Classifying Words with 3-sort Automata

Tomasz Jastrząb, Frédéric Lardeux, Éric Monfroy

Classifying Words with 3-sort Automata

Tomasz Jastrząb[1][a], Frédéric Lardeux[2][b] and Éric Monfroy[2][c]
[1]*Silesian University of Technology, Gliwice, Poland*
[2]*LERIA, University of Angers, Angers, France*
Tomasz.Jastrzab@polsl.pl, {Frederic.Lardeux, Eric.Monfroy}@univ-angers.fr

Keywords: Grammatical Inference, Nondeterministic Automata, SAT Models.

Abstract: Grammatical inference consists in learning a language or a grammar from data. In this paper, we consider a number of models for inferring a non-deterministic finite automaton (NFA) with 3 sorts of states, that must accept some words, and reject some other words from a given sample. We then propose a transformation from this 3-sort NFA into weighted-frequency and probabilistic NFA, and we apply the latter to a classification task. The experimental evaluation of our approach shows that the probabilistic NFAs can be successfully applied for classification tasks on both real-life and superficial benchmark data sets.

1 INTRODUCTION

Many real-world phenomena may be represented as syntactically structured sequences, e.g., DNA, natural language sentences, electrocardiograms, and chain codes. Grammatical Inference refers to learning grammars and languages from data, i.e., from such syntactically structured sequences. Machine learning of grammars has various applications in syntactic pattern recognition, adaptive intelligent agents, computational biology, and prediction.We are interested in learning grammars as finite state automata, with respect to a sample of the language composed of positive sequences that must be elements of the language, and negative ones that the automaton must reject.

The problem of learning finite automata has been studied from various angles: ad-hoc methods such as DeLeTe2 (Denis et al., 2004) which merges states from the prefix tree acceptor (PTA), a family of algorithms for regular language inference presented in (Vázquez de Parga et al., 2006), metaheuristics such as hill-climbing in (Tomita, 1982), or modeling the problem as a Constraint Satisfaction Problem (CSP) and solving it with generic tools (such as non-linear programming (Wieczorek, 2017), or Boolean formulas (Jastrząb, 2017; Jastrząb et al., 2023; Lardeux and Monfroy, 2021; Jastrząb et al., 2022)).

However, all these works consider Deterministic Finite Automata (DFA), or Non-deterministic Finite Automata (NFA). In both cases, this means that when using the automata on a word, the answer is "Yes, this word is part of the language", or "No, this word is not part of the language". Since samples are finite and usually limited in size (hundreds of words at most), and regular languages are infinite, this classification may be too restrictive. One could be interested in probabilistic answers such as "this word is part of the language with a probability of x%". The question is thus "How can we learn a probabilistic automaton from a sample of positive and negative words?".

In this paper, we propose a technique to derive a probabilistic automaton from a sample of positive and negative words. We first learn Non-deterministic Finite Automata with 3 sorts of states: accepting final states which validate positive words, rejecting final states which reject negative words, and whatever states that are not conclusive. We use these 3-sort NFA which seems of reasonable use for our goal (the usefulness of this kind of NFA is presented in (de la Higuera, 2010)). To improve the efficiency for generating such automata, we use a similar property to the one that was used for 2-sort NFA in (Jastrząb et al., 2023): here, we build a 3-sort automaton with only one accepting final state and one rejecting final state, and some extra constraints to reduce this size $k+2$ automaton into a size k automaton. Then, we want to reflect frequencies based on the sample, such as: how many positive words of the sample terminate in this accepting final state? How many times

[a] https://orcid.org/0000-0002-7854-9058
[b] https://orcid.org/0000-0001-8636-3870
[c] https://orcid.org/0000-0001-7970-1368

has a negative word of the sample passed by this transition? But we also want to be a bit more specific. For example, how many negative words of the sample passed by this transition and terminated in a rejecting final state (vs. a whatever state). To this end, we need to weigh differently some cases and patterns. We thus need to define what we call 3-sort Weighted-Frequency NFA, and we present the transformation of a 3-sort NFA into a 3-sort Weighted-Frequency NFA. This last can then be converted into a probabilistic NFA once weights have been instantiated.

The probabilistic NFA can then be used to determine the probability for a word to be a part of the language, or the probability of it not being a part of the language. Note that by modifying the weights, we can obtain more accepting or more rejecting automata.

We conduct a number of experiments on the Waltz DB database to classify peptides into amyloid ones that are dangerous, or non-amyloid ones that are harmless. We perform similar studies with languages generated by a regular expression. Our results look promising, leaving also some space for weights tuning depending on the aim of the classification, e.g., we can be safer (rejecting dangerous peptides and some harmless ones) or more risky (trying not to reject non-amyloid peptides).

The paper is organized as follows. In Sect. 2 we revise the already developed inference models and propose modifications required to construct 3-sort NFAs. In Sect. 3 we show how the 3-sort NFA can be transformed into weighted-frequency NFA and finally into probabilistic NFA. In Sect. 4 we describe conducted experiments and discuss obtained results. Finally, we conclude in Sect. 5.

2 THE NFA INFERENCE PROBLEM: FIRST MODELS

In this section, we formally present the NFA inference problem based on the propositional logic paradigm and we provide several models. These new models are similar to the ones of (Jastrząb et al., 2023) but using 3-sort non-deterministic automata.

Without the loss of generality[1], we consider in the following, that λ, the empty word, is not part of the sample. We also consider a unique initial state, q_1.

[1] If $\lambda \in S$, then it can be recognized or rejected directly, without the need of an automaton.

2.1 Notations

Let $\Sigma = \{s_1, \ldots, s_n\}$ be an alphabet of n symbols, and let λ denote the empty word, K be the set of integers $\{1, \ldots, k\}$, $Pref(w)$ (resp. $Suf(w)$) be the set of prefixes (resp. suffixes) of the word w, that we extend to $Pref(W)$ (resp. $Suf(W)$) for a set of words W.

Definition 1. *A 3-sort non-deterministic finite automaton (3NFA) is a 6-tuple $\mathcal{A} = (Q, \Sigma, I, F_+, F_-, \delta)$ with: $Q = \{q_1, \ldots, q_k\}$ – a finite set of states, Σ – a finite alphabet, I – the set of initial states, F_+ – the set of accepting final states, F_- – the set of rejecting final states, and $\delta : Q \times \Sigma \to 2^Q$ – the transition function. Note that in what follows, we will consider only one initial state, i.e., $I = \{q_1\}$.*

A learning sample $S = S^+ \cup S^-$ is given by a set S^+ of "positive" words from Σ^* that the inferred 3-sort NFA must accept, and a set S^- of "negative" words that it must reject.

The language recognized by $\mathcal{A}$, $L(\mathcal{A})_A$, is the set of words for which there exists a sequence of transitions from q_1 to a state of F_+. The language rejected by $\mathcal{A}$, $L(\mathcal{A})_R$, is the set of words for which there exists a sequence of transitions from q_1 to a state of F_-.

