Le temps, l'espace et l'évolutif en sciences du traitement de l'information, Éditions Cépaduès, H. Prade, R. Jeansoulin et C. Garbay éditeurs, p. 407-421, 2000.

Modèles de Markov cachés et apprentissage de séquences

Laurent Bréhélin Olivier Gascuel

Département d'Informatique Fondamentale et Applications, LIRMM,
161 rue Ada, 34392 Montpellier Cedex 5, France
{brehelin,gascuel}@lirmm.fr

Résumé

Les modèles de Markov cachés, introduits dans les années 1960, ont connus un succès important en reconnaissance de la parole où ils se sont imposés comme un des modèles de référence. Ils ont ensuite été appliqués avec le même succès à des domaines d'applications aussi divers que le traitement de textes manuscrits ou l'analyse de séquences biologiques, et sont depuis peu utilisés dans des applications d'extraction d'information à partir de données textuelles. Nous présentons ici une vue d'ensemble de ce modèle, de ses applications et de ses principaux algorithmes d'apprentissage et d'exploitation.

1 Introduction

Les modèles de Markov cachés (Hidden Markov Models ou HMMs) ont été introduits par Baum et ses collaborateurs dans les années 1960-70s [3]. Ce modèle est fortement apparenté aux automates probabilistes (PAs) [7]. Un automate probabiliste est défini par une structure composée d'états et de transitions, et par un ensemble de distributions de probabilité sur les transitions. À chaque transition est associée un symbole d'un alphabet fini. Ce symbole est générée à chaque fois que la transition est empruntée. Un HMM se définit également par une structure composée d'états et de transitions et par un ensemble de distributions de probabilité sur les transitions. La différence essentielle avec les PAs est que la génération de symboles s'effectue sur les états, et non sur les transitions. De plus, on associe à chaque état non pas un symbole, mais une distribution de probabilité sur les symboles de l'alphabet. Une illustration de la forte parenté entre HMMs et PAs peut être trouvée dans la propriété que n'importe quel HMM peut être simulé par un PA du même nombre d'états [1]. Notons que la propriété inverse n'est pas vraie.

Les HMMs sont utilisés pour modéliser des séquences d'observations. Ces observations peuvent être de nature discrète (par exemples les caractères d'un alphabet fini) ou continue (la fréquence d'un signal, une température, ...). Le premier domaine auquel les HMMs ont été appliqués est le traitement de la parole au début des années 1970 [2], [20]. Dans ce domaine, les HMMs se sont rapidement imposés comme le modèle de référence et une majeure partie des techniques d'utilisation et d'implémentation des HMMs (choix d'une structure ou d'un type de structure, problèmes d'*overfitting*, ...) on été développés dans le cadre de ces applications. Ces techniques furent ensuite appliquées et adaptées avec succès aux problèmes de la reconnaissance de textes manuscrits [15], [22]) et d'analyse de séquences biologiques [12], [9]. Les HMMs ont également été appliqués à d'autres domaines comme la reconnaissance d'images [19], la modélisation d'un signal musical [21] ou la génération de séquences de test pour circuits microélectroniques [5]. Depuis très récemment, avec l'augmentation importante de la masse des données électroniques, une nouvelle application prometteuse semble être l'extraction d'informations à partir de données textuelles [23], [10].

Notre objectif ici est de présenter une vue d'ensemble de la théorie des HMMs. Les algorithmes et les problèmes classiques de la littérature sont exposés et illustrés autant que possible par des applications réelles. Le chapitre 2 présente de manière formelle les modèles de Markov cachés et les deux principaux type de problèmes qu'ils peuvent résoudre : la reconnaissance et la segmentation. Aux chapitres 3 et 4 nous exposons les algorithmes classiques associés à la résolution de ces deux types de problème : algorithme de calcul de la probabilité d'une séquence pour les applications de reconnaissance et algorithme de calcul du chemin le plus probable pour les applications de segmentation. Quelle que soit l'application traitée, un des principaux problèmes que l'on rencontre lorsqu'on utilise des HMMs est la construction d'un modèle adapté aux séquences que l'on souhaite modéliser. La construction peut s'effectuer "à la main" à partir de certaines connaissances du domaine permettant d'inférer naturellement le ou les HMMs désirés, ou à l'aide d'un algorithme d'apprentissage à partir d'un ensembles de séquences d'apprentissage. Le chapitre 5 est dédié à ce problème. Nous verrons les différents cas de figure qu'il présente, dépendant des séquences à modéliser et des connaissances *a priori* dont on dispose.

2 Présentation et utilisations des HMMs

Un HMM H peut être défini[1] par un quadruplet $\langle \mathcal{S}, \Sigma, T, G \rangle$ où :

- $\mathcal{S}$ un ensemble de N états ; $\mathcal{S}$ contient deux états spéciaux muets *start* et *end* qui servent respectivement à débuter et conclure une séquence.
- Σ un alphabet de M symboles ;
- $T = \mathcal{S} - \{end\} \times \mathcal{S} - \{start\} \rightarrow [0,1]$, une matrice indiquant les probabilités de transition d'un état à l'autre ; on note $P(s \rightarrow s')$ la probabilité de transition de l'état s vers l'état s' ;
- $G = \mathcal{S} - \{start, end\} \times \Sigma \rightarrow [0,1]$, une matrice indiquant les probabilités de génération associées aux états ; on note $P(o|s)$ la probabilité de générer le symbole o à partir de l'état s.

Nous nous limiterons ici à la modélisation de séquences de symboles appartenant à un alphabet fini : un ensemble de *phonèmes* dans le cadre d'applications de reconnaissance de la parole, les quatre acides nucléiques (adénine, guanine, cytosine et timine) pour la modélisation de séquences d'ADN, l'ensemble des acides aminés pour les séquences protéiques, les mots d'un certain corpus pour l'extraction d'information à partir de données textuelles, ... Néanmoins les HMMs peuvent également être utilisés pour la modélisation de signaux continus [20]. On définit la structure d'un HMM par l'ensemble de ses états et par les transitions de probabilité non nulle. La Figure 1 représente un exemple de HMM avec sept états et onze transitions.

