Journal of Machine Learning Research 15 (2014) 3925-3963 Submitted 12/12; Revised 5/14; Published 12/14

Active Imitation Learning: Formal and Practical Reductions to I.I.D. Learning

Kshitij Judah JUDAHK@EECS.OREGONSTATE.EDU
Alan P. Fern AFERN@EECS.OREGONSTATE.EDU
Thomas G. Dietterich TGD@EECS.OREGONSTATE.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of Electrical Engineering and Computer Science
Oregon State University
1148 Kelley Engineering Center
Corvallis, OR 97331-5501, USA

Editor: Joelle Pineau

Abstract

In standard passive imitation learning, the goal is to learn a policy that performs as well as a target policy by passively observing full execution trajectories of it. Unfortunately, generating such trajectories can require substantial expert effort and be impractical in some cases. In this paper, we consider *active imitation learning* with the goal of reducing this effort by querying the expert about the desired action at individual states, which are selected based on answers to past queries and the learner's interactions with an environment simulator. We introduce a new approach based on reducing active imitation learning to active i.i.d. learning, which can leverage progress in the i.i.d. setting. Our first contribution is to analyze reductions for both non-stationary and stationary policies, showing for the first time that the label complexity (number of queries) of active imitation learning can be less than that of passive learning. Our second contribution is to introduce a practical algorithm inspired by the reductions, which is shown to be highly effective in five test domains compared to a number of alternatives.

Keywords: imitation learning, active learning, active imitation learning, reductions

1. Introduction

Traditionally, passive imitation learning involves learning a policy that performs nearly as well as an expert's policy based on a set of trajectories of that policy. However, generating such trajectories is often tedious or even impractical for an expert (e.g., real-time low-level control of multiple game agents). In order to address this issue, we consider active imitation learning where full trajectories are not required, but rather the learner asks queries about specific states, which the expert labels with the correct actions. The goal is to learn a policy that is nearly as good as the expert's policy using as few queries as possible.

The active learning problem for i.i.d. supervised learning[1] has received considerable attention both in theory and in practice (Settles, 2012), which motivates leveraging that

1. i.i.d. stands for independent and identically distributed. The i.i.d. supervised learning setting is where the input data during both training and testing are drawn independently from the same data distribution.

work for active imitation learning. However, the direct application of i.i.d. approaches to active imitation learning can be problematic. This is because active i.i.d. learning algorithms assume access to either a target distribution over unlabeled input data (in our case states) or a large sample drawn from it. The goal then is to select the most informative query to ask, usually based on some combination of label (in our case actions) uncertainty and unlabeled data density. Unfortunately, in active imitation learning, the learner does not have direct access to the target state distribution, which is the state distribution induced by the unknown expert policy.

In principle, one could approach active imitation learning by assuming a uniform or an arbitrary distribution over the state space and then apply an existing active i.i.d. learner. However, such an approach can perform very poorly. This is because if the assumed distribution is considerably different from that of the expert, then the learner is prone to ask queries in states rarely or even never visited by the expert. For example, consider a bicycle balancing problem. Clearly, asking queries in states where the bicycle has entered an unavoidable fall is not very useful, because no action can prevent a crash. However, an active i.i.d. learning technique will tend to query in such uninformative states, leading to poor performance, as shown in our experiments. Furthermore, in the case of a human expert, a large number of such queries poses serious usability issues, since labeling such states is clearly wasted effort from the expert's perspective.

In this paper, we consider the problem of reducing active imitation learning to active i.i.d. learning both in theory and practice. Our first contribution is to analyze the Probably Approximately Correct[2] (PAC) label complexity (number of expert queries) of a reduction for learning non-stationary policies, which requires only minor modification to existing results for passive learning. Our second contribution is to introduce a reduction for learning stationary policies resulting in a new algorithm, *Reduction-based Active Imitation Learning (RAIL)*, and an analysis of the label complexity. The resulting complexities for active imitation learning are expressed in terms of the label complexity for the i.i.d. case and show that there can be significant query savings compared to existing results for passive imitation learning. Our third contribution is to describe a new practical algorithm, RAIL-DA (for data aggregation), inspired by the RAIL algorithm, which makes a series of calls to an active i.i.d. learning algorithm. We evaluate RAIL-DA in five test domains and show that it is highly effective when used with an i.i.d. algorithm that takes the unlabeled data density into account.

The rest of the paper is organized as follows. We begin by reviewing the relevant related work in Section 2. In Section 3, we present the necessary background material and describe the active imitation learning problem setup. In Section 4, we present the proposed reductions for the cases of non-stationary and stationary policies. In Section 5, we present the RAIL-DA algorithm. In Section 6, experimental results are presented. In Section 7, we summarize and present some directions for future research.

2. Related Work

Active learning has been studied extensively in the i.i.d. supervised learning setting (Settles, 2012) but to a much lesser degree for sequential decision making, which is the focus of

2. See Section 3.3.

active imitation learning. Several studies have considered active learning for *reinforcement learning (RL)* (Clouse, 1996; Mihalkova and Mooney, 2006; Gil et al., 2009; Doshi et al., 2008), where learning is based on both autonomous exploration and queries to an expert. In our imitation learning framework, in contrast, we do not assume a reward signal and learn only from expert queries. Other work (Shon et al., 2007) studies active imitation learning in a multiagent setting, where the expert is itself a reward seeking agent which acts to maximize its own reward, and hence is not necessarily helpful for learning. In the current setting, we only consider helpful experts.

One approach to imitation learning is *inverse RL (IRL)* (Ng and Russell, 2000), where a reward function is learned based on a set of target policy trajectories. The learned reward function and transition dynamics are then given to a planner to obtain a policy. There has been limited work on active IRL. This includes Active Sampling (Lopes et al., 2009), a Bayesian approach where the posterior over reward functions is used to select the state with maximum uncertainty over the actions. Another Bayesian approach (Cohn et al., 2010, 2011) models uncertainty about the entire MDP model and uses a decision-theoretic criterion, Expected Myopic Gain (EMG), to select various types of queries to pose to the expert, e.g., queries about transition dynamics, the reward function, or optimal action at a particular state. For autonomous navigation tasks, Silver proposed two active IRL techniques that request demonstrations from the expert on examples that are either novel (novelty reduction) or uncertain (uncertainty reduction) to the learner (Silver et al., 2012). Specifically, in novelty reduction, a start and goal location is selected such that the path that may most likely be demonstrated by the expert results in the learner seeing novel portions of the navigation terrain. This helps in learning behavior on previously unseen regions of the navigation terrain. In uncertainty reduction, a start and goal location is selected such that there is high uncertainty about the best path from start to goal.

While promising, the scalability of these approaches is hindered by the assumptions made by IRL, and these approaches have only been demonstrated on small problems. In particular, they require that the exact domain dynamics are provided or can be learned, which is often not realistic, for example, in the Wargus domain considered in this paper. Furthermore, even when a model is available, prior IRL approaches require an efficient planner for the MDP model. For the large domains considered in this paper, standard planning techniques for flat MDPs, which scale polynomially in the number of states, are not practical. While there has been substantial work on MDP planning for large domains via the use of factored representations (e.g., Boutilier et al. 1999) or simulators (e.g., Kocsis and Szepesvri 2006), robustness and scalability are still problematic in general.

To facilitate scalability, rather than following an IRL framework we consider a *direct imitation* framework where we attempt to directly learn a policy instead of the reward function and/or transition dynamics. Unlike inverse RL, this framework does not require an exact dynamic model nor an efficient planner. Rather, our approach requires only a simulator of the environment dynamics that is able to generate trajectories given a policy. The simulator may or may not be exact, and the performance of our approach will depend on how precise the simulator is. Even though a precise simulator is not always available for a real world domain, for many domains such a simulator is often available (e.g., flight simulators, computer network simulators etc.), even when a compact description of the transition dynamics and/or a planner are not.

Active learning work in the direct imitation framework includes Confidence Based Autonomy (CBA) (Chernova and Veloso, 2009), and the related dogged learning framework (Grollman and Jenkins, 2007), where a policy is learned in an online manner as it is executed. When the learner is uncertain about what to do at the current state, the policy is paused and the expert is queried about what action to take, resulting in a policy update. The execution resumes from the current state with the learner taking the action suggested by the expert. One can roughly view CBA as a reduction of imitation learning to stream-based active learning where the learner receives unlabeled inputs (states) one at a time and must decide whether or not to request the label (action) of the current input. CBA makes this decision by estimating its uncertainty about the action to take at a given state and requesting an action label for states with uncertainty above a threshold. One difficulty in applying this approach is setting the uncertainty threshold for querying the expert. While an automated threshold selection approach is suggested by Chernova and Veloso (Chernova and Veloso, 2009), our experiments show that it is not always effective (See Section 6). In particular, we observed that the proposed threshold selection mechanism is quite sensitive to the initial training data supplied to the learner.

Recently, Ross et al. proposed novel algorithms for imitation learning that are able to actively query the expert on states encountered during the execution of the policy being trained (Ross and Bagnell, 2010; Ross et al., 2011). The motivation behind these algorithms is to eliminate the discrepancy between the training (expert's) and test (learner's) state distributions that arises in the traditional passive imitation learning approach whenever the learned policy is unable to exactly mimic the expert's policy. This discrepancy often leads to the poor performance of the traditional approach. Note that such an issue is not present in the i.i.d. learning setting, where mistakes made by the learner do not influence the distribution of future test samples. They show that under certain assumptions their algorithms achieve better theoretical performance guarantees than traditional passive imitation learning.

However, because the primary goal of these algorithms is not to minimize the labeling effort of the expert, these algorithms query the expert quite aggressively, which makes them impractical for human experts or computationally expensive with automated experts. To see this, consider the first iteration of the DAGGER algorithm proposed by Ross et al. (Ross et al., 2011). In the first iteration, DAGGER trains a policy on a set of expert-generated trajectories, as in passive imitation learning. Thus, in practice the query complexity of the first iteration of DAGGER will be similar to that of passive imitation learning. In subsequent iterations, additional queries are asked by querying the expert on states along trajectories produced by learned policies in prior iterations. In contrast, our work focuses on active querying for the purpose of minimizing the expert's labeling effort. In particular, we show that an active approach can achieve an improved query complexity over passive in theory (under certain assumptions) and in practice. Like our work, their approach also requires a dynamics simulator to help select queries.

Our goal in this paper is to study the problem of active imitation learning and show that it can achieve better label complexity than passive imitation learning. To this end, we mention some prior work on the theoretical analysis of the label complexity of passive imitation learning. Khardon formalized a model for passive imitation learning of deterministic stationary policies in the realizable setting and gave a PAC-style label complexity result

(Khardon, 1999). He showed that for any policy class for which there exists a consistent learner, the class is efficiently learnable in the sense that only a polynomial number of expert trajectories are required by the learner to produce a policy as good as the expert's policy. However, the result holds for only deterministic policies in the realizable setting and the generalizations to stochastic policies and the agnostic setting were left as future work.

More recently, Syed and Schapire performed theoretical analysis of passive imitation learning in a more general setting where the expert policy is allowed to be stochastic and the learning can be agnostic (Syed and Schapire, 2010). In their analysis, they take a reduction based approach, where the problem of passive imitation learning is reduced to classification, and they relate the performance of the learned policy to the accuracy of the classifier. Standard PAC analysis can then be used to show that only a polynomial number of expert trajectories are required to achieve the desired level of performance. A similar analysis was done by Ross and Bagnell (Ross and Bagnell, 2010). To our knowledge, no prior work has addressed the relative sample complexity of active versus passive imitation learning, which is one of the primary contributions of this paper. Some of the material in this paper appeared in an earlier version of the paper (Judah et al., 2012).

3. Problem Setup and Background

In this section, we present the necessary background material and formally set up the active imitation learning problem.

3.1 Markov Decision Processes

We consider imitation learning in the framework of Markov decision processes (MDPs). An MDP is a tuple $\langle S, A, P, R, I\rangle$, where S is the set of states, A is the finite set of actions, $P(s, a, s')$ is the transition function denoting the probability of transitioning to state s' upon taking action a in state s, $R(s, a) \in [0, 1]$ is the reward function giving the immediate reward in state s upon taking action a, and I is the initial state distribution. A stationary policy $\pi : S \mapsto A$ is a deterministic mapping from states to actions such that $\pi(s)$ indicates the action to take in state s when executing π. A non-stationary policy is a tuple $\pi = (\pi_1, \ldots, \pi_T)$ of T stationary policies such that $\pi(s, t) = \pi_t(s)$ indicates the action to take in state s at time t when executing π, where T is the time horizon. The expert's policy, which we assume is deterministic, is denoted by π^*.

A key concept used in this paper is the notion of a state distribution of a policy at a particular time step. We use $d^t_\pi : S \mapsto [0, 1]$ to denote the state distribution induced at time step t by starting in $s_1 \sim I$ and then executing π. Note that $d^1_\pi = I$ for all policies. We use $d_\pi = \frac{1}{T}\sum_{t=1}^{T} d^t_\pi$ to denote the state distribution induced by policy π over T time steps. To sample an (s, a) pair from d^t_π, we start in $s_1 \sim I$, execute π to generate a trajectory $\mathcal{T} = (s_1, a_1, \ldots, s_T, a_T, s_{T+1})$ and set $(s, a) = (s_t, a_t)$. Similarly, to sample from d_π, we first sample a random time step $t \in \{1, \ldots, T\}$, and then sample an (s, a) pair from d^t_π. Note that in order to sample from d_{π^*} (or $d^t_{\pi^*}$), we need to execute π^*. Throughout the paper, we assume that the only way π^* can be executed is by querying the expert for an action in the current state and executing the given action, which puts significant burden on the expert.

The T-horizon value of a policy $V(\pi)$ is the expected total reward of trajectories that start in $s_1 \sim I$ at time $t = 1$ and then execute π for T steps. This can be expressed as

$$V(\pi) = T \cdot \mathbb{E}_{s \sim d_\pi}[R(s, \pi(s))].$$

The regret of a policy π with respect to an expert policy π^* is equal to $V(\pi^*) - V(\pi)$.

3.2 Problem Setup

Passive Imitation Learning. In imitation learning, the goal is to learn a policy π with a small regret with respect to the expert. In this work, we consider the direct imitation learning setting, where the learner directly selects a policy π from a hypothesis class Π (e.g., linear action classifiers). In the *passive imitation learning* setup, the protocol is to provide the learner with a training set of full execution trajectories of π^* and the state-action pairs (or a sample of them) are passed to a passive i.i.d. supervised learning algorithm L_p. The hypothesis $\pi \in \Pi$ that is returned by L_p is used as the learned policy.

