JMLR: Workshop and Conference Proceedings 1–15

PAutomaC: a PFA/HMM Learning Competition

Sicco Verwer SICCOVERWER@GMAIL.COM
Radboud University Nijmegen

Rémi Eyraud REMI.EYRAUD@LIF.UNIV-MRS.FR
QARMA team, Laboratoire d'Informatique Fondamentale de Marseille

Colin de la Higuera CDLH@UNIV-NANTES.FR
TALN team, Laboratoire d'Informatique de Nantes Atlantique, Nantes University

Abstract

Approximating distributions over strings is a hard learning problem. Typical techniques involve using finite state machines as models and attempting to learn these; these machines can either be hand built and then have their weights estimated, or built by grammatical inference techniques: the structure and the weights are then learnt simultaneously. The PAutomaC competition is the first challenge that will allow to compare methods and build a first state of the art for these latter techniques. Both artificial data and real data will be proposed and contestants are to try to estimate the probabilities of unseen strings.

1. Introduction, motivations, history and background

1.1. Why learn a probabilistic automaton?

Finite state automata (or machines) (see, e.g., [Sudkamp, 2006]) are well-known models for characterizing the behavior of systems or processes. They have been used for several decades in computer and software engineering to model the complex behaviours of electronic circuits and software such as communication protocols [Lee, 1996]. A nice feature of an automaton model is that it is easy to interpret, allowing one to gain insight into the inner workings of an system. In many applications, unfortunately, the original design of a system is unknown, for instance:

- The modeling of DNA or protein sequences in bioinformatics [Sakakibara, 2005].
- Finding patterns underlying different sounds for speech processing [Young., 1994].
- Developing morphological or phonological rules for natural language processing [Gildea and Jurafsky, 1996].
- In physics, unknown mechanical processes have to be modeled [Shalizi and Crutchfield, 2001].
- In robotics, the exact environment of robots is often unknown beforehand [Rivest and Schapire, 1993].
- Anomaly detection for detecting intrusions in computer security [Milani Comparetti et al., 2009].

- Behavioral modeling of users in applications ranging from web systems [Borges and Levene, 2000] to the automotive sector [Verwer et al.].
- State machine construction is often omitted during software system design [Walkinshaw et al., 2010]
- The structure of music styles for music classification and generation [Cruz-Alcázar and Vidal, 2008].

In all such cases, an automaton model is learned from observations of the system, i.e., a finite set of strings. Usually, the data gathered from observations is unlabeled. Indeed, it is often possible to observe only strings that can be generated by the system, and strings that cannot be generated are thus unavailable. The standard method of dealing with this situation is to assume a probabilistic automaton model, i.e., a distribution over strings. In such a model, different states can generate different symbols with different probabilities. The goal of automata learning is then one of model selection [Grünwald, 2007]: find the probabilistic automaton model that gives the best fit to the observed strings, i.e., that has the largest probability to have generated the data. This usually implies also to take the model size into account in order to avoid overfitting.

1.2. Which probabilistic automata to learn?

Several variants of probabilistic automata have been proposed in the past. An important recurring rule with respect to these variants is the fact that the better the machine is at modelling string distributions, the harder it is going to be to learn it. The best known variants are probabilistic finite state automata (Pfa) and Hidden Markov Models (Hmm) (see Figure 1):

- Pfa [Paz, 1971] are non-deterministic automata in which every state is assigned an initial and a halting probability, and every transition is assigned a transition probability (weight). The sum of all initial probabilities equals 1, and for each state, the sum of the halting and all outgoing transition probabilities equals 1. A Pfa generates strings probabilistically by starting in a state determined at random using the initial state distribution, either halting or executing a transition randomly determined using their probabilities, and iterating and generating the transition symbol in case it has not halted. A study of these automata can be found in [Vidal et al., 2005a, b].
- Hidden Markov models (Hmms)[1] [Rabiner, 1989; Jelinek, 1998] are Pfa where the symbols are assigned to states instead of transitions. In addition to the initial and halting probabilities, every symbol in every state is assigned a probability. The sum of these probabilities equals 1. A Hmm generates strings like a Pfa, with the only exception that in every state a symbol is generated according to the state symbol distributions instead of the executed transition.

