Classifier-Based Policy Representation*

Ioannis Rexakis and Michail G. Lagoudakis
Intelligent Systems Laboratory
Department of Electronic and Computer Engineering
Technical University of Crete
Chania 73100, Crete, Greece
{yr,lagoudakis}@intelligence.tuc.gr

Abstract

Motivated by recent proposals that view a reinforcement learning problem as a collection of classification problems, we investigate various aspects of policy representation using classifiers. In particular, we derive optimal policies for two standard reinforcement learning domains (inverted pendulum and mountain car) in both deterministic and stochastic versions and we examine their internal structure. We then proceed in an evaluation of the representational ability of a variety of classifiers for these policies, using both a multi-class and a binary formulation of the classification problem. Finally, we evaluate the actual performance of the policies learned by the classifiers in the original control problem as a function of the amount of training examples provided. Our results offer significant insight in making the reinforcement-learning-via-classification technology successfully applicable to hard learning problems.

1 Introduction

Recent studies in reinforcement learning have investigated the potential of using classification technology for advancing reinforcement learning technology [7, 4, 5, 8, 3]. The main idea lies on the fact that (deterministic) action policies over the state space of a Markov Decision Process (MDP) can be approximately represented using (multi-class) classifiers; each action is viewed as a distinct class and the states are the instances to be classified. Such classifiers can be learned using appropriate training data sets that hint at the desired action choice over a finite set of states. Through this view, it is possible to incorporate classification algorithms within the inner loops of several reinforcement learning algorithms. The most attractive benefit of this approach is the elimination of value function representation which can be difficult even to approximate. While it is known for a long time [1] that it is easier to represent a policy, rather than a value function, the structure of the (optimal) policies for various common learning problems has not been studied in depth.

Our work aims at investigating policy representation using classifiers in depth. This paper summarizes our initial experience with classifier-based policy representation. In particular, we focus on two well-known learning domains and we uncover the structure present in the optimal policies for each domain. We then investigate the ability of various classifiers in accurately representing these policies, using perfectly correct training data. The apparent limitations of multi-class classification with tied actions led us in adopting a representation scheme which employs multiple binary classifiers, one for each action. Despite the requirement for manually resolving ties between actions, this new scheme offers much better representational accuracy. It is not surprising that best results were obtained using Nearest Neighbors classifiers, reflecting the fact that a high degree of locality is present in the structure of policies. Besides testing the representational abilities of each classifier in terms of classification accuracy, we also evaluated the quality of all learned policies in terms of their performance in the original control problem. Surprisingly, high classification accuracy of the classifier does not necessarily guarantee high performance of the corresponding policy. These findings suggest that careful study is required in choosing an appropriate classifier for a given learning problem.

The remainder of the paper is organized as follows. Section 2 provides the necessary background and Section 3 reviews reinforcement learning as classification. Subsequently, the domains we focus on are presented in Section 4 and the structure hidden in the corresponding optimal policies is uncovered in Section 5. Finally, Section 6 includes experimental results on representing these optimal policies using a variety of classifiers and training schemes.

*This project was partially supported by the Marie Curie International Reintegration Grant MCIRG-CT-2006-044980 awarded to Michail G. Lagoudakis within the 6th European Framework Programme.

2 Background

A *Markov Decision Process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is a finite set of actions, P is a Markovian transition model ($P(s, a, s')$ denotes the probability of a transition to state s' when taking action a in state s), R is a reward function ($R(s, a)$ is the expected reward for taking action a in state s), $\gamma \in [0, 1)$ is the discount factor for future rewards, and D is the initial state distribution. A *deterministic policy* π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice at state s. The value $V_\pi(s)$ of a state s under a policy π is the expected, total, discounted reward when the process begins in state s and all decisions at all steps are made according to π:

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \Big| s_0 = s\right] .$$

The goal of the decision maker is to find an optimal policy π^* that maximizes the expected, total, discounted reward from the initial state distribution D:

$$\pi^* = \arg\max_\pi E_{s\sim D; a_t \sim \pi; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \Big| s_0 = s\right] .$$

It is well-known that for every MDP, there exists at least one optimal deterministic policy. Value iteration, policy iteration, and linear programming are well-known methods for deriving an optimal policy given the full MDP model.

