JMLR: Workshop and Conference Proceedings 57:105–119, 2016

Sp2Learn: A Toolbox for the spectral learning of weighted automata*

Denis Arrivault DENIS.ARRIVAULT@LIF.UNIV-MRS.FR
LabEx Archimède, Aix-Marseille University, France

Dominique Benielli DOMINIQUE.BENIELLI@UNIV-AMU.FR
LabEx Archimède, Aix-Marseille University, France

François Denis FRANCOIS.DENIS@LIF.UNIV-MRS.FR
QARMA team, Laboratoire d'Informatique Fondamentale de Marseille, France

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

Abstract

Sp2Learn is a Python toolbox for the spectral learning of weighted automata from a set of strings, licensed under Free BSD. This paper gives the main formal ideas behind the spectral learning algorithm and details the content of the toolbox. Use cases and an experimental section are also provided.

Keywords: Toolbox, Spectral Learning, Weighted Automata

1. Introduction

Grammatical inference is a sub-field of machine learning that mainly focuses on the induction of grammatical models such as, for instance, finite state machines and generative grammars. However, the core of this field may appear distant from mainstream machine learning: the methods, algorithms, approaches, paradigms, and even mathematical tools used are usually not the ones of statistical machine learning.

There exists one important exception to this observation: the recent developments of what is called spectral learning are building a bridge between these two facets of machine learning. Indeed, by allowing the use of linear algebra in the context of finite state machine learning, tools of statistical machine learning are now usable to infer grammatical formalisms.

The initial idea of spectral learning is to describe finite state machines using linear representations: instead of sets of states and transitions, these equivalent models are made of vectors and matrices [Berstel and Reutenauer, 1988]. The class of machines representable with these formalisms is the one of Weighted Automata (WA)[1] [Mohri, 2009], sometimes called multiplicity automata [Beimel et al., 1996], that are a strict generalization of Probabilistic Automata (PA) [Schützenberger, 1961] and of Hidden Markov Models (HMM) [Dupont et al., 2005].

* This work was supported in part by the LabEx Archimède of the Aix-Marseille University (ANR-11-LABX-0033). This publication only reflects the authors' views.

1. Only WA whose weights are real numbers are considered in this work.

The corner stone of the spectral learning approach is the use of what is called the Hankel matrix. In its classical version, this bi-infinite matrix has rows that correspond to prefixes and columns to suffixes: the value of a cell is then the weight of the corresponding sequence in the corresponding WA. Importantly, the rank of this matrix is the number of states of the WA: this allows the construction of the automaton from a rank factorization of the matrix [Balle et al., 2014].

Following this result, the behavior of the spectral learning algorithm relies on the construction of a finite sub-block approximation of the Hankel matrix from a sample of sequences. Then, using a Singular Value Decomposition of this empirical Hankel matrix, one can obtain a rank factorization and thus a weighted automaton.

From the seminal work of Hsu et al. [2009] and Bailly et al. [2009], important developments have been achieved. For example, Siddiqi et al. [2010] obtain theoretical guaranties for low-rank HMM; A PAC-learning result is provided by Bailly [2011] for stochastic weighted automata; Balle et al. [2014] extend the algorithm to variants of the Hankel matrix and show their interest for natural language processing; Extensions to the spectral learning of weighted tree automata have been published by Bailly et al. [2010].

In the context of this great research effervescence, we felt that an important piece was missing which would help the widespread adoption of spectral learning techniques: an easy to use and install program with broad coverage to convince non-initiated researchers about the interest of this approach. This is the main motivation behind the project of the SPiCe Spectral Learning (Sp2Learn) Python toolbox[2] that this paper presents.

We notice that a code for 3 methods of moments, including a spectral learning algorithm, is available at https://github.com/ICML14MoMCompare/spectral-learn. However, this code was designed for a research paper and suffers many limitations, as for instance only a small number of data sets, the one studied in the article, can be used easily.

Section 2 gives formal details about the spectral learning of weighted automata. Section 3 carefully describes the toolbox content and provides use cases. Some experiments showing the potential of Sp2Learn are given in Section 4, while Section 5 concludes by giving ideas for future developments.

2. Spectral learning of weighted automata

2.1. Weighted Automata

A finite set of symbols is called an alphabet. A string over an alphabet Σ is a finite sequence of symbols of Σ. Σ^* is the set of all strings over Σ. The length of a string w is the number of symbols in the string. We let ϵ denote the empty string, that is the string of length 0. For any $w \in \Sigma^*$, let $pref(w) = \{u \in \Sigma^* : \exists v \in \Sigma^*, uv = w\}$ be its set of prefixes, $suff(w) = \{u \in \Sigma^* : \exists v \in \Sigma^*, vu = w\}$ be its set of suffixes, and $fact(w) = \{u \in \Sigma^* : \exists l, r \in \Sigma^*, lur = w\}$ be the set of factors of w (sometime called the set of substrings).

