A Cascaded Supervised Learning Approach to Inverse Reinforcement Learning

Edouard Klein[1,2], Bilal Piot[2,3], Matthieu Geist[2], and Olivier Pietquin[2,3,*]

[1] ABC Team LORIA-CNRS, France
[2] Supélec, IMS-MaLIS Research Group, France
firstname.lastname@supelec.fr
[3] UMI 2958 (GeorgiaTech-CNRS), France

Abstract. This paper considers the Inverse Reinforcement Learning (IRL) problem, that is inferring a reward function for which a demonstrated expert policy is optimal. We propose to break the IRL problem down into two generic Supervised Learning steps: this is the Cascaded Supervised IRL (CSI) approach. A classification step that defines a score function is followed by a regression step providing a reward function. A theoretical analysis shows that the demonstrated expert policy is near-optimal for the computed reward function. Not needing to repeatedly solve a Markov Decision Process (MDP) and the ability to leverage existing techniques for classification and regression are two important advantages of the CSI approach. It is furthermore empirically demonstrated to compare positively to state-of-the-art approaches when using only transitions sampled according to the expert policy, up to the use of some heuristics. This is exemplified on two classical benchmarks (the mountain car problem and a highway driving simulator).

1 Introduction

Sequential decision making consists in choosing the appropriate action given the available data in order to maximize a certain criterion. When framed in a Markov Decision Process (MDP) (see Sec. 2), (Approximate) Dynamic programming ((A)DP) or Reinforcement Learning (RL) are often used to solve the problem by maximizing the expected sum of discounted rewards. The Inverse Reinforcement Learning (IRL) [15] problem, which is addressed here, aims at inferring a reward function for which a demonstrated expert policy is optimal.

IRL is one of many ways to perform Apprenticeship Learning (AL): imitating a demonstrated expert policy, without necessarily explicitly looking for the reward function. The reward function nevertheless is of interest in its own right. As mentioned in [15], its semantics can be analyzed in biology or econometrics for instance. Practically, the reward can be seen as a succinct description of a task. Discovering it removes the coupling that exists in AL between understanding

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n°270780.

the task and learning how to fulfill it. IRL allows the use of (A)DP or RL techniques to learn how to do the task from the computed reward function. A very straightforward non-IRL way to do AL is for example to use a multi-class classifier to directly learn the expert policy. We provide in the experiments (Sec. 6) a comparison between AL and IRL algorithms by using IRL as a way to do AL.

A lot of existing approaches in either IRL or IRL-based AL need to repeatedly solve the underlying MDP to find the optimal policies of intermediate reward functions. Thus, their performance depends strongly on the quality of the associated subroutine. Consequently, they suffer from the same challenges of scalability, data scarcity, etc., as RL and (A)DP. In order to avoid *repeatedly* solving such problems, we adopt a different point of view.

Having in mind that there is a one to one relation between a reward function and its associated optimal action-value function (via the Bellman equation, see Eq. (1)), it is worth thinking of a method able to output an action-value function for which the greedy policy is the demonstrated expert policy. Thus, the demonstrated expert policy will be optimal for the corresponding reward function. We propose to use a score function-based multi-class classification step (see Sec. 3) to infer a score function. Besides, in order to retrieve via the Bellman equation the reward associated with the score function computed by the classification step, we introduce a regression step (see Sec. 3). That is why the method is called the Cascaded Supervised Inverse reinforcement learning (CSI). This method is analyzed in Sec. 4, where it is shown that the demonstrated expert policy is near-optimal for the reward the regression step outputs.

This algorithm does not need to iteratively solve an MDP and requires only sampled transitions from expert and non-expert policies as inputs. Moreover, up to the use of some heuristics (see Sec. 6.1), the algorithm is able to be trained only with transitions sampled from the demonstrated expert policy. A specific instantiation of CSI (proposed in Sec. 6.1) is tested on the mountain car problem (Sec. 6.2) and on a highway driving simulator (Sec. 6.3) where we compare it with a pure classification algorithm [20] and with two recent successful IRL methods [5] as well as with a random baseline. Differences and similarities with existing AL or IRL approaches are succinctly discussed in Sec. 5.

2 Background and Notation

First, we introduce some general notation. Let E and F be two non-empty sets, E^F is the set of functions from F to E. We note Δ_X the set of distributions over X. Let $\alpha \in \mathbb{R}^X$ and $\beta \in \mathbb{R}^X$: $\alpha \geq \beta \Leftrightarrow \forall x \in X, \alpha(x) \geq \beta(x)$. We will often slightly abuse the notation and consider (where applicable) most objects as if they were matrices and vectors indexed by the set they operate upon.

We work with finite MDPs [10], that is tuples $\{S, A, P, R, \gamma\}$. The state space is noted S, A is a finite action space, $R \in \mathbb{R}^{S\times A}$ is a reward-function, $\gamma \in (0,1)$ is a discount factor and $P \in \Delta_S^{S\times A}$ is the Markovian dynamics of the MDP. Thus, for each $(s,a) \in S \times A$, $P(.|s,a)$ is a distribution over S and $P(s'|s,a)$

is the probability to reach s' by choosing action a in state s. At each time step t, the agent uses the information encoded in the state $s_t \in S$ in order to choose an action $a_t \in A$ according to a (deterministic[1]) policy $\pi \in A^S$. The agent then steps to a new state $s_{t+1} \in S$ according to the Markovian transition probabilities $P(s_{t+1}|s_t, a_t)$. Given that $P_\pi = (P(s'|s, \pi(s)))_{s,s' \in S}$ is the transition probability matrix, the stationary distribution over the states ρ_π induced by a policy π satisfies $\rho_\pi^T P_\pi = \rho_\pi^T$, with X^T being the transpose of X. The stationary distribution relative to the expert policy π_E is ρ_E.

The reward function R is a local measure of the quality of the control. The global quality of the control induced by a policy π, with respect to a reward R, is assessed by the value function $V_R^\pi \in \mathbb{R}^S$ which associates to each state the expected discounted cumulative reward for following policy π from this state: $V_R^\pi(s) = \mathbf{E}[\sum_{t \geq 0} \gamma^t R(s_t, \pi(s_t))|s_0 = s, \pi]$. This long-term criterion is what is being optimized when solving an MDP. Therefore, an optimal policy π_R^* is a policy whose value function (the optimal value function V_R^*) is greater than that of any other policy, for all states: $\forall \pi, V_R^* \geq V_R^\pi$.

