.2. Hidden-Markov-Modelle

Es folgt nun die formale Beschreibung von *Hidden-Markov-Modellen*. Alle weiteren Definitionen der vorliegenden Arbeit bauen auf dieser Definition auf.

Ein *Alphabet* ist eine endliche, nichtleere Menge.

Ein Hidden-Markov-Modell, oft bezeichnet mit M, ist ein Tupel $(Q, O, q_\mathrm{s}, p, b)$, wobei

$$\begin{aligned}
Q & \text{ ein Alphabet (von } \textit{Zuständen}\text{)},\\
O & \text{ ein Alphabet (von } \textit{Beobachtungen}\text{)},\\
q_\mathrm{s} \in Q & \text{ ein } \textit{Spezialzustand},\\
p\colon (Q \times Q) \to [0,1] & \text{ eine Bedingte Wahrscheinlichkeitsverteilung über } Q \text{ gegeben } Q \text{ und}\\
b\colon \big(Q \setminus \{q_\mathrm{s}\} \times O\big) \to [0,1] & \text{ eine Bedingte Wahrscheinlichkeitsverteilung über } O \text{ gegeben } Q \setminus \{q_\mathrm{s}\} \text{ sind.}
\end{aligned}$$

Die Menge aller Hidden-Markov-Modelle wird mit $\mathbb{M}$ bezeichnet.

2.2.1. Ableitungen

Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell.
Die *Menge aller Ableitungen (bezüglich M)*, notiert durch D_M, ist die Menge $\big(Q \setminus \{q_\mathrm{s}\} \times O\big)^*$.

Eine *Ableitung (bezüglich M)* ist also ein Tupel der Form:

$$\begin{aligned}
&\big((q_1, o_1), (q_2, o_2), \ldots, (q_n, o_n)\big) \text{ wobei}\\
&(q_1, o_1), (q_2, o_2), \ldots, (q_n, o_n) \in Q \setminus \{q_\mathrm{s}\} \times O \text{ sind.}
\end{aligned}$$

2.2.2. Sequenzen von Beobachtungen

Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell.

Die *Menge aller Sequenzen (bezüglich M)* ist die Menge O^*.

Ein *Korpus (bezüglich O)* ist eine endliche Teilmenge von O^*.

Eine *Sequenz (bezüglich M)* ist also eine Folge der Form:

$$o_1 o_2 \ldots o_n \text{ wobei } o_1, o_2, \ldots, o_n \in O \text{ sind.}$$

Sei yield die Funktion, die aus einer Ableitung bezüglich M eine Sequenz bezüglich M wie folgt bestimmt:

$$\begin{aligned}
&\mathrm{yield}\colon D_M \to O^*\\
&\mathrm{yield}\big((q_1, o_1), (q_2, o_2), \ldots, (q_n, o_n)\big) = o_1 o_2 \ldots o_n
\end{aligned}$$

Ihre Rückabbildung ist wie folgt definiert:

$$\mathrm{yield}^{-1}(o_1 o_2 \ldots o_n) = \{d \mid d \in D_M, \mathrm{yield}(d) = o\}$$

2.2.3. Zusammensetzung der Wahrscheinlichkeiten

Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell und $d = \big((q_1, o_1), (q_2, o_2), \ldots, (q_n, o_n)\big)$ eine Ableitung bezüglich M.

Die Wahrscheinlichkeit der Ableitung d, bezeichnet mit $\mathrm{P}_M(d)$, berechnet sich aus:

$$\mathrm{P}_M(d) = p(q_1 \mid q_s) \cdot \Big(\prod_{i=1}^{n-1} p(q_{i+1} \mid q_i)\Big) \cdot \Big(\prod_{i=1}^{n} b(o_i \mid q_i)\Big) \cdot p(q_s \mid q_n).$$

Lemma 1: Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell, wobei für alle $q_1, q_2 \in Q : p(q_1 \mid q_2) > 0$. Dann ist P_M für Ableitungen eine Wahrscheinlichkeitsverteilung. Es gilt also:

$$\sum_{d \in D_M} \mathrm{P}_M(d) = 1.$$

Beweis: siehe Abschnitt A

Die Wahrscheinlichkeit einer Sequenz o bezüglich M, bezeichnet mit $\mathrm{P}_M(o)$, berechnet sich aus:

$$\mathrm{P}_M(o) = \sum_{d \in \mathrm{yield}^{-1}(o)} \mathrm{P}_M(d).$$

Lemma 2: Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell, wobei für alle $q_1, q_2 \in Q : p(q_1 \mid q_2) > 0$. Dann ist P_M für Beobachtungen eine Wahrscheinlichkeitsverteilung. Es gilt also:

$$\sum_{o \in O^*} \mathrm{P}_M(o) = 1.$$

Beweis: siehe Abschnitt A

2.2.4. Likelihood eines Hidden-Markov-Modells

Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell und $C \subseteq O^*$ ein Korpus über O. Die Likelihood des Hidden-Markov-Modells berechnet sich aus:

$$\mathrm{le}(C, \mathrm{P}_M) = \prod_{c \in C} \mathrm{P}_M(c).$$

2.3. Beispiel

Angenommen ein Hidden-Markov-Modell M soll zur Modellierung eines Sprachmodells verwendet werden. Dabei müssen der Zustandsmenge Q Wortarten und der Beobachtungsmenge O das Vokabular einer Sprache zugeordnet werden. Die Sätze, die analysiert werden, sind dann die Sequenzen von Beobachtungen.

Beispiel: Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell, wobei

$$\begin{aligned} Q &= \{\mathrm{V}, \mathrm{N}\} \cup \{q_\mathrm{s}\} && \text{die Menge von Wortarten und der Spezialzustand,} \\ O &= \{er, hat, hunger\} && \text{das Vokabular einer Sprache sind.} \end{aligned}$$

Die bedingte Wahrscheinlichkeitsverteilung p über Q gegeben Q ist

$$\begin{array}{lll} p(\mathrm{V} \mid q_\mathrm{s}) = 0{,}30 & p(N \mid q_\mathrm{s}) = 0{,}70 & p(q_\mathrm{s} \mid q_\mathrm{s}) = 0{,}00 \\ p(\mathrm{V} \mid \mathrm{V}) = 0{,}10 & p(\mathrm{N} \mid \mathrm{V}) = 0{,}70 & p(q_\mathrm{s} \mid \mathrm{V}) = 0{,}20 \\ p(\mathrm{V} \mid \mathrm{N}) = 0{,}30 & p(\mathrm{N} \mid \mathrm{N}) = 0{,}10 & p(q_\mathrm{s} \mid \mathrm{N}) = 0{,}60 \end{array}$$

und die Verteilung b über O gegeben $Q \setminus \{q_\mathrm{s}\}$ ist

$$\begin{array}{lll} b(er \mid \mathrm{V}) = 0{,}20 & b(hat \mid \mathrm{V}) = 0{,}50 & b(hunger \mid \mathrm{V}) = 0{,}30 \\ b(er \mid \mathrm{N}) = 0{,}60 & b(hat \mid \mathrm{N}) = 0{,}10 & b(hunger \mid \mathrm{N}) = 0{,}30. \end{array}$$

Das gezeichnete Hidden-Markov-Modell kann wie in Abbildung 2.1 grafisch dargestellt werden.

Betrachten wir nun die Ableitung $d = \big((er, \mathrm{V}), (hat, \mathrm{N})\big)$. Die Wahrscheinlichkeit ist dann:

$$\begin{array}{rcccccccccc} \mathrm{P}(d) & = & p(\mathrm{V} \mid q_\mathrm{s}) & \cdot & p(q_\mathrm{s} \mid \mathrm{N}) & \cdot & p(\mathrm{N} \mid \mathrm{V}) & \cdot & b(er \mid \mathrm{V}) & \cdot \\ & & b(hat \mid \mathrm{N}) & & & & & & & \\ & = & 0{,}30 & \cdot & 0{,}60 & \cdot & 0{,}70 & \cdot & 0{,}20 & \cdot \\ & & 0{,}10 & = & 0{,}00252 & & & & & \end{array}$$

Dagegen hat die Ableitung von $\big((er, \mathrm{N}), (hat, \mathrm{V})\big)$ eine höhere Wahrscheinlichkeit von 0,00420. Dies ist insofern sinnvoll, als dass in der Sprachwissenschaft *hat* als Verb und *er* als Nomen klassifiziert werden. Diese Werte sind allerdings von den Wahrscheinlichkeiten des Modells abhängig. Wie diese zu Stande kommen, wird in Abschnitt 2.4 erklärt.

Im Regelfall sind in der NLP nur Sequenzen von Beobachtungen vorhanden [JM09, S. 211]. Nehmen wir also an, der Satz $s = erhat$ sei gegeben. Um nun die Wahrscheinlichkeit $P(s)$ dieser Beobachtungssequenz herauszufinden, muss zunächst die Menge $\mathrm{yield}^{-1}(s)$ gefunden werden. In diesem Beispiel sieht die Menge, wie folgt, aus:

$$\begin{aligned} \mathrm{yield}^{-1}(s) = \big\{&\big((er, \mathrm{V}), (hat, \mathrm{V})\big),\ \big((er, \mathrm{V}), (hat, \mathrm{N})\big), \\ &\big((er, \mathrm{N}), (hat, \mathrm{V})\big),\ \big((er, \mathrm{N}), (hat, \mathrm{N})\big)\big\} \end{aligned}$$

Nun kann die Wahrscheinlichkeit der Sequenz ermittelt werden:

$$\begin{array}{rcccccccc} P(s) & = & \sum_{d \in \mathrm{yield}^{-1}(s)} P(d) & & & & & & \\ & = & P\big((er, \mathrm{V}), (hat, \mathrm{V})\big) & + & P\big((er, \mathrm{V}), (hat, \mathrm{N})\big) & + & & & \\ & & P\big((er, \mathrm{N}), (hat, \mathrm{V})\big) & + & P\big((er, \mathrm{N}), (hat, \mathrm{N})\big) & & & & \\ & = & 0{,}00060 & + & 0{,}00252 & + & & & \\ & & 0{,}00420 & + & 0{,}00756 & = & 0{,}01488 & & \end{array}$$

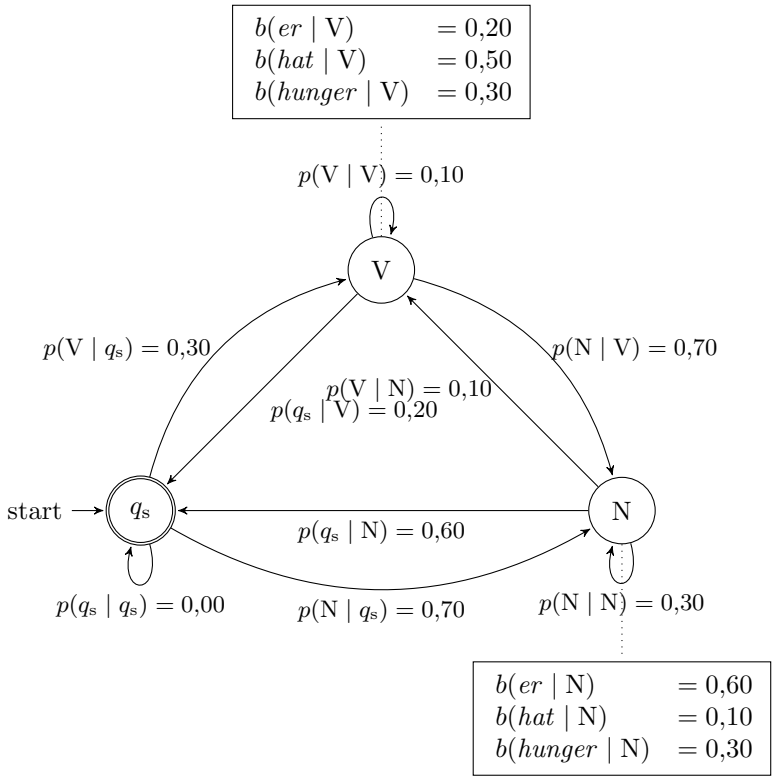

Abbildung 2.1.: Darstellung eines Hidden-Markov-Modells

2.4. Training

Damit ein Hidden-Markov-Modell sinnvoll eingesetzt werden kann, müssen dessen Wahrscheinlichkeitsverteilungen angemessen gewählt werden. Um dies zu erreichen wird das Modell trainiert. Hierfür muss ein Trainingsalgorithmus gefunden werden. In dieser Arbeit wird der *Baum-Welch-Algorithmus* verwendet [Bau72]. Da der Schwerpunkt dieser Arbeit auf dem State-Splitting liegt, wird der Algorithmus ohne Herleitung in Abschnitt 4.1.2 angegeben.

Um den Baum-Welch-Algorithmus verwenden zu können, müssen die Anzahl an Zuständen und eine Verteilung aller Wahrscheinlichkeiten von Anfang an vorgegeben werden. Wie gut das Modell nach dem Training ist, hängt sehr stark von diesen Startparametern ab. Da nicht von Beginn an klar ist, wie viele Zustände das Hidden-Markov-Modell benötigt, ist eine Möglichkeit dieses Problem zu lösen, mehrere Trainingsversuche mit verschiedener Zustandsanzahl und unterschiedlichen Verteilungen durchzuführen und zum Schluss das beste Modell zu verwenden. Das bedeutet allerdings einen sehr großen zeitlichen und auch rechnerischen Aufwand. Die zweite, zeitlich

und rechnerisch weniger intensive Möglichkeit ist das State-Splitting.

Dabei wird zunächst ein Modell mit einem Zustand und den entsprechenden Wahrscheinlichkeitsverteilungen initialisiert. Nach diesem Schritt ist das State-Splitting in mehrere Durchläufe eingeteilt. Als erstes werden alle Zustände des Hidden-Markov-Modell in jeweils zwei neue aufgeteilt. Danach erfolgt das Training mit dem *Baum-Welch-Algorithmus*. Auf diese Weise passt sich das Modell besser an die Trainingsdaten an. Nach diesem Schritt werden die Zustände des trainierten Hidden-Markov-Modells wieder zusammengeführt, die nur einen geringen Beitrag zur Verbesserung dieses Modells erlauben. Zuletzt folgt ein weiteres Training. Danach folgt der nächste Durchlauf des State-Splittings.