1. We only consider discrete Hmms.

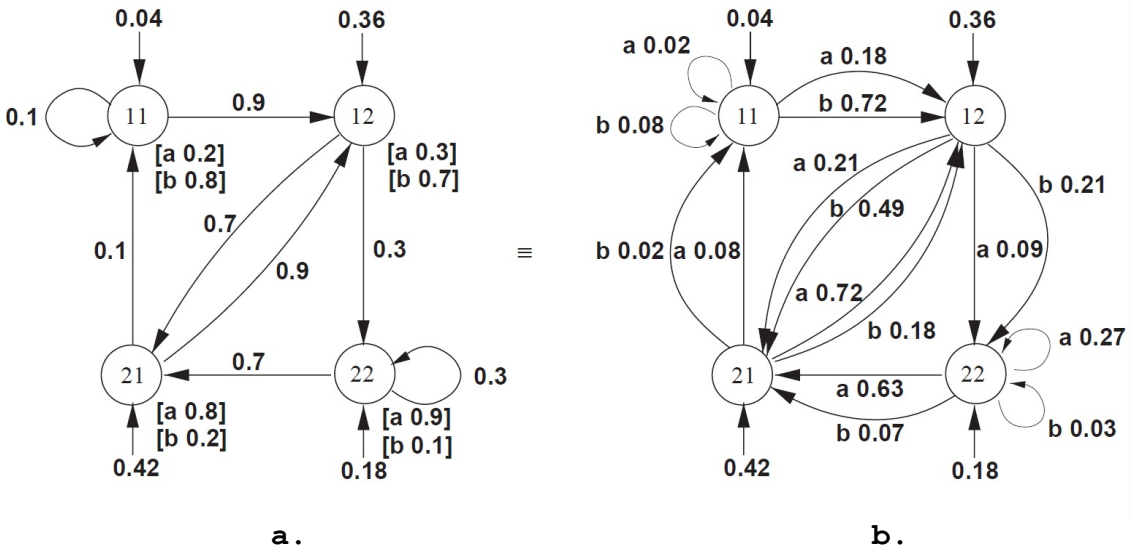

Figure 1: A HMM (a.) and a PFA (b.) that are equivalent: they correspond to the same probability distribution - this example is taken from Dupont et al. [2005].

Interestingly, although HMMs and PFA are commonly used in distinct areas of research, they are equivalent with respect to the distributions that can be modelled: an HMM can be converted into a PFA and vice-versa [Vidal et al., 2005a; Dupont et al., 2005].

Though it is easy to randomly generate strings from these models, determining the probability of a given string is more complicated because different executions can result in the same string. For both models, computing this probability can be solved optimally by dynamic programming using variations of the FORWARD-BACKWARD algorithm [Baum et al., 1970]. However, estimating the most likely parameter values (probabilities) for a given set of strings (maximizing the likelihood of model given the data) cannot be solved optimally unless P equals NP [Abe and Warmuth, 1992]. The traditional method of dealing with this problem is the BAUM-WELCH [Baum et al., 1970] greedy heuristic algorithm.

The deterministic counterpart of a PFA is a deterministic probabilistic finite automaton (DPFA) [Carrasco and Oncina, 1994]. These have been introduced for efficiency reasons essentially: in the non-probabilistic case, learning a DFA is provably easier than learning a NFA [de la Higuera, 2010]. However, although non-probabilistic deterministic automata are equivalent to non-probabilistic non-deterministic automata in terms of the languages they can generate, in [Vidal et al., 2005a, b; Dupont et al., 2005] it is shown that DPFA are strictly less powerful than PFA. Furthermore, distributions generated by PFA cannot be approximated by DPFA unless the size of the DPFA is allowed to be exponentially larger than the one of the corresponding PFA [Guttman et al., 2005; Guttman, 2006]. There is a positive side to this loss in power: estimating the parameter values of a DPFA is easy, and there exist algorithms that learn a DPFA structure in a probably approximately correct