The following definitions are adapted from Mohri [2009]:

2. SPiCe stands for Sequence PredIction ChallengE, an on-line competition where the toolbox was released as a baseline.

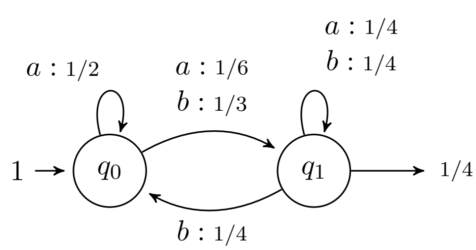
Definition 1 (Weighted automaton) *A weighted automaton (WA) is a tuple* $\langle \Sigma, Q, I, F, \mathcal{T}, \lambda, \rho \rangle$ *such that:* Σ *is a finite alphabet;* Q *is a finite set of states;* $\mathcal{T} : Q \times \Sigma \times Q \to \mathbb{R}$ *is the transition function;* $\lambda : Q \to \mathbb{R}$ *is an initial weight function;* $\rho : Q \to \mathbb{R}$ *is a final weight function.*

A transition is usually denoted (q_1, σ, p, q_2) instead of $\mathcal{T}(q_1, \sigma, q_2) = p$. We say that two transitions $t_1 = (q_1, \sigma_1, p_1, q_2)$ and $t_2 = (q_3, \sigma_2, p_2, q_4)$ are consecutive if $q_2 = q_3$. A path π is an element of $\mathcal{T}^*$ made of consecutive transitions. We denote by $o[\pi]$ its origin and by $d[\pi]$ its destination. The weight of a path is defined by $\mu(\pi) = \lambda(o[\pi]) \times \omega \times \rho(d[\pi])$ where ω is the multiplication of the weights of the constitutive transitions of π. We say that a path $(q_0, \sigma_1, p_1, q_1) \ldots (q_{n-1}, \sigma_n, p_n, q_n)$ reads a string w if $w = \sigma_1 \ldots \sigma_n$. The weight of a string w is the sum of the weights of the paths that read w.

A series r over an alphabet Σ is a mapping $r : \Sigma^* \to \mathbb{R}$. A series r over Σ^* is *rational* if there exist an integer $k \geq 1$, vectors $I, T \in \mathbb{R}^k$, and matrices $M_\sigma \in \mathbb{R}^{k \times k}$ for every $\sigma \in \Sigma$, such that for all $u = \sigma_1 \sigma_2 \ldots \sigma_m \in \Sigma^*$,

$$r(u) = I M_u T = I M_{\sigma_1} M_{\sigma_2} \ldots M_{\sigma_m} T$$

The triplet $\langle I, (M_\sigma)_{\sigma \in \Sigma}, T \rangle$ is called a *k-dimensional linear representation* of r. The rank of a rational series r is the minimal dimension of a linear representation of r. Linear representations are equivalent to weighted automata where each coordinate corresponds to a state, the vector I provides the initial weights (*i.e.* the values of function λ), the vector T is the terminal weights (*i.e.* the values of function ρ), and each matrix M_σ corresponds to the σ-labeled transition weights ($M_\sigma(q_1, q_2) = p \iff (q_1, \sigma, p, q_2)$ is a transition).



$$I = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad T = \begin{bmatrix} 0 \\ 1/4 \end{bmatrix}$$

$$M_a = \begin{bmatrix} 1/2 & 1/6 \\ 0 & 1/4 \end{bmatrix} \qquad M_b = \begin{bmatrix} 0 & 1/3 \\ 1/4 & 1/4 \end{bmatrix}$$

Figure 1: A weighted automaton and the equivalent linear representation.

In what follows, we will confound the two notions and consider that weighted automata are defined in terms of linear representations.

A particular kind of WA is of main interest in the spectral learning framework: a weighted automata A is *stochastic* if the series r it computes is a probability distribution over Σ^*, *i.e.* $\forall x \in \Sigma^*, r(x) \geq 0$ and $\sum_{x \in \Sigma^*} r(x) = 1$. These WA enjoy properties that are important for learning. For instance, in addition to the probability of a string $r(x)$, a WA can compute the probability of a string to be a prefix $r_p(x) = r(x\Sigma^*)$, or to be a suffix $r_s(x) = r(\Sigma^* x)$. It can be shown that the rank of the series r, r_p, and r_s are equal [Balle et al., 2014]. Other properties of stochastic WA are of great interest for spectral learning but it is beyond the scope of this paper to describe them all. We refer the Reader to the work of Balle et al. [2014] for more details.

Finally, stochastic weighted automata are related to other finite state models: they are strictly more expressive than Probabilistic Automata [Denis et al., 2006] (which are equivalent to discrete Hidden Markov Models [Dupont et al., 2005]) and thus than Deterministic Probabilistic Automata [Carrasco and Oncina, 1994].

2.2. Hankel matrices

The following definitions are based on the ones of Balle et al. [2014].

Definition 2 *Let r be a rational series over Σ. The Hankel matrix of r is a bi-infinite matrix $H \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ whose entries are defined as $H(u, v) = r(uv)$ for any $u, v \in \Sigma^*$. That is, rows are indexed by prefixes and columns by suffixes.*

For obvious reasons, only finite sub-blocks of Hankel matrices are going to be of interest. An easy way to define such sub-blocks is by using a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$, where $\mathcal{P}, \mathcal{S} \subseteq \Sigma^*$. We write $p = |\mathcal{P}|$ and $s = |\mathcal{S}|$. The sub-block of H defined by $\mathcal{B}$ is the matrix $H_\mathcal{B} \in \mathbb{R}^{p \times s}$ with $H_\mathcal{B}(u, v) = H(u, v)$ for any $u \in \mathcal{P}$ and $v \in \mathcal{S}$. We may just write H if the basis $\mathcal{B}$ is arbitrary or obvious from the context.