In reinforcement learning, the learner interacts with a process modeled as an MDP and typically observes the state of the process and the immediate reward at every step, however P and R are not accessible. The goal is to gradually learn an optimal policy using the experience collected through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r, thus experience comes in the form of (s, a, r, s') tuples. In many cases, it is further assumed that the learner has the ability to reset the process to any arbitrary state s. This amounts to having access to a generative model of the process (a simulator) from where the learner can draw arbitrarily many times a next state s' and a reward r for performing any given action a in any given state s. Several algorithms have been proposed for learning good or even optimal policies [9].

3 Reinforcement Learning as Classification

A key distinction among reinforcement learning algorithms relies on whether they are based on representing value functions, policies, or both. Value function based algorithms have received much criticism in recent years due to difficulties associated with the estimation and representation of value functions. Many learning problems of interest exhibit non-linear and non-smooth value functions that can be hardly compactly represented. Therefore, the necessary compromises in approximating a value function for the sake of feasibility may lead to non-convergence or convergence to a highly suboptimal policy. On the other hand, advocates of direct policy learning have employed parametric representations that differ little from their value function representation counterparts. Most of them rely on representations of stochastic policies which take the form of a softmax over a parametric real-valued function, which is not much different than a value function after all.

A deterministic policy typically suffices for optimal behavior in any MDP, therefore there is no fundamental reason in considering stochastic policies during learning. Intuitively, a deterministic policy for any given MDP is a less complex structure than a value function for the same MDP. The first only needs an integer label per state, assuming a finite, discrete action space, whereas the second needs a real-valued number per state whose relative magnitude does matter. Little attention has been given in comparative studies between policies and value functions, yet such studies confirm the intuition that policies are easier to represent than value functions [1].

A recent trend in policy learning [7, 4, 5, 8, 3] attempts to exploit the generalization abilities of modern classification technology under the assumption that optimal or good deterministic policies for real learning problems are not arbitrarily complex, but rather exhibit a high degree of structure. It should, therefore, be plausible to learn a good policy using only a small set of training data consisting of selected states over the state space labeled with the actions that are deemed to be best in those states.

To illustrate the learning process under such representations, we briefly review the seminal *Rollout Classification Policy Iteration* (RCPI) algorithm of Lagoudakis and Parr [7]. A similar algorithm was introduced by Fern et al. [4] at around the same time, however with a focus on discrete relational domains. The key idea behind RCPI is to cast the problem of policy learning as a classification problem. Finding a good policy becomes equivalent to finding a classifier that maps states to "good" actions, where the goodness of an action is measured in terms of its contribution to the long term goal of the agent. The state-action value function Q_π

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V_\pi(s') ,$$

provides such a measure given a fixed base policy π; the action that maximizes Q_π in state s is a "good" action in that state, whereas any action with smaller value of Q_π is a

"bad" one. The policy π' formed in this manner

$$\pi'(s) = \arg\max_a Q_\pi(s, a)$$

is guaranteed to be at least as good as π if not better. A training set for π' could be easily formed if the Q_π values for all actions were available for a subset of states. The Monte-Carlo estimation technique of *rollouts* provides a way of accurately estimating Q_π at any given state-action pair (s, a) without requiring an explicit representation of the value function. A rollout for (s, a) amounts to simulating a trajectory of the process beginning from state s, choosing action a for the first step and actions according to the policy π thereafter up to a certain horizon T, and computing the total discounted reward along this trajectory. The observed total discounted reward is averaged over a number of rollouts to yield an accurate estimate of $Q_\pi(s, a)$. Thus, using a sufficient amount of rollouts it is possible to create a training example of the improved policy in state s. A collection of such examples over a finite set of samples forms a valid training set for the improved policy over any base policy.

The goal of the learner is not only to improve a policy, but rather to find an optimal policy, therefore RCPI employs an approximate policy iteration scheme as described in Algorithm 1. At each iteration, a new policy/classifier is produced using training data obtained by rolling out the previous policy on a generative model of the process. Beginning with any initial policy π_0, a training set over a subset of states S_R is formed by querying the rollout procedure to identify dominating actions over the states in S_R. Notice that the training set contains both positive and negative examples for each state, where a clear domination is found. A new classifier is trained using these examples to yield an approximate representation of the improved policy over the previous one. This cycle is then repeated until a termination condition is met. Given the approximate nature of this policy iteration, the termination condition cannot rely on convergence to a single optimal policy. Rather, it terminates when the performance of the new policy (measured via simulation) does not exceed that of the previous policy.

