PAutomaC: a probabilistic automata and hidden Markov models learning competition

Sicco Verwer, Rémi Eyraud, Colin de La Higuera

Machine Learning manuscript No.
(will be inserted by the editor)

PAutomaC: a PFA/HMM Learning Competition

Sicco Verwer · Rémi Eyraud · Colin de la Higuera

Received: 16 December 2012 / Accepted: 9 August 2013

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 learned simultaneously. The Probabilistic Automata learning Competition (PAutomaC), run in 2012, was the first grammatical inference challenge that allowed the comparison between these methods and algorithms. Its main goal was to provide an overview of the state-of-the-art techniques for this hard learning problem. Both artificial data and real data were presented and contestants were to try to estimate the probabilities of strings. The purpose of this paper is to describe some of the technical and intrinsic challenges such a competition has to face, to give a broad state of the art concerning both the problems dealing with learning grammars and finite state machines and the relevant literature. This paper also provides the results of the competition and a brief description and analysis of the different approaches the main participants used.

S. Verwer
Institute for Computing and Information Sciences, Radboud University Nijmegen.
E-mail: siccoverwer@gmail.com,

R. Eyraud
QARMA team, Laboratoire d'Informatique Fondamentale de Marseille.
E-mail: remi.eyraud@lif.univ-mrs.fr

C. de la Higuera
TALN team, Laboratoire d'Informatique de Nantes Atlantique, Nantes University.
E-mail: cdlh@univ-nantes.fr

1 Introduction

This paper describes the PAutomaC probabilistic automaton learning competition and provides an overview of the relevant literature on this topic. PAutomaC was an on-line challenge that took place in 2012 at `http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/`. The goal of PAutomaC was 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. Many probabilistic automata learning methods have been produced in the past (see Section 2 for an overview). Most of these focus on deterministic probabilistic automata (Dpfa), where only the symbols are drawn from probability distributions but the transitions are uniquely determined given the generated symbol. There exist some exceptions, however, which aim to learn hidden Markov models [6], probabilistic residual automata [31], and multiplicity automata [26]. Another important approach is to learn Markov chains or n-grams by simply counting the occurrences of sub-strings [66, 56, 46]. These simple counting methods have been very successful in practice [12].

Although many methods have been proposed, there has been so far no thorough investigation of which model/algorithm is likely to perform best, why and when. Knowledge about this would be very helpful to scientists/practitioners faced with a data set made of strings and the problem of finding a likely distribution over these strings. PAutomaC aimed to fill this knowledge gap by providing the first elaborate test-suite for learning string distributions.

In addition to being very helpful for applications of automata learning methods, PAutomaC was designed in such a way that it provided directions to future theoretical work and algorithm development. For instance, unlike previous automata learning competitions (see Section 2.4 for details), in PAutomaC, the type of automaton device was not fixed: learning problems were generated using automaton models of increasing complexity. This is not only very useful for practical applications (where many different types of distributions can be encountered), but also aims to answer to the interesting question whether it is best to learn a non-deterministic model (e.g. Hmm) or a deterministic model (e.g. Dpfa) when the data is drawn from a (non-)deterministic distribution,as described for instance in the work of Gavaldà *et al.* [33]. PAutomaC also encouraged 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 actual structures of the learned automata were not evaluated in PAutomaC. Instead, the performance of the different algorithms were tested only on the quality of the resulting string distribution. Like previous automaton learning competitions, this evaluation was performed on-line using a test set and an evaluation oracle running on the competition server. Consequently, the participants could use the observed performance (and that of the competition) to update their algorithms.

The competition setup in PAutomaC contained some novel elements that may also be of interest for competitions of other (string) distribution learning algorithms. Above all, in PAutomaC the performance was evaluated using the actual probabilities assigned by a learned distribution, instead of the more traditional method of evaluating its predictive performance. This has the advantage of not only testing whether the high probability events are assigned the largest probabilities, but also whether the low probability events are assigned the correct low

probabilities. Furthermore, the actual strings that were being used for this evaluation were given to the participants beforehand.

The traditional approach to compare language models, which had also been considered for PAutomaC, is to test the learned model over some unseen data. Perplexity [19] is the usual measure and, in order to perform well on such a metric, it is necessary to learn a smoothed model, in which a non-null probability is assigned to all strings (the penalty is infinite otherwise). Experience shows that in that case, the smoothing method may become preponderant: the quality of the model can rely mainly on the smoothing. Another issue with such an evaluation task is that the model has to be checked somehow for consistency, since the probabilities of all possible strings must sum up to one.

The goal of PAutomaC being to compare learning algorithms (and not smoothing algorithms), a different protocol was chosen: the teams knew the test set in advance, and part of the problem for them consisted in reassigning the mass of probabilities the learned model used for the strings absent from the test set to those strings inside this set. In this way, a perplexity-like evaluation measure could be used to evaluate the differences in the probabilities assigned to different strings from the test-set. A couple of possible dangers of this protocol were identified by the PAutomaC Scientific Committee[1] and, later, by the participants. A first one was that the extra information in the test set (which was also randomly drawn from the unknown target distribution) could be used to learn. A second danger came from the fact that the teams could submit various solutions to the same problem (with no feedback about their score, but knowing their overall standing): this could have allowed some hill-climbing strategy. Both the Scientific Committee's analysis and the attempts by some participants showed that the PAutomaC evaluation process was resistant: the winning team is actually the one who submitted the least times. We detail in this paper the choices that were made to handle these dangers.

As main contributions of this paper we provide an overview of the literature on probabilistic automaton learning, and describe PAutomaC including its design issues and solutions. The results of the competition and the approach followed by the main participants are also provided. There is a clear winner to PAutomaC: a novel collapsed Gibbs sampling method for Pfa developed by team Shibata-Yoshinaka. As it is not common to use such a method when learning distributions over strings, we hope and expect this result will influence what people will use in practice. In addition to having an appealing winner, we can draw several interesting conclusions by analyzing the results. In particular, it can be observed that the Alergia-based method developed by team Llorens outperforms the winning team on the deterministic instances. This provides some additional insights into the important question whether it is better to learn deterministic or non-deterministic models and can serve as an important starting point for further researches on this topic. Furthermore, we analyze the PAutomaC results with the goal of determining when which method works best and why. Our analysis indicates the problem areas for each of the used methods, which forms a basis for future studies and hopefully further improvements of the used methods. Last but not least, all methods developed by the participating teams significantly outperform the provided

[1] http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/committee.php

baseline algorithms, clearly demonstrating the need for developing and evaluating (new) methods for learning string distributions.

This paper is organized in six sections: introduction (Section 1), motivations and history (Section 2), an overview of PAutomaC (Section 3), final results (Section 4), a brief description and analysis of the approaches used by main participants (Section 5), and a conclusion (Section 6).

2 Motivations and history

We assume the reader to be familiar with the theory of languages and automata [70], their probabilistic counterparts such as hidden Markov models [60], and basic concepts from computational complexity [65], computational learning theory [48], and information theory [19]. For more information on these topics the reader is referred to the corresponding references.

2.1 Why learn a probabilistic automaton?

Finite state automata (or machines) 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 behaviors of electronic circuits and software such as communication protocols [52]. A nice feature of an automaton model is that it is easy to interpret, allowing one to gain insight into the inner workings of a system. In many applications, unfortunately, the original design of a system is unknown. This is the case for instance when one wants to:

- model Dna or protein sequences in bioinformatics [64],
- find patterns underlying different sounds for speech processing [81],
- infer morphological or phonological rules for natural language processing [35],
- model unknown mechanical processes in physics [67],
- discover the exact environment of robots [61],
- detect anomaly for intrusion detections in computer security [53],
- do behavioral modeling of users in applications ranging from web systems [11] to the automotive sector [75],
- discover the structure of music styles for music classification/generation [20].