An automaton is non-ambiguous if $L(\mathcal{A})_A \cap L(\mathcal{A})_R = \emptyset$, i.e., no positive word terminates in a rejecting final state, and no negative word terminates in an accepting final state.

We discard models with 0/1 variables, either from INLP (Wieczorek, 2017) or CSP (Rossi et al., 2006): we made some tests with various models with PyCSP (Lecoutre and Szczepanski, 2020) and obtained some very poor results: the NFA inference problem is intrinsically a Boolean problem, and thus, well suited for SAT solvers. Hence, we consider the following variables:

- k, an integer, the size of the 3NFA to be generated,
- a set of k Boolean variables $F_+ = \{a_1, \ldots, a_k\}$ determining whether state i is a final accepting state or not,
- a set of k Boolean variables $F_- = \{r_1, \ldots, r_k\}$ determining if state i is rejecting,
- $\Delta = \{\delta_{s,\overrightarrow{q_i,q_j}} |\ s \in \Sigma \text{ and } (i,j) \in K^2\}$, a set of nk^2 Boolean variables representing the existence of transitions from state q_i to state q_j with the symbol $s \in \Sigma$, for each i, j, and s.
- we define $\rho_{w,\overrightarrow{q_1,q_{m+1}}}$ as the path $q_1, \ldots, q_{m+1}$ for a word $w = s_1 \ldots s_m$: $\rho_{w,\overrightarrow{q_1,q_{m+1}}} = \delta_{s_1,\overrightarrow{q_1,q_2}} \wedge \ldots \wedge \delta_{s_m,\overrightarrow{q_m,q_{m+1}}}$.

Although the path is directed from q_1 to q_{m+1} (it is a sequence of derivations), we will build it either starting from q_1, starting from q_{m+1}, or starting from both

sides (Jastrząb et al., 2022). Thus, to avoid confusion, we prefer keeping $\overline{q_1, q_{m+1}}$ without any direction. Paths will be built recursively, and we need at most $O(\sigma k^2)$ Boolean variables $\rho_{s,\overline{i,j}}$, with $\sigma = \sum_{w \in S} |w|$.

2.2 Core of the Models

The core of the models is independent of the way the paths are built. It will thus be common to each model. The core to define a 3NFA of size k (noted k_3NFA) can be defined as follows:

- a final state must be either accepting or rejecting:

$$\bigwedge_{i \in K} \neg(a_i \wedge r_i) \tag{1}$$

- a positive word must terminate in an accepting final state of the 3NFA, i.e., there must be a path from state q_1 to a final state i, such that $i \in F_+$:

$$\bigvee_{i \in K} \rho_{w,\overline{q_1,q_i}} \wedge a_i \tag{2}$$

- to build a non-ambiguous NFA, a positive word cannot terminate in a rejecting final state:

$$\bigwedge_{i \in K} (\neg \rho_{w,\overline{q_1,q_i}} \vee \neg r_i) \tag{3}$$

- similarly, negative words need the same constraints swapping rejecting and accepting:

$$\bigvee_{i \in K} \rho_{w,\overline{q_1,q_i}} \wedge r_i \tag{4}$$

$$\bigwedge_{i \in K} (\neg \rho_{w,\overline{q_1,q_i}} \vee \neg a_i) \tag{5}$$

Of course, the notion of a path can be defined and built in many ways, see (Jastrząb et al., 2022) for prefix, suffix, and hybrid approaches.

2.3 Building Paths

We consider here that a path for a word $w = uv$ is built as the concatenation of a path for the prefix u and one for the suffix v. Thus, we can have several joining states (j below) and ending states (k below) for the various paths of a word $w = uv$:

$$\bigvee_{(j,k) \in K^2} \rho_{u,\overline{q_1,q_j}} \wedge \rho_{v,\overline{q_j,q_k}} \tag{6}$$

Note that for the empty word λ we impose that[2]:

$$\rho_{\lambda,\overline{q_i,q_j}} = True \qquad \forall (i,j) \in K^2, \tag{7}$$

[2] As said before, we consider that $\lambda \notin S$, but when splitting a word, its prefix or suffix may be λ.

to ensure that splitting a word in such a way that the prefix or suffix is the empty word is valid. Note, however, that we do not allow λ-transitions in the NFA.

Prefixes and suffixes are then built recursively, starting from the beginning of words for prefixes:

- For each prefix $u = s$, $s \in \Sigma$:

$$\bigwedge_{i \in K} \delta_{s,\overrightarrow{q_1,q_i}} \leftrightarrow \rho_{s,\overline{q_1,q_i}} \tag{8}$$

- for each prefix $u = xs$, $s \in \Sigma$ of each word of S:

$$\bigwedge_{i \in K} \left(\rho_{u,\overline{q_1,q_i}} \leftrightarrow \left(\bigvee_{j \in K} \rho_{x,\overline{q_1,q_j}} \wedge \delta_{s,\overrightarrow{q_j,q_i}} \right) \right) \tag{9}$$

and from the ending of words for suffixes:

- for $v = s$, and $s \in \Sigma$

$$\bigwedge_{(i,j) \in K^2} \delta_{s,\overrightarrow{q_i,q_j}} \leftrightarrow \rho_{s,\overline{q_i,q_j}} \tag{10}$$

- for each suffix $v = sx$, $s \in \Sigma$:

$$\bigwedge_{(i,j) \in K^2} \left(\rho_{v,\overline{q_i,q_j}} \leftrightarrow \left(\bigvee_{l \in K} \delta_{s,\overrightarrow{q_i,q_l}} \wedge \rho_{x,\overline{q_l,q_j}} \right) \right) \tag{11}$$

2.4 The Models

We build the models similarly to standard NFA (i.e., without the notion of rejecting states). This means that we have to determine where to split each word $w \in S$ into a prefix u and a suffix v. We then consider the set of prefixes $S_u = \{u | w \in S \text{ and } w = uv\}$ and $S_v = \{v | w \in S \text{ and } w = uv\}$ the set of suffixes. Then, the model is the conjunction of Constraints (1)–(11).

Based on Constraint (7), if we split each word w as $w = w\lambda$, we obtain the prefix model P whose spatial complexity is in $O(\sigma k^2)$ clauses, and variables, with $\sigma = \sum_{w \in S} |w|$. Similarly, by splitting words as $w = \lambda w$, we obtain the suffix model S whose spatial complexity is in $O(\sigma k^3)$ clauses, and variables (the difference is because we do not know where a word terminates).

We then have hybrid models with non-empty suffixes and prefixes. Their complexity is in $O(\sigma k^3)$:

- the best suffix model $S^\star$ which consists in determining a minimal set of suffixes covering each word of S and maximizing a cost based on an order over suffixes ($\Omega(v) = \{w \in S \mid v \text{ is a suffix of } w\}$ and considering two suffixes v_1 and v_2, $v_1 \succcurlyeq v_2 \Leftrightarrow |v_1| \cdot |\Omega(v_1)| \geq |v_2| \cdot |\Omega(v_2)|$. Then, prefixes are computed to complete words.
- similarly, the best prefix model $P^\star$ is built optimizing prefixes.

