Towards the definition of learning systems with configurable operators and heuristics

Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, María José Ramírez-Quintana

NFMCP 2012

September 24, 2012

Table of contents

Introduction

- Machine learning techniques dealing with structured data:
  - **Distances or kernel methods** can be applied to any kind of data (similarity functions).
  - **Inductive programming (ILP, IFP or IFLP)** are able to tackle any kind of data (first-order logic, term rewriting systems).

Introduction

- The performance of these systems is linked to:
  - a *transformation of the feature space* to a more convenient, flat, representation, which typically leads to incomprehensible patterns in terms of the transformed (hyper-)space
  - use the original problem representation but *rely on specialised systems with embedded operators*
- It is very difficult to have general systems which are able to deal with different kinds of complex data.

Introduction

- We present a **general rule-based learning setting** where **operators can be defined and customised for each kind of problem**.
  - The generalisation operator to use depends on the structure of the data.
  - Adaptive and flexible rethinking of heuristics, with a model-based reinforcement learning approach.

Setting

- **Machine learning operators** are the tools to explore the hypothesis search space.
    - Some operators are usually associated to some heuristic strategies (e.g., generalisation operators and bottom-up strategies).
- Operators can be modified and finetuned for each problem:
    - Different to the use of feature transformations or specific background knowledge.
- This is a challenging proposal not sufficiently explored in machine learning.

Setting

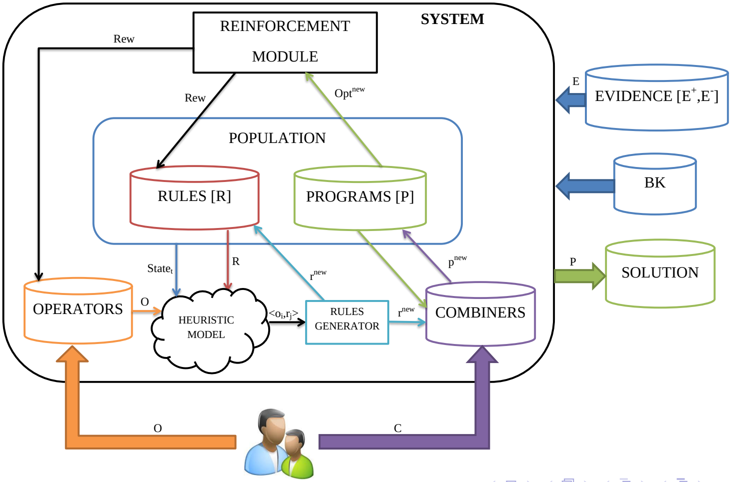
- Operators can be written or modified by the user
  - We need a language for defining operators which can integrate the representation of:
    - **Examples**.
    - **Patterns**.
    - **Operators**.

Setting

- We have chosen a powerful popular programming language, **Erlang**:

  - A functional programming language, with **reflection** and **higher-order primitives**.
  - Operators can be properly linked with the data structures used in the examples and background knowledge, so making the specification of new operators easier.
  - The language also sets the general representation of examples as equations, patterns as rules and models as sets of rules.

General Architecture



Rule and Program Repositories

- Two internal repositories containing **rules** and **programs**.
- Initially, the set of rules R is populated with the positive evidence E^+ and the set of programs P is populated defining unitary programs from the rules of R.
- Both repositories are updated at each step of the algorithm:
  1. The *Rule Generator* builds new rules (r^{new}) and they are added to R.
  2. By applying the combiners, (r^{new}) is mixed with the programs in P generating a new program p^{new}, and it is added to P.