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, that is to say that 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 [37]: find the probabilistic automaton model that gives the best fit to the observed strings, i.e., that is most likely to have generated the data. In addition to the data probability, this implies that the model size has to be taken into account in order to avoid over-fitting. Otherwise, the model that generates only the seen strings and whose probabilities correspond to the observed frequency perfectly achieves the goal. But this naive

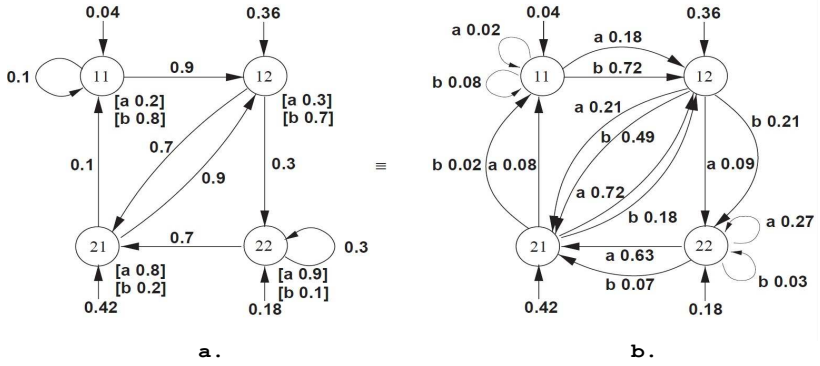

Fig. 1 An Hmm (a.) and a Pfa (b.) that are equivalent: they correspond to the same probability distribution - this example is taken from [30].

model is of little use: it assigns null probability to all unseen strings and therefore makes no generalization.

2.2 Which probabilistic automata to learn?

Several variants of probabilistic automata have been proposed in the past. An important and obvious recurring rule with respect to these variants is the fact that the better the machine is at modeling 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 [59] 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 [77, 78].
- Hidden Markov models (Hmms)[2] [60, 46] are Pfa (as described in the previous paragraph) where the symbols are emitted at the states instead of at the transitions which are only used to move. Initial probabilities are assigned to each state but there are no final probabilities, defining therefore a distribution over Σ^n for each value of n. In order to obtain a distribution over Σ^* a special halting symbol or state can be introduced. With such an addition an Hmm generates strings like a Pfa.

Interestingly, although Hmms and Pfa are commonly used in distinct areas of research, they are equivalent with respect to the distributions that can be modeled: an Hmm can be converted into a Pfa and vice-versa [77, 30]. Though it is

[2] We only consider discrete Hmms.

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 (or Backward) algorithm [7]. However, estimating the most likely parameter values (probabilities) for a given set of strings and a given model (maximizing the likelihood of model given the data) cannot be solved optimally unless RP equals NP [1]. The traditional method of dealing with this problem is the Baum-Welch [7] greedy algorithm.

The deterministic counterpart of a Pfa is a deterministic probabilistic finite automaton (Dpfa) [14]. These have been introduced for efficiency reasons essentially: in the non-probabilistic case, learning a Dfa is provably easier than learning a Nfa [21]. However, although non-probabilistic deterministic automata are equivalent to non-probabilistic non-deterministic automata in terms of the languages they can generate, it is shown in [77, 78, 30] 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 [39, 38]. 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 (Pac) like setting [18][3]. 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 [25], or Pac-learnable (under some restrictions) using a (possibly exponentially larger) Dpfa structure [33].

In addition to Pfa, Hmms, and Dpfa, other probabilistic finite state automata have been proposed such as: Markov chains [66], n-grams [56, 46], probabilistic suffix trees (Pst) [62], probabilistic residual finite state automata (Prfa) [31], and multiplicity automata (Ma) [9, 8] (or weighted automata [54]). Probabilistic *accepting* automata will use weights to assign probabilities to individual strings, whose meaning is the probability of accepting (Vs not accepting) the string. Such automata are also sometimes called fuzzy automata. 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 [62]. 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 [26]. The expressiveness power of the different types of probabilistic automata is summarized in Figure 2.

2.3 How to learn a probabilistic automaton?

Early work concerning the learning of distributions over strings can be found in [43] and [2]. 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 1 are studied. Although these initial studies were done decades

[3] The hardness of Pac-learning the structure of a Dpfa is shown in [47].

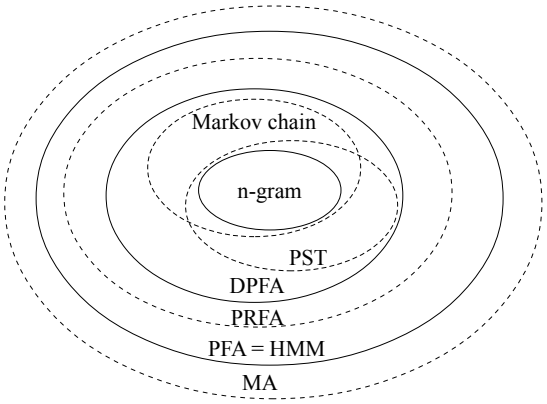

Fig. 2 The hierarchy of the different finite states machines. Multiplicity automata (MA) can model the most distributions (but can also model other functions), n-grams are the least expressive.

ago, only three techniques have become mainstream for learning Pfa, Hmms, and Dpfa.

Parameter estimation. The first family of techniques takes a standard structure or architecture for the machine, typically a complete graph, and then tries to find parameter settings that maximize the likelihood of model given the data. If the structure is deterministic, the optimization problem is quite simple: transition probabilities can be estimated using the maximum likelihood [80]. If not, the standard method is the Baum-Welch algorithm [7, 6] which iteratively computes a new estimate for the transition probabilities using the probabilities assigned to the input data. Although this technique is known to be sensitive to initial probabilities and may get stuck in a local optimum, it has frequently been applied successfully in practice.

Bayesian inference. The second family of techniques correspond to Bayesian methods such as Gibbs sampling [34], see, e.g., [55, 32]. Instead of learning a single model (a point[4] estimate), these methods aim to make predictions using the joint distribution formed by all possible models. This joint distribution is hard to compute and an Hmm Gibbs sampler estimates it by iteratively sampling the visited hidden states conditioned on earlier samples of all other state visits. The stationary distribution of the thus formed Markov chain is exactly this joint distribution. Although these methods are not yet commonplace for Pfa, we believe this is likely to change after this competition.

State-merging. Learning Dpfa typically relies on the technique of state-merging (see, e.g., [21]): 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.

[4] One instance of a model is a point in the space of all possible models.

The three main state-merging algorithms for probabilistic automata that have had the largest impact were proposed in the mid-nineties:

- Alergia [14] by Carrasco and Oncina,
- Bayesian model merging [69] by Stolcke, and
- Learn-PSA [63] by Ron, Singer, and Tishby.

The first deals with learning a Dpfa while the second tries to learn both the parameters and the structure of an Hmm. The third learns 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 [62], Dpfa have been proved to be learnable in the limit with probability 1 [14], and more recently it has been shown that they can also be learned in a Pac-like setting [18]. 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 [24, 15, 22, 23, 82, 36].
- Improvements of Ron *et al.* [63] based on the concept of distinguishable states have been developed [72, 57, 38, 16]. An incremental version also exists [33].
- Algorithm Mdi was introduced by Thollard *et al.* [73, 74, 71]. This algorithm also uses state merging.
- Recently, they have been extended to learn not only the distribution over strings of events/symbols but also over their timing behaviors [76] and from a continuous stream of data instead of a data set [5].

Other methods. Several other methods have been proposed that have not yet become mainstream, most notably:

- Esposito *et al.*'s [31] approach has consisted in learning probabilistic residual finite state automata based on the identification of the residuals of a rational language. These are the probabilistic counterparts to the residual finite state automata introduced by [27, 28].
- Denis *et al.* [26] and Habrard *et al.* [41] introduced the innovative algorithm Dees that learns a multiplicity automaton (the weights can be negative but in each state the weights sum to one) by iteratively solving equations on the residuals.
- Other algorithms learning multiplicity automata have been developed, using common approaches in machine learning such as recurrent neural networks [13], Principal Component Analysis [4] or a spectral approach [3].

Most of these methods estimate the model parameters based on maximum likelihood. This can cause problems when computing probabilities, especially for strings of low frequency. For some of these methods, therefore, smoothing methods have been developed that adjust the maximum likelihood estimate in order to hopefully overcome these difficulties [17]. Typically, these smoothing methods assign larger probabilities to infrequent strings, and consequently, less to more frequent ones. For n-gram learning, smoothing is very often used and sophisticated methods such as back-off smoothing exist [83]. For Dpfa learning, smoothing techniques can be found in [29, 71, 40]. Smoothing Pfa and Hmms is still a question requiring further research.

