Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Modified reward function on abstract features in inverse reinforcement learning

Shen-yi CHEN[†], Hui QIAN, Jia FAN, Zhuo-jun JIN, Miao-liang ZHU
(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)
[†]E-mail: charles_csy@zju.edu.cn

Received Aug. 7, 2009; Revision accepted Oct. 19, 2009; Crosschecked May 31, 2010; Published online Aug. 3, 2010

Abstract: We improve inverse reinforcement learning (IRL) by applying dimension reduction methods to automatically extract abstract features from human-demonstrated policies, to deal with the cases where features are either unknown or numerous. The importance rating of each abstract feature is incorporated into the reward function. Simulation is performed on a task of driving in a five-lane highway, where the controlled car has the largest fixed speed among all the cars. Performance is almost 10.6% better on average with than without importance ratings.

Key words: Importance rating, Abstract feature, Feature extraction, Inverse reinforcement learning (IRL), Markov decision process (MDP)
doi:10.1631/jzus.C0910486 **Document code:** A **CLC number:** TP181

1 Introduction

Given the reward function, there exist a number of standard algorithms for finding an optimal or near-optimal policy in Markov decision process (MDP) formalism (Puterman, 1994), such as the Q-learning method (Mitchell, 1997) in reinforcement learning (Sutton and Barto, 1998). When the reward function is unknown, which is often the case in practice, parameters need to be manually adjusted until the best model emerges (Ng *et al*., 1999). To overcome this difficulty, Ng and Russell put forward a method called inverse reinforcement learning (IRL) (Russell, 1998; Ng and Russell, 2000). Assuming that the reward function was a linear combination of a limited number of known features, they derived the constraints of the so-called feature functions.

IRL is a kind of imitation learning. Most of the imitation learning methods attempt to directly mimic the demonstration, including those described by Pomerleau (1989), Sammut *et al*. (1992), Hayes and Demiris (1994), Kuniyoshi *et al*. (1994), and Amit and Mataric (2002). One notable exception is given by Atkeson and Schaal (1997). They considered the problem of having a robot arm follow a demonstrated trajectory, and used a reward function that quadratically penalizes deviation from the desired trajectory. Inspired by their work, Abbeel and Ng (2004; 2005) turned the reward function learning problem into a quadratic programming formulation, with the assumption that any single-step deviation from expert's policy should be penalized as costly as possible.

IRL has had many successful applications, such as aerobatic demonstrations of helicopters (Abbeel and Ng, 2005; Coates *et al*., 2009), quadruped locomotion over rough terrain (Rebula *et al*., 2007; Kolter *et al*., 2008a; 2008b), and motion planning in parking lot navigation (Abbeel *et al*., 2008). One weak point in all these applications is the artificial selection of appropriate features for the reward function. Not only is the quality of the features important to the special applications, but also the quantity. The number of the features should not be too small to exclude major features, while it should not be too large to make it incomputable. Usually, it takes time to choose the most significant features, and further it is possible that not all of them can be completely discovered. In our method, IRL is improved by applying dimension reduction methods to automatically extract abstract features from human-demonstrated policies, with the consideration of the importance rating of these abstract features.

2 Preliminaries

A (finite-state) MDP is a tuple (S, A, T, γ, R), where S is a finite set of states, A is a finite set of actions, T: $S\times A\times S \mapsto [0, 1]$ is the state transition probability function, $\gamma\in[0, 1)$ is the discount factor, and R: $S\mapsto\mathbb{R}$ is the reward function.

A stationary stochastic policy has a mapping π': $S\times A \mapsto [0, 1]$. For any $s\in S$, $a\in A$, the policy $\pi(s)=\arg\max_{a\in A}\pi'(s,a)$. Thus, a stationary policy ('policy' for short), stochastic or non-stochastic, is a mapping π: $S\mapsto A$. The value function of policy π is: $V^{\pi}(s)=R(s)+\gamma\sum_{s'}T(s,\pi(s),s')V^{\pi}(s')$.

