Boosted and Reward-regularized Classification for Apprenticeship Learning

Bilal Piot
Supelec, MaLIS Research group, France
GeorgiaTech-CNRS UMI 2958, France
bilal.piot@supelec.fr

Matthieu Geist
Supelec, MaLIS Research group, France
GeorgiaTech-CNRS UMI 2958, France
matthieu.geist@supelec.fr

Olivier Pietquin
University Lille 1, France
LIFL (UMR 8022 CNRS / Lille 1), SequeL Team, France
olivier.pietquin@univ-lille1.fr

ABSTRACT

This paper deals with the problem of learning from demonstrations, where an agent called the apprentice tries to learn a behavior from demonstrations of another agent called the expert. To address this problem, we place ourselves into the Markov Decision Process (MDP) framework, which is well suited for sequential decision making problems. A way to tackle this problem is to reduce it to classification but doing so we do not take into account the MDP structure. Other methods which take into account the MDP structure need to solve MDPs which is a difficult task and/or need a choice of features which is problem-dependent. The main contribution of the paper is to extend a large margin approach, which is a classification method, by adding a regularization term which takes into account the MDP structure. The derived algorithm, called Reward-regularized Classification for Apprenticeship Learning (RCAL), does not need to solve MDPs. But, the major advantage is that it can be boosted: this avoids the choice of features, which is a drawback of parametric approaches. A state of the art experiment (Highway) and generic experiments (structured Garnets) are conducted to show the performance of RCAL compared to algorithms from the literature.

Categories and Subject Descriptors

G.3 [**PROBABILITY AND STATISTICS**]: Statistical computing

General Terms

Algorithms

Keywords

Learning from Demonstrations, Inverse Reinforcement Learning, Large margin methods, Boosting

1. INTRODUCTION

This paper addresses the problem of Learning from Demonstrations (LfD), where an agent called the apprentice tries to learn a good behavior from demonstrations of another agent called the expert. Using the Markov Decision Process (MDP) framework, which is well suited for sequential decision making, this problem can be cast as Apprenticeship Learning (AL) [1]. In the AL paradigm, the apprentice is observing the expert behaving optimally with respect to an unknown reward function in an MDP. The goal of the apprentice is, via some demonstrations or trajectories of the expert, to learn the expert policy or a policy which is as good as the expert policy relatively to the unknown reward. The first idea, in order to address the AL problem, is to reduce it to a pure classification problem [15, 17, 11, 18, 8]. This approach has been theoretically studied in the finite horizon case [23, 18, 8] and in the infinite horizon case [14]. Pure classification techniques such as the large margin method [17] have a lot of advantages: easy to implement, fast, no need to resolve MDPs, no need to do a choice of features thanks to boosting techniques. However, they do not take into account the structure of the MDP because the temporal structure of the expert trajectories is not used in a pure classification method. To tackle this drawback, the authors of [11] use a kernel-based approach to encapsulate the structure of the MDP into the classifier, which needs the calculation of the MDP metrics and thus the knowledge of the whole dynamics. A different approach to introduce this structure is to allow the apprentice to query the expert in a given state which does not appear in the expert demonstrations (see [18, 8]), but this requirement can be too strong for some real-life applications. The second idea to produce algorithms which attempt to solve the AL problem is inspired by the Inverse Reinforcement Learning (IRL) paradigm. IRL, introduced in [19] and formalized in [13], is closely related to AL. The only difference is that IRL tries to find a reward (that could explain the expert behavior) and not the expert policy. The key idea behind IRL is that the reward may be the most succinct hypothesis explaining the expert behavior. Some algorithms [1, 22, 12] use IRL as an intermediary step to find a policy but other algorithms [3, 9] are "pure" IRL algorithms and output a reward. This reward must then be optimized via a direct Reinforcement Learning algorithm (such as Least Squares Policy Iteration, LSPI [10]) in order to obtain a policy. IRL-inspired methods are interesting because they directly take into account the structure of the MDP by finding a policy (through some reward function) such that some measure of the underlying trajectories distribution matches the one of the expert policy. However, most of them need

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

to solve at least one MDP and are parametric methods: a choice of features must be done by the user. For instance in [1, 3, 9], the authors suppose the existence of $p \in \mathbb{N}^*$ features $(\phi_i)_{1 \leq i \leq p}$ such that the unknown reward function is a linear combination of those features. However, the choice of features is problem-dependent and can become a very hard task for complex problems. In [14], the authors propose a comparative study between classification algorithms and IRL algorithms through a theoretical and empirical points of view. They empirically show that classification algorithms work well when the unknown reward is informative (the optimization horizon is small) which corresponds to a problem where the temporal structure is not important; they show that IRL algorithms work better when the unknown reward is sparse (the optimization horizon is big) which corresponds to a problem where the temporal structure is important. This was an expected result as the classification methods do not take into account the temporal structure of the expert trajectories. Therefore, it would be interesting to adapt a classification algorithm for problems where the temporal structure seems important. The assumption made here is that problems where the temporal aspect is important must be the ones with sparse unknown reward. Our main contribution is to introduce a new algorithm which possesses the advantages of a pure classification method and uses the MDP structure without querying the expert or calculating MDP metrics. We manage to do that by introducing a specific reward-based regularization term, which imposes the unknown reward to be sparse, in a large margin approach. Our algorithm is named Reward-regularization Classification for Apprenticeship Learning (RCAL). We also present a way to use boosting for our algorithm which makes it a non-parametric method. Finally, experiments are conducted on a state-of the art benchmark (the Highway problem) and on a generic task (a new type of structured Garnet problems) to compare the performance of our algorithm against other recent methods.