In conclusion, many 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. Another important approach is to learn Markov chains or n-grams by simply counting the occurrences of sub-strings. As already stated, these simple methods have been very successful in practice [12]. 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 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 [42, 45] makes one wonder whether search-based strategies are also beneficial for probabilistic automaton learning. The Probabilistic Automaton learning Competition (PAutomaC) aims to answers these questions by providing an elaborate test-suite for learning string distributions.

2.4 About previous competitions

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

- The first grammatical inference competition was organized in 1999. The participants of Abbadingo (`http://abbadingo.cs.nuim.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 organized 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/`), organized 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.
- The Rers Grey Box Challenge (`http://leo.cs.tu-dortmund.de:8100/isola2012`) aimed to discover the complementary values of white-box and black-box software system analysis techniques, including tools for learning finite state machines.

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

3 An overview of PAutomaC

The goal of PAutomaC was 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. In order to stimulate this development, PAutomaC was set up using an oracle server that was able to evaluate the submissions by participants on-line. Furthermore, in contrast to the traditional methods used to evaluate predictive machine learning algorithms, the performance in PAutomaC was evaluated using the actual probabilities assigned by a learned distribution.

Two types of data were available: artificial and real-world data donated by researchers and industries. But we have to admit that the latter were after all of little interest in the context of the competition. The problem came from the fact that not knowing the targeted probabilities implies a biased way to evaluate them. We chose to use 3-grams trained on the complete data sets to fix these probabilities, hoping that the induced bias would be drastically reduced since the competition sets consisted of less than 10% of these data. Unfortunately, this goal was not achieved since the participants who scored the best on these data sets used n-grams (even when they were using more complex approaches on the artificial data sets). We will thus not discuss the real-world data sets in the rest of this paper (detailed information is available on the website).

In this section, we first describe the way the targets automata were generated. We then turn our attention on how the submissions of the participants were evaluated. Finally we discuss the choices made all along this process.

3.1 Generating artificial data

Artificial data was generated by building random probabilistic machines with 5 to 75 states and with an alphabet consisting of 4 to 24 symbols (both inclusive, and decided uniformly at random). These machines were subsequently used to generate data sets. Of all possible state-symbol pairs that could occur in transitions, between 20 and 80 percent (the symbol sparsity) of them were generated. These pairs were selected by first choosing a state at random, and subsequently choosing a symbol from the set of symbols that had not yet been selected for that state. This created a selection without replacement from the set of all possible state-symbol pairs that was modified to remain uniform over the states. This modification made it less likely that the resulting symbols were evenly distributed over the states. For every generated state-symbol pair, one transition was generated to a randomly chosen target state. Between 0 and 20 percent (the transition sparsity) transitions were generated in addition to these, selected without replacement from the set of possible transitions, modified to remain uniform over the source states and transition labels.

Initial and final states were selected without replacement until the percentages of selected states exceeded the transition and symbol sparsities, respectively. All initial, symbol, and transition probabilities were drawn from a Dirichlet distribution with concentration parameters set to 1 (making every probability distribution equally likely). The final probabilities were drawn together with the symbol probabilities.

From such a structure, one training set (with repetitions) and one test set (without repetitions) were generated from every target. With probability one out of four, the generated train set was of size 100 000, it was of size 20 000 otherwise. New test strings were generated using the target machine until 1 000 unique strings had been generated. The test strings were allowed to overlap with the strings used for training. If the average length of the generated strings was less than 5 or greater than 50, a new automaton and new data sets were generated using the same construction parameters. In total, 150 models and corresponding train and test sets were generated using this way. We evaluated the difficulty of the generated sets using a 3-gram baseline algorithm: the problem was considered easy if the baseline output was close to the target (a perplexity difference of less than 1.0), and difficult otherwise. We then selected 16 of them, aiming to obtain ranging values for the number of states, the size of the alphabet, sparsity values, and difficulty. We applied the same procedure for DPFA but without generating additional transitions; and for HMMS, we generated state-state pairs instead of state-symbol-state triples.

In total, this results in 48 (16 for every type) artificially generated problems for use in the competition. The participants were given no other information about the target than the two files of strings (one for the training set and one for the test set). The format of these files is given in Figure 3.

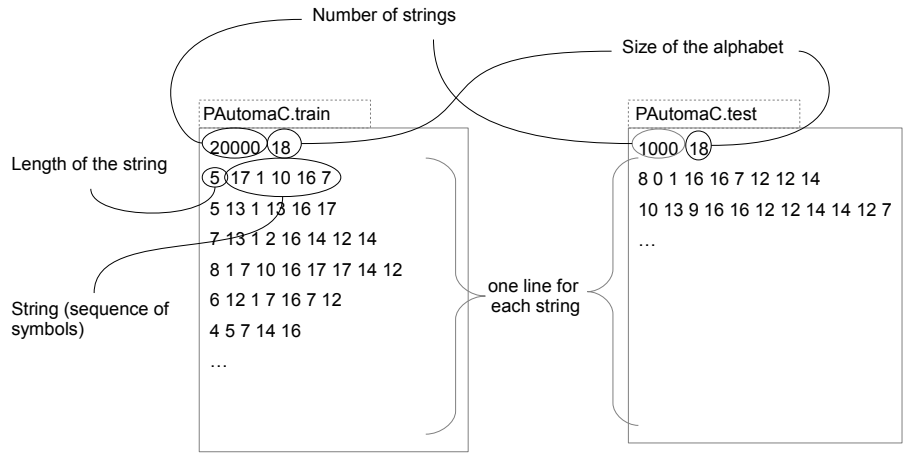

Fig. 3 Format of the files made available to the participants of the PAUTOMAC challenge.

3.2 Evaluation

The evaluation measure was based on perplexity. Given a test set S, it was defined by the formula:

$$Score(C, S) = 2^{-\sum_{x \in S} P_{\mathrm{T}}(x) * \log_2(P_{\mathrm{C}}(x))}$$

where $P_{\mathrm{T}}(x)$ is the normalized probability of x in the target and $P_{\mathrm{C}}(x)$ is the normalized candidate probability for x submitted by the participant. The normalization process is the usual one when perplexity is considered: it consists in modifying the probabilities so that they sum to 1 on the set S. A consequence of this normalization was that adding probability to one of the test strings removed probability from the others. Therefore, this perplexity score measured how well the differences in the assigned probabilities matched with the target probabilities.

Notice that this measure is equivalent to the well-known Kullback-Leibler (**KL**) divergence [50]. Indeed, given two distributions P and Q, the **KL** divergence is defined as $\mathbf{KL}(P,Q) = \sum_x P(x)\log_2(P(x)/Q(x))$ which can be rewritten into $\mathbf{KL}(P,Q) = (-\sum_x P(x)\log_2 Q(x)) - H(P)$ 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 were only interested in the divergence on a given test set S, the only varying element of the **KL** divergence is $-\sum_{x\in S} P(x)\log_2 Q(x)$ which is equivalent to our measure, up to a monotonous transformation.

To decide the final overall rank of each participant, points were attributed for each data set: the leader of a problem at the end of the competition scored 5 points, the second 3, the third 2 and the fourth 1. In case of equality on a problem (based on the first 10 digits of the perplexity score), the earliest submission won. The winner is the participant whose overall score was the highest. There was no restriction on the number of submissions a given participant could provide, but no feedback was given about the resulting perplexity. To compute the final score of a participant, only the best submission to each problem was considered.

3.3 Discussion on the design of the competition

When organizing an on-line competition, one has to make various choices about the generation of data and the evaluation of the participant submissions. We described above what was done for PAutomaC but we feel that the choices that were made have to be discussed. What follows thus contains arguments about the validity of our approach and therefore of the results of the competition.

Target generation. As already stated, we used a Dirichlet distribution for sampling the output probabilities. The main advantage of this method is that every possible distribution is equally likely when sampled using a Dirichlet distribution (with concentration parameters set to 1). Notice that this does not happen when every output probability is iteratively sampled uniformly at random. Since we did not intend to bias the distribution in PAutomaC towards certain types of distributions, using the Dirichlet distribution seemed the logical choice.

