In *Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning*, Montreal, Canada, June 2009.

An Empirical Comparison of Abstraction in Models of Markov Decision Processes

Todd Hester TODD@CS.UTEXAS.EDU
The University of Texas at Austin, 1 University Station C0500, Austin, Texas 78712

Peter Stone PSTONE@CS.UTEXAS.EDU
The University of Texas at Austin, 1 University Station C0500, Austin, Texas 78712

Abstract

Reinforcement learning studies the problem of solving sequential decision making problems. Model-based methods learn an effective policy in few actions by learning a model of the domain and simulating experience in their models. Typical model-based methods must visit each state at least once, which can be infeasible in large domains. To overcome this problem, the model learning algorithm needs to generalize knowledge to unseen states and provide information about the states in which it needs more experience. In this paper, we use existing supervised learning techniques to learn the model of the domain. We empirically compare their effectiveness at generalizing knowledge across states on three different domains. Our results indicate that tree-based models perform the best after training on a small number of transitions, while support vector machines perform the best after a large number of transitions.

1. Introduction

Reinforcement learning (RL) studies the problem of finding effective solutions to sequential decision making problems (Sutton & Barto, 1998). For many agent-based applications, it is critical that an RL algorithm be very sample efficient: that it takes very few actions to learn an effective policy. We focus on sample efficiency as the key evaluation criterion for RL algorithms because in many agent-based applications, acquiring experiences can be very expensive and time-consuming. Two of the main approaches towards this goal are to incorporate generalization (function approximation) into model-free methods and to develop model-based algorithms. Model-based methods achieve high sample efficiency by learning a model of the domain and simulating experiences in their model, thus saving precious samples in the real world.

This paper is motivated primarily by the observation that the world is too large to explore exhaustively. Consider, for example, the act of identifying your favorite restaurant. One possible strategy is to visit the restaurants in order of their distances from your house, visiting closest ones first. In most American cities, doing so would likely lead you to visit several pizza parlors, fast food joints, and coffee shops before getting to many gourmet establishments. Nonetheless, provided that you have time to try every restaurant, eventually you will find your favorite one.

In practice, however, visiting every restaurant is infeasible in any reasonably large city. Rather, you must make choices about which ones to explore, and, more to the point, which *not* to explore. Deciding not to ever visit a particular restaurant entails some risk: a particular pizza parlor *could* be the restaurant that you ultimately would prefer over any other. But at some point, you must generalize across restaurants, for instance based on the name, the menu, the appearance, or the existence of golden arches out front. Skipping some restaurants enables you to expand the radius of your search and thus to more quickly find a restaurant that is close to your favorite, if not your absolute favorite.

Following this example, we would like an RL agent to do the same thing: avoid exploring some states by generalizing knowledge it learned from other states. To do this properly, the agent needs a mechanism that

Appearing in *Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning*, Montreal, Canada, 2009.

can generalize knowledge across states when learning a model of the environment. It then needs a method for driving exploration such that it explores sufficiently to learn a correct model. One approach to solving this problem would be to have the agent explore the states in which the model knows it makes inaccurate predictions.

In this paper, we explore the possibility of using existing supervised learning techniques to build generalizable models of Markov Decision Processes (MDP) that provide information on the accuracy of their predictions that could be used to motivate an agent's exploration. We then empirically compare the predictions of these models across three example domains.

2. Background

We adopted the standard Markov Decision Process formalism for this work (Sutton & Barto, 1998). An MDP consists of a set of states S, a set of actions A, a reward function $R(s,a)$, and a transition function $P(s'|s,a)$. In each state $s \in S$, the agent takes an action $a \in A$. Upon taking this action, the agent receives a reward $R(s,a)$ and reaches a new state s'. The new state s' is determined from the probability distribution $P(s'|s,a)$.

Model-based reinforcement learning methods learn a model of the domain and then simulate actions inside their models. The domain can be modeled by approximating its transition and reward functions. In many domains, the discrete state s is represented by a vector of n discrete state variables $s = \langle x_1, x_2, ..., x_n \rangle$. The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agent's lifetime.