In the context of learning weighted automata, the focus is on a particular kind of bases. They are called *closed bases*: a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is prefix-closed[3] if there exists a basis $\mathcal{B}' = (\mathcal{P}', \mathcal{S})$ such that $\mathcal{P} = \mathcal{P}'\Sigma'$, where $\Sigma' = \Sigma \cup \{\epsilon\}$. A prefix-closed basis can be partitioned into $|\Sigma| + 1$ blocks of the same size: given a Hankel matrix H and a prefix-closed basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ with $\mathcal{P} = \mathcal{P}'\Sigma'$, we have, for a particular ordering of the elements of $\mathcal{P}$:

$$H_\mathcal{B}^\top = [H_\epsilon^\top | H_{\sigma_1}^\top | H_{\sigma_2}^\top | \dots | H_{\sigma_{|\Sigma|}}^\top]$$

where H_σ are sub-blocks defined over the basis $(\mathcal{P}'\sigma, \mathcal{S})$ such that $H_\sigma(u, v) = H(u\sigma, v)$. The notation uses here means that $H_\mathcal{B}^\top$ can be successively restricted to the other sub-blocks.

The rank of a rational series r is equal to the rank of its Hankel matrix H which is thus the number of states of a minimal weighted automaton that represents r. The rank of a sub-block cannot exceed the rank of H and we are interested by full rank sub-blocks: a basis $\mathcal{B}$ is *complete* if $H_\mathcal{B}$ has full rank, that is $rank(H_\mathcal{B}) = rank(H)$.

We will consider different variants of the classical Hankel matrix H of a series r:

- H^p is the *prefix Hankel matrix*, where $H^p(u, v) = r(uv\Sigma^*)$ for any $u, v \in \Sigma^*$. In this case rows are indexed by prefixes and columns by factors.
- H^s is the *suffix Hankel matrix*, where $H^s(u, v) = r(\Sigma^* uv)$ for any $u, v \in \Sigma^*$. In this matrix rows are indexed by factors and columns by suffixes.
- H^f is the *factor Hankel matrix*, where $H^f(u, v) = r(\Sigma^* uv\Sigma^*)$ for any $u, v \in \Sigma^*$. In this matrix both rows and columns are indexed by factors.

3. Similar notions of closure can be designed for suffix and factor.

2.3. Hankel matrices and WA

We consider a rational series r, H its Hankel matrix, and $A = \langle I, (M_\sigma)_{\sigma\in\Sigma}, T\rangle$ a minimal WA computing r. We suppose that A has n states.

We first notice that A induces a *rank factorization* of H: we have $H = PS$, where $P \in \mathbb{R}^{\Sigma^*\times n}$ is such that its u^{th} row equals $I^\top M_u$, and reciprocally $S \in \mathbb{R}^{n\times\Sigma^*}$ is such that its v^{th} column is $M_v T$. Similarly, given a sub-block $H_\mathcal{B}$ of H defined by the basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ we have $H_\mathcal{B} = P_\mathcal{B} S_\mathcal{B}$ where $P_\mathcal{B} \in \mathbb{R}^{P\times n}$ and $S_\mathcal{B} \in \mathbb{R}^{n\times S}$ are restrictions of P and S on $\mathcal{P}$ and $\mathcal{S}$, respectively. Besides, if $\mathcal{B}$ is complete then $H_\mathcal{B} = P_\mathcal{B} S_\mathcal{B}$ is a rank factorization.

Moreover, the converse occurs: given a sub-block $H_\mathcal{B}$ of H defined by the complete basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$, one can compute a minimal WA for the corresponding rational series r using a rank factorization PS of $H_\mathcal{B}$. Let H_σ be the sub-block of the prefix closure of $H_\mathcal{B}$ corresponding to the basis $(\mathcal{P}\sigma, \mathcal{S})$, and let $h_{\mathcal{P},\epsilon} \in \mathbb{R}^\mathcal{P}$ denotes the p-dimensional vector with coordinates $h_{\mathcal{P},\epsilon}(u) = r(u)$, and $h_{\epsilon,\mathcal{S}}$ the s-dimensional vector with coordinates $h_{\epsilon,\mathcal{S}}(v) = r(v)$. Then the WA $A = \langle I, (M_\sigma)_{\sigma\in\Sigma}, T\rangle$, with $I^\top = h_{\epsilon,\mathcal{S}}^\top S^+$, $T = P^+ h_{\mathcal{P},\epsilon}$, and $M_\sigma = P^+ H_\sigma S^+$, is minimal for r [Balle et al., 2014]. As usual, N^+ denotes the Moore-Penrose pseudo-inverse of a matrix N.

2.4. Learning weighted automata using spectral learning

The core idea of the spectral learning of weighted automata is to use a rank factorization of a complete sub-block of the Hankel matrix of a target series to induce a weighted automaton.

Of course, in a learning context, one does not have access to the Hankel matrix: all that is available is a (multi-)set of strings $LS = \{x^1, \dots, x^m\}$, usually called a learning sample. The learning process thus relies on *the empirical Hankel matrix* given by $\hat{H}_\mathcal{B}(u, v) = \hat{\mathbb{P}}_{LS}(u, v)$ where $\mathcal{B}$ is a given basis and $\hat{\mathbb{P}}_{LS}$ is the observed frequency of strings inside LS. Hsu et al. [2009] proves that with high probability we have $||H_\mathcal{B} - \hat{H}_\mathcal{B}||_F \leq \mathcal{O}(\frac{1}{\sqrt{m}})$.