If we were to sample all output probabilities from a Dirichlet distribution unconstrained, however, we would obtain a very densely connected Pfa with high probability. Such densely connected automata are uninteresting from a learning perspective: a simple one-gram will already reach a close to optimal perplexity. We therefore constrained this sampling using symbol and transition sparsity values. These two values were preselected and the generated Pfa was then forced to match these sparsity values. Afterwards, we sampled the transition probabilities for every state using a Dirichlet distribution.

The Pfa structure generator worked by iteratively adding new transitions until the preselected sparsity values were reached. This selection remained uniform over all states, lowering the probability that every state gets assigned the same number of symbols and transitions. The generator initialized by adding to every state one random symbol and one random transition for that symbol. This avoided the generation of states with a final probability of 1.0, i.e., sink states. This was done because we aimed for the final probability generation to be independent of the structure generation.

The final probability of each state was handled as the emission of a special symbol: this allowed a simple normalization process and did not influence the bias over distributions since their values were sampled together with the output probabilities using a Dirichlet distribution. Together with the consistency test (see below), this ensured that the generated machines corresponded to a proper distribution (probabilities over all possible strings summed to 1). The selection of which states had final probabilities, however, was performed independently of the process used to select output transitions. This ensured that having more output symbols does not lead to lower final probabilities.

An important step took place directly after the generation of a target. It consisted in checking that all states were reachable from an initial state and that they were all co-accessible. Indeed, verifying the consistency of the machine ensured that we did not have a path (and thus a probability mass) that reached a part of the machine that never led to an accepting state. In addition, we tested whether the generated probability distribution did not result in giving too much weight to long or short strings. Although this created some bias in the generation procedure, it was unavoidable because testing the different methods on instances that are too difficult or uninteresting makes no sense.

Evaluation. As already stated, the choice of an evaluation function that does not rely on a particular type of machine was a fundamental requirement of PAutomaC. Using a perplexity measure had the interest to be a widely accepted way to compare distributions and its link with the **KL** divergence was clearly a plus. Though we did not inform the participants about it, we also computed two other evaluation functions for each submission: the max-norm (maximal difference between the submitted probabilities and the target ones) and the sum-norm (the sum of the differences between the submitted and the target probabilities). While on a few problems the ranking of the participants was a bit different than the one obtained with the official perplexity measure, the overall ranking of the teams was the same.

A common issue when dealing with string distributions is smoothing. When using perplexity as a measure, smoothing becomes necessary because strings with zero probability obtain an infinite **KL** divergence when compared to the target (or any other non-zero assigning distribution) and thus an infinite perplexity. Although smoothing can be very beneficial in practice, we feel that the standard perplexity measure is too dependent on smoothing (compared to the max-norm, for instance) and therefore that a perplexity evaluation based on an unseen test set does not properly measure the quality of the string distribution. In PAutomaC we therefore decided to provide the participating teams with knowledge of the actual strings used to compute the perplexity measure. This removed the need for specialized smoothing methods since the participants could simply use a minimum value for the probability assigned to any string.

Collusion. A usual problem with on-line competitions is the one of the possibility of collusion. Indeed, a set of test data has to be made available to participants in order to evaluate the performance of their algorithm with respect to a given target. But if this set contains information about the target, then it can be used during the learning phase and may bias the results. In a competition where the targets are not stochastic devices, this problem is usually tackled by the requirement that elements of the test set do not occur in the train set (though they are generated by the same process). But this cannot be ensured when the aim is to learn a distribution as both sets have to be generated using the target: erasing elements of the test set that occur in the train set generates an important bias in the distribution of the test set. We therefore chose to keep these elements, expecting that the difference in size between the train and test sets sufficed to make the information contained in the test set useless.

But collusion can also result from the fact that the test set by itself contains information about the target distribution: duplicate strings are likely to be frequent in the target distribution. This is why we decided to remove redundant elements of the test sets, creating a small bias in the distribution of these sets. However, since the actual target distribution was used during the evaluation, and thanks to the choices made for this phase, this did not result in a bias or other problems during evaluation.

4 Results

4.1 Competition activity

38 participants registered to have access to the problem sets and 16 of them submitted at least one of their solutions to a problem. There were a total number of 2 787 submissions during the competition. 5 participants managed to score some points, 4 of them were ranked first at least once (see Figure 4).

During the competition phase, the website received 724 visits (with a maximum of 54 the last day of the competition) from 196 unique visitors with an average visit duration of a bit more than 5 minutes. IPs from 37 countries have been detected, between which 14 countries corresponded to 5 or more visitors.

4.2 Overall results

The final scores can be seen in Figure 4 and detailed results are presented in Table 1 (available in Appendix). There is a clear winner of PAutomaC: team Shibata-Yoshinaka. Of all participants, they obtained the best perplexity values on most instances and performed well on all others. This result is validated by the computation of other performance indicators (the max-norm and sum-norm). From Table 1 it can be observed that the method implemented by team Shibata-Yoshinaka really works well for all of the competition problems: the difference between the perplexity values of the solutions and their submissions was never greater than 0.1. Furthermore, this difference was even smaller on the instances with 100 000 strings, indicating that they made good use of additional data.

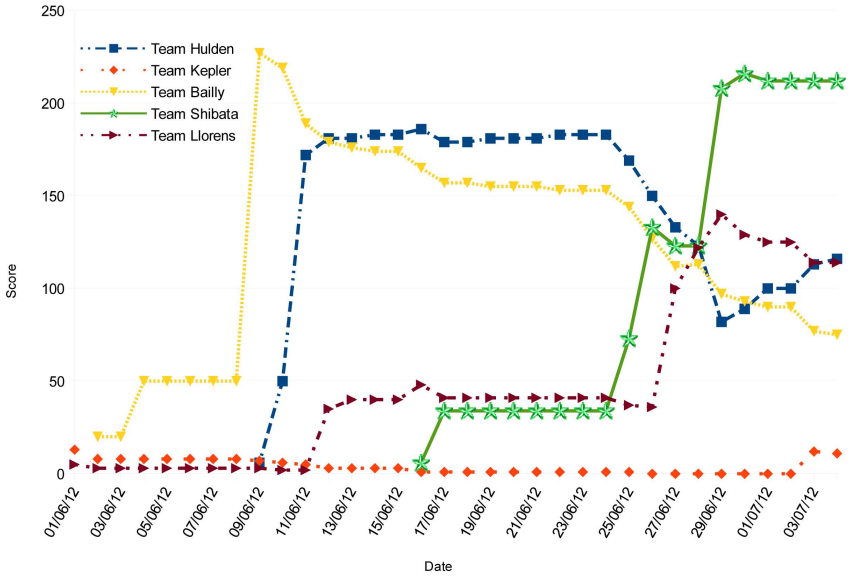

Fig. 4 Overall evolution of the score of the 5 leading teams (artificial data sets). For each problem, 5 points was given to the team whose best submission had the smallest perplexity, 3 points to the second best team, 2 for the third and 1 for the fourth.

4.3 Analysis of the results

In PAutomaC, the different approaches were tested on problem instances with a broad range of parameter values and coming from different probabilistic automaton models (see Table 2 in Appendix). This makes it possible to perform some additional analysis of the results with the goal of discovering when each method works best and and trying to understand why. Tables 1 and 2 (both in Appendix) clearly show that team Shibata-Yoshinaka is only outperformed on the (nearly) deterministic ones (Dpfa, or Pfa/Hmm with a small transition sparsity). On these instances team Llorens performs slightly better. Team Hulden's method also manages to obtain the best perplexity values on two instances, and actually beats team Llorens overall performance by just 2 points (rightmost points in Figure 4). Their method seems to perform best on dense instances with few states. The methods used by team Bailly and team Kepler have some difficulties with very sparse instances (and thus also with Dpfa), and perform well but not best on the other instances.

We further analyzed the results using a standard decision tree learning[5] for two prediction tasks:

1. Predicting the winner given the problem instance parameter values.
2. Predicting whether a deterministic distribution was used to generate the problem instance given the winner.

[5] rpart, implemented in R.