2. BACKGROUND AND NOTATIONS

First, we introduce general notations. Let $(\mathbb{R}, |.|)$ be the real space with its canonical norm and X a finite set, $\mathbb{R}^X$ is the set of functions from X to $\mathbb{R}$. The set of probability distributions over X is noted Δ_X. Let Y be a finite set, Δ_X^Y is the set of functions from Y to Δ_X. Let $\zeta \in \Delta_X^Y$ and $y \in Y$, $\zeta(y) \in \Delta_X$, the conditional distribution probability knowing y, is also noted $\zeta(.|y)$ and $\forall x \in X, \zeta(x|y) = [\zeta(y)](x)$. Let $\alpha, \beta \in \mathbb{R}^X$: $\alpha \leq \beta \Leftrightarrow \forall x \in X, \alpha(x) \leq \beta(x)$. Let $p \in \mathbb{N}^*$ and $\nu \in \Delta_X$, we define the $\mathbf{L}_{p,\nu}$-norm of α, noted $\|\alpha\|_{p,\nu}$, by: $\|\alpha\|_{p,\nu} = (\sum_{x \in X} |\nu(x)\alpha(x)|^p)^{\frac{1}{p}}$. Let $x \in X$, $x \sim \nu$ means that x is sampled according to ν and $\mathbb{E}_\nu[\alpha] = \sum_{x \in X} \nu(x)\alpha(x)$ is the expectation of α under ν. Finally, $\delta_x \in \mathbb{R}^X$ is the function such that $\forall y \in X$, if $y \neq x$ then $\delta_x(y) = 0$, else $\delta_x(y) = 1$.

2.1 Markov Decisions Processes

In this paper, the expert agent is supposed to act in a finite[1] MDP. It models the interactions of an agent evolving in a dynamic environment and is represented by a tuple $M = \{S, A, R, P, \gamma\}$ where $S = \{s_i\}_{1 \leq i \leq N_S}$ is the state space, $A = \{a_i\}_{1 \leq i \leq N_A}$ is the action space, $R \in \mathbb{R}^{S \times A}$ is the reward function (the local representation of the benefit of doing action a in state s), $\gamma \in]0, 1[$ is a discount factor and $P \in \Delta_S^{S \times A}$ is the Markovian dynamics which gives the probability, $P(s'|s, a)$, to reach s' by choosing the action a in the state s. A Markovian stationary and deterministic policy π is an element of A^S and defines the behavior of an agent. In order to quantify the quality of a policy π relatively to the reward R, we define the value function. For a given MDP $M = \{S, A, R, P, \gamma\}$ and a given policy $\pi \in A^S$, the value function $V_R^\pi \in \mathbb{R}^S$ is defined as $V_R^\pi(s) = \mathbb{E}_s^\pi[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)]$, where $\mathbb{E}_s^\pi$ is the expectation over the distribution of the admissible trajectories $(s_0, a_0, s_1, \dots)$ obtained by executing the policy π starting from $s_0 = s$. Moreover, the function $V_R^* \in \mathbb{R}^S$, defined as $V_R^* = \sup_{\pi \in A^S} V_R^\pi$, is called the optimal value function. A policy $\pi \in A^S$ such that $V_R^\pi = V_R^*$ is said optimal and always exists for a finite-MDP [16][Ch. 6]. A useful tool is, for a given $\pi \in A^S$, the action-value function $Q_R^\pi \in \mathbb{R}^{S \times A}$: $Q_R^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{P(.|s,a)}[V_R^\pi]$. It represents the quality of the agent's behavior if it chooses the action a in the state s and then follows the policy π. Moreover, the function $Q_R^* \in \mathbb{R}^{S \times A}$ defined as: $Q_R^* = \sup_{\pi \in A^S} Q_R^\pi$ is called the optimal action-value function and has the following property: π^* is an optimal policy relatively to the reward R if and only if $\pi^*(s) \in \operatorname{argmax}_{a \in A} Q_R^*(s, a)$ [13]. The link between Q_R^π and V_R^π is $Q_R^\pi(s, \pi(s)) = V_R^\pi(s)$ and the link between Q_R^* and V_R^* is $\max_{a \in A} Q_R^*(s, a) = V_R^*(s)$ [16]. Thus we have $\forall s \in S, \forall a \in A$:

$$R(s, a) = Q_R^*(s, a) - \gamma \sum_{s' \in S} P(s'|s, a) \max_{a \in A} Q_R^*(s', a). \quad (1)$$

Eq. (1) links the reward R to the optimal action-value function Q_R^*, it is an interesting equation: knowing Q_R^* it is straightforward to obtain R by a simple calculation. This equation will be used in Sec. 3 in order to impose the unknown reward to be sparse.

2.2 Learning from Demonstrations

In this paper, the expert agent is supposed to act optimally (with respect to the unknown reward function) in a finite MDP and the apprentice can only observe the expert policy π_E via sampled transitions of π_E which means that the apprentice is not able to query the expert in a given state, contrary to [18, 8] for example. Moreover, we suppose that the apprentice has some information about the dynamics which he could have collected by previous interactions with the MDP. So, we consider that the data we have come from a batch setting. In this setting, the aim of the apprentice is to find a policy π_A which is as good as the expert policy with respect to the unknown reward.

More precisely, we suppose that we have a fixed dataset of expert sampled transitions $D_E = (s_i, \pi'_E(s_i), s'_i)_{\{1 \leq i \leq N_E\}}$ where $s_i \sim \nu_E \in \Delta_S$ and $s'_i \sim P(.|s_i, \pi_E(s_i))$. In addition, we suppose that the apprentice has some information about the dynamics via a fixed dataset of sampled transitions $D_P = (s_j, a_j, s'_j)_{\{1 \leq j \leq N_P\}}$ where $s_j \sim \nu_P \in \Delta_S$ and $s'_j \sim P(.|s_j, a_j)$. We have $D_E \subset D_P$ and no particular assumptions are made considering the choice of the action a_j or the distributions ν_E and ν_P which can be considered unknown. Those requirements (used for example in [9, 3]) are not strong and can be fulfilled by an important number of real-life applications.

[1]This work could be easily extended to compact state spaces; we choose the finite case for the ease and clarity of exposition.