The IRL's objective is to find an appropriate reward function that maximizes the difference between the value function of human-demonstrated policy π^* and the optimal policy π generated by the reinforcement learning (RL) method (Sutton and Barto, 1998), starting from the state s_0. At the beginning of the algorithm, the reward function is randomly chosen, and the first corresponding policy $\pi^{(0)}$ is obtained by RL. When the reward function is adjusted as follows, the second $\pi^{(1)}$ can also be obtained. If there are k different policies, $\pi^{(0)}$, $\pi^{(1)}$, …, $\pi^{(k-1)}$, available up to now, the objective is changed to first select one of the policies that has the minimum difference from π^*, and second to maximize that difference. The formal definition could be given by

$$\max_{R}\min_{j\in\{0,1,2,\cdots,k-1\}}\left(V^{\pi^*}(s_0)-V^{\pi^{(j)}}(s_0)\right). \quad (1)$$

If the number of selected features is d, then the number of corresponding feature functions is d as well. The reward function is a linear combination of the feature functions, which is the assumption of IRL: $R(s)=\omega_1\phi_1(s)+\omega_2\phi_2(s)+...+\omega_d\phi_d(s)=\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\phi}(s)$, where $\boldsymbol{\phi}$: $S\mapsto[0, 1]^d$ is the vector of feature functions over states, and $\boldsymbol{\omega}\in\mathbb{R}^d$, $\|\boldsymbol{\omega}\|_2\leq1$ is the coefficient vector. The value function of policy π can be rewritten as

$$V^{\pi}(s_0)=\boldsymbol{\omega}^{\mathrm{T}}E\left[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\phi}(s_t)\,|\,\pi\right]=\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\mu}(\pi), \quad (2)$$

where $\boldsymbol{\mu}(\pi)$ is the so-called 'feature expectation':

$$\boldsymbol{\mu}(\pi)=E\left[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\phi}(s_t)\middle|\pi\right]\in\mathbb{R}^d. \quad (3)$$

In the finite horizon case, the feature expectation can be approximated to a vector of feature appearance frequency. Thus, the feature expectation of optimal policy π^* can be easily calculated.

From Eqs. (1)–(3), the problem's formulation turns into a quadratic program (QP):

$$\begin{aligned}&\max_{\boldsymbol{\omega}}\min_{j\in\{0,1,2,\cdots,k-1\}}\left(\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{\mu}(\pi^*)-\boldsymbol{\mu}(\pi^{(j)}))\right)\\&\text{s.t.}\qquad \|\boldsymbol{\omega}\|_2\leq1.\end{aligned} \quad (4)$$

Both the max-margin method and the projection method can solve this QP. When a new reward function is determined, the new optimal policy $\pi^{(k)}$ is obtained accordingly. Then policy $\pi^{(k)}$ is put back into Eq. (4), and the algorithm iteratively generates policies until the guaranteed convergence is reached.

The features could be picked in state-action pair space, but the number of these features is too large. It is wiser to extract abstract features from the human-demonstrated policies.

3 Algorithm

The policy features of complicated MDPs usually have high dimensions. It is reasonable to assume that there exists a subspace which suffices to express the characteristics of human-demonstrated policies, because these policies constitute a very small part of the whole policy space. The algorithm extracts new dimensions from this subspace by principal component analysis (PCA), and uses these dimensions as new features.