La procédure de génération d'une séquence de symboles à l'aide d'un HMM consiste à partir de l'état *start*, se déplacer d'état en état suivant les probabilités de transition, et générer un symbole sur chaque état rencontré en utilisant la distribution de probabilité de génération associée à l'état. Lorsqu'un symbole a été généré, on choisit une transition sortante suivant la distribution de probabilité de transition associée à l'état courant, et la procédure est réitérée jusqu'à atteindre l'état *end*. Par exemple, la séquence *abccb* peut être générée par le HMM de la Figure 1 en partant de l'état *start*, puis en allant sur l'état 1 puis sur l'état 3, puis 5, 5, 5, 2 et *end*. Notons que cette séquence de symboles peut également être générée en suivant la séquence d'état $start - 1 - 4 - 5 - 5 - 5 - 2 - end$, ou $start - 2 - 4 - 5 - 5 - 5 - 2 - end$. La probabilité de génération suivant la première séquence d'états – ou *chemin* – est la probabilité de passer de l'état *start* à l'état 1 (0.5), multiplié par la probabilité que l'état 1 émette le symbole a (1), multiplié par la probabilité de transition de 1 vers 3 (0.7), multiplié par ... $=6.6 \cdot 10^{-3}$. Suivant les deuxièmes et troisièmes chemins, cette probabilité est respectivement $2.3 \cdot 10^{-4}$ et $7.5 \cdot 10^{-5}$. Comme il n'y a pas d'autres chemins qui permettent de générer cette séquence, la probabilité de génération de *abccb* par le HMM est $6.6 \cdot 10^{-3} + 2.3 \cdot 10^{-4} + 7.5 \cdot 10^{-5} = 6.9 \cdot 10^{-3}$. Les HMMs définissent donc un processus

[1] La définition proposée ici n'est pas la définition originale donnée par Baum [3] dont elle diffère par l'introduction des états muets *start* et *end*. L'état *start* n'entraîne pas de modification essentielle du modèle et remplace avantageusement l'utilisation de la distribution de probabilité *a priori* associée aux états pour débuter les séquences dans la définition de Baum. L'état *end*, par contre, n'a pas d'équivalent dans la définition originale. Néanmoins, la définition présentée ici est celle habituellement utilisée dans la plupart des applications.

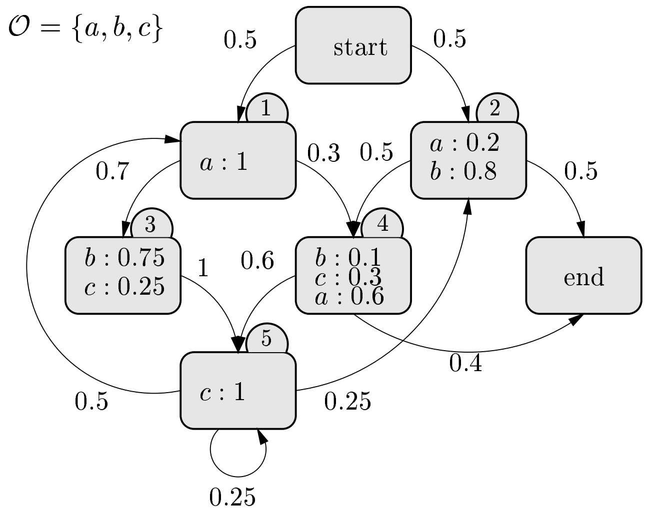

Fig. 1: Un exemple de HMM.

stochastique non déterministe – une même séquence de symboles peut être générée de plusieurs manières différents –, ce qui explique le nom donné à ce modèle : le processus de génération est un processus *markovien* – la probabilité de transition vers un état ne dépend que de l'état actuel et non des états rencontrés précédemment – qui est *caché* car il est impossible de connaître le processus suivi pour la génération d'un séquence de symboles donnée.

On peut classer les principales applications des HMMs en deux catégories. La première traite des problèmes de reconnaissance ou de classification : reconnaissance d'un mot parmi un ensemble de mots possibles à partir d'un signal audio (reconnaissance de la parole) [20] ou d'une écriture manuscrite [15], reconnaissance d'une famille de protéines à partir d'une séquence d'acides aminés [12] ... Pour ces applications, on construit généralement un HMM différent pour chaque classe à reconnaître – par exemple un HMM pour chaque mot dans le cadre de la reconnaissance de la parole. La reconnaissance d'une séquence consiste alors à calculer la probabilité de génération de la séquence par chacun des HMM, et à assigner à cette séquence sa classe la plus probable. Ce type d'application nécessite donc un algorithme efficace pour le calcul de la probabilité de génération d'une séquence. Cet algorithme est présenté à la section 3.

Le second type d'application des HMMs a trait aux problèmes de segmentation de séquences, c'est-à-dire au découpage d'une séquence en sous-séquences de différents types : découpage d'un signal musical en notes [21], localisation de régions codantes/non-codantes dans une chaîne de nucléotides [14], extraction de phrases clef d'un texte parmi un ensemble important de phrases à faible niveau informationnel [10], découpage d'un article en différents champs (titre, date, nom de l'auteur, ...) [23] ... On utilise pour ces applications un HMM dont les états sont *typés*. La procédure de segmentation d'une séquence de symboles consiste alors à calculer à l'intérieur du HMM le chemin qui a la probabilité maximale de générer cette séquence. On associe ensuite à chaque symbole de la séquence son type, en fonction du type de l'état correspondant dans le chemin calculé. Considérons par exemple le HMM de la Figure 2. En plus des états *start* et *end* ce HMM est composé de quatre autre états typés Article, Adjectif, Nom et Verbe. Remarquons que deux mots apparaissent dans plusieurs états : "modèle", qui peut être utilisé comme adjectif et comme nom, et "classe" qui peut être utilisé comme nom et comme verbe. Considérons maintenant la phrase "Le modèle classe la séquence.". Notre objectif est de segmenter cette phrase, c'est-à-dire d'assigner à chaque mot son type. Deux chemins du HMM permettent de générer cette phrase : *start*-Article-Adjectif-Nom-Article-Nom-*end* et *start*-Article-Nom-Verbe-Article-Nom-*end*. La probabilité de génération suivant le premier chemin est $1.8 \cdot 10^{-7}$, tandis qu'elle est de $6.22 \cdot 10^{-6}$ suivant le second chemin. La segmentation la plus probable est

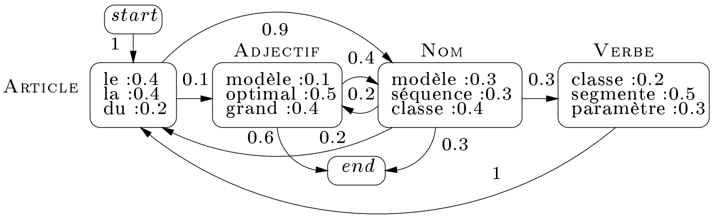

FIG. 2: Un HMM dont les états sont typés (ARTICLE, ADJECTIF, NOM, et VERBE) et qui peut être utilisé pour la segmentation de phrases.

donc celle induite par le second chemin. Cette segmentation nous permet alors par exemple de conclure que "classe" est utilisée dans cette phrase comme verbe et non comme nom, ce qui n'est *a priori* pas évident – toute considération sémantique mise à part – si l'on considère que ce mot est généralement plus souvent utilisé comme nom que comme verbe. Si l'on étudie de plus près les paramètres du HMM, on se rend compte que ce qui nous a permis d'arriver à cette conclusion est que, si ce n'était pas le cas (premier chemin), alors "modèle" serait un adjectif. Or un adjectif a peu de chance de suivre immédiatement un article (probabilité 0.1) et "modèle" n'est en définitive pas un adjectif très usité (probabilité 0.1). Ces paramètres dépendent bien évidemment du corpus de phrases étudié et le HMM, pour être efficace, doit donc refléter le plus fidèlement possible les caractéristiques de ce corpus. C'est tout le problème du paramétrage abordé à la Section 5. Les applications de segmentation reposent donc sur une recherche du chemin optimal et nécessitent un algorithme efficace pour résoudre ce problème. Cet algorithme est présenté à la Section 4.

