Learning from Entailment Cont', HW Assignment

B4M36SMU

This tutorial presents a homework assignment for the ILP part. After that, an extension of lgg is presented.

Homework Assignment

You are going to learn the rules of a card game from observing plays. You know the following: The game is played with a fragment of cards which has a color, value, and shape. The only known restriction is for the values of cards, e.g. from two to ace; the number of colors and shapes may vary from game to game. There are two players and both draw the same number of cards. Which player won is determined only by the cards they drew. The rules may be unfairly biased in favor of one of the players. You repeatedly observe what cards are dealt and whether the first player won. From this, you are to learn to predict whether the first player will win depending on which cards both players dealt.

Implementation Details

Download the archive, see *readme* file for installation instructions and get in touch with the framework, specifically, by running *minlog.ipynb* Jupyter notebook[1]. The structure of the classes is the same as the composition of first-order logic syntax described in the first ILP tutorial. Your task is to implement behavior of the agent in *agent.py* by implementing methods *receiveSample*, *receiveReward* and *getHypothesis*. To run the environment with a game see file *assignment2.py*.

These methods work similarly to previous homework. Firstly, the environment gives the agent an observed play by calling *recieveSample* and the agents returns 0 or 1 if the first player wins or the second one. After that, the environment returns reward to the agent by calling *recieveReward* with 0 or -1 indicating that the agent's prediction was correct or not. Upon a request of the environment, by calling *getHypothesis*, the agent returns a string representation of its current hypothesis.

The cards are in the form of $COLOR\_VALUE\_OF\_SHAPE$, e.g. $RED\_KING\_OF\_HEARTS$. A play is then a tuple of hands, i.e. a list of cards a player holds; for example $([RED\_ACE\_OF\_HEARTS, RED\_ACE\_OF\_LEAVES], [RED\_OBER\_OF\_BELLS, BLUE\_UNTER\_OF\_LEAVES])$ is a possible observed play in which the first players has got two aces while the second one has not got any ace. It is ensured that the values of cards are in scope of two, three, four,..., ten, jack, queen, king, ace with respective values (2, 3, 4,..., 10, 11, 12, 13, 14).

Besides the fact that a player's hand is represented as a list of raw cards, the agent has to deal with possible different parametrizations of the game and the target concept which may vary from game to game. For example, in one run of the players battle, each one picks only two cards, while in another battle they pick four. Similarly, the target concept may vary as well, e.g. one time wins the player with a bigger total sum of cards values he holds, while in another setting a specific combination of cards determines the winner, e.g. some kind of flush. Moreover, the target concept may be a conjunction of these, e.g. the first player wins only if it has a flush and the total sum of his cards is bigger than the opponent's one. For each battle, i.e. a sequence of observed plays, it holds that the target concept and the battle settings do not change during single plays.

The agent knows that order of the cards on a player's hand does not matter.

Write a (very) brief PDF report describing your representation and discussing its pros and cons. Run a few iterations of the environment and discuss whether your agent learns reasonable hypotheses. You may implement your own target class (see class *Rule* in *assignment2.py*) and discuss the behavior of the agent with this target concept.

You can modify only the *agent.py* file (and *assignment2.py* for testing purposes). Anything else is strictly forbidden[2].

[1] The framework is typed, so it is very beneficial to use some clever IDE for Python, which can help you greatly.

[2] However, if you feel that some general functionality should be added to the library, feel free to express this idea in an

What to Submit

Submit a zip archive which contains only *agent.py* file with Python 3.6+ and a PDF report.

Grading

The whole assignment is worth 15 points[3]. Correct implementation of the main loop concerning online learning agent is worth 5 points. A solution capable of learning all concepts based on colors, values, shapes, flushes, etc. and combinations of these is worth 7 points. Finally, the report is worth 3 points. Do not overfit your solution to the provided example because the agent will be evaluated on a various number of games with different target concepts and environment setting.

Take care of the readability of the solution; in the very unexpected case that the solution is not working, there is a low probability of gaining at least some points for general ideas, but unreadable code zeros this probability. This homework assignment has to be elaborated independently with no collaboration at all[4].

Relative Least General Generalization

In some cases, the agent may possess some additional information about the world, let say a background theory $\mathcal{B}$. Then, the learning task changes to searching a hypothesis Φ which entails just positive observations with the background theory, i.e.

$$\forall o \in O^+ : \mathcal{B} \cup \{\Phi\} \vdash o,$$
$$\forall o \in O^- : \mathcal{B} \cup \{\Phi\} \nvdash o.$$

We will restrict ourselves to a background theory $\mathcal{B}$ such that it is a finite set of ground facts, e.g. $\mathcal{B} = \{human(Adam), human(Eva)\}$. In general, the theory can contain non-ground first-order rules, but this out of the scope of this tutorial.