In the learning framework we are considering, we suppose that there exists an unknown rational series r of rank n and we want to infer a WA for r. We are assuming that we know a complete basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ and have access to a learning sample LS. Obviously, we can compute sub-blocks H_σ for $\sigma \in \Sigma'$, $h_{\mathcal{P},\epsilon}$, and $h_{\epsilon,\mathcal{S}}$ from LS. Thus, the only thing needed is a rank factorization of $\hat{H}_\mathcal{B} = H_\epsilon$. We are going to use the compact Singular Value Decomposition (SVD).

The SVD of a $p \times s$ matrix H_ϵ of rank n is $H_\epsilon = U\Lambda V^\top$ where $U \in \mathbb{R}^{p\times n}$ and $V \in \mathbb{R}^{s\times n}$ are orthogonal matrices, and $\Lambda \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the singular values of H_ϵ. An important property is that $H_\epsilon = (U\Lambda)V^\top$ is a rank factorization. As V is orthogonal, we have $V^\top V = I$ and thus $V^+ = V^\top$. Using previously described results (see Section 2.3), this allows the inference of a WA $A = \langle I, (M_\sigma)_{\sigma\in\Sigma}, T\rangle$ such that:

$$I^\top = h_{\epsilon,\mathcal{S}}^\top V$$

$$T = (H_\epsilon V)^+ h_{\mathcal{P},\epsilon}$$

$$M_\sigma = (H_\epsilon V)^+ H_\sigma V$$

These equations are what is called the spectral learning algorithm. The Reader interested in more details about spectral learning is referred to the work of Balle et al. [2014].

3. Toolbox description

The Sp2Learn toolbox is made of 5 Python classes and implements several variants of the spectral learning algorithm. Distributed as a baseline for the Sequence PredIction ChallengE (SPiCe), http://spice.lif.univ-mrs.fr, it enjoys an easy installation process and is extremely tunable due to the wild range of parameters allowed.

3.1. The different classes

The corner stone of the toolbox is the `Automaton` class. It implements weighted automata as linear representations and offers valuable methods, such as loading from a file, computing the weight of a string or the sum of the weights of all strings, testing absolute convergence, etc. Two particular methods are worth being detailed. The first one consists in a numerically robust and stable minimization following the work of Kiefer and Wachter [2014]. The second is a transformation method that constructs a WA from a given one computing $r(\cdot)$ such that the new WA computes the prefix weights $r_p(\cdot)$, the suffix ones $r_s(\cdot)$, or the factor ones $r_f(\cdot)$. Moreover, the reverse conversion is also doable with this method.

The second class, `Load`, is a private one that is used to parse a learning sample in the now standard PAutomaC format [Verwer et al., 2014] and to create a dictionary containing the strings of the sample and their number of occurrences.

Another important class is the one named `Sample`. Its main role is to create and to store the data in the needed dictionaries of prefixes, suffixes, or factors, in order to build the Hankel matrix from a sample. The aim is to take into account only the needed information. Therefore, in addition to the path to the sample file, its parameters correspond to the ones of the learning procedure:

- `version` indicates which variant of the Hankel matrix is going to be used (possible values are `classic` for $\hat{H}$, `prefix` for $\hat{H}^p$, `suffix` for $\hat{H}^s$, and `factor` for $\hat{H}^f$). This allows to compute only the needed dictionaries.
- `partial` indicates whether all the elements have to be taken into account in the Hankel matrix.
- `lrows` and `lcolumns` can either be lists of strings that form the basis $\mathcal{B}$ of the sub-block that is going to be considered, or integers corresponding to the maximal length of elements of the basis. In the latter case, all elements present in the sample whose length are smaller than the given values are in the basis. This ensures the basis to be complete if enough data is available. These parameters have to be set only when `partial` is activated.

The generated Python dictionaries contains for each elements its frequency in the sample. Among others not described here, a method is implemented in this class to heuristically select interesting rows and columns given the sample.

The class named `Hankel` is a private class that creates a Hankel matrix from a `Sample` instance containing all needed dictionaries.

Finally, the class `Learning` generates a WA from a sample. When creating an instance of that class, it is required to provide a `Sample` object. The main method of the class `Learning` is `LearnAutomaton`. Parameters of this method are the same than the ones of the instantiation of a `Sample` object, together with the expected rank value and a Boolean specifying whether the Hankel matrix has to be stored in a sparse format. It returns the automaton computed with the requested rank. The class `Learning` implements also some evaluation methods, like perplexity computation for instance.

The class diagram of Sp2Learn is given in Figure 5 in Annex.

3.2. Installing and using Sp2Learn

The installation of the toolbox is made easy by the use of `pip`: one just has to execute `pip install Sp2Learning` in a terminal. If needed, the package can be downloaded at https://pypi.python.org/pypi/Sp2Learning.
A technical documentation is available at http://pythonhosted.org/Sp2Learning.

The following code corresponds to a use case in a python interpreter:

```
>>> from sp2learn import Learning, Sample
>>> rank = 17
>>> lrows = 4
>>> lcolumns = 5
>>> version = "factor"
>>> partial = True
>>> train_file = "1.pautomac.train"
>>> LS = Sample(adr=train_file, lrows=lrows, lcolumns=lcolumns,
                version=version, partial=partial)
>>> sptrl = Learning(sample_instance=LS)
>>> A = sptrl.LearnAutomaton(rank=rank, lrows=lrows,
                             lcolumns=lcolumns, version=version,
                             partial=partial, sparse=True)
```

In this code, the 8 first lines defined the parameters of the spectral learning, the 9th instruction is the creation of an instance of the class `Sample`, the 10th of an instance of the class `Learning`, and the last one runs the learning method. This example corresponds to the learning of a WA on Problem 1 of the PAutomaC competition (see Section 4), using a sparse empirical Hankel matrix $\hat{H}^f$ with a basis containing all factors in the learning sample up to size 4 for rows and 5 for columns.