The Bellman evaluation operator $T_R^\pi : \mathbb{R}^S \to \mathbb{R}^S$ is defined by $T_R^\pi V = R_\pi + \gamma P_\pi V$ where $R_\pi = (R(s, \pi(s)))_{s \in S}$. The Bellman optimality operator follows naturally: $T_R^* V = \max_\pi T_R^\pi V$. Both operators are contractions. The fixed point of the Bellman evaluation operator T_R^π is the value function of π with respect to reward R: $V_R^\pi = T_R^\pi V_R^\pi \Leftrightarrow V_R^\pi = R_\pi + \gamma P_\pi V_R^\pi$. The Bellman optimality operator T_R^* also admits a fixed point, the optimal value function V_R^* with respect to reward R.

Another object of interest is the action-value function $Q_R^\pi \in \mathbb{R}^{S \times A}$ that adds a degree of freedom on the choice of the first action, formally defined by $Q_R^\pi(s, a) = T_R^a V_R^\pi(s)$, with a the policy that always returns action a ($T_R^a V = R_a + \gamma P_a V$ with $P_a = (P(s'|s, a))_{s,s' \in S}$ and $R_a = (R(s, a))_{s \in S}$). The value function V_R^π and the action-value function Q_R^π are quite directly related: $\forall s \in S, V_R^\pi(s) = Q_R^\pi(s, \pi(s))$. The Bellman evaluation equation for Q_R^π is therefore:

$$Q_R^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) Q(s', \pi(s')). \tag{1}$$

An optimal policy follows a greedy mechanism with respect to its optimal action-value function Q_R^*:

$$\pi_R^*(s) \in \operatorname*{argmax}_a Q_R^*(s, a). \tag{2}$$

When the state space is too large to allow matrix representations or when the transition probabilities or even the reward function are unknown except through observations gained by interacting with the system, RL or ADP may be used to approximate the optimal control policy [16].

We recall that solving the MDP is the direct problem. This contribution aims at solving the inverse one. We observe trajectories drawn from an expert's deterministic[1] policy π_E, assuming that there exists some unknown reward R_E

[1] We restrict ourselves here to deterministic policies, but the loss of generality is minimal as there exists at least one optimal deterministic policy.

for which the expert is optimal. The suboptimality of the expert is an interesting setting that has been discussed for example in [7,19], but that we are not addressing here. We do not try to find this unknown reward R_E but rather a non trivial reward R for which the expert is at least near-optimal. The trivial reward 0 is a solution to this ill-posed problem (no reward means that every behavior is optimal). Because of its ill-posed nature, this expression of *Inverse Reinforcement Learning* (IRL) still has to find a satisfactory solution although a lot of progress has been made, see Sec. 5.

3 The Cascading Algorithm

Our first step towards a reward function solving the IRL problem is a classification step using a *score function-based multi-class classifier* (SFMC^2 for short). This classifier learns a score function $q \in \mathbb{R}^{S\times A}$ that rates the association of a given action[2] $a \in A$ with a certain input $s \in S$. The classification rule $\pi_C \in A^S$ simply selects (one of) the action(s) that achieves the highest score for the given inputs:

$$\pi_C(s) \in \operatorname*{argmax}_a q(s,a). \tag{3}$$

For example, *Multi-class Support Vector Machines* [4] can be seen as SFMC^2 algorithms, the same can be said of the structured margin approach [20] both of which we consider in the experimental setting. Other algorithms may be envisioned (see Sec. 6.1).

Given a dataset $D_C = \{(s_i, a_i = \pi_E(s_i))_i\}$ of actions a_i (deterministically) chosen by the expert on states s_i, we train such a classifier. The classification policy π_C is not the end product we are looking for (that would be mere supervised imitation of the expert, not IRL). What is of particular interest to us is the score function q itself. One can easily notice the similarity between Eq. (3) and Eq. (2) that describes the relation between the optimal policy in an MDP and its optimal action-value function. The score function q of the classifier can thus be viewed as some kind of optimal action-value function for the classifier policy π_C. By inversing the Bellman equation (1) with q in lieu of Q_R^π, one gets R^C, the reward function relative to our score/action-value function q:

$$R^C(s,a) = q(s,a) - \gamma \sum_{s'} P(s'|s,a) q(s', \pi_C(s')). \tag{4}$$

As we wish to approximately solve the general IRL problem where the transition probabilities P are unknown, our reward function R^C will be approximated with the help of information gathered by interacting with the system. We assume that another dataset $D_R = \{(s_j, a_j, s'_j)_j\}$ is available where s'_j is the state an agent taking action a_j in state s_j transitioned to. Action a_j need not be chosen by any

[2] Here, actions play the role of what is known as *labels* or *categories* when talking about classifiers.

particular policy. The dataset D_R brings us information about the dynamics of the system. From it, we construct datapoints

$$\{\hat{r}_j = q(s_j, a_j) - \gamma q(s'_j, \pi_C(s'_j))\}_j. \tag{5}$$

As s'_j is sampled according to $P(\cdot|s_j, a_j)$ the constructed datapoints help building a good approximation of $R^C(s_j, a_j)$. A regressor (a simple least-square approximator can do but other solutions could also be envisioned, see Sec. 6.1) is then fed the datapoints $((s_j, a_i), \hat{r}_j)$ to obtain $\hat{R}^C$, a generalization of $\{((s_j, a_j), \hat{r}_j)_j\}$ over the whole state-action space. The complete algorithm is given in Alg. 1.

There is no particular constraint on D_C and D_R. Clearly, there is a direct link between various qualities of those two sets (amount of data, statistical representativity, etc.) and the classification and regression errors. The exact nature of the relationship between these quantities depends on which classifier and regressor are chosen. The theoretical analysis of Sec. 4 abstracts itself from the choice of a regressor and a classifier and from the composition of D_C and D_R by reasoning with the classification and regression errors. In Sec. 6, the use of a single dataset to create both D_C and D_R is thoroughly explored.