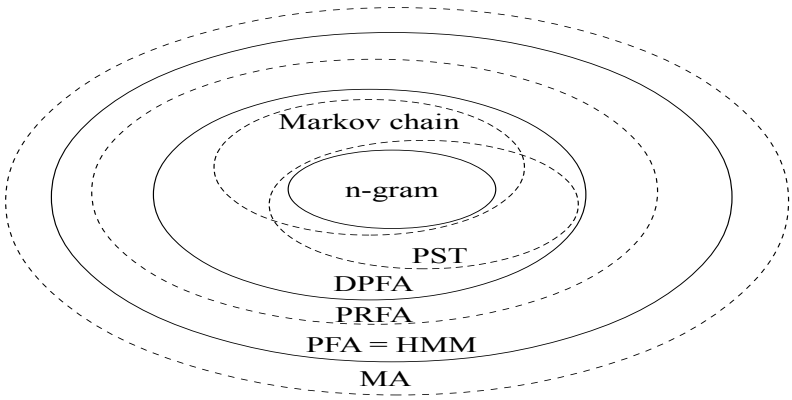

Figure 2: The hierarchy of the different finite states machines.

(PAC) like setting [Clark, 2004][2]. This is not known to be possible for Pfa or Hmms. For Pfa it has only been shown that they are strongly learnable in the limit [Denis and Esposito, 2004], or PAC-learnable using a (possibly exponentially larger) Dpfa structure [Gavaldà et al., 2006].

In additon to Pfa, Hmms, and Dpfa, other probabilistic finite state automata have been proposed such as: Markov chains [Saul and Pereira, 1997], n-grams [Ney et al., 1997; Jelinek, 1998], probabilistic suffix trees (Pst) [Ron et al., 1994], probabilistic residual finite state automata (Prfa)) [Esposito et al., 2002], and multiplicity automata (Ma) [Bergadano and Varricchio, 1996; Beimel et al., 2000] (or weighted automata [Mohri, 1997]). Although Markov chains and n-grams are a lot less powerful than Dpfa (both the structure and parameters are easy to compute given the data), they are very popular and often effective in practice. In fact, to the best of our knowledge, it is an open problem whether Pfa, Hmm, or Dpfa learners are able to consistently outperform n-gram models on prediction tasks. Probabilistic suffix trees are acyclic Dpfa that have a PAC-like learning algorithm [Ron et al., 1994]. Probabilistic residual finite state automata are more powerful than Dpfa, but less powerful than Pfa. Though multiplicity automata are more powerful than Pfa, they are also shown to be strongly learnable in the limit [Denis et al., 2006]. The power of the different types of probabilistic automata are summarized in Figure 2.

1.3. How to learn a probabilistic automaton?

Early work concerning the learning of distributions over strings can be found in [Horning, 1969] and [Angluin, 1988]. In the first case, the goal was to learn probabilistic context-free grammars; in the second, convergence issues concerning identification in the limit with probability one are studied. Although these initial studies were done decades ago, only two main techniques have become mainstream for learning Pfa or Hmms:

- The first is to take a standard structure or architecture for the machine, typically a complete graph, and then try to find parameter settings that maximize the likelihood

2. Learning a Dpfa structure in the pure PAC setting is impossible [Kearns et al., 1994].

of model given the data. If the structure is deterministic, the optimisation problem is quite simple [Wetherell, 1980]. If not, the standard method is the Baum-Welch algorithm [Baum et al., 1970; Baum, 1972]. This technique is known to be sensitive to initial probabilities and may get stuck in a local optimum.

- The second family of techniques relies on state-merging: the idea is to start with a very large automaton with enough states to describe the learning sample exactly, and then iteratively combining the states of this automaton in order to refine this model into a more compact one.

The three main state-merging algorithms for probabilistic automata that have had the largest impact were proposed in the mid-nineties: ALERGIA [Carrasco and Oncina, 1994], Bayesian model merging [Stolcke, 1994] and Learn-PSA [Ron et al., 1995]. The first deals with learning a Dpfa while the second tries to learn both the parameters and the structure of an Hmm. The third is learning probabilistic suffix trees.

Like the first technique, these are greedy algorithms that can get stuck in local optima. However, they do come with theoretical guarantees: probabilistic suffix trees can be PAC-learned [Ron et al., 1994], Dpfa have been proved to be learnable in the limit with probability 1 [Carrasco and Oncina, 1994], and more recently it has been shown that they can also be learned in a PAC setting [Clark, 2004].