In the context of the SPiCe competition, the interest was put on the probability of prefixes. One can transform the automaton A so that it computes the prefix probability:

```
>>> Ap = A.transformation(source="classic", target="prefix")
>>> A.val([1, 0, 2, 2])
>>> Ap.val([1, 0, 2, 2])
```

The last line computes $r_p(1022)$ while the previous one gives $r(1022)$, where r is the series represented by the learned automaton A.

4. Experiments

We tested the Sp2Learn toolbox on the 48 synthetic problems of the PAutomaC competition [Verwer et al., 2014]. These data sets correspond to randomly generated sets of strings from randomly generated probabilistic finite states machines: Hidden Markov Models (HMM), Probabilistic Automata (PA), and Deterministic Probabilistic Automata (DPA). Several sparsity parameters were used to generate various machines for each models.

4.1. Settings

For each problem, PAutomaC provides a training sample (11 with 100 000 strings, the rest with 20 000 strings), a test set of 1 000 strings, the finite state machine used to generate the data, and the probability in this target machine of each string in the test set.

We ran the toolbox on the 48 data sets using the 4 different variants of the (sparse) Hankel matrix. On each problem and for each version, we made the maximal size of elements used for the matrix range from 2 to 6 (these are parameters `lrows` and `lcolumns` of the toolbox). For each of these values, all ranks between 2 and 40 were tried. This represents 28 032 runs of the toolbox, to which we have to subtract 631 runs that correspond to rank values too large comparing to the size of the Hankel matrix. All these computations were done on a cluster and each process was allowed 20Go of RAM and 4 hours computation on equivalent CPUs.

We evaluated the quality of the learning using perplexity following what was done for the competition. Given a test set TS, it is given by the formula:

$$perplexity(C, TS) = 2^{-(\sum_{x \in TS} \mathbb{P}_T(x) * log(\mathbb{P}_C(x)))}$$

where $\mathbb{P}_T(x)$ is the true probability of x in the target model and $\mathbb{P}_C(x)$ is the candidate probability, that is the one given by the learned WA. Both probabilities have to be normalized to sum to 1 on strings of TS.

Finally, the main problem of spectral learning of weighted automata is that some strings can have negative weights, if not enough data is available. To tackle this issue, we trained a 3-gram using the same data and replaced the output of the learned weighted automata by the 3-gram one each time it gave a negative weight. We carefully kept track of this behavior and detail the result in the next section.

4.2. Results

We want first to notice that the aim of these experiments was to show the global behavior of the toolbox on a broad and vast class of problems. Indeed, spectral learning algorithms are usually used as a first step of a learning process, usually followed by a smoothing phase that tunes the weights of the model without modifying its structure (Gybels et al. [2014] use for instance a Baum-Welch approach for that second step). We did not work on that since the aim was to show the potentiality of the toolbox.

However, the results of the best run on each problem, given in Table 1 and Table 2 (in Annex), show that even without a smoothing phase the toolbox can obtain perplexity scores close to the optimal ones. This would not have permitted the winning of the competition, but it is good enough to be noticed.

The runs realized using the toolbox also allow to evaluate the impact of the value given to the rank parameter. Figure 2 shows the evolution of perplexity on the problems whose target machines are Probabilistic Automata (top) and Deterministic Probabilistic Automata (bottom). This curves were obtained using the classic version of the Hankel matrix for DPA and the factor variant for PA. In both cases, values of `lrows` and `lcolumns` were set to 5.

These results show that in a first phase the perplexity can oscillate when the rank increases. However, in a second phase, the perplexity seems to decrease to a minimal value and then stay stable. This second step is likely to be reached when the value of the rank parameter is equal to the rank of the target machine. At that moment, inferred singular values correspond to the target ones and adding other values later has no impact. Indeed, if the empirical Hankel matrix was the target one, these values would be null. But even if it is not the case, which is likely in this experimental context, the results show that their values are small enough to not degrade the quality of the learning.

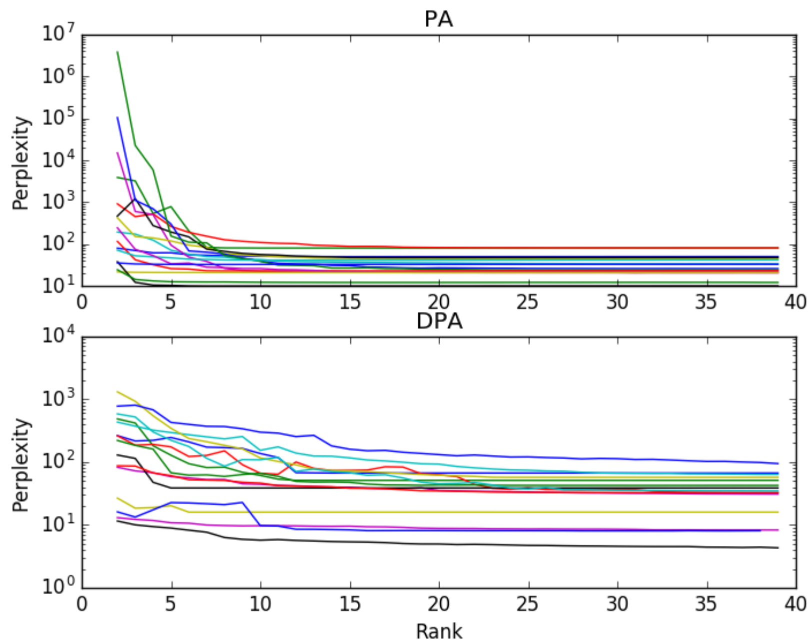


Figure 2: Perplexity evolution with the rank on problems whose targets are Probabilistic Automata (top) and Deterministic Probabilistic Automata (bottom). Each line corresponds to one PAutomaC problem.