The RCPI algorithm yielded promising results in the pendulum and the bicycle domains using Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) as classifiers. The algorithm of Fern et al. yielded satisfying results in seven planning domains mainly from the AIPS-2000 planning competition using Decision Lists as the underlying classifier. Both studies have identified sensitivities related to the distribution of states in S_R, however, none of them have examined closely the nature and the complexity of the resulting classification problem. What is the structure of the policies learned? Which family of classifiers is appropriate for representing policies? How many training examples suffice for good performance? Our work takes a first step at answering these questions by examining in detail classifier-based representation of policies for typical reinforcement learning domains, factoring out all issues related to estimation using rollouts and focusing solely on the classification problem itself. To this end, we first derive optimal policies for these domains, we examine their structure, and then we try to reconstruct them using a variety of classifiers; the resulting classifiers are tested for accuracy and performance of the corresponding policy. The only related work we are aware of examines the structure of policies for a continuous action pendulum and the performance of policies represented using classifiers for selecting the appropriate Gaussian process for control [2].

Algorithm 1 Rollout Classification Policy Iteration

Input: rollout states S_R, initial policy π_0, iterations K, horizon T, discount factor γ

$\pi' = \pi_0$ (default: uniformly random)
repeat
  $\pi = \pi'$
  TrainingSet $= \varnothing$
  for (each $s \in S_R$) **do**
    for (each $a \in \mathcal{A}$) **do**
      estimate $Q_\pi(s, a)$ using K rollouts of length T
    end for
    if (a dominating action a^* exists in s) **then**
      TrainingSet = TrainingSet $\cup \{(s, a^*)^+\}$
      TrainingSet = TrainingSet $\cup \{(s, a)^-\}, \forall\, a \neq a^*$
    end if
  end for
  π' = TRAINCLASSIFIER(TrainingSet)
until ($\pi \approx \pi'$)
return π

4 Benchmark Domains

We chose to study classifier-based policy representation in the context of two standard domains in reinforcement learning: inverted pendulum and mountain car. Both problems are defined on two-dimensional continuous state spaces and a small action space, therefore they are appropriate for visualization and inspection. For both problems we study a deterministic (without noise) and a stochastic (with noise) version.

4.1 Inverted Pendulum

The *inverted pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to (Figure 1, left). Three actions are allowed: left force LF (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons).

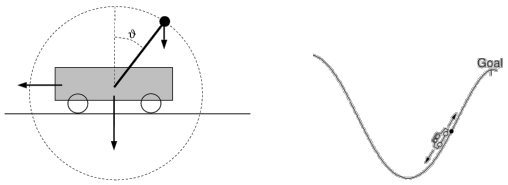

Figure 1. Inverted pendulum, mountain car.

In the stochastic version of the problem, all three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. There is no noise in the deterministic version. The state space of the problem is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system [10] and depend on the current state and the current (noisy or noiseless) control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)} \ ,$$

where g is the gravity constant ($g = 9.8m/s^2$), m is the mass of the pendulum (default: $m = 2.0$ kg), M is the mass of the cart (default: $M = 8.0$ kg), l is the length of the pendulum (default: $l = 0.5$ m), and $\alpha = 1/(m + M)$. The simulation step is 0.1 seconds, thus the control input is given at a rate of 10 Hz, at the beginning of each time step, and is kept constant during any time step. A reward of 0 is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ in absolute value signals the end of the episode and a reward (penalty) of -1. The discount factor of the process is set to 0.95.