Algorithm 1. CSI algorithm

```
***Given*** a training set $D_C = \{(s_i, a_i = \pi_E(s_i))\}_{1\leq i\leq D}$ and another training set $D_R = \{(s_j, a_j, s'_j)\}_{1\leq j\leq D'}$
***Train*** a score function-based classifier on $D_C$, obtaining decision rule $\pi_C$ and score function $q : S \times A \to \mathbb{R}$
***Learn*** a reward function $\hat{R}^C$ from the dataset $\{((s_j, a_j), \hat{r}_j)\}_{1\leq j\leq D'}$, $\forall (s_j, a_j, s'_j) \in D_R, \hat{r}_j = q(s_j, a_j) - \gamma q(s'_j, \pi_C(s'_j))$
***Output*** the reward function $\hat{R}^C$
```

Cascading two supervised approaches like we do is a way to inject the MDP structure into the resolution of the problem. Indeed, mere classification only takes into account information from the expert (i.e., which action goes with which state) whereas using the Bellman equation in the expression of $\hat{r}_j$ makes use of the information lying in the transitions (s_j, a_j, s'_j), namely information about the transition probabilities P. The final regression step is a way to generalize this information about P to the whole state-action space in order to have a well-behaved reward function. Being able to alleviate the ill effects of scalability or data scarcity by leveraging the wide range of techniques developed for the classification and regression problems is a strong advantage of the CSI approach.

4 Analysis

In this section, we prove that the deterministic expert policy π_E is near optimal for the reward $\hat{R}^C$ the regression step outputs. More formally, recalling from Sec. 2 that ρ_E is the stationary distribution of the expert policy, we prove that $\mathbf{E}_{s\sim\rho_E}[V^*_{\hat{R}^C}(s) - V^{\pi_E}_{\hat{R}^C}(s)]$ is bounded by a term that depends on:

- the classification error defined as $\epsilon_C = \mathbf{E}_{s\sim\rho_E}[\mathbb{1}_{\{\pi_C(s)\neq\pi_E(s)\}}]$;
- the regression error defined as $\epsilon_R = \max_{\pi\in A^S}\|\epsilon^R_\pi\|_{1,\rho_E}$, with:
  - the subscript notation already used for R_π and P_π in Sec. 2 meaning that, given an $X \in \mathbb{R}^{S\times A}$, $\pi \in A^S$, and $a \in A$, $X_\pi \in \mathbb{R}^S$ and $X_a \in \mathbb{R}^S$ are respectively such that: $\forall s \in S, X_\pi(s) = X(s,\pi(s))$ and $\forall s \in S, X_a(s) = X(s,a)$;
  - $\epsilon^R_\pi = R^C_\pi - \hat{R}^C_\pi$;
  - $\|.\|_{1,\mu}$ the μ-weighted L_1 norm: $\|f\|_{1,\mu} = \mathbf{E}_{x\sim\mu}[|f(x)|]$;
- the concentration coefficient $C_* = C_{\hat{\pi}_C}$ with:
  - $C_\pi = (1-\gamma)\sum_{t\geq 0}\gamma^t c_\pi(t)$, with $c_\pi(t) = \max_{s\in S}\frac{(\rho_E^T P_\pi^t)(s)}{\rho_E(s)}$;
  - $\hat{\pi}_C$, the optimal policy for the reward $\hat{R}^C$ output by the algorithm ;

  The constant C_* can be estimated *a posteriori* (after $\hat{R}^C$ is computed). *A priori*, C_* can be upper-bounded by a more usual and general concentration coefficient but C_* gives a tighter final result: one can informally see C_* as a measure of the similarity between the distributions induced by $\hat{\pi}_C$ and π_E (roughly, if $\hat{\pi}_C \approx \pi_E$ then $C_* \approx 1$).
- $\Delta q = \max_{s\in S}(\max_{a\in A} q(s,a) - \min_{a\in A} q(s,a)) = \max_{s\in S}(q(s,\pi_C(s)) - \min_{a\in A} q(s,a))$, which could be normalized to 1 without loss of generality. The range of variation of q is tied to the one of R^C, $\hat{R}^C$ and $V^{\pi_C}_{\hat{R}^C}$. What matters with these objects is the relative values for different state action couples, not the objective range. They can be shifted and positively scaled without consequence.

Theorem 1. *Let π_E be the deterministic expert policy, ρ_E its stationary distribution and $\hat{R}^C$ the reward the cascading algorithm outputs. We have:*

$$0 \leq \mathbf{E}_{s\sim\rho_E}[V^*_{\hat{R}^C}(s) - V^{\pi_E}_{\hat{R}^C}(s)] \leq \frac{1}{1-\gamma}\left(\epsilon_C \Delta q + \epsilon_R(1+C_*)\right).$$

Proof. First let's recall some notation, $q \in \mathbb{R}^{S\times A}$ is the score function output by the classification step, π_C is a deterministic classifier policy so that $\forall s \in S, \pi_C(s) \in \operatorname{argmax}_{a\in A} q(s,a)$, $R^C \in \mathbb{R}^{S\times A}$ is so that $\forall (s,a) \in S\times A, R^C(s,a) = q(s,a) - \gamma\sum_{s'\in S} P(s'|s,a)q(s',\pi_C(s'))$, and $\hat{R}^C \in \mathbb{R}^{S\times A}$ is the reward function output by the regression step.