Dem aufmerksamen Leser wird aufgefallen sein, dass das Zusammenführen von Zuständen, das Aufteilen selbiger teilweise wieder rückgängig macht. Dies ist eine nützliche Eigenschaft, da zu oft geteilte Zustände das Modell nur geringfügig verbessern. Daher ist es sinnvoller diese Zustände nicht weiter aufzuteilen. Andererseits werden Zustände, die noch einer Verfeinerung bedürfen, weiter geteilt. Damit kann das State-Splitting als Erweiterung des Baum-Welch-Algorithmus angesehen werden.

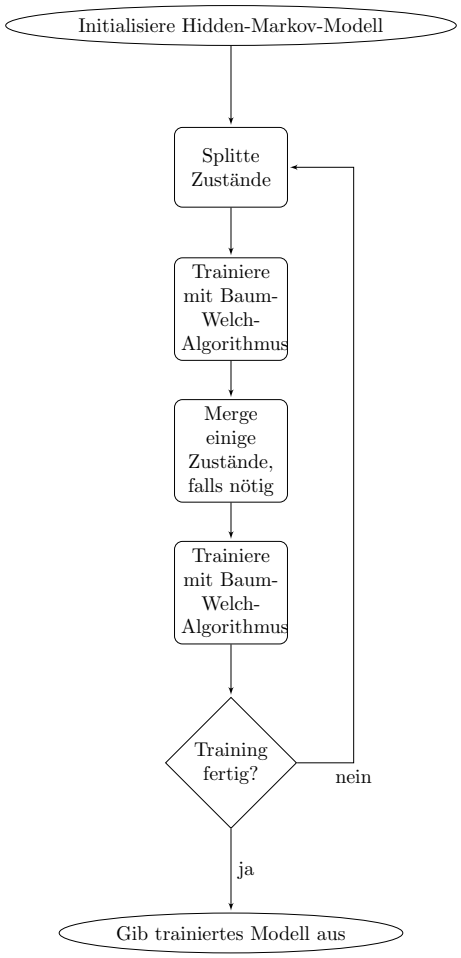

Abbildung 2.2.: Ablaufplan des Trainings eines Hidden-Markov-Modells mit State-Splitting

3. State-Splitting

Da in Abschnitt 2 geklärt wurde, was ein Hidden-Markov-Modell ist und wie das State-Splitting funktionieren soll, wird letzteres im Folgenden beschrieben.

Das Verfahren wurde bereits bei probabilistischen kontextfreien Grammatiken (PCFG) [Pet+06] angewandt. Dabei kam heraus, dass Splitting alleine eine sehr große Menge an Zuständen generiert. Effektiver ist es, wenn nach dem *Trainieren* eines Hidden-Markov-Modells Zustände, die nur eine geringe Verbesserung des Modells erlauben, wieder zusammengeführt, gemerged, werden. Hierbei darf das Merging nur bestimmte Zustände zusammenführen. Außerdem wird ein Maß benötigt, welches angibt, wie gut, oder schlecht ein Splitting war. - Das Verhältnis der Likelihood. Nachfolgend werden das Splitting, Merging und die Berechnung des Verhältnisses der Likelihood aufgestellt.

3.1. Splitting bei Hidden-Markov-Modellen

Um die Übersichtlichkeit zu wahren wird das Splitting zunächst lediglich für Zustände beschrieben. Danach wird es auf das Splitting von Hidden-Markov-Modellen geliftet.

Sei Q eine Teilmenge von Zuständen eines Hidden-Markov-Modells und q_s der zugehörige Spezialzustand. Dann gilt:

$$\mathrm{split}(Q) = \left\{q^1, q^2 \mid q \in Q \setminus \{q_\mathrm{s}\}\right\} \cup \left(\{q_\mathrm{s}\} \cap Q\right)$$

Der Schnitt $\left(\{q_\mathrm{s}\} \cap Q\right)$ am Ende der Definition sorgt dafür, dass $q_\mathrm{s} \in \mathrm{split}(Q)$ gdw. $q_\mathrm{s} \in Q$ gilt. Es ist nicht immer erwünscht, dass q_s in einer geteilten Menge von Zuständen enthalten ist (siehe Abschnitt 3.2).

Seien M, M' Hidden-Markov-Modelle mit $M = (Q, O, q_\mathrm{s}, b, p)$ und

$M' = (Q', O, q_\mathrm{s}, b', p')$. Dann gilt:

$$
\begin{aligned}
\mathrm{split}(M) = M' \text{ gdw. } & Q' = \mathrm{split}(Q),\\
& p'(q_\mathrm{s} \mid q_\mathrm{s}) = p(q_\mathrm{s} \mid q_\mathrm{s}),\\
& \text{für alle } q_1 \in Q, q_2 \in Q \setminus \{q_\mathrm{s}\} \text{ gilt:}\\
& \quad \{q_1^1, q_1^2, q_2^1, q_2^2\} = \mathrm{split}\big(\{q_1, q_2\}\big):\\
& \quad p'(q_2^1 \mid q_1^1) = p'(q_2^1 \mid q_1^2) = p'(q_2^2 \mid q_1^1) = p'(q_2^2 \mid q_1^2)\\
& \quad = \frac{p(q_2 \mid q_1)}{2},\\
& \text{für alle } q \in Q \setminus \{q_\mathrm{s}\} \text{ gilt:}\\
& \quad \{q^1, q^2\} = \mathrm{split}\big(\{q\}\big):\\
& \quad p'(q_\mathrm{s} \mid q^1) = p'(q_\mathrm{s} \mid q^2) = p(q_\mathrm{s} \mid q),\\
& \text{und für alle } o \in O \text{ und } q \in Q \setminus \{q_\mathrm{s}\} \text{ gilt:}\\
& \quad \{q^1, q^2\} = \mathrm{split}\big(\{q\}\big):\\
& \quad b'(o \mid q^1) = b'(o \mid q^2) = b(o \mid q).
\end{aligned}
$$

Beispiel: Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell wobei

$$
\begin{aligned}
Q &= \{q, q_\mathrm{s}\} && \text{die Menge von Zuständen,}\\
O &= \{er, hat, hunger\} && \text{die Menge von Beobachtungen sind.}
\end{aligned}
$$

Die bedingte Wahrscheinlichkeitsverteilung p über Q gegeben Q sei

$$
\begin{aligned}
p(q \mid q_\mathrm{s}) &= 1{,}00 & \qquad p(q_\mathrm{s} \mid q_\mathrm{s}) &= 0{,}00\\
p(q \mid q) &= 0{,}70 & \qquad p(q_\mathrm{s} \mid q) &= 0{,}30
\end{aligned}
$$

und die Verteilung b über O gegeben $Q \setminus \{q_\mathrm{s}\}$ sei

$$
b(er \mid q) = 0{,}20 \qquad b(hat \mid q) = 0{,}50 \qquad b(hunger \mid q) = 0{,}30.
$$

Für $\mathrm{split}(M) = (Q', O, q_\mathrm{s}, b', p')$ gilt dann:

$$
Q' = \{q^1, q^2\} \cup \{q_\mathrm{s}\}.
$$

Die bedingte Wahrscheinlichkeitsverteilung p' über Q' gegeben Q' ist dann

$$
\begin{aligned}
p'(q^1 \mid q_\mathrm{s}) &= 0{,}50 & \quad p'(q^2 \mid q_\mathrm{s}) &= 0{,}50 & \quad p'(q_\mathrm{s} \mid q_\mathrm{s}) &= 0{,}00\\
p'(q^1 \mid q^1) &= 0{,}35 & \quad p'(q^2 \mid q^1) &= 0{,}35 & \quad p'(q_\mathrm{s} \mid q^1) &= 0{,}30\\
p'(q^1 \mid q^2) &= 0{,}35 & \quad p'(q^2 \mid q^2) &= 0{,}35 & \quad p'(q_\mathrm{s} \mid q^2) &= 0{,}30
\end{aligned}
$$

und für die Verteilung b' über O' gegeben $Q' \setminus \{q_\mathrm{s}\}$ gilt

$$
\begin{aligned}
b'(er \mid q^1) &= 0{,}20 & \quad b('hat \mid q^1) &= 0{,}50 & \quad b'(hunger \mid q^1) &= 0{,}30\\
b'(er \mid q^2) &= 0{,}20 & \quad b'(hat \mid q^2) &= 0{,}50 & \quad b'(hunger \mid q^2) &= 0{,}30.
\end{aligned}
$$

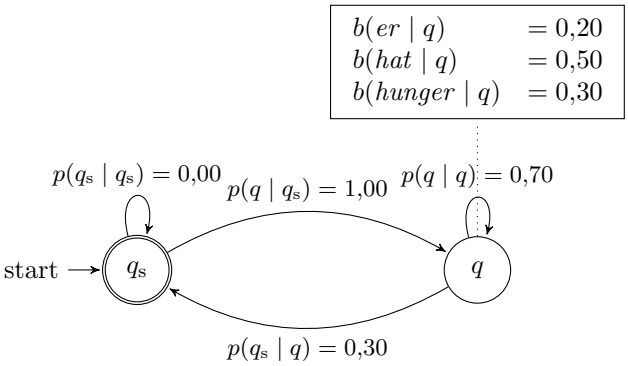

Abbildung 3.1.: Hidden-Markov-Modell M

Das gezeichnete Hidden-Markov-Modell M kann wie in Abbildung 3.1 und das gesplittete Modell split(M) kann wie in Abbildung 3.2 grafisch dargestellt werden.

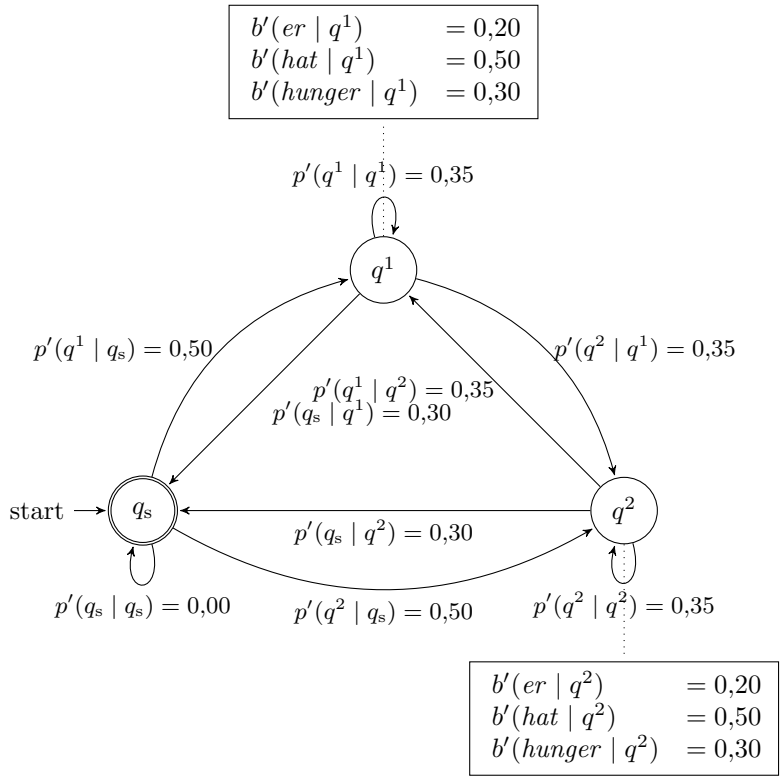

Abbildung 3.2.: gesplittetes Hidden-Markov-Modell split(M)

3.2. Merging bei Hidden-Markov-Modellen

Im Folgenden wird zunächst das Merging von Zuständen beschrieben, um die Übersichtlichkeit zu wahren. Dabei muss darauf geachtet werden, dass nur eine bestimmte Menge von Zuständen zusammengeführt werden soll. Dies ist wichtig, da Splits, die das Hidden-Markov-Modell stark verbessern, erhalten werden sollen. Anschließend wird das Merging auf Hidden-Markov-Modelle geliftet.

Sei $M' = (Q', O, q_s, p', b')$ ein Hidden-Markov-Modell und $\widehat{Q}$ eine Menge, sodass $\text{split}(\widehat{Q}) \subseteq Q' \setminus \{q_s\}$. Dann gilt:

$$\text{merge}(Q', \widehat{Q}) = \widehat{Q} \cup Q' \setminus \text{split}(\widehat{Q}).$$

Das Merging für Hidden-Markov-Modelle ist nun wie folgt definiert:

$\text{merge}(M', \widehat{Q}) = (Q, O, q_\text{s}, p, b)$, wobei $Q = \text{merge}(Q', \widehat{Q})$,

für alle $q_1, q_2 \in Q' \setminus \text{split}(\widehat{Q})$ gilt:

$p(q_2 \mid q_1) = p'(q_2 \mid q_1)$,

für alle $q' \in Q' \setminus \text{split}(\widehat{Q})$ und $q \in \widehat{Q}$ gilt:

$\{q^1, q^2\} = \text{split}\big(\{q\}\big)$:

$$p(q' \mid q) = \frac{p'(q' \mid q^1) + p'(q' \mid q^2)}{2},$$

$p(q \mid q') = p'(q^1 \mid q') + p'(q^2 \mid q')$,

für alle $q_1, q_2 \in \widehat{Q}$ gilt:

$\{q_1^1, q_1^2, q_2^1, q_2^2\} = \text{split}\big(\{q_1, q_2\}\big)$:

$$p(q_2 \mid q_1) = \frac{p'(q_2^1 \mid q_1^1) + p'(q_2^2 \mid q_1^1) + p'(q_2^1 \mid q_1^2) + p'(q_2^2 \mid q_1^2)}{2},$$

für alle $o \in O$ und $q \in Q' \setminus \text{split}(\widehat{Q})$ gilt:

$b(q \mid o) = b'(q \mid o)$

für alle $o \in O$ und $q \in \widehat{Q}$ gilt:

$\{q^1, q^2\} = \text{split}\big(\{q\}\big)$:

$$p(o \mid q) = \frac{p'(o \mid q^1) + p'(o \mid q^2)}{2}.$$

Beispiel: Sei $M' = (Q', O, q_\text{s}, p', b')$ ein Hidden-Markov-Modell welches durch Splitting entstanden ist, wobei

$$\begin{aligned} Q' &= \{q^1, q^2, q_\text{s}\} && \text{die Menge von Zuständen,} \\ O &= \{er, hat, hunger\} && \text{die Menge von Beobachtungen sind.} \end{aligned}$$

Die bedingte Wahrscheinlichkeitsverteilung p' über Q' gegeben Q' sei

$$\begin{array}{lll} p'(q^1 \mid q_\text{s}) = 0{,}30 & p'(q^2 \mid q_\text{s}) = 0{,}70 & p'(q_\text{s} \mid q_\text{s}) = 0{,}00 \\ p'(q^1 \mid q^1) = 0{,}20 & p'(q^2 \mid q^1) = 0{,}50 & p'(q_\text{s} \mid q^1) = 0{,}30 \\ p'(q^1 \mid q^2) = 0{,}70 & p'(q^2 \mid q^2) = 0{,}20 & p'(q_\text{s} \mid q^2) = 0{,}10 \end{array}$$

und die Verteilung b' über O gegeben $Q' \setminus \{q_\text{s}\}$ sei

$$\begin{array}{lll} b'(er \mid q^1) = 0{,}10 & b'(hat \mid q^1) = 0{,}10 & b'(hunger \mid q^1) = 0{,}80 \\ b'(er \mid q^2) = 0{,}50 & b'(hat \mid q^2) = 0{,}40 & b'(hunger \mid q^2) = 0{,}10. \end{array}$$

Für $\text{merge}(M', Q' \setminus \{q_\text{s}\}) = (Q, O, q_\text{s}, b, p)$ gilt dann:

$$Q = \{q\} \cup \{q_\text{s}\}.$$

Die bedingte Wahrscheinlichkeitsverteilung p über Q gegeben Q ist dann

$$p(q \mid q_\mathrm{s}) = 1{,}00 \qquad p(q_\mathrm{s} \mid q_\mathrm{s}) = 0{,}00$$
$$p(q \mid q) = 0{,}80 \qquad p(q_\mathrm{s} \mid q) = 0{,}20$$

und für die Verteilung b über O gegeben $Q \setminus \{q_\mathrm{s}\}$ gilt

$$b(er \mid q) = 0{,}30 \qquad b(hat \mid q) = 0{,}25 \qquad b(hunger \mid q) = 0{,}45.$$

Das gezeichnete Hidden-Markov-Modell M kann wie in Abbildung 3.3 und das gesplittete Modell split(M) kann wie in Abbildung 3.4 grafisch dargestellt werden.

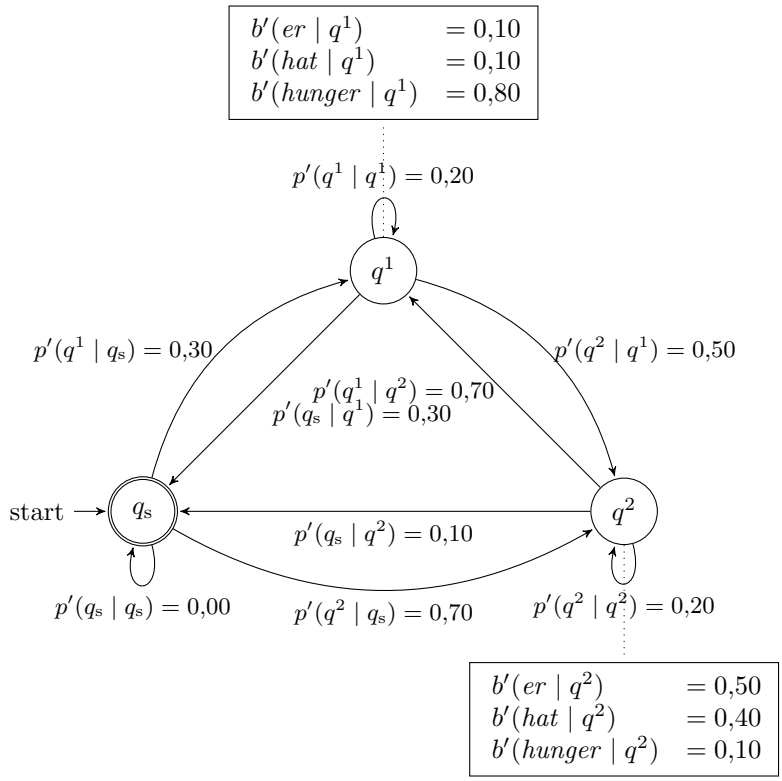

Abbildung 3.3.: Hidden-Markov-Modell M'

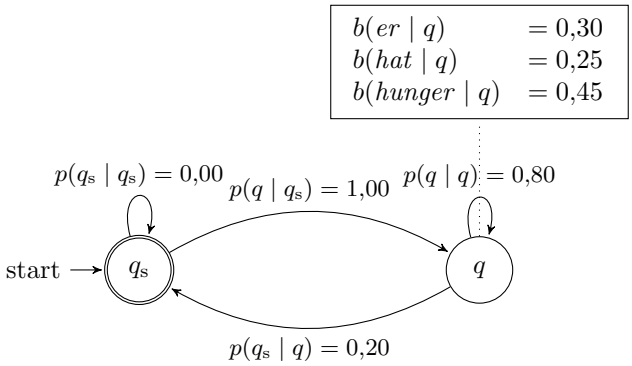

Abbildung 3.4.: zusammengeführtes Hidden-Markov-Modell merge(M')

3.3. Verhältnis der Likelihood

Die Funktion Δle berechnet den Verlust der Likelihood, wenn genau ein geteiltes Zustandspaar des Hidden-Markov-Modells zusammengeführt wird. Dazu wird das Verhältnis der Likelihood eines komplett geteilten Hidden-Markov-Modells, zu einem, bei dem ein Paar von Zuständen zusammengeführt wurde, errechnet. Dadurch ergibt sich die Veränderung der Likelihood über den ganzen Korpus.

Seien $M' = (Q', O', q_\mathrm{s}, p', b')$ ein Hidden-Markov-Modell, $\widehat{Q}$ eine Menge, sodass $\mathrm{split}(\widehat{Q}) \subseteq Q' \setminus \{q_\mathrm{s}\}$ und $C \subseteq O^*$ ein Korpus. Δle ist dann wie folgt definiert:

$$\begin{aligned}\Delta\mathrm{le}(C, M', \widehat{Q}) &= \frac{\mathrm{L}\big(C, \mathrm{merge}(M', \widehat{Q})\big)}{\mathrm{L}(C, M')} \\ &= \frac{\prod_{o \in C} P_{\mathrm{merge}(M', \widehat{Q})}(o)}{\prod_{o \in C} P_{M'}(o)} \\ &= \prod_{o \in C} \frac{P_{\mathrm{merge}(M', \widehat{Q})}(o)}{P_{M'}(o)}\end{aligned}$$

4. Training mit State-Splitting

Nachdem nun alle notwendigen Definitionen zusammengetragen wurden, wird das Training von Hidden-Markov-Modellen mit State-Splitting vorgestellt. Dafür wird zunächst ein Trainingsalgorithmus in Pseudocode entworfen und danach Implementiert.

4.1. Algorithmus

Im Folgenden wird der State-Splitting-Algorithmus vorgestellt. Für eine erfolgreiche Implementierung müsseneinige Annahmen und Vereinfachungen getroffen werden. Danach wird ein Pseudocode-Algorithmus für das Training von Hidden-Markov-Modellen mit State-Splitting entwickelt.

4.1.1. Einschränkungen und Annahmen

Die in dieser Arbeit aufgestellten Definitionen sind zwar formal korrekt, lassen sich stellenweise aber nur schwierig implementieren oder sind einfach nicht praktikabel. Daher sind noch einige, für die Implementierung wichtige Veränderungen an den Definitionen nötig, die im Folgenden beschrieben werden.

Splitting

In Abschnitt 3.1 werden die Wahrscheinlichkeiten aller Zustände durch 2 geteilt. Dadurch entsteht eine Symmetrie beider Zustände, die, wie sich im Rahmen der Implementierung des State-Splitting-Algorithmus gezeigt hat, dazu führt, dass beide Zustände exakt gleich trainiert werden. Um diesen Effekt zu vermeiden, dürfen die Wahrscheinlichkeiten nicht einfach halbiert werden.

Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell und seien $q_1, q_2 \in Q$. Beim Splitting wäre dann beispielsweise:

$$p'(q_2^1 \mid q_1^1) = p'(q_2^1 \mid q_1^2) = p'(q_2^2 \mid q_1^1) = p'(q_2^1 \mid q_2^1) = \frac{p(q_2 \mid q_1)}{2}.$$

Bei der Implementierung muss noch ein zusätzlicher Faktor addiert bzw. subtrahiert werden. Damit dieser Faktor die neue Wahrscheinlichkeit nicht zu sehr beeinflusst, sollte er von selbiger abhängig sein. Es ergibt sich also:

$$\begin{aligned} p'(q_2^1 \mid q_1^1) &= \frac{p(q_2 \mid q_1)}{2} \cdot (1+\alpha) & p'(q_2^1 \mid q_1^2) &= \frac{p(q_2 \mid q_1)}{2} \cdot (1+\beta)\,, \\ p'(q_2^2 \mid q_1^1) &= \frac{p(q_2 \mid q_1)}{2} \cdot (1-\alpha) & p'(q_2^2 \mid q_1^2) &= \frac{p(q_2 \mid q_1)}{2} \cdot (1-\beta)\,. \end{aligned}$$

Die Parameter α und β müssen jeweils im Intervall $]0,1[$ liegen und beeinflussen, wie stark die Symmetrie verändert wird. Sie müssen für jedes Paar von Zuständen neu festgelegt werden, damit keinerlei Gleichheit auftritt.

Auch die Wahrscheinlichkeiten der Sequenzen von Beobachtungen sind nach dem Splitting bei dem neu entstandenem Zustandspaar gleich. Damit auch hier das Training nicht dieselben Ergebnisse liefert, muss ebenfalls diese Gleichheit aufgelöst werden.

Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell und seien $q \in Q$ und $o_1, o_2, \ldots, o_m \in O$. Beim Splitting wäre dann:

$$\begin{aligned} b'(o_1 \mid q^1) &= b'(o_1 \mid q^2) = b(o_1 \mid q), \\ b'(o_2 \mid q^1) &= b'(o_2 \mid q^2) = b(o_2 \mid q), \\ &\ldots \\ b'(o_m \mid q^1) &= b'(o_m \mid q^2) = b(o_m \mid q). \end{aligned}$$

Hier müssen bei der Implementierung ebenfalls Parameter eingerechnet und normalisiert werden:

$$\begin{aligned} b'(o_1 \mid q^1) &= \frac{b(o_1 \mid q) \cdot (1+\alpha_1)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\alpha_i)} & b'(o_1 \mid q^2) &= \frac{b(o_1 \mid q) \cdot (1+\beta_1)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\beta_i)}, \\ b'(o_2 \mid q^1) &= \frac{b(o_2 \mid q) \cdot (1+\alpha_2)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\alpha_i)} & b'(o_2 \mid q^2) &= \frac{b(o_2 \mid q) \cdot (1+\beta_2)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\beta_i)}, \\ &\ldots & &\ldots \\ b'(o_m \mid q^1) &= \frac{b(o_m \mid q) \cdot (1+\alpha_m)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\alpha_i)} & b'(o_m \mid q^2) &= \frac{b(o_m \mid q) \cdot (1+\beta_m)}{\sum_{i=1}^{m} b(o_i \mid q) \cdot (1+\beta_i)}, \end{aligned}$$

Dabei muss für α_i und β_i

$$\sum_{i=1}^{m} \alpha_i = 1, \quad \alpha_i \in]0,1[, \qquad\qquad \sum_{i=1}^{m} \beta_i = 1, \quad \beta_i \in]0,1[,$$

$$\text{mit } \alpha_i \neq \beta_i$$

gelten.

Die Parameter beeinflussen, wie stark die Symmetrie beim Splitting der Sequenzen von Beobachtungen geändert wird. Sie müssen ebenfalls für jedes Paar von Zuständen neu festgelegt werden.

Merging

Beim Zusammenführen von Zuständen eines Hidden-Markov-Modells wird in Abschnitt 3.2 die Summe der Wahrscheinlichkeiten durch 2 geteilt. In der Praxis könnte es sich anbieten einzelne Teilsummen unterschiedlich zu gewichten. Ähnliches wurde auch bei *PCFG*s vorgenommen. [Pet+06] Dies herzuleiten würde über den Umfang dieser Arbeit hinausgehen und soll nur als Ausblick dienen, wie der Algorithmus verbessert werden kann. Gleiches gilt für die Sequenzen von Beobachtungen, die zusammengeführt werden.

Likelihood, Forward- und Backward-Algorithmus

In Abschnitt 3.3 wird der Verlust der Likelihood mit Hilfe der Wahrscheinlichkeitsverteilung $P_M(o)$ berechnet. Laut Definition wird dabei die Funktion yield^{-1} verwendet, welche alle Ableitungen bezüglich eines Hidden-Markov-Modells M findet, die eine bestimmte Sequenz von Beobachtungen o erzeugen kann.

In der Praxis wird dazu der sogenannte *Forward-Algorithmus* verwendet [Bau72; JM09, S. 213–217].

Sei $M = (Q, O, q_\text{s}, p, b)$ ein Hidden-Markov-Modell, $q \in Q$ und $o \in O^*$. Dann gilt:

$$P_M(o) = \alpha^o_{|o|}(q_\text{s}) \qquad \text{Forward-Algorithmus für Sequenz } o \text{ unter } M$$

$$\begin{aligned} \alpha^o_1(q) &= p(q_j \mid q_\text{s}) \cdot b(o_1 \mid q), \\ \alpha^o_t(q) &= \sum_{i=0}^{|Q|} \alpha_{t-1}(q_i) \cdot p(q \mid q_i) \cdot b(o_t \mid q_j) \qquad \text{[basiert auf JM09, S. 216, 217].} \end{aligned}$$

Dadurch kann Δle algorithmisch einfacher berechnet werden. Der Forward-Algorithmus verwendet dafür dynamisches Programmieren. Für jede Beobachtung wird über alle möglichen Wahrscheinlichkeiten von Zuständen im Hidden-Markov-Modell, die von der zugehörigen Beobachtungssequenz erreicht werden können, aufsummiert [JM09, S. 215].