Figure 3 shows the learning time of the toolbox on the PAutomaC problems. Each point corresponds to the average computation time of all runs using a given version of the Hankel matrix, in seconds. Clearly, the classic version is the fastest while the factor one is the slowest, suffix and prefix ones are standing in a middle ground. This is expected since the factor version is the less sparse of the Hankel matrix variants. These results seem to show that the classic version is 100 times faster than the factor one. Another not really surprising observation is that the behavior of the prefix and suffix versions are extremely close.

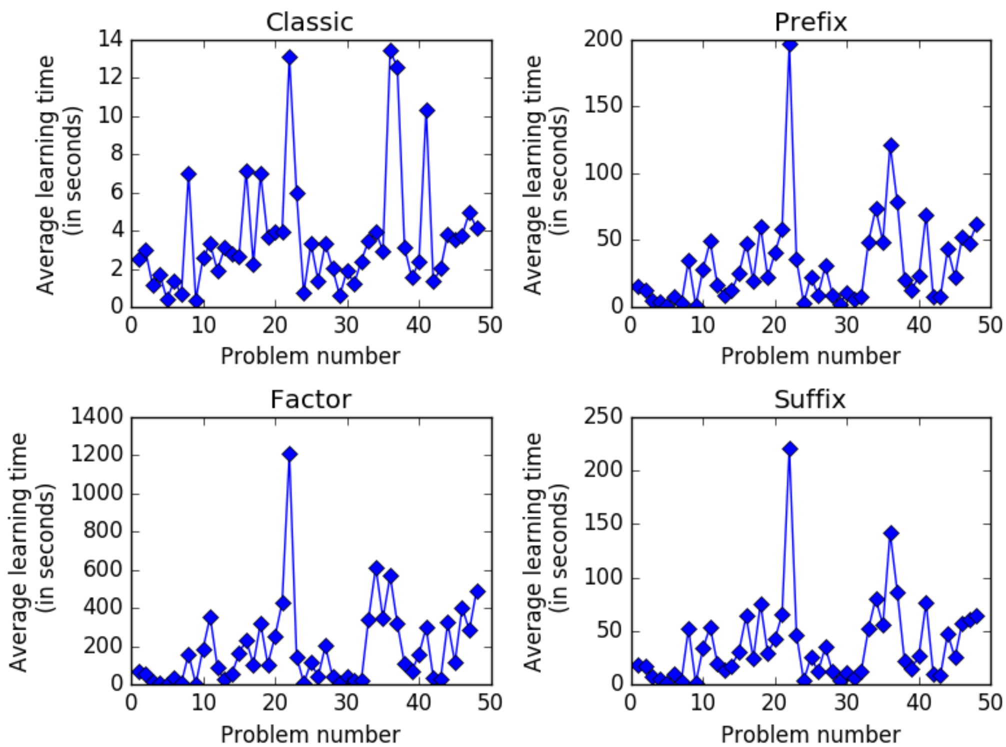


Figure 3: Average learning time using the 4 different variants of the Hankel matrix on the 48 PAutomaC problems.

Globally, these values show that the running time of the toolbox is reasonable, even for the slowest variant: on all but one problem the factor version took less than an hour and a half on average.

Finally, Figure 4 shows the average percentage of the 3-gram uses to find the probability of a test string. Remember that this happens for strings on which the learned automata returns a non-positive weight. These percentages are given for each possible rank parameter. Each curve corresponds to a given value of the size parameter, that is the maximal length of elements taken into account to build the Hankel matrix.

Globally, the use of 3-gram is quite rare, as less than 1.3% of strings requires its use. On the one hand, models built on large Hankel matrices tend to need less uses of 3-gram. On the other hand, models made using a large rank seem to require slightly more uses of 3-gram. This might be due to the overfitting that may occur when the rank parameter is set higher than the actual rank. Notice that no result is possible with large ranks for Hankel matrices made of too few rows and columns: as the dimensions of the matrix are smaller than the asked rank, a SVD cannot be computed.

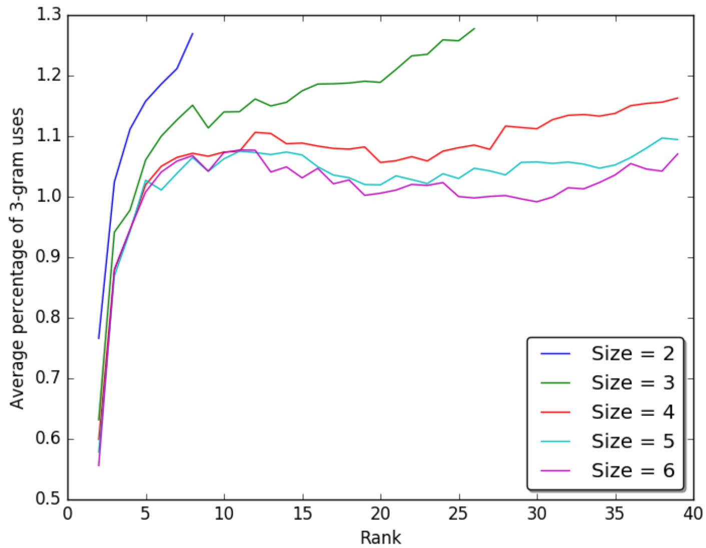


Figure 4: Average percentage of uses of 3-gram to find the probability of a test string giving different values of the rank parameter. Each curve corresponds to a different value of the maximal length of elements of the Hankel matrix.