Active Imitation Learning. To help avoid the cost of generating full trajectories, the *active imitation learning* setup allows the learner to pose *action queries.* Each action query involves presenting a state s to the expert and then obtaining the desired action $\pi^*(s)$ from the expert. In addition to having access to the expert for answering queries, we assume that the learner has access to a simulator of the MDP. The input to the simulator is a policy π and a horizon T. The simulator output is a state trajectory that results from executing π for T steps starting in the initial state. The learner is allowed to interact with this simulator as part of its query selection process. The simulator is not assumed to provide a reward signal, which means that the learner cannot find π by pure reinforcement learning. The only way for the learner to gain information about the target policy is through queries to the expert at selected states.

Given access to the expert and the simulator of the MDP, the goal in active imitation learning is to learn a policy $\pi \in \Pi$ that has a small regret by posing as few queries to the expert as possible. Note that it is straightforward for the active learner to generate full expert trajectories by querying the expert at each state of the simulator it encounters. Thus, an important baseline active learning approach is to generate an appropriate number N of expert trajectories for consumption by a passive learner. The number of queries for this baseline is $N \cdot T$. A fundamental question that we seek to address is whether an active learner can achieve the same performance with significantly fewer queries both in theory and in practice.

3.3 Background on I.I.D. Learning

Since our analysis in the next two sections is based on reducing to active i.i.d. learning and comparing to passive i.i.d. learning, we briefly review the *Probably Approximately Correct (PAC)* (Valiant, 1984) learning formulation for the i.i.d. setting. Here we consider the realizable PAC setting, which will be the focus of our initial analysis. Section 4.3 extends to the non-realizable or agnostic setting.

Passive Learning. In passive i.i.d. supervised learning, N i.i.d. data samples are drawn from an unknown distribution $D_{\mathcal{X}}$ over an input space $\mathcal{X}$ and are labeled according to an unknown target classifier $f : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{Y}$ denotes the label space. In the realizable PAC setting it is assumed that f is an element of a known class of classifiers H and, given a set of N examples, a learner outputs a hypothesis $h \in H$. Let $e_f(h, D_{\mathcal{X}}) = \mathbb{E}_{x \sim D_{\mathcal{X}}}[h(x) \neq f(x)]$ denote the generalization error of the returned classifier h. Standard PAC learning theory provides a bound on the number of labeled examples that is sufficient to guarantee that for any distribution $D_{\mathcal{X}}$, with probability at least $1 - \delta$, the returned classifier h will satisfy $e_f(h, D_{\mathcal{X}}) \leq \epsilon$. We will denote this bound by $N_p(\epsilon, \delta)$, which corresponds to the label/query complexity of i.i.d. passive supervised learning for a class H. We will also denote a passive learner that achieves this label complexity as $L_p(\epsilon, \delta)$.

Active Learning. In active i.i.d. learning, the learner is given access to two resources rather than just a set of training data: 1) A "cheap" resource (Sample) that can draw an unlabeled sample from $D_{\mathcal{X}}$ and provide it to the learner when requested, 2) An "expensive" resource (Label) that can label a given unlabeled sample according to target concept f when requested. Given access to these two resources, an active learning algorithm is required to learn a hypothesis $h \in H$ while posing as few queries to Label as possible. It can, however, pose a much larger number of queries to Sample (though still polynomial), as it is cheap.

Unlike passive i.i.d. learning, formal label/query complexity results for active i.i.d. learning depend not only on the hypothesis class being considered, but also on joint properties of the target hypothesis and data distribution (e.g., as measured by the disagreement coefficient proposed by Hanneke, 2009). We use $N_a(\epsilon, \delta, D_{\mathcal{X}})$ to denote the label complexity (i.e., calls to Label) that is sufficient for an active learner to return an h that for distribution $D_{\mathcal{X}}$ with probability at least $1 - \delta$ satisfies $e_f(h, D_{\mathcal{X}}) \leq \epsilon$. Note that here we did not explicitly parameterize N_a by the target hypothesis f since, in the context of our work, f will correspond to the expert policy and can be considered as fixed. We will denote an active learner that achieves this label complexity as $L_a(\epsilon, \delta, D)$, where the final argument D indicates that the Sample function used by L_a samples from distribution D.

It has been shown that for certain problem classes, N_a can be exponentially smaller than N_p (Hanneke, 2009; Dasgupta, 2011). For example, in the realizable learning setting (i.e., the target concept is in the hypothesis space), for any active learning problem with finite VC-dimension and finite disagreement coefficient, the sample complexity is exponentially smaller for active learning compared to passive learning with respect to $\frac{1}{\epsilon}$. That is, ignoring the dependence on δ, $N_p = O(\frac{1}{\epsilon})$ whereas $N_a = O(\log(\frac{1}{\epsilon}))$. A concrete problem for which this is the case is when the data are uniformly distributed on a unit sphere in a d dimensional input space $\mathbb{R}^d$, and the hypothesis space H consists of homogeneous linear separators. As an example active learning algorithm that achieves this performance, the algorithm of Cohn et al. (Cohn et al., 1994) simply samples a sequence of unlabeled examples and queries for the label of example x only when there are at least two hypotheses that disagree on the label of x, but agree on all previously labeled examples.

While results such as the above gives some theoretical justification for the use of active learning over passive learning in the i.i.d. setting, the results and understanding are not nearly as broad as for passive learning. Further, there are known limitations to the advantages of active versus passive learning. For example, lower bounds have been shown (Dasgupta, 2006; Beygelzimer et al., 2009a) implying that no active learning algorithm can asymptotically improve over passive learning across all problems with finite VC-dimension. However, despite the limited theoretical understanding, there is much empirical evidence that in practice active learning algorithms can often dramatically reduce the required amount of labeled data compared to passive learning. Further, there are active learning algorithms that in the worst case are guaranteed to achieve performance similar to passive learning in the worst case, while also showing exponential improvement in the best case (Beygelzimer et al., 2009a).[3]

4. Reductions for Active Imitation Learning

One approach to solving novel machine learning problems is via reduction to well-studied core problems. A key advantage of this reduction approach is that theoretical and empirical advances on the core problems can be translated to the more complex problem. For example, i.i.d. multi-class and cost-sensitive classification have been reduced to i.i.d. binary classification (Zadrozny et al., 2003; Beygelzimer et al., 2009b). In particular, these reductions allow guarantees regarding binary classification to translate to the target problems. Further, the reduction-based algorithms have shown equal or better empirical performance compared to specialized algorithms. More closely related to our work, in the context of sequential decision making, both imitation learning and structured prediction have been reduced to i.i.d. classification (Daumé et al., 2009; Syed and Schapire, 2010; Ross and Bagnell, 2010).

In this section, we consider a reduction approach to active imitation learning. In particular, we reduce to active i.i.d. learning, which is a core problem that has been the focus of much theoretical and empirical work. The key result is to relate the label complexity of active imitation learning to the label complexity of active i.i.d. learning. In doing so, we can assess when improved label complexity (either empirical or theoretical) of active i.i.d. learning over passive i.i.d. learning can translate to improved label complexity of active imitation learning over passive imitation learning. In what follows, we first present a reduction for the case of deterministic non-stationary policies. Next, we give a reduction for the more difficult case of deterministic stationary policies.

4.1 Non-Stationary Policies

Syed and Schapire analyze the traditional reduction from passive imitation learning to passive i.i.d. learning for non-stationary policies (Syed and Schapire, 2010). The algorithm receives N expert trajectories as input, noting that the state-action pairs at time t across trajectories can be viewed as i.i.d. draws from distribution $d^t_{\pi^*}$. The algorithm, then returns the non-stationary policy $\hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_T)$, where $\hat{\pi}_t$ is the policy returned by running the learner L_p on examples from time t.

3. A simple example of such an algorithm is the previously mentioned algorithm of Cohn et al. (Cohn et al., 1994) for the realizable learning setting. In this case, active learning can be stopped after drawing a number of unlabeled instances equal to the passive query complexity of the hypothesis class. This is because, for each instance, the algorithm either asks a query to get the label or the algorithm knows the label in cases when there is no disagreement. Thus, in the worst case, the algorithm will query each drawn example and ask for the same number of labels as a passive algorithm. But when the disagreement coefficient is finite, exponentially fewer queries will be made.

Let $\epsilon_t = e_{\pi_t^*}(\hat{\pi}_t, d^t_{\pi^*})$ be the generalization error of $\hat{\pi}_t$ at time t. Syed and Schapire (Syed and Schapire, 2010, Lemma 3)[4] show that if at each time step $\epsilon_t \leq \epsilon'$, then $V(\hat{\pi}) \geq V(\pi^*) - \epsilon' T^2$. Hence, if we are interested in learning a $\hat{\pi}$ whose regret is no more than ϵ with high probability, then we must simultaneously guarantee that with high probability $\epsilon_t \leq \frac{\epsilon}{T^2}$ at all time steps. This can be achieved by calling the passive learner L_p at each time step with $N_p(\frac{\epsilon}{T^2}, \frac{\delta}{T})$ examples. Thus, the overall passive label complexity of this algorithm (i.e., the number of actions provided by the expert) is $T \cdot N_p(\frac{\epsilon}{T^2}, \frac{\delta}{T})$. To our knowledge, this is the best known label complexity for passive imitation learning of non-stationary policies.

Our goal now is to provide a reduction from active imitation learning to active i.i.d. learning that can achieve an improved label complexity. A naive way to do this would simply replace calls to L_p in the above approach with calls to an active learner L_a. Note, however, that in order to do this the active learner at time step t requires the ability to sample from the unlabeled distribution $d^t_{\pi^*}$. Generating each such unlabeled sample requires executing the expert policy for t steps from the initial state, which in turn requires t label queries to the expert. Thus, the label complexity of this naive approach will be at least linearly related to the number of unlabeled examples required by the active i.i.d. learning algorithm. Typically, this number is similar to the passive label complexity rather than the potentially much smaller active label complexity. Thus, the naive reduction does not yield an advantage over passive imitation learning.

It turns out that for a slightly more sophisticated reduction to passive i.i.d. learning, introduced by Ross and Bagnell (Ross and Bagnell, 2010), it is possible to simply replace L_p with L_a and maintain the potential benefit of active learning. Ross and Bagnell introduced the forward training algorithm for non-stationary policies, which trains a non-stationary policy in a series of T iterations. In particular, iteration t trains policy $\hat{\pi}_t$ by calling a passive learner L_p on a labeled data set drawn from the state distribution induced at time t by the non-stationary policy $\hat{\pi}^{t-1} = (\hat{\pi}_1, \ldots, \hat{\pi}_{t-1})$, where $\hat{\pi}_1$ is learned on states drawn from the initial distribution I. The motivation for this approach is to train the policy at time step t based on the same state-distribution that it will encounter when being run after learning. By doing this, they show that the algorithm has a worst case regret of ϵT^2 and under certain assumptions can achieve a regret as low as $O(\epsilon T)$.

Importantly, the state-distribution used to train $\hat{\pi}_t$ given by $d^t_{\hat{\pi}^{t-1}}$ is easy for the learner to sample from without making queries to the expert. In particular, to generate a sample the learner can simply simulate $\hat{\pi}^{t-1}$, which is available from previous iterations, from a random initial state and return the state at time t. Thus, we can simply replace the call to L_p at iteration t with a call to L_a with unlabeled state distribution $d^t_{\hat{\pi}^{t-1}}$ as input. More formally, the *active forward training algorithm* is presented in Algorithm 1.

Ross and Bagnell (Ross and Bagnell, 2010, Theorem 3.1) give the worst case bound on the regret of the forward training algorithm which assumes the generalization error at each iteration is bounded by ϵ. Since we also maintain that assumption when replacing L_p with L_a (the active variant) we immediately inherit that bound.

4. The main result of Syed and Schapire (Syed and Schapire, 2010) holds for stochastic expert policies and requires a more complicated analysis that results in a looser bound. Lemma 3 is strong enough for deterministic expert policies, which is the assumption made in our work.

Algorithm 1 Active Forward Training

```
Input: active i.i.d. learning algorithm $L_a$, $\epsilon$, $\delta$
Output: non-stationary policy $\hat{\pi} = (\hat{\pi}_1, \dots, \hat{\pi}_T)$
1: Initialize $\hat{\pi}_1 = L_a(\epsilon, \frac{\delta}{T}, I)$    ▷ queries by $L_a$ answered by expert; unlabeled data
   is generated from initial state distribution $I$.
2: for $t = 2$ to $T$ do
3:     $\hat{\pi}^{t-1} = (\hat{\pi}_1, \dots, \hat{\pi}_{t-1})$
4:     $\hat{\pi}_t = L_a(\epsilon, \frac{\delta}{T}, d^t_{\hat{\pi}^{t-1}})$    ▷ queries by $L_a$ answered by expert; unlabeled data is
   generated using simulator and $\hat{\pi}^{t-1}$ as described in the main text.
5: end for
6: return $\hat{\pi} = (\hat{\pi}_1, ..., \hat{\pi}_T)$
```

Proposition 1 *Given a PAC active i.i.d. learning algorithm L_a, if active forward training is run by giving L_a parameters ϵ and $\frac{\delta}{T}$ at each step, then with probability at least $1 - \delta$ it will return a non-stationary policy $\hat{\pi}$ such that $V(\hat{\pi}) \geq V(\pi^*) - \epsilon T^2$.*

Note that L_a is run with $\frac{\delta}{T}$ as the reliability parameter to ensure that all T iterations succeed with probability at least $1 - \delta$.

We can apply Proposition 1 to obtain the overall label complexity of active forward training required to achieve a regret of less than ϵ with probability at least $1 - \delta$. In particular, we must run the active learner at each of the T iterations with parameters $\frac{\epsilon}{T^2}$ and $\frac{\delta}{T}$, giving an overall label complexity of $\sum_{t=1}^{T} N_a(\frac{\epsilon}{T^2}, \frac{\delta}{T}, d^t_{\hat{\pi}^{t-1}})$, where $d^1_{\hat{\pi}^0} = I$ and the $\hat{\pi}^{t-1}$ are random variables in this expression. Recall, from above, that the best known label complexity of passive imitation learning is $T \cdot N_p(\frac{\epsilon}{T^2}, \frac{\delta}{T})$.