- we can also try to optimize each word splitting using some metaheuristics, for example, iterated local search (ILS). The model $ILS(Init)$, based on a local search optimization (Stützle and Ruiz, 2018) of word splittings (starting with an initial configuration $Init$, being either a random splitting of words, the splitting found by the $P^\star$ model, or by the $S^\star$ model), tries to minimize the fitness function $f(S_p,S_s) = |Pref(S_p)| + k \cdot |Suf(S_s)|$.

2.5 From $O(k^3)$ to $O((k+2)^2)$

Consider a sample S. If there is a k_3NFA for S, i.e., a 3NFA of size k, to recognize words of S^+ and reject words of S^-, there is also a $(k+2)$_3NFA for S. We can refine this property by adding some constraints to build what we call $(k+2)$_NFA* extensions.

Let $\mathcal{A} = (Q,\Sigma,\{q_1\},F_+,F_-,\delta)$ be a k_3NFA. Then, there always exists a $(k+2)$_3NFA, $\mathcal{A}' = (Q \cup \{q_{k+1},q_{k+2}\},\Sigma,\{q_1\},\{q_{k+1}\},\{q_{k+2}\},\delta')$, such that:

- there is only one final accepting state q_{k+1} and one rejecting state q_{k+2}, thus we do not need anymore the a_i and r_i variables,
- each transition is copied:
$$\forall_{i,j\in K^2,s\in\Sigma}\ \delta_{s,\overrightarrow{q_i,q_j}} \leftrightarrow \delta'_{s,\overrightarrow{q_i,q_j}}\ ,$$
- incoming transitions to accepting final state are duplicated to the new accepting final state q_{k+1}:
$$\forall_{i\in K,q_j\in F_+}\ \delta_{a,\overrightarrow{q_i,q_j}} \leftrightarrow \delta'_{a,\overrightarrow{q_i,q_{k+1}}}\ ,$$
- the same transition duplication is made for rejecting final states to the new rejecting final state q_{k+2},
- there is no outgoing transition from states q_{k+1} and q_{k+2} ,
- no negative (resp. positive) words terminate in the states from F_+ (resp. F_-). This is obvious in $\mathcal{A}$, we have to make it effective in $\mathcal{A}'$.

The interest of this $(k+2)$_3NFA for S is that the complexity for building suffixes is now in $O(k^2)$ since both positive and negative words must terminate in a given state (resp. q_{k+1} and q_{k+2}).

We now give the constraints of the $(k+2)$_3NFA. Let $K_+ = \{1,2,\ldots,k+2\}$:

- Constraint (1) disappears,
- Constraints (2) and (3) become, for each $w \in S^+$:
$$\rho_{w,\overline{q_1,q_{k+1}}} \tag{12}$$
$$\neg\rho_{w,\overline{q_1,q_{k+2}}} \tag{13}$$
- the same happens for Constraints (4)–(5), replaced by, for $w \in S^-$:
$$\rho_{w,\overline{q_1,q_{k+2}}} \tag{14}$$
$$\neg\rho_{w,\overline{q_1,q_{k+1}}} \tag{15}$$
- Constraints (6) must be split into two, for positive (16) and negative (17) words:
$$\bigvee_{j\in K} \rho_{u,\overline{q_1,q_j}} \wedge \rho_{v,\overline{q_j,q_{k+1}}} \tag{16}$$
$$\bigvee_{j\in K} \rho_{u,\overline{q_1,q_j}} \wedge \rho_{v,\overline{q_j,q_{k+2}}} \tag{17}$$
- Constraints (8)–(9) are not modified
- Constraints (10)–(11) are respectively modified for positive words into:
$$\bigwedge_{i\in K} \delta_{s,\overrightarrow{q_i,q_{k+1}}} \leftrightarrow \rho_{s,\overline{q_i,q_{k+1}}} \tag{18}$$
$$\bigwedge_{i\in K}\left(\rho_{v,\overline{q_i,q_{k+1}}} \leftrightarrow \left(\bigvee_{j\in K} \delta_{s,\overrightarrow{q_i,q_j}} \wedge \rho_{x,\overline{q_j,q_{k+1}}}\right)\right) \tag{19}$$
and for negative words into:
$$\bigwedge_{i\in K} \delta_{s,\overrightarrow{q_i,q_{k+2}}} \leftrightarrow \rho_{s,\overline{q_i,q_{k+2}}} \tag{20}$$
$$\bigwedge_{i\in K}\left(\rho_{v,\overline{q_i,q_{k+2}}} \leftrightarrow \left(\bigvee_{j\in K} \delta_{s,\overrightarrow{q_i,q_j}} \wedge \rho_{x,\overline{q_j,q_{k+2}}}\right)\right) \tag{21}$$
- There is no outgoing transition from the final states:
$$\bigwedge_{s\in\Sigma}\bigwedge_{i\in K_+} \neg\delta_{s,\overrightarrow{q_{k+1},q_i}} \wedge \neg\delta_{s,\overrightarrow{q_{k+2},q_i}} \tag{22}$$
- Each incoming transition of the accepting (resp. rejecting) final state q_{k+1} (resp. q_{k+2}) must also terminate in another state (duplication):
$$\bigwedge_{s\in\Sigma}\left(\bigwedge_{i\in K}\left(\delta_{s,\overrightarrow{q_i,q_{k+1}}} \rightarrow \bigvee_{j\in K} \delta_{s,\overrightarrow{q_i,q_j}}\right)\right) \tag{23}$$
$$\bigwedge_{s\in\Sigma}\left(\bigwedge_{i\in K}\left(\delta_{s,\overrightarrow{q_i,q_{k+2}}} \rightarrow \bigvee_{j\in K} \delta_{s,\overrightarrow{q_i,q_j}}\right)\right) \tag{24}$$

In order to be able to reduce the $(k+2)$_3NFA* into a k_3NFA, we must take care about the possibly rejecting and accepting final states of the k_3NFA. To this end, we need a new set of Boolean variables representing possibly accepting (resp. rejecting) final states for the corresponding k_NFA: $\{a_1^\star,\cdots,a_k^\star\}$ (resp. $\{r_1^\star,\cdots,r_k^\star\}$). The $(k+2)$_3NFA* may be reduced to a k_3NFA by just removing states q_{k+1}, q_{k+2}, and their incoming transitions, and by fixing the final states among the possible final states, i.e., determining the $a_i^\star$ and the $r_i^\star$ which are final states of the k_3NFA, either accepting or rejecting. To determine these possible final states, we have to ensure:

- A negative (resp. positive) word cannot terminate in an accepting (resp. rejecting) possible final state:

$$\bigwedge_{i\in K}\left(a_i^\star \rightarrow \bigwedge_{w\in S^-} \neg\rho_{w,\overrightarrow{q_1,q_i}}\right) \tag{25}$$

$$\bigwedge_{i\in K}\left(r_i^\star \rightarrow \bigwedge_{w\in S^+} \neg\rho_{w,\overrightarrow{q_1,q_i}}\right) \tag{26}$$

  Note that with Constraints (25)–(26), Constraints (3)–(5) are no longer needed.