3 Algorithme forward-backward

Si l'on veut calculer la probabilité de générer la séquence de symboles $O = o_1 \ldots o_T$ à l'aide du HMM $H = \langle \mathcal{S}, \Sigma, T, G \rangle$, l'approche directe consiste à calculer, comme on l'a fait dans la section précédente, la probabilité de génération pour chaque chemin possible et faire la somme de ces probabilités. La probabilité de générer O suivant le chemin $S = s_0 s_1 \ldots s_T s_{T+1}$ avec $s_0 = start$ et $s_{T+1} = end$ est

$$P(O|S) = P(s_0 \to s_1)P(o_1|s_1)\ldots P(s_{T-1} \to s_T)P(o_T|s_T)P(s_T \to s_{T+1}). \quad (1)$$

La probabilité de générer O avec H obtenue en sommant sur l'ensemble des séquences d'états possibles est donc

$$P(O|H) = \sum_{\langle s_0 \cdots s_{T+1} \rangle} P(s_0 \to s_1)P(o_1|s_1)\ldots P(s_{T-1} \to s_T)P(o_T|s_T)P(s_T \to s_{T+1}), \quad (2)$$

avec $\langle s_0 \cdots s_{T+1} \rangle$ l'ensemble des séquences d'états possibles. Si N est le nombre d'états du HMM, alors le nombre de chemins possibles pour générer une séquence de symboles de longueur T est de l'ordre de N^T. Comme, pour chaque chemin, le calcul de la Formule (1) demande de l'ordre de T opérations, le calcul de la probabilité de génération de O par H suivant l'équation (2) est en $O(TN^T)$. Une rapide application numérique nous montre que même pour des valeurs de T et N peu élevées, l'approche directe n'est clairement pas acceptable : avec $N = 5$ et $T = 100$ le calcul de $P(O|H)$ demande approximativement 10^{72} opérations. Heureusement une procédure bien plus efficace existe pour réaliser ce calcul : l'algorithme forward-backward [20].

Considérons la variable forward $\alpha_t(s)$ définie par

$$\alpha_t(s) = P(o_1 o_2 \cdots o_t, s_t = s|H),$$

qui exprime la probabilité d'avoir généré la séquence $o_1 \ldots o_t$ en partant de l'état *start* et d'être arrivé sur l'état s à l'instant t. Cette variable peut être calculée de manière inductive :

1. Initialisation. À $t = 1$ on a :

$$\alpha_1(s) = P(start \to s) \cdot P(o_1|s).$$

2. Induction. Pour $t = 2, \ldots, T$ on a :

$$\alpha_t(s) = \left(\sum_{s' \in \mathcal{S}} \alpha_{t-1}(s') P(s' \to s) \right) P(o_t|s). \tag{3}$$

Or, connaissant $\alpha_T(s)$ – c'est à dire la probabilité d'avoir généré la séquence O et d'être arrivé sur l'état s – pour tout $s \in \mathcal{S}$, le calcul de $P(O|H)$ est immédiat :

$$P(O|H) = \sum_{s \in \mathcal{S}} \alpha_T(s) P(s \to end). \tag{4}$$

Examinons la complexité de cet algorithme. La phase d'initialisation requiert une opération pour chaque état du HMM, donc au total $O(N)$ opérations. Pour la phase d'induction, pour chaque instant et chaque état, on réalise $O(N)$ opérations. Sommé sur l'ensemble des états et la totalité des instants, la phase d'induction requiert donc $O(N^2T)$ opérations. Une fois les $\alpha_T(s)$ calculés, le calcul de $P(O|H)$ suivant la Formule (4) demande une opération pour chaque état, donc au total $O(N)$ opérations. Le calcul de $P(O|H)$ à l'aide de l'algorithme forward ne requiert donc au total que $O(N^2T)$ opérations. En reprenant la même application numérique que précédemment ($N = 5$ et $T = 100$), le calcul de $P(O|H)$ suivant cet algorithme nécessite approximativement 3000 opérations.

Cet algorithme est appelé forward car l'induction est réalisée en avant : on calcule tout d'abord la probabilité de générer le premier symbole de la séquence, puis à chaque étape de l'induction on rajoute un symbole et on réitère la procédure jusqu'à ce qu'on ait calculé la probabilité de génération de la séquence entière. Bien que moins naturel, un algorithme similaire, l'algorithme backward, peut être utilisé pour réaliser ce calcul à l'envers. On utilise alors la variable backward $\beta_t(s)$:

$$\beta_t(s) = P(o_{t+1} o_{t+2} \cdots o_T | s_t = s, H),$$

qui exprime la probabilité de générer la séquence $o_{t+1} o_{t+2} \ldots o_T$ en partant de l'état s et d'arriver sur l'état *end*. La procédure d'induction est alors la suivante :

1. Initialisation. À $t = T$ on a :

$$\beta_T(s) = P(s \to end).$$

2. Induction. Pour $t = T - 1, \ldots, 1$ on a :

$$\beta_t(s) = \left(\sum_{s' \in \mathcal{S}} P(s \to s') P(o_{t+1}|s') \beta_{t+1}(s') \right).$$

Et, connaissant $\beta_1(s)$ – c'est à dire la probabilité de générer la séquence O en partant de l'état s et d'arriver sur l'état *end* – pour chaque état s, le calcul de $P(O|H)$ peut alors être réalisé suivant la formule

$$P(O|H) = \sum_{s \in \mathcal{S}} P(start \to s) \beta_1(s). \tag{5}$$

La complexité de l'algorithme backward est, comme pour l'algorithme forward, $O(N^2T)$.

4 Algorithme de Viterbi

Le problème que l'on se pose ici est de trouver, étant donnés une séquence de symboles $O = o_1 \dots o_T$ et un HMM $H = \langle \mathcal{S}, \Sigma, T, G \rangle$, la séquence d'états du HMM qui a la probabilité maximale de générer O. Ce qui nous préoccupe n'est pas la valeur de la probabilité maximale mais le chemin – appelé *chemin de Viterbi* – qui permet de générer la séquence O avec cette probabilité. De manière similaire à l'approche utilisée pour le calcul de $P(O|H)$, l'approche directe pour résoudre ce problème consiste à calculer la probabilité de génération suivant tous les chemins possibles et de choisir parmi ces chemins celui qui a la probabilité la plus élevée. Cette approche a, comme pour le calcul de $P(O|H)$, une complexité en $O(TN^T)$ et est donc également inapplicable. L'algorithme de Viterbi [20] est un algorithme de programmation dynamique très similaire à l'algorithme forward et qui permet de résoudre efficacement ce problème.

