Logical Hidden Markov Models (Extended Abstract)

K. Kersting[1] and T. Raiko[1,2] and L. De Raedt[1]
[1]Institute for Computer Science
Machine Learning Lab
Albert-Ludwigs University of Freiburg
Georges-Koehler-Allee, Building 079
79112 Freiburg, Germany
[2]Helsinki University of Technology
Laboratory of Computer and
Information Science,
P.O. Box 5400,
02015 HUT, Finland

Abstract

Logical hidden Markov models (LOHMMs) are a generalization of hidden Markov models (HMMs) to analyze sequences of logical atoms. In LOHMMs, abstract states summarize sets of states and are represented by logical atoms. Transitions are defined between abstract states to summarize sets of transitions between states. Unification is used to share information among states, and between states and observations. A LOHMM can be designed to be smaller than an equivalent HMM by an order of magnitude in the number of parameters. We devised adaptations of the classical HMM procedures such as the forward and backward procedures. Our experiments show that LOHMMs have a good generalization performance, and that it is easy to extract characteristic patterns from the trained LOHMM.

1 Introduction

Hidden Markov models (Rabiner and Juang, 1986) (HMMs) are among the most widely and successfully used tools for the analysis of sequential data. Areas of application include computational biology, user modeling, speech recognition, (stochastic) natural language processing, and robotics. HMMs are Markov chains, where each state generates an observation. They encompass as special cases time-independent multinomial models, fully observable Markov chains and edit distance-based models. Despite their successes, however, it is well-known that HMMs have a number of weaknesses. One of the major weaknesses is that HMMs handle only sequences of unstructured symbols, i.e. they lack the structure that exists in most real-world domains.

Consider e.g. UNIX command prediction (Davison and Hirsh, 1998), i.e. predicting the next element in a sequence of user commands such as *emacs lohmms.tex*, *ls*, *latex lohmms.tex*, ... Commands can take parameters (such as the filename on which the command needs to be performed), can return information (such as a return code) or can have other properties (such as current working directory, cost, etc.). These features can be very important. Ignoring e.g. the filename, we lose information as *emacs*, *ls*, *latex* equally likely represents *emacs lohmms.tex*, *ls*, *latex hmms.tex*. Taking all possible filenames into account yields an unreasonably large number of parameters and, more over, inhibits generalization (cf. (Davison and Hirsh, 1998; Korvemaker and Greiner, 2000; Jacobs and Blockeel, 2001)). Another example comes from computational biology. With the number of determined protein structures and the availability of classification schemes, it becomes increasingly important to develop computer methods that automatically extract structural signatures for classes of proteins. Protein secondary structures can naturally be encoded as structured sequences of *strands* and *helices* having certain orientations, types, and lengths. To summarize, HMMs cumbersomely

handle sequences where the symbols of the output alphabet are structured.

In this paper, we will overcome this weakness by introducing logical hidden Markov models (LOHMMs). LOHMMs process sequences of logical atoms (hence the term "logical") which are the primary syntactic component of the predicate calculus [1]. E.g. the atom $helix(h(left, alpha), 9)$ denotes a left-handed alpha helix of length 9, and $emacs(lohmms.tex, luc)$ denotes that the user Luc edits the file *lohmms.tex* using *Emacs*. In our framework, such logical atoms are used as symbols in the alphabet of a LOHMM and also to denote the states. Thus, LOHMMs are capable of handling structured sequences represented as sequences of logical atoms such as $emacs(lohmms.tex), ls, latex(lohmms.tex), \ldots$

We chose the logical approach for two reasons. Firstly, variables in the atoms allow us to abstract from specific symbols. E.g. the logical atom $emacs(X, luc)$ would represent all files that the user Luc could edit using Emacs. Secondly, unification allows us to share information among hidden states and between hidden states and observations. E.g. the sequence $emacs(X, luc), latex(X, luc)$ represents that the same file is used as an argument for both Emacs and LaTeX.

This paper is organized as follows. In the next section, we briefly introduce basic logical concepts and notations. In Section 3, we introduce logical hidden Markov models. Section 4 describes our empirical evaluation of logical hidden Markov models. In Section 5 we discuss related work. Subsequently, we conclude and discuss future work.