Außerdem wird in dem Werk von Jurafsky und Martin [JM09, S. 222] der *Backward-Algorithmus* beschrieben: Sei $M = (Q, O, q_\text{s}, p, b)$ ein Hidden-Markov-Modell, $q \in Q$ und $o \in O^*$. Dann gilt:

$$\begin{aligned} \beta^o_{|o|}(q) &= p(q_\text{s} \mid q) \\ \beta^o_t(q_j) &= \sum_{i=0}^{|Q|} p(q \mid q_i) \cdot b(o_{t+1} \mid q_j) \beta_{t+1}(q_j) \qquad \text{[basiert auf JM09, S. 222].} \end{aligned}$$

4.1.2. Pseudocode

Nachfolgend sind unter anderem die in Abschnitt 3 dargelegten Formalismen in einen Pseudocode überführt worden, um die spätere Implementierung zu vereinfachen und zu visualisieren.

Baum-Welch-Algorithmus

Der Baum-Welch-Algorithmus trainiert die Wahrscheinlichkeiten der Zustände und der von Sequenzen von Beobachtungen eines initialisierten Hidden-Markov-Modells. Dabei werden der Forward- und der Backward-Algorithmus verwendet [JM09, S. 220–226].

Algorithmus 1 Baum-Welch-Algorithmus [basiert auf JM09, S. 226]

Require: $M = (Q, O, q_s, p, b)$ ist ein Hidden-Markov-Modell
$C \subseteq O^*$ ist ein Korpus
Ensure: p, b sind bereits zufällig verteilt

function BAUMWELCH(M, C)
  while nicht konvergiert **do**
    for all $o \in C, q_1 \in Q$ **do**
      ▷ E-Schritt
      for $0 \leq t \leq |o|$ **do**
        $\gamma_t(q_1) = \frac{\alpha_t(q_1) \cdot \beta_t(q_1)}{\alpha_{|o|}(q_{|Q|-1})}$
        for all $q_2 \in Q$ **do**
          $\xi_t(q_2, q_1) = \frac{\alpha_t(q_2) \cdot p(q_1|q_2) \cdot b(o_{t+1}|q_1) \cdot \beta_{t+1}(q1)}{\alpha_{|o|}(q_{|Q|-1})}$
        end for
      end for
      ▷ M-Schritt
      for all $q_2 \in Q$ **do**
        $p'(q_2 \mid q_1) = \frac{\sum_{t=1}^{|o|-1} \xi_t(q_2, q_1)}{\sum_{t=1}^{|o|-1} \sum_{q' \in Q} \xi_t(q_2|q')}$
      end for
      for all $o' \in O$ **do**
        $b'(o' \mid q_1) = \frac{\sum_{t=1, o_t = o'}^{|o|} \gamma_t(q_1)}{\sum_{t=1}^{|o|} \gamma_t(q_1)}$
      end for
    end for
    $p \leftarrow p'$
    $b \leftarrow b'$
  end while
  return $M = (Q, O, q_s, p, b)$
end function

Splitting

Der Splitting-Algorithmus teilt alle Zustände eines gegebenen Hidden-Markov-Modells in je zwei neue auf. Dabei müssen auch die einzelnen Wahrscheinlichkeiten neu berechnet werden.

Algorithmus 2 Splitting

Require: $M = (Q, O, q_\mathrm{s}, p, b)$ ist ein Hidden-Markov-Modell

function SPLIT(M)
  $Q' = \mathrm{split}(Q)$
  $p'(q_\mathrm{s} \mid q_\mathrm{s}) \leftarrow p(q_\mathrm{s} \mid q_\mathrm{s})$

  for all $q_1 \in Q \setminus \{q_\mathrm{s}\}$ **do**

    Wahrscheinlichkeiten vom Spezial- zum gesplitteten Zustand berechnen
    $\alpha \leftarrow$CREATEVALUE() ▷ Erzeugt einen zufälligen Wert im Intervall $]0, 1[$
    $p'(q_\mathrm{s} \mid q_1^1) \leftarrow \frac{p(q_\mathrm{s}|q_1)}{2} \cdot (1 + \alpha)$
    $p'(q_\mathrm{s} \mid q_1^2) \leftarrow \frac{p(q_\mathrm{s}|q_1)}{2} \cdot (1 - \alpha)$

    Wahrscheinlichkeiten von Zustand zu Zustand berechnen
    for all $q_2 \in Q$ **do**
      $\alpha \leftarrow$CREATEVALUE()
      $\beta \leftarrow$CREATEVALUE()
      $p'(q_1^1 \mid q_2^1) \leftarrow \frac{p(q_1|q_2)}{2} \cdot (1 + \alpha)$
      $p'(q_1^2 \mid q_2^1) \leftarrow \frac{p(q_1|q_2)}{2} \cdot (1 - \alpha)$
      $p'(q_1^1 \mid q_2^2) \leftarrow \frac{p(q_1|q_2)}{2} \cdot (1 + \beta)$
      $p'(q_1^2 \mid q_2^2) \leftarrow \frac{p(q_1|q_2)}{2} \cdot (1 - \beta)$
    end for

    Wahrscheinlichkeiten von Beobachtungen berechnen
    $\gamma[] \leftarrow$ CREATEDISTRIBUTION($|O|$)
    $\delta[] \leftarrow$ CREATEDISTRIBUTION($|O|$)
      ▷ Erzeugt jeweils eine zufällige Wahrscheinlichkeitsverteilung und speichert sie in einem Array

    $i \leftarrow 0$
    for all $o \in O$ **do**
      $b'(o \mid q_1^1) \leftarrow \frac{b(o|q) \cdot (1+\gamma[i])}{\sum_{j=1}^{|O|} b(o_j|q) \cdot (1+\gamma[j])}$
      $b'(o \mid q_1^2) \leftarrow \frac{b(o|q) \cdot (1+\delta[i])}{\sum_{j=1}^{|O|} b(o_j|q) \cdot (1+\delta[j])}$
      $i \leftarrow i + 1$
    end for

  end for

  return $M' = \left(\{q_1^1, q_1^2, q_2^1, q_2^2, \ldots, q_{|Q|-1}^1, q_{|Q|-1}^2, q_\mathrm{s}\}, O, q_\mathrm{s}, p', b'\right)$
end function

Merging

Der Merging-Algorithmus fasst gesplittete Zustände wieder zusammen. Dabei werden wieder die zugehörigen Wahrscheinlichkeiten verrechnet. Es werden nur die Zustände gemerged, die nicht in der Menge $\widehat{Q}$ liegen.

Algorithmus 3 Merging

Require: $M' = (Q', O, q_\mathrm{s}, p', b')$ ist ein Hidden-Markov-Modell
$\widehat{Q}$ ist Menge von Zuständen
Ensure: M' ist durch SPLIT erzeugt worden
Q Zustandsmenge eines Hidden-Markov-Modells mit $Q' = \mathrm{split}(Q)$
$\widehat{Q} \subseteq Q$

1: **function** MERGE(M', $\widehat{Q}$)

2: $Q \leftarrow \mathrm{merge}(Q', \widehat{Q})$
3: **for all** $q' \in Q' \setminus \mathrm{split}(\widehat{Q})$ **do** ▷ Alle Zustände, die nicht vereint werden
4: *Zustandsübergänge zu allen anderen Zuständen berechnen*
5: **for all** q mit $q^1, q^2 \in \widehat{Q}$ **do**
6: $p(q' \mid q) \leftarrow \frac{p'(q'|q^1)+p'(q'|q^2)}{2}$
7: $p(q \mid q') \leftarrow p'(q^1 \mid q') + p'(q^2 \mid q')$
8: **end for**
9: **for all** $q \in \widehat{Q} \cup \{q_\mathrm{s}\}$ **do**
10: $p(q' \mid q) \leftarrow p'(q' \mid q)$
11: **end for**
12: *Neue Beobachtungswahrscheinlichkeiten berechnen*
13: **for all** $o \in O$ **do**
14: $b(o \mid q') \leftarrow b'(o \mid q')$
15: **end for**

16: **end for**
17: **for all** q_1 mit $q_1^1, q_1^2 \in \widehat{Q}$ **do** ▷ Alle Zustände, die vereint werden
18: *Zustandsübergänge zu allen anderen Zuständen berechnen*
19: **for all** q_2 mit $q_2^1, q_2^2 \in Q' \setminus \mathrm{split}(\widehat{Q})$ **do**
20: $p(q_2 \mid q_1) \leftarrow \frac{p'(q_2^1|q_1^1)+p'(q_2^2|q_1^1)+p'(q_2^1|q_1^2)+p'(q_2^2|q_1^2)}{2}$
21: **end for**
22: *Neue Beobachtungswahrscheinlichkeiten berechnen*
23: **for all** $o \in O$ **do**
24: $b(o \mid q) \leftarrow \frac{b'(o|q_1^1)+b'(o|q_1^2)}{2}$
25: **end for**

26: **end for**
27: **return** $M = \big(Q, O, q_\mathrm{s}, p, b\big)$
28: **end function**

Merging wenn Verlust der Likelihood unterhalb eines Schwellwertes

Dieser Algorithmus berechnet für alle Zustände eines gegebenen Hidden-Markov-Modells den Verlust der Likelihood und vergleicht ihn mit dem Schwellwert ε. Ist der Verlust größer als ε, so würde ein Merging das Modell zu ungenau werden lassen. Ist dies nicht der Fall, wird das zugehörige Zustandspaar zunächst in die Menge *toMerge* eingefügt. Zum Schluss werden alle Zustände zusammengeführt, die in der Menge *toMerge* liegen.

Algorithmus 4 Merging, wenn nötig

Require: $M' = (Q', O, q_\mathrm{s}, p', b')$ ist ein Hidden-Markov-Modell
$C \subseteq O^*$ ist ein Korpus
$\varepsilon \in [0, 1]$ ist der minimale Verlust an Likelihood, der auftreten darf, bis zwei Zustände vereint werden. Je näher ε an 0 ist, desto öfter wird ein Merging durchgeführt[2].
Ensure: M' ist durch SPLIT erzeugt worden

```
1: function MERGEIFNECESSARY(M', C, ε)
2:     toMerge ← ∅

3:     for all q mit split({q}) ∈ Q' do

4:         Berechne den Verlust der Likelihood, wenn q zusammengeführt wurde
5:         δ ← Δle(C, M', {q})
6:         if ε ≤ δ then                ▷ Ist der Verlust noch oberhalb des Schwellwerts?
7:             toMerge ← toMerge ∪ {q}
8:         end if

9:     end for

10:    return MERGE(M', toMerge)
11: end function
```

Training mit State-Splitting

Nun kann der eigentliche Trainingsalgorithmus angegeben werden. Zunächst wird ein Hidden-Markov-Modell erstellt und initialisiert. Danach wird es gesplittet und trainiert. Nach dem Training erfolgt das Merging von Splitts, die die Likelihood des Modells nur geringfügig verbessern und ein weiteres Training. Dieser Vorgang wird so lange wiederholt, bis ein ausreichend gutes Hidden-Markov-Modell entstanden ist.

[2] In der Implementierung ist es genau anders herum. Je näher ε an 1 ist, desto öfter wird ein Merging durchgeführt. Dies liegt am Einsatz von logarithmischen Wahrscheinlichkeiten (siehe Abschnitt 4.2.1).

Algorithmus 5 Training mit State-Splitting

Require: O ist eine Menge von Beobachtungen
$C \subseteq O^*$ ist ein Korpus
$\varepsilon \in [0, 1]$ ist der minimale Verlust an Likelihood, der auftreten darf, bis zwei Zustände vereint werden. Je näher ε an 0 ist, desto öfter wird ein Merging durchgeführt[3].

1: **function** TRAIN(O, C, ε)
2: ▷ Initialisiere Hidden-Markov-Modell M
3: $Q \leftarrow \{q_s, q\}$
4: $p \leftarrow$ CREATEDISTRIBUTIONFORP($Q \setminus \{q_s\}$)
5: $b \leftarrow$ CREATEDISTRIBUTIONFORO(O)
6: $M = (Q, O, q_s, p, b)$
7: ▷ Trainiere...
8: **while** nicht konvergierend **do**
9: $M' \leftarrow$SPLIT(M)
10: $M' \leftarrow$BAUMWELCH(M, C)
11: $M \leftarrow$MERGE_IF_NECESSARY(M', C, ε)
12: $M \leftarrow$BAUMWELCH(M, C)
13: **end while**
14: **return** M
15: **end function**

4.2. Implementierung

Da nun alle benötigten Formalismen und Pseudocodes erstellt worden sind, kann die Implementierung des Programms beschrieben werden. Dazu wird zunächst der allgemeine Aufbau erläutert und die Verbindung zu den Pseudocodes hergestellt. Danach wird beschrieben, wie das Programm kompiliert und gestartet werden kann. Schließlich werden die Bestandteile erklärt, die in das Programm *Vanda Studio* übernommen wurden.

4.2.1. Dokumentation

An dieser Stelle wird der Quellcode der Implementierung grob beschrieben. Eine genaue Dokumentation ist in der *JavaDoc* [Ora09] enthalten. Außerdem wurde der Quellcode ausführlich kommentiert. An diesem Abschnitt werden der Allgemeine Aufbau des Programms und Abweichungen zum Pseudocode beschrieben.

[3] In der Implementierung ist es genau anders herum. Je näher ε an 1 ist, desto öfter wird ein Merging durchgeführt. Dies liegt am Einsatz von logarithmischen Wahrscheinlichkeiten (siehe Abschnitt 4.2.1).