Considérons la variable $\delta_t(s)$ défini par

$$\delta_t(s) = \max_{s_0 \cdots s_{t-1}} P(s_0 \cdots s_t = s, o_1 \cdots o_t | H),$$

et qui exprime la probabilité maximale de générer la séquence $o_1 \cdots o_t$ suivant un unique chemin partant de l'état *start* et arrivant sur l'état s à l'instant t. De la même manière que pour $\alpha_t(s)$, cette variable peut être calculée de manière inductive :

1. Initialisation. À $t = 1$ on a :
$$\delta_1(s) = P(start \to s_1) P(o_1|s_1).$$
2. Induction. Pour $t = 2, \dots, T$ on a :
$$\delta_t(s) = \max_{s' \in \mathcal{S}} \left(\delta_{t-1}(s') P(s' \to s)\right) P(o_t|s). \tag{6}$$

Alors, connaissant $\delta_T(s)$ pour tous les états $s \in \mathcal{S}$, on peut calculer la probabilité maximale de générer O avec H suivant un simple chemin (on désignera cette probabilité par $P(O|H, V)$) en appliquant la formule :

$$P(O|H, V) = \max_{s \in \mathcal{S}} \left(\delta_T(s) P(s \to end)\right).$$

Malheureusement, ce n'est pas la valeur de cette probabilité qui nous intéresse mais réellement le chemin qui permet de générer O avec cette probabilité. On doit donc, à chaque étape t de l'induction et pour chaque état s, mémoriser l'état s' qui maximise l'Équation (6). Désignons par $\psi_t(s)$ cet état. Alors :

1. À $t = 1$ on a :
$$\psi_1(s) = start.$$
2. Pour $t = 2, \dots, T$ on a :
$$\psi_t(s) = \operatorname*{argmax}_{s' \in \mathcal{S}} \left(\delta_{t-1}(s') P(s' \to s)\right).$$

Une fois les variables $\delta_t(s)$ et $\psi_t(s)$ calculées pour chaque étape de l'induction et pour chaque état, il ne reste plus qu'à lancer une procédure inductive de rétro-propagation pour "dérouler" le chemin de Viterbi $start, s_1^*, \dots, s_T^*, end$ en partant de l'état end :

1. Initialisation. À $t = T$:
$$s_T^* = \operatorname*{argmax}_{s \in \mathcal{S}} \left(\delta_T(s) P(s \to end)\right).$$
2. Rétro-propagation. Pour $t = T - 1, T - 2, \dots, 0$:
$$s_t^* = \psi_t(s_{t+1}^*).$$

On notera que, mise à part la phase de rétro-propagation, l'algorithme de Viterbi est très similaire à l'algorithme forward. La principale différence résulte de la maximisation des probabilités attachées aux états précédents (Formule (6)) au lieu du calcul de la somme de ces probabilités (Formule (3)) dans le cas de l'algorithme forward.

5 Apprentissage d'un HMM

On a vu au chapitre précédent les deux algorithmes classiques utilisés dans le cadre des applications de reconnaissance et de segmentation utilisant les HMMs. Ces deux algorithmes supposent que l'on dispose d'un HMM construit et paramétré de manière à modéliser de façon satisfaisante les séquences que l'on souhaite traiter. La question qui nous préoccupe dans ce chapitre est celle de la construction d'un tel HMM. Dans le cas le plus favorable, le HMM recherché peut être construit directement à partir des connaissances *a priori* dont on dispose sur le domaine. Ce cas est, par exemple celui de Krogh *et al.* dans leur article sur la modélisation de séquences d'ADN de *e.coli* [14]. Dans cet article, ils utilisent les HMMs pour segmenter les séquences d'ADN en régions codantes et non-codantes. Les séquences manipulées sont des séquences d'acides nucléiques et un certain nombre de connaissances – telles que la structure générale du HMM et la fréquence des acides nucléiques en fonction de leur rôle dans les séquences – leur permettent de construire facilement leur modèle. Malheureusement ce cas de figure ne se rencontre que rarement, et, dans la plupart des applications, le ou les HMMs désirés doivent être construits à l'aide d'un algorithme d'apprentissage. Ces algorithmes sont appliqués sur un ensemble de séquences représentatives des séquences que l'on souhaite modéliser et appelées *séquences d'apprentissage*.

On peut distinguer dans le problème de l'apprentissage d'un HMM deux cas de figure distincts, suivant que la structure – le nombre d'états du HMM et les transitions autorisées – est connue ou ne l'est pas. Lorsque la structure est connue, le problème se réduit à un problème d'*entraînement* consistant à estimer les paramètres numériques – les distributions de probabilité de transition et de génération – de manière à expliquer au mieux les séquences d'apprentissage. Ce problème est celui adressé à la Section 5.1. Pour certaines applications, on ne dispose pas de connaissances suffisantes pour inférer naturellement la structure du HMM. L'apprentissage devient alors encore plus difficile. Il ne suffit plus de paramétrer une structure mais il faut également déduire cette structure des exemples fournis. Nous abordons ce problème à la Section 5.2.

5.1 Apprentissage quand la structure est connue

Nous nous intéressons dans cette section a l'apprentissage – ou entraînement – d'un HMM à partir d'une structure connue. On dispose pour cela d'un ensemble d'apprentissage composé de séquences supposées représentatives des séquences que l'on souhaite modéliser. Une approche possible est, suivant le principe du maximum de vraisemblance, de chercher les paramètres $\lambda = \langle T, G \rangle$ du HMM qui maximisent la probabilité de génération des séquences d'apprentissage. Soit $\mathcal{O} = \{O^1, \ldots, O^K\}$ l'ensemble des séquences d'apprentissage. On suppose que ces séquences sont indépendantes et donc que la probabilité de générer l'ensemble d'apprentissage est simplement le produit des probabilités de génération de chacune des séquences. Notre but est alors de trouver les paramètres $\lambda = \langle T, G \rangle$ de H qui maximisent

$$P(\mathcal{O}|H) = \prod_{k=1}^{K} P(O^k|H). \tag{7}$$

Une approche alternative au principe du maximum de vraisemblance est de chercher à maximiser la probabilité de génération des séquences d'apprentissage suivant leur chemin de Viterbi. Le but est alors de trouver les paramètres $\lambda = \langle T, G \rangle$ de H qui maximisent