4.2 Mountain Car

The *mountain car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top of the mountain on the right (Figure 1, right). The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build some energy by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT (+1), reverse throttle RT (-1), or no throttle NT (0). In the deterministic version of the problem, as originally specified [9], there is no noise. In the stochastic version, to make the problem a little more challenging, we have added noise to all three actions; Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. The state space of the problem is continuous and consists of the position x and the velocity $\dot{x}$ of the car along the horizontal axis. The transitions are governed by the simplified nonlinear dynamics of the system [9] and depend on the current state $(x(t), \dot{x}(t))$ and the current (noisy or noiseless) control $u(t)$:

$$\begin{aligned} x(t+1) &= \textsc{Bound}_x[x(t) + \dot{x}(t+1)] \\ \dot{x}(t+1) &= \textsc{Bound}_{\dot{x}}[\dot{x}(t) + 0.001u(t) - 0.0025\cos(3x(t))] \ , \end{aligned}$$

where $\textsc{Bound}_x$ is a function that keeps x within $[-1.2, 0.5]$, while $\textsc{Bound}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. If the car hits the bounds of the position x, the velocity $\dot{x}$ is set to zero. A penalty of -1 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound of position, it has reached the goal; the episode ends successfully and a reward of 0 is given. The discount factor of the process is set to 0.99.

5 Optimal Policy Structure

Our first goal is to uncover the structure in an optimal policy for each problem. Even though the model of the underlying MDP is known, applying an exact algorithm for solving the MDP and obtaining a trully optimal policy is infeasible due to the continuous nature of the state space. Instead, we used a fine discretization of the 2-dimensional state space into a uniform grid, a large amount of (s, a, r, s') samples at each discrete tile (s, a), and the Least-Squares Policy Iteration (LSPI) algorithm [6] with indicator basis functions over the state grid and all actions, to converge to a near-optimal policy. Notice that the only source of suboptimality in this procedure is the resolution of the discretization itself, as well as the amount of samples collected at each point; there is no error due to approximation of the value function or the policy. The error due to discretization cannot be avoided, however the error due to sampling could be practically eliminated by a large number of samples. Alternatively, one could analytically determine the transition model over the state grid, that is, the possible next states and the related transition probabilities at each tile of the grid. Despite the theoretical feasibility of such a difficult task, we chose to use a sampling approach instead, in order to accommodate any kind of change in the system (different levels of control noise, simulation step, discretization resolution, parameter values, etc.) without changes. Finally, in deriving the optimal policy from the resulting optimal value function, the action values were compared within an ϵ-margin to eliminate small numerical errors. The settings we used were: a grid of 250×250 tiles with 25 uniform samples per tile for each action in the deterministic versions, a grid of 250×250 tiles with 500 uniform samples per tile for each action in the stochastic versions, and $\epsilon = 10^{-5}$.

5.1 Inverted Pendulum

The optimal policy for the stochastic inverted pendulum problem over the 2-dimensional state space is shown in Fig-

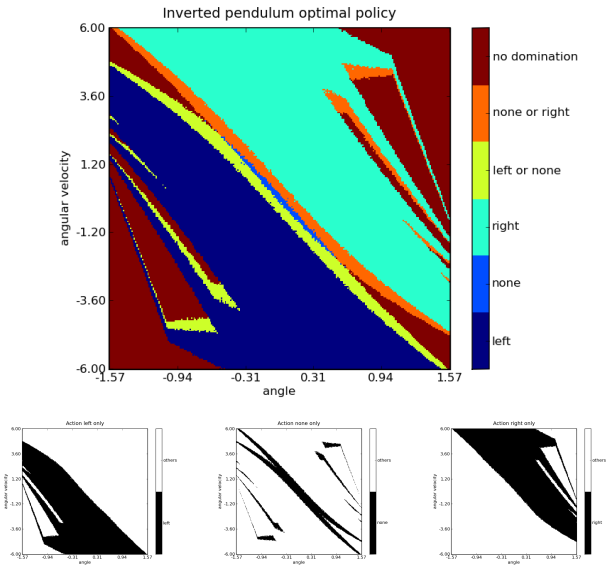

Figure 2. Optimal policy for the stochastic inverted pendulum. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