5. Future developments

The version of the Sp2Learn presented here is 1.1. We are currently working on a different version that will be usable in the same way than the well-known statistical machine learning toolbox Scikit-learn [Pedregosa et al., 2011]. This will allow the use of the tuning functions of Scikit-learn, like cross-validation and grid search. Given the large public using Scikit-learn, this could convince the statistical machine learning community to get interested in spectral learning.

We are also planning to develop new useful methods, starting with a Baum-Welch one, that would complete the learning process by allowing a smoothing phase after the spectral learning one. We might also turn our attention to closely related and promising new algorithms, like the one of Glaude and Pietquin [2016].

References

R. Bailly. *Méthodes spectrales pour l'inférence grammaticale probabiliste de langages stochastiques rationells.* PhD thesis, Aix-Marseille University, 2011.

R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *26th International Conference on Machine Learning*, pages 33–40, 2009.

R. Bailly, A. Habrard, and F. Denis. *A Spectral Approach for Probabilistic Grammatical Inference on Trees*, pages 74–88. Springer Berlin Heidelberg, 2010.

B. Balle, X. Carreras, F. Luque, and A. Quattoni. Spectral learning of weighted automata. *Machine Learning*, 96(1-2):33–63, 2014.

A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. On the applications of multiplicity automata in learning. In *37th Annual Symposium on Foundations of Computer Science, FOCS*, pages 349–358, 1996.

J. Berstel and C. Reutenauer. *Rational series and their languages*. EATCS monographs on theoretical computer science. Springer-Verlag, 1988.

R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *2nd International Colloquium on Grammatical Inference, ICGI*, volume 862 of *LNAI*, pages 139–150. Springer-Verlag, 1994.

F. Denis, Y. Esposito, and A. Habrard. 19th annual conference on learning theory (COLT) 2. pages 274–288. Springer Berlin Heidelberg, 2006.

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.

H. Glaude and O. Pietquin. PAC learning of probabilistic automaton based on the method of moments. In *33rd International Conference on Machine Learning*, pages 820–829, 2016.

M. Gybels, F. Denis, and A. Habrard. Some improvements of the spectral learning approach for probabilistic grammatical inference. In *12th International Conference on Grammatical Inference, ICGI*, pages 64–78, 2014.

D. Hsu, S. Kakade, and T. Zhang. A spectral algorithm for learning hidden markov models. In *Conference on Computational Learning Theory (COLT)*, 2009.

S. Kiefer and B. Wachter. *Stability and Complexity of Minimising Probabilistic Automata*, pages 268–279. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

M. Mohri. *Handbook of Weighted Automata*, chapter Weighted Automata Algorithms, pages 213–254. Springer Berlin Heidelberg, 2009.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245 – 270, 1961.

S. Siddiqi, B. Boots, and G. Gordon. Reduced-rank hidden Markov models. In *13th International Conference on Artificial Intelligence and Statistics*, 2010.

S. Verwer, R. Eyraud, and C. de la Higuera. PAutomaC: a probabilistic automata and hidden markov models learning competition. *Machine Learning*, 96(1-2):129–154, 2014.