The most efficient model-based methods are able to specifically target and explore the states in which their models need improvement. Knows What It Knows (KWIK) (Li et al., 2008) is a learning framework for efficient model learning that formalizes this approach. A learning algorithm that fits the KWIK framework must, with high probability, make an accurate prediction, or reply "I don't know" and request a label for that example. KWIK algorithms can be used effectively in an RL setting by encouraging the agent to explore the states for which the agent says "I don't know". With this encouragement, the agent is driven to collect the experiences necessary to learn a fully accurate model. Algorithms such as R-MAX (Brafman & Tennenholtz, 2001) and SLF-RMAX (Strehl et al., 2007) learn models of the domain using techniques that fit the KWIK framework.

3. Models

For our agent to behave as desired, the technique for learning a model of the MDP should have the following properties:

- Generalizes predictions well to unvisited states
- Has some measure of confidence in the accuracy of its predictions (It knows what it knows)

Using a model with these properties, the agent can explore the states in which the model has low confidence, and learn a model about the world without exploring every state by generalizing its knowledge to unseen states.

In this paper, we examine whether existing supervised learning techniques can be used to learn this model. Each model needs to predict the $P(s'|s,a)$, $R(s,a)$ and its confidence in its prediction $C(s,a)$ for each state-action pair. The confidence is used to compare the relative prediction accuracy of the model across different state-action pairs, but not across models. Therefore if $C(s,a) > C(t,b)$, then the agent is more confident in its estimates of $P(s'|s,a)$ and $R(s,a)$ than it is of its estimates of $P(t'|t,b)$ and $R(t,b)$. We compare against a tabular model as well:

- Tabular
  - *Model Learning*: For each input (s,a), the tabular model counted the number K of transitions to each next state s', the sum of rewards G from this state-action, and the number of visits T to this state-action. The predictions were then: $P(s'|s,a) = K(s')/T$ and $R(s,a) = G/T$. This model did not perform any generalization; the transition and reward functions were learned separately for each state-action pair.
  - *Confidence estimation*: The confidence measure C equaled the number of visits to the state-action: T, similar to R-Max.

For the remaining techniques that we tested, we built separate models to predict the change in each state feature as well as the reward. We predicted the relative change in each state feature rather than the absolute change because the relative change generalizes better across states. The first n models each made a prediction of the probabilities of the change in each state feature: $P(x_i^r|s,a)$, while the last model predicted the reward: $R(s,a)$. The input to each model was a vector containing the n state features and action

a: $\langle a, s_1, s_2, ..., s_n \rangle$. For the first n models, the desired output was the relative change in the state variable. For the last model, the desired output was the reward r.

After all the models are updated, they can be used to predict the full model of the domain. The first n models output probabilities for the relative change, x_i^r, of their particular state features. The predictions $P(x_i^r|s,a)$ for the n state features are combined to create a prediction of probabilities of the relative change of the state $s^r = \langle x_1^r, x_2^r, ..., x_n^r \rangle$. Assuming that each of the state variables transition independently, the probability of the change in state $P(s^r|s,a)$ is the product of the probabilities of each of its n state features:

$$P(s^r|s,a) = \Pi_{i=0}^{n} P(x_i^r|s,a) \qquad (1)$$

The relative change in the state, s^r, is added to the current state s to get the next state s'. The last model predicts reward $R(s,a)$. The combination of the model of the transition function and reward function make up a complete model of the underlying MDP.

The confidence C of the model was the minimum of the confidence reported by each of the individual models for each state feature and reward, because it is assumed the model is only as good as its worst prediction. The six models based on different supervised learning techniques are described below.