2 Logical Preliminaries

A *first-order alphabet* Σ is a set of relation symbols r with arity $m \geq 0$, and a set of functor symbols f with arity $n \geq 0$. If $n = 0$ then f is called a constant, if $m = 0$ then p is called a propositional variable. (We assume that at least one constant is given.) An *atom* $r(t_1, \ldots, t_n)$ is a relation symbol r followed by a bracketed n-tuple of terms t_i. A *term* T is a variable V or a functor symbol $f(t_1, \ldots, t_k)$ immediately followed by a bracketed n-tuple of terms t_i. An *iterative clause* is a formula of the form $A \leftarrow B$ where the *head* A and the *the body* B are logical atoms. A substitution $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$, e.g. $\{X/tex\}$, is an assignment of terms t_i to variables V_i. Applying a substitution σ to a term, atom or clause e yields the instantiated term, atom, or clause $e\sigma$ where all occurrences of the variables V_i are simultaneously replaced by the term t_i. e.g. $ls(X) \leftarrow emacs(F, X)\{X/tex\}$ yields $ls(tex) \leftarrow emacs(F, tex)$. A substitution σ is called a *unifier* for a finite set S of atoms if $S\sigma$ is singleton. A unifier σ for S is called a *most general unifier* (MGU) for S if, for each unifier σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$. A term, atom or clause E is called *ground* when it contains no variables, i.e., $vars(E) = \emptyset$. The *Herbrand base* HB_Σ of Σ, is the set of all ground atoms constructed with the predicate and functor symbols in Σ. The set $G_\Sigma(A)$ of an atom A consists of all ground atoms $A\theta$ that belong to HB_Σ.

3 Logical Hidden Markov Models

In LOHMMs, we summarize sets of states by *abstract states*, which are represented by *logical atoms*. An abstract state then represents all states that can be obtained by instantiating the atom (i.e. replacing the variables by terms). E.g. the abstract state $emacs(X)$, where X is a variable, could represent the set of states $\{emacs(lohmms.tex), emacs(.cshrc)\}$ depending on the terms (lohmms.tex and .cshrc in this case) in the LOHMM. If the abstract state does not contain any variables, e.g. $emacs(lohmms.tex)$, it is an ordinary state representing a singleton set. In analogy to Rabiner's interpretation, abstract states correspond to "urns of urns". In a first step, a state is sampled from the encompassing abstract state. Subsequently, a symbol is generated from the state. This also works for ordinary states where the encompassing abstract state contains ex-

[1] See (Lloyd, 1989) for a general introduction to logic programming.

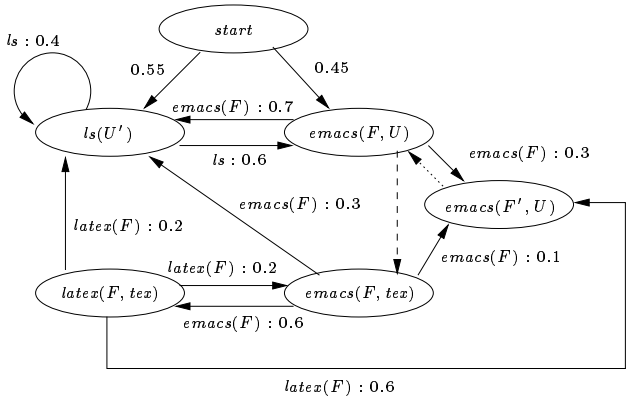

Figure 1: A first-order logical hidden Markov model.

actly one state, which is always selected.

Abstract states are connected by *abstract transitions*, which summarize sets of transitions between states. Together, the abstract states and abstract transitions of a logical (hidden) Markov model provide a compact representation of an equivalent HMM. We will explain these concepts on an example for UNIX command line prediction. For more technical details and for an application within computational biology we refer to (Kersting et al., 2002).