Before jumping to the lgg enriched by a background theory, we have to define a few things[5], but most of them are analogical to what you have seen in the last tutorial. Firstly, note the fact that if $\gamma_1 \subseteq_\theta \gamma_2$, then $\gamma_1\theta \implies \gamma_2$ is a tautology. The core idea here is to use θ-subsumption with respect to a background theory $\mathcal{B}$, that is, $\gamma_1 \subseteq_\theta^{\mathcal{B}} \gamma_2$ if there exists a substitution θ such that $\mathcal{B} \vdash (\gamma_1\theta \implies \gamma_2)$. While having defined $\subseteq_\theta^{\mathcal{B}}$, the definitions of θ-subsume equivalence with respect to $\mathcal{B}$, $\approx_\theta^{\mathcal{B}}$, strict θ-subsumption w.r.t. $\mathcal{B}$, etc., are analogical to θ-subsumption ones.

Now, we will define the *relative least general generalization* with respect to a background theory $\mathcal{B}$ of two clauses γ_1 and γ_2, which is given by

$$\begin{aligned} RLGG_{\mathcal{B}}(\gamma_1, \gamma_2) &= \gamma \\ s.t.\ \ &\gamma \subseteq_\theta^{\mathcal{B}} \gamma_1 \\ &\gamma \subseteq_\theta^{\mathcal{B}} \gamma_2 \\ &\forall \gamma' s.t. \gamma' \subseteq_\theta^{\mathcal{B}} \gamma_1 \wedge \gamma' \subseteq_\theta^{\mathcal{B}} \gamma_2 : \gamma' \subseteq_\theta^{\mathcal{B}} \gamma \end{aligned}$$

Note that the only difference with lgg is replacing $\subseteq_\theta$ by $\subseteq_\theta^{\mathcal{B}}$. The last step here is to formally define the computation of rlgg, which can be processed as follows:

$$RLGG_{\mathcal{B}} = LGG(\gamma_1 \vee_{l \in \mathcal{B}} \neg l, \gamma_2 \vee_{l \in \mathcal{B}} \neg l).$$

Consider $\gamma_1 = p(x)$, $\gamma_2 = q(x)$ and $\mathcal{B} = \{r(A, B), r(A, A)\}$, rlgg of these two is:

$$\neg r(A, B) \vee \neg r(A, A) \vee \neg r(A, x_{A,B}) \vee \neg r(A, x_{B,A})$$

e-mail or the course forum.

[3] In case you find a bug in the provided code, you may get one extra point, but the maximum of points from the assignment is still 15 points. Please report these bugs at the course forum, so that only one point is distributed among all students for each found bug.

[4] Think for your future position in a government.

[5] You've seen it at the lecture [1].

One can easily see that γ_1 and γ_2 do not have any lgg, yet rlgg products a non-empty disjunction of literals. However, this follows from the fact that the clause above is not reduced. So, we can either return a reduced clause from the computation of lgg inside the rlgg call and remove redundant literals w.r.t. $\mathcal{B}$ afterwards, e.g.:

$$RLGG_{\mathcal{B}} = LGG(\gamma_1 \impliedby \mathcal{B}, \gamma_2 \impliedby \mathcal{B}) \setminus \{\neg l \mid l \in \mathcal{B}\}$$

Another possible way to compute rlgg of two clauses is to compute clause reduction w.r.t. $\mathcal{B}$ of non-reduced lgg of the extended clauses by the $\mathcal{B}$. Reduction of a clause w.r.t. $\mathcal{B}$ can be done by the same algorithm as for standard reduction just with the small difference of using $\subseteq_\theta^{\mathcal{B}}$ instead of $\subseteq_\theta$. Recall that γ_1 θ-subsumes γ_2 w.r.t. $\mathcal{B}$ if $\gamma_1 \subseteq_\theta \gamma_2 \vee \neg\mathcal{B}$.

Exercise

- Compute rlgg of γ_1 and γ_2 w.r.t. $\mathcal{B}$.
  - $\gamma_1 = moveable(A)$
  - $\gamma_2 = moveable(B)$
  - $\mathcal{B} = \{on(A, C), on(C, D), on(B, D), shape(A, Box), shape(B, Box), shape(C, Box), shape(D, Floor)\}$

References

[1] Filip Železný and Jiří Kléma. *SMU textbook*. 2017.