- C4.5 decision trees
  - *Model Learning*: The model based on C4.5 decision trees (Quinlan, 1986) built one tree to predict the change in each state feature. C4.5 built trees on the training samples by making optimal splits based on information gain ratio. It stopped making splits when the information gain ratio from the optimal split was below 0.001. All of the training samples are saved in their appropriate leaf. When classifying a state-action pair (s,a), it is classified into a leaf. Then $P(s'|s,a)$ is the count of transitions to s' in the leaf divided by the total number of instances in the leaf. Similarly, the reward is the average reward of the instances in the leaf.
  - *Confidence estimation*: The confidence C is the number of instances in the leaf that the input (s,a) was classified into.
- Committee of C4.5 decision trees
  - *Model Learning*: We built a model combining three C4.5 decision tree models together into a committee. Each tree model was trained with some randomness added into the decision of which split to make - the split was chosen from all splits with gain ratio within 0.05 of the optimal split. The model reported the predictions $P(s'|s,a)$ and $R(s,a)$ of a randomly selected model from the committee.
  - *Confidence estimation*: If we define the expected value of model x's prediction as $E_x(s'|s,a)$, then the confidence C is the maximum of $E_x(s'|s,a) - E_y(s'|s,a)$ over all s', x, y.
- Random Forest of decision trees
  - *Model Learning*: The Random Forests (Breiman, 2001) model combined the predictions of ten tree-based models. The trees were the same as C4.5 decision trees except that when making each split, 25% of the input features were eliminated. The optimal split was selected from the remaining input features. The predictions from the random forest were the average of the predictions of its ten member trees.
  - *Confidence estimation*: The confidence measure was the same as in the tree committee.
- Neural Networks
  - *Model Learning*: The neural network models were trained with back-propagation (Rumelhart et al., 1986) with 50 hidden neurons, and one output for each possible outcome of the state feature. The networks were trained to predict the probability of each outcome by training the output matching the input transition's outcome with a target of 1.0, while the remaining outputs had a target of 0.0. The outputs of the network should sum to 1.0. A final network was trained with one output whose target was the reward for that transition. The predictions $P(s'|s,a)$ were the values of the corresponding output neurons and $R(s,a)$ was the output of the reward network.
  - *Confidence estimation*: The confidence measure C was the negative of the absolute value of the difference between the sum of the outputs and 1.0. This was used as a measure of how much the outputs are miscalibrated, because they should sum to 1.0.
- Support Vector Machines
  - *Model Learning*: Multi-class Support Vector Machines (SVMs) (Burges, 1998) were the

sixth model. Each SVM was trained to predict the outcome and its probability using libsvm with an RBF kernel (Chang & Lin, 2001).
  - *Confidence estimation*: The confidence C was the distance of the input point (s, a) to the decision boundary in the SVM.

- K Nearest Neighbors
  - *Model Learning*: The K nearest neighbors (Cover & Hart, 1967) model saved all sample transitions. Distances between points were calculated as the sum of differences between all state features plus 1 if the action was different. It made predictions by averaging the outcomes and rewards of the five nearest neighbors to the test point.
  - *Confidence estimation*: The confidence C was the negative of the average distance to the five nearest points.

4. Experiments

We performed experiments comparing the seven different models on three domains. The domains were selected as examples of factored domains where it should be possible to generalize knowledge across states. For each experiment, some number n of (s, a, s', r) transitions would be randomly sampled from the MDP. The model would be trained on these n samples. We would then record the predictions of the model for every state-action in the MDP along with the model's confidence in each prediction. The model's predictions were then compared with the correct transitions in the MDP.