3.1 An Example of a LOHMM

Figure 1 shows a graphically represented example of a LOHMM for user modeling. Assume that there are two classes of users namely *tex* and *others*. A *tex* user will call LaTeXwith a high probability after editing a file F using *emacs*, whereas *other* users are more likely to call *ls*. However, the class of a user is hidden. For now, we assume there are two filenames. The vertices in the model represent abstract states, and we find three different types of edges:

- **Solid edges** between abstract states specify the abstract transitions. Transition probabilities and emission symbols are associated to them. An example transition from Figure 1 is $ls(U') \xleftarrow{emacs(F):0.7} emacs(F,U)$. Such a solid edge expresses that if one is in one of the states represented by $emacs(F,U)$, one will go to one of the states in $ls(U')$ with probability 0.7 while emitting the (abstract) symbol $emacs(F)$.

- **Dotted edges** indicate that two abstract states behave in exactly the same way. If we follow a transition to an abstract state with an outgoing dotted edge, we will automatically follow that edge. An example dotted edge in Figure 1 goes from $emacs(F',U)$ to $emacs(F,U)$. It represents the fact that the two abstract states are identical. The dotted edge is needed in this case because the variables appearing in the abstract states are different. We could not have written this using solid edges alone as the meaning of the solid edge $emacs(F,U) \xleftarrow{emacs(F):0.3} emacs(F,U)$ is different from that of $emacs(F',U) \xleftarrow{emacs(U):0.5} emacs(F,U)$. Whereas the first transition only allows a transition between the same state say $emacs(lohmms.tex, others)$ (because the F is identical), the second one allows transition between different states such as $emacs(lohmms.tex, others)$ and $emacs(.cshrc, others)$. In a logical sense, dotted edges correspond to some form of recursion.

- **Dashed edges** represent the *more general* relation, which is used for a kind of default reasoning. We follow those rules that are most specific for the current state. Consider the dashed edge in Figure 1 connecting $emacs(F,U)$ and $emacs(F,tex)$. This dashed edge denotes that $emacs(F,tex)$ is a more specific state than $emacs(F,U)$. This implies that the set of states represented by the more specific (abstract) state is a subset of that represented by the more general one. Logically speaking, the more specific state $emacs(F,tex)$ can be obtained by substituting U by *tex* in the more general state $emacs(F,U)$. Dashed edges and default reasoning are useful because they represent exceptions. Indeed, in our current example, the outgoing edges and probability labels associated to $emacs(F,tex)$

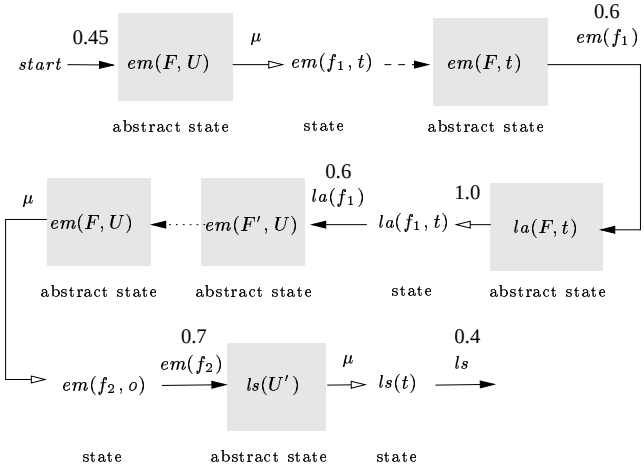

Figure 2: Generating the observation sequence $emacs(lohmms.tex), latex(lohmms.tex), emacs(hmm.txt), ls$ by the LOHMM in Figure 1. The command *emacs* is abbreviated by *em*, f_1 denotes the filename *lohmms.tex*, and f_2 represents *hmm.txt*. White filled arrows indicate selections.

are different from those for $emacs(F, U)$. This actually implies that $emacs(F, tex)$ acts as an exception to the states represented by $emacs(F, U)$. So for $U = tex$ we employ the transitions from $emacs(F, tex)$ and for $U \neq tex$ we follow those indicated by $emacs(F, U)$.

3.2 Generating Observations

LOHMMs are generative models. Let us explain how the model in Figure 1 generates the sequence of observations

$$emacs(lohmms.tex), latex(lohmms.tex), emacs(hmm.txt), ls$$

(cf. Figure 2). It chooses an initial abstract state, say $emacs(F, U)$. In each abstract state, the model samples values for all variables that are not instantiated yet according to a *selection distribution* μ.