Annex

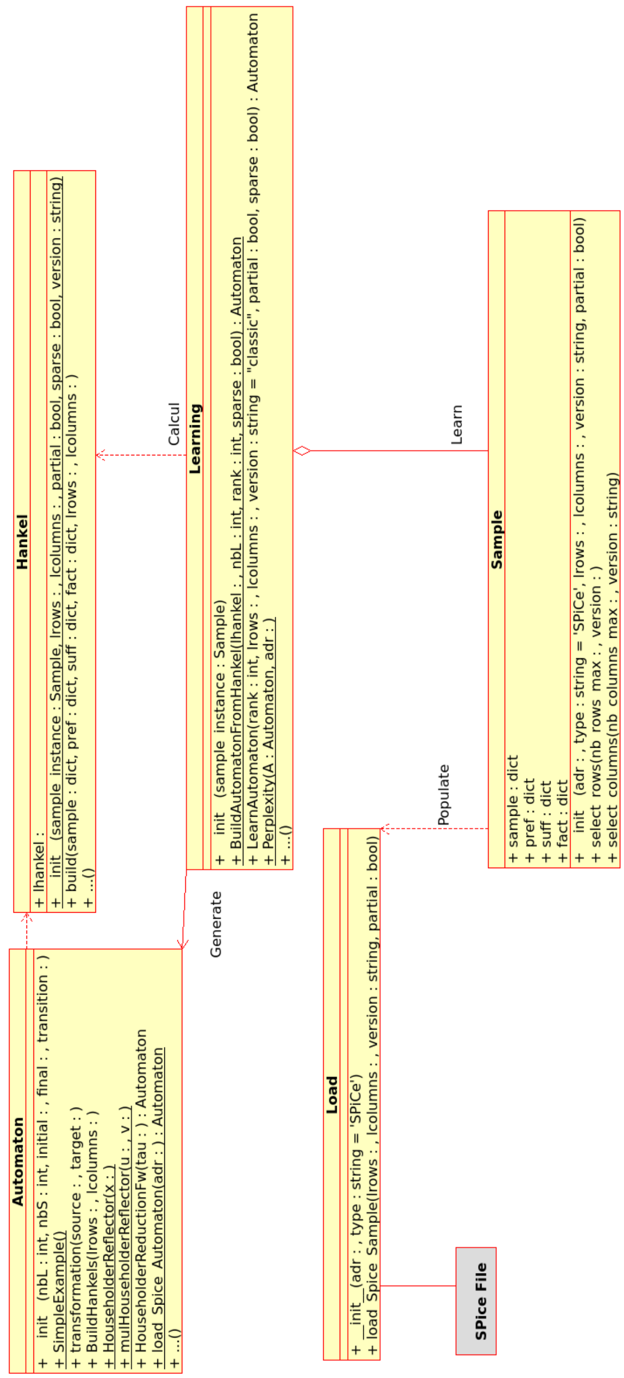


Figure 5: Class diagram of Sp2Learn.

Problem	Solution	Perplexity	Version	Rank	Size	Time
1	29.8978935527	30.4365057849	classic	22	3	0.917114973068
2	168.330805339	168.496533827	factor	30	6	130.30452013
3	49.956082986	50.2761373276	factor	29	4	7.71120405197
4*	80.8184226132	80.8558017072	factor	11	6	9.42034196854
5	33.2352988504	33.2390299543	factor	9	6	1.26393389702
6	66.9849579244	67.0522875724	factor	18	6	59.2427239418
7	51.2242694583	51.2542707632	factor	12	6	8.79874491692
8*	81.3750634047	81.6742917934	classic	35	6	17.385874033
9	20.8395901703	21.0402440195	factor	34	6	1.72923922539
10	33.3030058501	33.99830351	classic	39	4	1.75096201897
11	31.8113642161	32.4618574848	factor	38	4	260.691053867
12	21.655287002	21.6701591045	factor	17	5	94.2339758873
13*	62.8058396015	63.0458473691	classic	39	6	3.14246487617
14	116.791881846	116.854374304	factor	7	6	65.0472741127
15	44.2420495474	44.3621902064	factor	30	4	99.350990057
16*	30.7110624887	30.851413939	classic	39	4	7.41966795921
17	47.3112160937	47.4890664272	factor	26	5	127.645046949
18*	57.3288608287	57.3331676752	factor	24	5	284.276638985
19	17.8768660563	17.9142466522	factor	39	5	101.450515032
20	90.9717263176	91.5984851723	factor	8	3	31.4403400421
21	30.518860165	32.0618301238	factor	34	4	283.474653006
22*	25.9815361778	26.1277647043	classic	36	4	8.18691301346
23	18.4081615041	18.4352765238	factor	32	5	190.789381981
24	38.7287795405	38.761782116	factor	7	5	9.56825995445
25	65.7350539501	66.2159906104	factor	23	3	14.8863971233
26	80.7427626831	84.9777239451	classic	39	6	2.27566695213
27	42.427078513	42.6194221003	factor	39	4	166.635137081
28	52.7435104626	53.1105801399	factor	15	3	4.2344379425
29	24.0308339109	24.084604107	factor	39	5	17.505715847
30	22.925985377	23.0171753219	factor	24	3	8.12572789192

Table 1: Best obtained results on the first 30 data sets of the PAutomaC competition. Column Solution corresponds to the minimal perplexity, *i.e.* the one of the target machine; Column Perplexity is the perplexity obtained by the best run of the toolbox; Version indicates which version of the Hankel matrix was used; Rank gives the value of parameter rank for that run; Size is the maximal length of elements considered to build the Hankel matrix; Time is the computation time of the run, given in seconds. Problem numbers marked with a star are the ones whose training set contains 100 000 strings (the other are made of 20 000 sequences).

Problem	Solution	Perplexity	Version	Rank	Size	Time
31	41.2136431636	41.4211835629	factor	14	4	8.21831202507
32*	32.6134162732	32.6754979794	factor	39	6	61.3909471035
33	31.8650289444	31.9141918025	factor	21	3	40.8976488113
34	19.9546848395	20.7051491426	classic	34	4	2.75224304199
35	33.776935538	34.6956389516	classic	39	4	2.0509660244
36*	37.985692906	38.1214706816	classic	11	3	5.38864707947
37*	20.9797622037	21.0288128706	classic	11	4	7.00510692596
38	21.4457989928	21.5279850109	classic	3	2	0.706127166748
39	10.0020442634	10.002996462	factor	6	6	89.0673789978
40	8.2009545433	8.31253842976	factor	39	4	176.795210123
41*	13.9124713717	13.9384593977	classic	8	3	3.88122415543
42	16.0037636643	16.0087620127	factor	7	3	2.48271298409
43	32.6370243149	32.8343363438	classic	6	6	1.82530999184
44	11.7089059654	11.8353479581	classic	6	3	1.67373609543
45	24.0422109361	24.0468379329	factor	3	3	27.9223351479
46	11.9819819343	12.0312721955	factor	38	4	331.469185114
47*	4.1189756456	4.17530125251	classic	39	6	6.05857491493
48	8.0362199917	8.05347816088	factor	33	5	766.088064194

Table 2: Best obtained results on the last 18 data sets of the PAutomaC competition.