System Operators

- The user can define his/her own set of *operators*, especially suited for the data structures of the problem: **Adaptive system**.
- An *operator* is defined as a function which is applied to a rule in order to generate new rules:
  - Given a rule $f(X) \rightarrow Y$ where the input attribute X is a list, the operator can extract the head of X and return it as the rhs of the new rule.
  - The operator could be defined as:

$$takeHead(f(X) \rightarrow Y)\ [when\ X\ is\ a\ List] \rightarrow (f(X) \rightarrow head(X))$$

System Combiners

- Combiners evolve the population of programs.
  - **Addition**: adds the program that results from joining the new rule r^{new} generated by the *Rule Generator* with the best program (in terms of optimality);
  - **Union**: joins the two best programs (also in terms of optimality) in P.

Reinforcement Module

- A reinforcement learning module guides the *Rule Generator* in each step of the algorithm.
  - S represents the system state as the set composed by R and P.
  - An action A is a tuple $< r_i, o_i >$ where r_i is a rule and o_i is an operator.
- Given an state S, an action A is chosen by the *Heuristic Model* and sent to the *Rule Generator*. This creates new rules (and programs), which causes the system to move to a new state.

Reinforcement Module

- Initially, the *Heuristic Model* does not have enough evidence and the choice is random, but after a few iterations, the model is learnt by using a machine learning technique.
- This model is trained to predict the reward after a given action A, and with it we choose the action which maximises the estimated reward.
- Rewards:
  - From the optimality Opt^{new} of the new program p^{new}, the *Reinforcement Module* calculates a reward Rew.
  - Rew is used to update the optimality of the action $A = < r_i, o_i >$.

Sequence Processing

- Learning a transformation over the words formed by a given alphabet.
  - Alphabet $\Sigma = \{a, t, c, g, u\}$
  - Transformation just replaces t with u.

Instance

$$trans([t, c, g, a, t]) \rightarrow [u, c, g, a, u]$$

Sequence Processing

Background Knowledge

$$f_{at}(a) \rightarrow t;\ \ f_{cg}(c) \rightarrow g; \ldots \tag{1}$$

Operators

$$applyMap(trans(X) \rightarrow Y) \ \Rightarrow\ trans(X) \rightarrow map(V_F, X) \tag{2}$$

$$addBK_f(trans(X) \rightarrow map(V_F, X)) \ \Rightarrow\ trans(X) \rightarrow map(f, X)$$

$$genPat(trans(X) \rightarrow Y) \ \Rightarrow\ trans(V_S) \rightarrow Y \tag{3}$$

Sequence Processing

- There is a simple sequence of operator applications which turns a simple example into a general solution.
- Given the instance $trans([t, c, g, a, t]) \rightarrow [u, c, g, a, u]$:

Solution *Sequence Processing* problem

$$\begin{aligned}
genPat(trans([t, c, g, a, t]) \rightarrow [u, c, g, a, u]) &\quad \Rightarrow \quad trans(V_S) \rightarrow [u, c, g, a, u] \\
applyMap(trans(V_S) \rightarrow [u, c, g, a, u]) &\quad \Rightarrow \quad trans(V_S) \rightarrow map(V_F, V_S) \\
addBK_{f_{tu}}(trans(V_S) \rightarrow map(V_F, V_S)) &\quad \Rightarrow \quad trans(V_S) \rightarrow map(f_{tu}, V_S)
\end{aligned}$$

Bunches of Keys

- Consider the well-known problem of determining whether a key in a bunch of keys can open a door.
- Each instance is given by a bunch of keys, where each key has several features: two-level structure (sets of lists).

Instance

$$opens([[abloy, 3, medium, narrow], [chubb, 6, medium, normal]]) = \top$$

Bunches of Keys

Background Knowledge

$$setExists(Key, Bunch) \tag{4}$$

Operators

$$addBK(opens(X) = \top) \;\Rightarrow\; opens(X) \rightarrow setExists([], X) \tag{5}$$

$$KCond_{cond_i}(opens(X) \rightarrow setExists(C, X)) \;\Rightarrow\; opens(X) \rightarrow setExists([cond_i|C], X) \tag{6}$$

$$genPat(opens(X) = Y) \;\Rightarrow\; opens(V_L) \rightarrow Y \tag{7}$$

Bunches of Keys

- If the prototype and operators are provided, given the original evidence for this example (five $\top$ instances and four $\perp$ instances), it will return the following definition:

Solution *Key of Bunches* problem

$$opens(X) \rightarrow setExists([abloy, medium], X)$$

- *A bunch of keys opens the door if and only if it contains an abloy key of medium length.*

Web categorisation

- *Web* classification problem: web pages are assigned to pre-defined categories mainly according to their content (content mining).
- The evidence of the problem is modelled with 3 parameters described as follows:
    - *Structure*: the graph of links between pages is represented as ordered pairs where each node encodes a linked page
    - *Content*: the content of the web page is represented as a set of attributes with the keywords, the title, etc.
    - *Use*: the information derived from connections to a web server which is encoded by means of a numerical attribute with the daily number of connections.

Web categorisation

- The goal of the problem is to categorise which web pages are about sports.
- A training example may look like this:

Instance

$$sportsWeb(Structure, Content, Connections) \rightarrow \top$$

where:

- $Structure = [\{[olympics, games], [swim]\}, \{[swim], [win]\}, \{[win], [medal]\}]$
- $Content = [\{olympics, 30\}, \{held, 10\}, \{summer, 40\}]$
- $Connections = 20$

Web categorisation

Background Knowledge

$$graphExists(Edge, Graph) \tag{8}$$

$$setExists(Key, List) \tag{9}$$

Operators

$$addBK_{graph}(sportsWeb(S, C, U) \rightarrow \top) \Rightarrow \\ sportsWeb(S, C, U) \rightarrow graphExists(\{[], []\}, S) \tag{10}$$

$$linkl_{cond_i}(sportsWeb(S, C, U) \rightarrow graphExists(\{X, Y\}, S)) \Rightarrow \\ sportsWeb(S, C, U) \rightarrow graphExists(\{[cond_i|X], Y\}, S) \tag{11}$$

Web categorisation

Operators

$$linkr_{cond_i}(sportsWeb(S, C, U) \rightarrow graphExists(\{X, Y\}, S)) \Rightarrow \\ sportsWeb(S, C, U) \rightarrow graphExists(\{X, [cond_i|Y]\}, S) \tag{12}$$

$$genPat_1(sportsWeb(S, C, U) \rightarrow \top) \Rightarrow \\ sportsWeb(V_S, C, U) \rightarrow \top \tag{13}$$

There are also some other operators to generalise the second and third arguments.

Web categorisation

- Our system found the following program which defines the *sportsWeb* function:

Solution *Key of Bunches* problem

$$
\begin{array}{lcl}
\{sportsWeb(V_S, V_C, V_U) & \rightarrow & graphExists(\{[final],[match]\}, V_S). \\
sportsWeb(V_S, V_C, V_U) & \rightarrow & setExists([\{athens]\}, V_C). \\
sportsWeb(V_S, V_C, V_U) & \rightarrow & setExists([\{europe]\}, V_C). \quad \}
\end{array}
$$

- *If the word 'athens' or 'europe' appears in Content, and Structure contains the link $\{[final],[match]\}$ then this is a sport web page.*

Conclusions

- More general systems can be constructed by a flexible operator redefinition and the reuse of heuristics across problems and systems.
- In order to reduce the search space we rely on the definition of customised operators, depending on the data structures and problem at hand.
- We need a language for expressing operators for defining new operators easily.

Conclusions

- The use of different operators precludes the system to use specialised heuristics for each of them.
- We have proposed this as a decision process, where operators are actions to be taken, and this is also seen as a *reinforcement* learning problem.

Future Work

- Transforming the prototype into a learning system, including all the issues in the architecture.
- We need to further develop and refine the heuristics module of the system:
  - Improved description of the state
  - Better reinforcement learning models (which could eliminate many useless explorations of the search space).