One can argue that having a dataset of sampled transitions D_P is a strong assumption. However, our algorithm (see Sec.5) can be run with $D_P = D_E$. In addition, in this batch setting, IRL algorithms, which output a reward function, need the set D_P in order to approximately resolve the MDP (*i.e.* find the optimal policy corresponding to the outputted reward function) because online optimization is not possible. In our experiments (see Sec. 5), the batch optimization algorithm used is LSPI and do not often obtain good results when $D_P = D_E$ for IRL algorithms. This batch setting is interesting because it allows us comparing classification algorithms and IRL algorithms with the same amount of data and see which one is able to provide a policy that can imitate the expert.

2.3 MultiClass Classification and the Large Margin Approach

To tackle the problem of LfD, it is possible to reduce it to a Multi-Class Classification (MCC) problem [15, 17, 18, 23]. The goal of MCC is, given a training set $D = (x_i \in X, y_i \in Y)_{\{1 \leq i \leq N\}}$ where X is a compact set of inputs and Y a finite set of labels, to find a decision rule $g \in Y^X$ that generalizes the relation between inputs and labels. In [17], the authors use a large margin approach which is a score-based MCC where the decision rule $g \in Y^X$ is obtained via a score function $q \in \mathbb{R}^{X \times Y}$ such that $\forall x \in X, g(x) \in \operatorname{argmax}_{y \in Y} q(x, y)$. The large margin approach consists, given the training set D, in solving the following optimization problem:

$$q^* = \operatorname*{argmin}_{q \in \mathbb{R}^{X \times Y}} J(q), \tag{2}$$

$$J(q) = \frac{1}{N} \sum_{i=1}^{N} \max_{y \in Y} \{q(x_i, y) + l(x_i, y_i, y)\} - q(x_i, y_i),$$

where $l \in \mathbb{R}_+^{X \times Y \times Y}$ is called the margin function. If this function is zero, minimizing $J(q)$ attempts to find a score function q^* for which the example labels are scored higher than all other labels. Choosing a nonzero margin function improves generalization [17]. Instead of requiring only that the example label is scored higher than all other labels, we require it to be better than each label y by an amount given by the margin function. Applying the large margin approach to the LfD problem is straightforward. From the set of expert trajectories D_E, we extract the set of expert state-action couples $\tilde{D}_E = (s_i, \pi_E(s_i))_{\{1 \leq i \leq N_E \in \mathbb{N}^*\}}$ and we try to solve:

$$q^* = \operatorname*{argmin}_{q \in \mathbb{R}^{S \times A}} J(q) \tag{3}$$

$$J(q) = \frac{1}{N_E} \sum_{i=1}^{N_E} \max_{a \in A} \{q(s_i, a) + l(s_i, \pi_E(s_i), a)\} - q(s_i, \pi_E(s_i)).$$

The policy outputted by this algorithm would be $\pi_A(s) \in \operatorname{argmax}_{a \in A} \hat{q}(s, a)$ where $\hat{q}$ is the output of the minimization. The advantages of this method are its simplicity and the possibility to use a boosting technique [17] to solve the optimization problem given by Eq. (3). However, this is a pure classification technique which does not take into account the dynamics information contained in the sets D_E and D_P. In the following section, we derive an algorithm from the large margin approach which uses the dynamics information by introducing an original regularization term in $J(q)$. We will also propose a boosting method [7] to resolve the new optimization problem and by doing so we obtain a non-parametric algorithm.

3. REGULARIZED CLASSIFICATION FOR APPRENTICESHIP LEARNING

The optimization problem given by Eq. (2) tries to approximate a function from possibly sparse data. This is, in general, an ill-posed problem and a way to solve it is the regularization theory [24] which adds a regularization term that can be seen as a new constraint on the sought function. In [6], the authors show how the work of Vapnik [25] set the foundations for a general theory which justifies regularization in order to learn from sparse data. Indeed, the basic idea of Vapnik's theory is that the search for the best approximating function from sparse data must be constrained to a *small* hypothesis space. If the hypothesis space is too large, we will find functions that exactly fit the data but they will have a poor generalization capability (overfitting). In [6], the authors show that the choice of the regularization parameter λ corresponds to the choice of an hypothesis space: if λ is *small* the hypothesis space is large and vice versa. A general way to introduce regularization is to consider the following optimization problem:

$$q^* = \operatorname*{argmin}_{q \in \mathbb{R}^{S \times A}} J_W(q) = \operatorname*{argmin}_{q \in \mathbb{R}^{S \times A}} \left(J(q) + \lambda W(q)\right).$$

where $\lambda \in \mathbb{R}_+^*$ and W is a continuous function from $\mathbb{R}^{S \times A}$ to $\mathbb{R}_+$. We would like to introduce the information about the structure contained in the dataset D_P and the fact that we are assuming that the unknown reward function is sparse in the regularization term $W(q)$. The regularization allows searching the function q^* in a more constrained hypothesis space and reduces the variance of the method. A way to do it is to remark that we search for a score function which verifies: $\forall s \in S, \pi_E(s) \in \operatorname{argmax}_{a \in A} q(s, a)$, which means that there exists a reward function $R_q \in \mathbb{R}^{S \times A}$ for which π_E is optimal and such that $q(s, a) = Q_R^*(s, a)$ (see Sec. 2). This reward is given via the inverse Bellman equation (Eq. (1)) $\forall s \in S, \forall a \in A$:

$$R_q(s, a) = q(s, a) - \gamma \sum_{s' \in S} P(s'|s, a) \max_{a \in A} q(s', a).$$

The reward R_q must be sparse in our problem, so a natural choice for $W(q)$ would be $\|R_q\|_{1,\nu_P}$ where $\nu_P \in \Delta_{S \times A}$ is the distribution from where the data are generated. But, as we do not have $R_q(s, a)$ (as $P(.|s, a)$ is unknown) we will rather consider, for each transition in D_P, the unbiased estimate of $R_q(s_j, a_j)$ noted $\hat{R}_q(j)$ which is easy to obtain from the data:

$$\hat{R}_q(j) = q(s_j, a_j) - \gamma \max_{a \in A} q(s'_j, a).$$

Thus, in order to introduce the dynamics information contained in D_P, we choose as regularization term W:

$$W(q) = \frac{1}{N_P} \sum_{j=1}^{N_P} |\hat{R}_q(j)| = \frac{1}{N_P} \sum_{j=1}^{N_P} |q(s_j, a_j) - \gamma \max_{a \in A} q(s'_j, a)|,$$

