PAutomaC (Verwer et al.2013) - summary

Magdalena Markowska

Statistics, Fall 2020

Goal

What probabilistic automaton learning techniques work best in which setting for approximating distribution over strings?

Outline

1. Overview of probabilistic finite-state automata
   - types and
   - learning techniques
2. Description of the PAutomaC challenge
   - data
   - evaluation
   - result analysis

Automaton model

- Is used for characterizing the behavior of systems or processes
- is learned from a finite set of string. Because often times the data gathered from observation is unlabeled, it could happen that we can only observe the strings that were generated by the system. What if we want to find out whether the system may also generate some unseen strings?

Probabilistic automaton model

- Here different symbols are generated with different probabilities in different states. As a result, we could calculate the probability of a string, and we could also calculate a probability of previously unseen strings.

What’s the purpose of learning such automata?

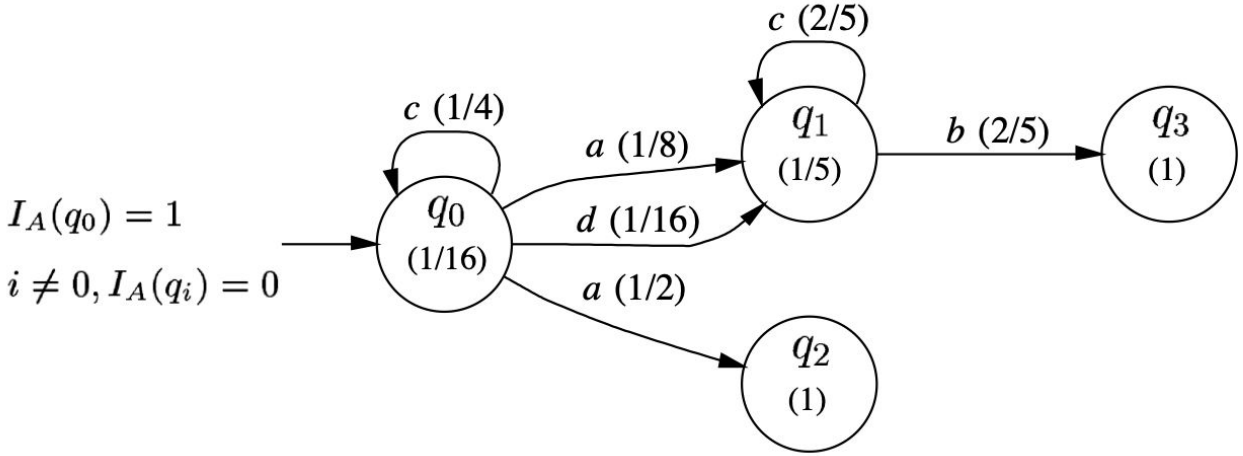
- To find the best probabilistic model that gives the highest probability of generating the data.

Probabilistic finite state machines-PFA

PFA - probabilistic finite state automata

- Every state is assigned an initial and final probability, and every transition is assigned a transition probability
- For the entire machine: the sum of all initial probabilities must be equal to 1
- For each state: the sum of all the transition probabilities and final state must be equal to 1

Probabilistic finite state machines-PFA



Vidal el al. (2005)

Probabilistic finite state machines-PFA

Since PFA is defined as non-deterministic, it allows one string to be generated through different paths. Let's look at string *a* - it can be generated either via q_0 -> q_1 or q_0 -> q_2. Therefore, the probability of string a will be calculated as follows:

$$\begin{aligned}\Pr_{\mathcal{A}}(a) &= I(q_0)\cdot P(q_0,a,q_1)\cdot F(q_1) + I(q_0)\cdot P(q_0,a,q_2)\cdot F(q_2)\\ &= 1.0\cdot 0.125\cdot 0.2 + 1.0\cdot 0.5\cdot 1.0\\ &= 0.525\ .\end{aligned}$$

Vidal el al. (2005)

