Tutorial for using SCIRL

Edouard KLEIN

August 10, 2012

Contents

SCIRL is an algorithm for Inverse Reinforcement Learning (IRL). [2].

1 On a discrete problem (GridWorld)

A classical toy problem in IRL is the GridWorld. Please refer to [4] for a description of the setting.

The dimensionality of the state space is 2, and the action space has a dimensionality of 1 and a cardinality of 4 (the 4 cardinal directions).

The input of the algorithm is a matrix of transitions from the expert file:D_GW.mat. The heavily commented python source file file:SCIRL_GridWorld.py show how to use the code of the algorithm (which can be found in file:LAFEM.py) to output a reward.

For the sake of simplicity and legibility, we do nothing with the reward here although in the papers we published about SCIRL, we optimize it using an exact dynamic programming algorithm in order to assess the its quality. The parameters used in this example are the same as those used in the papers. This reward thus is a "good" reward. The complete code for reproducing the figures of the papers is available upon request, although its value is limited

by the huge amount of problem specific or ugly code that lay not translate well in the reader's setting. The goal of this tutoriel is only to show how to use the code of the algorithm to get a reward from expert transition, nothing more.

Invoking

```
python SCIRL_GridWorld.py D_GW.mat
```

on the command line should print the reward on the standard output and some information about the gradient descent at the heart of SCIRL on the standard error. You may want to redirect the standard output to a file with `>`.

Alternatively, one can invoke

```
make SCIRL_GW
```

1.1 A note about the computation of μ_E

The data needed by SCIRL alone need not be trajectories. Mere (s_i, a_i) couples suffice. However, one must give SCIRL a estimate of μ_E for every couple $(s_i, a \in A)$, where s_i are the states in the data given to SCIRL. The technique (discussed in the papers) used here is a simple heuristic based on a Monte-Carlo estimation.

The data in the file:D_GW.mat matrix is in the form of one transition per line, each transition been of the form s, a, s', r, eoe. A few notes :

- Here r, the reward part of the transition, is always 0 and will not be used. It is present because this format of transition is the one used by my implementation of Reinforcement Learning algorithm and I did not want to use many different formats.
- *eoe* is a marker of end of episode. When it is 1, it means that the transition on the next line is in the same trajectory as the current transition (indeed, you can verify that s' for this transition is s for the next). When it is 0, it means that the trajectory ends here.

The `data` member of the SCIRL class is built using the first three columns of the data file, that is to say only the s and a fields of each transitions. The estimate of mu_E, however, makes use of the information of *eoe* as well. For every (s, a) couple in the file, every trajectory is cut from (s, a) (if it appears in the trajectory at all) onwards, and this subtajectory is added in a database. Then, a simple Monte-Carlo estimation gives an estimate for every

couple (s_i, a_i) in the datafile. The heuristics happens when SCIRL request a μ_E estimate for a couple $(s_i, a \neq a_i)$: we answer $\gamma\hat{\mu}_E(s_i, a_i)$. The rationale behind this is to say that using a non expert transition will make the agent "loose some time" and thus we multiply the estimate by the discount factor.

2 On a continuous problem (Inverted Pendulum)

Looking for a toy problem the state space of which is continuous, we settled on the Inverted Pendulum problem, a classical benchmark in the Reinforcement Learning litterature, a torough description of which can be found in [3].

The dimensionality of the state space is 2 (position and speed), and the action space has a dimensionality of 1 and a cardinality of 3 (Right, Left, or Nothing).

The input of the algorithm is a matrix of transitions from the expert file:D_IP.mat. The non-commented python source file (see the example on the GridWorld for comments) file:SCIRL_InvertedPendulum.py show how to use the code of the algorithm (which can be found in file:LAFEM.py) to output a reward. The estimate of μ_E is done with the LSTDμ algorithm in its on-policy form, the code of which lies in file:LSTDmu.c.

For the same reasons as on the GridWorld, we only output the reward but do not train an agent on it.

Invoking

```
make SCIRL_IP
```

on the command line will compile the C code for LSTDMu and run it, and then run SCIRL.

2.1 Note on the estimation of μ_E

For this problem, unlike what has been done on the GridWorld, μ_E is evaluated with an algorithm called LSTDμ [1]. This is well suited for problems with e.g. spatial coherence.

3 Further Work and things susceptible to change

The LSTDμ algorithm should be implemented in python in order to avoid having to write the feature function both in C and python.

Both the Monte-Carlo heuristics and the LSTD-μ algorithm implementations should be moved in the parent class with an easy to set option to choose which one to use or to let the uer implement his/her own.

References

[1] Edouard Klein, Matthieu Geist, and Olivier Pietquin. Batch, Off-policy and Model-free Apprenticeship Learning. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL 2011)*, Lecture Notes in Computer Science (LNCS), page 12 pages, Athens (Greece), september 2011. Springer Verlag - Heidelberg Berlin.

[2] Edouard Klein, Bilal PIOT, Matthieu Geist, and Olivier Pietquin. Structured Classification for Inverse Reinforcement Learning. In *European Workshop on Reinforcement Learning (EWRL 2012)*, Edinburgh (UK), 2012.

[3] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[4] A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.