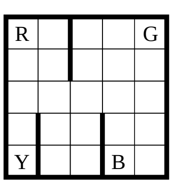

Figure 1. Taxi

The first domain used in the experiments was the classic Taxi domain introduced by Dietterich (Dietterich, 1998). The domain, shown in Figure 1, is a 5x5 gridworld with four landmarks that are labeled with one of the following colors: *red*, *green*, *blue* or *yellow*. The agent's state consists of its location in the gridworld in x, y coordinates, the location of the passenger (a landmark or in the *taxi*), and the passenger's destination (a landmark). The agent's goal is to navigate to the passenger's location, pick the passenger up, navigate to the passenger's destination and drop the passenger off. The agent has six actions that it can take. The first four (*north*, *south*, *west*, *east*) move the agent to the square in that respective direction with probability 0.8 and in a perpendicular direction with probability 0.1. If the resulting direction is blocked by a wall, the agent stays where it is. The fifth action is the *pickup* action, which picks up the passenger if she is at the taxi's location. The sixth action is the *putdown* action, which attempts to drop off the passenger. Each of the actions incurs a reward of -1, except for unsuccessful *pickup* or *putdown* actions, which produce a reward of -10. The episode is terminated by a successful *putdown* action, which provides a reward of $+20$.

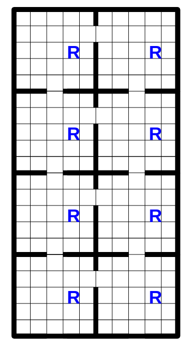

Figure 2. Castle

We introduce the Castle domain as another example of a highly structured domain where transition and reward dynamics are easily generalized. It consists of a gridworld made up of eight 5x5 rooms as shown in Figure 2. The agent has three state variables: its x and y coordinates relative to the room it is in, and the *id* of the current room. The agent has four actions, *north*, *south*, *east*, and *west*, which behave the same as in the Taxi domain. Each room has doors in identical locations as well as a square, marked R in the figure, that provides an additional reward ranging from 0 to 7. Since the doors and reward squares are in the same location in all the inner rooms, it is easy to generalize their locations across rooms.

The last domain we tested on was a non-gridworld domain we created called the Lights domain. In this domain, there is a light level that is controlled with a switch. The switch has three modes which vary how it is controlled. The domain has three state features: the *mode* of the light switch, the current *lightlevel*, and an *extra* variable added to confuse the models. The mode can be in three states: *random*, *locked* (boring), or *unlocked* (interesting). The *lightlevel* ranges from 0 to 9 and the *extra* variable ranges from 0 to 24. There are five actions: The first three change the *mode* to the three different modes and the last two attempt to move the *lightlevel* up or down. In the *random* mode, the *lightlevel* is moved to a random level between 0 and 7. In the *boring* mode, the *lightlevel* stays constant. In the *interesting* mode, the last two actions have an effect: they move the *lightlevel* up or down. The *extra* variable never changes. The agent receives a reward of -1 if the *lightlevel* is below 8, a reward of 0 when the *lightlevel* is 8, and a reward of 10 if the *lightlevel* is 9.

5. Results

We tested each algorithm on the three domains while varying the number of training samples from 50 to

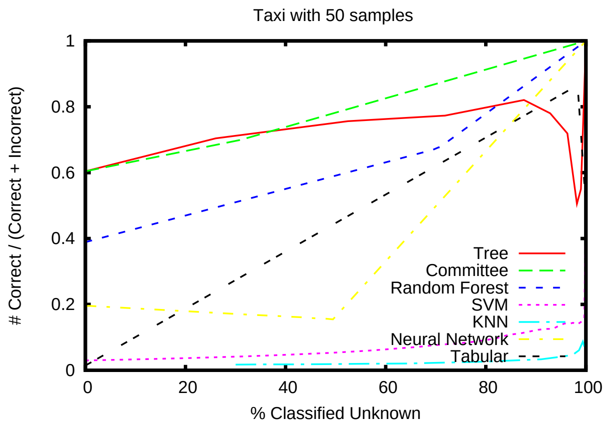

Figure 3. Operating characteristics of the models after training on 50 samples in the Taxi domain