Comparing these quantities we see that if we use an active learning algorithm whose sample complexity is no worse than that of passive, i.e., $N_a(\frac{\epsilon}{T^2}, \frac{\delta}{T}, d^t_{\hat{\pi}^{t-1}})$ is no worse than $N_p(\frac{\epsilon}{T^2}, \frac{\delta}{T})$ for any t, then the expected sample complexity of active imitation learning will be no worse than the passive case. As mentioned in the previous section, such i.i.d. active learning algorithms can be realized. Further, if in addition, for some iterations the expected value of $N_a(\frac{\epsilon}{T^2}, \frac{\delta}{T}, d^t_{\hat{\pi}^{t-1}})$ for some values of t is better than the passive complexity, then there will be an overall expected improvement over passive imitation learning. While this additional condition cannot be verified in general, we know that such cases can exist, including cases of exponential improvement. Further, empirical experience in the i.i.d. setting also suggests that in practice N_a can often be expected to be substantially smaller than N_p and rarely worse. The above result suggests that those empirical gains will be able to transfer to the imitation learning setting.

4.2 Stationary Policies

A drawback of active forward training is that it is impractical for large T and the resulting policy cannot be run indefinitely. We now consider the case of learning stationary policies; first we review the existing results for passive imitation learning.

In the traditional approach, a stationary policy $\hat{\pi}$ is trained on the expert state distribution d_{π^*} using a passive learning algorithm L_p and returning a stationary policy $\hat{\pi}$. Ross and Bagnell (Ross and Bagnell, 2010, Theorem 2.1) show that if the generalization error of $\hat{\pi}$

Algorithm 2 RAIL

```
Input: active i.i.d. learning algorithm $L_a$, $\epsilon$, $\delta$
Output: stationary policy $\hat{\pi}$

1: Initialize $\hat{\pi}^0$ to arbitrary policy or based on prior knowledge
2: for $t = 1$ to $T$ do
3:     $\hat{\pi}^t = L_a(\epsilon, \frac{\delta}{T}, d_{\hat{\pi}^{t-1}})$    ▷ queries by $L_a$ answered by expert; unlabeled data is
   generated using simulator as described in Section 3
4: end for
5: return $\hat{\pi}^T$
```

with respect to the i.i.d. distribution d_{π^*} is bounded by ϵ' then $V(\hat{\pi}) \geq V(\pi^*) - \epsilon' T^2$. Since generating i.i.d. samples from d_{π^*} can require up to T queries (see Section 3) the passive label complexity of this approach for guaranteeing a regret less than ϵ with probability at least $1 - \delta$ is $T \cdot N_p(\frac{\epsilon}{T^2}, \delta)$. Again, to our knowledge, this is the best known label complexity for passive imitation learning. Further, Ross and Bagnell (Ross and Bagnell, 2010) show that there are imitation learning problems where this bound is tight, showing that in the worst case, the traditional approach cannot be shown to do better.

The above approach cannot be converted into an active imitation learner by simply replacing the call to L_p with L_a, since again we cannot sample from the unlabeled distribution d_{π^*} without querying the expert. To address this issue, we introduce a new algorithm called *RAIL (Reduction-based Active Imitation Learning)* which makes a sequence of T calls to an active i.i.d. learner, noting that it is likely to find a useful stationary policy well before all T calls are issued. RAIL is an idealized algorithm intended for analysis, which achieves the theoretical goals but has a number of inefficiencies from a practical perspective. Later in Section 5, we describe the practical instantiation that is used in our experiments.

RAIL is similar in spirit to active forward training, though its analysis is quite different and more involved. Like forward-training, RAIL iterates for T iterations, but on each iteration, RAIL learns a new stationary policy $\hat{\pi}^t$ that can be applied across all time steps $t = 1 \ldots T$. Note that T denotes the length of the horizon as well as the total number of iterations that RAIL runs for. Similarly t denotes a single time step as well as a single iteration of RAIL. Iteration $t+1$ of RAIL learns a new policy $\hat{\pi}^{t+1}$ that achieves a low error rate at predicting the expert's actions with respect to the state distribution of the previous policy $d_{\hat{\pi}^t}$. More formally, Algorithm 2 gives pseudocode for RAIL. The initial policy $\hat{\pi}^0$ is arbitrary and could be based on prior knowledge and the algorithm returns the final policy $\hat{\pi}^T$, which is learned using the active learning applied to unlabeled state distribution $d_{\hat{\pi}^{T-1}}$.

Similar to active forward training, RAIL makes a sequence of T calls to an active learner. Unlike forward training, however, the unlabeled data distributions used at each iteration contains states from all time points within the horizon, rather than being restricted to states arising at a particular time point. Because of this difference, the active learner is able to ask queries across a range of time points and we might expect policies learned in earlier iterations to achieve non-trivial performance throughout the entire horizon. In contrast, at iteration t the policy produced by forward training is only well defined up to time t.

The complication faced by RAIL, however, compared to forward training, is that the distribution used to train $\hat{\pi}^{t+1}$ differs from the state distribution of the expert policy d_{π^*}. This is particularly true in early iterations of RAIL, since $\hat{\pi}^0$ is initialized arbitrarily. Intuitively, however, we might expect that as the iterations proceed, the unlabeled distributions used for training d_{π^t} will become similar to d_{π^*}. To see this, consider the first iteration. While $d_{\hat{\pi}^0}$ need not be at all similar to d_{π^*} overall, we know that they will agree on the initial state distribution. That is, we have that $d^1_{\hat{\pi}^0} = d^1_{\pi^*} = I$. Because of this, the policy $\hat{\pi}_1$ learned on $d_{\hat{\pi}^0}$ can be expected to agree with the expert on the first step. This implies that the states encountered after the first action of the expert and learned policy will tend to be similar. That is $d^2_{\hat{\pi}^1}$ will be similar to $d^2_{\pi^*}$. In this same fashion we might expect $d^{t+1}_{\hat{\pi}^t}$ to be similar to $d^{t+1}_{\pi^*}$ after iteration t. We now show that this intuition can be formalized in order to bound the disparity between $d_{\hat{\pi}^T}$ and d_{π^*}, which will allow us to bound the regret of the learned policy. We first state the main result, which we prove below.

Theorem 2 *Given a PAC active i.i.d. learning algorithm L_a, if RAIL is run with parameters ϵ and $\frac{\delta}{T}$ passed to L_a at each iteration, then with probability at least $1-\delta$ it will return a stationary policy $\hat{\pi}$ such that $V(\hat{\pi}) \geq V(\pi^*) - \epsilon T^3$.*

Recall that the corresponding regret for active forward training of non-stationary policies was ϵT^2. From this we see that the impact of moving from non-stationary to stationary policies in the worst case is a factor of T in the regret bound. Similarly the bound is a factor of T worse than the comparable result above for passive imitation learning, which suffered a worst-case regret of ϵT^2. From this we see that the total label complexity for RAIL required to guarantee a regret of ϵ with probability $1-\delta$ is $\sum_{t=1}^{T} N_a(\frac{\epsilon}{T^3}, \frac{\delta}{T}, d_{\hat{\pi}^{t-1}})$ compared to the above label complexity of passive learning $T \cdot N_p(\frac{\epsilon}{T^2}, \delta)$.

We first compare these quantities in the worst case. If, in each iteration, the active i.i.d. label complexity is the same as the passive complexity, then active imitation learning via RAIL can ask more queries than passive. That is, the active complexity would scale as $T \cdot N_p(\frac{\epsilon}{T^3}, \frac{\delta}{T})$ versus $T \cdot N_p(\frac{\epsilon}{T^2}, \delta)$, which is dominated by the factor of $\frac{1}{T}$ difference in the accuracy parameters. In the realizable setting with finite VC-dimension, RAIL's complexity could be a factor of T higher than passive in this worst-case scenario.

However, if across the iterations the expected active i.i.d. label complexity $N_a(\frac{\epsilon}{T^3}, \frac{\delta}{T}, d_{\hat{\pi}^{t-1}})$ is substantially better than $N_p(\frac{\epsilon}{T^3}, \frac{\delta}{T})$, then RAIL will leverage those savings. For example, in the realizable setting with finite VC-dimension, if all distributions $d_{\hat{\pi}^{t-1}}$ result in a finite disagreement coefficient, then we can get exponential savings. In particular, ignoring the dependence on δ (which is only logarithmic), we get an active label complexity of $O(T \log \frac{T^3}{\epsilon})$ versus the corresponding passive complexity of $O(\frac{T^3}{\epsilon})$.

The above analysis points to an interesting open problem. Is there an active imitation learning algorithm that can guarantee to never perform worse than passive, while at the same time showing exponential improvement in the best case?

For the proof of Theorem 2, we introduce the quantity $P^t_{\pi}(M)$, which is the probability that a policy π is consistent with a length t trajectory generated by the expert policy π^* in MDP M. It will also be useful to index the state distribution of π by the MDP M, denoted by $d_{\pi}(M)$. The main idea is to show that at iteration t, $P^t_{\hat{\pi}^t}(M)$ is not too small, meaning that the policy at iteration t mostly agrees with the expert for the first t actions. We first state two lemmas, that are useful for the final proof. First, we bound the regret of a policy in terms of $P_\pi^T(M)$.

Lemma 3 *For any policy π, if $P_\pi^T(M) \geq 1-\epsilon$, then $V(\pi) \geq V(\pi^*) - \epsilon T$.*

Proof Let Γ^* and Γ be all state-action sequences of length T that are consistent with π^* and π respectively. If $R(\mathcal{T})$ is the total reward for a sequence $\mathcal{T}$ then we get the following

$$\begin{aligned}
V(\pi) &= \sum_{\mathcal{T}\in\Gamma} \Pr(\mathcal{T} \mid M,\pi) R(\mathcal{T}) \\
&\geq \sum_{\mathcal{T}\in\Gamma\cap\Gamma^*} \Pr(\mathcal{T} \mid M,\pi) R(\mathcal{T}) \\
&= \sum_{\mathcal{T}\in\Gamma^*} \Pr(\mathcal{T} \mid M,\pi^*) R(\mathcal{T}) - \sum_{\mathcal{T}\in\Gamma^*-\Gamma} \Pr(\mathcal{T} \mid M,\pi^*) R(\mathcal{T}) \\
&= V(\pi^*) - \sum_{\mathcal{T}\in\Gamma^*-\Gamma} \Pr(\mathcal{T} \mid M,\pi^*) R(\mathcal{T}) \\
&\geq V(\pi^*) - T\cdot \sum_{\mathcal{T}\in\Gamma^*-\Gamma} \Pr(\mathcal{T} \mid M,\pi^*) \\
&\geq V(\pi^*) - \epsilon T.
\end{aligned}$$

The last two inequalities follow since the reward for a sequence must be no more than T, and due to our assumption about $P_\pi^T(M)$. ■

Next, we show how the value of $P_\pi^t(M)$ changes across one iteration of RAIL. We show that if we learn a policy $\hat{\pi}$ on state distribution $d_\pi(M)$ of policy π whose error rate $e_{\pi^*}(\hat{\pi}, d_\pi(M))$ (see Section 3.3) w.r.t. to the expert's policy π^* is no more than ϵ, then $P_{\hat{\pi}}^{t+1}(M)$ is at least as large as $P_\pi^t(M) - T\epsilon$. When π and $\hat{\pi}$ correspond to policies learned at iteration t and $(t+1)$ respectively, then Lemma 4 describes change in the value of $P_\pi^t(M)$ across one iteration.

Lemma 4 *For any policies π and $\hat{\pi}$ and $1 \leq t < T$, if $e_{\pi^*}(\hat{\pi}, d_\pi(M)) \leq \epsilon$, then $P_{\hat{\pi}}^{t+1}(M) \geq P_\pi^t(M) - T\epsilon$.*

Proof We define $\hat{\Gamma}$ to be all sequences of state-action pairs of length $t+1$ that are consistent with $\hat{\pi}$. Also define Γ to be all length $t+1$ state-action sequences that are consistent with π on the first t state-action pairs (so need not be consistent on the final pair). We also define M' to be an MDP that is identical to M, except that the transition distribution of any state-action pair (s,a) is equal to the transition distribution of action $\pi(s)$ in state s. That is, all actions taken in a state s behave like the action selected by π in s.

We start by arguing that if $e_{\pi^*}(\hat{\pi}, d_\pi(M)) \leq \epsilon$ then $P_{\hat{\pi}}^{t+1}(M') \geq 1 - T\epsilon$, which relates our error assumption to the MDP M'. To see this, note that for MDP M', all policies, including π^*, have state distribution given by d_π. Thus by the union bound $1 - P_{\hat{\pi}}^{t+1}(M') \leq \sum_{i=1}^{t+1} \epsilon_i$, where ϵ_i is the error of $\hat{\pi}$ at predicting π^* on distribution d_π^i. This sum is bounded by $T\epsilon$

since $e_{\pi^*}(\hat{\pi}, d_{\pi}(M)) = \frac{1}{T}\sum_{i=1}^{T}\epsilon_i$. Using this fact we can now derive the following

$$\begin{aligned}
P_{\hat{\pi}}^{t+1}(M) &= \sum_{\mathcal{T}\in\hat{\Gamma}} \Pr(\mathcal{T} \mid M, \pi^*) \\
&\geq \sum_{\mathcal{T}\in\Gamma\cap\hat{\Gamma}} \Pr(\mathcal{T} \mid M, \pi^*) \\
&= \sum_{\mathcal{T}\in\Gamma} \Pr(\mathcal{T} \mid M, \pi^*) - \sum_{\mathcal{T}\in\Gamma-\hat{\Gamma}} \Pr(\mathcal{T} \mid M, \pi^*) \\
&= P_{\pi}^{t}(M) - \sum_{\mathcal{T}\in\Gamma-\hat{\Gamma}} \Pr(\mathcal{T} \mid M, \pi^*) \\
&= P_{\pi}^{t}(M) - \sum_{\mathcal{T}\in\Gamma-\hat{\Gamma}} \Pr(\mathcal{T} \mid M', \pi^*) \\
&\geq P_{\pi}^{t}(M) - \sum_{\mathcal{T}\notin\hat{\Gamma}} \Pr(\mathcal{T} \mid M', \pi^*) \\
&\geq P_{\pi}^{t}(M) - (1 - P_{\hat{\pi}}^{t+1}(M')) \\
&\geq P_{\pi}^{t}(M) - T\epsilon.
\end{aligned}$$

The equality of the fourth line follows because Γ contains all sequences whose first t actions are consistent with π with all possible combinations of the remaining action and state transition. Thus, summing over all such sequences yields the probability that π^* agrees with the first t steps. The equality of the fifth line follows because $\Pr(\mathcal{T} \mid M, \pi^*) = \Pr(\mathcal{T} \mid M', \pi^*)$ for any $\mathcal{T}$ that is in Γ and for which π^* is consistent (has non-zero probability under π^*). The final line follows from the above observation that $P_{\hat{\pi}}^{t+1}(M') \geq 1 - T\epsilon$. ■

We can now complete the proof of the main theorem.