- Each accepting (resp. rejecting) possible final state validates at least one positive (resp. negative) word of S:

$$\bigwedge_{i\in K} a_i^\star \rightarrow \bigvee_{vs\in S^+}\bigvee_{j\in K}\left(\rho_{v,\overrightarrow{q_1,q_j}}\wedge\delta_{s,\overrightarrow{q_j,q_i}}\wedge\delta_{s,\overrightarrow{q_j,q_{k+1}}}\right) \tag{27}$$

$$\bigwedge_{i\in K} r_i^\star \rightarrow \bigvee_{vs\in S^-}\bigvee_{j\in K}\left(\rho_{v,\overrightarrow{q_1,q_j}}\wedge\delta_{s,\overrightarrow{q_j,q_i}}\wedge\delta_{s,\overrightarrow{q_j,q_{k+2}}}\right) \tag{28}$$

- Each positive (resp. negative) word terminates in at least one accepting (resp. rejecting) possible final state:

$$\bigwedge_{w\in S^+}\bigvee_{i\in K}\left(\rho_{w,\overrightarrow{q_1,q_i}}\wedge a_i^\star\right) \tag{29}$$

$$\bigwedge_{w\in S^-}\bigvee_{i\in K}\left(\rho_{w,\overrightarrow{q_1,q_i}}\wedge r_i^\star\right) \tag{30}$$

- a state cannot be both accepting and rejecting possible final state:

$$\bigwedge_{i\in K}\neg(a_i^\star\wedge r_i^\star) \tag{31}$$

3 FROM 3NFA TO WEIGHTED-FREQUENCY NFA AND PROBABILISTIC NFA

Using a sample of positive and negative words, we are able to generate a k_3NFA. However, we cannot directly obtain probabilistic automata to decide the probability for a word to be part or not of the language represented by the sample. However, we can use the sample and the generated k_3NFA to build a weighted-frequency automaton: weighted-frequencies will be determined with respect to the sample words. We can then create, from this last automaton, a probabilistic automaton to classify words.

3.1 Weighted-Frequency Automata

We now define what we call a weighted-frequency automaton. In a frequency automata, the integer n attached to a transition $\delta(q,a,q')$ means that this transition was used n times (see (de la Higuera, 2010)). Here, we want to count differently positive (resp. negative) words terminating in an accepting (resp. rejecting) final state from positive (resp. negative) words terminating in whatever state. We thus need to weigh these cases with different real numbers. Thus, we obtain automata that reflect weighted frequencies and are still based on 3-sort automata.

Definition 2 (3_NWFFA)**.** *A 3-sort nondeterministic weighted-frequency finite automaton (3_NWFFA) is a 10-tuple* $\mathcal{A} = (Q,\Sigma,I,F_+,F_-,\delta,\Omega_{(f,+)},\Omega_{(f,-)},\Omega_{(\delta,+)},\Omega_{(\delta,-)})$ *with:*

- $Q=\{q_1,\ldots,q_k\}$ *– a finite set of states,*
- Σ *– a finite alphabet,*
- $I=\{q_1\}$ *– the set of initial states,*
- F_+ *– the set of accepting final states,*
- F_- *– the set of rejecting final states,*
- $\delta : Q\times\Sigma\rightarrow 2^Q$ *– the transition function,*
- $\Omega_{(f,+)}$ *– a function* $Q\rightarrow\mathbb{N}$*, i.e.,* $\Omega_{(f,+)}(q) =$

$$\begin{cases} \omega_{(f,+,+)}\cdot\phi_{(f,+,+)}(q) & \text{if} \quad q\in F_+ \\ \omega_{(f,+,?)}\cdot\phi_{(f,+,?)}(q) & \text{if} \quad q\in Q\setminus(F_+\cup F_-) \\ 0 & \text{otherwise} \end{cases}$$

  where:
  - $\omega_{(f,+,+)}$ *and* $\omega_{(f,+,?)}$ *two weights associated respectively to positive words terminating in accepting states and to positive words terminating in whatever states,*
  - *two counting functions* $\phi_{(f,+,+)}(q)$ *and* $\phi_{(f,+,?)}(q)$ *for counting the number of times (i.e., the numbers of physical paths for all positive words) a positive word terminates in accepting state q and respectively in whatever state q. These two functions are detailed later.*
- $\Omega_{(f,-)}$ *– a function* $Q\rightarrow\mathbb{N}$*,* $\Omega_{(f,-)}(q)$ *is defined as above but for negative words and rejecting states (i.e., "+" is replaced by "−", and* F_+ *by* F_-*).*
- $\Omega_{(\delta,+)}$ *– a function* $Q\times\Sigma\times Q\rightarrow\mathbb{N}$*, i.e.,* $\Omega_{(\delta,+)}(q,s,q') =$

  $\omega_{(\delta,+,+)}\cdot\phi_{(\delta,+,+)}(q,s,q')+\omega_{(\delta,+,?)}\cdot\phi_{(\delta,+,?)}(q,s,q')$

  where:
  - $\omega_{(\delta,+,+)}$ *and* $\omega_{(\delta,+,?)}$ *are two weights associated respectively with positive words terminating in accepting states and positive words terminating in whatever states,*
  - *two counting functions* $\phi_{(\delta,+,+)}(q,s,q')$ *and* $\phi_{(\delta,+,?)}(q,s,q')$ *for counting the number of times a positive word uses given transition within a physical path that terminates in an accepting state and respectively in whatever state. These two functions are detailed later.*

- $\Omega_{(\delta,-)}$ – *a function* $Q \times \Sigma \times Q \to \mathbb{N}$, *i.e.,* $\Omega_{(\delta,-)}(q,s,q') = \omega_{(\delta,-,-)} \cdot \phi_{(\delta,-,-)}(q,s,q') + \omega_{(\delta,-,?)} \cdot \phi_{(\delta,-,?)}(q,s,q')$ *defined as above but for negative words.*

Remember that these automata are non-deterministic, and thus, there can be several paths for a word, terminating in different states. Thus, for a given word, we are interested in all terminating paths independently from the sort of the terminating state.

3.2 From 3-sort NFA to Weighted-Frequency Automata

We need a "physical" view of transitions and paths of the k_3NFA we have built. Consider the transition function $\delta : Q \times \Sigma \to 2^Q$ from a k_3NFA $\mathcal{A}$. We rename by $\Delta_{s,\vec{i,j}}$ the value $\delta(q_i,s,q_j)$. Note that if $\delta_{s,\overrightarrow{q_i,q_j}}$ is true, $\Delta_{s,\vec{i,j}}$ exists, otherwise $\Delta_{s,\vec{i,j}}$ does not exist.