As $W(q)$ is not an unbiased estimate of $\|R_q\|_{\nu_P,1}$, one can argue that W_q does not favor spareness such as $\|R_q\|_{\nu_P,1}$. However, we have:

$$W(q) = \sum_{(s,a) \in S \times A} \hat{\nu}_P(s, a) \sum_{s' \in S} \hat{P}(s'|s, a) \left| q(s, a) - \gamma \max_{a \in A} q(s', a) \right|,$$

where $\hat{\nu}_P$ and $\hat{P}(.|s, a)$ converges to ν_P and $P(.|s, a)$. By the

Jensen inequality:

$$W(q) \geq \sum_{(s,a)\in S\times A} \hat{\nu}_P(s,a) \left| q(s,a) - \sum_{s'\in S} \hat{P}(s'|s,a)\gamma \max_{a\in A} q(s',a) \right|,$$

$$\underset{N_P\to\infty}{\to} \|R_q\|_{\nu_P,p}.$$

Therefore, the criterion W_q is even stronger than $\|R_q\|_{\nu_P,1}$, which means that (asymptotically) the function q with a small W_q will have also a small $\|R_q\|_{\nu_P,1}$.

Our algorithm consists in solving the following optimization problem:

$$q^* = \underset{q\in\mathbb{R}^{S\times A}}{\operatorname{argmin}} J_W(q)$$

$$J_W(q) = J(q) + \frac{\lambda}{N_P}\sum_{j=1}^{N_P} |\hat{R}_q(j)|.$$

Then, the policy outputted by RCAL would be $\pi_A(s) \in \operatorname{argmax}_{a\in A} \hat{q}(s,a)$ where $\hat{q}$ is the output of the minimization of J_W. This criterion is not convex in the variable q but it is a perturbed convex criterion because it is composed of the term $J(q)$ which is convex in q and of the term $\frac{\lambda}{N_P}\sum_{j=1}^{N_P}|\hat{R}_q(j)|$ which is the regularization term. The regularization term is not convex in q, but it is Lipschitz. In order to minimize this criterion, we use a boosting technique which is suited to a convex criterion but which can be directly adapted to a non-convex one. However, there is no guarantee of converging to a global minimum.

4. BOOSTING RCAL

A boosting method is an interesting optimization technique: it minimizes directly the criterion without the step of choosing features, which is one of the major drawback of several Apprenticeship Learning methods. As presented in [7], a boosting algorithm is a projected sub-gradient descent [20] of a convex functional in a specific functions space (here $\mathbb{R}^{S\times A}$) which has to be a Hilbert space. The principle is to minimize a convex functional $L \in \mathbb{R}^H$ where H is a Hilbert space: $\min_{h\in H} L(h)$. This technique can be extended to non-smooth and non-convex functionals, yet the functional has to be Lipschitz in order to guarantee that the gradient of the functional exists almost everywhere [4]. For a Lipschitz and non smooth functional, the gradient can be calculated almost everywhere and if not the notion of generalized gradient is used (see [4] for details). To realize this minimization, we need to calculate the gradient $\partial_h L$ and define $K \subset H$ a set of allowable directions (also called the restriction set) where the gradient is projected. Boosting algorithms use a projection step when optimizing over function space because the functions representing the gradient are often computationally difficult to manipulate and do not generalize well to new inputs [7]. In boosting literature, the restriction set corresponds directly to the set of hypotheses generated by a weak learner. The nearest direction k^*, which is the projection of the gradient $\partial_h L$, is defined by:

$$k^* = \underset{k\in K}{\operatorname{argmax}} \frac{\langle k, \partial_h L\rangle}{\|k\|},$$

where $\langle .,. \rangle$ is the inner product associated to the Hilbert space H and $\|.\|$ is the associated canonical norm. Then, the naive algorithm to realize the minimization of L is given by Algo. 1. More sophisticated boosting algorithms and their convergence proofs are presented in [7]. However the naive approach is sufficient to obtain good results for the AL problem as it is shown in [7]. For our specific problem, $H = \mathbb{R}^{S\times A}$, $L = J_W$ and:

Algorithm 1 Naive boosting algorithm

Require: $h_0 \in \mathbb{R}^H$, $i = 0$, $T \in \mathbb{N}^*$ (number of iterations) and $(\xi_j)_{\{j\in\mathbb{N}\}}$ a family of learning rates.
1: While $i < T$ do
2: Calculate $\partial_{h_i} L(h_i)$.
3: Calculate k^* associated to $\partial_{h_i} L(h_i)$ (projection step).
4: $h_{i+1} = h_i - \xi_i k^*$
5: $i = i + 1$
6: end While, output h_T

$$\langle q_1, q_2\rangle = \sum_{s\in S}\sum_{a\in A} q_1(s,a)q_2(s,a).$$

Moreover, in our experiments, we choose the restriction set K to be weighted classification trees from $\mathbb{R}^{S\times A}$ to $\{-1,1\}$ which is a particular choice of weak learners. Calculating $\partial_q J_W$ for a given $q \in \mathbb{R}^{S\times A}$ gives:

$$\partial_q \max_{a\in A}\{q(s_i,a) + l(s_i,\pi_E(s_i),a)\} = \delta_{(s_i,a_i^*)},$$
$$\partial_q q(s_i,\pi_E(s_i)) = \delta_{(s_i,\pi_E(s_i))},$$
$$\partial_q|\hat{R}_q(j)| = \operatorname{sgn}(\hat{R}_q(j))(\delta_{(s_j,a_j)} - \gamma\delta_{(s'_j,a_j^*)}),$$
$$\partial_q J_W = \frac{1}{N_E}\sum_{1\leq i\leq N_E} \delta_{(s_i,a_i^*)} - \delta_{(s_i,\pi_E(s_i))} + \frac{\lambda}{N_P}\sum_{1\leq j\leq N_P} \partial_q|\hat{R}_q(j)|.$$

