Symbolic Machine Learning - Student project 1

March 26, 2017

1 Batch learning of s-CNF

The task of the student is to implement an agent which will interact with an oracle ("environment") an eventually learn a s-CNF concept that is hidden in the oracle. Randomly chosen $\epsilon, \delta \in [0, 1]$ are provided to the agent when it is initialized. The oracle engages in the following session with the agent. Here $x_k = (o_k, r_k)$ is what the environment says (an observation provided to agent, and a reward), and y_k is what the agent says.

$o_1 = (n, s)$ $r_1 = 0$	Oracle samples $s, n \in N$ randomly but reasonably (w.r.t. resources, instructor's responsibility), and a random s-CNF, i.e a CNF with no more than s literals in any clause, involving no more than n propositional variables.
$y_1 = m$	Agent requests m examples, which will be the remaining length of the training phase. The agent has to do a PAC-theoretical calculation to work out m.
$o_2 = (v_1, v_2, \ldots, v_n)$ $r_2 = 0$	$v_i \in \{0; 1\}$ are truth values of propositional symbols p_i in an example sampled i.i.d from the uniform distribution on $\{0; 1\}^n$
$y_2 = y$	$y \in \{0; 1\}$ is the truth value predicted for o_2 (target CNF false or true with assignment o_2) by agent's current model.
$o_3 = (v_1, v_2, \ldots, v_n)$ $r_2 = r$	v_i just like for o_2. $r = 1$ if y_2 was correct, $r = 0$ otherwise.

and so on until step $K = m+2$. r_K is the last reward communicated to the agent. o_K is the first testing example. The testing phase rewards $r_{K+1}, r_{K+2}, \ldots$ are not revealed to the agent (it receives e.g. a blank symbol instead - implemented as a `None` object) so it cannot update its model any longer. The testing phase continues until some time K' where

$$\widehat{Acc} = \frac{1}{K' - K - 1} \sum_{k=K+1}^{K'} r_k$$

converges, and then the oracle says $o_{K'} = \texttt{stop}$. (This is implemented by the method `has_more_samples` of the oracle returning `False`.)

The *session is a success* if $\widehat{Acc} \geq 1 - \epsilon$.

The sessions are repeated until the relative frequency of success converges to $\widehat{S}$.

The *project is a success* if $\widehat{S} \geq 1 - \delta$. In the case of success, the full score is achieved if the number of examples requested is at most (roughly) the number predicted by PAC theory. With more examples requested, score should go down.

The *anticipated learning algorithm* first establishes its own propositional variables $p'_1, p'_2, \ldots$ each equivalent to one possible clause made out of $p_1, p_2, \ldots p_n$ (there is only $\mathcal{O}(n^s)$ of such clauses) and then uses the standard procedure for learning monotone conjuctions on the $p'_1, p'_2, \ldots$ variables.

2 Implementation

The student is provided an archive file with the following files inside:

- `tutorial3.py` – a modified file used on the third tutorial with classes for conjunctive and disjunctive concepts,
- `cnf.py` – a file with the implementation of j-CNF concept without a learning method,
- `oracle.py` – the implementation of the oracle,
- `project1.py` – an executable file which runs and evaluates the interaction of the agent with the oracle, i.e. it computes the values $\widehat{Acc}$ and $\widehat{S}$ and
- `agent.py` – the implementation of the agent itself which the student is supposed to complete.

The code in `agent.py` contains a code skeleton of a class called `Agent`. The student must implement a method of that class called `interact_with_oracle` which takes one parameter – an instance of `OracleSession` defined in `oracle.py`. The requirements for the implementation are the following: The agent should first call the `request_parameters` method of the oracle which returns a tuple $(n, s) = o_1$. Then the agent should call the `request_dataset` method with an integer parameter m. The method returns a single initial observation o_2. Finally, the agent should then call the `OracleSession.predict` method with a Boolean parameter indicating the prediction for the last observation until the condition `OracleSession.has_more_samples` is no longer satisfied. The `OracleSession.predict` method returns a tuple containing a new observation o_k and reward $r_{k-1} \in \{0, 1, \texttt{None}\}$ for the last prediction. The reward is `None` iff the agent is in the testing phase, i.e. $k > m$. An observation $o_k \in \{0, 1\}^n$ is represented as a 1-D numpy array of Booleans of length n. The student *must NOT* interact with the oracle in any other way than using the methods mentioned here. Specifically, it is forbidden to access (read or write) any attributes of the `OracleSesssion` instance. The student *may* include additional files besides `agent.py`, but *must not* modify the other provided files. Please submit the project as an archive file containing all the files.

The student may run the file `project1.py` to test their implementation. However, whether the number of requested samples m was good will be determined by the instructor after the work is submitted. Specifically, m should be generated by a theoretically derived formula for PAC-learnability of s-CNF.

The student is allowed to use any functions from the Python standard library and also the `numpy` library, but no other external libraries. The solution must be compatible with Python version $\geq$ 3.5. Hint: The `itertools` module from the standard Python library may come handy.

The student may optionally submit a report along the source code, documenting whatever may not be clear from the source code.

3 Scoring

The project is worth 12 points. (Out of the 50 available during the course.) Correct implementation of the learning algorithm makes for 7 points. (Mandatory) Correct formula for m makes for 3 points. (Mandatory) If the student implements the online version of the algorithm (i.e. not storing observations in memory) he or she earns additional 2 points. (Optional)