The function μ specifies for each abstract state a distribution over the possible instantiations of the abstract state. E.g.

$$\mu(emacs(lohmms.tex, tex) \mid emacs(lohmms.tex, U)) = 0.5$$

says that the model samples $emacs(lohmms.tex, tex)$ with probability 0.5 from $emacs(lohmms.tex, U)$ whereas

$$\mu(emacs(hmm.txt, tex) \mid emacs(F, U)) = 0.05$$

specifies that $emacs(hmm.txt, tex)$ is sampled with probability 0.05 from $emacs(F, U)$. In general, any probabilistic representation such as Bayesian networks might be used to represent μ. In our experiments, we followed the naïve Bayes approach to reduce the model complexity. Each argument of a predicate is assumed to be independent of the other arguments. E.g., to compute $\mu(emacs(hmm.txt, tex))$, we compute the product of $P_F(hmm.txt)$ and $P_U(tex)$.

To go on in the example, since both variables F and U are uninstantiated, the model samples the state $emacs(lohmms.tex, tex)$. Forced to follow the dotted edge, it enters the abstract state $emacs(F, tex)$ which represents an exception to $emacs(F, U)$. Since the value of F was already instantiated in the previous abstract states, the model samples with probability 1.0 the state $emacs(lohmms.tex, tex)$. Now, the model goes over to the abstract state $latex(F, tex)$, emitting $emacs(lohmms.tex)$ because the abstract observation $emacs(F)$ is already fully instantiated. Again, since the value of F was already instantiated in the previous abstract state the model samples with probability 1.0 the state $latex(lohmms.tex, tex)$. Next, we move on to abstract state $emacs(F', tex)$, emitting $latex(lohmms.tex)$. Variable F' in $emacs(F', tex)$ is not yet bound; so, a value, say *hmm.txt*, is sampled from μ. The dotted edge brings us back to $emacs(F, U)$ and automatically unifies F with F', which is bound to *hmm.txt*. The variable U is already instantiated. Emitting $emacs(hmm.txt)$, the model makes a transition to abstract state $ls(U')$. Assume that it samples *others* for variable U'. Then, it remains in the abstract state $ls(U')$ with probability 0.4. Considering all possible samples, this process is similar to unrolling recurrent neural networks (Dean and Kanazawa, 1988) or dynamic Bayesian networks (Williams and Zipser, 1995), or to grounding clause programs (Lloyd, 1989).

To summarize, an abstract transition is an expression of the from $p : H \xleftarrow{O} B$ where $p \in [0, 1]$, and H, B and O are logical atoms. The semantics of an abstract transition is as follows. Let h (resp. b, o) be a ground atom over H (resp. B, O). Let σ_b be the most general unifier of b and B, and σ_h be the *most general unifier* of $H\sigma_b$ and O. Then, the model makes a transition from state b to h emitting o with probability

$$p \cdot \mu(h \mid H\sigma_b) \cdot \mu(o \mid O\sigma_b\sigma_h).$$

A *logical hidden Markov model* over a first-order language Σ is a tuple (L, μ) where L is a set of abstract transitions and μ specifies the selection probability for all abstract states over Σ. The *subsumption lattice* among the bodies B's of all abstract transitions in L is assumed to be *well-founded*. An atom B subsumes an atom B' if the set of all ground instances of B' is a subset of the corresponding set of B, i.e. $G_\Sigma(B') \subseteq G_\Sigma(B)$. A partial-order $\prec$ among a set S is *well-founded* if each non-empty subset of S has a minimal element w.r.t. $\prec$.

So far, we left out one important type of domains. Usually, identifiers such as arbitrary filenames are only needed for references. It does not make sense to select a specific identifier. Therefore, we select an unspecified one with the joint probability of all of them. However, unspecified identifiers are still subject to unification. For more details, we refer to (Kersting et al., 2002) and to the (forthcoming) long paper.

3.3 Semantics and Evaluation

There are two primary differences to HMMs:

1. Transition probabilities are defined by a product of the abstract transition probability and the selection probability (see also Figure 3).

2. The set of states represented by an abstract state and therefore the set of transitions can vary according to the domains associated to predicates.