Based on these three basic algorithms a number of refinements for state merging learning algorithms have been proposed:

- There have been several extensions of Alergia [de la Higuera and Thollard, 2000; Carrasco et al., 2001; de la Higuera and Oncina, 2003, 2004; Young-Lai and Tompa, 2000; Goan et al., 1996].

- Improvements of Ron et al. [1995] using their concept of distinguishable states can be found in [Thollard and Clark, 2004; Palmer and Goldberg, 2005; Guttman, 2006; Castro and Gavaldá, 2008]. An incremental version was proposed in [Gavaldà et al., 2006].

- Algorithm Mdi was introduced by Franck Thollard *et al.* [Thollard and Dupont, 1999; Thollard et al., 2000; Thollard, 2001]. This algorithms also uses state merging.

Another family of techniques has been developed, based on the identification of the residuals of a rational language:

- Esposito et al. [2002] approach has consisted in learning probabilistic residual finite state automata, that are the probabilistic counterparts to the residual finite state automata introduced by Denis et al. [2000, 2001].

- Denis et al. [2006] and Habrard et al. [2006] introduced the innovative algorithm Dees that learns a multiplicity automaton (the weights can be negative but each states weights sum to one) by iteratively solving equations on the residuals.

- Other algorithms using multiplicity automata have been developed, using common approaches in machine learning such as recurrent neural networks [Carrasco et al., 1996],

Principal Component Analysis [Bailly et al., 2009] or a spectral approach [Bailly, 2011].

A number of algorithms for learning probabilistic automata have been produced. Due to the difficulty of the learning problem, most of them focus on some form of Dpfa, Dees and Bayesian model merging being the exceptions. Another important approach is to learn Markov chains or n-grams by simply counting the occurrences of substrings. As already stated, these simple methods have been very successful in practice [Brill et al., 1998]. When one is faced with a data set made of strings and one needs to find a likely distribution over these strings for tasks such as for instance prediction, anomaly detection, or modeling, it would be very helpful to know which model is likely to perform best and why. Due to the lack of a thorough test of all of these techniques, this is currently an open question.

Furthermore, the facts that all known algorithms are of the greedy type and the recent successes of search-based approaches for non-probabilistic automaton learning [Heule and Verwer, 2010; Ibne et al., 2010] makes one wonder whether search-based strategies are also beneficial for probabilistic automaton learning. The PAutomaC Probabilistic Automaton learning Competition aims to answers both these questions by providing an elaborate test-suite for learning string distributions.

1.4. About previous competitions

There have been in the past competitions related with learning finite state machines or grammars.

- The first grammatical inference competition was organised in 1999. The participants of Abbadingo (http://www.bcl.cs.may.ie) had to learn Dfa of sizes ranging from 64 to 512 states from positive and negative data, strings over a two letter alphabet.
- A follow-up was system Gowachin (http://www.irisa.fr/Gowachin/), developed to generate new automata for classification tasks: the possibility of having a certain level of noise was introduced.
- The Omphalos competition (http://www.irisa.fr/Omphalos/) involved learning context-free grammars, given samples which in certain cases contained both positive and negative strings, and in others, just text.
- In the Tenjinno competition, the contestants had to learn finite state transducers (http://web.science.mq.edu.au/tenjinno/).
- The Gecco conference organised a competition involving learning Dfa from noisy samples (http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html).
- The Stamina competition (http://stamina.chefbe.net/), organised in 2010, also involved learning Dfa but new methods were used and permitted to solve even harder problems.
- The Zulu competition (http://labh-curien.univ-st-etienne.fr/zulu/) concerned the task of actively learning Dfa through requests to an oracle.

A number of other machine learning competitions have been organised during the past years. A specific effort has been made by the Pascal network (http://pascallin2.ecs.soton.ac.uk/Challenges/).

2. An overview of PAutomaC

PAutomaC is an on-line challenge and its webpages can be found at the address http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/.

The goal of PAutomaC is to provide an overview of which probabilistic automaton learning techniques work best in which setting and to stimulate the development of new techniques for learning distributions over strings. Such an overview will be very helpful for practitioners of automata learning and provide directions to future theoretical work and algorithm development. PAutomaC provides the first elaborate test-suite for learning string distributions.