The resulting decision trees are depicted in Figure 5. Interestingly, although team Shibata-Yoshinaka performs well on all problem instances, they are outperformed by team Llorens on sparse problem instances. Sparse instances are generated using an automaton that contains only a tiny fraction of all possible transitions given the number of states and the alphabet size (see Section 3.1). Since deterministic automata are fixed to use such a fraction, most of these automata are deterministic. This is confirmed by the second prediction task, which indicates that when team Llorens performs best there is an 80% chance that the generator is deterministic. This result is very interesting since team Shibata-Yoshinaka and team Hulden used methods based on non-deterministic automaton models, while team Llorens used deterministic models (see Section 5). Of course, we cannot be sure that the used model or the used method is important when predicting the type of generator, but it seems to indicate that it is best to learn a non-deterministic model when the data is drawn from a non-deterministic distribution, and that it is best to learn a deterministic model when the data is drawn from a deterministic distribution. In fact, this result also shows that it is possible to detect whether a given set of strings is drawn from a deterministic or non-deterministic generator (the second tree in Figure 5): use team Shibata-Yoshinaka's, team Huldens's, and team Llorens's methods to learn a predictor, test their performance on a validation set, return the type of model used by the best performing method. Such a method has several interesting applications like evaluating possible discretization of values coming from an abstract deterministic generator. In the next section, we provide some detailed descriptions and individual analyses for each of the methods.

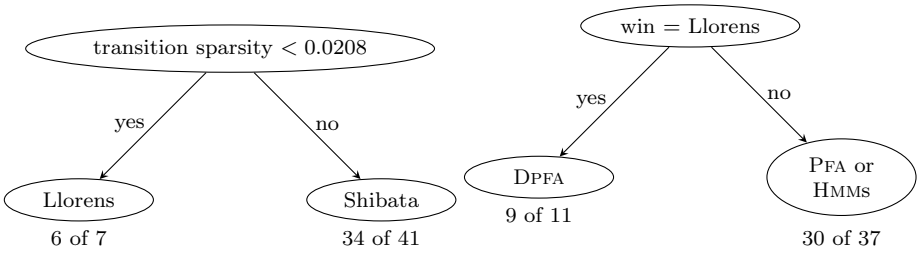

Fig. 5 Decision trees predicting the winner given the parameters of a problem instance (left), and whether a deterministic or non-deterministic generator was used given who won (right).

5 The different approaches and individual results

A wide spectrum of learning approaches has been used during the competition. We describe in this section the ones of the main participants - those who scored at least a point - and provide a small detailed analysis of their performance in PAutomaC. This section is the result of deep discussions and electronic exchanges the authors had with the different teams. However, the overview presented here is superficial and the reader is therefore referred to the original paper describing the team's work.

5.1 Team Shibata-Yoshinaka

Shibata and Yoshinaka [68] used a Gibbs sampling method to estimate the probability $Pr(\mathbb{b}|\mathbb{a})$ of a future sentence $\mathbb{b}$ given training data $\mathbb{a}$ generated by an unknown Pfa. The probability that a Pfa generates a sentence $\mathbb{a} = a_1 \dots a_T$ by passing states $\mathbb{z} = z_0 \dots z_T$ in this order is given as

$$Pr(\mathbb{a}, \mathbb{z} \mid \xi) = \prod_{1 \le t \le T} \xi_{z_{t-1} a_t z_t} = \prod_{i,a,j} \xi_{iaj}^{C_{iaj}} , \tag{1}$$

where ξ_{iaj} is the probability of the state change from i to j with a letter a and C_{iaj} counts the times when that transition occurs. Applying Gibbs sampling directly to ξ is somewhat tricky. For instance, it requires one to continuously compute new state sequences, see, e.g., [32]. Therefore, they first marginalize ξ out from Eq. (1) under the assumption that the prior of ξ is a Dirichlet distribution. Intuitively, this computes the sum of all possible values of Eq. (1) for every ξ multiplied by the probability of that ξ. Although this is a very large sum to compute, under the assumption the ξ is Dirichlet distributed, many terms cancel out making the resulting computation easy. This technique is called *Collapsed Gibbs Sampling*, see, e.g., [10].

Shibata and Yoshinaka sample different values $\mathbb{z}^{(1)}, \dots, \mathbb{z}^{(S)}$ for $\mathbb{z}$ independently from the resulting distribution by Gibbs sampling, i.e., by iteratively sampling from $Pr(z_t \mid \mathbb{a}, z_0 \dots z_{t-1} z_{t+1} \dots z_T)$. The exact values of $\widetilde{\xi}^{(1)}, \dots, \widetilde{\xi}^{(S)}$ are then simply the expectation based on the state transition history:

$$\widetilde{\xi}_{iaj}^{(s)} = \mathrm{E}\left[\xi_{iaj} \,\middle|\, \mathbb{a}, \mathbb{z}^{(s)}\right] = \frac{C_{iaj} + \beta}{C_i + AN\beta} ,$$

where N is the (maximum) number of states of the target Pfa, A is the size of the letter alphabet and β is the smoothing parameter (the prior).

In the actual implementation[6], they have fixed the number of iterations of CGS and sampling points a priori. The values of N and β were determined by 10-fold cross validation amongst a finite number of candidates. Finding good settings for these values required quite some computational resources.

Analysis. The result of learning a decision tree that aims to predict the performance of team Shibata-Yoshinaka given the problem parameters (unknown to the participants during the competition) is shown in Figure 6. In the learned tree, we can clearly observe that the collapsed Gibbs sampling approach of team Shibata-Yoshinaka performs best when there are many (100k) strings available for training, or when the target contains few (< 21) states. Moreover, in the other cases, it still finds distributions close to the optimal one (with an average perplexity difference of 0.0467).

5.2 Team Hulden

The inference approach of Mans Hulden [44] used three strategies:

[6] A version of their algorithm is available at `http://www.iip.ist.i.kyoto-u.ac.jp/member/ry/pfai/`

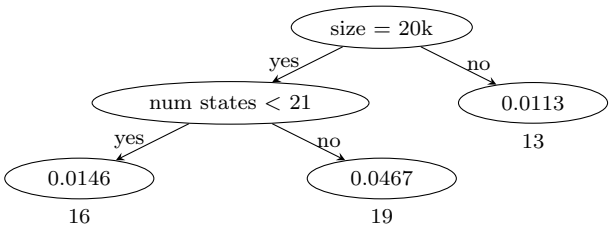

Fig. 6 Decision tree predicting the performance (perplexity difference with the solution) of team Shibata-Yoshinaka given the parameters of a problem instance.

1. A basic "baseline" n-gram strategy with smoothing.
2. Another "baseline" n-gram strategy without smoothing, but using interpolated test data.
3. The construction of a fully connected PFA inferred with BAUM-WELCH (EM), each between 5 and 40 states in size. Training was done using only the original training data, and separately also using reconstructed training data, as in (2).

In the first strategy, the n-gram counts were extracted from the training data for various values of n (between 2 and 9). Then, the log likelihood of the training data was calculated and the n yielding the highest log likelihood was used to issue the probabilities to the test strings for submission. Witten-Bell smoothing (see, e.g., [17]) was used in all cases.

For the second approach, the n chosen in the first one was used to decide the optimal window size to use for n-grams. In this strategy, the test data was used for training as well, and was augmented in an iterative fashion. This because the original test data represented a skewed distribution as duplicates had been removed. First, the expected number of occurrences of each string in the test set was calculated based on the total number of occurrences of that string in the training and test sets. Based on this expected number, a fractional count of strings was "added" to the test data, reflecting a guess that the original test data had contained these duplicates. This process was repeated until convergence (when the expected string count in the test data no longer changed). These counts were then used for calculating the probabilities of each string in the test data.

For the third strategy, three randomly initialized PFA of 5, 10, 20, and 40 states were trained with BAUM-WELCH for each problem, after which the one with the highest log likelihood was submitted (several results in case of approximate ties). Similarly to the n-gram case, another three runs for each state size were made using both training and reconstructed test data. However, contrary to the n-gram strategy, using reconstructed test data for training failed to ever improve on the basic BAUM-WELCH that used only the PAUTOMAC training data for training.

The n-gram solutions were submitted early and the EM solutions later. This allowed the observation, based on the server feedback, that EM outperformed the n-grams in most cases (roughly 85% of problems). A notable exception is the two real data problems where the interpolated n-grams performed best in each case. As mentioned, using reconstructed test data for training helped in the n-gram strategy, but not with BAUM-WELCH, probably because of severe over-fitting.