Therefore, having the described grounding/unrolling process in mind, it is clear that

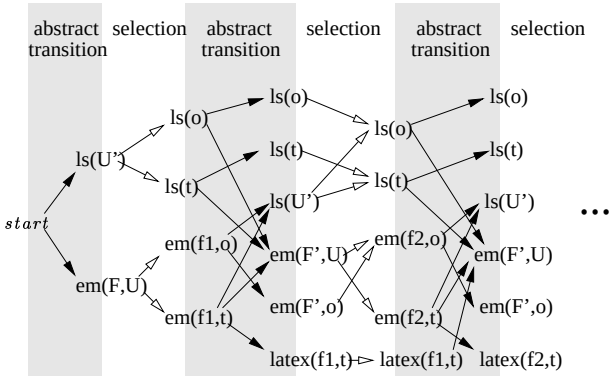

Figure 3: Illustration of the trellis induced by the LOHMM in Figure 1. In contrast with HMMs, there is an additional layer where the states are sampled from abstract states.

a LOHMM defines a random process with a unique probability distribution assuming that there is a finite set of abstract transitions and each domain associated to an argument of a predicate is finite. Thus, the set of concrete transitions is finite.

Having the analogy to HMMs in mind, there are three key problems of interest to be solved for LOHMMs. Let be $O = o_1, o_2, \ldots, o_T$ a finite sequence of ground observations:

Evaluation: Given a LOHMM, its parameter set λ, and a set μ of selection distributions, what is the probability $P(O \mid \mu, \lambda)$ that the sequence O was generated by the model?

Most likely state sequence: Given a LOHMM, its parameter set λ, a set μ of selection distributions, and an observation sequence O, find a state sequence S^* that is most likely to produce the observation sequence, i.e. $S^* = \arg\max_S P(S \mid O, \mu, \lambda)$.

Parameter estimation: Given the structure of an LOHMM, a set μ of selection distributions, and a set of observation sequences $\{O_t\}$, what is the most likely parameter set λ^* of the model, i.e. $\lambda^* = \arg\max_\lambda P(\lambda \mid \mu, \{O_t\})$.

Efficient ways to evaluate LOHMMs are adaptations of the *forward* and *backward pro-*

cedures. The main difference to the HMM case is that a transition is specified in two phases, as shown in Figure 3. After selecting an abstract transition, μ generates the relevant states from the head of the abstract transition.

Building the trellis in this way, it is easy to adapt the *forward- backward*, the *Viterbi* and the *Baum-Welch* algorithms for HMMs to LOHMMs. E.g. in the *forward* procedure, the α probabilities and are computed for each reachable state (sets S_t) at time t recursively. The $\alpha_t(s)$ is the probability of the partial observation sequence $o_1, \ldots, o_{t-1}$ and state s at time t given the LOHMM. The inductive definition of $\alpha_t(s)$ is as follows: Setting $\alpha_0(start) = 1.0$ the inductive formulae are $\alpha_0(start) = 1.0$ and

$$\alpha_t(h) = \sum_{cl} \sum_{b \in S_{t-1}} \alpha_{t-1}(b) P_{cl} P_\mu \delta(cl, b, h, o_t)$$

where $h \in S_t$, cl is an abstract transition in the LOHMM, P_{cl} is the transition probability of cl, and P_μ is the corresponding selection probability given by μ. The indicator function $\delta(cl, b, h, o_t) = 1$ whenever transition cl can take from state b to h observing o_t and the transition cl has the most specific body for b. The other algorithms can be adapted analogously. E.g. to upgrade the *Baum-Welch* algorithm, the expected counts of an abstract transition are basically the sum of the expected counts of its ground instances on the data – a situation similar to (Koller and Pfeffer, 1997; Kersting and De Raedt, 2001a). However, due to space restrictions, we will not explain here how to adapt the procedures.

To estimate the probabilities over the domains associated to predicates, i.e. to estimate the selection probabilities implicitly, we adapt the usual parameter learning of naïve Bayes models. The probabilities are the fraction of times the domain element was observed to occur over the total number of opportunities. This leads to the following overall EM scheme: In each iteration, estimate first the abstract transition probabilities and then the probabilities over the domains associated to each predicate.