Unlike previous automata learning competitions, in PAutomaC, the type of automaton device is not fixed: learning problems are generated using automata of increasing complexity. Since many different types of distributions are encountered in practice, this may be very useful for applications of automata learning. Furthermore, it gives an indication of the answer to the interesting question whether it is best to learn an Hmm or a Dpfa approximation when the data is generated using a Hmm and vice versa, see, e.g. the work of Gavaldà et al. [2006]. In PAutomaC, the participants know the type of machine that was used to generate the data and they are free to use different methods for different data sets. The team that achieves the best overall performance will be named the winner.

PAutomaC also encourages the development and use of new techniques from machine learning that do not build an automaton structure, but do result in a string distribution. Therefore, the structures of the learned automata is not be evaluated in PAutomaC. The performance of the different algorithms is based only on the quality of the resulting string distribution. Participants have access to seperate data sets for training (with strings repeated, these are multisets) and testing (no repetitions). Care is taken that the test set contains little information that can be used during training. The distance of the learned distribution from the true distribution over the testing strings is measured and used as a performance indicator. During the main phase of the competition, the only feedback the participants receive on their performance is their ranking in respect to the other participants.

There is no constraint on the computation time, except that an answer needs to be given before the competition has ended. Immediately after the competition, the winner is announced. Participants are encouraged to send in a short paper that can be presented at a special session of Icgi 2012.

2.1. About the data

There would be two types of data, artificial and real-world data donated by researchers and industries. The artificial data is generated by building a random probabilistic automaton of a given size and using this machine to generate data sets. Data will be generated using different types of automata of different sizes in order to discover the limits of the state-of-the-art approaches.

The automata are constructed completely at random, i.e., not biased towards a certain structure such as in the recent Stamina competition that was geared towards learning software models. The construction procedure takes the following parameters as inputs:

- The number of states N
- The size of the alphabet A
- The sparsity of symbols S_S (over all possible state-symbol pairs, only S_S percent of them are generated, chosen uniformly at random)
- The sparsity of transitions T_S (over all possible state-state-symbol triples, only T_S percent of them are generated, chosen uniformly at random)

The number of initial states is then $T_S * N$ and the number of final states is $S_S * N$. All initial, symbol, and transition probabilisties are drawn from a Dirichlet distribution. The final probabilities are drawn together with the symbol probabilities. One training and one test set are generated from every target. Repetitions are erased from the test set and resampled, again using the target automata. If the average length of the generated strings is outside of a predefined range, a new automaton and new data sets are generated. The complexity of the generated targets is tested using the baselines, which allows a selection of targets of different complexity. Four kinds of machines are generated: Markov Chains, Dpfa, Hmm and Pfa.

2.2. Baseline algorithms

Two simple baselines are provided in python code. The first one returns the frequency of each string of the test set evaluated on the elements of the train and test sets. The second one is a classic 3-gram computed also on both sets.

A more complex baseline is also provided: the ALERGIA algorithm [Carrasco and Oncina, 1994]. Although this algorithm is no longer state-of-the-art, many of the current state-of-the-art algorithms are based on the simple yet effective ALERGIA approach. It is therefore a very usefull starting point from which new techniques can be constructed. The algorithm is implemented in the OpenFST framework [Allauzen et al., 2007], an open source toolset for building finite state transducers (an extension of finite state automata).

2.3. Evaluation

One of the key issues to resolve is the possibility of collusion. For this reason, we aim to provide a test set that is (nearly) useless for training purposes. But this test set distribution has to be close to the one of the training set as we do not want the task to become a transfert learning problem. After discussion with the members of the scientific committee, we chose to generate the test set using the target automata and to remove duplicable strings. The fact that the size of training sets is much higher than the one of the test set should imply that informations about the target distribution contained in the test set are likely to be of little use in learning.