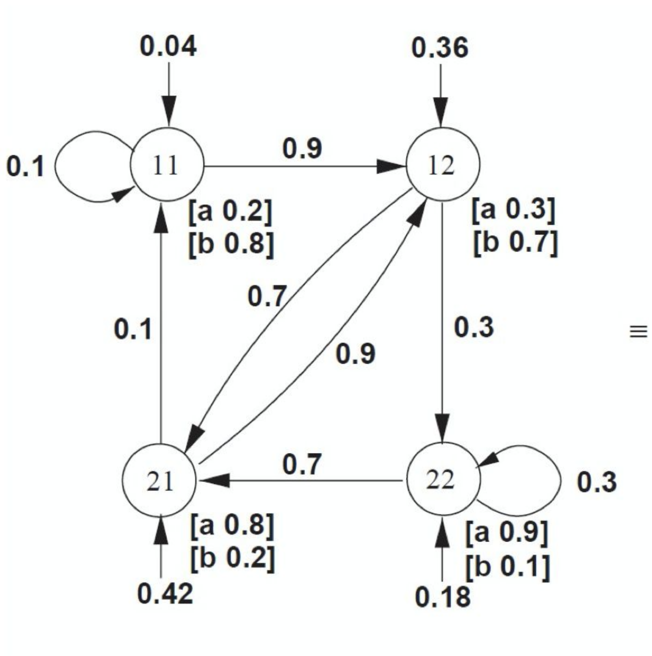
---silly question---

Does those different paths of string generation have anything to do with polysemy (e.g. bank or wood) or semanitc diversity?

Probabilistic finite state machines-HMM

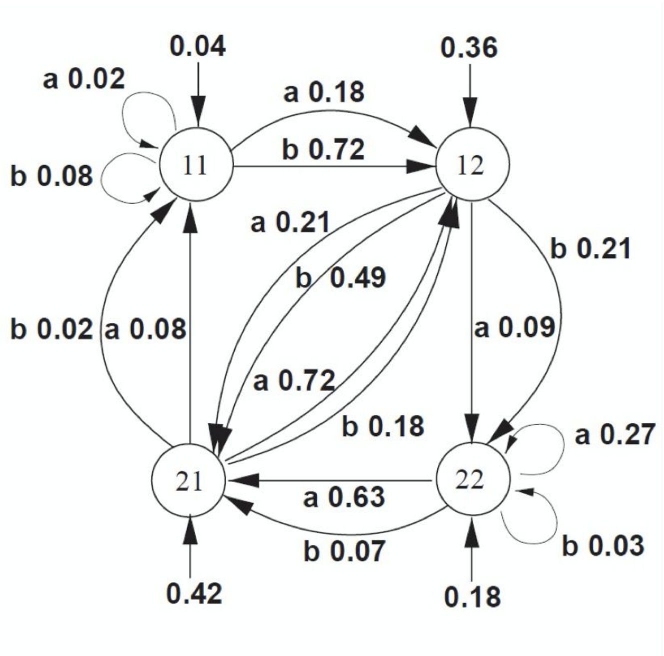
- Symbols are emitted at the states (PFA - symbols are generated at the transitions)
- Every state is assign an initial probability (PFA - both initial and final)
- Every transition is assigned a probability
- The probabilities of the symbols at a state sum up 1
- The probabilities of the paths from a state sum up 1
- If we add an EOS symbol to HMM model, it will generate the same strings as PFA.

HMM vs PFA machines corresponding to the same probability distribution



a.