Since our algorithm is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to (McLachlan and Krishnan, 1997), it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. We used e.g. pseudo-counts to ensure that no probabilities are estimated as zero. The computational complexity of parameter reestimation is $\mathcal{O}(k \cdot l \cdot g + d)$ where k is the number of sequences, l is the maximal length of a sequence, g is the number of (ground) transitions and d is the sum over the sizes of the domains.

4 Empirical Evaluation

Our objectives were to empirically show that (1) the basic algorithms work, (2) sampling is possible, (3) the parameter estimation is consistent, (4) LOHMMs easily scale up, and (5) LOHMMs are applicable to real world data.

Basic Algorithms We implemented [2] the forward and backward procedures, the Viterbi algorithm and the EM algorithm using the Prolog system Sicstus-3.8.6. Consider again the LOHMM in Figure 1. The observation sequence $emacs(lohmms.tex)$, $latex(lohmms.tex)$ has a probability 0.108, whereas $emacs(lohmms.tex)$, $emacs(lohmms.tex)$ has the probability 0.0576. The Viterbi path of the former observation was $emacs(lohmms.tex, tex)$, $latex(lohmms.tex, tex)$, $emacs(lohmms.tex, tex)$, i.e. it is most likely that a LaTeXuser typed in the commands. In contrast, it is more likely that some *other* user typed in the latter command sequence. Its Viterbi path was $emacs(lohmms.tex, others)$, $emacs(lohmms.tex, others)$, $ls(others)$.

Sampling We implemented *forward sampling*. The following two sequences were among a sample of 100 sequences of length 10 from our example LOHMM (where filenames were modeled using identifiers):

$$\{ls, ls, ls, emacs(f_1), ls, ls, emacs(f_2),$$
$$latex(f_2), emacs(f_3), latex(f_3)\},$$
$$\{ls, emacs(f_1), latex(f_1), emacs(f_1), latex(f_1),$$
$$emacs(f_1), latex(f_1), ls, emacs(f_2), ls\}.$$

[2] The software is available upon request from `kersting@informatik.uni-freiburg.de`.

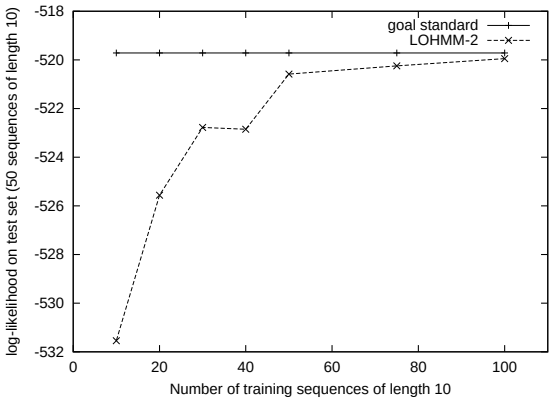

Figure 4: Learning curve of the example LOHMM assuming two filename and averaged over five restarts.

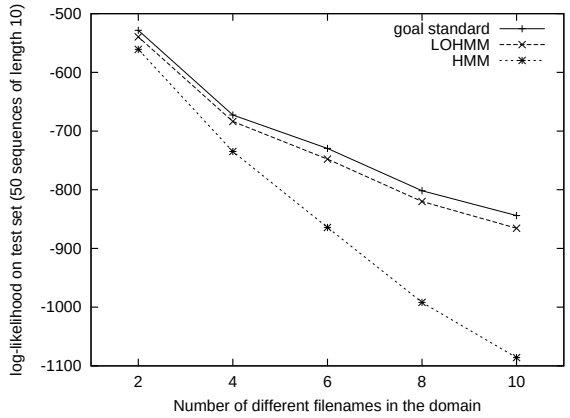

Figure 5: The performance of LOHMMs compared with HMMs with a varying number of filenames.