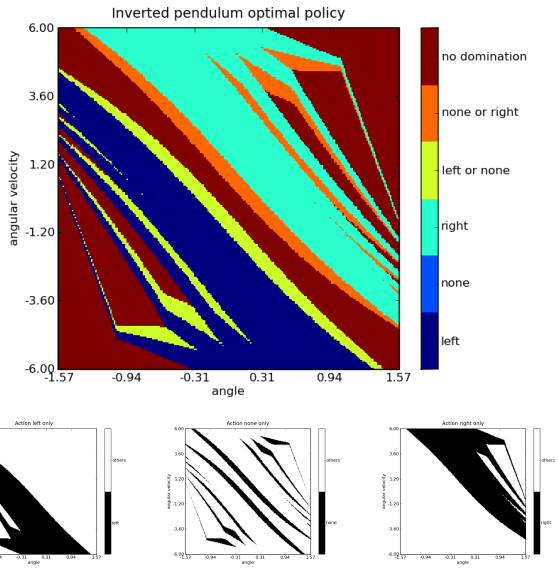

Figure 3. Optimal policy for the deterministic inverted pendulum. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

ure 2. The horizontal axis is the angle θ of the pendulum ranging from $-\pi/2$ to $\pi/2$ and the vertical axis is the angular velocity $\dot{\theta}$ ranging from -6 to 6. The point of full balance is the point $(0, 0)$ in the middle of the state space. One can clearly identify large areas where a single action consists the best choice. In general, as soon as the angle θ becomes positive enough, the best action choice is to apply a right force. This is also true when the angular velocity $\dot{\theta}$ takes on positive values, even though the angle θ itself might be negative; a right force will proactively prevent the pendulum from falling on the right side. Of course, beyond some values of θ and $\dot{\theta}$ any action is hopeless; the pendulum is doomed to falling down. Since the problem is symmetric, similar action choices appear on the left side of the state space. Notice that there is a small area around the balancing point where the best action to perform is to apply no force at all, and a small zone around this area where the left or no force actions are equally good and the right or no force actions are equally good. As expected, there are no areas where the left and right force actions are equally good. Apparently, the optimal policy bears sufficient structure to facilitate the classification problem. Especially, the critical area around the balancing point poses no difficulties in classifying the states correctly to the corresponding actions. The thin stripes on the left and right sides are not intuitive and may be challenging from a classification viewpoint, however a good enough policy will most likely prevent the pendulum from ever reaching those areas, therefore the loss from misclassification in those areas will be minimal.

The optimal policy for the deterministic inverted pendulum problem over the 2-dimensional state space is shown in Figure 3. Surprisingly, this policy exhibits richer structure. The critical area around the balancing point is wider and somewhat more relaxed, but there are more and thinner stripes and many more ties between the left/none actions and the right/none actions. As expected there are no ties between the left/right actions. One would expect a simpler policy for a simpler problem, but apparently the optimal policy for the deterministic version of the problem poses a somewhat more challenging classification problem. We conjecture that the richer and more complex structure is in fact an artifact of the deterministic nature of the problem, the discretization resolution, the time step of the simulation, and the boundary conditions.

5.2 Mountain Car

The optimal policy for the stochastic mountain car problem over the 2-dimensional state space is shown in Figure 4. The horizontal axis is the position x of the car ranging from -1.2 to 0.5 and the vertical axis is the velocity $\dot{x}$ ranging from -0.07 to 0.07. The point $(-0.52, 0)$ corresponds to the bottom of the valley when the car is not moving. Again, one can clearly identify large areas where a single action consists the best choice. In general, the key decision seems

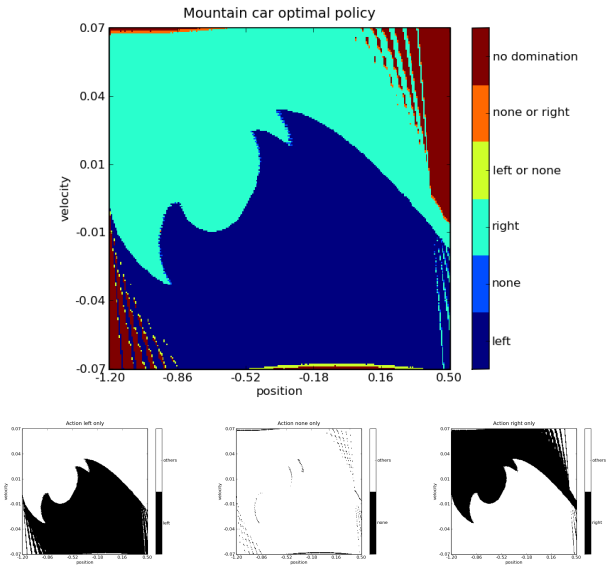

Figure 4. Optimal policy for the stochastic mountain car. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