$$P(\mathcal{O}|H, \mathcal{V}) = \prod_{k=1}^{K} P(O^k|H, V^k), \tag{8}$$

avec V^k le chemin de Viterbi de la séquence O^k dans H, et $\mathcal{V} = \{V^1, \ldots, V^K\}$. Les paramètres ne sont donc pas estimés en maximisant la *vraie* probabilité de génération des séquences d'apprentissage mais la probabilité de génération suivant les chemins les plus probables. Cette approche peut sembler moins rigoureuse que l'approche basée sur le principe du maximum de vraisemblance, mais elle est très utilisée en pratique et possède quelques arguments en sa faveur. D'abord on peut se dire que si le HMM recherché est destiné à être appliqué à des problèmes de segmentation *via* la recherche des chemins de Viterbi, il semble judicieux de l'entraîner de cette manière. Ensuite on remarque dans la pratique que la probabilité de génération d'un séquence de symboles suivant son chemin de Viterbi est en général beaucoup plus élevée que suivant n'importe quel autre chemin. Cette observation a conduit à l'hypothèse, appelée *hypothèse de Viterbi*, que tous les chemins, excepté le chemin de Viterbi, ont une probabilité nulle ou négligeable d'engendrer la séquence. Sous cette hypothèse, la probabilité de génération d'une séquence O^k par un HMM H peut être approximée par la probabilité de génération de O^k suivant son chemin de Viterbi dans H. On a alors :

$$P(\mathcal{O}|H) \approx P(\mathcal{O}|H, \mathcal{V}),$$

et maximiser l'Équation (8) revient à maximiser une approximation de l'Équation (7). Le dernier argument en faveur de cette méthode d'entraînement est que l'on peut, dans un cas de figure spécifique exposé à la Section 5.1.1, trouver la solution optimale au sens de la maximisation de l'Équation (7) alors que l'on ne connaît aucune méthode optimale pour la maximisation de l'Équation (8). Abe et Warmuth [1] ont étudié la question de savoir s'il existe un algorithme d'entraînement efficace d'un automate probabiliste, dans le cadre d'une adaptation naturelle du paradigme de *PAC-learning* introduit par Valiant [25]. Leur modèle est inspiré du modèle d'apprentissage non supervisé de Laird [16]. La question est de savoir, pour un ensemble d'exemples de taille *réduite*, s'il existe un algorithme qui converge, avec une *forte probabilité*, vers une solution *optimale*. Dans ce cadre, ils prouvent que la classe des automates probabilistes n'est pas entraînable en temps polynômial à la taille de l'alphabet, à moins que $RP = NP$. Pour les HMMs, la question reste ouverte à notre connaissance, mais on peut raisonnablement penser que le problème est aussi difficile.

Suivant la méthode d'estimation des paramètres choisie, il existe deux heuristiques permettant l'entraînement des HMMs. Ces deux algorithmes, très similaires, sont tous deux issus d'une méthode générale servant à l'estimation des paramètres d'une grande famille de modèles probabilistes et appelée algorithme d'*Expectation-Maximization* (EM) [8]. Pour des raisons de simplicité, nous commencerons par exposer l'algorithme adapté à la maximisation de l'Équation (8) (Section 5.1.1), et nous verrons à la Section 5.1.2 l'algorithme servant à maximiser l'Équation (7).

5.1.1 Entraînement de Viterbi

Comme évoqué dans la section précédente, il existe un cas où il est possible de paramétrer la structure du HMM de manière à maximiser l'Équation (8) de manière optimale. Ce cas est celui rencontré lorsque l'on connait les chemins de Viterbi des séquences d'apprentissage dans la structure. En effet, on peut alors associer à chaque état, chaque transition et chaque symbole attaché aux états du HMM le nombre de fois où ils sont utilisés pour générer l'ensemble d'apprentissage. Soit n_s, $n_{s\to s'}$ et n_s^o respectivement le nombre de fois où l'état s est utilisé, le nombre de fois où la transition $s \to s'$ est utilisée et le nombre de fois où le symbole o est généré par l'état s dans les chemins de Viterbi. Alors la Formule (8) peut être réécrite

$$P(\mathcal{O}|H, \mathcal{V}) = \prod_{s\in S} \left(\prod_{o\in\Sigma} P(o|s)^{n_s^o} \prod_{s'\in\mathcal{S}} P(s \to s')^{n_{s\to s'}} \right). \tag{9}$$

Maximiser cette formule revient à maximiser indépendamment chacun de ses sous-produits. On peut donc estimer les paramètres T en maximisant l'expression $\prod_{s'\in\mathcal{S}} P(s \to s')^{n_{s\to s'}}$ et les paramètres G en maximisant $\prod_{o\in\Sigma} P(o|s)^{n_s^o}$. Les estimateurs de T et G sont donc respectivement

$$\widehat{P}(s \to s') = \frac{n_{s\to s'}}{n_s} \tag{10}$$

et

$$\widehat{P}(o|s) = \frac{n_s^o}{n_s}. \tag{11}$$

Cette méthode d'estimation n'est possible que dans le cas favorable où les chemins de Viterbi sont connus. Lorsque ce n'est pas le cas, le problème est évidemment plus difficile et l'algorithme d'entraînement de Viterbi peut alors être un solution.

L'algorithme d'entraînement de Viterbi (voir Algorithme 1) est un algorithme de réestimation itératif. Il consiste, à partir d'un paramétrage initial du HMM, à calculer les chemins de Viterbi des séquences d'apprentissage à l'aide de l'algorithme de Viterbi décrit à la Section 4. Les chemins de Viterbi son utilisés pour calculer, comme on l'a fait ci-dessus, le nombre de fois où chaque transition, chaque état et chaque symbole attaché aux états est utilisé. On réestime alors à l'aide des formules (10) et (11) les paramètres du HMM, on reparamètre la structure à l'aide de ces estimations et on réitère la procédure jusqu'à obtention de la stabilité.

Algorithme 1: ALGORITHME D'ENTRAÎNEMENT DE VITERBI

Données : $\mathcal{O} = \{O^1, \ldots, O^n\}$;
H une structure de HMM.

Choisir un paramétrage initial $\lambda = \langle T, G\rangle$ de H;
répéter
  Initialiser toutes les variables n_s, $n_{s\to s'}$ et n_s^o à 0;
  pour chaque $O^k \in \mathcal{O}$ **faire**
    Calculer le chemin de Viterbi de O^k dans H;
    Ajouter aux variables n_s, $n_{s\to s'}$ et n_s^o la contribution de O^k;
  Réestimer les paramètres λ de H en utilisant les formules (10) et (11);
jusqu'à *stabilité du paramétrage*;

On peut montrer [13] que la valeur de $P(\mathcal{O}|H, \mathcal{V})$ augmente à chaque itération et que l'algorithme converge vers un optimum local. Malheureusement, il existe généralement un grand nombre d'optima locaux et le paramétrage obtenu par ce processus dépend fortement du paramétrage initial choisi.

5.1.2 Entraînement de Baum-Welch

L'algorithme d'entraînement de Baum-Welch est un algorithme qui cherche à estimer les paramètres $\lambda = \langle T, G\rangle$ du HMM en maximisant l'Équation (7). C'est un algorithme de réestimation itératif qui fonctionne sur le même principe que l'algorithme d'entraînement de Viterbi. Dans ce dernier, on compte le nombre de fois où chaque état, chaque transition et chaque symbole attaché aux état est utilisé dans les chemins de Viterbi. Ici on veut maximiser les probabilités de génération réelles, et non celles des chemins les plus probables. On va donc associer au états, aux transitions et aux symboles le nombre de fois où ils sont utilisés pour toutes les séquences et *tous* les chemins susceptibles de générer les séquences, pondéré par la probabilité du chemin. Ces comptes pondérés sont alors utilisés pour réestimer les paramètres du modèle de la même manière que pour l'algorithme d'entraînement de Viterbi.