Because the state space is finite, let $|S|=N$, which means there are N different states. Likewise, let $|A|=M$, which means there are M different actions. Our goal is to find some abstract features (suppose the number is d) to calculate the feature expectation $\boldsymbol{\mu}_\pi^d$ for any policy π, such that

$$\boldsymbol{\mu}_\pi^d = E\left[\sum_{t=0}^{\infty}\gamma^t \boldsymbol{\phi}(s_t,\pi(s_t))\middle|\pi\right]\in\mathbb{R}^d. \quad (5)$$

Abstract features can be extracted from state-action pairs. State-action pairs are good features for IRL, but there will be $N\times M=\Omega$ initial features at the maximum. Denoting $\boldsymbol{\Phi}$: $S\times A\mapsto[0,\ 1]^{\Omega}$ as the vector of feature functions over a state-action pair, and $\boldsymbol{\zeta}_\pi^{\Omega}$ as the feature expectation, their relation is clear:

$$\boldsymbol{\zeta}_\pi^{\Omega} = E\left[\sum_{t=0}^{\infty}\gamma^t \boldsymbol{\Phi}(s_t,\pi(s_t))\middle|\pi\right]\in\mathbb{R}^{\Omega}. \quad (6)$$

Given τ ($\tau\in\mathbb{N}$) human-demonstrated policies π_1, π_2, …, π_τ, the feature expectation matrix is $\boldsymbol{\zeta}^{\Omega}=\left[\boldsymbol{\zeta}_{\pi_1}^{\Omega},\boldsymbol{\zeta}_{\pi_2}^{\Omega},\ldots,\boldsymbol{\zeta}_{\pi_\tau}^{\Omega}\right]$. Using PCA, eigenvectors of symmetric matrix $\boldsymbol{\zeta}^{\Omega}$ must be calculated first. Eigenvectors are sorted by eigenvalues in descending order. Those with large eigenvalues are chosen as the abstract features. Experimentally, those constitute more than 99.9% of the sum of eigenvalues are selected. The number of abstract features is denoted as d.

Although these features are abstract, the feature expectation is computable. For policy π', the feature expectation is

$$\boldsymbol{\mu}_{\pi'}^d = \left[\boldsymbol{e}_1,\boldsymbol{e}_2,\cdots,\boldsymbol{e}_d\right]^{\mathrm{T}}(\boldsymbol{\zeta}_{\pi'}^{\Omega}-\overline{\boldsymbol{\zeta}^{\Omega}}), \quad (7)$$

where $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, …, $\boldsymbol{e}_d$ are d eigenvectors, and $\overline{\boldsymbol{\zeta}^{\Omega}}$ is the mean vector of the feature expectation of τ human-demonstrated policies:

$$\overline{\boldsymbol{\zeta}^{\Omega}} = \frac{1}{\tau}\sum_{\eta=1}^{\tau}E\left[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\Phi}(s_t,\pi_\eta(s_t))\mid\pi_\eta\right]. \quad (8)$$

From Eqs. (5)–(7), the vector of abstract feature functions is

$$\boldsymbol{\phi}(s,\pi(s)) = \left[\boldsymbol{e}_1,\boldsymbol{e}_2,\cdots,\boldsymbol{e}_d\right]^{\mathrm{T}}\boldsymbol{\Phi}(s,\pi(s)), \quad (9)$$

where the codomain of feature functions is extended: $\boldsymbol{\phi}$: $S\mapsto[-1,\ 1]^d$. Negative values stand for absence of the corresponding feature, which can be regarded in the reward function as negative reinforcement.

Different abstract features have different importance ratings. By coincidence, those eigenvalues are the natural indicators of importance. An eigenvalue is in essence the square of standard deviation that implies the importance of extracted space dimension. So, the larger the eigenvalue, the more important the corresponding abstract feature. For this reason, the new reward function is defined as

$$\begin{aligned}R(s,\pi(s)) = {} & \omega_1\phi_1(s,\pi(s))\sqrt{\mathrm{ev}_1}+\omega_2\phi_2(s,\pi(s))\sqrt{\mathrm{ev}_2}\\ & +\ldots+\omega_d\phi_d(s,\pi(s))\sqrt{\mathrm{ev}_d},\end{aligned} \quad (10)$$

where ev_1, ev_2, …, ev_d are d eigenvalues.

The modification of the reward function favors the more important abstract features, and ignores the less important. Abstract features with small eigenvalues are in fact omitted. This form of reward function explains why those eigenvectors can be deleted. The unimportant abstract feature has negligible items to be added in the reward function.