≡



b.

a=Hmm
b=PFA

Dupont et al. (2005)

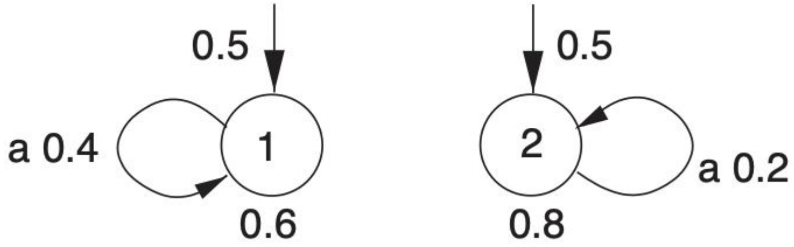
Probabilistic finite state machines-DPFA

PDFA - deterministic probabilistic finite automaton

- Subclass of PFA that is much less powerful

PFA generating a language that

Cannot be generated by PDFA:



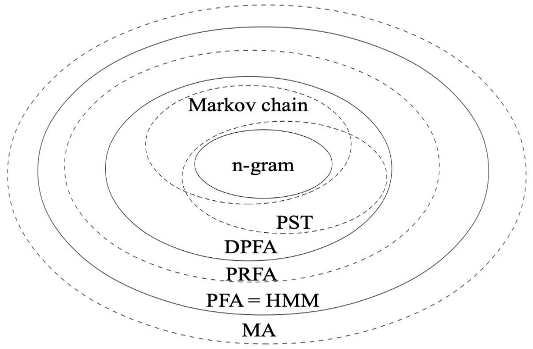
Dupont et al. (2005)

Other probabilistic finite state automata

- Markov chains (Saul and Pierra, 1997)
- N-grams (Ney et al. 1997; Jelinek 1998)
- PST -- probabilistic suffix trees (Ron et al. 1994)
- PRFA -- probabilistic residual finite state automata (Esposito et al. 2002)
- MA -- multiplicity automata (Bergadano and Varricchio, 1996; Beimel at el. 200) ->

MA is the most expressive of all.

It can model the most distributions.



Verwer et al. (2013)

Learning probabilistic finite state machines

According to Verwer er al. (2013), there are three major techniques for learning PFA, HMMs, and DPFA.

1. Parameter estimation
2. Bayesian inference
3. State-merging

Parameter estimation

Takes a standard architecture for the machine and tries to find parameter settings that maximize the likelihood of the the data given the model.

If the machine is deterministic, the standard method is MLE (Wetherell, 1980), which estimates the value of a parameter for which the observed data have the biggest probability.

If the machine in non-deterministic, the standard method is the Baum-Welch algorithm (Baum et al. 1970), which iteratively computes a new estimate for the transition probabilities using the probabilities assigned to the input data. (example from Wikipedia)

Bayesian inference

Instead of learning a point estimate, the predictions are made by using the join distribution formed by all possible models. For example, Gibbs sampling in HMMs estimates this joint distribution by iteratively sampling the visited hidden states conditioned on earlier samples of all other state visits.

Video on training LDA (Latent Dirchilet Allocation) using Gibbs sampling.

State-merging

Used for DPFA

We first start with a very large automaton with enough states to describe the learning sample. Then the states of this automaton are combined iteratively to create a more compact model.

The most recognized state merging algorithms for probabilistic automata are:

- ALERGIA (Carrasco and Oncina, 1994)
- Bayesian model merging (Stolcke, 1994)
- Learn-PSA (Ron et al. 1995)

Smoothing the savior

In the cases of the learning methods that estimate the model parameters are based on maximum likelihood, smoothing techniques are applied to deal with low frequency strings (larger probability is assigned to a rare strings, lowering the probability of more frequent strings).

According to the Verwer et al. (2013), multiple smoothing methods were proposed for the DPFA learning. The same is not true for DFA and HMMs.

PAUTOMAC aims to tell us which learning technique use when

PAUTOMAC was an online challenge developed to:

- provide an overview of which probabilistic automaton learning techniques work best, when to they work the best and why
- stimulate the development of new techniques for learning distributions over strings

PAC (probability approximately correct) learning is a theoretical framework used for mathematical analysis of machine learning. Let’s say we are given a distribution of samples, and we must select a generalization function from some set of possible functions. We want the selected function to have low generalization error with high probability.

PAutomaC

- The competition was set up using an oracle server, which was able to evaluate the submissions by participants on-line
- The performance was evaluated using probabilities assigned by a learned distribution
- The data types available for this competition were both artificial and real data donated by other researchers. Of interest is the former type for reasons I didn’t quite understand.