We also define $\pi_{s_1\ldots s_n,\overline{i_1,\ldots,i_{n+1}}}$ as the sequence of physical transitions $\Delta_{s_1,\vec{i_1,i_2}},\ldots,\Delta_{s_n,\overrightarrow{i_n,i_{n+1}}}$. For a given word $w = s_1\ldots s_n$,

$$\Pi_{w,\overline{i_1,i_{n+1}}} = \{\pi_{s_1\ldots s_n,\overline{i_1,\ldots,i_{n+1}}} \mid i_1,\ldots,i_{n+1} \in Q^{n+1}\}$$

is the set of all sequences for w in $\mathcal{A}$.

Consider a sequence $\pi_{s_1\ldots s_n,\overline{i_1,\ldots,i_{n+1}}} = \Delta_{s_1,\vec{i_1,i_2}},\ldots,\Delta_{s_n,\overrightarrow{i_n,i_{n+1}}}$. Then, $occ(\pi_{s_1\ldots s_n,\overline{i_1,\ldots,i_{n+1}}})(\Delta_{s,\vec{i,j}})$ is the number of occurrences of $\Delta_{s,\vec{i,j}}$ in the sequence $\pi_{s_1\ldots..s_n,\overline{i_1,\ldots,i_{n+1}}}$ defined recursively as follows:

- $occ(\Lambda)(\Delta_{s,\vec{i,j}}) = 0$
- $occ(\Delta_{s_1,\vec{k_1,l_1}},\Delta_{s_2,\vec{k_2,l_2}},\ldots,\Delta_{s_3,\vec{k_3,l_3}})(\Delta_{s,\vec{i,j}}) =$

$$\begin{cases} 1 + occ(\Delta_{s_2,\vec{k_2,l_2}},\ldots,\Delta_{s_3,\vec{k_3,l_3}})(\Delta_{s,\vec{i,j}}) \\ \qquad \text{if } (s,i,j) = (s_1,k_1,l_1), \\ occ(\Delta_{s_2,\vec{k_2,l_2}},\ldots,\Delta_{s_3,\vec{k_3,l_3}})(\Delta_{s,\vec{i,j}}) \\ \qquad \text{otherwise} \end{cases}$$

.

We now propose an implementation of the counting functions for a weighted-frequency automaton:

- $\phi_{(f,+,+)}$: if $q \in F_+$, $\phi_{(f,+,+)}(q) = |\bigcup_{w\in S^+} \Pi_{w,\overline{1,q}}|$, 0 otherwise
- $\phi_{(f,+,?)}$: if $q \in Q \setminus (F_+ \cup F_-)$, $\phi_{(f,+,?)}(q) = |\bigcup_{w\in S^+} \Pi_{w,\overline{1,q}}|$, 0 otherwise
- $\phi_{(f,-,-)}$ and $\phi_{(f,-,?)}$ can be defined similarly
- $\phi_{(\delta,+,+)}$: $\phi_{(\delta,+,+)}(q,s,q') =$

$$\sum_{w\in S^+} \sum_{q_A\in F_+} \sum_{p\in\Pi_{w,\overline{q_1,q_A}}} occ(p)(\Delta_{s,\overrightarrow{q,q'}})$$

- $\phi_{(\delta,+,?)}$, $\phi_{(\delta,-,-)}$, and $\phi_{(\delta,-,?)}$ are defined similarly.

We can imagine other counting functions, for example not considering all possible paths for a word, but only one path, or only a given number of paths.

The different weights enable us to consider only positive words for example ($\omega_{(F,-,*)} = 0$ for $F \in \{f,\delta\}$ and $* = \{-,?\}$), or considering for example only positive words terminating in a positive state ($\omega_{(f,+,?)} = 0$, and $\omega_{(\delta,+,?)} = 0$).

3.3 Probabilistic Automata

We can now define the probabilistic automata we are interested in: 3-sort automata with probabilities for transitions and probabilities for states to be final accepting and rejecting.

Definition 3 (3_NPFA)**.** *A 3-sort non-deterministic probabilistic finite automaton is an 8-tuple* $\mathcal{A} = (Q,\Sigma,I,\delta,\Gamma_{(f,+)},\Gamma_{(f,-)},\Gamma_{(\delta,+)},\Gamma_{(\delta,-)})$ *with:*

- $Q = \{q_1,\ldots,q_k\}$ *– a finite set of states,*
- Σ *– a finite alphabet,*
- $I = \{q_1\}$ *– the set of initial states,*
- $\delta : Q \times \Sigma \to 2^Q$ *– the transition function,*
- $\Gamma_{(f,+)}$ *– a function* $Q \to [0,1]$*, i.e.,* $\Gamma_{(f,+)}(q)$ *is the probability of state q to be accepting final,*
- $\Gamma_{(f,-)}$ *– a function* $Q \to [0,1]$*, i.e.,* $\Gamma_{(f,-)}(q)$ *is the probability of state q to be rejecting final,*
- $\Gamma_{(\delta,+)}$ – *a function* $Q \times \Sigma \times Q \to [0,1]$*, i.e.,* $\Gamma_{(\delta,+)}(q,s,q')$ *is the probability for a positive word to pass by the transition* $\delta(q,s,q')$*,*
- $\Gamma_{(\delta,-)}$ *– similar to* $\Gamma_{(\delta,+)}$ *for negative words.*

A 3-sort non-deterministic probabilistic automaton $\mathcal{A} = (Q,\Sigma,I,\delta,\Gamma_{(f,+)},\Gamma_{(f,-)},\Gamma_{(\delta,+)},\Gamma_{(\delta,-)})$ *must respect the following constraint:*

$$\forall q \in Q, \begin{cases} \sum_{q'\in Q,s\in\Sigma} \left(\Gamma_{(\delta,+)}(q,s,q') + \Gamma_{(f,+)}(q)\right) = 1 \\ \sum_{q'\in Q,s\in\Sigma} \left(\Gamma_{(\delta,-)}(q,s,q') + \Gamma_{(f,-)}(q)\right) = 1 \end{cases}$$

Remember that we consider only one initial state which is q_1. In case one wants to consider several initial states, some probabilities of being initial positive and initial negative can be added.

3.4 From Weighted-Frequency to Probabilistic Automata

We now present the transformation of a 3_NWFFA into a 3_NPFA: weighted-frequencies are converted into probabilities.