Proof [Proof of Theorem 2] Using failure parameter $\frac{\delta}{T}$ in the call to L_a in each iteration of RAIL ensures that with at least probability $(1-\delta)$ that for all $1 \leq t \leq T$, we will have $e_{\pi^*}(\hat{\pi}^t, d_{\hat{\pi}^{t-1}}(M)) \leq \epsilon$, where $d_{\hat{\pi}^0}(M)$ denotes the state distribution of the initial policy $\hat{\pi}^0$. This can be easily shown using the union bound. Next, we show using induction that for $1 \leq t \leq T$, we have $P_{\hat{\pi}^t}^{t} \geq 1 - tT\epsilon$. As a base case for iteration $t = 1$, we have $P_{\hat{\pi}^1}^{1} \geq 1 - T\epsilon$, since the the error rate of $\hat{\pi}^1$ relative to the initial state distribution at time step $t = 1$ is at most $T\epsilon$ (this is the worst case when all errors are committed at time step 1). Assume that the inequality holds for $t = k$, i.e., $P_{\hat{\pi}^k}^{k} \geq 1 - kT\epsilon$. Consider $\hat{\pi}^{k+1}$ trained on $d_{\hat{\pi}^k}(M)$. By the union bound argument above, we know that $e_{\pi^*}(\hat{\pi}^{k+1}, d_{\hat{\pi}^k}(M)) \leq \epsilon$. Hence, $\hat{\pi}^{k+1}$ and $\hat{\pi}^k$ satisfy the precondition of Lemma 4. Therefore we have

$$\begin{aligned}
P_{\hat{\pi}^{k+1}}^{k+1} &\geq P_{\hat{\pi}^k}^{k} - T\epsilon && \text{(by Lemma 4)} \\
&\geq 1 - kT\epsilon - T\epsilon && \text{(by inductive argument)} \\
&\geq 1 - (k+1)T\epsilon.
\end{aligned}$$

Hence, for $1 \leq t \leq T$, we have $P_{\hat{\pi}^t}^{t} \geq 1 - tT\epsilon$. In particular, when $t = T$, we have $P_{\hat{\pi}^T}^{T} \geq 1 - T^2\epsilon$. Combining this with Lemma 3 completes the proof. ■

4.3 Agnostic Case

Above we considered the realizable setting, where the expert's policy was assumed to be in a known hypothesis class H. In the agnostic case, we do not make such an assumption. The learner still outputs a hypothesis from a class H, but the unknown policy is not necessarily in H. The agnostic i.i.d. PAC learning setting is defined similarly to the realizable setting, except that rather than achieving a specified error bound of ϵ with high probability, a learner must guarantee an error bound of $\inf_{\pi \in H} e_f(\pi, D_{\mathcal{X}}) + \epsilon$ with high probability (where f is the target), where $D_{\mathcal{X}}$ is the unknown data distribution. That is, the learner is able to achieve close to the best possible accuracy given class H. In the agnostic case, it has been shown that exponential improvement in label complexity with respect to $\frac{1}{\epsilon}$ is achievable when $\inf_{\pi \in H} e_f(\pi, D_{\mathcal{X}})$ is relatively small compared to ϵ (Dasgupta, 2011). Further, there are many empirical results for practical active learning algorithms that demonstrate improved label complexity compared to passive learning.

It is straightforward to extend our above results for non-stationary and stationary policies to the agnostic case by using agnostic PAC learners for L_p and L_a. Here we outline the extension for RAIL. Note that the RAIL algorithm will call L_a using a sequence of unlabeled data distributions, where each distribution is of the form d_π for some $\pi \in H$ and each of which may yield a different minimum error given H. For this purpose, we define $\epsilon^* = \sup_{\pi \in H} \inf_{\hat{\pi} \in H} e_{\pi^*}(\hat{\pi}, d_\pi)$ to be the minimum generalization error achievable in the worst case considering all possible state distributions d_π that RAIL might possibly encounter. With minimal changes to the proof of Theorem 1, we can get an identical result, except that the regret is $(\epsilon^* + \epsilon)T^3$ rather than just ϵT^3. A similar change in regret holds for passive imitation learning. This shows that in the agnostic setting we can get significant improvements in label complexity via active imitation learning when there are significant savings in the i.i.d. case.

5. RAIL-DA: A Practical Variant of RAIL

Despite the theoretical guarantees, there are at least two potential drawbacks of the RAIL algorithm from a practical perspective. First, RAIL does not share labeled data across iterations which is potentially wasteful in practice, though important for our analysis. In practice, we might expect that aggregating labeled data across iterations would be beneficial due to the larger amount of data. This is somewhat confirmed by the empirical success of the DAGGER algorithm (Ross et al., 2011) and motivates evaluating the use of data aggregation within RAIL. Incorporating data aggregation into RAIL, however, complicates the theoretical analysis, which we leave for future work. The second practical inefficiency of RAIL is that the unlabeled state distributions used at early iterations may be quite different from d_{π^*}. In particular, the state distribution of policy $\hat{\pi}^t$, which is used to train the policy at iteration $t+1$, is only guaranteed to be close to the expert's state distribution for the first t time steps and can (in the worst case) differ arbitrarily at later times. Thus, early iterations may focus substantial query effort on parts of the state space that are not the most relevant for learning π^*.

Algorithm 3 RAIL$^+$

```
Input: $L_0$, $n$, $AccumData$, $N$, $K$
        ▷ $L_0$ : initial set of labeled data
        ▷ $n$ : no. of queries per iteration
        ▷ $AccumData$ : accumulate data across iterations or not
        ▷ $N$ : committee size for DWQBC
        ▷ $K$ : # trajectories used to generate unlabeled data
Output: stationary policy $\hat{\pi}$

 1: Initialize $L = L_0$
 2: while query budget remaining do
 3:     $U$ = SampleUnlabeledData($K$,$L$)     ▷ generates pool of unlabeled data
 4:     if !$AccumData$ then
 5:         Initialize $L = L_0$
 6:     end if
 7:     for $i = 1$ to $n$ do     ▷ select $n$ queries from pool $U$
 8:         $s$ = DWQBC($L$,$U$,$N$)     ▷ density-weighted QBC is used as active i.i.d. learner
 9:         $L = L \cup \{(s,$ Label(s)$)\}$     ▷ obtain label from expert
10:     end for
11: end while
12: return $\hat{\pi}$ = SupervisedLearn($L$)
```

We now describe a parameterized practical instantiation of RAIL used in our experiments, which is intended to address the above issues and also specify certain other implementation details. Algorithm 3 gives pseudocode for this algorithm, which we call RAIL$^+$ to distinguish it from the idealized version of RAIL in our analysis. In Section 6, we will compare different instances of RAIL$^+$, including parameterizations corresponding to pure RAIL and RAIL-DA (for data aggregation), which is the primary algorithm in our empirical study. We note that our description assumes the use of a pool-based active learner, which is a common active learning setting, meaning that the learner requires as input a pool of unlabeled examples U that represents the unlabeled target distribution.

5.1 Data Aggregation and Incremental Querying

The first major difference compared to RAIL is that RAIL$^+$ can aggregate data across iterations when the Boolean parameter *AccumData* is set to true. In this case, during each iteration the newly labeled data is added to the set of labeled data from previous iterations (lines 7-10). Otherwise, the labeled data from previous iterations is discarded after generating unlabeled data U (lines 4-6).

The second major difference is that RAIL$^+$ may ask fewer queries per iteration than RAIL as specified by the parameter n. RAIL corresponds to a version of RAIL$^+$ that does not use data aggregation and has $n = N_a$. Because RAIL$^+$ can aggregates data, it opens up the possibility of asking only a small number of queries per iteration ($n \ll N_a$) and hence behaving more incrementally. This is reflected in the main loop of Algorithm 3

Algorithm 4 Density-Weighted Query-By-Committee Algorithm

```
1: procedure DWQBC($L$,$U$,$N$)
2:     $C$ = SampleCommittee($N$,$L$)        ▷ committee represents posterior over policies
3:     $\hat{d}_L$ =EstimateDensity($U$)        ▷ estimate density of states in $U$ (see text)
4:     $s^* = \text{argmax}\{VE(s, C) * \hat{d}_L(s) : s \in U\}$        ▷ selection heuristic (see text)
5:     return $s^*$
6: end procedure
```

(lines 2-11). Each iteration starts with the current set of labeled examples L, which have been accumulated across all previous iterations by RAIL$^+$. This set of examples is used to generate a pool U of unlabeled examples/states (see details below) intended to represent the unlabeled target distribution. A pool-based active learner, DWQBC (explained later), is then called n times to select n queries from this pool. Each query is labeled by the expert and added to the growing set of labeled training data L. After the n queries have been issued and L is updated, the next iteration begins.

This incremental version of RAIL allows for rapid updating of the unlabeled state distributions used for learning (represented via U) and prevents RAIL from using its query budget on earlier less accurate distributions. In our experiments, we find that using $n = 1$ is most effective compared to larger values, which facilitates the most rapid update of the distribution. In this case, each query selected by the active i.i.d. learner is based on the most up-to-date state distribution, which takes all prior data into account. We refer to this best performing variant of RAIL$^+$ with $n = 1$ and data aggregation as RAIL-DA.

An interesting variation to RAIL$^+$ is when queries take a non-trivial amount of real-time to answer and there are k experts available that can answer queries in parallel. In this case, it can be beneficial to ask k simultaneous queries per iteration in order to reduce the amount of real-time required to learn a policy. The problem of selecting k such queries is known as *batch active learning*, and a variety of approaches are available for the i.i.d. setting (Brinker, 2003; Xu et al., 2007; Hoi et al., 2006a,b; Guo and Schuurmans, 2008; Azimi et al., 2012). An advantage of our reduction-based approach to active imitation learning is that we can directly plug in the i.i.d. batch active learner in our framework without requiring any other changes to be made.

5.2 Density Weighted QBC

Since it is important that the active i.i.d. learner be sensitive to the unlabeled data distribution, we choose a density-weighted learning algorithm. In particular, we use density-weighted query-by-committee (McCallum and Nigam, 1998) in our implementation. Given a set of labeled data L and unlabeled data U, this approach first uses bootstrap aggregation on L in order to generate a policy committee for query selection (Algorithm 4, line 2). In our experiments we use a committee of size 5. The approach then computes a density estimator over the unlabeled data U (line 3). In our implementation we use a simple distance based binning approach to density estimation, though more complex approaches could be used. The selected query is the state that maximizes the product of state density and committee disagreement (line 4). As a measure of disagreement we use the entropy of the vote distribution (Dagan and Engelson, 1995) (denoted as VE), which is a common choice.

Algorithm 5 Procedure SampleUnlabeledData

```
1: procedure SampleUnlabeledData($K$,$L$)
2:     $C$ = SampleCommittee($K$,$L$)    ▷ committee represents posterior over policies
3:     $U = \{\}$    ▷ initialize multi-set of unlabeled data
4:     for $\pi \in C$ do
5:         $S$ = SimulateTrajectory($\pi$)    ▷ states generated on trajectory of $\pi$
6:         $U = U \cup S$
7:     end for
8:     return $U$
9: end procedure
```

Intuitively, the selection heuristic of DWQBC attempts to trade off the uncertainty about what to do at a state (measured by VE) with the likelihood that the state is relevant to learning the target policy (measured by the density).

5.3 Bayesian Learner

The final choice we make while implementing RAIL$^+$ (and hence RAIL-DA) is again motivated by the goal of arriving at an accurate unlabeled data distribution as quickly as possible. Recall that at iteration $t + 1$, RAIL learns using an unlabeled data distribution $d_{\hat{\pi}^t}$, where $\hat{\pi}^t$ is a point estimate of the policy based on the labeled data from iteration t. In order to help improve the accuracy of this unlabeled distribution (with respect to d_{π^*}), instead of using a point estimate, we adopt a Bayesian approach in RAIL$^+$. In particular, at iteration t let L be the set of state-action pairs collected from the previous iteration (or from all previous iterations if data is accumulated). We use this to define a posterior $P(\hat{\pi}|L)$ over policies in our policy class H. This distribution, in turn, defines a posterior unlabeled state distribution $d_L = \mathbb{E}_{\hat{\pi}\sim P(\hat{\pi}|\mathcal{L})}[d_{\hat{\pi}}(s)]$ that RAIL$^+$ effectively uses in place of $d_{\hat{\pi}^t}$ as used in RAIL. Note that we can sample states from d_L by first sampling a policy $\hat{\pi}$ and then sampling a state from $d_{\hat{\pi}}$, all of which can be done without interaction with the expert.

Our implementation of this idea uses bootstrap aggregation (Breiman, 1996) in order to approximate d_L by an unlabeled data pool U via a call to the procedure SampleUnlabeledData in Algorithm 3 (line 3). Our implementation assumes a class of linear parametric policies with a zero-mean Gaussian prior over the parameters. The procedure first uses bootstrap aggregation to approximate sampling a set of policies from the posterior (Algorithm 5, line 2) forming a "committee" C of K policies. We view C as an empirical distribution representing the posterior over policies. In particular, each policy is the result of first generating a bootstrap sample of the current labeled data and then calling a supervised learner on the sampled data. Each member of the committee is then simulated to form a state trajectory, and the states on those trajectories are aggregated to produce the unlabeled data pool U (Algorithm 5, lines 4-7). Our implementation uses $K = 5$.

From a theoretical perspective, the use of a Bayesian classifier does not impact the validity of RAIL's performance guarantee provided that the Bayesian approach provides PAC guarantees. In fact, early theoretical work in active learning (Freund et al., 1997) used exactly this type of assumption in their analysis of the query-by-committee algorithm.