The difference between R^C and $\hat{R}^C$ is noted $\epsilon^R = R^C - \hat{R}^C \in \mathbb{R}^{S\times A}$. We also introduce the reward function $\mathfrak{R}^E \in \mathbb{R}^{S\times A}$ which will be useful in our proof, not to be confused with R_E the unknown reward function the expert optimizes:

$$\forall (s,a) \in S\times A, \mathfrak{R}^E(s,a) = q(s,a) - \gamma\sum_{s'\in S} P(s'|s,a)q(s',\pi_E(s')).$$

We now have the following vectorial equalities $R^C_a = q_a - \gamma P_a q_{\pi_C}; \mathfrak{R}^E_a = q_a - \gamma P_a q_{\pi_E}; \epsilon^R_a = R^C_a - \hat{R}^C_a$. Now, we are going to upper bound the term: $\mathbf{E}_{s\sim\rho_E}[V^*_{\hat{R}^C} - V^{\pi_E}_{\hat{R}^C}] \geq 0$ (the lower bound is obvious as V^* is optimal). Recall that $\hat{\pi}_C$ is a deterministic optimal policy of the reward $\hat{R}^C$. First, the term $V^*_{\hat{R}^C} - V^{\pi_E}_{\hat{R}^C}$ is decomposed:

$$V^*_{\hat{R}^C} - V^{\pi_E}_{\hat{R}^C} = (V^{\hat{\pi}_C}_{\hat{R}^C} - V^{\hat{\pi}_C}_{R^C}) + (V^{\hat{\pi}_C}_{R^C} - V^{\pi_E}_{R^C}) + (V^{\pi_E}_{R^C} - V^{\pi_E}_{\hat{R}^C}).$$

We are going to bound each of these three terms. First, let π be a given deterministic policy. We have, using $\epsilon^R = R^C - \hat{R}^C$: $V^\pi_{R^C} - V^\pi_{\hat{R}^C} = V^\pi_{\epsilon^R} = (I - \gamma P_\pi)^{-1}\epsilon^R_\pi$. If $\pi = \pi_E$, we have, thanks to the power series expression of $(I - \gamma P_{\pi_E})^{-1}$, the definition of ρ_E and the definition of the μ-weighted L_1 norm, one property of which is that $\forall X, \mu^T X \leq \|X\|_{1,\mu}$:

$$\rho_E^T(V^\pi_{R^C} - V^\pi_{\hat{R}^C}) = \rho_E^T(I - \gamma P_{\pi_E})^{-1}\epsilon^R_{\pi_E} = \frac{1}{1-\gamma}\rho_E^T\epsilon^R_{\pi_E} \leq \frac{1}{1-\gamma}\|\epsilon^R_{\pi_E}\|_{1,\rho_E}.$$

If $\pi \neq \pi_E$, we use the concentration coefficient C_π. We have then:

$$\rho_E^T(V^\pi_{R^C} - V^\pi_{\hat{R}^C}) \leq \frac{C_\pi}{1-\gamma}\rho_E^T\epsilon^R_\pi \leq \frac{C_\pi}{1-\gamma}\|\epsilon^R_\pi\|_{1,\rho_E}.$$

So, using the notation introduced before we stated the theorem, we are able to give an upper bound to the first and third terms (recall also the notation $C_{\hat{\pi}_C} = C_*$): $\rho_E^T((V^{\pi_E}_{R^C} - V^{\pi_E}_{\hat{R}^C}) + (V^{\hat{\pi}_C}_{\hat{R}^C} - V^{\hat{\pi}_C}_{R^C})) \leq \frac{1+C_*}{1-\gamma}\epsilon_R$. Now, there is still an upper bound to find for the second term. It is possible to decompose it as follows:

$$V^{\hat{\pi}_C}_{R^C} - V^{\pi_E}_{R^C} = (V^{\hat{\pi}_C}_{R^C} - V^{\pi_C}_{R^C}) + (V^{\pi_C}_{R^C} - V^{\pi_E}_{\mathfrak{R}^E}) + (V^{\pi_E}_{\mathfrak{R}^E} - V^{\pi_E}_{R^C}).$$

By construction, π_C is optimal for R^C, so $V^{\hat{\pi}_C}_{R^C} - V^{\pi_C}_{R^C} \leq 0$ which implies:

$$V^{\hat{\pi}_C}_{R^C} - V^{\pi_E}_{R^C} \leq (V^{\pi_C}_{R^C} - V^{\pi_E}_{\mathfrak{R}^E}) + (V^{\pi_E}_{\mathfrak{R}^E} - V^{\pi_E}_{R^C}).$$

By construction, we have $V^{\pi_C}_{R^C} = q_{\pi_C}$ and $V^{\pi_E}_{\mathfrak{R}^E} = q_{\pi_E}$, thus:

$$\begin{aligned}\rho_E^T(V^{\pi_C}_{R^C} - V^{\pi_E}_{\mathfrak{R}^E}) &= \rho_E^T(q_{\pi_C} - q_{\pi_E}) \\ &= \sum_{s \in S}\rho_E(s)(q(s,\pi_C(s)) - q(s,\pi_E(s)))[\mathbb{1}_{\{\pi_C(s)\neq\pi_E(s)\}}].\end{aligned}$$

Using Δq, we have: $\rho_E^T(V^{\pi_C}_{R^C} - V^{\pi_E}_{\mathfrak{R}^E}) \leq \Delta q \sum_{s\in S}\rho_E(s)[\mathbb{1}_{\{\pi_C(s)\neq\pi_E(s)\}}] = \Delta q \epsilon_C$. Finally, we also have:

$$\rho_E^T(V^{\pi_E}_{\mathfrak{R}^E} - V^{\pi_E}_{R^C}) = \rho_E^T(I - \gamma P_{\pi_E})^{-1}(\mathfrak{R}^E_{\pi_E} - R^C_{\pi_E}) = \rho_E^T(I - \gamma P_{\pi_E})^{-1}\gamma P_{\pi_E}(q_{\pi_C} - q_{\pi_E}),$$

$$= \frac{\gamma}{1-\gamma}\rho_E^T(q_{\pi_C} - q_{\pi_E}) \leq \frac{\gamma}{1-\gamma}\Delta q \epsilon_C.$$

So the upper bound for the second term is: $\rho_E^T(V^{\hat{\pi}_C}_{R^C} - V^{\pi_E}_{R^C}) \leq (\Delta q + \frac{\gamma}{1-\gamma}\Delta q)\epsilon_C = \frac{\Delta q}{1-\gamma}\epsilon_C$. If we combine all of the results, we obtain the final bound as stated in the theorem. □

Readers familiar with the work presented in [5] will see some similarities between the theoretical analyses of SCIRL and CSI as both study error propagation in IRL algorithms. Another shared feature is the use of the score function q of the classifier as a proxy for the action-value function of the expert $Q^{\pi_E}_{R_E}$.