The algorithm continues with the projection method until the guaranteed convergence is reached.

4 Experiments

The car driving simulation has the same user interface as Abbeel's, for an easier comparison (Fig. 1). The car is driving on a highway at a fixed speed, faster than all the other cars. Other cars are driving in left, middle, and right lanes. This MDP has

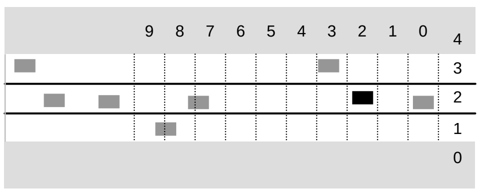

Fig. 1 Highway driving simulation interface with 5 lanes and 10 grids distance in lanes 1, 2, and 3

five different actions, three of which cause the car to steer smoothly to one of the lanes, and two of which cause it to drive off (but parallel to) the road, on either the left or the right side. Because the speed is fixed, it is sometimes necessary to drive off the road if one wants to avoid hitting other cars. So there are altogether 5 lanes, including 2 off-road lanes, marked 0, 1, 2, 3, and 4 respectively. To describe a state, one needs to know which lane the car is in, and the distances of the closest car in each lane (no other cars in the two off-road lanes). The distance to the car is discretized uniformly into 10 grids (Fig. 1).

The number of total states is 5×10×10×10=5000. If all the state-action pairs are considered as features, the number is obviously very large (5000×5=25 000). Abstract features must be extracted.

The simulation was run at 10 Hz, and the feature expectations were estimated from a single trajectory of 1200 time steps (corresponding to 2 min of driving time), standing for the finite horizon of the MDP.

A variety of driving styles were demonstrated so as to see whether the algorithm could mimic the same 'style' in every instance. There were 19 different driving styles generated by lab colleagues, as shown in Table 1.

Some of the driving styles were selected to form the space of human-demonstrated policies, and the others were left as test sets. The experiment shown here is a worst case with relatively complicated driving styles (LM3 and RM3) chosen as test sets, and 17 others forming the demonstration space. Those that constituted more than 99.9% of the sum of eigenvalues were selected. Thus, in this experiment, 11 eigenvectors (Fig. 2) were extracted as abstract features. Note that there were four particularly large eigenvalues, indicating the lane feature information. It makes sense because four dimensions are enough to express five lanes.

Table 1 Nineteen different driving styles generated by lab colleagues

No.	Driving style	Description
1	LN	Drive mainly in the left lane, and avoid hitting cars. If nowhere to go, go off-road
2	MN	Drive mainly in the middle lane, and avoid hitting cars. If nowhere to go, go off-road
3	RN	Drive mainly in the right lane, and avoid hitting cars. If nowhere to go, go off-road
4	LA	Always drive in the left lane, regardless of hitting cars
5	MA	Always drive in the middle lane, regardless of hitting cars
6	RA	Always drive in the right lane, regardless of hitting cars
7	LOA	Always drive in the left off-road way
8	ROA	Always drive in the right off-road way
9	LM1	Drive mainly in the left lane, and drive to the left off-road to avoid hitting cars from a close distance. Then back to the left lane
10	LM2	Drive mainly in the left lane, and drive to the left off-road to avoid hitting cars from a far distance. Then back to the left lane
11	LM3	Drive mainly in the left lane, and avoid hitting cars. When overtaking, the middle lane is preferred. If the middle lane is dangerous (there will be a collision), go left off-road. If the left lane is safe, go back to the left lane
12	MM1	Drive mainly in the middle lane, and avoid hitting cars. When overtaking, the left lane is preferred. Secondly, go left off-road. Thirdly, go to the right lane. Fourthly, go right off-road. If the middle lane is safe, go back to the middle lane
13	MM2	Drive mainly in the middle lane, and avoid hitting cars. When overtaking, the right lane is preferred. Secondly, go right off-road. Thirdly, go to the left lane. Fourthly, go left off-road. If the middle lane is safe, go back to the middle lane
14	RM1	Drive mainly in the right lane, and drive to right off-road to avoid hitting cars from a close distance. Then go back to the right lane
15	RM2	Drive mainly in the right lane, and drive to right off-road to avoid hitting cars from a far distance. Then go back to the right lane
16	RM3	Drive mainly in the right lane, and avoid hitting cars. When overtaking, the middle lane is preferred. Secondly, go right off-road. If the right lane is safe, go back to the right lane
17	LOB	Drive mainly in the left off-road lane and hit as many left lane cars as possible
18	MB	Drive mainly in the middle lane and hit as many other cars as possible
19	ROB	Drive mainly in the right off-road lane and hit as many right lane cars as possible