Aufbau des Programms

Für die Implementierung des State-Splitting wurde auf einem bereits bestehenden-den Trainingsprogramm für Hidden-Markov-Modelle aufgebaut [jmf09]. Es ist in Java programmiert, weshalb diese Sprache auch zur Implementierung des State-Splitting-Algorithmus verwendet wurde. Das gesamte Programm besteht im Wesentlichen aus zwei *Packages*:

- BE.AC.ULG.MONTEFIORE.RUN - Package der bereits vorhandenen Implementierung
- HMM - Selbst implementiertes Package

Weitere Informationen zur bereits vorhandenen Implementierung können der Projekthomepage [jmf09] und der *JavaDoc* entnommen werden.

Das HMM-Package ist wie folgt aufgebaut:

- HMM
  - EVALUATION
    - PERPLEXITY.JAVA ..... berechnen der *Perplexity* zur Auswertung
  - MAIN
    - CONFIGURATION.JAVA ... verarbeiten und speichern übergebener Parameter
    - MAIN.JAVA ..... Verarbeitung der Ein- und Ausgabe, Errorhandling
    - PAIR.JAVA ..... Implementierung einer Paar-Klasse
    - RUNNINGMODE.JAVA ..... Enumeration für verschiedene Modi in denen das Programm laufen kann
  - PREPROCESSING
    - OBSERVATIONREADER.JAVA ..... Vorverarbeitung des Korpus
    - WORDMAP.JAVA ..... Wort- $\leftrightarrow$ Integer-Zuordnung
  - STATESPLITTING
    - ALPHA.JAVA ..... erstellt zufällige Wahrscheinlichkeitsverteilungen
    - STATESPLITTING.JAVA ..... State-Splitting-Alogrithmus

Lizenzen Die bereits vorgefertigte Implementierung steht, bis auf die Datei BE.AC.UGL.MONTEFIORE.RUN.IO.CUSTOMSTREAMTOKENIZER.JAVA unter der *BSD 3-Clause License* [Ini]. die Datei CUSTOMSTREAMTOKENIZER.JAVA ist eine modifizierte Version der gleichnamigen Klasse von *Sun Microsystems, Inc.* Daher steht sie unter der *GNU General Public License version 2* [Koh13].

Dies bedeutet, dass die gesamte Implementierung ebenfalls unter die *GNU General Public License version 2* gestellt werden muss.

Unterschiede vom Pseudocode zur Implementierung

Fast alle Pseudocodes, die in Abschnitt 4.1.2 spezifiziert wurden, sind in der Klasse HMM.STATESPLITTING.STATESPLITTING.JAVA implementiert. Einzige Ausnahme bilden der *Algorithmus 1: Baum-Welch-Algorithmus* und *Algorithmus 5: Training mit State-Splitting*. Letzterer findet sich als Methode TRAIN_SS() in der Klasse

HMM.MAIN.MAIN.JAVA. Der Grund hierfür ist, dass in der STATESPLITTING-Klasse nur die Hilfs-Algorithmen[4] für das Training implementiert worden sind. Für den Baum-Welch-Algorithmus wurde eine bereits bestehende Implementierung verwendet, genau wie für die Modellierung der Hidden-Markov-Modelle. Beides findet sich in dem *Package*
BE.AC.ULG.MONTEFIORE.RUN.

Die Java-Klasse HMM der Datei BE.AC.ULG.MONTEFIORE.RUN.JAHMM.HMM.JAVA welche ein Hidden-Markov-Modell darstellt war bereits implementiert. Ein Unterschied zu den in dieser Arbeit definierten Hidden-Markov-Modellen ist, dass die Wahrscheinlichkeit, um von q_s in einen anderen Zustand zu gelangen, durch eine einfache Wahrscheinlichkeitsverteilung über alle Zustände gegeben ist. Dadurch kann der Spezialzustand entfernt werden. Diese Eigenschaft ändert auch die Berechnung der Wahrscheinlichkeiten von q_s in den Pseudocodes *Algorithmus 2: Splitting* und *Algorithmus 3: Merging*.

Letzterer weicht sehr stark von seinem zugehörigen Pseudocode ab. Der Algorithmus sieht vor, zunächst über alle Zustände zu iterieren, die nicht vereint werden sollen und dann alle Zustände zu durchlaufen, die vereint werden sollen. Die Implementierung hingegen iteriert über die gesamte, nicht zusammengeführte, Anzahl an Zuständen einen Hidden-Markov-Modells. Dies hängt damit zusammen, dass $\widehat{Q}$ in der Implementierung Paare von Zuständen enthält, die zusammengeführt werden sollen. Dadurch ist es einfacher, zunächst über alle Zustände zu iterieren und sie mit den Paaren zu vergleichen.

Dass der MERGING-Algorithmus Paare von Zuständen übergeben bekommt, die zusammengeführt werden sollen, hängt mit der Implementierung von *Algorithmus 4: mergeIfNecessary* zusammen. Auch dieser zeichnet sich durch einige Veränderungen gegenüber des Pseudocodes aus. Bei der Implementierung wird zunächst über jeden zweiten Zustand des an die Methode übergebenen Hidden-Markov-Modells iteriert. Dabei bildet bei jeder Iteration der jeweilige Zustand, mit seinem Folgezustand ein Paar, welches zusammengeführt wird. Dies ist legitim, da beim SPLITTING-Algorithmus die Zustände in jeweils zwei aufeinanderfolgende Zustände aufgeteilt werden. Danach wird das Verhältnis zwischen dem teilweise zusammengeführten und dem übergeben Hidden-Markov-Modell berechnet. Dabei entstehen bei größeren Datenmengen zu hohe Werte. Deshalb werden beim Berechnen des Verhältnisses logarithmische Wahrscheinlichkeiten verwendet. Damit die Merging-Schwelle ε dennoch im Intervall $[0, 1]$ liegt, werden diese Verhältnisse, bevor der Vergleich mit ε stattfindet, Normalisiert. Da logarithmische Wahrscheinlichkeiten negativ sind, liegen nach diesem Schritt Modelle, deren Verlust an Likelihood hoch ist, näher bei 1, anstatt wie im Pseudocode angegeben, bei 0. Daher muss die Prüfung mit ε invertiert werden.

An einigen Stellen der Implementierung wurde der Code in weitere, kleinere Methoden aufgeteilt, um die Übersichtlichkeit des Programms zu wahren. Dies stellt einen weiterem Unterschied gegenüber des Pseudocodes dar.

[4] Splitting, Merging, Likelihood-Berechnung, usw...

4.2.2. Programm kompilieren

Um das Programm benutzen zu können, muss es zunächst kompiliert werden. Dafür wird mindestens das *JavaSDK 1.6* benötigt (während der Implementierung wurde das Programm mit *OpenJDK 6* unter *Ubuntu* kompiliert und getestet). Die Implementierung ist aufgrund der Struktur von Java plattformunabhängig. Um den Kompilierungsvorgang einzuleiten, muss zunächst in das Programmverzeichnis *HMM-SS* gewechselt werden.

Verzeichniswechsel
```
$ cd HMM-SS
```

Danach wird der Java-Compiler gestartet.

Kompilieren
```
$ javac -sourcepath src src/hmm/main/Main.java -Xlint:unchecked -d classes
```

Danach sollten im Ordner *classes* die *.class-Dateien des Programms liegen.
Nun kann noch eine *.jar-Datei erstellt werden.

*.jar erstellen
```
$ jar -cfvm HMM-SS.jar manifest -C classes /
```

Im Programmverzeichnis sollte sich nun eine Datei mit dem Namen *HMM-SS.jar* befinden.

4.2.3. Programmaufruf

Das erstellte Programm kann in vier verschiedenen Betriebsmodi laufen: *State-Splitting, Normales Training, Parsing, Perplexity.* Je nach Modus werden unterschiedliche Übergabeparameter erwartet. Welcher Modus welche Funktionalität ausführt und was für Parameter erwartet werden, wird im Folgenden beschrieben.

Übergabeparameter

Es gibt neben den Programmmodi noch weitere, zum Teil optionale Parameter. Diese sollten, zum besseren Verständnis vorher erklärt werden.

- ⟨HMM-FILE⟩ - Dateiname derjenigen Datei, in der das Hidden-Markov-Modell gespeichert ist.
- ⟨WORDMAP-FILE⟩ - Dateiname derjenigen Datei, in die Wort- ↔ Integer-Zuordnung des Korpus gespeichert ist.
- ⟨CORPUS-FILE⟩ - Dateiname der Datei, die den Korpus enthält. Dabei muss jeder Satz in einer Zeile stehen. Damit die Interpunktion mitbewertet werden kann,

muss jedes Satzzeichen von einem Leerzeichen eingerahmt sein. Außerdem wird zwischen Groß- und Kleinschreibung unterschieden.[5]

- ⟨TRAINING CYCLES⟩ - Ein optionaler Parameter, der angibt, wie viele Iterationen der *Baum-Welch-Alogithmus* durchführen soll. Der Standardwert liegt bei 5.
- ⟨ALPHA⟩ - Ein Wert im Intervall $[0, 1]$, der angibt, wie stark die Symmetrie beim State-Splitting und auch beim Initialisieren der Hidden-Markov-Modelle verändert werden soll (siehe Abschnitt 4.1.1). Der Parameter ist optional, der Standardwert liegt bei 0,01.
- ⟨EPSILON⟩ - Der Wert gibt an, ab welchem Likelihood-Verhältnis ein Merging durchgeführt wird. Je näher der Wert bei 0 liegt, desto seltener wird gemerged. Außerdem muss er ebenfalls im Intervall $[0, 1]$ liegen. Der Standardwert liegt bei 0,01.
- ⟨SEED⟩ - Zum Vergleich zweier Trainingsmethoden kann es von Nutzen sein, wenn die gleichen pseudo-zufälligen Werte zum Training verwendet werden. So kann ein Training mit gleichen Voraussetzungen geschaffen werden. Der *seed* bildet die Basis auf derer alle zufälligen Werte des Programms generiert werden. Ein gleicher *seed* bedeutet, dass die gleichen Zufallszahlen erzeugt werden. Standardmäßig wird ein zufälliger *seed* vom System gewählt.
- ⟨TOTAL-STATE-SPLIT-STEPS⟩ - Optionaler Parameter, der angibt, wie viele State-Splitting-Iterationen durchgeführt werden. Viele Iterationen können zu einer äußerst langen Laufzeit des Programms führen. Der Standardwert liegt hier bei 5 Iterationen.
- ⟨NUMBER OF STATES⟩ - Wert, der angibt, wie vielen Zuständen das Hidden-Markov-Modell initialisiert wird. Der Parameter ist optional und der Standardwert liegt bei 1.

State-Splitting

In diesem Modus wird mit Hilfe eines Korpus ein Hidden-Markov-Modell mit der angegebenen Anzahl an Zuständen erstellt und mit Hilfe des State-Splitting-Algorithmus trainiert. Die initiale Verteilung der Beobachtungswahrscheinlichkeiten richtet sich nach der Auftrittshäufigkeit der Wörter im Korpus. Die Übergangswahrscheinlichkeiten von einem Zustand in einen anderen werden zunächst zufällig festgelegt. Das trainierte Modell wird zuletzt in eine Ausgabedatei geschrieben.

Aufruf

```
$ java -jar HMM-SS.jar -statesplitting ⟨PARAMETERLISTE⟩
```

Die ⟨PARAMETERLISTE⟩ besteht hierbei aus:

[5] *er␣sagt␣:␣"␣hallo␣!␣"*, anstatt von *Er␣sagt:␣"Hallo!"*

⟨PARAMETERLISTE⟩

-output ⟨HMM-FILE⟩ ⟨WORDMAP-FILE⟩
-input ⟨KORPUS-FILE⟩
[-c ⟨TRAINING CYCLES⟩]
[-a ⟨ALPHA⟩]
[-e ⟨EPSILON⟩]
[-s ⟨SEED⟩]
[-t ⟨TOTAL-STATE-SPLIT-STEPS⟩]
[-n ⟨NUMBER OF STATES⟩]

Normales Training

Im normalen Modus wird zunächst ein Hidden-Markov-Modell mit der angegebenen Anzahl an Zuständen erstellt. Die Verteilung der Beobachtungswahrscheinlichkeiten und der Übergangswahrscheinlichkeiten werden genau, wie beim *State-Splitting-Modus* festgelegt.

Aufruf

$ java -jar HMM-SS.jar -normal ⟨PARAMETERLISTE⟩

Die ⟨PARAMETERLISTE⟩ besteht hierbei aus:

⟨PARAMETERLISTE⟩

-output ⟨HMM-FILE⟩ ⟨WORDMAP-FILE⟩
-input ⟨KORPUS-FILE⟩
[-c ⟨TRAINING CYCLES⟩]
[-a ⟨ALPHA⟩]
[-s ⟨SEED⟩]
[-n ⟨NUMBER OF STATES⟩]

Parsing

Im Parsing-Modus werden ein gegebenes Hidden-Markov-Modell geladen und danach Werte von der Standard-Eingabe gelesen. Nach jeder Eingabe wird die Wahrscheinlichkeit des Satzes, nach dem *Forward-Algorithmus* (siehe Abschnitt 4.1.1) berechnet und mittels der Standard-Ausgabe zurückgegeben. Dieser Vorgang läuft solange bis ein *End-Of-File* (EOF) Signal an der Standard-Eingabe gelesen wird. Dadurch eignet sich dieser Modus gut, um Textdateien, beispielsweise durch Pipelining, zu parsen.

Aufruf

$ java -jar HMM-SS.jar -parse ⟨PARAMETERLISTE⟩

Die ⟨PARAMETERLISTE⟩ besteht hierbei aus:

⟨PARAMETERLISTE⟩
-input ⟨HMM-FILE⟩ ⟨WORDMAP-FILE⟩

Perplexity

Im Perplexity-Modus werden Wahrscheinlichkeiten von der Standard-Eingabe gelesen und zunächst nur gespeichert. Erscheint ein *End-Of-File* (EOF) Signal, so wird mit Hilfe der eingegebenen Daten die *Perplexity* (siehe Abschnitt 5.1) berechnet. Das Ergebnis wird an die Standard-Ausgabe weitergeben. Dieses Methode der Ein- und Ausgabe eignet sich sehr gut, um beispielsweise einen im *Parsing-Modus* berechneten Text mit Hilfe einer Pipeline direkt zu bewerten.