Parameter Estimation We did the following experiment on synthetical data. From the LOHMM in Figure 1 and a domain of 2 equally likely filenames, we sampled a test set of 50 sequences of length 10, and training sets of 10, 20, 30, 40, 50, 75 and 100 sequences of length 10. We ran both algorithms on each data set starting from 5 random initial set of parameters. We stopped when a limit of 30 iteration was exceeded or a change in log-likelihood was less than 10^{-1} from one iteration to the next. All learned models were evaluated on the test set. The learning curve in Figure 4 shows the expected result: the more training data, the better the learned model performed.

Scaling Up Figure 5 shows the results of another experiment. The parameters of both our LOHMM and its corresponding HMM for $2, 4, 6, 8$, and 10 different (uniformly distributed) filenames were estimated from a sampled training set of 20 sequences of length 10 and evaluated as before. The number of HMM parameters grows quadratically with the number of filenames ($56, 156, 304, 500, 744$), whereas the number of LOHMM parameters grows linearly ($14+2, 14+4, 14+6, 14+8, 14+10$) because we only had to change the domain representing the list of filenames. The performance of the estimated HMMs decreased with the number of filenames. More over, the HMMs had different transitions, whereas all LOHMMs had the same set of abstract transitions, and the transition probabilities of all of them (including the one using identifiers) were quite close to each other. Thus, the LOHMM structure represents a general rule holding in the data.

Real World Data Finally, we applied LOHMMs to real world data (for details, we refer to (Kersting et al., 2002)). We extracted a new dataset from the Protein Data Bank (PDB) and the SCOP (Structural Classification of Proteins) database. The resulting dataset contained the secondary structure of domains (of proteins) for five large fold classes (in total 2187 sequences). The goal was to discover structural characteristics of these selected folds. Our results indicate that LOHMMs are in fact applicable to the task: The number of parameters of our LOHMM is by an order of magnitude smaller than the number of an equivalent HMM (120 vs. approx. 62000), and the generalization performance, a 74% accuracy, is comparable to (Turcotte et al., 2001), a 75% accuracy, although the datasets were slightly different. Furthermore, we showed that it is easy to extract characteristic patterns from the trained LOHMMs. More precisely, we plotted the (selection) distribution associated to each argument. By visual inspection we were able to show differences among the folds. This is a feature of LOHMMs which HMMs do not have. But above all, the learning was quite fast. Our implementation took at

most 5 iterations and approx. 5 minutes per fold on a Pentium-III-600 MHz Linux machine.

5 Related Work

Hidden Markov models have been extended in a number of different ways. For example, in hierarchical HMMs (see e.g. (Fine et al., 1998)), states themselves may be composed of HMMs on a smaller scale. The difference is that the abstract states and transitions in our approach do not *consist* of more detailed states and transitions, but *summarize* states and transitions.

Factorial HMMs (Ghahramani and Jordan, 1997) are HMMs in which the hidden state variable is factored into k many state variables which depend on each other only through the observation. This can be seen as a LOHMM where the hidden state is summarized by a $2 \cdot k$-ary abstract state. The first k arguments encode the k state variables, and the last k arguments serve as a memory of the previous joint state. The selection distribution of the i-th argument is conditioned on the $i + k$-th argument.

Furthermore, LOHMMs are closely related to HMMs based on tree automata (see e.g. (Frasconi et al., 2002)). These approaches do not center logical concepts such as atoms and unification, and their graphical representation is not as close to HMMs as the one for LOHMMs.

Recently, Anderson *et. al* (Anderson et al., 2002) introduced *Relational Markov Models* (RMMs). Here, states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured. The main differences are that they do not facilitate variable binding and unification nor hidden states.

Finally, in recent years there has been an increasing interest in integrating probability theory with first order logic (Muggleton, 1996; Ngo and Haddawy, 1997; Jaeger, 1997; Friedman et al., 1999; Kersting and De Raedt, 2001b; Sato and Kameya, 2001). LOHMMs can be seen as an initial approach towards *downgrading* such highly expressive frameworks. This is important since such fragment are much easier to understand, adapt and refine, and are likely to lead towards efficient learning techniques. This is akin to contemporary considerations in *inductive logic programming* (Muggleton and De Raedt, 1994).