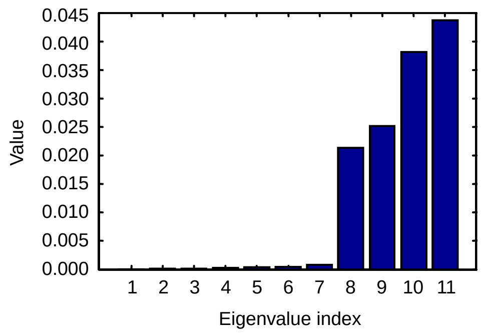

Fig. 2 The 11 eigenvalues for abstract features
Note that there were four particularly large eigenvalues, indicating the lane feature information

The following two cases were compared by policy similarity in the 19 driving styles: (1) the original algorithm without importance rating; (2) the modified algorithm with importance rating.

Policy similarity is a measure of how well the learned policy imitates the human-demonstrated policy. It equals the number of the same state-action pairs in both policies divided by the total number of the state-action pairs. The comparison results are shown in Table 2 and Fig. 3.

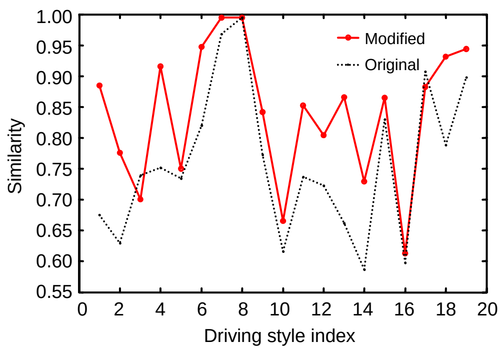

Fig. 3 Similarities of learned policies for the 19 driving styles (Table 1) in two learning cases, with the original algorithm and with the modified algorithm
The modified algorithm was almost 10.6% better on average. In learning complicated styles, the similarity was improving

Notice that the policy similarities of two test sets, LM3 (No. 11) and RM3 (No. 16), were improving. As well, other complex styles such as LN (No. 1), MN (No. 2), MM1 (No. 12), MM2 (No. 13), RM1 (No. 14), and MB (No. 18) were also improving. The mean performance of the modified algorithm was almost 10.6% better than that of the original.

Table 2 Comparison of the original algorithm and the modified one between 11 abstract features

Driving style	Style similarity (%)	
	Original algorithm	Modified algorithm
LN	67.50	88.50
MN	62.93	77.56
RN	73.91	70.05
LA	75.13	91.53
MA	73.40	75.00
RA	82.01	94.71
LOA	96.83	99.47
ROA	99.48	99.48
LM1	77.25	84.13
LM2	61.65	66.50
LM3*	73.68	85.26
MM1	72.25	80.38
MM2	66.17	86.57
RM1	58.67	72.96
RM2	83.01	86.47
RM3*	59.80	61.31
LOB	90.69	88.24
MB	78.82	93.10
ROB	89.85	94.42
Mean	75.95	83.98

* LM3 and RM3 were chosen as the test sets, while the others formed the driving style subspace

Although the policy similarity cannot exactly describe the closeness of two policies, the learning results looked significantly better in our videos.