Artificial data generation

Data was generated by building random probabilistic machines. This process takes 4 parameters:

- *N* (number of states): 5-75
- *A* (size of the alphabet): 4-24 symbols
- S_S (the symbol sparsity): of all possible state-symbol pairs, only 20%-80% were generated
- T_S (the transition sparsity): for every generated state-symbol pair, one transition was generated to a random target state. Of all such transitions, between 0-20% were generated.

This created a selection without replacement to remain uniform over the states and transition labels.

Each generated path had to have an accepting state.

Artificial data generation

Initial and final states were selected without replacement until the percentages of selected states exceeded the sparsities of transition and symbol ($T_S * N$ and $S_S * N$), respectively.

All initial, symbol, and transition probabilities were drawn from a Dirichlet distribution. The final probabilities were drawn together with the symbol probabilities.

Dirichlet distribution

- Distribution over distribution (similar to multinomial distribution but continuous as opposed to discrete; in Dirichlet distribution we have infinite number of samples).
- Examples: the Chinese restaurant process (probability distribution over the tables is a sample of probability observations drawn from a Dirichlet process) or Polya urn sampling -- both creating clustering effect

Artificial data generation

One training set and one test set (without repetitions) were generated from every target. If the average length of the generated strings is outside of a predefined range ($5<len<50$), a new automaton and new data sets are generated.

150 models, together with training and test sets were generated. Their complexity was then evaluated using 3-gram baseline algorithm. 16 models were chosen based on the diversity in N, A, S_S, T_S, and difficulty.

The same procedure was applied for DPFA (without generating additional transitions), and for HMMs (generating state-state paris instead of state-symbol-state triples).

Total: 46 artificially generated problems for the competition.

Evaluation

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

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

where $P_T(x)$ is the normalized probability of *x* in the target and $P_C(x)$ is the normalized candidate probability for *x* submitted by the participant. The normalization process was to modify the probabilities so that they all sum to 1 on the set TS. Consequently, adding probability to one of the test string removed probability from the others.

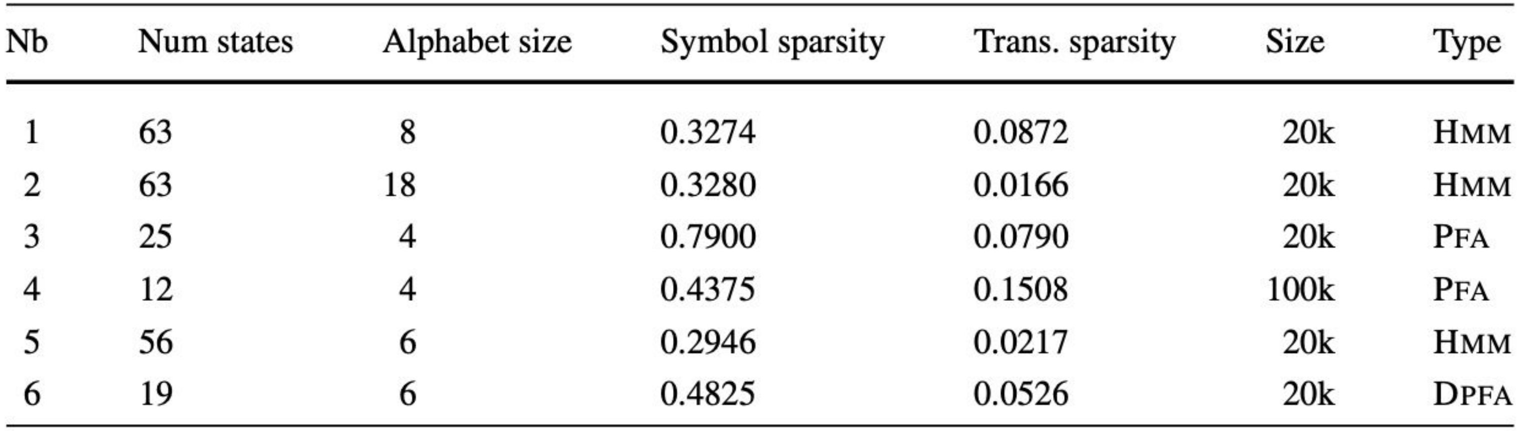
Results

38 registered participants, out of which 16 submitted at least one of their solutions to a problem set. In total, there were 2787 submissions.