where $a_i^* = \operatorname{argmax}_{a\in A}[q(s_i,a_i) + l(s_i,\pi_E(s_i),a)]$ and $a_j^* = \operatorname{argmax}_{a\in A} q(s'_j,a)$. Obtaining k^* associated to $\partial_q J_W$ when K is the set of classification trees from $\mathbb{R}^{S\times A}$ to $\{-1,1\}$ is done as follows. First, we calculate $\langle k, \partial_q J_W\rangle$:

$$\langle k, \partial_q J_W\rangle = \frac{1}{N_E}\sum_{i=1}^{N_E} k(s_i,a_i^*) - k(s_i,\pi_E(s_i)) + \frac{\lambda}{N_P}\sum_{j=1}^{j=N_P} \operatorname{sgn}(\hat{R}_q(j))(k(s_j,a_j) - \gamma k(s'_j,a_j^*)). \quad (4)$$

To maximize $\langle k, \partial_q J_W\rangle$, we have to find a classifier k such that $k(s_i,a_i^*)$,$k(s_j,a_j) = \operatorname{sgn}(\hat{R}_q(j))$, $k(s_i,\pi_E(s_i)) = -1$ and $k(s'_j,a_j^*) = -\operatorname{sgn}(\hat{R}_q(j))$ for a maximum of inputs while taking into consideration the weight factors for each input in Eq. (4). Thus, in order to obtain k^*, we train a weighted classification tree with the following training set:

$$D_C = (((s_i,\pi_E(s_i)),w_E,-1)\cup((s_i,a_i^*),w_E,1))_{1\leq i\leq N_E} \cup \Big(((s'_j,a_j^*),\gamma w_P,-\operatorname{sgn}(\hat{R}_q(j)))\cup((s_j,a_j),w_P,\operatorname{sgn}(\hat{R}_q(j)))\Big)_{1\leq j\leq N_P}$$

where $w_E = 1$, $w_P = \frac{\lambda N_E}{N_P}$ are weight factors and where an element of a data training set of a weighted classification tree as the following form: (x,w,o) where x is the input, w the weight and o is the output. So, RCAL with boosting can be summarized by the Algo. 2. The output q_T is a weighted

Algorithm 2 RCAL with boosting

Require: $q_0 \equiv 0$, $i = 0$, $T \in \mathbb{N}^*$ and $(\xi_j)_{\{j \in \mathbb{N}\}}$ a family of learning rates.

While $i < T$ do
Calculate $\partial_{q_i} J_W(q_i)$.
Train a classifier with D_C and obtain k^*.
$q_{i+1} = q_i - \xi_i k^*$, $i = i + 1$
end While, output q_T

addition of T classification trees. Those T trees can be seen as the features of the problem which are automatically found by the boosting algorithm.

5. EXPERIMENTS

This section shows different empirical results which tend to corroborate the fact that our algorithm has a good performance. The first experiment is conducted on a state of the art benchmark which is the Highway problem. The second experiment is a more generic one where RCAL is tested on structured Garnets which generalize the results obtained for the Highway. In our experiments, we compare RCAL with two IRL algorithms (Structured Classification IRL (SCIRL) [9] and Relative Entropy (RE) [3]) that use only sampled trajectories, and with two pure classification algorithms (a large margin algorithm [17] called Boosted Classification which corresponds to RCAL with $\lambda = 0$ and a standard classification tree only for the structured Garnets experiment). The regularization parameter λ is fixed at 0.1, the learning rates are $\xi_i = \frac{1}{i+1}, i \in \mathbb{N}$ and the discount factor is $\gamma = 0.99$ in all of our experiments. Finally, the margin function $l(s, a_1, a_2)$ is such that $l(s_i, \pi_E(s_i), \pi_E(s_i)) = 0$ and $l(s, a_1, a_2) = 1$ elsewhere.

5.1 A classical benchmark: The Highway

This problem is used as benchmark in the IRL literature [1, 21, 9]. We use the same car driving simulator as in [9]. In this problem the goal is to drive a car on a busy three-lane highway with randomly generated traffic. The car can move left and right, accelerate, decelerate and keep a constant speed (5 action-features). The expert optimizes a handcrafted reward R which favorises speed, punishes off-road, punishes even more collisions and is neutral otherwise. This reward is quite sparse. We have 729 state-features corresponding to: 9 horizontal positions for the car, 3 horizontal and 9 vertical positions for the closest traffic car and 3 possible speeds. Those features are used for the IRL algorithms and LSPI which allows us to obtain a policy from the reward functions outputted by the IRL algorithms. RCAL does not need those features as it is a boosted algorithm and we fix the number T of classification trees (weak learners) to 30. Here is the protocol of the first experiment where $D_P = D_E$. We compute π_E via the policy iteration algorithm as the dynamics P and the reward R of the car driving simulator are known (but unknown for the algorithm user). Then, we collect the data for the experiment. The evolving parameter chosen is the length of the expert trajectory (H_E). The collect of data consists in choosing a given number of expert trajectories (K_E) of a given size (H_E) and see the performance of the algorithm in mean on such type of data. Here the mean is realized on 100 expert data sets collected randomly for a given H_E and K_E. Thus, we construct 100 expert data sets $D_P^k = D_E^k = (\omega_j)_{\{1 \leq j \leq K_E\}}$ with $K_E = 5$ where $\omega_j = (s_{i,j}, a_{i,j}, s'_{i,j})_{\{1 \leq i \leq H_E\}}$ is a trajectory obtained by starting from a random state $s_{1,j}$ (chosen uniformly) of size H_E where $s'_{i,j} \sim P(.|s_{i,j}, \pi_E(s_{i,j}))$. Then, we choose as criterion of performance:

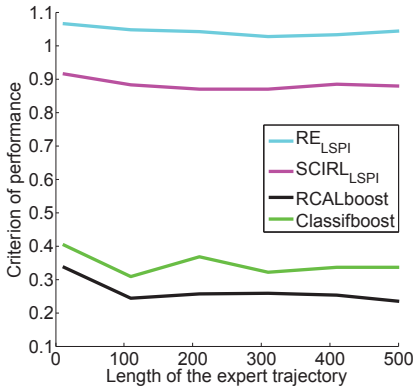
$$T^k(H_E) = \frac{\mathbb{E}_\rho[V_R^{\pi_E} - V_R^{\pi_A^k}]}{\mathbb{E}_\rho[V_R^{\pi_E}]},$$

where π_A^k is the policy obtain by a given algorithm fed by the set $D_E^k = D_P^k$ and ρ is a uniform distribution over the state space. For IRL algorithms, the real output is a reward R^k, then in order to obtain π_A^k the algorithm LSPI is fed by $\hat{D}_P^k = (s, a, s', R^k(s, a))_{(s,a,s') \in D_P^k}$. Finally for a given H_E, the mean performance is: $T(H_E) = \frac{1}{100} \sum_{1 \leq n,k \leq 100} T^k(H_E)$. We plot $(H_E, T(H_E))$ in Fig. 1(a).

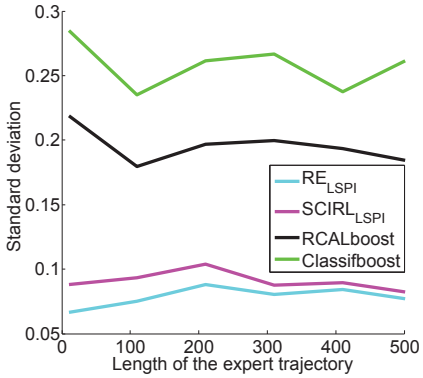
Another criterion is also useful in order to interpret the results. We calculate the standard deviation $\text{std}(H_E)$:

$$\text{std}(H_E) = \left\{ \frac{1}{100} \sum_{1 \leq i \leq 100} [T^k(H_E) - T(H_E)]^2 \right\}^{\frac{1}{2}}.$$

And we plot $(\text{std}(H_E), T(H_E))$ in 1(b). The first important



(a) Algorithms Performance



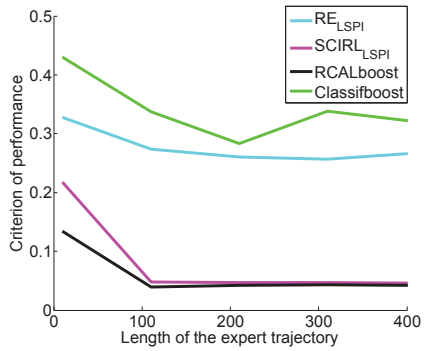
(b) Standard deviation

Figure 1: The Highway experiment with $D_P = D_E$.

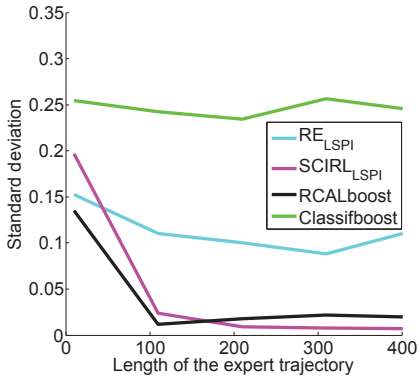
remark is that LSPI does not manage to converge because the amount of non expert data is not sufficient. Thus, in this particular batch setting, IRL algorithm are not efficient. The second remark is that RCAL has a better performance

than the Boosted Classification algorithm which shows that the regularization works. It seems also that RCAL has a better standard deviation than the Boosted Classification.

The second experiment consists in adding some information about the dynamics of the MDP via another fixed set of transitions. This experiment is exactly the same as the first one, except that to each set D_E^k we add a second set D_R^k in order to obtain the set $D_P^k = D_R^k \cup D_E^k$. D_R^k is composed of 100 random trajectories of size 10. The data set D_R^k is such that: $D_R^k = (\omega_j)_{\{1 \leq j \leq K_R\}}$ with $K_R = 100$ where $\omega_j = (s_{i,j}, a_{i,j}, s'_{i,j})_{\{1 \leq i \leq H_R\}}$ is a trajectory obtained by starting from a random state $s_{1,j}$ (chosen uniformly) of size $H_R = 10$ where $s'_{i,j} \sim P(.|s_{i,j}, \pi_R(s_{i,j}))$ and π_R is the random policy. Then, we calculate, as in the previous experiment, $T(H_E)$ and std(H_E) for each algorithm. We plot $(H_E, T(H_E))$ in Fig. 2(a) and $(\text{std}(H_E), T(H_E))$ in 2(b). When, we add some random trajectories LSPI manages to converge to a good policy. We see that RCAL and SCIRL have quite the same performance and the same standard deviation even if RCAL manages to attain faster the best performance. Here, we see that the regularization term helps a lot because there is a big gap between RCAL and the Boosted Classification. An important remark is that RCAL manages to have the same performance than SCIRL with only 30 weak classifiers (which can be seen as features) whereas SCIRL uses $729 * 5 = 3645$ features. In the next experiment, we construct a generic framework to see if the results observed in this particular problem can be extended to a general class of problems of the same kind.



(a) Algorithms Performance



(b) Standard deviation

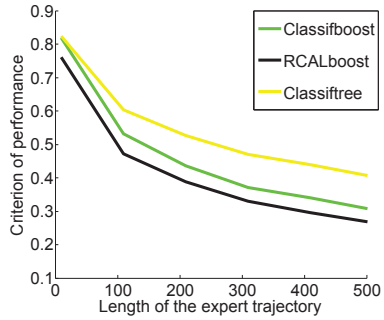
Figure 2: Highway with random trajectories.