Aufruf
$ java -jar HMM-SS.jar -perplexity

4.2.4. Integration in Vanda Studio

Die Implementierung wurde ebenfalls in Form eines *Packages* in Vanda-Studio integriert. Der Name des Packages lautet stateSplitting. Dabei sind zehn neue Tools hinzugekommen, die nachfolgend beschrieben werden.

1 **HMM parsing** - Nimmt als Eingabe einen *Satzkorpus*, ein *Hidden-Markov-Modell*, die entsprechende *Wordmap* und liefert als Ausgabe die *Wahrscheinlichkeit der einzelnen Sätze*.

2, 3 **HMM training** - Existiert zweimal. Als Eingabe reicht entweder nur der *Satzkorpus*, oder *alle Parameter des Programms* müssen übergeben werden. Wird nur das *HMM training* mit dem *Satzkorpus* als Eingang verwendet, werden die Standardparameter an das Programm übergeben. Als Ausgabe liefert das Tool ein *Hidden-Markov-Modell* und die zugehörige *Wordmap* für Modell und Korpus.

4, 5 **HMM training with custom seed** - Wie beim *HMM training* existieren hier ebenfalls zwei Varianten. Zusätzlich muss bei jeder ein *seed* (siehe Abschnitt 4.2.3) mit angegeben werden. So können Modelle unter gleichen Voraussetzungen trainiert werden.

6, 7 **HMM training with State-Splitting** - Auch hier gibt es zwei Varianten. Einmal muss nur der *Satzkorpus* und einmal müssen wieder *sämtliche Programmparameter* übergeben werden. Als Ausgabe wird ebenfalls ein *Hidden-Markov-Modell* und die zugehörige *Wordmap* geliefert.

8, 9 **HMM training with State-Splitting custom seed** - Wie bei *HMM training with State-Splitting*. Allerdings muss hier jeweils zusätzlich ein *seed* mit am Eingang angelegt werden.

10 **Perplexity** - Nimmt als Eingabe eine *Liste von Wahrscheinlichkeiten* und gibt die *Perplexity der Wahrscheinlichkeiten* zurück.

Beispiel:

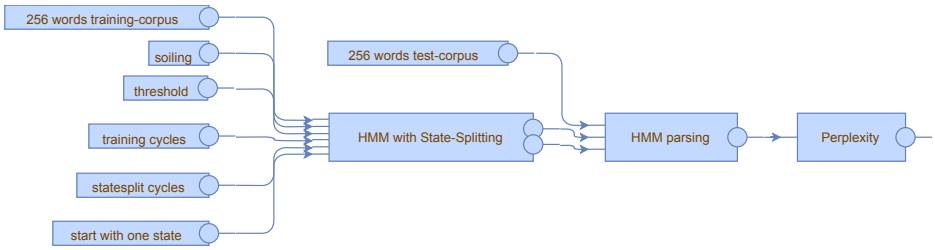

Abbildung 4.1.: Beispielworkflow in Vanda-Studio

5. Vergleich

Im Folgenden soll ein Vergleich durchgeführt werden, um festzustellen, ob Hidden-Markov-Modelle, die mit State-Splitting trainiert wurden, eine Sprache besser abbilden, als solche, die ohne State-Splitting trainiert wurden. Dafür muss zunächst ein Güte-Maß eingeführt werden, welches beschreibt, wann ein Hidden-Markov-Modell gut, oder schlecht ist. Danach wird mit Hilfe von Vanda-Studio eine Versuchsanordnung aufgebaut und mit Hilfe des Gütekriteriums ein Vergleich angestellt.

5.1. Perplexity

In dieser Arbeit wird als Güte-Maß das Konzept der *Perplexity* angewandt. Dies ist aus folgenden Gründen sinnvoll [JM09, S. 129]:

- Schnell und einfach zu berechnen
- Weit verbreitet
- Unabhängig von der letztendlichen Anwendung des Modells

Sie basiert auf der *Cross-Entropy* [Bro+92] und setzt sich folgendermaßen zusammen:

Seien $M = (Q, O, q_s, b, p)$ ein Hidden-Markov-Modell und $C = \{o_1, o_2, \dots o_n\} \subseteq O^*$ eine Menge von Sequenzen von Beobachtungen.

Die *Cross-Entropy (von C bezüglich M)*, notiert durch $\mathrm{H}_M(C)$, berechnet sich aus:

$$\begin{aligned}
\mathrm{H}_M(C) &= -\frac{1}{n} \cdot \log_2\big(\mathrm{P}(o_1) \cdot \mathrm{P}(o_2) \dots \mathrm{P}(o_n)\big) \\
&= -\frac{1}{n} \cdot \Big(\log\big(\mathrm{P}(o_1)\big) + \log_2\big(\mathrm{P}(o_2)\big) + \dots + \log_2\big(\mathrm{P}(o_n)\big)\Big) \\
&= -\Big(\frac{1}{n}\log_2\big(\mathrm{P}(o_1)\big) + \frac{1}{n}\log_2\big(\mathrm{P}(o_2)\big) + \dots + \frac{1}{n}\log_2\big(\mathrm{P}(o_n)\big)\Big) \\
&= -\sum_{i=1}^{n} \frac{1}{n}\log_2\big(\mathrm{P}(o_i)\big)
\end{aligned}$$

$\frac{1}{n}$ ist die durch C gegebene, empirische Wahrscheinlichkeit einer Beobachtung $o \in C$. Alle anderen Beobachtungen $(O^* \setminus C)$ haben die Wahrscheinlichkeit 0.

Die *Perplexity (von C bezüglich M)*, notiert durch $\mathrm{Perplexity}_M(C)$, berechnet sich aus [JM09, S. 150, 151]

$$\begin{aligned}
\mathrm{Perplexity}_M(C) &= 2^{\mathrm{H}_M(C)} \\
&= 2^{-\sum_{i=1}^{n} \frac{1}{n}\log_2\left(\mathrm{P}(o_i)\right)}
\end{aligned}$$

Je besser ein Modell eine Menge von Sequenzen von Beobachtungen C bewertet, desto geringer ist dessen *Cross-Entropy / Perplexity*. Es ist weniger „verwirrt“ von den Daten. In der Versuchsanordnung waren diese Mengen von Sequenzen von Beobachtungen die Testdaten des jeweiligen Textkorpus.

5.2. Versuchsanordnung

Im Folgenden soll beschrieben werden, welche Korpora für die Versuche verwendet und wie die einzelnen Versuche aufgebaut werden.

5.2.1. Wahl der Korpora

Als erstes muss eine Auswahl an verschiedenen Trainings- und Testdaten getroffen werden. Hierbei wird der deutschsprachige Teil der *Europarl-v7-Korpus-Sammlung* [Koe05; Die13] verwendet. Aus diesem monolingualen Korpus werden insgesamt vier Korpora erstellt, wobei jeweils zwei Korpora zum Trainieren und zwei zum Testen der Modelle verwendet werden:

Name	Vokabeln	Sätze	Wörter	Einsatzgebiet
256.train.de.txt	256	1 443	9 036	Training
256.test.de.txt	256	128	705	Test
2048.train.de.txt	2 048	67 687	732 702	Training
2048.train.de.txt	2 048	1 024	11 408	Test

5.2.2. Versuchsaufbau

Als nächstes werden die Versuche werden mit Hilfe von *Vanda Studio* durchgeführt. Dafür ist für jeden Versuch ein Workflow erstellt worden. Um die Ergebnisse wiederholbar zu machen, werden bei allen Versuchen festgelegte *Random-Seeds* verwendet (siehe Abschnitt 4.2.4).

Mit jedem Korpuspaar werden jeweils zwei Versuche durchgeführt: Zunächst wird ein Training eines normalen Hidden-Markov-Modells und einem, dass mit State-Splitting trainiert wird, unter genau denselben Voraussetzungen durchgeführt. Beide Modelle werden daraufhin auf ihre Perplexity hin untersucht. Danach werden fünf Hidden-Markov-Modelle mit unterschiedlichen Ausgangssituationen mit Hilfe des normalen Trainings verarbeitet und mit einem Modell vergleichen, dass mit State-Splitting trainiert wird.

1 zu 1

Bei diesem Versuch ist wichtig, dass beide Modelle dieselben oder zueinander passende Eingangsdaten haben.

Für das Hidden-Markov-Modell mit State-Splitting wird zunächst der Wert für die Merging-Schwelle (0,1) und die Anzahl der State-Splitting-Iterationen (6) festgelegt.

Die anderen Werte werden für das Training mit dem Koprus mit 256 Vokabeln wie folgt verteilt:

Modell	**Trainingsschritte**	**Verschmutzung**	**Seed**	**Zustände**
Normal	180	0,01	12 345	11
State-Splitting	15	0,01	12 345	1

Bei einem Hidden-Markov-Modell, das mit State-Splitting trainiert wurde, wird pro State-Splitting-Iteration zwei mal die Anzahl der Trainingsschritte durchgeführt. Dies geschieht jeweils nach dem Splitting und nach dem Merging (siehe Abschnitt 4.1.2). Damit das normale Hidden-Markov-Modell genauso viele Trainingsiterationen durchführen kann wie das Modell mit State-Splitting, müssen die Anzahl der Trainingsschritte wie folgt berechnet werden:

$$\begin{aligned}\text{Trainingsschritte} &= \text{Trainingsschritte State-Splitting} \cdot 2 \cdot \text{State-Splitting Iterationen} \\ 180 &= 15 \cdot 2 \cdot 6\end{aligned}$$

Die Anzahl der Zustände des Hidden-Markov-Modells mit normalem Training ergibt sich aus der Anzahl der Zustände des Modells mit State-Splitting am Ende dessen Trainings.

Beim Training mit dem Korpus mit 2048 Vokabeln wird das Ergebnis eindeutiger, wenn die Anzahl der einzelnen Trainingsschritte etwas erhöht wird. Daher werden für das Modell mit State-Splitting 20 und für das normale Hidden-Markov-Modell 240 Trainingsiterationen festgelegt. Die Anzahl der Zustände des normal trainierten Modells beträgt 14.

Wenn das Training beider Modelle durchgeführt wurde, kann jeweils der Testkorpus geparsed werden. Die Ergebnis-Wahrscheinlichkeiten der einzelnen Sätze werden zur Berechnung der Perplexity herangezogen.

Versuch:

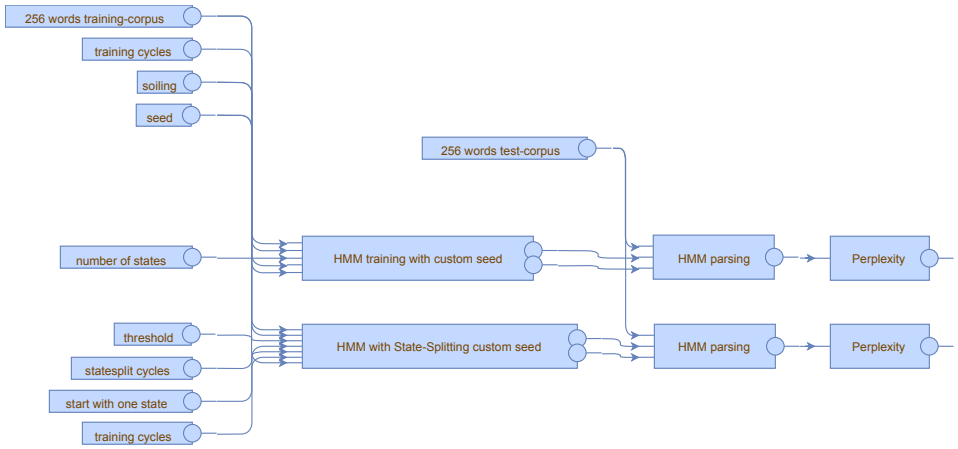

Abbildung 5.1.: Versuchsaufbau: Hidden-Markov-Modell ohne und mit State-Splitting (Korpus mit 256 Vokabeln)

5 zu 1

Bei diesem Versuch soll herausgefunden werden, wie gut ein Hidden-Markov-Modell mit State-Splitting im Vergleich zu verschieden konfigurierten Hidden-Markov-Modellen mit normalem Training abschneidet.

Für das Hidden-Markov-Modell mit State-Splitting werden zunächst der Wert für die Merging-Schwelle (0,1) und die Anzahl der State-Splitting-Iterationen (6) festgelegt. Die anderen Werte werden wie folgt verteilt:

Modell	Trainingsschritte	Verschmutzung	Seed	Zustände
Normal 1	400	0,01	3 735	12
Normal 2	70	0,01	7 860	43
Normal 3	256	0,01	1 563	20
Normal 4	150	0,01	127 624	28
Normal 5	200	0,01	96 345	2
Statesplitting	30	0,01	4 714 483	1

Wenn das Training der Modelle beendet ist, können jeweils der Testkorpus geparsed und die Ergebnisse zur Berechnung der Perplexity herangezogen werden.

Versuch:

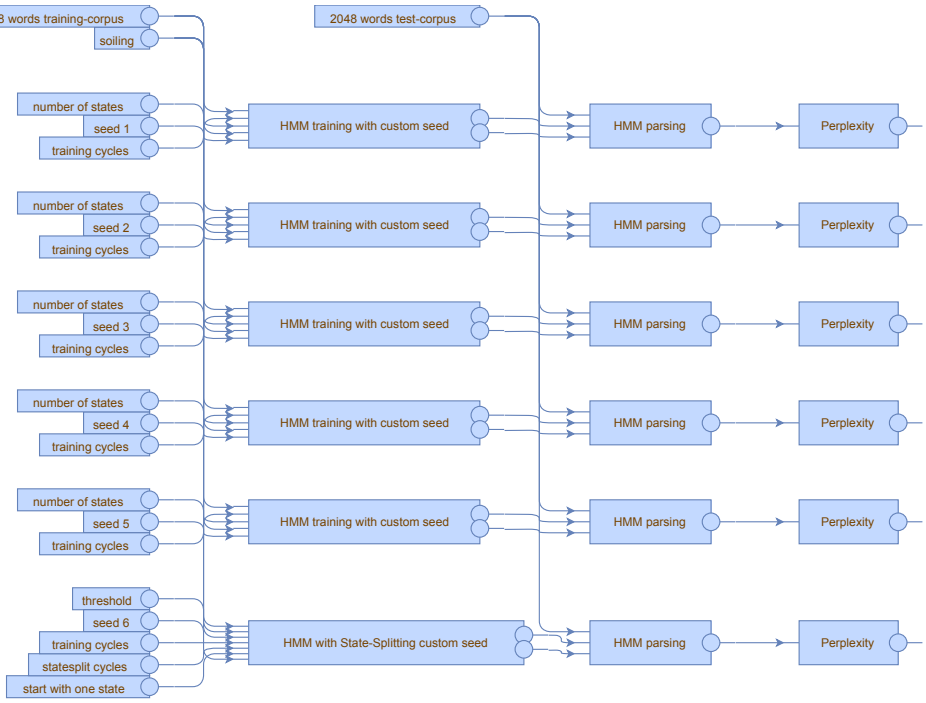

Abbildung 5.2.: Versuchsaufbau: Hidden-Markov-Modell ohne und mit State-Splitting (Korpus mit 2048 Vokabeln)

5.3. Auswertung

Nachdem die Workflows abgearbeitet worden sind, können die Trainingsdaten miteinander verglichen und ausgewertet werden. Dabei ist vor allem wichtig, wie sich die Perplexity der einzelnen Modelle verhält.