The attentive reader, however, will perceive that similarities stop there. The error terms occurring in the two bounds are not related to one another. As CSI makes no use of the feature expectation of the expert, what is known as $\bar{\epsilon}_Q$ in [5] does not appear in this analysis. Likewise, the regression error ϵ_R of this paper does not appear in the analysis of SCIRL, which does not use a regressor. Perhaps more subtly, the classification error and classification policy known in both papers as ϵ_C and π_C are not the same. The classification policy of SCIRL is not tantamount to what is called π_C here. For SCIRL, π_C is the greedy policy for an approximation of the value function of the expert with respect to the reward output by the algorithm. For CSI, π_C is the decision rule of the classifier, an object that is not aware of the structure of the MDP. We shall also mention that the error terms appearing in the CSI bound are more standard than the ones of SCIRL (*e.g.*, regression error vs feature expectation estimation error) thus they may be easier to control. A direct corollary of this theorem is that, given perfect classifier and regressor, CSI produces a reward function for which π_E is the *unique* optimal policy.

Corollary 1. *Assume that $\rho_E > 0$ and that the classifier and the regressor are perfect ($\epsilon_C = 0$ and $\epsilon_R = 0$). Then π_E is the* unique *optimal policy for $\hat{R}^C$.*

Proof. The function q is the optimal action-value function for π_C with respect to the reward R^C, by definition (see Eq. (4)). As $\epsilon_C = 0$, we have $\pi_C = \pi_E$. This means that $\forall s, \pi_E(s)$ is the only element of the set $\text{argmax}_{a \in A}\, q(s, a)$. Therefore, $\pi_C = \pi_E$ is the unique optimal policy for R^C. As $\epsilon_R = 0$, we have $\hat{R}^C = R^C$, hence the result.

This corollary hints at the fact that we found a non-trivial reward (we recall that the null reward admits *every* policy as optimal). Therefore, obtaining $\hat{R}_C = 0$ (for which the bound is obviously true: the bounded term is 0, the bounding term is positive) is unlikely as long as the classifier and the regressor exhibit decent performance.

The only constraints the bound of Th. 1 implies on datasets D_R and D_C is that they provide enough information to the supervised algorithms to keep both error terms ϵ_C and ϵ_R low. In Sec. 6 we deal with a lack of data in dataset D_R. We address the problem with the use of heuristics (Sec. 6.1) in order to show the behavior of the CSI algorithm in somewhat more realistic (but difficult) conditions.

More generally, the error terms ϵ_C and ϵ_R can be reduced by a wise choice for the classification and regression algorithms. The literature is wide enough for methods accommodating most of use cases (lack of data, fast computation, bias/variance trade-off, etc.) to be found. Being able to leverage such common algorithms as multi-class classifiers and regressors is a big advantage of our cascading approach over existing IRL algorithms.

Other differences between existing IRL or apprenticeship learning approaches and the proposed cascading algorithm are further examined in Sec. 5.

5 Related Work

IRL was first introduced in [15] and then formalized in [9]. Approaches summarized in [8] can be seen as iteratively constructing a reward function, solving an MDP at each iteration. Some of these algorithms are IRL algorithms while others fall in the Apprenticeship Learning (AL) category, as for example the projection version of the algorithm in [1]. In both cases the need to solve an MDP at each step may be very demanding, both sample-wise and computationally. CSI being able to output a reward function without having to solve the MDP is thus a significant improvement.

AL via classification has been proposed for example in [12], with the help of a structured margin method. Using the non-trivial notion of metric in an MDP, the authors of [6] build a kernel which is used in a classification algorithm, showing improvements compared to a non-structured kernel.

Classification and IRL have met in the past in [13], but the labels were complete optimal policies rather than actions and the inputs were MDPs, which had to be solved. It may be unclear how SCIRL [5] relates to the proposed approach of his paper. Both algorithms use the score function of a classifier as a proxy to the action-value function of the expert with respect to the (unknown) true reward: $Q_R^{\pi_E}$. The way this proxy is constructed and used, however, fundamentally differs in the two algorithms. This difference will cause the theoretical analysis of both approaches (see Sec. 4) to be distinct. In SCIRL, the score function of the classifier is approximated *via* a linear parametrization that relies on the feature expectation of the expert $\mu^E(s) = E[\sum_{t\geq 0} \gamma^t \phi(s_t) | s_0 = s, \pi_E]$. This entails the use of a specific kind of classifier (namely linearly-parametrized-score-function-based classifiers) and of a method of approximation of μ^E. By contrast, almost any off-the-shelf classifier can be used in the first step of the cascading approach of this paper. The classification step of CSI is unaware of the structure of the MDP whereas SCIRL knows about it thanks to the use of μ^E. In CSI, the structure of the MDP is injected by reversing the Bellman equation prior to the regression step (Eq. 4 and (5)), a step that does not exist in SCIRL as SCIRL directly outputs the parameter vector found by its linearly-parametrized-score-function-based classifier. The regressor of CSI can be chosen off-the-shelf. One can argue that this and not having to approximate μ^E increases the ease-of-use of CSI over SCIRL and makes for a more versatile algorithm. In practice, as seen in Sec. 6, performance of SCIRL and CSI are very close to one another thus CSI may be a better choice as it is easier to deploy. Neither approach is a generalization of the other.

Few IRL or AL algorithms do not require solving an MDP. The approach of [17] requires knowing the transition probabilities of the MDP (which CSI does not need) and outputs a policy (and not a reward). The algorithm in [3] only applies to linearly-solvable MDPs whereas our approach does not place such restrictions. Closer to our use-case is the idea presented in [2] to use a subgradient ascent of a utility function based on the notion of relative entropy. Importance sampling is suggested as a way to avoid solving the MDP. This requires sampling trajectories according to a non-expert policy and the direct problem remains at the core of the approach (even if solving it is avoided).

6 Experiments

In this section, we empirically demonstrate the behavior of our approach. We begin by providing information pertaining to both benchmarks. An explanation about the amount and source of the available data, the rationale behind the heuristics we use to compensate for the dire data scarcity and a quick word about the contenders CSI is compared to are given Sec. 6.1. We supply quantitative results and comparisons of CSI with state-of-the art approaches on first a classical RL benchmark (the mountain car) in Sec. 6.2 and then on a highway driving simulator (Sec. 6.3).