De manière plus formelle, considérons pour commencer une unique séquence d'observations $O = o_1 \ldots o_T$. Notre but est de trouver les paramètres $\lambda = \langle T, G\rangle$ qui maximisent la probabilité $P(O|H)$ de générer O avec H. Nous verrons ensuite comment étendre l'algorithme à un ensemble de séquences d'observations. Soit

$$\xi_t(s, s') = P(s_t = s, s_{t+1} = s'|O, H)$$

la probabilité qu'en générant O avec H on passe par l'état s à l'instant t et par l'état s' à l'instant $t + 1$. Cette équation peut se réécrire :

$$\xi_t(s, s') = \frac{P(s_t = s, s_{t+1} = s', O|H)}{P(O|H)},$$

et en utilisant les variable forward et backward introduites à la Section 3 :

$$\xi_t(s, s') = \frac{\alpha_t(s)P(s \to s')P(o_{t+1}|s')\beta_{t+1}(s')}{P(O|H)}. \tag{12}$$

Considérons également la variable $\gamma_t(s)$:

$$\gamma_t(s) = P(s_t = s|O, H),$$

qui exprime la probabilité qu'en générant O avec H on on se trouve sur l'état s à l'instant t. Exprimée en fonction de $\xi_t(s, s')$, $\gamma_t(s)$ s'écrit :

$$\gamma_t(s) = \sum_{s' \in \mathcal{S}} \xi_t(s, s'). \tag{13}$$

Si on somme $\gamma_t(s)$ sur l'ensemble des instants t, on obtient une quantité que l'on peut interpréter comme l'espérance du nombre de fois où l'état s est utilisé pour générer la séquence O. De même, si on somme $\xi_t(s, s')$ sur l'ensemble des instants t, on obtient une quantité que l'on peut interpréter comme l'espérance du nombre de fois où la transition $s \to s'$ est utilisée pour générer la séquence O. On a donc :

$$\sum_{t=0}^{T} \gamma_t(s) = \text{espérance du nombre de passages dans } s, \tag{14}$$

$$\sum_{t=0}^{T} \xi_t(s, s') = \text{espérance du nombre de transitions de } s \text{ vers } s', \tag{15}$$

et de manière similaire à (14) on a également :

$$\sum_{t=1\ /\ o_t=o}^{T} \gamma_t(s) = \text{espérance du nombre de générations de } o \text{ dans } s. \tag{16}$$

En combinant les Formules (14), (15) et (16) on peut alors proposer un ensemble d'estimateurs des paramètres $\lambda = \langle T, G \rangle$ du HMM :

$$\begin{aligned}
\widehat{P}(s \to s') &= \frac{\text{espérance du nombre de transitions de } s \text{ vers } s'}{\text{espérance du nombre de passages dans } s} \\
&= \frac{\sum_{t=0}^{T} \xi_t(s, s')}{\sum_{t=0}^{T} \gamma_t(s)}
\end{aligned}$$

$$\begin{aligned}
\widehat{P}(o|s) &= \frac{\text{espérance du nombre de générations de } o \text{ dans } s}{\text{espérance du nombre de passages dans } s} \\
&= \frac{\sum_{t=1\ /\ o_t=o}^{T} \gamma_t(s)}{\sum_{t=0}^{T} \gamma_t(s)}
\end{aligned}$$

Dans le cas où l'on dispose d'un ensemble de séquences d'observations $\mathcal{O} = \{O^1, \ldots, O^K\}$, ces estimateurs se généralisent simplement de la manière suivante :

$$\widehat{P}(s \to s') = \frac{\sum_{k=1}^{K} \sum_{t=0}^{T} \xi_t^k(s, s')}{\sum_{k=1}^{K} \sum_{t=0}^{T} \gamma_t^k(s)} \tag{17}$$

$$\widehat{P}(o|s) = \frac{\sum_{k=1}^{K} \sum_{t=1\ /\ o_t=o}^{T} \gamma_t^k(s)}{\sum_{k=1}^{K} \sum_{t=0}^{T} \gamma_t^k(s)} \tag{18}$$

On peut alors proposer une version complète de l'algorithme de Baum-Welch (voir Algorithme 2). Comme pour l'algorithme d'entraînement de Viterbi, on peut prouver [3] que la probabilité de génération des séquences $P(\mathcal{O}|H)$ augmente à chaque itération et que l'algorithme converge vers un optimum local. Néanmoins, dans ce cas là, l'optimum local n'est jamais atteint et il est nécessaire de définir un critère d'arrêt – par exemple en stoppant la procédure lorsque l'augmentation de $P(\mathcal{O}|H)$ est inférieure à un certain seuil, ou lorsque le nombre d'itérations effectuées est jugé suffisamment important.

Algorithme 2: ALGORITHME D'ENTRAÎNEMENT DE BAUM-WELCH

Données : $\mathcal{O} = \{O^1, \ldots, O^n\}$;
H une structure de HMM.

Choisir un paramétrage initial $\lambda = \langle T, G \rangle$ de H;
répéter
  pour chaque $O^k \in \mathcal{O}$ **faire**
    Calculer les valeurs des variables $\alpha_t^k(s)$ à l'aide de l'algorithme forward;
    Calculer les valeurs des variables $\beta_t^k(s)$ à l'aide de l'algorithme backward;
    Calculer les valeurs des variables $\xi_t^k(s, s')$ à l'aide de la Formule (12);
    Calculer les valeurs des variables $\gamma_t^k(s)$ à l'aide de la Formule (13);
    Calculer et stocker les valeurs des expressions (14), (15) et (16)
  Réestimer les paramètres λ de H en utilisant les formules (17) et (18);
jusqu'à *critère d'arrêt*;