Algorithm 6 Procedure SampleCommittee

```
1: procedure SampleCommittee($K$,$L$)
2:     $C = \{\}$    ▷ initialize the committee
3:     for $i = 1$ to $K$ do
4:         $L' =$ BootstrapSample($L$)    ▷ create a bootstrap sample of $L$
5:         $\pi' =$ SupervisedLearn($L'$)    ▷ learn a classifier(policy) using $L'$
6:         $C = C \cup \pi'$
7:     end for
8:     return $C$
9: end procedure
```

In practice, d_L is a significantly more useful estimate of d_{π^*} than the point estimate with respect to learning a policy. This is because it places more weight on states that are more frequently visited by policies drawn from the posterior rather than just a single policy. As an extreme example of the advantage of using d_L in practice, consider active imitation learning in an MDP with deterministic dynamics and a single start state. At iteration t, RAIL will use the state distribution of the point estimate $\hat{\pi}^{t-1}$, which for our assumed MDP will be uniform over the deterministic state sequence generated by $\hat{\pi}^{t-1}$. Thus, active learning will place equal emphasis on learning among that set of states. This is despite the fact that, in early iterations, we should expect that states appearing later in the trajectory are less likely to be relevant to learning π^*. This is because inaccuracies in $\hat{\pi}^{t-1}$ lead to error propagation as the trajectory unfolds. In contrast, d_L will weigh states according to the trajectories produced by all policies, weighted by the posterior. The practical effect is a non-uniform distribution over states in those trajectories, roughly weighted by how many policies visit the states. Thus, states at the tail end of trajectories in early iterations will generally carry very little weight, since they are only visited by one or a small number of policies.

6. Experiments

We conduct our empirical evaluation on five domains: 1) Cart-pole, 2) Bicycle, 3) Wargus, 4) Driving, and 5) NETtalk. Below we first describe the details of these domains. We then present various experiments that we performed in these domains. In the first experiment, we study the impact of data aggregation and query size on the performance of RAIL$^+$ from Algorithm 3. For this we compare several parameter settings, varying from versions that are close to pure RAIL to RAIL-DA ($n = 1$ with data aggregation). We show that versions closer to RAIL-DA that aggregate data and ask queries incrementally are more effective in practice. In particular, we show that RAIL-DA (which aggregates data and asks only one query per iteration) is the most effective parameterization. Next, we evaluate the impact of another implementation choice discussed in Section 5, the choice of base active learner in RAIL$^+$. We show that a density-weighted base active learner leads to better performance in practice than using other base active learners that ignore the data distribution. Once we have shown that RAIL-DA with a density-weighted base active learner is the best, we compare it with a number of baseline approaches to active imitation learning in our last set of experiments. Finally, we provide some overall observations.

For all the learners in the experiments that are presented in this section, we employed the SimpleLogistic classifier from Weka (Hall et al., 2009) to learn policies over the set of features that were provided for each domain.

6.1 Domain Details

In this subsection, we give the details of all the domains used in our experiments.

6.1.1 Cart-Pole

Cart-pole is a well-known RL benchmark. In this domain, there is a cart on which rests a vertical pole. The objective is to keep the attached pole balanced by applying left or right force to the cart. An episode ends when either the pole falls or the cart goes out of bounds. There are two actions, left and right, and four state variables $(x, \dot{x}, \theta, \dot{\theta})$ describing the position and velocity of the cart and the angle and angular velocity of the pole. We made slight modifications to the usual setting where we allow the pole to fall down and become horizontal and the cart to go out of bounds (we used [-2.4, 2.4] as the in bounds region). We let each episode run for a fixed length of 5000 time steps. This opens up the possibility of generating several "undesirable" states where either the pole has fallen or the cart is out of bounds that are rarely or never generated by the expert's state distribution.

For all the experiments in cart-pole, the learner's policy is represented via a linear logistic regression classifier using features of state-action pairs where features correspond to state variables. The expert policy was a hand-coded policy that can balance the pole indefinitely. For each learner, we ran experiments from 150 random initial states close to the equilibrium start state $((x, \dot{x}, \theta, \dot{\theta}) = (0.0, 0.0, 0.0, 0.0))$. For each start state a policy is learned and a learning curve is generated measuring the performance as function of number of queries posed to the expert. To measure performance, we use a reward function (unknown to the learner) that gives $+1$ reward for each time step where the pole is kept balanced and the cart is within bounds and -1 otherwise. The final learning curve is the average of the individual curves.

6.1.2 Bicycle Balancing

This domain is a variant of the RL benchmark of bicycle balancing and riding (Randløv and Alstrøm, 1998). The goal is to balance a bicycle moving at a constant speed for 1000 time steps. If the bicycle falls, it remains fallen for the rest of the episode. Similar to the cart-pole domain, in bicycle balancing there is a huge possibility of spending significant amount of time in "undesirable" states where the bicycle has fallen down. The state space is described using nine variables $(\omega, \dot{\omega}, \theta, \dot{\theta}, \psi, x_f, y_f, x_b, y_b)$, where ω and $\dot{\omega}$ are the vertical angle and angular velocity of the bicycle, θ and $\dot{\theta}$ are the angle and angular velocity of the handlebar, ψ is the angle of the bicycle to the goal, x_f and y_f are x and y coordinates of the front tire and x_b and y_b are x and y coordinates of the rear tire of the bicycle. There are five possible actions $A = \{(\tau = 0, v = -0.02), (\tau = 0, v = 0), (\tau = 0, v = 0.02), (\tau = 2, v = 0), (\tau = -2, v = 0)\}$, where the first component is the torque applied to the handlebar and the second is the displacement of the rider.

As in the cart-pole domain, for all experiments in bicycle balancing, the learner's policy is represented as a linear logistic regression classifier over features of state-action pairs. A feature vector for a state-action pair is defined as follows: Given a state s, a vector consisting of following 20 basis functions is computed:

$$(1,\ \omega,\ \dot{\omega},\ \omega^2,\ \dot{\omega}^2,\ \omega\dot{\omega},\ \theta,\ \dot{\theta},\ \theta^2,\ \dot{\theta}^2,\ \theta\dot{\theta},\ \omega\theta,\ \omega\theta^2,\ \omega^2\theta,\ \psi,\ \psi^2,\ \psi\theta,\ \bar{\psi},\ \bar{\psi}^2,\ \bar{\psi}\theta)^T,$$

where $\bar{\psi} = \pi - \psi$ if $\psi > 0$ and $\bar{\psi} = -\pi - \psi$ if $\psi < 0$. This vector of basis functions is repeated for each of the 5 actions giving a feature vector of length 100. The expert policy was hand-coded and can balance the bicycle for up to 26K time steps. We used a similar evaluation procedure as for cart-pole where we generated 150 random start states and for each start state, a policy was learned using each of the learning algorithms and a learning curve was generated measuring total reward as function of number of queries posed to the expert. We give a $+1$ reward for each time step where the bicycle is kept balanced and a -1 reward otherwise.

6.1.3 Wargus

We consider controlling a group of 5 friendly close-range military units against a group of 5 enemy units in the real-time strategy game Wargus, similar to the setup used by Judah et al. (Judah et al., 2010). The objective is to win the battle while minimizing the loss in total health of friendly units. The set of actions available to each friendly unit is to attack any one of the remaining units present in the battle (including other friendly units, which is always a bad choice). In our setup, we allow the learner to control one of the units throughout the battle, whereas the other friendly units are controlled by a fixed "reasonably good" policy. This situation would arise when training the group via coordinate ascent on the performance of individual units. The expert policy corresponds to the same policy used by the other units. Note that poor behavior from even a single unit generally results in a huge loss.

The learner's policy is represented using 27 state-action features that capture different information about the current battle situation such as the distance between the friendly agent and the target of attack, whether the target is already under attack by other friendly units, health of the target relative to friendly unit, whether the target is actually a friendly unit, etc. Providing full demonstrations in real time in such tactical battles is very difficult for human players and quite time consuming if demonstrations are done in slow motion, which motivates state-based active learning for this domain. For experiments, we designed 21 battle maps differing in the initial unit positions, using 5 for training and 16 for testing. We report results in the form of learning curves showing the performance metric as a function of number of queries posed to the expert. We use the difference in the total health of friendly and enemy units at the end of the battle as the performance metric (which is positive for a win). Due to the slow pace of the experiments running on the Wargus infrastructure, we average results across at most 20 trials.

6.1.4 Driving Domain

The driving domain is a traffic navigation problem often used as a test domain in the imitation learning literature (Abbeel and Ng, 2004). This domain was also the main test

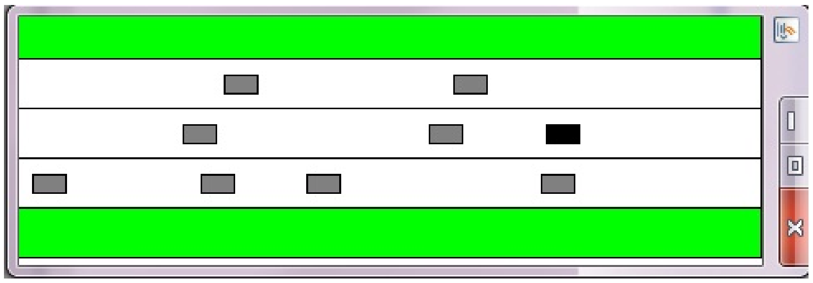


Figure 1: Screenshot of the driving simulator.

bed used to evaluate the confidence based autonomy (CBA) learner in prior work (Chernova and Veloso, 2009). Here we evaluate RAIL-DA on a particular implementation of the driving domain used by Cohn et al. (Cohn et al., 2011). In this domain, the goal is to successfully navigate a car through traffic on a busy five lane highway. The highway consists of three traffic lanes and two shoulder lanes (see Figure 1). The learner controls the black car, which moves at a constant speed. The other cars move at a randomly chosen continuous-valued constant speed, and they don't change lanes. At each discrete time step, the learner controls the car by taking one of the three actions: 1) *Left*, which moves the car to the adjacent left lane, 2) *Right*, which moves the car to the adjacent right lane, and 3) *Stay*, which keeps the car in the current lane. The agent is allowed to drive on the shoulder lane but cannot move off the shoulder lanes.

The learner's policy is represented as a linear logistic regression classifier that maps a given state to one of the three actions. The state space is represented using 68 features. The first 5 features are binary features that specify the learner's current lane. The next 3 binary features specify whether the learner is colliding with, tailgating or trailing another car. The remaining 60 features, consisting of three parts, one for the learner's current lane and two for the two adjacent lanes, which are binary features that specify whether the learner's car will collide with or pass another car in $2X$ time steps, where X ranges from 0 to 19. This captures the agent's view of the traffic while taking car velocities into account.

To conduct experiments in the driving domain, we carefully designed a reward function that induces good driving behavior and used Sarsa(λ) (Sutton and Barto, 1998) with linear function approximation to learn a policy to serve as the expert policy. The expert policy uses the same set of 68 features as used by the agent for value function approximation. For each learner, we ran 100 different learning trials where during each trial the learner is allowed to pose a maximum of 500 (1000 in Experiment 1) queries to the expert and learn from it. For each trial, a learning curve is generated that measures the performance as a function of the number of queries posed to the expert. To measure performance, after each query is posed and the policy is updated, the updated policy is allowed to navigate the car for 1000 time steps in a test episode, and the total reward is recorded. The final performance measure is the average total reward per episode measured over 500 test episodes. The final learning curve is the average over all 100 trials.

6.1.5 Structured Prediction

We evaluate RAIL-DA on two structured prediction tasks, stress prediction and phoneme prediction, both based on the NETtalk data set (Dietterich et al., 2008). In stress prediction, given a word, the goal is to assign one of the 5 stress labels to each letter of the word in left-to-right order so that the word is pronounced correctly. The output labels are '2' (strong stress), '1' (medium stress), '0' (light stress), '$<$' (unstressed consonant, center of syllable to the left), and '$>$' (unstressed consonant, center of syllable to the right). In phoneme prediction, the task is to assign one of the 51 phoneme labels to each letter of the word. It is straightforward to view structured prediction as imitation learning (see for example Ross et al., 2011,Daumé et al., 2009) where at each time step (letter location), the learner has to execute the correct action (i.e., predict correct label) given the current state. The state consists of features describing the input (the current letter and its immediate neighbors) and the previous L predictions made by the learner (the prediction context). In our experiments, we use $L = 1, 2$.

The NETtalk data set consists of 2000 words divided into 1000 training words and 1000 test words. Each method is allowed to select a state located on any of the words in the training data and pose it as a query. The expert reveals the correct label at that location. We use character accuracy as a measure of performance. The details of how in each learning trial RAIL-DA and other baselines select a query from the set of training words will be described when we present our results in the following subsections. We report final performance in the form of learning curves averaged across 50 learning trials.

6.2 Experiment 1: Evaluation of the Effects of Data Aggregation and Query Size

We compare different versions of RAIL$^+$, by varying the parameters *AccumData* and n in Algorithm 3, in order to observe the impact of data aggregation and query size. We use the notation *RAIL$^+$-n-DA* for versions that aggregate data and ask n queries per iteration and use *RAIL$^+$-n* to denote variants that ask n queries per iteration without data aggregation. Note that RAIL$^+$-1-DA is the same as RAIL-DA and that RAIL$^+$-n with a large value of n corresponds to the original version of RAIL from our analysis. Note that all these variants use the DWQBC active i.i.d. learner as described in Section 5.

For this experiment, we focus on three domains: 1) Cart-Pole, 2) Bicycle Balancing, and 3) Driving Domain. In each domain, we evaluated RAIL$^+$-n-DA and RAIL$^+$-n for values of n starting at $n = 1$ in increments until some maximum value. The maximum value of n in each domain was selected to be a value where i.i.d. active learning from the true expert state distribution reliably converged to a near perfect policy. For each RAIL$^+$ variant and domain, we show the averaged learning curve as described earlier.