6.1 Generalities

Data Scarcity. The CSI algorithm was designed to avoid repeatedly solving the RL problem. This feature makes it particularly well-suited to environments where sampling the MDP is difficult or costly. In the experiments, CSI is fed only with data sampled according to the expert policy. This corresponds for example to a situation where a costly system can only be controlled by a trained operator as a bad control sequence could lead to a system breakdown.

More precisely, the expert controls the system for M runs of lengths $\{L_i\}_{1\leq i\leq M}$, giving samples $\{(s_k, a_k = \pi_E(a_k), s'_k)_k\} = D_E$. The dataset D_C fed to the classifier is straightforwardly constructed from D_E by dropping the s'_k terms: $D_C = \{(s_i = s_k, a_i = a_k)_i\}$.

Heuristics. It is not reasonable to construct the dataset $D_R = \{((s_k, a_k = \pi_E(s_k)), \hat{r}_k)_k\}$ only from expert transitions and expect a small regression error term ϵ_R. Indeed, the dataset D_E only samples the dynamics induced by the expert's policy and not the whole dynamics of the MDP. This means that for a certain state s_k we only know the corresponding expert action $a_k = \pi_E(s_k)$ and the following state s'_k sampled according to the MDP dynamics : $s'_k \sim P(\cdot|s_k, a_k)$. For the regression to be meaningful, we need samples associating the same state s_k and a different action $a \neq a_k$ with a datapoint $\hat{r} \neq \hat{r}_k$.
Recall that $\hat{r}_j = q(s_j, a_j) - \gamma q(s'_j, \pi_C(s'_j))$ (Eq. (5)); without knowing $s'_k \sim P(\cdot|s_k, a \neq a_k)$, we cannot provide the regressor with a datapoint to associate with $(s_k, a \neq a_k)$. We artificially augment the dataset D_R with samples $((s_j = s_k, a), r_{min})_{j;\forall a\neq\pi_E(s_j)=a_k}$ where $r_{min} = \min_k \hat{r}_k - 1$. This heuristics instructs the regressor to associate a state-action tuple disagreeing with the expert (*i.e.*, $(s_k, a \neq a_k)$) with a reward strictly inferior to any of those associated with expert state action tuples (*i.e.*, $(s_k, a_k = \pi_E(s_k))$). Semantically, we are asserting that disagreeing with the expert in states the expert visited is a bad idea. This heuristics says nothing about states absent from the expert dataset. For such states the generalization capabilities of the regressor and, later on, the exploration of the MDP by an agent optimizing the reward will solve the problem. Although this heuristics was not analyzed in Sec. 4 (where the availability of a more complete dataset D_R was assumed), the results shown in the next two subsections demonstrate its soundness.

Comparison with State-of-the-Art Approaches. The similar looking yet fundamentally different algorithm SCIRL [5] is an obvious choice as a contender to CSI as it advertises the same ability to work with very little data, without repeatedly solving the RL problem. In both experiments we give the exact same data to CSI and SCIRL.

The algorithm of [3] also advertises not having to solve the RL problem, but needs to deal with linearly solvable MDPs, therefore we do not include it in our tests. The Relative Entropy (RE) method of [2] has no such need, so we included it in our benchmarks. It could not, however, work with the small amount of data we provided SCIRL and CSI with, and so to allow for importance sampling, we created another dataset D_{random} that was used by RE but not by SCIRL nor CSI.

Finally, the classification policy π_C output by the classification step of CSI was evaluated as well. Comparing classification and IRL algorithms makes no sense if the object of interest is the reward itself as can be envisioned in a biological or economical context. It is however sound to do so in an imitation context where what matters is the performance of the agent with respect to some objective criterion. Both experiments use such a criterion. Classification algorithms don't have to optimize any reward since the classification policy can directly be used in the environment. IRL algorithms on the other hand output a reward that must then be plugged in an RL or DP algorithm to get a policy. In each benchmark we used the same (benchmark-dependent) algorithm to get a policy from each of the three rewards output by SCIRL, CSI and RE. It is these policies whose performance we show. Finding a policy from a reward is of course a non-trivial problem that should not be swept under the rug; nevertheless we choose not to concern ourselves with it here as we wish to focus on IRL algorithms, not RL or DP algorithms. In this regard, using a classifier that directly outputs a policy may seem a much simpler solution, but we hope that the reader will be convinced that the gap in performance between classification and IRL is worth the trouble of solving the RL problem (once and for all, and not repeatedly as a subroutine like some other IRL algorithms).

We do not compare CSI to other IRL algorithms requiring repeatedly solving the MDP. As we would need to provide them with enough data to do so, the comparison makes little sense.

Supervised Steps. The cascading algorithm can be instantiated with some standard classification algorithms and any regression algorithm. The choice of such subroutines may be dictated by the kind and amount of available data, by ease of use or by computational complexity, for example.

We referred in Sec.3 to *score-function based multi-class classifiers* and explained how the classification rule is similar to the greedy mechanism that exists between an optimal action-value function and an optimal policy in an MDP. Most classifications algorithms can be seen as such a classifier. In a simple k-nearest neighbor approach, for example, the score function $q(s,a)$ is the number of elements of class a among the k-nearest neighbors of s. The generic M-SVM model makes the score function explicit (see [4]) (we use a SVM in the mountain car experiment Sec. 6.2). In the highway experiment, we choose to use a structured margin classification approach [20]. We chose a SVR as a regressor in the mountain car experiment and a simple least-square regressor on the highway.

It is possible to get imaginative in the last step. For example, using a Gaussian process regressor [11] that outputs both expectation and variance can enable (notwithstanding a nontrivial amount of work) the use of reward-uncertain reinforcement learning [14]. Our complete instantiation of CSI is summed up in Alg. 2.