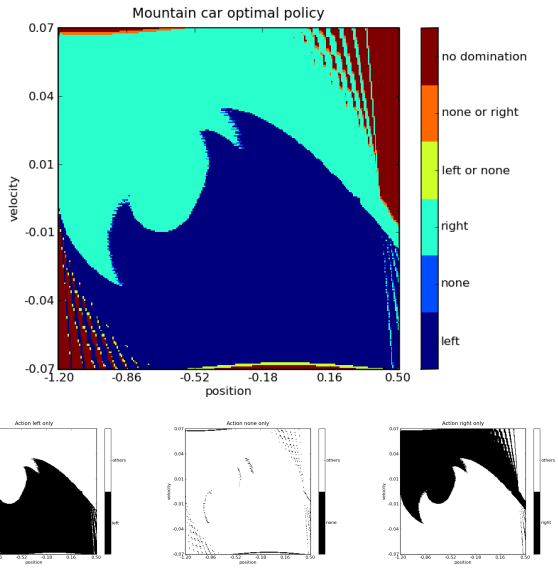

Figure 5. Optimal policy for the deterministic mountain car. Top: dominating action loci including tied cases. Bottom: individual (non-pure) domination locus for each action.

to be to choose the action that gives more thrust in direction the car is currently moving; forward/right throttle for positive velocity and reverse/left throttle for negative velocity. Apparently, such a policy can help the car build the necessary energy for exiting the valley. Note that the no throttle action barely constitutes a best action choice in any state, which is somewhat expected. As expected, there are no areas where the left and right force actions are equally good. At bottom-left and top-right extremes all actions are indifferent, corresponding to the cases where the car is going to exit the valley or hit the left barrier anyway independently of the action choice. Again, the optimal policy bears sufficient structure to facilitate the classification problem. The critical area around the bottom of the valley point poses no particular difficulties in classifying the states correctly to the corresponding actions.

The optimal policy for the deterministic mountain car problem over the 2-dimensional state space is shown in Figure 5. This policy does not differ significantly from the optimal one for the stochastic version, unlike the pendulum policies.

6 Optimal Policy Approximation

Our next goal is to compare a variety of classifiers in terms of their ability to represent the optimal policies identified in the previous section. To this end, we have used classifiers from five different families (our specific classifier choices from each family are shown in parenthesis using the name they are known with): Nearest Neighbors (IB1), Decision Trees (J48-C4.5), Support Vector Machines (LibSVM), Neural Networks (LVQ3), Naive Bayes (Bayes). In all cases, we adopted the following experimental methodology. The optimal policies provide a comprehensive labeled data set of size 250×250 over the entire state space. These $62,500$ training examples are randomly shuffled to eliminate any ordering bias and classifiers are trained and tested using a 4-fold cross-validation procedure. Implementation, training, and testing of these classifiers were automated using the WEKA software package [11] with the required plugins for LibSVM and LVQ3.

The resulting scores for the accuracy of the various classifiers under a multi-class formulation are shown in Table 1. In this case, the learned policy is represented by a single multi-class classifier. Best accuracy in each case (column) is marked in bold and the classifiers are listed (approximately) in rows of decreasing accuracy. Clearly, the Nearest Neighbors classifier (IB1) and the Decision Tree classifier (J48-C4.5) exhibit superior accuracy. The policies of the pendulum problem seem to be harder to represent, whereas policies from the mountain car problem are more accurately represented. This difference can be confirmed visually by the structure of the corresponding policies. There is a significant accuracy deterioration from the stochastic to the deterministic version of the pendulum problem, as expected and discussed in the previous section; no significant deteri-

Table 1. Multi-Class Formulation Accuracy

	Pendulum		Mountain Car	
Classifier	Stoch.	Determ.	Stoch.	Determ.
IB1	**95,76%**	**94,64%**	97,00%	96,53%
J48-C4.5	94,54%	91,71%	**97,57%**	**97,26%**
LibSVM	84,46%	76,81%	92,39%	91,73%
LVQ3	75,73%	60,61%	89,22%	86,31%
Bayes	57,98%	48,59%	80,99%	80,04%