5 Conclusions

Our algorithm assumed that the reward function is expressible as a linear function of some abstract features that can be extracted from human-demonstrated policies. It is worth incorporating the importance rating of each abstract feature into the form of the reward function.

Direct policy learning is inclined to lead one to do the most frequent actions in a certain state. This method, on the other hand, is not as short-sighted. Despite the fact that the visited state-action pair reinforced in the reward function is favored in the corresponding generated policy, the algorithm emphasizes the long run effect, due to the use of the value function. Another advantage is that although the number of used features is greatly reduced, the most important information is still maintained.

Future work should examine whether there is a better form of reward function than the linear one, and how to make it rapid enough to learn policy online.

References

Abbeel, P., Ng, A.Y., 2004. Apprenticeship Learning via Inverse Reinforcement Learning. Proc. 21st Int. Conf. on Machine Learning, p.1-8.

Abbeel, P., Ng, A.Y., 2005. Exploration and Apprenticeship Learning in Reinforcement Learning. Proc. 22nd Int. Conf. on Machine Learning, p.1-8. [doi:10.1145/1102351.1102352]

Abbeel, P., Dolgov, D., Ng, A.Y., Thrun, S., 2008. Apprenticeship Learning for Motion Planning with Application to Parking Lot Navigation. Proc. Int. Conf. on Intelligent Robots and Systems, p.1083-1090.

Amit, R., Mataric, M., 2002. Learning Movement Sequences from Demonstration. Proc. 2nd Int. Conf. on Development and Learning, p.203-208. [doi:10.1109/DEVLRN.2002.1011867]

Atkeson, C., Schaal, S., 1997. Robot Learning from Demonstration. Proc. 14th Int. Conf. on Machine Learning, p.12-20.

Coates, A., Abbeel, P., Ng, A.Y., 2009. Apprenticeship learning for helicopter control. *Commun. ACM*, **52**(**7**):97-105. [doi:10.1145/1538788.1538812]

Hayes, G., Demiris, J., 1994. A Robot Controller Using Learning by Imitation. Proc. 2nd Int. Symp. on Intelligent Robotic Systems, p.198-204.

Kolter, J.Z., Abbeel, P., Ng, A.Y., 2008a. Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion. Advances in Neural Information Processing Systems. MIT Press, Cambridge, p.769-776.

Kolter, J.Z., Rodgers, M.P., Ng, A.Y., 2008b. A Complete Control Architecture for Quadruped Locomotion over Rough Terrain. Proc. Int. Conf. on Robotics and Automation, p.811-818.

Kuniyoshi, Y., Inaba, M., Inoue, H., 1994. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Trans. Rob. Autom.*, **10**(6):799-822. [doi:10.1109/70.338535]

Mitchell, T., 1997. Machine Learning. McGraw Hill, New York, p.385-392.

Ng, A.Y., Russell, S., 2000. Algorithms for Inverse Reinforcement Learning. Proc.17th Int. Conf. on Machine Learning, p.663-670.

Ng, A.Y., Harada, D., Russell, S., 1999. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. Proc. 16th Int. Conf. on Machine Learning, p.278-287.

Pomerleau, D., 1989. Alvinn: an Autonomous Land Vehicle in a Neural Network. Advances in Neural Information Processing Systems 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p.305-313.

Puterman, M., 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, New York, NY.

Rebula, J.R., Neuhaus, P.D., Bonnlander, B.V., Johnson, M.J., Pratt, J.E., 2007. A Controller for the LittleDog Quadruped Walking on Rough Terrain. IEEE Int. Conf. on Robotics and Automation, p.1467-1473.

Russell, S., 1998. Learning Agents for Uncertain Environments. Proc. 11th Annual Conf. on Computational Learning Theory, p.101-103.

Sammut, C., Hurst, S., Kedzier, D., Michie, D., 1992. Learning to Fly. Proc. 9th Int. Workshop on Machine Learning, p.385-393.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning. MIT Press, USA.