A difficult issue when measuring the performance of distributions over strings is the one of smoothing. Smoothing is the process of moving probabilistic mass from the frequent strings in a learned model to the infrequent ones [Chen and Goodman, 1996]. Most

importantly, this assigns non-zero probabilities to string that the model believes cannot occur. Using the traditional evaluation metrics such as perplexity [Cover and Thomas, 1991], smoothing is a necessity: the penalty of giving zero probability to an occurring string is infinite. Therefore, it is currently unknown whether smoothing is essential for a model to be vaild or if it is only necessary because of the used evaluation metrics. For n-gram learning, smoothing is very often used and sophisticated methods such as backoff smoothing exist [Zhai and Lafferty, 2004]. For Dpfa learning, smoothing techniques can be found in [Dupont and Amengual, 2000; Thollard, 2001; Habrard et al., 2003]. It is an open question how to smooth Pfa and Hmm models.

In PAutomaC, we try to avoid the smoothing issue by providing the used test-cases. Knowing that these cases should all receive a non-zero probability, it would be unwise to assign a zero probabilty to one of them. In addition, we ask the competitor to normalize their submitted probabilities such that they sum to one.

The evaluation measure is a perplexity one. Given a test set TS, it is given by the formula is:

$$Score(C, TS) = 2^{-(\sum_{x \in TS} Pr_T(x) * log(Pr_C(x)))}$$

where $Pr_T(x)$ is the normalize probability of x in the target and $Pr_C(x)$ is the candidate probability for x submitted by the participant.

Notice that this measure is closed to the well-known Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951] in our case. Indeed, given to distribution P and Q, the KL divergence is defined by $KL(P,Q) = \sum_x P(x)log(P(x)/Q(x))$ which can be rewritten into $KL(P,Q) = (-\sum_x P(x)logQ(x)) - H(P)$ where $H(P)$ is the entropy of the target distribution. $H(P)$ is constant in our case since the aim is to compare various candidate distributions Q. As we are only interested in the divergence on a given test set TS, the only varying element of the KL divergence is then $-\sum_{x \in TS} P(x)logQ(x)$ which is equivalent to our measure, up to a monotonous transformation.

2.4. Progress of the competition

The real competition phase takes place starting May 2012, but the first training data sets and the baseline algorithm are available since the beginning of 2012. From February until the end of the competition, teams can register for PAutomaC and test their algorithms using the testing dataset and the PAutomaC test server.

During the first phase of the competition, from February to May, the server provides a detailed feedback for each submission: in addition to the rank of this submission among all previous one, the score of the submission and the one of the baselines is provided. This should allow participants to develop and fine-tune their methods.

When the competition starts on May 20th, the test server is closed and the competition server is opened. The competition server is identical to the test server but provides only the ranking of the submission for that problem instance. When the competition ends, the leader of the overall ranking will be announced as the winner. The winning team does not necessarily have to be the leader in any of the individual problem instances.

The competition timeline is:

- December: competition is announced

- Mid-February: first data sets are produced and baseline algorithm is made available
- 1st March: the test server is opened
- 20th May: competition server is opened
- 30th June: competition is over
- 20th July: short papers are submitted
- 5th-8th September: special session takes place at ICGI

Acknowledgement

The authors would like to thank the members of the scientific committee for their constructive comments and suggestions that have really helped us to design this competition.

References

N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning Journal*, 9:205–260, 1992.

C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. `http://www.openfst.org`.

D. Angluin. Identifying languages from stochastic examples. Technical Report Yaleu/Dcs/RR-614, Yale University, March 1988.

R. Bailly. Qwa: Spectral algorithm. *Journal of Machine Learning Research - Proceedings Track*, 20:147–163, 2011.

R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In L. Bottou and M. Littman, editors, *Proceedings of the 26th* ICML, pages 33–40, Montreal, June 2009. Omnipress.

L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

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

A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the* Acm, 47(3):506–530, 2000.

F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. Siam *Journal of Computing*, 25(6):1268–1280, 1996.

J. Borges and M. Levene. Data mining of user navigation patterns. In B. Masand and M. Spiliopoulou, editors, *Web Usage Mining and User Profiling*, number 1836 in Lncs, pages 92–111. Springer-Verlag, 2000.

E. Brill, R. Florian, J. C. Henderson, and L. Mangu. Beyond n-grams: Can linguistic sophistication improve language modeling. In *In Proc. of COLING-ACL-98*, pages 186–190, 1998.