Table 2. Binary Formulation Accuracy

	Pendulum		Mountain Car	
Classifier	Stoch.	Determ.	Stoch.	Determ.
IB1	**98,18%**	**97,73%**	**98,18%**	**97,70%**
J48-C4.5	96,05%	94,42%	97,88%	97,37%
LibSVM	96,43%	90,84%	91,79%	91,43%
LVQ3	89,49%	85,65%	90,74%	90,84%
Bayes	70,80%	64,29%	86,50%	86,27%

oration is observer for the mountain car problem.

The resulting scores for the accuracy of the various classifiers under a binary formulation are shown in Table 2. In this case, the learned policy is represented by an ensemble of binary classifiers; each classifier corresponds to a single action. The final decision of the represented policy could be any action that is reported as positive by a binary classifier, since all of them are considered equally good. For evaluation purposes, an outcome of the ensemble is considered correct if at least one of the classifiers outputs a correct positive classification and no classifier returns a wrong positive classification; in other words, the ensemble offers at least one good action choice and no bad choices. Conversely, an outcome is considered wrong if even one classifier returns a wrong positive classification or no classifier returns a positive classification. Again, as before, best accuracy in each case (column) is marked in bold and the classifiers are listed (approximately) in rows of decreasing accuracy. The relative accuracy is similar to the multi-class case, with the Nearest Neighbors classifier (IB1) being the leader. What is important is that accuracy is boosted in all cases with no exception and in many cases significantly. We assert that this binary formulation of the classification problem is much more appropriate in the context of policy representation for reinforcement learning.

7 Learned Policy Performance

Our last goal is to test the performance of the learned policies in the original control problem. Reconstructing an optimal policy using one or many classifiers is not simply a matter of reproducing every single detail of it, but rather a matter of focusing at those details that reflect critical areas of the state space and yield a policy that performs equally well to an optimal one. To this end, we conducted wide experimentation to assess the performance quality of the learned policies. From initial experiments, it was found that using a large amount of training data (more than 30% of the total) always yields policies that perform optimally. Therefore, we focused our attention to training sets of small sizes to assess the degradation of each classifier as training examples become sparser and sparser.

To this end, we adopted the following experimental methodology. For each of the 5 classifiers (IB1, J48-C4.5, LibSVM, LVQ3, Bayes) and the 2 formulations (multi-class and binary), we drew randomly 10 different training sets for each of 10 percentage levels of training set size (from 0.2% to 30%) from the 62,500 total training examples. After learning, these 1000 learned policies were tested on the original control problem by running 100 trajectories using each one of them and measuring the number of failed trajectories in the pendulum starting at the upright position with a 10% disturbance and the average escape time in the mountain car starting the bottom of the valley. A failed trajectory in the pendulum problem is one where the pendulum falls down before the completion of 3000 steps corresponding to 5 minutes of balancing time. In the mountain car problem, all policies eventually manage to get the car out of the valley, so there are no failed trajectories, however the average escape time compared to the optimal escape time (about 101 steps) is a fair performance measure.

In all cases, one can clearly see that IB1 yields policies of superior performance even with a fraction of training examples. In the pendulum problem, LibSVM behaves equally well, however in the mountain car problem it fails to deliver improved policies with increasing amounts of training data. On the other hand, J48-C4.5 eventually delivers good policies in most cases, but only after a large amount of training data. Notice the huge performance difference between LibSVM and J48-C4.5 in binary policies for the pendulum problem, despite their almost identical classification accuracy. Finally, both LVQ3 and Bayes yield low performance analogous to their representational abilities. It is worth noticing that optimal performance is achieved in some cases with as few as 3% of the training data, capitalizing on the fact that modern classifiers are powerful enough to capture the critical structure of a target policy.