Figure 2 shows the results of the experiment. We first discuss results in the Bicycle domain due to more discernible trends in this domain. Figure 2(b) shows results of the experiment in the Bicycle domain. First, observe the impact of data aggregation by comparing each pair RAIL$^+$-n and RAIL$^+$-n-DA. Clearly, RAIL$^+$-n-DA is significantly better than RAIL$^+$-n for each value of n, indicating that it is generally better to aggregate data across iterations in this domain. This is likely explained by the fact that the use of aggregation allows for the algorithms to learn from more data at later iterations. This is

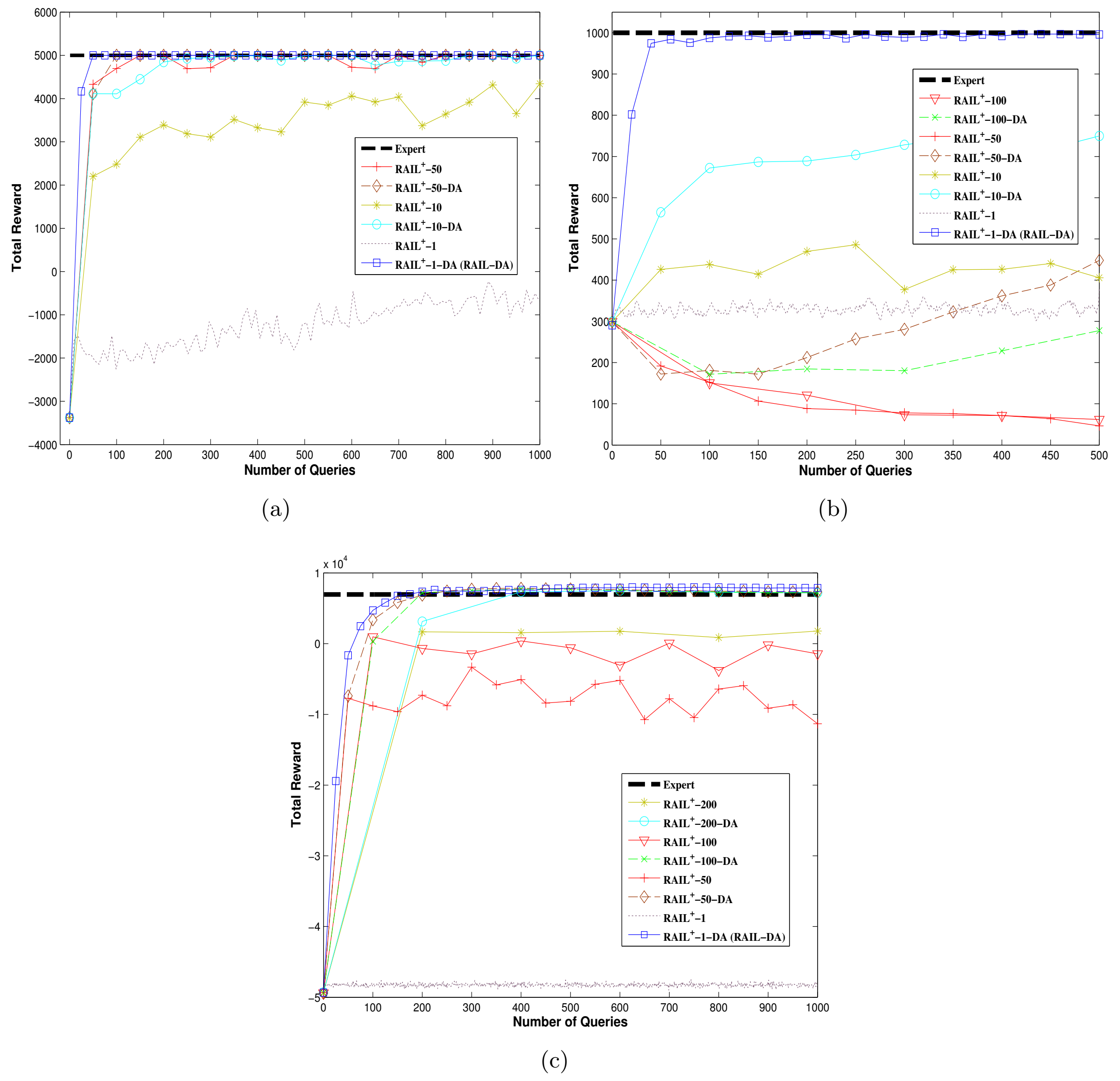


Figure 2: Evaluation of the effects of data aggregation and query size in RAIL$^+$: (a) Cart-pole (b) Bicycle balancing (c) Driving domain.

particularly important for small values of n, such as $n = 1$ and $n = 10$, where without aggregation each iteration is learning from only a small amount of data (the small amount of initial data + n).

Now consider the impact of varying n with the use of aggregation in Bicycle. The clear trend is that when aggregation is being used the performance improves significantly as n decreases from $n = 100$ to $n = 1$. That is, the more incremental variants of RAIL$^+$ dominate, with $n = 1$ (i.e. the RAIL-DA algorithm) dominating all others by a large margin. The main reason for this particularly striking trend is that in Bicycle the distributions being learned from in early iterations are quite distant from that of the expert. In particular, the early iterations involve learning from state distributions of policies that result in early crashes, and hence many useless states from the point of view of learning. This means that when using $n = 100$, the first 100 queries are selected from such a distribution and not much is learned. In fact, we see a downward trend, indicating that learning from those distributions hurts performance. On the other hand, as n decreases fewer queries are spent on those early inaccurate distributions, and the data from a small number of queries per iteration accumulates, resulting in policies that have better state distributions to learn from. Indeed, for $n = 1$, only one query is asked for each distribution, and we see very rapid improvement until reaching expert performance after just over 50 queries.

Figure 2(a) shows results for Cart-Pole. The target concept in this domain is simpler, and hence the learning curves improve more quickly compared to Bicycle. However, we see the same general trends as for Bicycle. Aggregation is beneficial and we get the best performance for small values of n when using aggregation. Again we see that RAIL$^+$-1-DA (i.e., RAIL-DA) is the top performer. These same trends are observed in Figure 2(c) for Driving.

Overall, these experiments provide strong evidence that in practice aggregation is an important enhancement to RAIL$^+$ over RAIL, and that more incremental variants are preferable. Thus, for the remainder of the paper we will use the RAIL-DA algorithm which uses aggregation and $n = 1$.

6.3 Experiment 2: Evaluation of RAIL-DA with Different Base Active Learners

In Section 5, we mentioned that it is important that the active i.i.d. learner used with RAIL be sensitive to the unlabeled data distribution. To test this hypothesis, we conducted experiments to study the effects of using active i.i.d. learners that take data distribution into consideration against active learners that ignore the data distribution altogether in RAIL-DA. We compare the performance of three different versions of RAIL-DA: 1) *RAIL-DA*, the RAIL-DA algorithm from the previous experiment that uses density-weighted QBC as the base active learner, 2) *RAIL-DA-QBC*, RAIL-DA but with density-weighted QBC replaced with the standard QBC (without density weighting), and 3) *RAIL-DA-RAND*, which uses random selection of unlabeled data points.

Figure 3 shows the performance of the three versions of RAIL-DA on our first four test domains. We see that, in Cart-Pole, Bicycle and Wargus, RAIL-DA performs better than both RAIL-DA-QBC and RAIL-DA-RAND. This shows that it is critical for the active i.i.d. learner to exploit the state density information that is estimated by RAIL-DA at

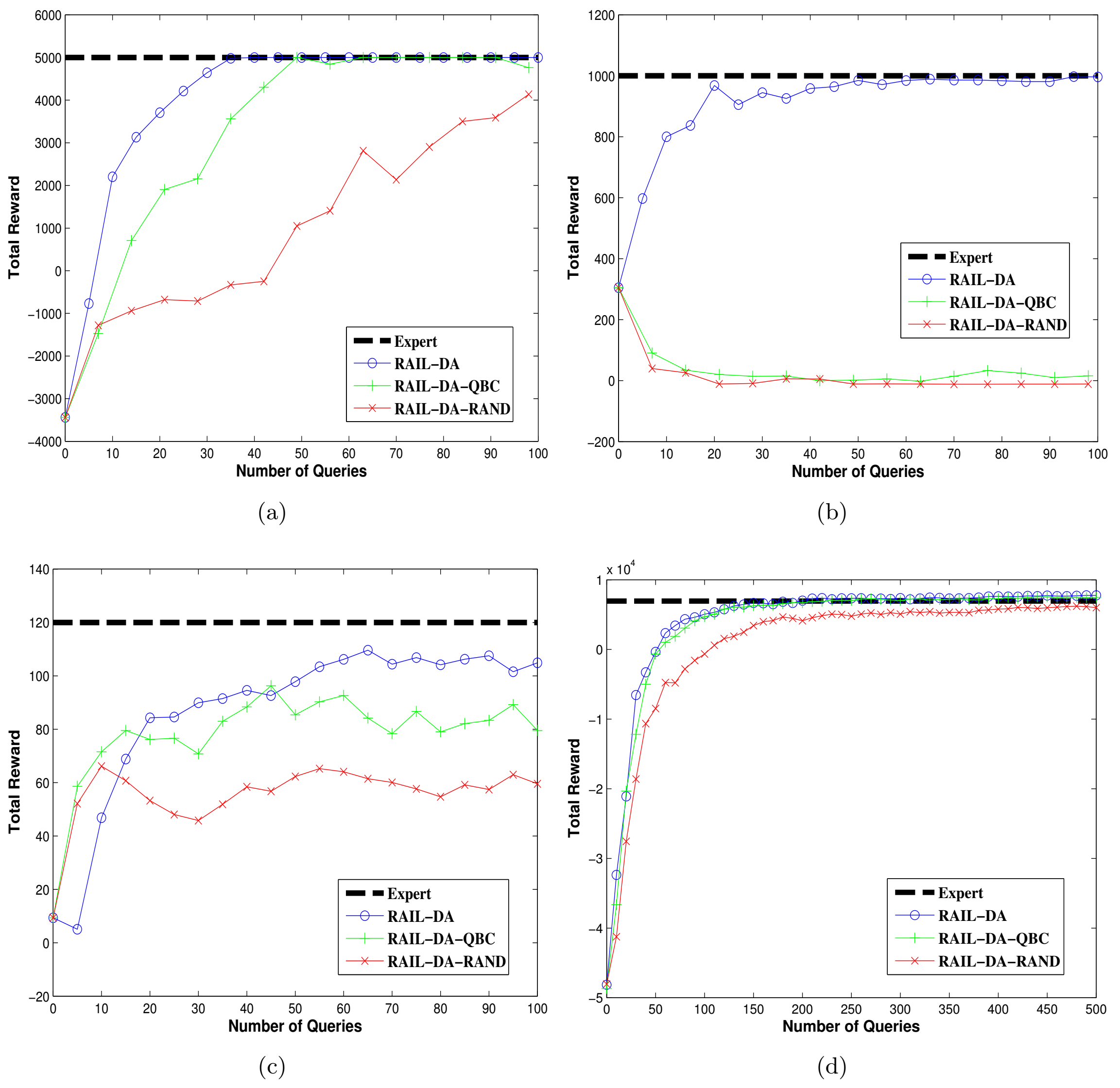
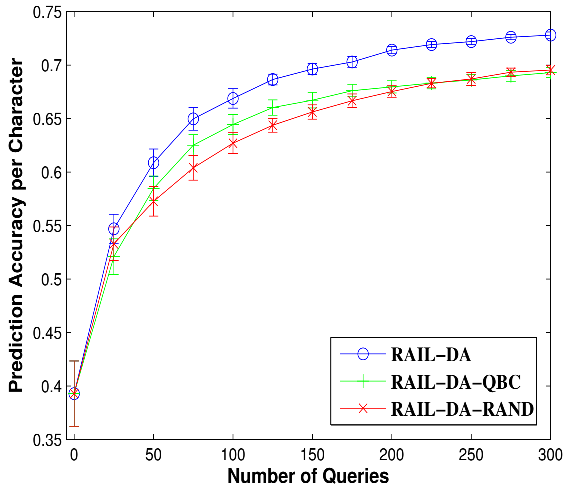
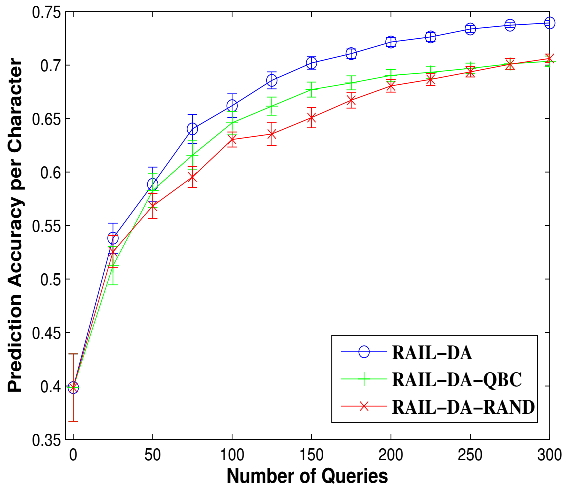


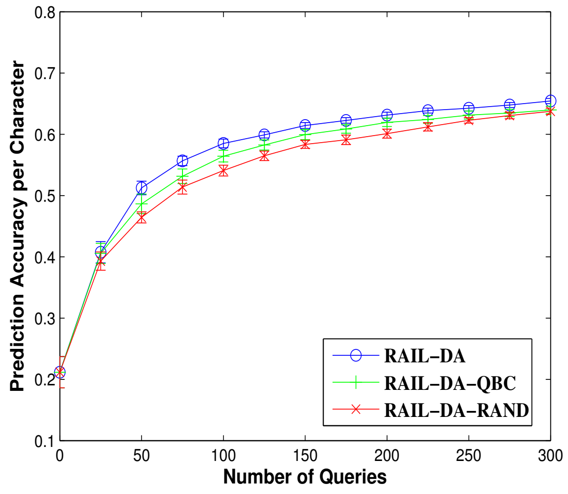
Figure 3: Performance of RAIL-DA with different base active learners on (a) Cart-pole (b) Bicycle balancing (c) Wargus (d) Driving Domain.



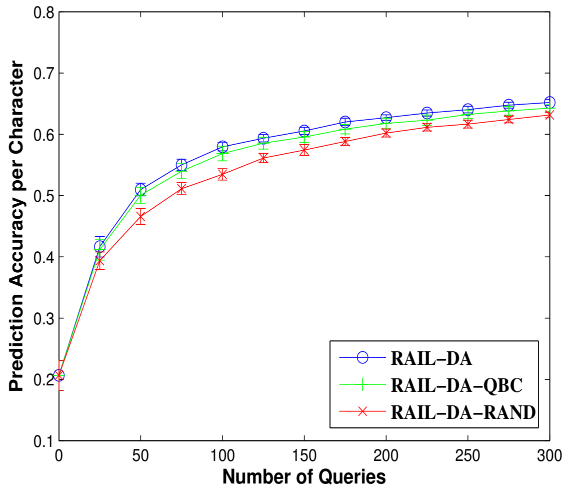
(a)



(b)



(c)



(d)

Figure 4: Performance of RAIL-DA with different base active learners on NETtalk: (a) Stress prediction, $L = 1$ (b) Stress prediction, $L = 2$ (c) Phoneme prediction, $L = 1$ (d) Phoneme prediction, $L = 2$. Character accuracy of the expert is 1.

each iteration. Recall that in these domains, especially during early iterations, RAIL-DA is quite likely to generate several uninformative states that are far from the expert's state distribution (states with fallen pole or bike, or states where the friendly team is heavily outnumbered by the enemy team in Wargus). Taking state density information into account helps to avoid querying in such states compared to a learner that ignores density information. In the driving domain, we see that although RAIL-DA performs better than RAIL-DA-RAND, its performance is comparable to that of RAIL-DA-QBC. This is because in the driving domain, even states that are not close to the expert's state distribution provide informative training information. This allows RAIL-DA-QBC to pose useful queries and generalize well from them.