Consider a 3-sort non-deterministic weighted-frequency finite automaton $\mathcal{A} = (Q,\Sigma,I,F_+,F_-,\delta,\Omega_{(f,+)},\Omega_{(f,-)},\Omega_{(\delta,+)},\Omega_{(\delta,-)})$. Then, from $\mathcal{A}$, we can derive a 3-sort non-deterministic probabilistic finite automaton $\mathcal{A}' = (Q',\Sigma',I',\delta',\Gamma_{(f,+)},\Gamma_{(f,-)},\Gamma_{(\delta,+)},\Gamma_{(\delta,-)})$ such that:

- the states, alphabet, transitions, and initial state remain unchanged: $Q = Q'$, $\Sigma = \Sigma'$, $I = I'$, and $\delta = \delta'$
- the probability for q to be an accepting final state is the weighted frequency of words of S^+ terminating in q, divided by the sum of the weighted frequencies of the positive words from the sample outgoing from q plus the weighted-frequency of positive words ending in q:

$$\forall q \in Q,\ \Gamma_{(f,+)}(q) = \Omega_{(f,+)}(q) / \left(\Omega_{(f,+)}(q) + \sum_{s \in \Sigma, q' \in Q} \Omega_{(\delta,+)}(q,s,q') \right)$$

- the probabilities $\Gamma_{(f,-)}$ are computed similarly for negative words replacing "+" by "−".
- the probability for a positive word to follow transition $\delta(q,s,q')$ is computed similarly as the probability of ending in q:

$$\forall q,q' \in Q, \forall s \in \Sigma, \Gamma_{(\delta,+)}(q,s,q') = \Omega_{(\delta,+)}(q,s,q') / \left(\Omega_{(f,+)}(q) + \sum_{s' \in \Sigma, q'' \in Q} \Omega_{(\delta,+)}(q,s',q'') \right)$$

- the computation is similar for negative words replacing "+" by "−".

These probabilities respect the properties of 3-sort non-deterministic frequency finite automata.

3.5 Classifying Words

Given the two sets of independent weights (for states and transitions) and the non-deterministic nature of the NFA, implying possibly multiple paths for a word, we consider four classifiers:

- $\mathcal{C}_{MM}$ – computes the positive and negative scores for a word by *multiplying* the probabilities of the transitions and the probability of the last state on each path, selecting as the final score the *maximum* of all paths.
- $\mathcal{C}_{MA}$ – computes the positive and negative scores for a word by *multiplying* the probabilities of the transitions and the probability of the last state on each path, selecting as the final score the *average* of all paths.
- $\mathcal{C}_{SM}$ – computes the positive and negative scores for a word by *summing up* the probabilities of the transitions and the probability of the last state on each path, selecting as the final score the *maximum* of all paths. Summed up probabilities for each path for word w are divided by $|w|+1$, to scale them to the range $[0,1]$.
- $\mathcal{C}_{SA}$ – computes the positive and negative scores for a word by *summing up* the probabilities of the transitions and the probability of the last state on each path, selecting as the final score the *average* of all paths. Summed up probabilities for each path for word w are divided by $|w|+1$, to scale them to the range $[0,1]$.

The final classifier decision, i.e., acceptance or rejection of a word, is based on the comparison of positive and negative scores—the greater score wins.

To illustrate the operation of the classifiers let us consider an example. Assume that we have a word $w = abb$ for which there are two paths in some NFA. For simplicity, assume that all weights $\omega_{(F,\bullet,*)} = 1$, with $F = \{f,\delta\}$, $\bullet = \{+,-\}$, and $* = \{+,-,?\}$. Let us also assume that the transition and last state probabilities for the first path are: $(0.2, 0.5, 0.35, 0.6)$ for the acceptance scenario, and $(0.6, 0.5, 0.65, 0.4)$ for the rejection scenario. For the second path assume probabilities $(0.2, 0.15, 0.55, 0.9)$ and $(0.6, 0.5, 0.5, 0.75)$. Then the scores for the respective classifiers are as follows:

- for $\mathcal{C}_{MM}$ the positive score is 0.02, and the negative is 0.11,
- for $\mathcal{C}_{MA}$ the positive score is 0.02, and the negative is 0.10,
- for $\mathcal{C}_{SM}$ the positive score is 0.45, and the negative is 0.59,
- for $\mathcal{C}_{SA}$ the positive score is 0.43, and the negative is 0.56.

It is clear that each classifier indicates that the word should be rejected as the negative scores are always greater than the positive ones. Note also that the $\mathcal{C}_{SM}$ and $\mathcal{C}_{SA}$ produce larger margin between the scores than the $\mathcal{C}_{MM}$ and $\mathcal{C}_{MA}$ (0.13–0.14 vs. 0.08–0.09).

4 EXPERIMENTATION

4.1 Experiment I

To evaluate the proposed probabilistic automata and classifiers, we have created a benchmark set based on the peptides stored in WaltzDB database (Beerten et al., 2015; Louros et al., 2019). The benchmark set was composed of several samples containing amyloid (positive) and non-amyloid (negative) peptides, each having a length of 6 characters. The samples were created based on pep-

tide subsets available on the WaltzDB website (`http://waltzdb.switchlab.org/sequences`).

Based on each sample, the training and test samples were created, with the training sample consisting of 10%, 30%, and 50% of the first peptide sequences in the given subset. The training sample was used to infer the probabilistic NFA, which acted then as a classifier for the test sample, comprising the whole subset without the elements included in the training sample. Since some of the subsets contained very few positive/negative sequences, for the final evaluation we selected only five of them, i.e., Amylhex (AH), Apoai mutant set (AMS), Literature (Lit), Newscores (NS), and Tau mutant set (TMS). Table 1 summarizes the characteristics of the data set. Note that all samples are quite imbalanced and they differ both in the total number of words and the size of the alphabet.

Table 1: Characteristics of the benchmark set.

		Train 10%		Train 30%		Train 50%		Whole subset	
Subset	$\|\Sigma\|$	$\|S^+\|$	$\|S^-\|$	$\|S^+\|$	$\|S^-\|$	$\|S^+\|$	$\|S^-\|$	$\|S^+\|$	$\|S^-\|$
AH	19	7	12	23	36	39	60	79	121
AMS	20	7	3	23	10	39	18	79	36
Lit	20	20	6	61	19	102	33	204	66
NS	18	3	1	9	4	16	7	32	15
TMS	19	9	2	27	6	46	11	92	22

The inference models were implemented in Python using PySAT library and the Glucose SAT solver with default options. The experiments were carried out on a computing cluster with Intel-E5-2695 CPUs, and a fixed limit of 10 GB of memory. Running times were limited to 15 minutes, including model generation and solving time. The classification was conducted using a Java application running on a single machine with Intel Core i7-7560U 2.40GHz processor and 8 GB of RAM.

Since weights tuning lies outside of the scope of the current paper, we decided to conduct the experiments by setting respective weights to 0s or 1s only, analyzing all possible combinations of 0s and 1s for the eight weights defined before. Thus, for each training sample we analyzed 256 different weight assignments, which along with 115 inferred NFAs[3] and 4 classifiers gave us a total of 117 760 classifications. The whole process took around 186 minutes, with the $(k+2)$_3NFA models taking on average 2.9 times longer to perform the classification than the k_3NFA ones. This difference may be attributed to the larger size of the former NFAs, which makes the path building process more time-consuming.