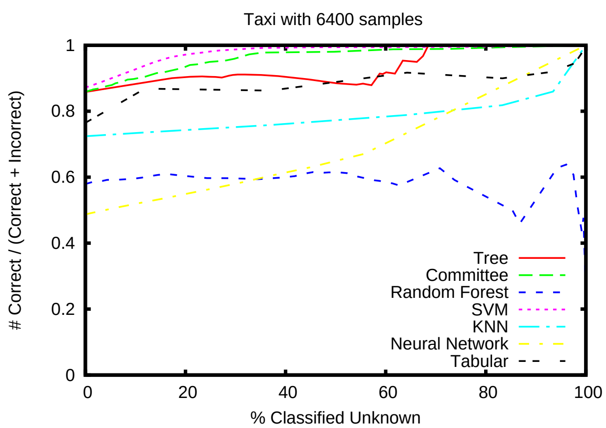

Figure 4. Operating characteristics of the models after training on 6400 samples in the Taxi domain

25600, doubling each time. We thresholded the confidence measures of the models to classify some predictions as *unknown*. Predictions with confidence below threshold were labeled as unknown and predictions with confidence over the threshold were tested and labeled as *correct* or *incorrect*.

By varying the confidence threshold over the entire range of confidences reported by the model, we built plots of the operating characteristics of each model. The data was plotted with the x axis representing the percentage of transitions classified as unknown, which varies from 0 to 100%. The y axis shows the percentage of the known predictions that were classified correctly. In these plots, we would like to find a method that classifies all of the transitions correctly while labeling all of them (the upper left corner of the graph). The method that performs the best after few training samples is the apparent best candidate for our RL agent.

Figure 3 shows the operating characteristics of the models after being trained on 50 sample transitions in

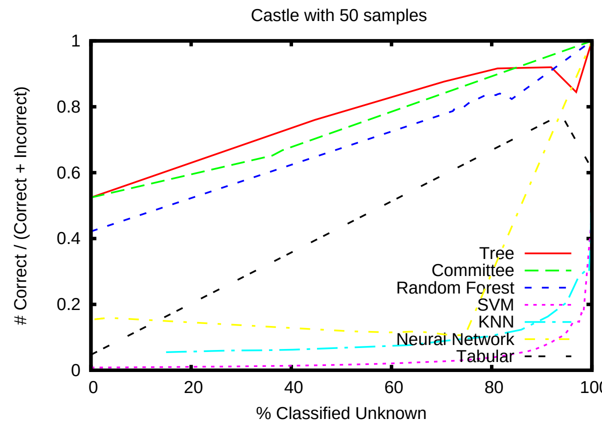

Figure 5. Operating characteristics of the models after training on 50 samples in the Castle domain

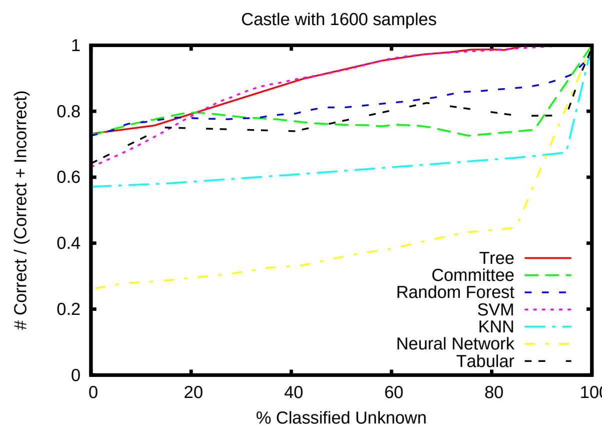

Figure 6. Operating characteristics of the models after training on 1600 samples in the Castle domain

the Taxi domain. Here the tree committee performed the best, followed by the single tree. The accuracy of the models improved with more samples, but the ordering remained roughly the same. At 200 samples, the single tree model surpassed the tree committee as the best model and the random forest separated itself from the remaining models as the third best model. The SVM started improving rapidly after 800 samples. Next, Figure 4 shows the operating characteristics after 6400 samples, when the SVM model surpassed the tree methods as the best model. By 25600 samples, the SVM model was nearly perfect, classifying more than 95% of the samples correctly at 0% unknown.