5.2 Structured Garnet Problems

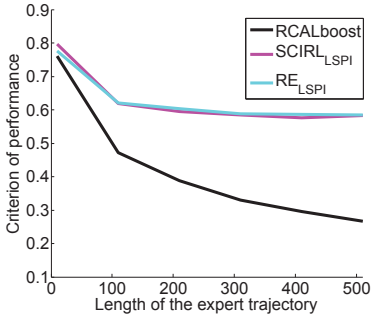
This experiment focuses on stationary Garnet problems, which are a class of randomly constructed finite MDPs representative of the kind of finite MDPs that might be encountered in practice [2]. A stationary Garnet problem is characterized by 3 parameters: $Garnet(N_S, N_A, N_B)$. The parameters N_S and N_A are the number of states and actions respectively, and N_B is a branching factor specifying the number of next states for each state action pair. In this experiment, we choose a particular type of Garnets which presents a topological structure relative to real dynamical systems. Those systems are generally multi-dimensional state spaces MDPs where an action leads to different next states close to each other. The fact that an action leads to close next states can model the noise in a real system for instance. Thus, problems such as the highway simulator [9], the mountain car or the inverted pendulum (possibly discretized) are particular cases of this type of Garnets. For those particular Garnets, the state space is composed of d dimensions (d is chosen uniformly between 2 and 5) and each dimension i has a finite number of elements x_i (chosen uniformly between 1 and 10). So, a state $s = [s^1, s^2, .., s^i, .., s^d]$ is a d-uple where each composent s^i can take a finite value between 2 and x_i. In addition, the distance between two states s, s' is $\|s - s'\|^2 = \sum_{i=1}^{i=d}(s^i - s'^i)^2$. Thus, we obtain MDPs with a possible state space size of 10^5. The number of actions is chosen uniformly between 1 and 10. For each state action couple (s, a), we choose randomly a state s' and N_B next states (N_B is chosen uniformly between 2 and 10) are chosen via a Gaussian distribution of d dimensions centered in s' to select states close from s'. The probability of going to each next state is generated by partitioning the unit interval at $N_B - 1$ cut points selected randomly. We construct a sparse reward R by choosing $\frac{N_S}{10}$ states (uniform random choice without replacement) where $R(s, a) = 1$, elsewhere $R(s, a) = 0$. For each Garnet problem, it is possible to compute an expert policy π_E via the policy iteration algorithm. For IRL algorithms and LSPI, a choice of features must be done. This choice is realized via an Approximate Linearly Dependency (ALD) method with a Gaussian kernel [5]. It consists in choosing as features, an approximately linear dependent set of gaussian functions centered in the samples (see [5] for details). In our experiment, we decide to choose as features, for a given $D_P = (s_j, a_j, s'_j)_{\{1 \leq j \leq N\}}$, all the gaussian functions $(\phi_j \in \mathbb{R}^{S \times A})_{\{1 \leq j \leq N\}}$ such that $\forall s, \forall a$:

$$\phi_j(s, a) = exp(-\|s - s_j\|^2)\delta_{a_j}(a).$$

In the first experiment, we show that RCAL has good performance even when $D_E = D_P$. The experiment consists in generating 100 Garnets $(G_n)_{\{1 \leq n \leq 100\}}$, where π_E^n is the expert policy. For each G_n, we construct 100 expert trajectories $D_E^{n,k} = (s_i, \pi_E^n(s_i), s'_i)_{\{1 \leq i \leq H_E\}}$ of size H_E where $s'_i \sim P(.|s_i, \pi_E^n(s_i))$. Then, we choose as criterion of performance: $T^{n,k}(H) = \frac{\mathbb{E}_{\rho^n}[V_R^{\pi_E^n} - V_R^{\pi_A^{n,k}}]}{\mathbb{E}_{\rho^n}[V_R^{\pi_E^n}]}$, where $\pi_A^{n,k}$ is the policy obtain by a given algorithm fed by the set $D_E^{n,k}$ and ρ^n is a uniform distribution over the state space. Then for a given H_E, the mean performance is: $T(H_E) = 10^{-4} \sum_{1 \leq n,k \leq 100} T^{n,k}(H)$. We plot the performances of classification algorithms in Fig. 3(a) and the performances of IRL algorithms in Fig. 3(b). We observe, in Figs. 3(a) and 3(b), that RCAL has the best performance. Moreover, even when $D_E = D_P$, RCAL manages

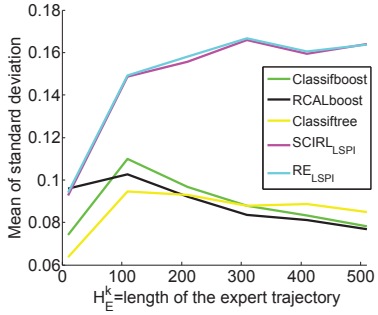


(a) RCAL vs Classification

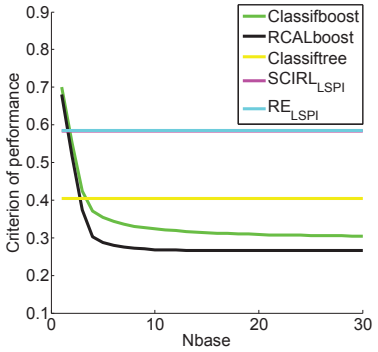


(b) RCAL vs IRL

Figure 3: The Garnets experiment $D_P = D_E$.



(a) Standard deviation

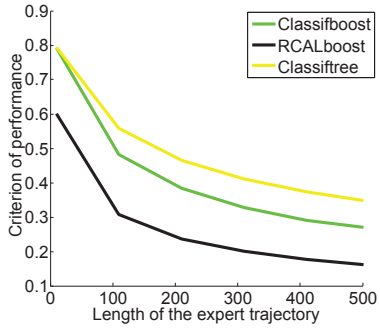


(b) Boosting Performance

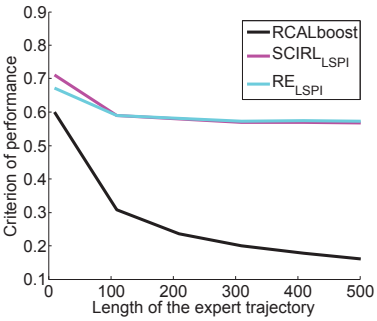
Figure 4: The Garnets experiment $D_P = D_E$.

to have a slight advantage compared to the Boosted Classification (it exploits the transition information). In Fig. 3(b), SCIRL and RE do not provide good results because LSPI does not converge depending on the size of the problem or because the choice of features is not adapted to the problem. That is why, the standard deviation of IRL algorithms (Fig. 4(a)) is important because on some problems LSPI works (small size problem) and on others LSPI fails (large scale problems). Finally, on Fig. 4(b), we plot the mean performance as a function of the number of weak learners (The number of weak learners is noted Nbase in the figures). Here, only 10 weak learners are necessary to obtain a good performance for RCAL.

