Learning Automata and Grammars

Peter Černo

Learning Automata and Grammars

- **The problem of learning** or inferring **automata and grammars** has been studied for decades and has connections to many disciplines:
  - Bio-informatics.
  - Computational linguistics.
  - Pattern recognition.

Learning Automata and Grammars

* In this presentation we:
  * Introduce the **formal language theory (FLT)**.
  * Emphasize the **importance of learnability**.
  * Explain the **identification in the limit.**
  * Give as an example the **algorithm LARS**.

Formal Language Theory

- The central notion in FLT is a **(formal) language** which is a finite or infinite set of words.
- **Word** is a finite sequence consisting of zero or more **letters**. The same letter may occur several times.
- The sequence of zero letters = the **empty word** λ.
- We restrict ourselves to some specific **alphabet**, which is a finite nonempty set of letters.
- The **set of all words** from Σ is denoted as Σ^*.

Formal Language Theory

* How to define a language?
  * **Acceptors**: usually **automata** – they are given an input word and after some processing they either **accept** or **reject** this input word.
  * **Generators**: usually **grammars** – they **generate** the language using some finite set of rules.
* We want to **learn automata** / **grammars** under suitable learning regime.

Induction / Inference

* **Grammar induction**: finding a grammar (or automaton) that can **explain the data.**
* **Grammatical inference**: relies on the fact that there is a (true) **target grammar** (or automaton), and that the quality of the learning process has to be measured relatively to this target.

Importance of Learnability

- **Alexander Clark** in *[Clark, A.: Three learnable models for the description of language]* emphasized the **importance of learnability**.
- He proposed that one way to build learnable representations is by making them **objective** or **empiricist**: the structure of the representation should be based on the structure of the language.

Importance of Learnability

- In defining these representation classes the author followed a simple **slogan**: "**Put learnability first!**"
- In the conclusive remarks the author suggested that the representations, which are both **efficiently learnable** and capable of representing **mildly context-sensitive languages** seem to be good candidates for models of **human linguistic competence**.

General Setting

* In a typical grammatical inference scenario we are concerned with learning language representations based on some **source of information**:
  * Text.
  * Examples and counter-examples.
  * Etc.
* We assume a perfect source of information.

General Setting

* Let us fix the alphabet Σ.
* Let $\mathcal{L}$ be a **language class**.
* Let $\mathcal{R}$ be a **class of representations** for $\mathcal{L}$.
* Let $L: \mathcal{R} \rightarrow \mathcal{L}$ be the **naming function**, i.e. $L(R)$ is the language denoted, accepted, recognized or represented by the object $R \in \mathcal{R}$.

General Setting

- There are two important problems:
- **Membership problem**: given $w \in \Sigma^*$ and $R \in \mathcal{R}$, is the query $w \in L(R)$ decidable?
- **Equivalence problem:** given $R_1, R_2 \in \mathcal{R}$, is the query $L(R_1) = L(R_2)$ decidable?

Identification in the Limit

- A **presentation** Φ is an enumeration of elements, which represents a **source of information** about some specific language $L \in \mathcal{L}$.
  - For instance, the enumeration of all positive and negative samples of L (in some order).
- A **learning algorithm** A is a program that takes the first n elements of a presentation (denoted as Φ_n) and returns some object $R \in \mathcal{R}$.

Identification in the Limit

- We say that $\mathcal{R}$ is **identifiable in the limit** if there exists a learning **algorithm** A such that for any **target object** $R \in \mathcal{R}$ and any **presentation** Φ of $L(R)$ there exists a rank m such that for all $n \geq m$ $\boldsymbol{A}(\Phi_n)$ does not change and $L(\boldsymbol{A}(\Phi_n)) = L(R)$.
- The above definition does not force us to learn the target object, but only to **learn an object equivalent to the target**.

Identification in the Limit

* However, there are some **complexity issues** with the identification in limit:
  * It neither tells us **how we know** when we have found what we are looking for nor **how long** it is going to take.
* We illustrate this methodology on the so-called **delimited string-rewriting systems**.
* The learning algorithm is called **LARS**.

String Rewriting Systems

* **String rewriting systems** are usually specified by:
  * Some rewriting mechanism,
  * Some base of simple (accepted) words.
* Let us introduce **two special symbols** that do not belong to our alphabet Σ:
  * $¢$ left sentinel,
  * $\$$ right sentinel.

String Rewriting Systems

- **Term** is a string from $T(\Sigma) = \{\lambda, ¢\}.\Sigma^*.\{\lambda, \$\}$.
- **Term** can be of one of the following **types**:
  - Type 1: $w \in \Sigma^*$ (substring)
  - Type 2: $w \in ¢.\Sigma^*$ (prefix)
  - Type 3: $w \in \Sigma^*.\$$ (suffix)
  - Type 4: $w \in ¢.\Sigma^*.\$$ (whole string)
- Given a term w, the **root** of w is w without sentinels.