Figure 5 shows the operating characteristics of the models on the Castle domain after 50 sample transitions. Similar to the Taxi domain, the tree-based models performed the best, with the single tree outperforming both the tree committee and the random forest. Figure 6 shows the models after 1600 samples, when SVMs and the single tree performed equally well as the best models, with many of the other ap-

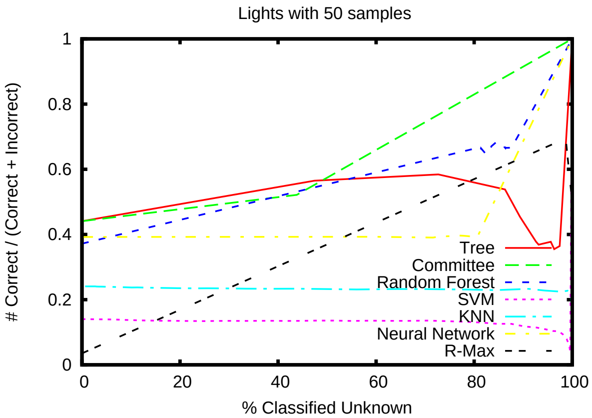

Figure 7. Operating characteristics of the models after training on 50 samples in the Lights domain

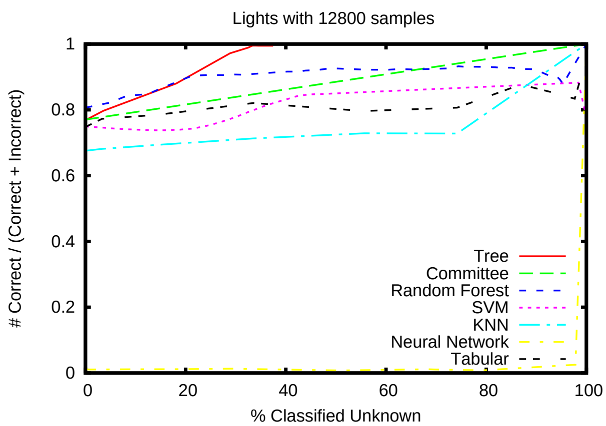

Figure 8. Operating characteristics of the models after training on 12800 samples in the Lights domain

proaches close behind. From that point forward, SVM performed the best, once again nearing perfect predictions by 25600 samples.

The operating characteristics of the models after training on 50 sample transitions on the Lights domain are shown in Figure 7. Again, the tree-based methods performed the best, with the single tree slightly outperforming the other two methods. By 200 samples, the three tree-based methods were clearly much better than the other algorithms. The methods all continued improving, although after about 1600 samples, the tree methods slowed down at about 80% correct at the 0% unknown level. Next we show the models after 12800 samples (Figure 8), when all the methods performed very well, with the exception of the neural networks.

6. Discussion

Our results showed that existing supervised learning techniques can be used as a model learning technique for an RL agent. Experiments across three domains showed that the models generalize well and have sufficient knowledge of what they know. While these results look promising, the ultimate test of these methods is to use them to learn models in a model-based reinforcement learning algorithm.

In all three domains, the tree-based models performed the best after low numbers of training samples. We hypothesize that this occurs because the tree methods generalize well from very small amounts of data. For example, in a gridworld domain, the tree models can correctly predict the outcomes for all the states in open space (states with no walls blocking movement) after seeing each action just once.

After enough training data is acquired, the SVM model surpassed the tree-based approaches on the two gridworld domains. This pattern suggests that the best technique may be a hybrid approach that uses trees early in learning and switches to SVMs once enough training samples have been acquired.

References

Brafman, R. I., & Tennenholtz, M. (2001). R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *IJCAI* (pp. 953–958).

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, *2*, 121–167.

Chang, C., & Lin, C. (2001). LIBSVM: a library for support vector machines.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*, 21–27.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *ICML* (pp. 118–126).

Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: a framework for self-aware learning. *ICML* (pp. 568–575).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Rumelhart, D., Hintont, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state mdps. *AAAI* (pp. 645–650). AAAI Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.