Figure 4 shows the performance of the three versions of RAIL-DA on the NETtalk data set along with 95% confidence intervals. RAIL-DA and RAIL-DA-QBC can select the best query across the entire training set. RAIL-DA-RAND selects a random query from the set of unlabeled states generated on a random training sequence. For stress prediction, we see that RAIL-DA performs better than both RAIL-DA-QBC and RAIL-DA-RAND. For phoneme prediction, RAIL-DA performs better than RAIL-DA-RAND, but its performance is comparable to RAIL-DA-QBC. Overall, we see that RAIL-DA performs best, so it will be compared against the other baselines in the next section.

6.4 Experiment 3: Comparison of RAIL-DA with Baselines

We compare RAIL-DA against the following baselines:

1. *Passive*. This baseline simulates the traditional approach by starting at the initial state and querying the expert about what to do at each visited state.

2. *unif-QBC*. This baseline views all the MDP states as i.i.d. according to the uniform distribution and applies the standard query-by-committee (QBC) (Seung et al., 1992) active learning approach. Intuitively, this approach will select the state with highest action uncertainty according to the current data set and ignores the state distribution.

3. *unif-RAND*. This baseline selects states to query uniformly at random.

4. *Confidence based autonomy (CBA)* (Chernova and Veloso, 2009). This approach requires the use of policies that provide some form of confidence estimate (e.g., probabilities over actions). Given the current set of labeled data, the approach executes trajectories of the current policy until it reaches a state where the policy confidence falls below a threshold. It then queries the expert for the correct action and updates the policy accordingly. It then executes the correct action and continues until the next low confidence state is reach. It is possible for CBA to stop asking queries once the confidence exceeds the threshold in all states visited by the current policy. We use the same automated threshold adjustment strategy proposed by Chernova and Veloso (Chernova and Veloso, 2009). We also experimented with other threshold adjustment mechanisms as well as fixed thresholds, but were unable to find an improvement that performed better across our domains.

Figure 5 shows the results of this experiment along with 95% confidence intervals on our first four test domains. Figure 5(a) shows the performance of RAIL-DA on cart-pole.

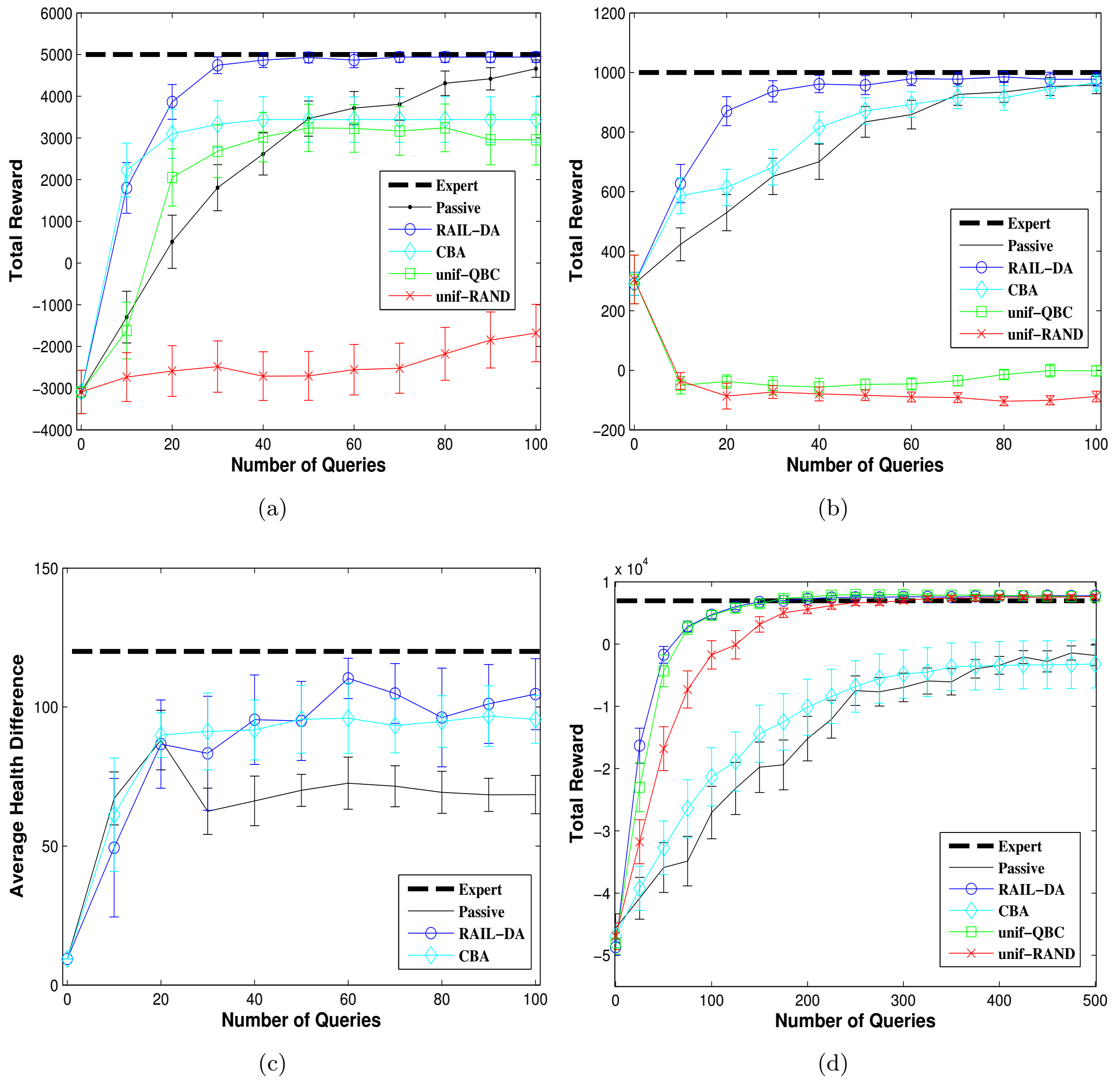
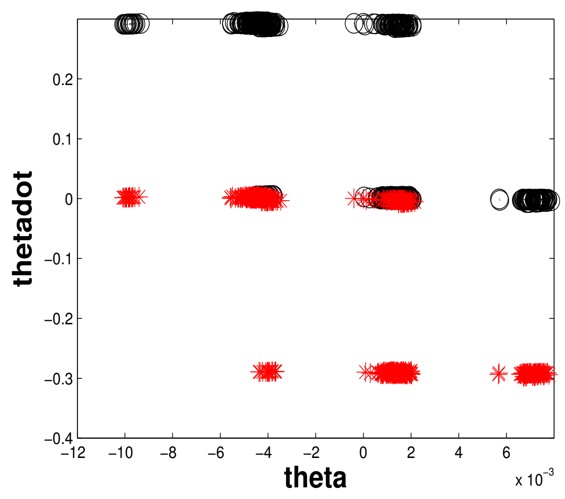
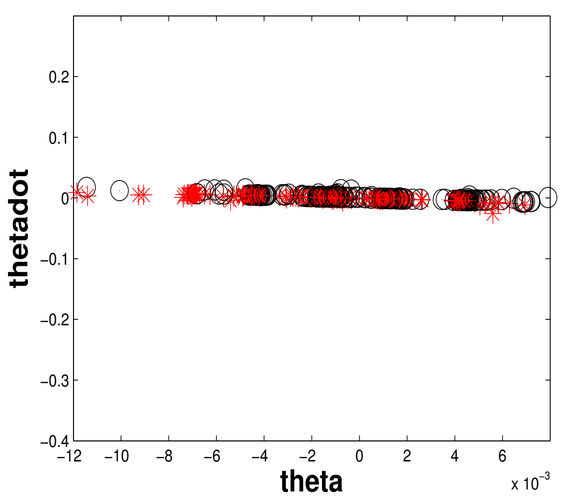


Figure 5: Active imitation learning results: (a) Cart-pole (b) Bicycle balancing (c) Wargus (d) Driving domain.



(a)



(b)

Figure 6: Queried states in Cart-pole for learners: (a) Passive, (b) RAIL-DA. The black circles represent states where the right action is suggested by the expert policy. The red stars represent states where the left action is suggested. The decision boundary separating these two sets of states is a line very close to $\dot{\theta} = 0$. We see that unlike Passive, RAIL-DA focuses its queries around the decision boundary. This figure is best viewed in color.

We observe that RAIL-DA learns quickly and achieves optimal performance with only 30-35 queries. Passive, on the other hand, takes 100 queries to get close to the optimal performance. The reason for this difference is clear when one visualizes the states queried by RAIL-DA versus Passive. Figure 6(a) shows the states queried by Passive. The black circles represent states where the right action is suggested by the expert. The red stars represent states where the left action is optimal according to the expert. The decision boundary of the expert policy is the line separating these two sets of states (very close to $\dot{\theta} = 0$). We notice that Passive asks many uninformative queries that are not close to the decision boundary. Figure 6(b) shows the queries posed by RAIL-DA. We see that the queries posed by RAIL-DA tend to be close to the decision boundary of the expert policy.

A naive reduction to active learning can be dangerous, as demonstrated by the poor performance of unif-QBC. Further, RAIL-DA performs much better than random query selection as demonstrated by the performance of unif-RAND. By ignoring the real data distribution altogether and incorrectly assuming it to be uniform, these naive methods end up asking many queries that are not relevant to learning the expert policy (e.g., states where the pole is in an irrecoverable fall or the cart is out of bounds). CBA, like RAIL-DA, learns quickly but settles at a suboptimal level of performance. This is because it becomes confident prematurely and stops asking queries. This shows that CBA's automatic threshold adjustment mechanism did not work well in this domain. We did experiment with several modifications to the threshold adjustment strategy, but we were unable to find one that was robust across all our domains. Thus we report results for the original strategy.

Figure 5(b) compares each approach in the bicycle domain. The results are similar to those of cart-pole with RAIL-DA being the top performer. Unif-RAND and Unif-QBC show notably poor performance in this domain. This is because bicycle balancing is a harder learning problem than cart-pole with many more uninformative states (an unrecoverable fall or fallen state). We found that almost all queries posed by Unif-RAND and Unif-QBC were in these states. CBA does only slightly better than Passive, though unlike cart-pole, it achieves near-optimal asymptotic performance.

The results in the Wargus domain are shown in Figure 5(c). Passive learns along the expert's trajectory in each map on all 5 training maps considered sequentially according to a random ordering. For RAIL-DA, in each iteration a training map is selected randomly and a query is posed in the chosen map. For CBA, a map is selected randomly and CBA is allowed to play an episode in it, pausing and querying as and when needed. If the episode ends, another map is chosen randomly and CBA continues to learn in it. After each query, the learned policy is tested on the 16 test maps. We use the difference in the total health of friendly and enemy units at the end of the battle as the performance metric (which is positive for a win). We did not run experiments for unif-QBC and unif-RAND, because it is difficult to define the space of feasible states over which to sample uniformly.

We see that although Passive learns quickly for the first 20 queries, it fails to improve further. This shows that the states located in this initial prefix of the expert's trajectory are very useful, but thereafter Passive gets stuck on the uninformative part of the trajectory until its query budget is over. On the other hand, RAIL-DA and CBA continue to improve beyond Passive, with the performance of CBA being comparable to RAIL-DA in this domain, which indicates that both these active learners are able to locate and query more informative states.

The results for the driving domain are shown in Figure 5(d). We see qualitatively similar performance trends as in the previous three domains with RAIL-DA still being the best performing learner and outperforming most of the competing baselines. The exception however is that unif-QBC and unif-RAND perform quite well in this domain, both being able to outperform Passive and even CBA. Furthermore, performance of unif-QBC is quite comparable to that of RAIL-DA. The good performance of unif-QBC and unif-RAND in the driving domain is due to the fact that obtaining action labels on most states in the driving domain can serve as useful training data for learning the expert policy. This is unlike the previous three domains, where labels obtained for many states were relatively useless for learning the target policy (e.g., states with a fallen pole or bike).

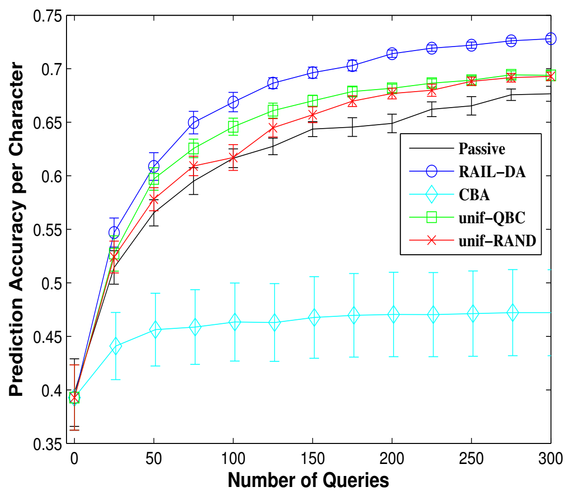
Our final set of results are in the structured prediction domain. Passive learns along the expert's trajectory on each training sequence considered in the order it appears in the training set. Therefore, Passive always learns on the correct context, i.e., previous L characters correctly labeled. RAIL-DA and unif-QBC can select the best query across the entire training set. CBA, like Passive, considers training sequences in the order they appear in the training set and learns on each sequence by pausing and querying as and when needed. To minimize the effects of the ordering of the training sequences on the performance of Passive and CBA, for all learners, we ran 50 different trials where in each trial we randomize the order of the training sequences. We report final performance as the learning curves averaged across all 50 trials. The learning curves along with 95% confidence intervals are shown in figure 7.

