accuracy of this unlabeled distribution (with respect to 
\( d_\pi \)), instead of using a point estimate for \( \hat{\pi} \), we treat
\( \hat{\pi} \) as a Bayesian classifier for purposes of defining \( d_\pi \).
In particular, at iteration \( t \) let \( D_t \) be the set of state-
action pairs collected from previous iterations. We use
this to define a posterior \( P(\pi|D) \) over policies in our
policy class \( H \). This, in turn, defines a posterior unla-
beled state distribution \( d_{D_t} = \mathbb{E}_{d_{\pi} \sim P(\pi|D)} [d_{\pi}(s)] \) which
is used in place of \( d_\pi \) in RAIL. Note that we can sam-
ple from this distribution by first sampling a policy \( \hat{\pi} \n\) and then sampling a state from \( d_\pi \), all of which can be
done without interaction with the expert. We observe
that in practice \( d_{D_t} \) is a significantly more useful esti-
mate of \( d_\pi \) than the point estimate, since its more
highly weighted states tend to carry significant weight
according to \( d_\pi \).

To summarize, iteration \( t \) of RAIL-DW carries out the
following steps, adding one state-action pair to the
data set: (\( D_1 \) is the initial, possibly empty, data set)

1. Run active learner \( L_a \) using unlabeled data dis-
bution \( d_{D_t} \) to select a single expert state \( s \).

2. Query the expert about \( s \) to obtain action \( \pi^*(s) \)
and let \( D_{t+1} = D_t \cup \{(s, \pi^*(s))\} \).

The iteration repeats until the query budget is ex-
ceeded. It remains to specify the specific i.i.d. active
learning algorithm that we use and how we sample
from \( d_{D_t} \).

Since it is important that the i.i.d. active learner be
sensitive to the unlabeled data distribution, we employ
a density-weighted learning algorithm (hence the name
RAIL-DW). In particular, we use density-weighted
query-by-committee (McCallum and Nigam, 1998) in
our implementation. Given a sample of unlabeled data
points, this approach uses bagging (Breiman, 1996) to
generate a committee for query selection and also uses
a density estimator to estimate the density of the unla-
beled data points. The selected query is the state that
maximizes the product of state density and committee
disagreement. We use the entropy of the vote distribu-
tion (Dagan and Engelson, 1995) as a typical measure
of committee disagreement. We use a committee of
size 5 in our experiments and estimate density of unla-
beled points via simple distance based binning.

Finally, for sampling we assume a class of linear para-
metric policies and assume a uniform prior over the
parameters. We approximate sampling from \( d_{D_t} \) via
bagging: where we generate \( K \) bootstrap samples of
\( D_t \) and a policy is learned from each using a supervised
learner. This produces a set of \( K \) policies and each is
executed to generate \( K \) trajectories. The states on
those trajectories are given to the active learner as the
unlabeled data set. We set \( K = 5 \) in our experiments.

6 Experiments

We empirically evaluate RAIL-DW on four domains: 1) Cart-pole, 2) Bicycle, 3) Wargus and 4) The struc-
tured prediction domain NETtalk. We compare RAIL-
DW against the following baselines: 1) Passive, which
simulates the traditional approach by starting at the
initial state and querying the expert about what to do
at each visited state, 2) \( \text{unif-QBC} \), which views all the
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. Intu-
itively, this approach will select the state with high-
est action uncertainty according to the current data
set and ignore the state distribution, 3) \( \text{unif-RAND} \),
which selects states to query uniformly at random, and
4) Confidence based autonomy (CBA) (Chernova and
Veloso, 2009), which, starting at the initial state, exe-
cutes the current policy until the learner’s confidence
falls below an automatically determined threshold at
which point it queries the expert for an action. CBA
may decide to stop asking queries once the confidence
exceeds the threshold in all states. We use the exact
automated threshold adjustment strategy proposed in
(Chernova and Veloso, 2009). For all of these meth-
ods, we employed the SimpleLogistic classifier in Weka
(Hall et al., 2009) to learn policies.

Cart-Pole. Cart-pole is an RL benchmark where a
cart must balance an attached vertical pole by ap-
plying left or right forces 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 describing the position and veloc-
ity of the cart and the angle and angular velocity of the
pole. We made slight modifications to the usual set-
ing where we allow the pole to fall down and become
horizontal and the cart to go out of bounds (we used
default \([-2.4, 2.4]\) as the bounds for the cart). We let
each episode run for a fixed length of 5000 time steps.
This opens up the possibility of generating several “un-
desirable” states where either the pole has fallen or the
cart is out of bounds that are rarely or never gener-
ated by the expert’s state distribution. The expert
policy was a hand-coded policy that can balance the
pole indefinitely. For each learner, we ran experiments
from 30 random initial states close to the equilibrium
start state, generating learning curves for each one and
averaging the results. We report total reward with a
reward function (unknown to the learner) that is +1
for each time step where the pole is balanced and the
cart is in bounds and -1 otherwise.

Figure 1(a) shows the results for cart-pole. We observe
that RAIL learns quickly and achieves optimal perfor-
ance with only 30-35 queries. Passive, on the other
hand, takes 100 queries to get close to the optimal per-