The classification results were evaluated based on accuracy and F1-score given by Eqs. (32) and (33):

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{32}$$

$$F1 = \frac{2 \cdot TP}{2 \cdot TP+FP+FN} \tag{33}$$

where TP denotes true positives (amyloid peptides classified as amyloid), TN denotes true negatives (non-amyloid peptides classified as such), FP denotes false positives (non-amyloid peptides classified as amyloid ones), and FN denotes false negatives (amyloid peptide classified as non-amyloid).

Table 2 shows the best accuracy values and their corresponding F1-scores obtained for the test sets over all analyzed weight combinations and all classifiers. The metrics were obtained by NFAs inferred using 8 different models. Boldfaced values denote the best column-wise values. The entries with an asterisk denote the cases in which the best F1-score did not correspond to the best accuracy.

Table 2: Best accuracy and corresponding F1-score metrics obtained by the NFAs for the analyzed benchmark sets.

	Accuracy					F1-score				
Model	AH	AMS	Lit	NS	TMS	AH	AMS	Lit	NS	TMS
P_k	0.63	0.59	0.76	0.63	0.72	0.63*	0.66*	**0.86**	0.73	**0.82**
$P_{(k+2)}$	**0.69**	**0.68**	0.76	0.67	**0.73**	0.64*	**0.79**	**0.86**	0.76	**0.82**
$P_k^\star$	0.68	0.62	0.76	**0.71**	0.72	**0.66**	0.72	**0.86**	**0.77**	**0.82**
$P_{(k+2)}^\star$	0.64	0.67	0.73	0.50	**0.73**	0.62*	0.77	0.82	0.51	**0.82**
S_k	0.61	0.55	0.76	0.62	0.72	0.48*	0.56*	**0.86**	0.70	**0.82**
$S_{(k+2)}$	0.65	0.64	**0.77**	0.50	**0.73**	0.60*	0.77	**0.86**	0.48	**0.82**
$S_k^\star$	0.61	0.62	0.66	0.50	0.72	0.58*	0.76	0.79	0.51	**0.82**
$S_{(k+2)}^\star$	0.63	0.66	0.74	0.47	**0.73**	0.42*	0.77	0.85	0.44	**0.82**

Based on the accuracy values we can state that the prefix-based models perform best among all eight models, regardless of the benchmark set. It is also clear that NS data set turned out to be the hardest one, since some of the models achieved accuracy smaller than 0.5, which is the probability of success with a random decision in binary classification. The analysis of F1-score, being the harmonic mean between precision and recall, confirms the strong position of prefix-based models, with a small advantage given to the $P_k^\star$ model. Overall, the achieved metrics are not very satisfactory, which may be attributed to, e.g.:

- the way the training samples were constructed – it is not infrequent that the training sample does not cover the whole alphabet, which results in the rejection of words from the test set using the symbols outside of the training sample's alphabet,
- the lack of some language behind the subsets of peptides – there is no guarantee that the peptides from a certain subset share some common features reflected in the sequences,
- limited parameter tuning – so far we analyzed only the extreme values for the weights, more

[3] In five cases for the NS subset, we failed to infer an NFA for the Train 50% training set. The models that failed were the $P_{k+2}^\star$ and all suffix-based models.

advanced parameter tuning may be required to achieve better results.

Interestingly, the best results were typically achieved with 30% training sets. The NS data set, whenever the NFA for it could be inferred, required the 50% training set to achieve its peak performance. There were also rare cases in which the smallest training data set was sufficient (e.g., for the P_k model with AMS data set).

Table 3 shows the best accuracy values and corresponding F1-scores for different classifiers over all analyzed weight combinations and all subsets. The metrics pertain to NFAs inferred using eight different models. The meaning of boldfaced entries and entries with an asterisk is the same as for Table 2.

Table 3: Best accuracy and corresponding F1-score metrics obtained by the NFAs for the analyzed classifiers.

Model	Accuracy				F1-score			
	C_{MM}	C_{MA}	C_{SM}	C_{SA}	C_{MM}	C_{MA}	C_{SM}	C_{SA}
P_k	**0.70**	**0.70**	0.76	0.76	**0.80**	**0.81**	**0.86**	**0.86**
$P_{(k+2)}$	**0.70**	0.69	0.76	0.76	**0.80**	0.79	**0.86**	**0.86**
$P_k^\star$	0.56	0.56	0.76	0.76	0.45*	0.45*	**0.86**	**0.86**
$P_{(k+2)}^\star$	0.59	0.59	0.73	0.73	0.70	0.70	0.82*	0.82*
S_k	0.53	0.53	0.76	0.76	0.61*	0.69	**0.86**	**0.86**
$S_{(k+2)}$	0.64	0.64	**0.77**	**0.77**	0.73	0.73	**0.86**	**0.86**
$S_k^\star$	0.53	0.53	0.72	0.72	0.69	0.69	0.82	0.82
$S_{(k+2)}^\star$	0.68	0.68	0.74	0.74	0.79	0.79	0.85	0.85

The analysis shows that this time P_k is clearly the best model. In terms of classifiers, we do not observe many differences between the pairs of classifiers based on multiplication (C_{MM}, C_{MA}) and summation (C_{SM}, C_{SA}). However, the differences between multiplication- and summation-based classifiers for the given model are statistically significant (based on ANOVA test with $\alpha = 0.05$ and post hoc Tukey HSD) in terms of both accuracy and F1-score.

Figure 1 shows the best accuracy values and corresponding F1-scores obtained by all classifiers over all analyzed weight combinations and NFAs inferred by all models. We can confirm that the differences between the classifiers are statistically significant at $\alpha = 0.05$, with the C_{SA} and C_{SM} classifiers consistently achieving better results than the other two. We can also note that Lit data set was the most favorable in terms of satisfactory metric values.

4.2 Experiment II

To evaluate the proposed solutions even further, we created the second benchmark composed of two data sets. The data sets were built based on regular expressions (regexp)[4]—we defined two languages described

[4]The regular expressions were: (0|11)(001|000|10)*0 and [0-9][0-4][5-9](024|135|(98|87))*(0|6).