String Rewriting Systems

* We define an **order relation** over $T(\Sigma)$:
* We define $u < v$ if and only if:
  * $root(u) <_{lex\text{-}length} root(v)$ or
  * $root(u) = root(v)$ and $type(u) < type(v)$.
* For instance, for $\Sigma = \{a, b\}$:
  * $ab < ¢ab < ab\$ < ¢ab\$ < ba$

String Rewriting Systems

* A **rewrite rule** is an ordered pair $\rho = (l, r)$, generally written as $\rho = l \vdash r$, where:
  * l is the **left-hand side** of ρ and
  * r is the **right-hand side** of ρ.
* We say that $\rho = l \vdash r$ is a **delimited rewrite rule** if l and r are of the same type.
* **Delimited string-rewriting system (DSRS)** $\mathcal{R}$ is a finite set of delimited rewrite rules.

String Rewriting Systems

* The order extends to rules:
* We define $(l_1, r_1) < (l_2, r_2)$ if and only if:
  * $l_1 < l_2$ or
  * $l_1 = l_2$ and $r_1 < r_2$.
* A system is **deterministic** if not two rules share a common left-hand side.

String Rewriting Systems

* Given a DSRS $\mathcal{R}$ and a string w, there may be several applicable rules.
* Nevertheless, **only one rule is eligible**.
* This is the rule having the **smallest left-hand side**.
* This rule might be eligible in different places. We privilege **the leftmost position**.

String Rewriting Systems

- Given a DSRS $\mathcal{R}$ and strings $w_1, w_2 \in T(\Sigma)$, we say that w_1 **rewrites in one step into** w_2, i.e. $w_1 \vdash_{\mathcal{R}} w_2$ ($w_1 \vdash w_2$), if there exists an eligible rule $(l \vdash r) \in \mathcal{R}$:
  - $w_1 = ulv$, $w_2 = urv$, and
  - u is **shortest** for this rule.
- String w is **reducible** if there exists a string w' such that $w \vdash w'$, and **irreducible** otherwise.

String Rewriting Systems

- We denote by $\vdash^*_{\mathcal{R}}$ the reflexive and transitive closure of $\vdash_{\mathcal{R}}$. We say that w_1 **reduces to** w_2 or that w_2 is **derivable from** w_1 if $w_1 \vdash^*_{\mathcal{R}} w_2$.
- Given a system $\mathcal{R}$ and an irreducible string $e \in \Sigma^*$, we define the **language**:

$$L(\mathcal{R}, e) = \{w \in \Sigma^* \mid ¢w\$ \vdash^*_{\mathcal{R}} ¢e\$\}.$$

String Rewriting Systems

* **Example:**
  * $L(\{ab \vdash \lambda\}, \lambda)$ is the **Dyck language**, i.e.:

    $¢\underline{ab}aabb\$ \vdash ¢a\underline{ab}b\$ \vdash ¢\underline{ab}\$ \vdash ¢\lambda\$$.
  * $L(\{aabb \vdash ab,\ ¢ab\$ \vdash ¢\lambda\$\}, \lambda) = \{a^n b^n \mid n \geq 0\}$, i.e.:

    $¢a\underline{aabb}b\$ \vdash ¢\underline{aabb}\$ \vdash ¢\underline{ab}\$ \vdash ¢\lambda\$$.
  * $L(\{¢ab \vdash ¢\}, \lambda)$ is the regular language $(ab)^*$.
  * It can be shown that **any regular language** can be represented in this way.

String Rewriting Systems

* **Deciding** whether a string w belongs to a language $L(\mathcal{R}, e)$ consists of trying to obtain e from w.
* We will denote by $APPLY(\mathcal{R}, w)$ the string obtained by applying different rules in $\mathcal{R}$ until no more rules can be applied.
* This can be naturally extended to **sets**:

$$APPLY(\mathcal{R}, S) = \{ APPLY(\mathcal{R}, w) \mid w \in S \}.$$

Algorithm LARS

* **Learning Algorithm for Rewriting Systems**.
* Generates the possible rules that can be applied over the positive data S_+.
* Tries using them and keeps them if they do not create inconsistency (using the negative data S_- for that).
* Algorithm calls the function $NEWRULE$, which generates the next possible rule.

Algorithm LARS

* One should choose **useful rules**, i.e. those that can be applied on at least one string from positive data S_+.
* Moreover, a rule should allow to **diminish** the size of the set S_+ (i.e. two different strings rewrite into an identical string).
* The function *CONSISTENT* checks the **consistency** of the system.

Algorithm LARS

- The goal is to be able to learn **any DSRS** with LARS.
- The simplified version proposed here does **identify in the limit** any DSRS.
- Formal study of the algorithm is beyond scope of this presentations.

Algorithm LARS

- **Input:** S_+, S_-.
- **Output:** $\mathcal{R}$.
- $\mathcal{R} := \emptyset; \rho := (\lambda \vdash \lambda);$
- **while** $|S_+| > 1$ **do**
  - $\rho := NEWRULE(S_+, \rho);$
  - **if** $CONSISTENT(S_+, S_-, \mathcal{R} \cup \{\rho\})$ **then**
    - $\mathcal{R} := \mathcal{R} \cup \{\rho\};$
    - $S_+ := APPLY(\mathcal{R}, S_+);\ S_- := APPLY(\mathcal{R}, S_-);$

References

* Alexander Clark (2010): **Three Learnable Models for the Description of Language**.
* Colin de la Higuera (2010): **Grammatical Inference**

  Learning Automata and Grammars