8 Discussion and Conclusion

We have presented a study on policy representations using classifiers. Our goal was to uncover the structure that

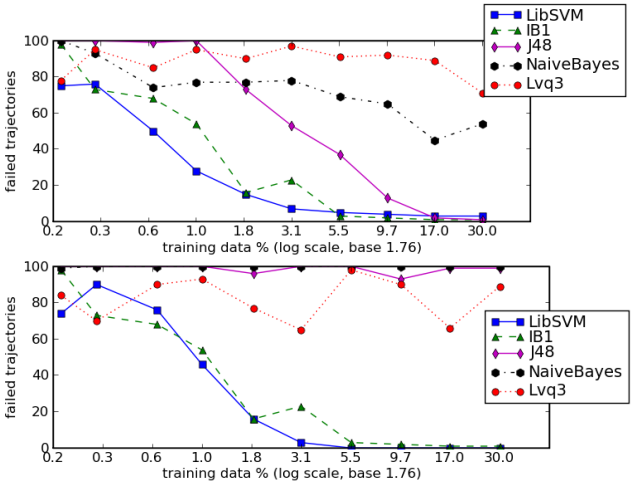

Figure 6. Pendulum: performance of multi-class (top) and binary (bottom) policies.

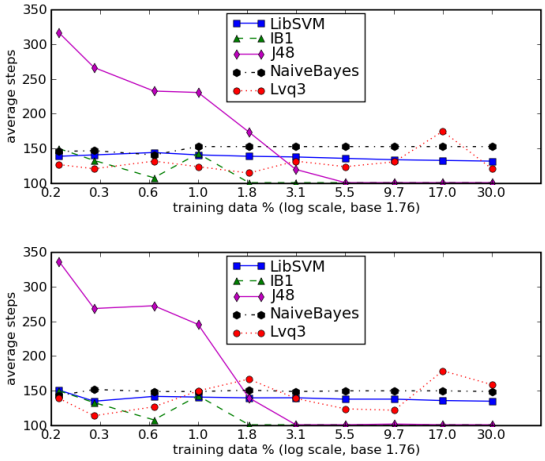

Figure 7. Mountain car: performance of multi-class (top) and binary (bottom) policies.

intuitively exists in optimal policies and test the suitability of various classifiers for representing them. Our findings are mixed: occasionally expected and occasionally counterintuitive. In summary, we conclude that: (a) (optimal) policies for typical reinforcement learning problems exhibit significant structure which can be exploited for representation purposes, (b) the multi-class formulation of the classification problem throws away all flexibility in selecting among equally good actions, whereas the binary formulation allows for incorporation of action selection flexibility, (c) good classification accuracy does not necessarily imply good policy performance, and (d) good policy performance can be achieved with only few examples if the critical structure of the policy is captured. From our results, we can conjecture that Nearest-Neighbors classifiers are the best choice for classifier-based policy representation, followed by Support Vector Machines (when only few training data are available) and Decision Trees (when many training data are available) which seem to work well in certain cases.

This study is only an initial step in a promising direction. In the future, we plan to combine our findings with a recent improved learning algorithm [3] for approximate policy iteration using classifier-based policy representation to assess performance within a complete learning framework where no optimal policy and clean training data are available. In the present study, no measure of criticality is utilized, however when the training examples for the classifiers are generated through experience, it is possible to define such measures and proceed with cost-sensitive classification. We strongly believe that this and similar future studies will bring the fields of reinforcement learning and supervised learning closer, effectively enabling a powerful technology that could make a difference in practice.

References

[1] C. W. Anderson. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Colorado State University, 2000.

[2] M. P. Deisenroth, J. Peters, and C. E. Rasmussen. Approximate dynamic programming with gaussian processes. In *Proceedings of the 2008 American Control Conference (ACC 2008)*, Seattle, WA, USA, 2008.

[3] C. Dimitrakakis and M. Lagoudakis. Rollout sampling approximate policy iteration. *Machine Learning*, 72(3):157–171, September 2008.

[4] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias. *Advances in Neural Information Processing Systems*, 16(3), 2004.

[5] A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

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

[7] M. G. Lagoudakis and R. Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th International Conference on Machine Learning*, pages 424–431, Washington, DC, USA, 2003.

[8] J. Langford and B. Zadrozny. Relating reinforcement learning performance to classification performance. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 473–480, Bonn, Germany, 2005.

[9] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[10] H. O. Wang, K. Tanaka, and M. F. Griffin. An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Trans. on Fuzzy Systems*, 4(1):14–23, 1996.

[11] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.