Machine Learning of Existential Paths

by Sven Nilsen, 2017

In this paper I formalize the concepts required to create machine learning algorithms for existential paths. These ideas can be used both for exhaustive and non-exhaustive checking. I also describe an idea of using number of new predictions for smarter learning.

Assume we have a function:

$$f : A \to B$$

The existential path is a function that tells whether some member of `B` is returned by `f`:

$$\exists f : B \to bool$$

To find `∃f` without supervision, one needs three generators:

$$a\_rnd : real \wedge probl \to A$$
$$b\_rnd : real \wedge probl \to B$$
$$ex\_rnd : real \wedge probl \to (B \to bool)$$

$$probl := \backslash(x : real) = x >= 0 \wedge x < 1$$

From these generators one can produce three sets:

A_r	randomly generated members of `A`
B_r	randomly generated members of `B`
$(B \to bool)_r$	randomly generated functions

$$candidate := \backslash(g : (B \to bool)_r) = \forall\ a : A_r \{ g(f(a)) \}$$
$$new\_predictions := \backslash(g : candidate) = \sum b : B_r \wedge \neg A_r \{ \text{if } g(b) \{1\} \text{ else } \{0\} \}$$
$$strict\_candidate := \backslash(g : candidate) = \forall\ b : B_r \wedge \neg A_r \{ \neg g(b) \}$$
$$strict\_candidate <=> [new\_predictions]\ 0$$

Processing every element of `A_r` with `f` and storing the results in a set:

$$\exists f\{A_r\} : strict\_candidate$$

A user specified utility function evaluates candidates and gives them a score:

utility : candidate → real

Utility functions that rate candidates by number of new predictions have an asymmetric path:

utility[new_predictions → id] : nat → real

If the size of the set `B` returned by `f` is known, written `|∃f|`, then one can calculate the number of missing members:

missing := |∃f| - |∃f{A_r}|

The utility function should rate a score zero when a candidate makes too few or many new predictions:

utility{[new_predictions] (¬= missing)} => (= 0)

This is because the candidate will return `true` when `∃f` returns `false`, or return `false` when `∃f` returns `true`, at least once. Since `$A_r$` and `$B_r$` are known, it is possible to perform this check for every candidate function.

If `|∃f|` is not known, the number of new predictions could make the algorithm smarter:

1. Sometimes the existential path has many logically equivalent expressions.
2. Because of 1) there is a chance `ex_rnd` returns multiple versions of the same function.
3. When 2) happens there will be multiple candidates yielding same number of new predictions.

Shorter programs are often more likely to be generated by `ex_rnd`, so logically equivalent expressions will be more common when they also are short. Searching for the shortest program is a mathematical formulation of Occam's razor called "Solomonoff's theory of inductive reference"[1]. However, instead of naively following Occam's razor, the algorithm could take advantage of a more sophisticated technique:

- A short program does not always mean it is a good candidate
- Sometimes the `ex_rnd` function produces similar programs for similar inputs
- By manipulating the input given to the `ex_rnd` function,
  e.g. by using a neural net, one could improve the quality of candidates over time

The problem with Occam's razor is that the shortest program gets much higher weight, as long as it is generated by a non-zero probability. Voting on the number of new predictions could be a way to un-bias the shortest programs and get the "center of attention span" from the trained candidate generator.

References

[1] Solomonoff's theory of inductive reference
https://en.wikipedia.org/wiki/Solomonoff%27s_theory_of_inductive_inference