5.3.1. 1 zu 1

In folgender Tabelle sind die Perplexity-Werte der Modelle den beiden Korpora gegenübergestellt.

	Normal	**State-Splitting**
256 Wort Korpus	$8{,}29 \cdot 10^{8}$	$1{,}85 \cdot 10^{8}$
2048 Wort Korpus	$2{,}21 \cdot 10^{12}$	$7{,}13 \cdot 10^{6}$

Wie zu erkennen ist, sind die Perplexity-Werte beider mit State-Splitting trainierten Modelle erheblich niedriger, als die der normal trainierten Modelle. Das bedeutet, dass die mit State-Splitting trainierten Modelle in beiden Fällen die Sprache besser modelliert haben, als die normal trainierten. Aus dieser Stichprobe ist anzunehmen, dass ein Modell, welches mit State-Splitting trainiert wird besser sein kann, als ein Modell, welches ausschließlich mit dem Baum-Welch Algorithmus trainiert wurde. Das Ergebnis ist, wie weitere Tests gezeigt haben, allerdings sehr stark von der Wahl der Merging-Schwelle ε, sowie der Anzahl an Trainings- und State-Splitting-Iterationen abhängig.

5.3.2. 5 zu 1

Das nachfolgende Diagramm zeigt die Perplexity-Werte der einzelnen Modelle, die mit dem Korpus mit 256 Vokabeln trainiert wurden, im Vergleich.

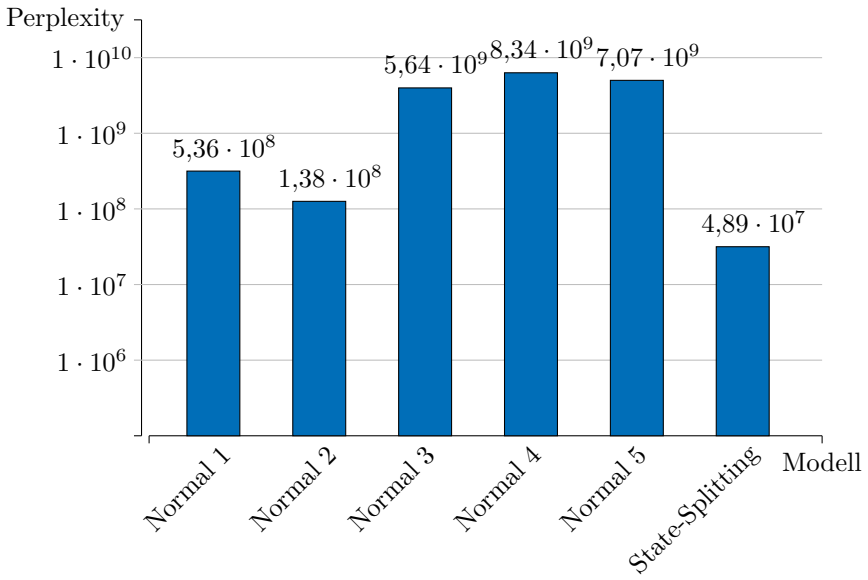

Abbildung 5.3.: Vergleich der Perplexity von fünf normal trainierten Hidden-Markov-Modellen und einem, welches mit State-Splitting trainiert wurde (Korpus mit 256 Vokabeln)

Auf der X-Achse des Balkendiagramms ist das jeweilige Modell und auf der Y-Achse die zugehörige Perplexity abgebildet. Außerdem ist die Y-Achse logarithmisch von $1 \cdot 10^{7}$ bis $1 \cdot 10^{10}$ skaliert. An der Spitze der einzelnen Balken sind die, auf zwei Nachkommastellen gerundeten, Perplexity-Werte des jeweiligen Modells abgebildet.

Wie eindeutig zu erkennen ist, schwanken die Werte der mit dem Baum-Welch-Algorithmus trainierten Hidden-Markov-Modelle zwischen $1{,}38 \cdot 10^{8}$ und $7{,}07 \cdot 10^{9}$.

Dies hängt mit den unterschiedlichen Anfangskonfigurationen und der vorgegebenen Anzahl an Zuständen zusammen. Dabei sind keine Zusammenhänge mit den Werten aus der Tabelle in Abschnitt 5.2.2 zu erkennen. Der Balken und somit die Perplexity des Modells, welches mit State-Splitting trainiert wurde, ist mit einem Wert von $4{,}89 \cdot 10^7$ am kleinsten. Dies bedeutet, dass das State-Splitting-Modell die Sprache am besten modelliert.

Aus dem Diagramm ist zu erkennen, dass unterschiedlich konfigurierte Hidden-Markov-Modelle, die nur mit dem Baum-Welch-Algorithmus trainiert worden sind, zum einen sehr unterschiedliche und zum anderen auch schlechtere Ergebnisse liefern, als ein Modell, welches mit State-Splitting trainiert worden ist.

Nimmt man die Informationen aus dem vorherigen Abschnitt noch hinzu, so ist aber auch zu erkennen, dass ein zu wenig trainiertes Hidden-Markov-Modell schlechter sein kann, als ein normal trainiertes. Das Modell aus dem vorigen Abschnitt hatte bei dem kleineren Korpus einen Perplexity-Wert von $1{,}85 \cdot 10^8$ wogegen das Modell *Normal 2* nur einen Wert von $1{,}38 \cdot 10^8$ hat. Dies könnte zwei Gründe haben: Das State-Splitting-Modell aus dem voherigen Abschnitt kann entweder zu wenig trainiert worden sein, oder die Mergeing-Schwelle wurde nicht angemessen gewählt.

Nachfolgend wird das Trainingsergebnis für den Korpus mit 2048 Vokabeln diskutiert.

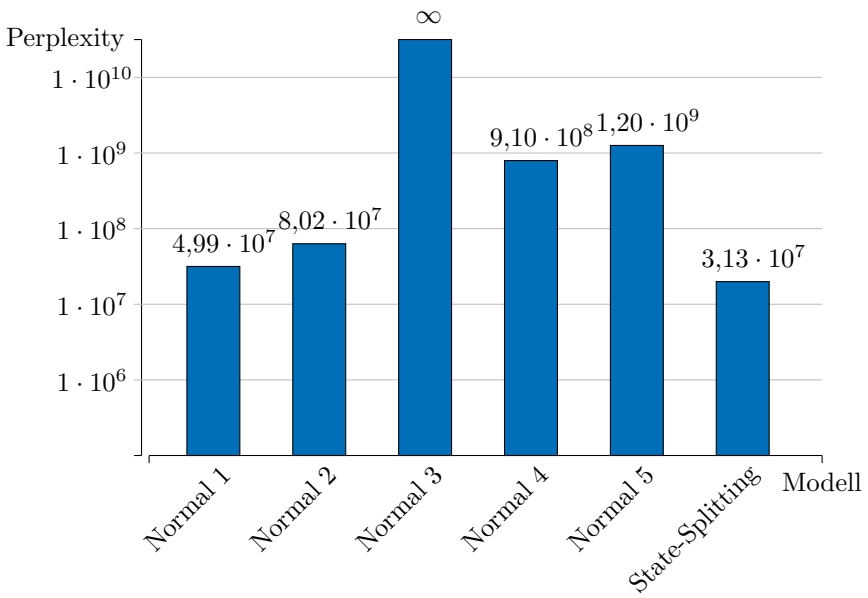

Abbildung 5.4.: Vergleich der Perplexity von fünf normal trainierten Hidden-Markov-Modellen und einem, welches mit State-Splitting trainiert wurde (Korpus mit 2048 Vokabeln)

Der Aufbau des Diagramms, ist derselbe wie oben: Die X-Achse des Balkendiagramms ist das jeweilige Modell und die Y-Achse zeigt die zugehörige Perplexity. Die Y-Achse ist ebenfalls logarithmisch von $1 \cdot 10^7$ bis $1 \cdot 10^10$ skaliert. An der Spitze der einzelnen Balken sind die, auf zwei Nachkommastellen gerundeten, Perplexity-Werte des jeweiligen Modells abgebildet.

Die Perplexity Werte der Modelle *Normal 1* bis *Normal 5* liegen im Bereich zwischen $4{,}99 \cdot 10^7$ und unendlich. Warum *Normal 3* solch einen Ausschlag nach oben zeigt, wird in einem gesonderten Abschnitt erläutert. Wie auch im vorherigen Diagramm sind keine Parallelen zu den gewählten Startkonfigurationen und der letztendlichen Perplexity erkennbar. Außerdem weist das Modell, welches mit State-Splitting trainiert wurde, erneut die geringste Perplexity auf, was den Schluss zulässt, dass es auch hier die vorliegende Sprache am besten modelliert.

In diesem Fall ist die Perplexity des Modells *Normal 1* sehr nah an der des mit State-Splitting trainierten Modells. Ein mögliche Ursache dafür könnte sein, dass das State-Splitting Modell nicht genügend trainiert wurde oder die Startkonfiguration für *Normal 1* sehr gut zum vorliegenden Korpus passt.

Auch in diesem Versuch war das Hidden-Markov-Modell, welches mit State-Splitting trainiert wurde, besser, als alle anderen. Wird aber wieder der Versuch in Abschnitt 5.3.1 zur Auswertung hinzugezogen, welches bei dem Korpus mit 2048 Vokabeln eine Perplexity von $7{,}13 \cdot 10^6$ hat, wird klar, dass auch ein Modell, welches mit State-Splitting trainiert wird immernoch äußerst stark von den gewählten Parametern abhängig ist.

Modell Normal 3 und seine Perplexity Aus Diagramm 5.4 ist ersichtlich, dass das Hidden-Markov-Modell *Normal 3* eine Perplexity von unendlich hat. Damit dieser Fall eintreten kann, muss die Liste von Wahrscheinlichkeiten, mit denen die Perlexity berechnet wird, mindestens einen Eintrag enthalten, der den Wert 0 hat. Denn nach 5.1 enthält dann die Cross-Entropy und demnach auch die Perplexity unter anderem den Summanden $\frac{1}{n} \log_2 0$. Da aber der Logarithmus von 0 nicht definiert ist, kommt ein undefiniertes Ergebnis heraus.

Um diese Vermutung zu bestätigen muss die Liste der Wahrscheinlichkeiten der Sätze, die von *Normal 3* geparsed wurden, analysiert werden. Nachfolgend steht einen Auszug dieser Liste.

Zeile	**Wahrscheinlichkeit**
...	...
273	$1.9191943450296038 \cdot 10^{-14}$
274	$1{,}762252931495319 \cdot 10^{-22}$
275	0,0
276	$7{,}405909118210191 \cdot 10^{-15}$
277	$1{,}6686342011260448 \cdot 10^{-14}$
...	...

Tatsächlich befindet sich in Zeile 275 eine Wahrscheinlichkeit mit dem Wert 0. Das bedeutet, dass Satz 275 im Testkorpus eigentlich nicht in der Sprache enthalten sein

darf. Bei genauerer Betrachtung der Wahrscheinlichkeiten der anderen Modelle fällt auf, dass auch diese den Satz nur gering bewerten. So erhält der Satz bei Modell *Normal 1* beispielsweise eine Wahrscheinlichkeit von nur $5{,}504283430790282 \cdot 10^{-136}$. Daher liegt die Vermutung nahe, dass *Normal 3* dem Satz 275 eine so geringe Wahrscheinlichkeit zugeordnet hat, dass durch Rundungsfehler des Rechners der Wert 0 gespeichert wurde. Der Satz als solcher erscheint wenig unrealistisch, er lautet: *„bei der heutigen abstimmung wurden von unseren vier änderungsanträgen zwei angenommen und zwei abgelehnt .“* [Die13][6]. Dies deutet darauf hin, dass der Korpus nicht ausreichend für ein angemessenes Training ist.

5.3.3. Fazit

Das Training von Hidden-Markov-Modellen mit State-Splitting kann Ergebnisse liefern, die eine Sprache wesentlich besser modellieren, als ein herkömmliches Training mit dem Baum-Welch-Algorithmus. Dennoch ist es sehr stark von der Anzahl der Trainingsiterationen, der State-Splitting-Iterationen und der Merging-Schwelle ε abhängig. Sind diese Werte jedoch gut gewählt, kann das State-Splitting schneller und auch besser Trainieren, als ein Hidden-Markov-Modell, welches mit der gleichen Anzahl an Zuständen nur mit Hilfe des herkömmlichen Trainings erstellt wird.

[6] Zeile 275, in der Datei „europarl-v7.de-en.tokenizer.lowercase.clean.nubbed.2048.test.de.txt“, komprimiert in der Datei „europarl-v7.de-en.tokenizer.lowercase.clean.nubbed.2048.txz“

6. Schluss

Im letzten Abschnitt dieser Bachelorarbeit wird ein Ausblick gegeben, wie weiter verfahren werden kann und was die Ergebnisse dieser Arbeit bedeuteten. Schließlich wird ein persönliches Fazit gezogen.