Figures 7(a) and (b) presents the stress prediction results. The results are qualitatively similar to cart-pole, except that Unif-QBC and unif-RAND do quite well in this domain but not as well as RAIL-DA. We see that CBA performs quite poorly because we found that in several trials it prematurely stops asking queries,[5] which reveals its sensitivity to its threshold adjustment mechanism. Similar trends are seen in the phoneme prediction results shown in Figures 7(c) and (d). CBA does well on this task but not as well as RAIL-DA.

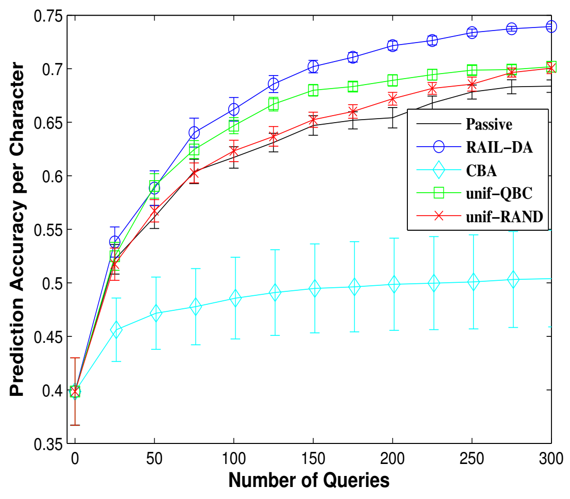
6.5 Overall Observations

We can draw a number of conclusions from the experiments. First, the implementation choices discussed in Section 5 are necessary to make RAIL more practical. In particular, RAIL-DA, which aggregates data and asks only one query per iteration, proved to be the most robust among all the variants of RAIL. Second, the choice of the active i.i.d. learner used for RAIL is important. In particular, performance can be poor when the active learning algorithm does not take density information into account. Using density weighted query-by-committee was effective in all of our domains. Third, RAIL-DA proved to be the most robust and effective active imitation learning algorithm among several baselines in all of our domains. It outperformed all other baselines in our experiments. Fourth, we found that CBA is quite sensitive to the threshold adjustment mechanism, and we were unable to find an alternative mechanism that works across our domains. In the original CBA paper by Chernova and Veloso (Chernova and Veloso, 2009), CBA was tested only on

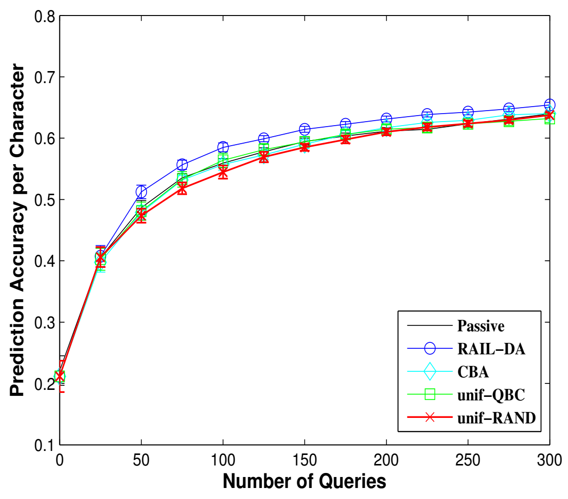
5. To plot the learning curve for CBA, for each trial, we took the character accuracy after the last query and extrapolated it to 300 queries to obtain a curve. The final curve is the average of the extrapolated curves.



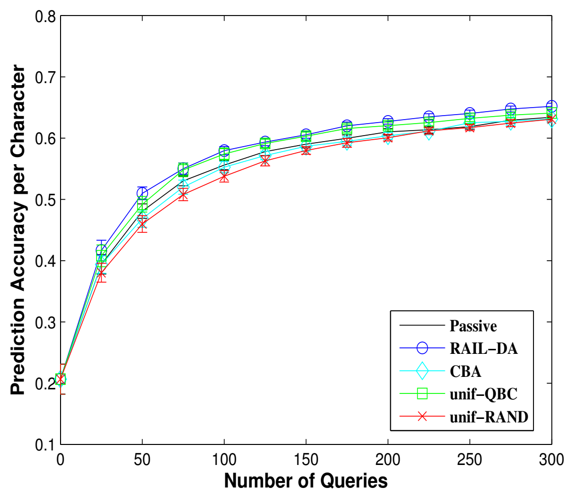
(a)



(b)



(c)



(d)

Figure 7: Active imitation learning results on NETtalk: (a) Stress prediction, $L = 1$ (b) Stress prediction, $L = 2$ (c) Phoneme prediction, $L = 1$ (d) Phoneme prediction, $L = 2$. Character accuracy of the expert is 1.

	Passive Imitation Learning	Active Imitation Learning
Non-stationary	$T \cdot N_p(\frac{\epsilon}{T^2}, \frac{\delta}{T})$	$\sum_{t=1}^{T} N_a(\frac{\epsilon}{T^2}, \frac{\delta}{T}, d^t_{\hat{\pi}^{t-1}})$
Stationary	$T \cdot N_p(\frac{\epsilon}{T^2}, \delta)$	$\sum_{t=1}^{T} N_a(\frac{\epsilon}{T^3}, \frac{\delta}{T}, d_{\hat{\pi}^{t-1}})$

Table 1: A comparison of the best known label complexities for PAC learning deterministic stationary and non-stationary policies in the passive and active settings. The label complexities are for learning policies with regret no more than ϵ with probability at least $1-\delta$. $N_p(\epsilon', \delta')$ and $N_a(\epsilon', \delta', D)$ are label complexities of passive and active i.i.d. learning respectively with accuracy and reliability parameters ϵ' and δ' and data distribution D. When N_a is exponentially smaller than N_p on some or all of the distributions $d_{\hat{\pi}^{t-1}}, t = 1 \ldots T$, then these savings in label complexity in the i.i.d. setting translate to imitation learning resulting in improved label complexity of active imitation learning compared to passive.

the driving domain, and in their experiment CBA was initialized with a large amount of training data. We found that initializing CBA with a larger training set also resulted in improved performance of CBA in all of our domains. However, this conflicts with our goal of minimizing the amount of training data required from the expert, and hence we initialized all learners with the minimum training data required by the SimpleLogistic classifier. This turned out to be insufficient for CBA to function properly. Fifth, we showed that a more naive application of active i.i.d. learning in the imitation setting is not always effective.

7. Summary and Future Work

We considered reductions from active imitation learning to active i.i.d. learning, which allow for advances in the i.i.d. setting to translate to imitation learning. First, we analyzed the label complexity of reductions for both non-stationary and stationary policies, showing the number of queries required for active imitation learning in terms of the active sample complexity in the i.i.d. setting. These results for the realizable learning setting are summarized in Table 1. In the non-stationary case, the results show that active IL will not be worse than passive IL provided that the i.i.d. active learning algorithm is guaranteed to be no worse than passive. Further we can expect significant improvement in query complexity when the active i.i.d. algorithm is significantly better than the passive i.i.d. learner.

For the case of stationary policies, our current reduction RAIL only guarantees improvement or equivalence to passive IL when there is significant reduction in sample complexity of i.i.d. active learning over passive learning. While this is often the case in practice, it leaves an open theoretical problem. If we use an active i.i.d. learner that is guaranteed to do no worse than passive, then can we find a reduction such that active IL also has that guarantee?

Our second contribution was to introduced RAIL-DA, a practical variant of the reduction for stationary policies. RAIL-DA employees data aggregation and incremental learning in order to address several practical inefficiencies noted for the RAIL algorithm studied in the analysis. Our experiments showed that RAIL-DA significantly improved over RAIL and other variants of RAIL$^+$. Further, we showed that in five domains RAIL-DA significantly outperformed a number of natural alternatives and the CBA algorithm from prior work.

The work presented in this paper is a first theoretical effort towards analyzing an active imitation learning approach and showing that it enjoys better label complexity than the traditional passive approach. In addition to the above open problem, an interesting line of followup work is to analyze RAIL-DA or other variants of RAIL that use data aggregation. Further, it is of interest to consider the online active learning setting, where the learner is embedded in a real environment, rather than having access to a simulator that can be reset. Such an algorithm might resemble the CBA algorithm, which would continually execute the current policy and only query the expert when it was uncertain. This is similar to the traditional QBC active learning algorithm in the i.i.d. setting. The only difference is that in the imitation learning setting the unlabeled data stream does not come from a fixed i.i.d. distribution, but rather from the policy being executed. It seems plausible that the theoretical results for QBC could be extended to the imitation learning setting.

We are also interested in extending the allowed query responses. For example, it is natural to allow the expert to declare a query as "bad" and refuse to provide an action label. This is useful in situations where the queries are posed at states that the expert would rarely or never encounter, and hence may have not natural preference about what to do. In our bicycle balancing example, these correspond to states where the bicycle is in an unavoidable fall, and no action can prevent the crash. In such cases, the expert is likely to be uncertain or agnostic about the action, since the choice does not arise for the expert or is unimportant. We would like to study how to incorporate the "bad query" response into the query selection process and update policy parameters based on such responses as those responses effectively indicate states to be avoided. A first step in this direction has already been taken, where we used the "bad query" responses within a Bayesian active learning framework to select queries (Judah et al., 2011). However, the responses were not incorporated into the learning of policy parameters (or updating the posterior) in that work.

Finally, we would like to apply active imitation learning to other types of imitation learning applications, e.g., the development of policy learning agents that learn by imitation of computationally expensive automated experts. For example, given a domain model or a simulator, automated experts based on various types of search can make near optimal decisions, if provided enough time. However, generating full trajectories of near-optimal behavior can be extremely time consuming, and require many trajectories if the learned policy representation is complex. Active imitation learning could be a viable approach to speed up the learning of more reactive policies, based on data from such computationally expensive experts.

It is important to consider usability issues that arise when interacting with human experts. In particular, it is likely that the cost model of queries studied in this paper (cost of 1 per query) is overly simplistic. For example, it may be less work for an expert to answer sets of queries about related states/scenarios, rather than arbitrary sets of queries. Understanding how to acquire and use such cost models for active learning is an interesting future direction. Finally, real experts will often not correspond to deterministic policies. Rather, they may appear to be non-deterministic or stochastic policies. Studying both passive and active imitation learning in such a setting is an interesting an important direction.

Acknowledgments

The authors acknowledge support of the ONR ATL program N00014-11-1-0106. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ONR. We would also like to thank Robert Cohn and Aaron Wilson for making driving and bicycle simulators available.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 1–8, 2004.

J. Azimi, A. Fern, X. Fern, G. Borradaile, and B. Heeringa. Batch active learning via coordinated matching. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, pages 1199–1206, 2012.

A. Beygelzimer, S. Dasgupta, and J. Langford. Importance weighted active learning. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning*, pages 49–56, 2009a.

A. Beygelzimer, J. Langford, and P. Ravikumar. Error-correcting tournaments. In *Proceedings of the Twentieth International Conference on Algorithmic Learning Theory*, pages 247–262, 2009b.

C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 1999.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

K. Brinker. Incorporating diversity in active learning with support vector machines. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 59–66, 2003.

S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009.

J. A. Clouse. An introspection approach to querying a trainer. Technical report, University of Massachusetts, Amherst, MA, USA, 1996.

D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

R. Cohn, M. Maxim, E. Durfee, and S. Singh. Selecting operator queries using expected myopic gain. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 40–47, 2010.

R. Cohn, E. Durfee, and S. Singh. Comparing action-query strategies in semi-autonomous agents. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1102–1107, 2011.

I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157, 1995.

S. Dasgupta. Coarse sample complexity bounds for active learning. In *Advances in Neural Information Processing Systems 18*, pages 235–242. MIT Press, 2006.

S. Dasgupta. Two faces of active learning. *Theoretical Computer Science*, 412(19):1767–1781, 2011.

H. Daumé, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

T. Dietterich, G. Hao, and A. Ashenfelter. Gradient tree boosting for training conditional random fields. *Journal of Machine Learning Research*, 9:2113–2139, 2008.

F. Doshi, J. Pineau, and N. Roy. Reinforcement learning with limited reinforcement: using bayes risk for active learning in POMDPs. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 256–263, 2008.

Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

A. Gil, H. Stern, and Y. Edan. A Cognitive Robot Collaborative Reinforcement Learning Algorithm. *International Journal of Information and Mathematical Sciences*, 5:273–280, 2009.

D. H. Grollman and O. C. Jenkins. Dogged learning for robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2483–2488, 2007.

Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems 20*, pages 593–600. Curran Associates, Inc., 2008.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

S. Hanneke. *Theoretical Foundations of Active Learning.* PhD thesis, CMU Machine Learning Department, 2009.

S. C. H. Hoi, R. Jin, and M. R. Lyu. Large-scale text categorization by batch mode active learning. In *Proceedings of the Fifteenth International Conference on World Wide Web*, pages 633–642, 2006a.

S. C. H. Hoi, R. Jin, J. Zhu, and M. R. Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 417–424, 2006b.

K. Judah, S. Roy, A. Fern, and T. Dietterich. Reinforcement learning via practice and critique advice. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 481–486, 2010.

K. Judah, A. Fern, and T. Dietterich. Active imitation learning via state queries. In *Proceedings of the ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

K. Judah, A. Fern, and T. Dietterich. Active imitation learning via reduction to i.i.d. active learning. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artifial Intelligence*, pages 428–437, 2012.

R. Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999.

L. Kocsis and C. Szepesvri. Bandit based monte-carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning*, pages 282–293, 2006.

M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 31–46, 2009.

A. McCallum and K. Nigam. Employing em and pool-based active learning for text classification. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 359–367, 1998.

L. Mihalkova and R. Mooney. Using active relocation to aid reinforcement learning. In *Prodeedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, pages 580–585, 2006.

A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, 1998.

S. Ross and J. A. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

S. Ross, G. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artifical Intelligence and Statistics*, pages 627–635, 2011.

B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 287–294, 1992.

A. P. Shon, D. Verma, and R. P. N. Rao. Active imitation learning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 756–762, 2007.

D. Silver, J. A. Bagnell, and A. Stentz. Active learning from demonstration for robust autonomous navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 200–207, 2012.

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

Umar Syed and Robert E. Schapire. A reduction from apprenticeship learning to classification. In *Advances in Neural Information Processing Systems 23*, pages 2253–2261. Curran Associates, Inc., 2010.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Z. Xu, R. Akella, and Y. Zhang. Incorporating diversity and density in active learning for relevance feedback. In *Proceedings of the Twenty-Ninth European Conference on IR Research*, pages 246–257, 2007.

B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceeings of the IEEE International Conference on Data Mining*, pages 435–442, 2003.