5.2 Apprentissage quand la structure est inconnue

On a vu à la section précédente les algorithmes d'entraînement classiques à partir d'une structure connue. Pour certaines applications, néanmoins, aucune connaissance ne permet d'inférer naturellement cette structure (voir par exemple [5]). D'autre part, comme le soulignent Freitag et McCallum [11], la construction "à la main" de HMMs n'est pas aisée. Enfin, il est à noter que l'intuition humaine ne correspond pas toujours à la structure qui tire le mieux parti du potentiel des HMMs. Dans le même article [11], Freitag et McCallum montrent expérimentalement sur un ensemble de problèmes d'extraction d'informations à partir de données textuelles, que les modèles construits à partir des données donnent généralement de meilleurs résultats que les modèles construits à la main et entraînés à l'aide de l'algorithme de Baum-Welch. Le problème est alors, non seulement de paramétrer une structure, mais encore d'inférer cette structure à partir des exemples d'apprentissages. Plusieurs méthodes ont été proposées pour résoudre ce problème. Remarquons tout d'abord qu'une approche du type maximum de vraisemblance n'est clairement pas suffisante. En effet, il n'est pas difficile de construire un HMM qui a la probabilité maximale de générer les séquences d'apprentissage. Il suffit de construire un HMM équivalent à une disjonction de ces séquences : chaque chemin du *start* vers le *end* représente exactement une séquence d'apprentissage ; chaque état ne génère qu'un des symboles de l'alphabet et n'a – mis à part l'état *start* – qu'une transition sortante (voir l'exemple de la Figure 3). Le HMM construit de cette manière est spécifique aux séquences d'apprentissage – il ne peut générer que ces séquences – et est donc inutilisable en pratique puisque incapable de reconnaître d'autres séquences. On se trouve alors face à un problème classique en apprentissage, celui de la recherche d'un *biais inductif*, c'est-à-dire d'une règle de généralisation *a priori* permettant d'inférer des HMMs qui puissent être utilisés sur de nouvelles séquences. Dans la suite de cette section, nous décrivons brièvement quelques approches qui ont été utilisées pour résoudre ce problème.

5.2.1 Apprentissage par généralisation

L'apprentissage d'un HMM par généralisation, introduite par Stolcke et Omohundro [24], procède de manière gloutonne et ascendante. Cette technique est inspirée d'une méthode très utilisée

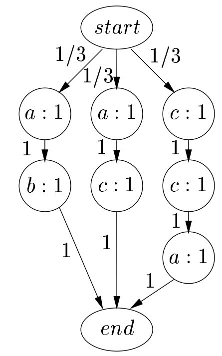

FIG. 3: HMM de vraisemblance maximale construit à partir des séquences d'apprentissage $a - b$, $a - c$ et $c - c - a$.

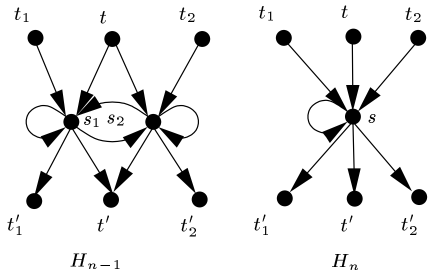

FIG. 4: Structure avant et après la fusion des états s_1 et s_2.

pour l'inférence d'automates déterministes probabilistes [6] et non probabilistes [18], et qui s'est imposée comme la technique de référence lors du récent concours d'inférence d'automates *Abbadingo One DFA Learning Competition* [17]. Elle consiste à partir d'un HMM H_0 spécifique – construit comme exposé ci-dessus – et, par une opération élémentaire de *fusion d'états* – voir Figure 4 –, de transformer progressivement ce HMM en un modèle plus simple et plus général. Ce gain en généralité s'accompagne généralement d'une baisse de la vraisemblance des séquences d'apprentissage. Cette méthode construit ainsi une suite de HMMs $H_0, H_1, \cdots, H_n$ où H_n est le HMM obtenu en fusionnant deux états de H_{n-1}. Après chaque fusion, les paramètres de la structure sont réestimés de manière à maximiser les chemins de Viterbi des séquences d'apprentissage – Expression (8). Plutôt que d'utiliser l'algorithme d'entraînement de Viterbi à chaque itération, les chemins de Viterbi sont supposés préservés par les fusions. En d'autres termes, on suppose que les chemins de Viterbi d'après la fusion peuvent être déduits des chemins de Viterbi d'avant la fusion en remplaçant simplement les occurrences des états fusionnés par l'état résultant de la fusion. Sous cette hypothèse, les paramètres du HMM se réestiment aisément grâce aux Formules (10) et (11). De plus, seuls les paramètres attachés au nouvel état et à ses transitions adjacentes ont à être réestimés puisque la fusion n'a pas d'effet sur les autres états et transitions.

Différentes techniques peuvent être utilisées pour guider la fusion au cours de l'apprentissage. Il faut : 1) choisir un couple d'états à fusionner à chaque itération, 2) définir un critère d'arrêt pour l'algorithme. Stolcke et Omohundro [24] proposent d'utiliser un biais s'exprimant comme une probabilité *a priori* sur l'ensemble des HMMs possibles. Leur critère de fusion est alors la maximisation d'une probabilité *a posteriori* construite suivant le théorème de Bayes :

$$P(H|\mathcal{O}) = \frac{P(H)P(\mathcal{O}|H)}{P(\mathcal{O})}, \tag{19}$$

où $P(H)$ est la probabilité *a priori* du HMM H, $P(\mathcal{O}|H)$ est la probabilité de générer les séquences d'apprentissage $\mathcal{O}$ avec H – calculée à l'aide de la Formule (7) ou approximée par l'Expression (8) –, et $P(\mathcal{O})$ est la probabilité d'observer les séquences $\mathcal{O}$. De manière à favoriser les structures de tailles réduites, ils proposent d'exprimer la probabilité *a priori* du HMM à l'aide d'une fonction du type $P(H) = \exp(-l(H))$, où $l(H)$ est la taille du HMM calculée en fonction du nombre d'états et de transitions. À chaque itération, ils choisissent alors le couple d'états dont la fusion engendre l'augmentation la plus importante de la probabilité *a posteriori* $P(H|\mathcal{O})$ – ou plus simplement du produit $P(H)P(\mathcal{O}|H)$, puisque $P(\mathcal{O})$ est une constante indépendante du HMM. La procédure est stoppée lorsque plus aucun couple ne permet d'augmenter cette probabilité. Leur critère de fusion et d'arrêt définit donc un compromis entre généralisation et vraisemblance des séquences d'apprentissage. Bréhélin *et al.* [5] utilisent un algorithme similaire pour l'apprentissage de leur modèle. Néanmoins leur domaine d'application impose un critère d'arrêt spécifique et ils choisissent à chaque itération le couple d'états dont la fusion engendre la plus petite perte de vraisemblance. Seymore *et al.* [23] utilisent également une approche de ce type pour l'apprentissage de HMMs destinés à la segmentation des entêtes d'articles scientifiques. Leur critère de fusion n'est pas basé sur des probabilités mais sur certaines relations topologiques entre les états. Ils proposent deux règles de fusion simples : la première, "neighbor-merging", fusionne les états du même type reliés par une transition ; la seconde, "V-merging", fusionne deux états du même type ayant un voisin commun.