Winners: team Shibata-Yoshinaka had the best perplexity score on most instances and performed well on others. The difference between the perplexity values of the solutions and their submissions was never greater than 0.1. With the number of strings = 100k, such difference was even smaller

Results

The different approaches were evaluated on problem sets with many different parameters, which, in return, enabled for some additional analysis of the results. The aim was to see which method works best when and why.



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

Results

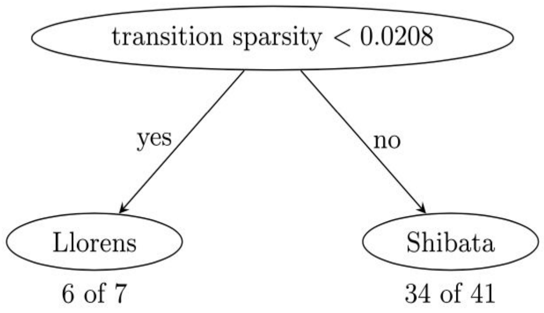
The results were further analyzed using a standard decision tree learning for two prediction tasks:

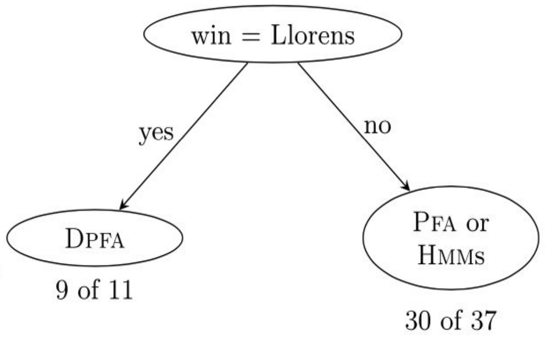
1. Predicting the winner given the problem parameter values.
2. Predicting whether a deterministic distribution was used to generate the problem given the winner.

Results

Even though Shibata team was the winner after all, the Llorens teams outperformed it on sparse problem instances. Whenever team Llorens won, in 9 out of 11 cases, they used probabilistic generator. Most of the sparse problems were generated using probabilistic automata, which are fixed to use only a fraction of all possible transitions given *N* and *A*.

That could indicate that it is best to learn the deterministic model when the data is drawn from the deterministic distribution.





Team Shibata-Yoshinaka

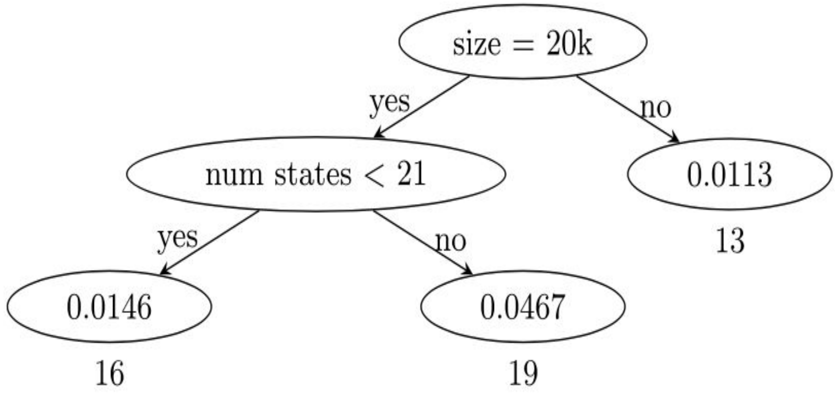
Used Gibbs sampling to calculate the probability ($\Pr(\mathbb{b}|\mathbb{a})$) of a future sentence $\mathbb{b}$ given the training data $\mathbb{a}$ generated by an unknown PFA. The probability of generating a sentence $\mathbb{a}=a_1, \ldots, a_T$ by passing the states $\mathbb{z}=z_0,\ldots,z_T$ is given as:

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

Verwer et al. (2013:143)

Where ξ_{iaj} (/gzai/) is the probability of the transition from *i* to *j* with a letter *a*. C_{iaj} counts the times when the transition occurs.