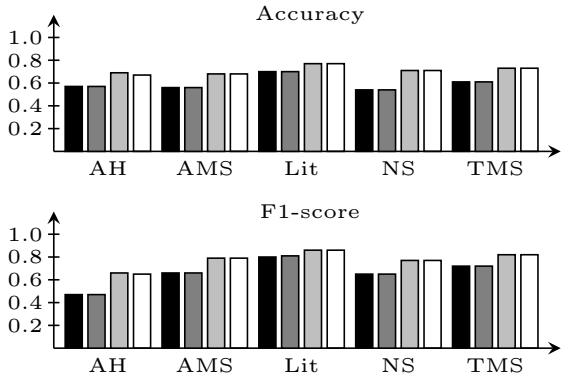

Figure 1: Best accuracy and F1-score metrics obtained by all NFAs for the analyzed benchmark sets and classifiers C_{MM} (black), C_{MA} (gray), C_{SM} (light gray), and C_{SA}.

by different regular expressions from which we sampled the words of 1 to 15 characters. These words represented the sets S^+. The sets S^- contained words constructed by randomly shuffling the positive examples and ensuring they do not match the regexp. Similarly to the first experiment, we created the training data sets used for NFA inference and the test sets for evaluation. The sizes of complete samples were equal to 200 words split equally between S^+ and S^-. The experimental setup, i.e., computing machines, metrics, and weight settings were kept as before.

In Table 4 we show the results obtained for the various models across all classifiers for the two regexp-based data sets. Comparing the results to the ones presented in Tab. 2, we note a significant improvement in the achieved metrics. We also observe that for RegExp1 data set, except for the S_k model, all models achieve perfect scores. Finally, we note that for RegExp2 all $(k+2)$-based models, except $PM(k+2)$, improve over their k-based counterparts, while for RegExp1 it only applies to $S^\star$ model, since the others achieved perfect scores. Detailed analysis have shown that in most cases, the best accuracy and F1-score were obtained with the 50% training set, but for the $P_{(k+2)}$, S_k, and $S^\star_{(k+2)}$ models, 10% was enough.

Table 4: Best accuracy and corresponding F1-score metrics obtained by the NFAs for the analyzed benchmark sets.

	Data set	P_k	$P_{(k+2)}$	$P_k^\star$	$P_{(k+2)}^\star$	S_k	$S_{(k+2)}$	$S_k^\star$	$S_{(k+2)}^\star$
Acc	RegExp1	**1.00**	**1.00**	**1.00**	**1.00**	0.91	**1.00**	**1.00**	**1.00**
	RegExp2	**0.93**	0.88	0.88	**0.93**	0.77	0.92	0.85	0.87
F1-score	RegExp1	**1.00**	**1.00**	**1.00**	**1.00**	0.90	**1.00**	**1.00**	**1.00**
	RegExp2	0.93	0.86*	0.87	**0.95**	0.71*	0.92	0.81	0.87

In Table 5, we show the analysis of best accuracy and corresponding F1-score for the models vs. classifiers comparison. It can be observed that the results improved significantly as compared to the ones presented in Tab. 3. We can also note that with this

benchmark, only the S_k model failed to achieve perfect scores. Clearly, there are no significant differences between classifiers as they performed equally well regardless of the model.

Table 5: Best accuracy and corresponding F1-score metrics obtained by the NFAs for the analyzed classifiers.

Model	Accuracy				F1-score			
	C_{MM}	C_{MA}	C_{SM}	C_{SA}	C_{MM}	C_{MA}	C_{SM}	C_{SA}
P_k	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$P_{(k+2)}$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$P_k^\star$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$P_{(k+2)}^\star$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
S_k	0.91	0.91	0.91	0.91	0.90	0.90	0.90	0.90
$S_{(k+2)}$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$S_k^\star$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$S_{(k+2)}^\star$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

5 CONCLUSIONS

In this paper, we have proposed a method to transform an NFA with three types of states (accepting, rejecting and non-conclusive) to a weighted frequency automaton, which could be further transformed into a probabilistic NFA. The developed transformation process is generic since it allows to control the relative importance of the different types of states and/or transitions by customizable weights.

We have evaluated the proposed probabilistic automata on the classification task performed over two distinct benchmarks. The first one, based on real-life samples of peptide sequences proved to be quite challenging, yielding relatively low quality metrics. The second benchmark, based on a random sampling of a language described by a regular expression enabled us to show the power of probabilistic NFA, producing accuracy scores of 0.81–1.00 with F1-score ranging between 0.69 up to 1.00. The second benchmark allowed us to prove that given a representative sample of an underlying language, the probabilistic NFA can achieve very good classification quality, even without sophisticated parameter tuning.

In the future, we plan to apply some heuristics to tune the weights so that the classifiers perform even better, especially for real-life benchmarks. Given the generic nature of the proposed weighted-frequency automata we also plan to consider using a parallel ensemble of classifiers, differing not only in terms of weights, but also in how probabilities are combined.

REFERENCES

Beerten, J., van Durme, J. J. J., Gallardo, R., Capriotti, E., Serpell, L. C., Rousseau, F., and Schymkowitz, J. (2015). WALTZ-DB: a benchmark database of amyloidogenic hexapeptides. Bioinform., 31(10):1698–1700.

de la Higuera, C. (2010). Grammatical Inference: Learning Automata and Grammars. Cambridge University Press.

Denis, F., Lemay, A., and Terlutte, A. (2004). Learning regular languages using rfsas. Theor. Comput. Sci., 313(2):267–294.

Jastrząb, T. (2017). Two parallelization schemes for the induction of nondeterministic finite automata on PCs. In Proc. of PPAM 2017, volume 10777 of LNCS, pages 279–289. Springer.

Jastrząb, T., Lardeux, F., and Monfroy, É. (2022). Taking advantage of a very simple property to efficiently infer NFAs. In 34th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2022, pages 1355–1361. IEEE.

Jastrząb, T., Lardeux, F., and Monfroy, É. (2023). Inference of over-constrained NFA of size $k+1$ to efficiently and systematically derive NFA of size k for grammar learning. In Proceedings of the International Conference on Computational Science – ICCS 2023, Part I, volume 14073 of LNCS, pages 134–147. Springer.

Lardeux, F. and Monfroy, É. (2021). Optimized models and symmetry breaking for the NFA inference problem. In 33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, pages 396–403. IEEE.

Lecoutre, C. and Szczepanski, N. (2020). PYCSP3: modeling combinatorial constrained problems in python. CoRR, abs/2009.00326.

Louros, N., Konstantoulea, K., De Vleeschouwer, M., Ramakers, M., Schymkowitz, J., and Rousseau, F. (2019). WALTZ-DB 2.0: an updated database containing structural information of experimentally determined amyloid-forming peptides. Nucleic Acids Research, 48(D1):D389–D393.

Rossi, F., van Beek, P., and Walsh, T., editors (2006). Handbook of Constraint Programming, volume 2 of Foundations of Artificial Intelligence. Elsevier.

Stützle, T. and Ruiz, R. (2018). Iterated Local Search, pages 579–605. Springer International Publishing, Cham.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hill-climbing. Proc. of the Fourth Annual Conference of the Cognitive Science Society, pages 105–108.

Vázquez de Parga, M., García, P., and Ruiz, J. (2006). A family of algorithms for non deterministic regular languages inference. In Proc. of CIAA 2006, volume 4094 of LNCS, pages 265–274. Springer.

Wieczorek, W. (2017). Grammatical Inference – Algorithms, Routines and Applications, volume 673 of Studies in Computational Intelligence. Springer.