5.2.2 Apprentissage par spécialisation

À l'inverse de l'apprentissage par généralisation, l'apprentissage par spécialisation consiste à partir d'un HMM très général – par exemple composé d'un unique état qui boucle sur lui-même et des états *start* et *end* –, et par une opération de spécialisation – la création d'un nouvel état, ou l'ajout d'une transition entre deux états existants – de construire progressivement un HMM plus spécifique de manière à augmenter la vraisemblance des séquences d'apprentissage. Freitag et McCallum [11] utilisent une approche de ce type pour construire des HMMs permettant d'isoler certains champs à partir de données textuelles. Ils définissent à la fois un type de structure approprié à leur domaine d'application et un ensemble de règles de spécialisation qui permettent, à partir d'un HMM général, la construction incrémentale de HMMs plus spécifiques et conformes à leur type de structure. À chaque itération, leur algorithme construit et paramètre à l'aide de l'algorithme de Baum-Welch, tous les HMMs qui peuvent être composées à l'aide d'une des règle de spécialisation appliquée sur le HMM courant. Le meilleur HMM obtenu – au sens d'un critère basé sur le taux d'erreur de prédiction calculé sur un échantillon indépendant des séquences d'apprentissage – est alors stocké et devient le HMM courant. À l'issue d'un nombre d'itérations prédéfini, l'algorithme renvoie le meilleur HMM parmi tous les HMMs stockés.

D'autres approches peuvent être utilisées pour l'apprentissage de la structure des HMMs. Brand [4] propose par exemple une variante de l'algorithme de Baum-Welch dont l'objet est la maximisation, non pas de la vraisemblance des séquences d'apprentissage comme c'est le cas dans l'algorithme classique, mais d'une probabilité *a posteriori* construite à l'aide d'une probabilité *a priori* de forme entropique. Cette probabilité *a priori* est fonction des paramètres du HMM – probabilités de transition et d'émission – et sa forme entropique favorise les probabilités proche des extrêmes (0 ou 1). Alliée à une opération d'extinction – suppression des transition et des symboles qui ont une probabilité trop faible –, elle permet de construire, à partir d'un HMM initial fortement connexe ayant un nombre d'états important, des HMMs de tailles réduites et avec peu de transitions.

6 Conclusion

Nous avons présenté ici une vue d'ensemble des HMMs, de leurs applications et des algorithmes classiques utilisés dans la littérature : les algorithmes de calcul de la probabilité de génération d'une séquence par un HMM, l'algorithme de recherche du chemin optimal, ainsi que les algorithmes d'entraînements et quelques pistes pour l'apprentissage de la structure. Bien d'autres sujets aurait pu être abordés, comme les différents types de structure qui peuvent être utilisées, les problèmes d'*overfitting* lors de l'entraînement, les problèmes liés à l'implémentation des algorithmes (notamment les problèmes d'arrondis posés par les calculs de probabilité), les méthodes de comparaison des HMMs, ... Pour tous ces sujets, nous renvoyons le lecteur intéressé à des articles et ouvrages qui font référence : l'excellent tutorial de Rabiner [20] aborde la plupart de ces sujets ainsi que les approches classiques pour la reconnaissance de la parole ; l'ouvrage de Durbin *et al.* [9] complète efficacement ce tutorial et expose les principales applications en Bioinformatiques ; l'article de Freitag et McCallum [10] propose quelques pistes supplémentaires pour la résolution du problème d'*overfitting*. Il existe sur le Web quelques articles en ligne : l'Université de Santa Cruz [2] (Californie) propose une série d'articles sur l'application des HMMs à la Bioinformatique ; la page Web du "World-Wide Knowledge Base Project"[3] de Tom Mitchell propose pour sa part des articles sur l'extraction d'information à partir de données textuelles.

Remarquons pour conclure, que les HMMs se sont imposés comme le modèle de référence pour la résolution de certains types de problèmes dans plusieurs domaines d'application, que ce soit en reconnaissance de la parole, en modélisation de séquences biologiques ou pour l'extraction d'information à partir de données textuelles. La lisibilité du modèle et surtout la simplicité et l'efficacité de ses principaux algorithmes sont très certainement responsable de ce succès.

Références

[1] Naoki Abe and Manfred Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9(2-3) :205–260, 1992.

[2] J. K. Baker. The DRAGON system – an overview. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23 :24–29, 1975.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in statistical analysis of probabilistic functions in Markov chains. *The Annals of Mathematical Statistics*, 41(1) :164–171, 1970.

[4] Matthew Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, 11(5) :1155–1182, 1999.

[5] Laurent Bréhélin, Olivier Gascuel, and Gilles Caraux. Hidden Markov Models with Patterns and their application to integrated circuit testing. In *11th European Conference on Machine Learning*, 2000.

[6] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the Second International ICGI Colloqium on Grammatical Inference and Applications*, volume 862 of *LNAI*, pages 139–152, 1994.

[7] F. Casacuberta. Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7) :691–695, July 1990.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc. B*, 39 :1–38, 1977.

[9] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[10] Dayne Freitag and Andrew McCallum. Information Extraction with HMM and Shrinkage. In *AAAI'99 Workshop on ML for Information Extraction*, pages 31–36, 1999.

[11] Dayne Freitag and Andrew McCallum. Information Extraction with HMM structures learned by stochastic optimization. In *AAAI*, 2000.

[12] David Haussler, Anders Krogh, Saira Mian, and Kimmen Sjolander. Protein modeling using hidden Markov models : analysis of globins. Technical Report UCSC-CRL-92-23, June 1992.

[2] `http ://www.cse.ucsc.edu/research/compbio/`

[3] `http ://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/`

[13] B. H. Juang and L. R. Rabiner. The segmental K-means algorithm for estimating parameters of hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9) :1639, 1990.

[14] Anders Krogh, I. Saira Mian, and David Haussler. A hidden Markov model that finds genes in e.coli DNA. Technical Report UCSC-CRL-93-33, University of California, Santa Cruz, Jack Baskin School of Engineering, May 1994.

[15] A. Kundu, Y. He, and P. Bahl. Recognition of handwritten word : First and second order Hidden Markov Model based approach. In *CVPR'88 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, June 5–9, 1988)*, pages 457–462, 1988.

[16] Philip D. Laird. Efficient unsupervised learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 297–311, MIT, August 1988.

[17] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA learning competition and new evidence-driven state merging algorithm. In *Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI-98)*, pages 1–12, July 1998.

[18] J. Oncina and P. Garcia. Inferring regular languages in polynomial updated time. *Pattern Recognition and Image Analysis*, pages 49–61, 1992.

[19] B.R. Povlow and S.M. Dunn. Texture classification using noncausal hidden Markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10) :1010–1014, October 1995.

[20] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–285, 1989.

[21] C. Raphael. Automatic segmentation of acoustic musical signals using hidden Markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4) :360–370, 1998.

[22] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time delay neural networks and hidden Markov models. In *Proc. ICASSP '94*, pages II–637–II–640, Adelaide, Austrailia, 1994.

[23] Kristie Seymore, Andrew McCallum, and Ronald Rosenfeld. Learning hidden Markov model structure for Information Extraction. In *AAAI'99 Workshop on ML for Information Extraction*, pages 37–42, 1999.

[24] Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. In *Proceedings of the ICGI*, volume 862 of *LNAI*, pages 106–118, Berlin, 1994.

[25] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11) :1134–1142, November 1984.