6 Conclusion and Future Work

In this paper, we have introduced logical Hidden Markov models (LOHMMs), a new Machine Learning technique combining probability theory and logic to probabilistically model sequences of logical atoms. LOHMMs offer the possibility to specify states and transitions in an abstract fashion, and thereby offer a significant reduction in the model size compared to regular HMMs. Our experiments have shown that the generalization performance is satisfactory, and that it is easy to extract characteristic patterns from the trained models. The abstraction enables us to learn even when the data is scarce, but abundant for conceptually similar states. Shrinkage is carried out over the abstract states.

In future research, we plan to apply LOHMMs on real world user modeling, and to explore additional applications. We are currently extending *inductive logic programming* (Muggleton and De Raedt, 1994) techniques to learn the (logical) structure of LOHMMs. Traditional HMM structure learning techniques seem not to be promising as they would not take advantage of the structured search space. So far, we adapted the traditional HMM problems only. It is interesting to compute the probability of non-ground observations and the most likely most abstract hidden state sequence given an observation sequence. The domains associated to arguments could be hierarchically structured. Furthermore, other representations of the selection distribution then naïve Bayes can be explored. E.g. Bayesian networks can express dependencies among arguments of abstract states.

Acknowledgments The authors would like to thank Stefan Kramer for his contributions to the biological experiments and Nico Jacobs for fruitful discussion on user modeling. This research was partly supported by the European Union IST programme under contract number

IST-2001-33053 (APRIL). T. Raiko was supported by a Marie Curie fellowship at DAISY, HPMT-CT-2001-00251.

References

C. R. Anderson, P. Domingos, and D. S. Weld. 2002. Relational Markov Models and their Application to Adaptive Web Navigation. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 143–152, Edmonton, Canada. ACM Press.

B. D. Davison and H. Hirsh. 1998. Predicting Sequences of User Actions. In *Predicting the Future: AI Approaches to Time-Series Analysis*, pages 5–12. AAAI Press.

T. Dean and K. Kanazawa. 1988. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-1988)*, pages 524–528.

S. Fine, Y. Singer, and N. Tishby. 1998. The hierarchical hidden markov model: analysis and applications. *Machine Learning*, 32:41–62.

P. Frasconi, G. Soda, and A. Vullo. 2002. Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, 18:195–217.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. 1999. Learning probabilistic relational models. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pages 1300–1307. Morgan Kaufmann.

Z. Ghahramani and M. Jordan. 1997. Factorial hidden Markov models. *Machine Learning*, 29:245–273.

N. Jacobs and H. Blockeel. 2001. The Learning Shell: Automated Macro Construction. In *User Modeling 2001*, pages 34–43.

M. Jaeger. 1997. Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 266–273. Morgan Kaufmann.

K. Kersting and L. De Raedt. 2001a. Adaptive Bayesian Logic Programs. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *LNAI*, pages 118–131. Springer.

K. Kersting and L. De Raedt. 2001b. Towards Combining Inductive Logic Programming with Bayesian Networks. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *LNAI*, pages 118–131. Springer.

K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. 2002. Towards discovering structural signatures of protein folds based on logical hidden markov models. Technical Report 175, University of Freiburg, Germany, June. (Short version to appear in the Proceedings of the *Pacific Symposium on Biocomputing 2003*).

D. Koller and A. Pfeffer. 1997. Learning probabilities for noisy first-order rules. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-1997)*, pages 1316–1321.

B. Korvemaker and R. Greiner. 2000. Predicting UNIX command files: Addjusting to user patterns. In *Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium*, pages 59–64.

J. W. Lloyd. 1989. *Foundations of Logic Programming.* Springer, Berlin, 2. edition.

G. McLachlan and T. Krishnan. 1997. *The EM Algorithm and Extensions.* Wiley, New York.

S. Muggleton and L. De Raedt. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679.

S. Muggleton. 1996. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press.

L. Ngo and P. Haddawy. 1997. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177.

L. R. Rabiner and B. H. Juang. 1986. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 3(1):4–16.

T. Sato and Y. Kameya. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454.

M. Turcotte, S. Muggleton, and M. J. E. Sternberg. 2001. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, 43(1/2):81–95.

R. J. Williams and D. Zipser. 1995. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In *Back-propagation:Theory, Architectures and Applications*, pages 433–486. Hillsdale, NJ: Erlbaum.