Team Shibata-Yoshinaka



Best performance is achieved when many strings are available for training, or when the target contains less than 21 states. In other cases, the average perplexity difference is still a low value (0.0467).

The perplexity score measured how well the differences in the assigned probabilities matched with the target probabilities

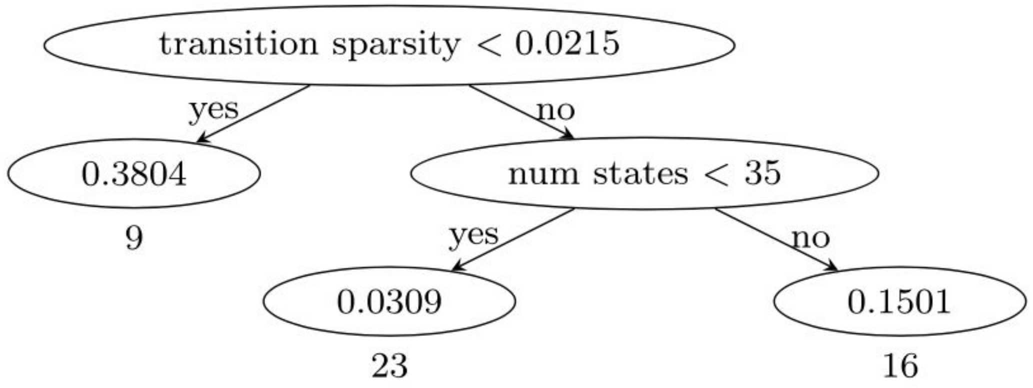
Hulden team

Used three strategies:

1. A basic baseline n-gram with smoothing
2. A basic baseline n-gram without smoothing but using interpolated test data (interpolation = estimating a (value of a) function given some known values of the function)
3. The construction of a fully connected PFA inferred with Baum-Welch, each between 5 and 40 states. Training was done by using the original training data, and separately also using reconstructed training data.

Hulden team

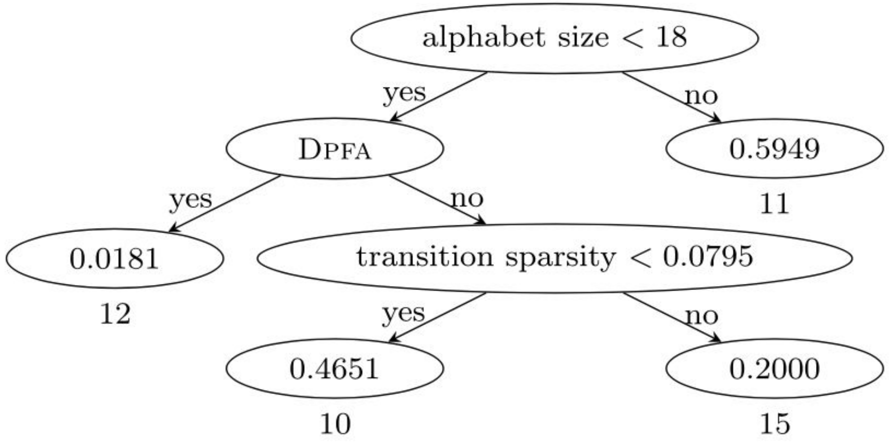
Best performance on dense problems with N < 35.



Team Llorens

Used un upgraded version of the ALERGIA algorithm (the state-merging approach).