Algorithm 2. A CSI instantiation with heuristics

```
Given a dataset $D_E = (s_k, a_k = \pi_E(a_k), s'_k)_k$
Construct the dataset $D_C = \{(s_i = s_k, a_i = \pi^E(s_i)) = a_k\}$
Train a score function-based classifier on $D_C$, obtaining decision rule $\pi_C$ and score function $q : S \times A \to \mathbb{R}$
Construct the dataset $\{((s_j = s_k, a_j = a_k), \hat{r}_j)_j\}$ with $\hat{r}_j = q(s_j, a_j) - \gamma q(s'_j = s'_k, \pi_C(s'_j = s'_k))$
Set $r_{min} = \min_j \hat{r}_j - 1$.
Construct the training set $D_R = \{((s_j = s_k, a_j = a_k), \hat{r}_j)_j\} \cup \{((s_j = s_k, a), r_{min})_{j; \forall a \neq \pi_E(s_j) = a_k}\}$
Learn a reward function $\hat{R}^C$ from the training set $D_R$
Output the reward function $\hat{R}^C : (s, a) \mapsto \omega^T \phi(s, a)$
```

6.2 Mountain Car

The mountain car is a classical toy problem in RL: an underpowered car is tasked with climbing a steep hill. In order to do so, it has to first move away from the target and climb the slope on its left, and then it moves right, gaining enough momentum to climb the hill on the right on top of which lies the target. We used standard parameters for this problem, as can for example be found in [16]. When training an RL agent, the reward is, for example, 1 if the car's position is greater than 0.5 and 0 anywhere else. The expert policy was a very simple hand crafted policy that uses the power of the car to go in the direction it already moves (*i.e.*, go left when the speed is negative, right when it is positive).

The initial position of the car was uniformly randomly picked in $[-1.2; -0.9]$ and its speed uniformly randomly picked in $[-0.07; 0]$. From this position, the hand-crafted policy was left to play until the car reached the objective (*i.e.*, a position greater than 0.5) at which point the episode ended. Enough episodes were played (and the last one was truncated) so that the dataset D_E contained exactly n samples, with n successively equal to 10, 30, 100 and 300. With these parameters, the expert is always able to reach the top on the first time it tries to climb the hill on the right. Therefore, a whole part of the state space (when the position is on the hill on the right and the speed is negative) is not visited by the expert. This hole about the state space in the data will be dealt with differently by the classifier and the IRL algorithms. The classifier will use its generalization power to find a default action in this part of the state space, while the IRL algorithms will devise a default reward; a (potentially untrained) RL agent finding itself in this part of the state space will use the reward signal to decide what to do, making use of new data available at that time.

In order to get a policy from the rewards given by SCIRL, CSI and RE, the RL problem was solved by LSPI fed with a dataset D_{random} of 1000 episodes of length 5 with a starting point uniformly and randomly chosen in the whole state space and actions picked at random. This dataset was also used by RE (and not by SCIRL nor CSI).

The classifier for CSI was an off-the-shelf SVM[3] which also was the classifier we evaluate, the regressor of CSI was an off-the-shelf SVR[4]. RE and SCIRL need features over the state space, we used the same evenly-spaced hand-tuned 7×7 RBF network for both algorithms.

The objective criterion for success is the number of steps needed to attain the goal when starting from a state picked at random ; the lesser the better. We can see Fig. 1 that the optimal policies for the rewards found by SCIRL and CSI very rapidly attain expert-level performance and outperform the optimal policy for the reward of RE and the classification policy. When very few samples are available, CSI does better than SCIRL (with such a low p-value for $n = 10$, see Tab. 1a, the hypothesis that the mean performance is equal can be rejected); SCIRL catches up when more samples are available. Furthermore, CSI required very little engineering as we cascaded two off-the-shelf implementations whereas SCIRL used hand-tuned features and a custom classifier.

Table 1. Student or Welch test of mean equality (depending on whether a Bartlett test of variance equality succeeds) p-values for CSI and SCIRL on the mountain car (1a) and the highway driving simulator (1b). High values ($> 1.0 \times 10^{-02}$) means that the hypothesis that the means are equal cannot be rejected.

(a) Mountain Car

Number of expert samples	p-value
10	$\mathbf{1.5e-12}$
30	$3.8e-01$
100	$1.3e-02$
300	$7.4e-01$

(b) Highway Driving

Number of expert samples	p-value
9	$3.0e-01$
49	$\mathbf{8.9e-03}$
100	$\mathbf{1.8e-03}$
225	$\mathbf{2.4e-05}$
400	$\mathbf{2.0e-50}$

6.3 Highway Driving Simulator

The setting of the experiment is a driving simulator inspired from a benchmark already used in [17,18]. The agent controls a car that can switch between the three lanes of the road, go off-road on either side and modulate between three speed levels. At all timesteps, there will be one car in one of the three lanes.

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

[4] `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html`

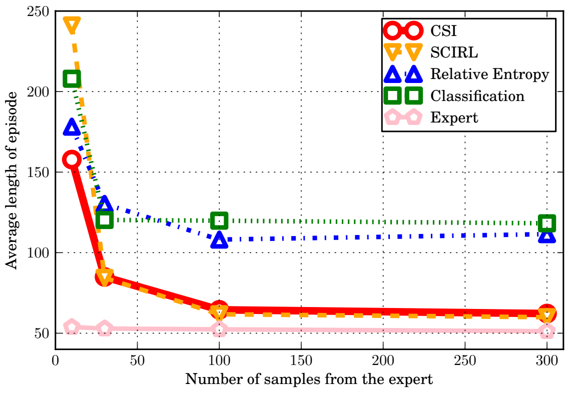
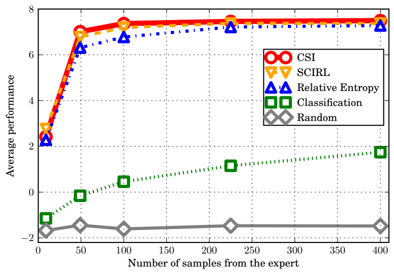
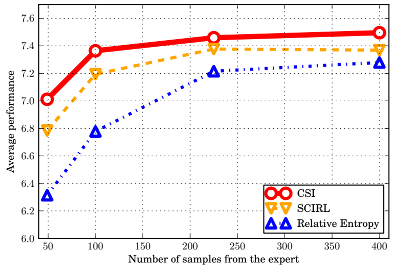


Fig. 1. Performance of various policies on the mountain car problem. This is the mean over 100 runs.





(a) Mean performance over 100 runs on the Highway driving problem.

(b) Zoom of Fig 2a showing the ranking of the three IRL algorithms.