Analysis. The tree predicting the performance of team Hulden's Baum-Welch/EM approach is depicted in Figure 7. Their method performs best on dense problems (transition sparsity > 0.0215), and excels when the target contains not too many states (< 35). Overall, the performance is close to the one of the winning team. From personal communication, we discovered that the amount of computing power used by team Shibata-Yoshinaka's method is much more than that of team Hulden's. Unfortunately, the influence on the results of the computational resources could not be measured, nor was it a criterion for the competition itself.

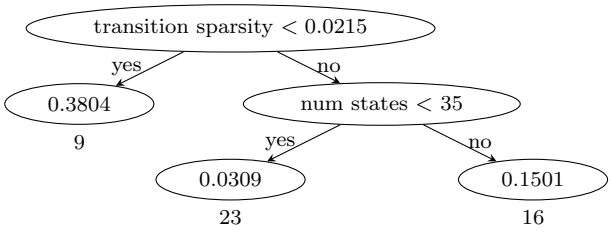

Fig. 7 Decision tree predicting the performance (perplexity difference with the solution) of team Hulden given the parameters of a problem instance.

5.3 Team Llorens

The approach followed by the Llorens team was two-fold: on one hand, they upgraded the Alergia algorithm [14] by using ideas from evidence-driven approaches to state merging. Specifically, they computed all possible merges in a red-blue framework (see [51]), and performed the one that passed the most statistical tests, which are computed using Hoeffding's bound as in Alergia. The second line they followed was to work on the fact that the test data was known and that there could be a better strategy than the simple normalization to make probabilities sum to 1 on the test set.

Analysis. The tree predicting the performance of team Llorens's Alergia-based approach is depicted in Figure 8. This tree is quite interesting because the state merging approach adopted by team Llorens is very different from the first two approaches. First of all, the root decision shows that their method performs best on target distributions with a small alphabet (< 18). An interesting question is whether this can be linked to the known problems of state merging methods for non-probabilistic automata on large alphabets [79]. Secondly, from this tree it is very clear that the type of generating distribution has a large effect on the performance. In particular, it confirms that learning a Dpfa works best when the generating distribution is a Dpfa. Interestingly, in the non-Dpfa case, it performs better on dense problems (transition sparsity > 0.0795). This seems to indicate that learning dense non-deterministic distributions is easier (in terms of perplexity) than learning sparse ones, even when a deterministic model is learned.

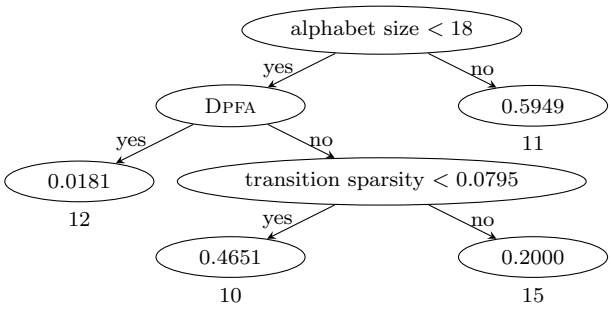

Fig. 8 Decision tree predicting the performance (perplexity difference with the solution) of team Llorens given the parameters of a problem instance.

5.4 Team Bailly

Team Bailly tackled the competition by using a spectral approach (see [3]). The main component which is manipulated is the Hankel matrix [58], representing the counts for every possible prefix-suffix pair. The core of the spectral technique is the Hankel matrix factorization, from which the parameters of a probabilistic model can be directly deduced.

Analysis. Team Bailly approached the competition using a new and promising method for learning probability distributions over strings. This emphasizes the interest in determining when their algorithm performs well. The PAutomaC data clearly shows when this is the case (see Figure 9). Although their method performed well on many instances (32), and was leading the competition for a long time (see Section 4.2), their performance shows large drops on sparse problem instances (transition sparsity < 0.0428). All methods have some trouble with sparse problems, but significantly less than team Bailly's spectral approach. Future research is needed to try to determine exactly why the spectral approach has so much trouble with these instances, the PAutomaC data and generator remain available for this purpose.

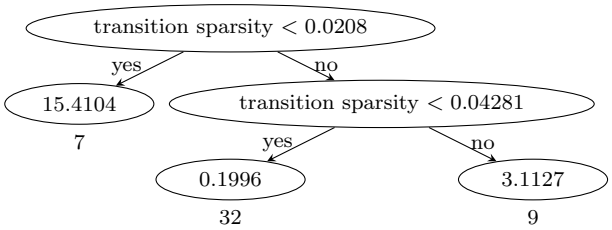

Fig. 9 Decision tree predicting the performance (perplexity difference with the solution) of team Bailly given the parameters of a problem instance.

5.5 Team Kepler

The approach applied by the team Kepler [49] uses n-gram models with variable length. n-grams are represented as a context tree that maps the probabilities of sequences of symbols. To shrink the state space while working with large n-grams, the context tree is pruned based on the Kullback-Leibler divergence. Experiments showed that this approach almost always achieves lower perplexity than the fixed 3-gram model on the PAutomaC training data. However, it is not clear how to define the maximum size of the n-gram or the pruning threshold value.

Analysis. Team Kepler's method shows the same behavior with sparse instances than team Bailly. However, it has in addition trouble learning distributions coming from generators with a small alphabet. Again, future research will need to point out why this happens. It is surprising to see that although the relatively simple n-gram based approach adopted by team Kepler are very popular in practice, it did not perform as well as the other more complex approaches to learning string distributions on the PAutomaC data.

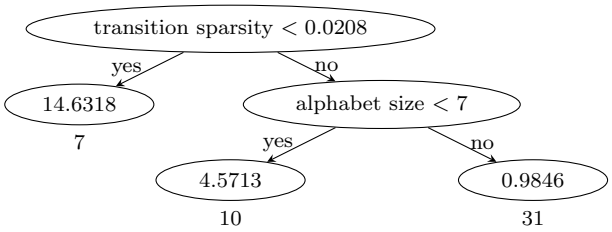

Fig. 10 Decision tree predicting the performance (perplexity difference with the solution) of team Kepler given the parameters of a problem instance.

6 Conclusion

We presented an overview of PAutomaC, the relevant literature on learning probabilistic automata, a brief explanation of the methods used during the competition, and an analysis of their results. The results of PAutomaC presented in this paper indicate that the competition was fruitful:

- There were 5 active participating teams from around the world.
- All participants used different (both old and new) methods and were stimulated to improve these. All methods performed much better than the provided baseline algorithms.
- The PAutomaC data set provides a detailed comparison of the performance of each of these methods.
- There is a clear winner, and interestingly, they used a method that is in practice not (yet) commonly applied when learning Pfa.
- The results remain valid using different evaluation criteria.
- Interesting conclusions can be drawn by analyzing the results.

In particular, the observation that team Llorens outperforms the winning team on the deterministic instances is very interesting for future research as it could provide a method for deciding whether a given data sample is drawn from a deterministic distribution or from a non-deterministic one. This could be very useful during the discretization of data, for instance. Moreover, it would be very interesting to further investigate and hopefully improve the performance of the spectral and n-gram based approaches developed by team Bailly and team Kepler on sparse problem instances. Last but not least, new Gibbs sampling and EM/Baum-Welch methods have been developed for Pfa by team Shibata-Yoshinaka and Team Hulden. Based on their excellent performance in PAutomaC, we can encourage anyone interested in learning probability distributions over strings to use one of these methods. The developed Gibbs sampler performed consistently better in PAutomaC, but required much more computational resources. When the generating distribution is known to be deterministic, we advise a state merging approach such as the one developed by team Llorens.

Acknowledgements We are very thankful to the members of the scientific committee for their help in designing this competition. We want to thank all participants and in particular Raphael Bailly, Cleo Billa, Mans Hulden, Fabio Kepler, David Llorens, Sergio Mergen, Shihiro Shibata, and Ryo Yoshinaka for their help during the writing of this paper.