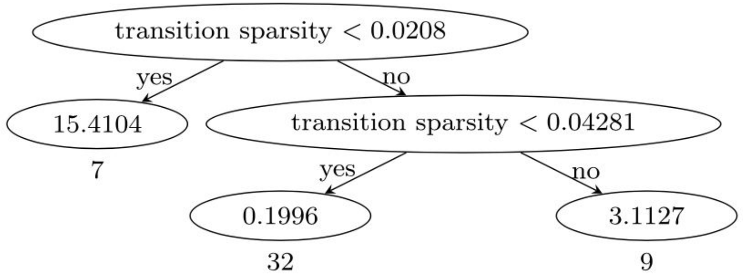
Best performance with A < 18, and with the distribution generated by a deterministic automata. If the distribution was generated by non-deterministic machine, it is significantly easier to learn such distribution on dense problems (T_S > 0.0795), even in the case when the learned model is deterministic (ALGERIA).



Team Bailly

Used a spectral approach -- the crucial factor of which is the Hankel matrix factorization (representing the counts of every possible prefix-suffix pair). For references see Bailly (2011) and Partington (1988).

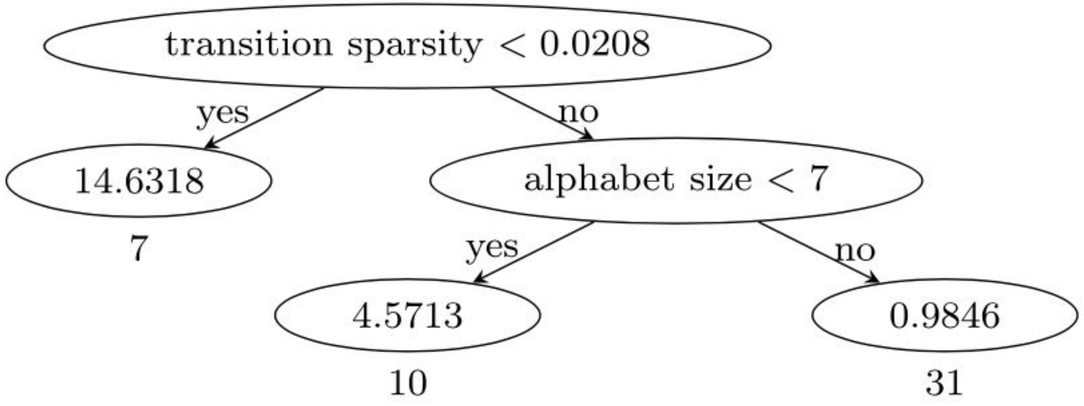
Best performance in the most number of problems (32/48), but they didn't perform well on sparse data ($T_S < 0.04281$).



Team Kepler

Used n-gram models with variable length. N-grams are represented as a context tree that maps the probabilities of sequences of symbols. With large n-grams, the context tree is cut based on the Kullback-Leibler divergence (Kullback and Leibler, 1951).

The best performance was on more dense problems ($T_S > 0.0208$) with $A > 7$.



References

Bailly, R. (2011). QWA: spectral algorithm. In *JMLR—workshop and conference proceedings: Vol. 20*. *Pro- ceedings of the Asian conference on machine learning, ACML'11* (pp. 147–163). Cambridge: JMLR.

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

Carrasco, R. C., & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *LNAI: Vol. 862*. *Proceedings of the international colloquium on grammatical inference ICGI'94* (pp. 139–150). Berlin: Springer.

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

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Partington, J. R. (1988). *An introduction to Hankel operators*. *London mathematical society student texts*. Cambridge: Cambridge University Press.

Ron, D., Singer, Y., & Tishby, N. (1995). On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the conference on learning theory COLT'95* (pp. 31–40). New York: ACM.

References

Stolcke, A. (1994). *Bayesian learning of probabilistic language models*. Ph.D. dissertation, University of California.

Verwer, S., Eyraud, R. & de la Higuera, C. PAUTOMAC: a probabilistic automata and hidden Markov models learning competition. *Mach Learn* 96, 129–154 (2014). https://doi.org/10.1007/s10994-013-5409-9

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

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

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