R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications, Proceedings of* Icgi *'94*, number 862 in Lnai, pages 139–150. Springer-Verlag, 1994.

R. C. Carrasco, M. Forcada, and L. Santamaria. Inferring stochastic regular grammars with recurrent neural networks. In L. Miclet and C. de la Higuera, editors, *Proceedings of* Icgi *'96*, number 1147 in Lnai, pages 274–281. Springer-Verlag, 1996.

R. C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning Journal*, 44(1):185–197, 2001.

J. Castro and R. Gavaldá. Towards feasible PAC-learning of probabilistic deterministic finite automata. In A. Clark, F. Coste, and L. Miclet, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'08*, volume 5278 of Lncs, pages 163–174. Springer-Verlag, 2008.

S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics (ACL '96)*, pages 310–318. Association for Computational Linguistics, 1996.

A. Clark. Learning deterministic context free grammars: the Omphalos competition. Technical report, 2004.

T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, New York, NY, 1991.

P. Cruz-Alcázar and E. Vidal. Two grammatical inference applications in music processing. *Applied Artificial Intelligence*, 22(1–2):53–76, 2008.

C. de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

C. de la Higuera and J. Oncina. Identification with probability one of stochastic deterministic linear languages. In R. Gavaldà, K. Jantke, and E. Takimoto, editors, *Proceedings of* Alt *2003*, number 2842 in Lncs, pages 134–148. Springer-Verlag, 2003.

C. de la Higuera and J. Oncina. Learning probabilistic finite automata. In G. Paliouras and Y. Sakakibara, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'04*, volume 3264 of Lnai, pages 175–186. Springer-Verlag, 2004.

C. de la Higuera and F. Thollard. Identication in the limit with probability one of stochastic deterministic finite automata. In de Oliveira [2000], pages 15–24.

A. L. de Oliveira, editor. *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'00*, volume 1891 of Lnai, 2000. Springer-Verlag.

F. Denis and Y. Esposito. Learning classes of probabilistic automata. In J. Shawe-Taylor and Y. Singer, editors, *Proceedings of* Colt *2004*, volume 3120 of Lncs, pages 124–139. Springer-Verlag, 2004.

F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using non deterministic finite automata. In de Oliveira [2000], pages 39–50.

F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSA. In N. Abe, R. Khardon, and T. Zeugmann, editors, *Proceedings of* Alt *2001*, number 2225 in Lncs, pages 348–363. Springer-Verlag, 2001.

F. Denis, Y. Esposito, and A. Habrard. Learning rational stochastic languages. In *Proceedings of* Colt *2006*, volume 4005 of Lncs, pages 274–288. Springer-Verlag, 2006.

P. Dupont and J.-C. Amengual. Smoothing probabilistic automata: an error-correcting approach. In de Oliveira [2000], pages 51–62.

P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, 2005.

Y. Esposito, A. Lemay, F. Denis, and P. Dupont. Learning probabilistic residual finite state automata. In P. Adriaans, H. Fernau, and M. van Zaannen, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'02*, volume 2484 of Lnai, pages 77–91. Springer-Verlag, 2002.

R. Gavaldà, P. W. Keller, J. Pineau, and D. Precup. Pac-learning of markov models with hidden state. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proceedings of* Ecml *'06*, volume 4212 of Lncs, pages 150–161. Springer-Verlag, 2006.

D. Gildea and D. Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22:497–530, 1996.

T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *Proceedings of* Aaai *Spring Symposium on Machine Learning in Information Access*, Stanford, CA, 1996. Aaai Press.

P. Grünwald. The minimum description length principle, 2007. MIT Press.

O. Guttman. *Probabilistic Automata and Distributions over Sequences*. PhD thesis, The Australian National University, 2006.

O. Guttman, S. V. N. Vishwanathan, and R. C. Williamson. Learnability of probabilistic automata via oracles. In Jain et al. [2005], pages 171–182.

A. Habrard, M. Bernard, and M. Sebban. Improvement of the state merging rule on noisy data in probabilistic grammatical inference. In N. Lavrac, D. Gramberger, H. Blockeel, and L. Todorovski, editors, *Proceedings of* Ecml *'03*, number 2837 in Lnai, pages 169–1180. Springer-Verlag, 2003.