In the second experiment, we show how RCAL manages to do even better when a set of non-expert transitions are added to D_E. To each set $D_E^{p,k}$, we had 100 trajectories of size 10 of a random policy. Then we plot the new performances in Figs. 5(a) and 5(b). We observe, that RCAL is clearly



(a) RCAL vs Classification



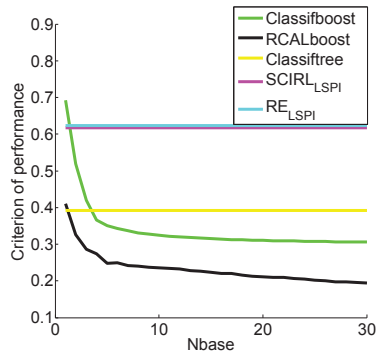
(b) RCAL vs IRL

Figure 5: The second Garnets experiment.

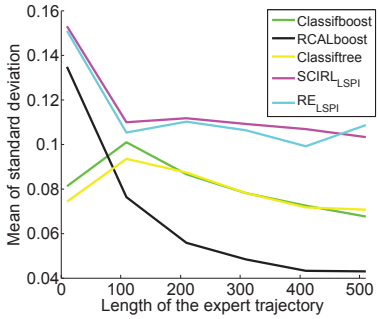
better than others algorithms: the gap between RCAL and the Boosted Classification is important. Besides, RCAL has a low standard deviation (see Fig. 6(b)) which makes it a very stable algorithm. In Fig. 6(a), we see that we can attain even better performance if we decide to let the number of weak classifiers be greater than 30.

6. CONCLUSION

RCAL is a large margin algorithm with a regularization term that imposes a certain sparseness of the unknown reward function. Thanks to this regularization term, RCAL is able to take into account the structure of the MDP contrary to [17] and without querying the expert like [18, 8] or calculating the MDP metrics like [11]. Those requirements can be too strong for real life applications where querying the ex-



(a) Boosting Performance



(b) Standard deviation

Figure 6: The second Garnets experiment.

pert is not possible because only batch samples of the expert are available and where the dynamics is not provided for the calculus of the MDP metrics. Moreover, RCAL, contrary to most IRL-inspired methods, does not solve MDPs, which is a difficult problem for large state or action spaces. Besides, thanks to boosting, RCAL becomes a non-parametric algorithm and the choice of features is not needed contrary to [1, 3, 9] for example. The choice of features is often an engineering problem for a given real-life situation and a bad choice can lead to very bad performances. In the experiments, we see that RCAL offers better performances than IRL and classification algorithms as it is able to use the MDP structure and the boosting which are great advantages of each domain. The performance of RCAL can be improved when non-experts trajectories are collected, however this is not mandatory. In future works, we want to apply RCAL to real-life applications and study the theoretical performance of the algorithm.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement nř270780.

8. REFERENCES

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. of ICML*, 2004.

[2] T. Archibald, K. McKinnon, and L. Thomas. On the generation of markov decision processes. *Journal of the Operational Research Society*, 1995.

[3] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proc. of AISTATS*, 2011.

[4] F. H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205:247–262, 1975.

[5] Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In *Proc. of ECML*, 2002.

[6] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[7] A. Grubb and J. Bagnell. Generalized boosting algorithms for convex optimization. In *Proc. of ICML*, 2011.

[8] K. Judah, A. Fern, and T. Dietterich. Active imitation learning via reduction to iid active learning. In *Proc. of UAI*, 2012.

[9] E. Klein, M. Geist, B. Piot, and O. Pietquin. Inverse reinforcement learning through structured classification. In *Proc. of NIPS*, 2012.

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

[11] F. Melo and M. Lopes. Learning from demonstration using mdp induced metrics. In *Proc. of ECML*, 2010.

[12] G. Neu and C. Szepesvári. Training parsers by inverse reinforcement learning. *Machine learning*, 77(2), 2009.

[13] A. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *Proc. of ICML*, 2000.

[14] B. Piot, M. Geist, and O. Pietquin. Learning from demonstrations: Is it worth estimating a reward function? In *Proc. of ECML*, 2013.

[15] D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document, 1989.

[16] M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

[17] N. Ratliff, J. Bagnell, and S. Srinivasa. Imitation learning for locomotion and manipulation. In *Proc. of IEEE-RAS International Conference on Humanoid Robots*, 2007.

[18] S. Ross and J. Bagnell. Efficient reductions for imitation learning. In *Proc. of AISTATS*, 2010.

[19] S. Russell. Learning agents for uncertain environments. In *Proc. of COLT*, 1998.

[20] N. Shor, K. Kiwiel, and A. Ruszcaynski. *Minimization methods for non-differentiable functions*. Springer-Verlag, 1985.

[21] U. Syed, M. Bowling, and R. Schapire. Apprenticeship learning using linear programming. In *Proc. of ICML*, 2008.

[22] U. Syed and R. Schapire. A game-theoretic approach to apprenticeship learning. In *Proc. of NIPS*, 2008.

[23] U. Syed and R. Schapire. A reduction from apprenticeship learning to classification. In *Proc. of NIPS*, 2010.

[24] A. Tikhonov and V. Arsenin. *Methods for solving ill-posed problems*, volume 15. Nauka, Moscow, 1979.

[25] V. Vapnik. *Statistical learning theory*. Wiley, 1998.