Fig. 2. Results on the highway driving problem

Even at the lowest speed, the player's car moves faster than the others. When the other car disappears at the bottom of the screen, another one appears at the top in a randomly chosen lane. It takes two transitions to completely change lanes, as the player can move left or right for half a lane's length at a time. At the highest speed setting, if the other car appears in the lane the player is in, it is not possible to avoid the collision. The main difference between the original benchmark [17,18] and ours is that we made the problem more ergodic by allowing the player to change speed whenever he wishes so, not just during the first transition. If anything, by adding two actions, we enlarged the state-action space and thus made the problem tougher. The reward function R_E the expert is trained by a DP algorithm on makes it go as fast as possible (high reward) while avoiding collisions (harshly penalized) and avoiding going off-road (moderately penalised). Any other situation receives a null reward.

The performance criterion for a policy π is the mean (over the uniform distribution) value function with respect to R_E : $\mathbf{E}_{s\sim\mathcal{U}}[V^{\pi}_{R_E}(s)]$. Expert performance averages to 7.74 ; we also show the natural random baseline that consists in drawing a random reward vector (with a uniform law) and training an agent on it. The reward functions found by SCIRL, CSI and RE are then optimized using a DP algorithm. The dataset D_{random} needed by RE (and neither by CSI nor SCIRL) is made of 100 episodes of length 10 starting randomly in the state space and following a random policy. the dataset D_E is made of n episodes of length n, with $n \in \{3, 7, 10, 15, 20\}$.

Results are shown Fig. 2. We give the values of $\mathbf{E}_{s\sim\mathcal{U}}[V^{\pi}_{R_E}(s)]$ with π being in turn the optimal policy for the rewards given by SCIRL, CSI and RE, the policy π_C of the classifier (the very one the classification step of CSI outputs), and the optimal policy for a randomly drawn reward. Performance for CSI is slightly but definitively higher than for SCIRL (see the p-values for the mean equality test in Tab. 1b, from 49 samples on), slightly below the performance of the expert itself. Very few samples (100) are needed to reliably achieve expert-level performance.

It is very interesting to compare our algorithm to the behavior of a classifier alone (respectively red and green plots on Fig. 2a). With *the exact same data*, albeit the use of a very simple heuristics, the cascading approach demonstrates far better performance from the start. This is a clear illustration of the fact that using the Bellman equation to construct the data fed to the regressor and outputting not a policy, but a reward function that can be optimized on the MDP truly makes use of the information that the transitions (s, a, s') bear (we recall that the classifier only uses (s, a) couples). Furthermore, the classifier whose results are displayed here is the output of the first step of the algorithm. The classification performance is obviously not that good, which points to the fact that our algorithm may be empirically more forgiving of classification errors than our theoretical bound lets us expect.

7 Conclusion

We have introduced a new way to perform IRL by cascading two supervised approaches. The expert is theoretically shown to be near-optimal for the reward function the proposed algorithm outputs, given small classification and regression errors. Practical examples of classifiers and regressors have been given, and two combinations have been empirically (on two classic benchmarks) shown to be very resilient to dire lack of data on the input (only data from the expert was used to retrieve the reward function), with the help of simple heuristics. On both benchmarks, our algorithm is shown to outperform other state-of-the-art approaches although SCIRL catches up on the mountain car. We plan on deepening the analysis of the theoretical properties of our approach and on applying it to real world robotics problems.

References

1. Abbeel, P., Ng, A.: Apprenticeship learning via inverse reinforcement learning. In: Proc. ICML (2004)
2. Boularias, A., Kober, J.: Peters: Relative entropy inverse reinforcement learning. In: Proc. ICAPS, vol. 15, pp. 20–27 (2011)
3. Dvijotham, K., Todorov, E.: Inverse optimal control with linearly-solvable MDPs. In: Proc. ICML (2010)
4. Guermeur, Y.: A generic model of multi-class support vector machine. International Journal of Intelligent Information and Database Systems (2011)
5. Klein, E., Geist, M., Piot, B., Pietquin, O.: Inverse Reinforcement Learning through Structured Classification. In: Proc. NIPS, Lake Tahoe, NV, USA (December 2012)
6. Melo, F.S., Lopes, M.: Learning from demonstration using MDP induced metrics. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part II. LNCS, vol. 6322, pp. 385–401. Springer, Heidelberg (2010)
7. Melo, F., Lopes, M., Ferreira, R.: Analysis of inverse reinforcement learning with perturbed demonstrations. In: Proc. ECAI, pp. 349–354. IOS Press (2010)
8. Neu, G., Szepesvári, C.: Training parsers by inverse reinforcement learning. Machine Learning 77(2), 303–337 (2009)
9. Ng, A., Russell, S.: Algorithms for inverse reinforcement learning. In: Proc. ICML, pp. 663–670. Morgan Kaufmann Publishers Inc. (2000)
10. Puterman, M.: Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons, Inc., New York (1994)
11. Rasmussen, C., Williams, C.: Gaussian processes for machine learning, vol. 1. MIT press, Cambridge (2006)
12. Ratliff, N., Bagnell, J., Srinivasa, S.: Imitation learning for locomotion and manipulation. In: International Conference on Humanoid Robots, pp. 392–397. IEEE (2007)
13. Ratliff, N., Bagnell, J., Zinkevich, M.: Maximum margin planning. In: Proc. ICML, p. 736. ACM (2006)
14. Regan, K., Boutilier, C.: Robust online optimization of reward-uncertain MDPs. In: Proc. IJCAI 2011 (2011)
15. Russell, S.: Learning agents for uncertain environments (extended abstract). In: Annual Conference on Computational Learning Theory, p. 103. ACM (1998)
16. Sutton, R., Barto, A.: Reinforcement learning. MIT Press (1998)
17. Syed, U., Bowling, M., Schapire, R.: Apprenticeship learning using linear programming. In: Proc. ICML, pp. 1032–1039. ACM (2008)
18. Syed, U., Schapire, R.: A game-theoretic approach to apprenticeship learning. In: Proc. NIPS, vol. 20, pp. 1449–1456 (2008)
19. Syed, U., Schapire, R.: A reduction from apprenticeship learning to classification. In: Proc. NIPS, vol. 24, pp. 2253–2261 (2010)
20. Taskar, B., Chatalbashev, V., Koller, D., Guestrin, C.: Learning structured prediction models: A large margin approach. In: Proc. ICML, p. 903. ACM (2005)