References

1. N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning Journal*, 9:205–260, 1992.
2. D. Angluin. Identifying languages from stochastic examples. Technical Report Yaleu/Dcs/RR-614, Yale University, March 1988.
3. R. Bailly. QWA: Spectral algorithm. *Journal of Machine Learning Research - Workshop and Conference Proceedings, ACML'11*, 20:147–163, 2011.
4. R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of ICML'09*, pages 33–40. Omnipress, 2009.
5. B. Balle, J. Castro, and R. Gavaldà. Bootstrapping and learning PDFA in data streams. *Journal of Machine Learning Research - Workshop and Conference Proceedings, ICGI'12*, 21:34–48, 2012.
6. L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.
7. 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.
8. 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.
9. F. Bergadano and S. Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. Siam *Journal of Computing*, 25(6):1268–1280, 1996.

10. D. M Blei and M. Jordan. Variational inference for dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–143, 2006.
11. J. Borges and M. Levene. Data mining of user navigation patterns. In *Web Usage Mining and User Profiling*, number 1836 in Lncs, pages 92–111. Springer-Verlag, 2000.
12. 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.
13. R. C. Carrasco, M. Forcada, and L. Santamaria. Inferring stochastic regular grammars with recurrent neural networks. In *Proceedings of ICGI'96*, volume 1147 of Lnai, pages 274–281. Springer-Verlag, 1996.
14. R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proceedings of ICGI'94*, volume 862 of Lnai, pages 139–150. Springer-Verlag, 1994.
15. R. C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning Journal*, 44(1):185–197, 2001.
16. J. Castro and R. Gavaldá. Towards feasible PAC-learning of probabilistic deterministic finite automata. In *Proceedings of ICGI'08*, volume 5278 of Lncs, pages 163–174. Springer-Verlag, 2008.
17. S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In Acl, pages 310–318. Association for Computational Linguistics, 1996.
18. A. Clark and F. Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, December 2004.
19. T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, New York, NY, 1991.
20. P. Cruz-Alcázar and E. Vidal. Two grammatical inference applications in music processing. *Applied Artificial Intelligence*, 22(1–2):53–76, 2008.
21. C. de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
22. C. de la Higuera and J. Oncina. Identification with probability one of stochastic deterministic linear languages. In *Proceedings of ALT'03*, volume 2842 of Lncs, pages 134–148. Springer-Verlag, 2003.
23. C. de la Higuera and J. Oncina. Learning probabilistic finite automata. In *Proceedings of ICGI'04*, volume 3264 of Lnai, pages 175–186. Springer-Verlag, 2004.
24. C. de la Higuera and F. Thollard. Identication in the limit with probability one of stochastic deterministic finite automata. In *Proceedings of ICGI'00*, volume 1891 of Lnai, pages 15–24. Springer-Verlag, 2000.
25. F. Denis and Y. Esposito. Learning classes of probabilistic automata. In *Proceedings of* Colt *2004*, volume 3120 of Lncs. Springer-Verlag, 2004.
26. 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.
27. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using non deterministic finite automata. In *Proceedings of ICGI'00*, volume 1891 of Lnai, pages 39–50. Springer-Verlag, 2000.

28. F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSA. In *Proceedings of ALT'01*, volume 2225 of Lncs, pages 348–363. Springer-Verlag, 2001.
29. P. Dupont and J.-C. Amengual. Smoothing probabilistic automata: an error-correcting approach. In *Proceedings of ICGI'00*, volume 1891 of Lnai, pages 51–62. Springer-Verlag, 2000.
30. 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.
31. Y. Esposito, A. Lemay, F. Denis, and P. Dupont. Learning probabilistic residual finite state automata. In *Proceedings of ICGI'02*, volume 2484 of Lnai, pages 77–91. Springer-Verlag, 2002.
32. J. Gao and M. Johnson. A comparison of bayesian estimators for unsupervised hidden markov model pos taggers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 344–352. Association for Computational Linguistics, 2008.
33. R. Gavaldà, P. W. Keller, J. Pineau, and D. Precup. Pac-learning of Markov models with hidden state. In *Proceedings of ECML'06*, volume 4212 of Lncs, pages 150–161. Springer-Verlag, 2006.
34. A. Gelfand and A. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85(410):pp. 398–409, 1990.
35. D. Gildea and D. Jurafsky. Learning bias and phonological-rule induction. *Computational Linguistics*, 22:497–530, 1996.
36. 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.
37. P. Grünwald. The minimum description length principle, 2007. MIT Press.
38. O. Guttman. *Probabilistic Automata and Distributions over Sequences*. PhD thesis, The Australian National University, 2006.
39. O. Guttman, S. V. N. Vishwanathan, and R. C. Williamson. Learnability of probabilistic automata via oracles. In *Proceedings of ALT'05*, volume 3734 of Lncs, pages 171–182. Springer-Verlag, 2005.
40. A. Habrard, M. Bernard, and M. Sebban. Improvement of the state merging rule on noisy data in probabilistic grammatical inference. In *Proceedings of ECML'03*, volume 2837 of Lnai, pages 169–1180. Springer-Verlag, 2003.
41. A. Habrard, F. Denis, and Y. Esposito. Using pseudo-stochastic rational languages in probabilistic grammatical inference. In *Proceedings of ICGI'06*, volume 4201 of Lnai, pages 112–124. Springer-Verlag, 2006.
42. M. Heule and S. Verwer. Exact Dfa identification using Sat solvers. In *Proceedings of ICGI'10*, volume 6339 of Lncs, pages 66–79, 2010.
43. J. J. Horning. *A study of Grammatical Inference*. PhD thesis, Stanford University, 1969.
44. M. Hulden. Treba: Efficient numerically stable EM for Pfa. *Journal of Machine Learning Research - Workshop and Conference Proceedings ICGI'12*, 21:249–253, 2012.
45. 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.

46. F. Jelinek. *Statistical Methods for Speech Recognition*. The Mit Press, Cambridge, Massachusetts, 1998.
47. 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.
48. M. J. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. Mit press, 1994.
49. F. Kepler, S. Mergen, and C. Billa. Simple variable length n-grams for probabilistic automata learning. *Journal of Machine Learning Research - Workshop and Conference Proceedings, ICGI'12*, 21:254–258, 2012.
50. S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.
51. K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo one Dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of ICGI'98*, volume 1433 of Lnai, pages 1–12. Springer-Verlag, 1998.
52. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proceedings Of the IEEE*, 84(8):1090–1123, 1996.
53. P. Milani Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. ProSpEx: Protocol Specification Extraction. In *IEEE Symposium on Security and Privacy*, 2009.
54. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(3):269–311, 1997.
55. R. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.
56. 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. Kluwer Academic Publishers, 1997.
57. N. Palmer and P. W. Goldberg. Pac-learnability of probabilistic deterministic finite state automata in terms of variation distance. In *Proceedings of ALT'05*, volume 3734 of Lncs, pages 157–170. Springer-Verlag, 2005.
58. J.R. Partington. *An Introduction to Hankel Operators*. London Mathematical Society Student Texts. Cambridge University Press, 1988.
59. A. Paz. *Introduction to probabilistic automata*. Academic Press, New York, 1971.
60. L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recoginition. *Proceedings of the* Ieee, 77:257–286, 1989.
61. R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103:299–347, 1993.
62. D. Ron, Y. Singer, and N. Tishby. Learning probabilistic automata with variable memory length. In *Proceedings of* Colt *1994*, pages 35–46. Acm Press, 1994.
63. 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.
64. Y. Sakakibara. Grammatical inference in bioinformatics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1051–1062, 2005.
65. A. Sanjeev and B. Boaz. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
66. L. Saul and F. Pereira. Aggregate and mixed-order Markov models for statistical language processing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89. ACL, 1997.