6.1. Ausblick

Die Arbeit und insbesondere der Vergleich der Hidden-Markov-Modelle hat gezeigt, dass ein Training mit Hilfe von State-Splitting durchaus in der Lage ist, selbige mit einem guten Ergebnis zu trainieren. Dabei ist jedoch nach wie vor sehr wichtig, wie der Merging-Parameter ε und die Anfangsverteilung des Modells gewählt werden. Es konnte somit nur eines der in Abschnitt 2.4 angesprochenen Probleme von Hidden-Markov-Modellen gelöst werden. Es bliebe nun noch zu ergründen, ob die Wahl des Parameters ε schwieriger ist, als die Wahl der Zustände und deren Wahrscheinlichkeitsverteilungen eines Hidden-Markov-Modells zu Beginn eines Trainings.

Auch das Vergleichen der beiden Möglichkeiten des Trainings kann weiter fortgeführt werden. In dieser Arbeit wurden nur einige stichprobenartige Tests durchgeführt. Sinnvoll wäre in jedem Fall ein empirischer Test, mit mehreren Korpora, verschiedenen State-Splitting-Trainings mit unterschiedlichen ε-Parametern und vielen Hidden-Markov-Modellen, die nur mit dem Baum-Welch-Algorithmus trainiert werden. Auch könnte versucht werden einen mathematisch-fundierten Beweis zu führen, welches Training besser ist.

Eine Möglichkeit, das Training mit State-Splitting noch weiter zu verbessern, wäre, wie bereits in Abschnitt 4.1.1 erwähnt, die Wahrscheinlichkeiten der Zustände bei ihrer Zusammenführung, je nachdem, wie oft sie genutzt werden, unterschiedlich zu gewichten. Dadurch würden beim Merging weniger Informationen über das vorherige Training mit dem Baum-Welch-Alogrithmus verloren gehen, was das Trainingsergebnis insgesamt verbessern kann.

Auch die Implementierung könnte noch weiter ausgearbeitet werden. So ist, wie sich im Laufe der Arbeit herausgestellt hat, das in Abschnitt 4.2.1 erwähnte, bereits implementierte Training des Baum-Welch-Algorithmus sehr speicherintensiv und könnte dahingehend möglicherweise noch optimiert werden.

6.2. Fazit

Nach dem Schreiben dieser Arbeit sind mir einige Punkte besonders im Gedächtnis geblieben: Einige Nachteile sind zum Beispiel, dass das Trainingsergebnis eines Hidden-

Markov-Moddels nach wie vor sehr stark von einer sinnvollen Belegung der Wahrscheinlichkeiten vor Beginn des Trainings abhängt. Außerdem muss zusätzlich der Merging-Parameter ε festgelegt werden, welcher das Ergebnis, die Größe des letztendlichen Modells und daher auch die benötigte Zeit und Rechenleistung stark beeinflusst. Vorteilhaft hingegen ist, dass das State-Splitting ein performantes Training eines Hidden-Markov-Modells bietet, ohne dass über die Anzahl der nötigen Zustände des Modells nachgedacht werden muss. Außerdem kann ein mit State-Splitting trainiertes Hidden-Markov-Modell ein deutlich besseres Ergebnis liefern, als herkömmlich trainierte Modelle mit denselben Voraussetzungen. Daher denke ich, dass das State-Splitting für Hidden-Markov-Modelle eine sehr effektive Erweiterung des bestehenden Trainingsverfahren bildet.

A. Beweise

Nachfolgend wurde versucht die beiden Lemmata aus Abschnitt 2.2.3 zu beweisen.

Lemma 1: Sei $M = (Q, O, q_s, p, b)$ ein Hidden-Markov-Modell, wobei für alle $q_1, q_2 \in Q : p(q_1 \mid q_2) > 0$. Dann ist P_M für Ableitungen eine Wahrscheinlichkeitsverteilung. Es gilt also:

$$\sum_{d \in D_M} \mathrm{P}_M(d) = 1.$$

Beweis: Der Beweis kann zunächst durch Umformung auf einen Beweis über Markov-Ketten zurückgeführt werden.

$$\text{Sei } P(q_1, \ldots, q_n) = p(q_1 \mid q_s) \cdot \left(\prod_{i=1}^{n-1} p(q_{i+1} \mid q_i) \right) \cdot p(q_s \mid q_n)$$

$$\begin{aligned}
&\sum_{d \in D_M} \mathrm{P}_M(d) \\
&= \sum_{\substack{\big((q_1,o_1),(q_2,o_2),\\ \ldots,(q_n,o_n)\big) \in D_M}} p(q_1 \mid q_s) \cdot \left(\prod_{i=1}^{n-1} p(q_{i+1} \mid q_i) \right) \cdot p(q_s \mid q_n) \cdot \left(\prod_{i=1}^{n} b(o_i \mid q_i) \right) && (*)^1 \\
&= \sum_{\substack{\big((q_1,o_1),(q_2,o_2),\\ \ldots,(q_n,o_n)\big) \in D_M}} P(q_1, \ldots, q_n) \cdot \left(\prod_{i=1}^{n} b(o_i \mid q_i) \right) && (*)^2 \\
&= \sum_{n \in \mathbb{N}} \sum_{q_1 \in Q \setminus \{q_s\}} \cdots \sum_{q_n \in Q \setminus \{q_s\}} \sum_{o_1 \in O} \cdots \sum_{o_n \in O} P(q_1, \ldots, q_n) \cdot \left(\prod_{i=1}^{n} b(o_i \mid q_i) \right) && (*)^3
\end{aligned}$$

$\sum_{q \in Q \setminus \{q_s\}}$ und $\sum_{o \in O}$ werden bei allen Umformungen nachfolgend mit $\sum_q$ und $\sum_o$ bezeichnet.

$(*)^1$ = Definition von $\mathrm{P}_M(d)$
$(*)^2$ = Einsetzen von $P(q_1, \ldots, q_n)$ zum Zwecke der Abkürzung
$(*)^3$ = Definition von D_M

$$= \sum_{n\in\mathbb{N}} \sum_{q_1} \cdots \sum_{q_n} \left[P(q_1, \ldots, q_n) \cdot \sum_{o_1} \cdots \sum_{o_n} \left(\prod_{i=1}^{n} b(o_i \mid q_i) \right) \right] \qquad (*)^4$$

$$= \sum_{n\in\mathbb{N}} \sum_{q_1} \cdots \sum_{q_n} \left[P(q_1, \ldots, q_n) \cdot \sum_{o_1} \cdots \sum_{o_n} \left(\prod_{i=1}^{n-1} b(o_i \mid q_i) \right) \cdot b(o_n \mid q_n) \right] \qquad (*)^4$$

$$= \sum_{n\in\mathbb{N}} \sum_{q_1} \cdots \sum_{q_n} \left[P(q_1, \ldots, q_n) \cdot \sum_{o_1} \cdots \sum_{o_{n-1}} \left(\prod_{i=1}^{n-1} b(o_i \mid q_i) \right) \cdot \sum_{o_n} b(o_n \mid q_n) \right] \qquad (*)^4$$

$$= \sum_{n\in\mathbb{N}} \sum_{q_1} \cdots \sum_{q_n} \left[P(q_1, \ldots, q_n) \cdot \prod_{i=1}^{n} \sum_{o_i} b(o_i \mid q_i) \right] \qquad (*)^5$$

$$= \sum_{n\in\mathbb{N}} \sum_{q_1} \cdots \sum_{q_n} \left[P(q_1, \ldots, q_n) \cdot \prod_{i=1}^{n} 1 \right] \qquad (*)^6$$

$$= \sum_{n\in\mathbb{N}} \sum_{q_1\in Q\setminus\{q_s\}} \cdots \sum_{q_n\in Q\setminus\{q_s\}} p(q_1 \mid q_s) \cdot \left(\prod_{i=1}^{n-1} p(q_{i+1} \mid q_i) \right) \cdot p(q_s \mid q_n) \qquad (*)^7$$

Es stellt sich heraus, dass der Rest der Formel unabhängig von den Beobachtungswahrscheinlichkeiten ist. Daher handelt es sich um eine Folge von Übergangswahrscheinlichkeiten von Zuständen, weshalb diese Formel ein einfaches Markov-Modell beschreibt [Fer70, S. 12–20].

In der verbleibenden Formel werden die Wahrscheinlichkeiten, aller Zustandsfolgen die in q_s beginnen und den Zustand ausschließlich am Ende wieder betreten, aufsummiert. Die Formel gibt also die Wahrscheinlichkeit an, dass eine Folge von Zuständen, die in q_s beginnt, nach endlich vielen Schritten dort wieder endet. Ist diese Wahrscheinlichkeit, wie gefordert, 1, so wird q_s *reccurent* genannt [Fer70, S. 63–72].

In Gallager [Gal13, S. 171] wird beschrieben, dass (bei einer endlichen Zustandsmenge) ein Zustand q genau dann recurrent ist, wenn jeder Zustand, der von q erreicht werden kann, wieder q erreichen kann. Das bedeutet in diesem Fall, dass jeder von q_s mit einer positiven Wahrscheinlichkeit erreichbare Zustand q_s wieder mit einer positiven Wahrscheinlichkeit erreichen kann, wenn q_s recurrent ist. Diese Eigenschaft ist trivialerweise erfüllt, da gefordert ist, dass alle Übergangswahrscheinlichkeiten positiv sind. □

Lemma 2: Sei $M = (Q, O, q_\mathrm{s}, p, b)$ ein Hidden-Markov-Modell, wobei für alle $q_1, q_2 \in Q : p(q_1 \mid q_2) > 0$. Dann ist P_M für Beobachtungen eine Wahrscheinlichkeitsverteilung. Es gilt also:

$$\sum_{o\in O^*} \mathrm{P}_M(o) = 1.$$

$(*)^4$ = Assoziativität und Kommutativität von Summen und Produkten
$(*)^5$ = Vorherigen Schritt $(n-1)$-Mal wiederholen
$(*)^6$ = Definition von b
$(*)^7$ = Einsetzen von $P(q_1, \ldots, q_n)$

Beweis: Der Beweis kann mittels Einsetzen von Definitionen und durch Umformungen geführt werden.

$$
\begin{aligned}
& \sum_{o \in O^*} \mathrm{P}_M(o) \\
= & \sum_{o \in O^*} \sum_{d \in \mathrm{yield}^{-1}(o)} \mathrm{P}_M(d) && (*)^1 \\
= & \sum_{\substack{o \in O^*, \\ d \in \mathrm{yield}^{-1}(o)}} \mathrm{P}_M(d) && (*)^2 \\
= & \sum_{\substack{o \in O^*, \\ d \in D_M, \mathrm{yield}(d) = o}} \mathrm{P}_M(d) && (*)^3 \\
= & \sum_{d \in D_M} \mathrm{P}_M(d) && (*)^4 \\
= & 1 && (*)^5
\end{aligned}
$$

□

$(*)^1$ = Definition von $\mathrm{P}_M(o)$
$(*)^2$ = Assoziativität und Kommutativität von Summen
$(*)^3$ = Definition von $\mathrm{yield}^{-1}(o)$
$(*)^4$ = Da zu sämtlichen Sequenzen von Beobachtungen o die Ableitungen ermittelt werden, ist es nur nötig die Menge aller Ableitungen D_M zu betrachten.
$(*)^5$ = $\mathrm{P}_M(d)$ ist eine Wahrscheinlichkeitsverteilung (siehe Lemma 1)

B. Literaturverzeichnis

[Bau72] Leonard E. Baum. „An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes“. In: *Inequalities III: Proceedings of the Third Symposium on Inequalities*. Hrsg. von Oved Shisha. University of California, Los Angeles: Academic Press, 1972, S. 1–8.

[Bro+92] Peter F. Brown u. a. „An Estimate of an Upper Bound for the Entropy of English“. In: *Computational Linguistics*. Cambridge, MA, USA: MIT Press, 1992, S. 31–40.

[Die13] Toni Dietze. *TUD - Grundlagen der Programmierung - Machine Translation Lab*. 15. Jan. 2013. URL: `http://www.inf.tu-dresden.de/index.php?node_id=3230&ln=de` (besucht am 24. 06. 2012).

[Fer70] F. Ferschl. „Markovketten“. In: *Lecture Notes in Operations Research and Mathematical Systems*. 35. Universität Bonn, Deutschland: Springer-Verlag Berlin, 1970.

[Gal13] Robert G. Gallager. „4. Finite state Markov chains“. In: *Draft of Stochastic Processes: Theory for Applications*. MIT, Massachusetts, USA, 2013, S. 167–222. URL: `http://www.rle.mit.edu/rgallager/notes.htm` (besucht am 27. 06. 2013).

[Ini] Open Source Initiative. *The BSD 3-Clause License — Open Source Initiative*. URL: `http://opensource.org/licenses/BSD-3-Clause` (besucht am 19. 06. 2012).

[JM09] Daniel Jurafsky und James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Second Edition. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2009. ISBN: 978-0-13-504196-3.

[jmf09] jm.francois. *jahmm: An implementation of Hidden Markov Models in Java*. Feb. 2009. URL: `https://code.google.com/p/jahmm/` (besucht am 19. 06. 2012).

[Koe05] Philipp Koehn. „Europarl: A Parallel Corpus for Statistical Machine Translation“. In: University of Edinburgh, Scotland, 2005. URL: `http://www.statmt.org/europarl/` (besucht am 19. 06. 2012).

[Koh13] Joerg Kohne. *GNU General Public License (GPL), Version 2.0 - GNU-Projekt - Free Software Foundation*. 17. März 2013. URL: `http://www.gnu.org/licenses/old-licenses/gpl-2.0` (besucht am 19. 06. 2012).

[Ora09] Oracle. *Javadoc Tool Home Page*. 11. Feb. 2009. URL: http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html (besucht am 22. 06. 2012).

[Pet+06] Slav Petrov u. a. „Learning accurate, compact, and interpretable tree annotation“. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. ACL-44. Sydney, Australia: Association for Computational Linguistics, 2006, S. 433–440. DOI: 10.3115/1220175.1220230.

[PPK07] Slav Petrov, Adam Pauls und Dan Klein. „Learning Structured Models for Phone Recognition“. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007, S. 897–905. URL: http://www.aclweb.org/anthology/D/D07/D07-1094.