A. Habrard, F. Denis, and Y. Esposito. Using pseudo-stochastic rational languages in probabilistic grammatical inference. In Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, and E. Tomita, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'06*, volume 4201 of Lnai, pages 112–124. Springer-Verlag, 2006.

M. Heule and S. Verwer. Exact dfa identification using sat solvers. In J. M. Sempere and P. García, editors, *ICGI*, volume 6339 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2010. ISBN 978-3-642-15487-4.

J. J. Horning. *A study of Grammatical Inference*. PhD thesis, Stanford University, 1969.

A. Hasan Ibne, A. Batard, C. de la Higuera, and C. Eckert. Psma: A parallel algorithm for learning regular languages. In *NIPS Workshop on Learning on Cores, Clusters and Clouds*, 2010.

S. Jain, H.-U. Simon, and E. Tomita, editors. *Proceedings of* Alt *2005*, volume 3734 of Lncs, 2005. Springer-Verlag.

F. Jelinek. *Statistical Methods for Speech Recognition*. The Mit Press, Cambridge, Massachusetts, 1998.

M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *STOC*, pages 273–282, 1994.

S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1): 79–86, 1951.

S. Lee. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1996.

P. Milani Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *IEEE Symposium on Security and Privacy*, 2009.

M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(3):269–311, 1997.

H. Ney, S. Martin, and F. Wessel. *Corpus-Based Statiscal Methods in Speech and Language Processing*, chapter Statistical Language Modeling Using Leaving-One-Out, pages 174–207. S. Young and G. Bloothooft, Kluwer Academic Publishers, 1997.

N. Palmer and P. W. Goldberg. Pac-learnability of probabilistic deterministic finite state automata in terms of variation distance. In Jain et al. [2005], pages 157–170.

A. Paz. *Introduction to probabilistic automata*. Academic Press, New York, 1971.

L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the* Ieee, 77:257–286, 1989.

R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.

D. Ron, Y. Singer, and N. Tishby. Learning probabilistic automata with variable memory length. In *Proceedings of* Colt *1994*, pages 35–46, New Brunswick, New Jersey, 1994. Acm Press.

D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of* Colt *1995*, pages 31–40, 1995.

Y. Sakakibara. Grammatical inference in bioinformatics. 27(7):1051–1062, 2005.

L. Saul and F. Pereira. Aggregate and mixed-order Markov models for statistical language processing. In C. Cardie and R. Weischedel, editors, *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89. Association for Computational Linguistics, Somerset, New Jersey, 1997.

C.R. Shalizi and J. P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of statistical Physics*, 104:817–879, 2001.

A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. Ph.D. dissertation, University of California, 1994.

A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.

F. Thollard. Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *Proceedings 8th International Conference on Machine Learning*, pages 561–568. Morgan Kauffman, 2001.

F. Thollard and A. Clark. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004.

F. Thollard and P. Dupont. Entropie relative et algorithmes d'inférence grammaticale probabiliste. In M. Sebag, editor, *Actes de la conférence* Cap *'99*, pages 115–122, 1999.

F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic Dfa inference using Kullback-Leibler divergence and minimality. In *Proceedings of the 17th International Conference on Machine Learning*, pages 975–982. Morgan Kaufmann, San Francisco, CA, 2000.

S. Verwer, M. de Weerdt, and C. Witteveen. Learning driving behavior by timed syntactic pattern recognition. In *IJCAI'11*.

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite state automata – part I. *Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005a.

E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite state automata – part II. *Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005b.

N. Walkinshaw, K. Bogdanov, C. Damas, B. Lambeau, and P. Dupont. A framework for the competitive evaluation of model inference techniques. In *Proceedings of the First International Workshop on Model Inference in Testing*, pages 1–9, 2010.

C. S. Wetherell. Probabilistic languages: a review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

Tzay Y. Young. *Handbook of Pattern Recognition and Image Processing: Computer Vision*, volume 2. Academic Press, 1994.

M. Young-Lai and F. W. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning Journal*, 40(2):111–137, 2000.

C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22:179–214, 2004. ISSN 1046-8188.