67. C.R. Shalizi and J.P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of statistical Physics*, 104:817–879, 2001.
68. C. Shibata and R. Yoshinaka. Marginalizing out transition probabilities for several subclasses of PFAs. *Journal of Machine Learning Research - Workshop and Conference Proceedings, ICGI'12*, 21:259–263, 2012.
69. A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. Ph.D. dissertation, University of California, 1994.
70. A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.
71. F. Thollard. Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *Proceedings of ICML'01*, pages 561–568. Morgan Kauffman, 2001.
72. F. Thollard and A. Clark. PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, 2004.
73. F. Thollard and P. Dupont. Entropie relative et algorithmes d'inférence grammaticale probabiliste. In *Actes de la conférence* CAP *'99*, pages 115–122, 1999.
74. F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proceedings of ICML'00*, pages 975–982. Morgan Kaufmann publishing, 2000.
75. S. Verwer, M. de Weerdt, and C. Witteveen. Learning driving behavior by timed syntactic pattern recognition. In *IJCAI'11*, pages 1529–1534, 2011.
76. S. Verwer, M. Weerdt, and C. Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *Proceedings of ICGI'10*, volume 6339 of *LNCS*, pages 203–216. Springer-Verlag, 2010.
77. 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, 2005.
78. 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, 2005.
79. N. Walkinshaw, B. Lambeau, C. Damas, K. Bogdanov, and P. Dupont. Stamina: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, pages 1–34, 2012.
80. C. S. Wetherell. Probabilistic languages: a review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.
81. Tzay Y. Young. *Handbook of Pattern Recognition and Image Processing: Computer Vision*, volume 2. Academic Press, 1994.
82. M. Young-Lai and F. W. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning Journal*, 40(2):111–137, 2000.
83. 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.

Appendix

Nb	Solution	Hulden	Kepler	Bailly	Shibata	Llorens	3-gram
1	29.898	30.131	30.547	30.147	**29.994**	30.395	33.415
2	168.331	168.455	174.866	168.429	168.430	**168.420**	177.950
3	49.956	50.044	55.540	50.174	**50.042**	50.675	68.248
4	80.818	80.837	85.480	80.844	**80.827**	80.843	101.161
5	33.235	33.241	33.427	33.237	**33.237**	33.238	44.616
6	66.985	67.044	82.35	67.059	67.007	**67.000**	110.059
7	51.224	51.265	52.092	51.264	**51.249**	51.259	57.134
8	81.375	81.710	85.849	81.799	**81.403**	81.710	106.194
9	20.840	20.889	26.920	25.229	20.856	**20.850**	66.094
10	33.303	33.401	34.554	33.724	**33.334**	34.039	44.986
11	31.811	32.138	33.248	32.138	**31.853**	32.546	37.909
12	21.655	21.671	21.912	21.671	**21.663**	21.769	25.086
13	62.806	63.073	120.565	100.681	62.820	**62.816**	157.789
14	116.792	116.841	118.602	116.914	**116.836**	116.839	125.827
15	44.242	44.285	45.208	45.285	**44.274**	44.701	48.155
16	30.711	30.844	31.809	35.586	30.7187	**30.7186**	41.168
17	47.311	47.354	48.109	48.735	**47.352**	47.9215	51.804
18	57.329	57.339	57.534	76.103	**57.3316**	57.3320	65.941
19	17.877	17.930	18.816	19.316	**17.880**	17.919	21.118
20	90.972	91.016	95.304	91.351	**90.999**	93.504	108.990
21	30.519	30.605	35.578	30.714	**30.568**	32.217	46.839
22	25.982	26.078	26.136	26.010	**25.988**	26.080	29.904
23	18.408	18.418	18.720	18.547	**18.413**	18.447	20.805
24	38.729	38.737	42.366	38.753	**38.7317**	38.7322	48.068
25	65.735	65.978	67.929	66.069	**65.783**	67.266	86.387
26	80.743	82.657	111.502	141.082	**80.833**	80.837	211.986
27	42.427	42.473	43.511	42.712	42.464	**42.456**	46.825
28	52.744	52.855	53.583	53.084	**52.841**	53.198	65.570
29	24.031	24.199	28.580	24.817	**24.042**	24.106	43.389
30	22.926	**22.932**	23.394	22.960	22.934	23.211	25.291
31	41.214	41.243	42.531	41.417	**41.233**	41.623	46.323
32	32.613	32.743	41.975	38.300	32.622	**32.619**	52.932
33	31.865	31.872	32.2194	31.920	**31.871**	32.030	34.485
34	19.955	20.428	20.581	20.476	**19.969**	20.542	23.499
35	33.777	34.326	34.714	33.835	**33.800**	34.295	42.808
36	37.986	38.203	38.206	38.176	**38.018**	38.405	39.686
37	20.980	21.016	21.025	21.027	**21.001**	21.016	21.106
38	21.446	21.494	21.650	21.514	**21.459**	21.596	21.736
39	10.002	**10.0029**	10.054	10.005	10.0034	10.004	10.262
40	8.201	8.255	8.366	8.496	8.207	**8.206**	8.847
41	13.912	13.941	13.942	13.932	**13.921**	13.940	14.0142
42	16.004	16.008	16.080	16.008	16.007	**16.005**	16.431
43	32.637	32.747	32.841	32.817	**32.723**	32.777	32.862
44	11.709	11.798	11.920	11.778	**11.725**	12.041	12.082
45	24.042	24.048	24.252	24.084	24.050	**24.045**	24.153
46	11.982	11.999	12.136	12.082	**11.988**	12.106	12.272
47	4.1190	4.124	4.144	4.120	4.1192	**4.1191**	4.671
48	8.036	8.042	8.183	8.045	**8.039**	8.191	8.417

Table 1 Perplexity scores of active participants, the 3-gram baseline, and the solutions for all problem instances.

Nb	num states	alphabet size	symbol sparsity	trans. sparsity	size	type
1	63	8	0.3274	0.0872	20k	HMM
2	63	18	0.3280	0.0166	20k	HMM
3	25	4	0.7900	0.0790	20k	PFA
4	12	4	0.4375	0.1508	100k	PFA
5	56	6	0.2946	0.0217	20k	HMM
6	19	6	0.4825	0.0526	20k	DPFA
7	12	13	0.2372	0.0833	20k	DPFA
8	49	8	0.3622	0.0645	100k	PFA
9	71	4	0.3873	0.0141	20k	DPFA
10	49	11	0.6327	0.0221	20k	PFA
11	47	20	0.4947	0.0213	20k	DPFA
12	12	13	0.3526	0.1116	20k	PFA
13	63	4	0.6905	0.0159	100k	DPFA
14	15	12	0.4944	0.0800	20k	HMM
15	26	14	0.4121	0.0672	20k	PFA
16	49	10	0.6184	0.0204	100k	DPFA
17	22	13	0.2168	0.1738	20k	PFA
18	25	20	0.2260	0.0400	100k	DPFA
19	68	7	0.3256	0.0350	100k	HMM
20	11	18	0.3939	0.1570	20k	HMM
21	56	23	0.2531	0.0497	20k	HMM
22	55	21	0.0575	0.2411	100k	PFA
23	33	7	0.3810	0.1148	100k	HMM
24	6	5	0.5000	0.1666	20k	DPFA
25	40	10	0.5775	0.0456	20k	HMM
26	73	6	0.5868	0.0137	20k	DPFA
27	19	17	0.6378	0.0526	20k	DPFA
28	23	6	0.7464	0.1134	20k	HMM
29	36	6	0.3750	0.0384	20k	PFA
30	9	10	0.6555	0.1751	20k	PFA
31	12	5	0.3833	0.1992	20k	PFA
32	43	4	0.7733	0.0233	100k	DPFA
33	13	15	0.5949	0.1183	20k	HMM
34	64	21	0.3705	0.0281	20k	PFA
35	47	20	0.3553	0.0213	20k	DPFA
36	54	9	0.6317	0.0748	100k	HMM
37	69	8	0.5217	0.1825	100k	PFA
38	14	10	0.7857	0.1939	20k	HMM
39	6	14	0.4167	0.1810	20k	PFA
40	65	14	0.6473	0.0154	20k	DPFA
41	54	7	0.6931	0.1430	100k	HMM
42	6	9	0.5185	0.1667	20k	DPFA
43	67	5	0.5970	0.1641	20k	PFA
44	73	13	0.6333	0.0561	20k	HMM
45	14	19	0.8008	0.0867	20k	HMM
46	19	23	0.4851	0.0973	20k	PFA
47	61	15	0.3027	0.0164	100k	DPFA
48	16	23	0.6957	0.0625	20k	DPFA

Table 2 The parameters of all generated problem instances: number of states, alphabet size, symbol sparsity, transition sparsity, size of training set, and type of machine.