COMS 6998-4 Fall 2017 Presenter: Wei Zhang
October 23, 2017 Scribe: Kiran Vodrahalli

Imitation Learning

1 Introduction

A canonical example in imitation learning is the self-driving car. This task is a sequential decision-making problem: The car must decide how to move, observe the way the world around it has changed, and then decide how to move again. The sequence iterates. The difficult aspect of this setting is that the current actions of the agent (in our example, the car) will affect the examples the agent receives in the future.

Let us rigorously define the setting.

Definition 1. *Imitation learning setting.*
*In **imitation learning**, we have access to an expert that already knows how to perform the task at hand (e.g. driving). At each of a sequence of timepoints $t = 1, \cdots, T$, the agent receives environment information x_t. It then must decide to take an action a_t from some set of actions. Finally, the agent suffers some cost C_t based on the action a_t and and the environment x_t. The world then changes, and we move to timestep $t+1$ where the process repeats. Note that we may not actually observe the true cost C_t at each time step! Instead, we will introduce a surrogate loss ℓ_t which is observed, depending on our setting, which upper bounds the true cost C_t. Typically ℓ_t is defined with respect to some expert that we are trying to learn to imitate. If our goal is to just predict the expert's actions, then in fact C_t is observed and we could just use it directly as our surrogate loss (though we may want to still replace the $0-1$ loss, for instance, with a convex-surrogate like the square loss or hinge loss).*

*The goal of imitation learning is to learn a **policy function** $\pi_t : X^t \times A^{t-1} \to A$, where the policy function at time t maps the previous set of tuples of environments seen and actions taken to the next action. In particular, we want to learn $\{\pi_t\}_{t=1}^{T-1}$ such that we minimize*

$$\mathop{\mathbb{E}}_{\text{randomness in the world}} \left[\sum_{t=1}^{T} C_t \left(x_t, \pi_t(x_1, \cdots, x_t, a_1, \cdots, a_{t-1}) \right) \right]$$

*Here, the randomness in the world can affect the transition in state from $x_t \to x_{t+1}$ as a result of action a_t, can affect the way the loss function at time t is defined, and can also affect the application of action a_t itself. In specific settings, we make clear what assumptions we want to make about the randomness. For convenience, we call this assumption **r.i.w.**.*

The idea is to learn this sequence of policies based on seeing how the expert reacts in different situations, and mimicking the expert.

2 Methods for Implementing Imitation Learning

There are several approaches one could imagine that would allow us to learn a policy in the imitation learning setting. First, we consider a reduction to the supervised learning setting.

2.1 Imitation Learning via Supervised Learning

One can view the imitation learning problem in a supervised learning framework. We will use the provided expert as follows: Ask the expert to perform some sequential task with T decisions in the world N times. For instance, ask an expert to drive a car down a road that requires T choices to be made N different times. We record what happens at each step: We get to see the different state x_t and the action selected a_t at each time point for each of the N sequences, as well as the loss experienced by the expert (Here, $C_t = \ell_t$). Then, we can think of this data as NT data points of the form (state, action). Assuming that there are a finite number of possible actions, we can frame this problem as supervised multiclass learning with NT examples. One could then train, for instance, some logistic regression model with this data to get a policy function mapping from states to actions directly.

We can bound the "performance" of this approach in terms of the success of the expert. From now on, we will understand that the error refers to the following definition:

Definition 2. *Error term.*
The error of a particular policy π is given with respect to the performance of the expert policy $\tilde{\pi}$:

$$\mathop{\mathbb{E}}_{r.i.w.}\left[\sum_{t=1}^{T} C_t(x_t, \pi_t)\right] - \mathop{\mathbb{E}}_{r.i.w.}\left[\sum_{t=1}^{T} C_t(\tilde{x}_t, \tilde{\pi}_t)\right]$$

Here, we abuse notation for C_t: $C_t(x_t, \pi_t)$ means the loss due to the state x_t and the action taken at x_t according to π_t, which may depend on previous states and actions. We will use similar notation when discussing $\ell_t(x_t, \pi_t)$.

The following theorem holds:

Theorem 1. *Error bound for Supervised Imitation Learning.*
Let ϵ be the expected error rate over r.i.w. of the supervised classifier used in the algorithm. Assume the surrogate loss functions ℓ_t are bounded by L. Then,

$$\mathop{\mathbb{E}}_{r.i.w.}\left[\sum_{t=1}^{T} C_t(x_t, \pi_t)\right] \leq \mathop{\mathbb{E}}_{r.i.w.}\left[\sum_{t=1}^{T} C_t(\tilde{x}_t, \tilde{\pi}_t)\right] + LT^2\epsilon$$

where the $\tilde{\cdot}$ denotes the states and actions of the expert.

It turns out this bound is tight for the approach. Consider the following (contrived) example of imitation learning.

Example 1. *Worst-case Supervised Imitation Learning.*
At each round, the learner sees a "0" button and a "1" button. Exactly one of the buttons is correct each round. If correct, the learner receives no loss, and a loss of 1 *otherwise. To decide which to press, the learner is shown a picture of either* 0 *or* 1*.*

1. *Time* $t = 1$*: At* $t = 1$*, the correct answer to press the value illustrated on the picture.*
2. *Time* $t = 2, \cdots$*: The correct answer is XOR(*x_t*,* a_{t-1}*) where we note both* $x_t, a_{t-1} \in \{0, 1\}$*.*

For this example, note that it is possible for the expert to get zero loss quite easily. Second, note that if the learner makes a mistake even once, choosing the (correct) rule of applying XOR will end up being wrong! The learner will have to figure out how to make another mistake in order to get back on track, but will have no precedent for doing so, since the expert made zero mistakes. It is possible to show a lower bound on the additional error of order $T^2\epsilon$*.*

2.2 Initial Approaches for Dealing with Off-Policy Scenarios

The main issue in the previous setting was that we had no way to handle situations where the learner's policy took us to states which we had never seen before. In this sense, the learner became **off-policy** from the training data (expert's) policy. How might we deal with this problem?

2.2.1 Forward Training

The first approach is to consider training our policy π_t on the states that it will actually encounter while executing policies $\pi_1, \cdots, \pi_{t-1}$. If we train it on the policies that we know the learner will see, then the learner will avoid getting off-policy! We still want to use the expert of course: We just look at what the expert policy does on the states generated by the learned policy on the previous $t - 1$ timesteps. For now, we are ignoring whether the learned policy actually covers the best set of choices, we just want to make sure that we stay on policy. This approach is known as **forward training**, since we use our learned policy to figure out what states to train over later on. The main drawback of this approach is that for very large T, or T undefined, this method is not implementable: We have to train T different policies in sequence, and will not have a complete policy until we train all T of them.

If we claim that our policy π has ϵ loss over the states that it itself will induce, then the following theorem holds:

Theorem 2. *Forward training error bound.*
Let $Q_t^{\pi'}(x, \pi)$ *denote the loss if we use the action suggested by* π *on state* x*, and afterwards follow the actions suggested by* π' *for* t *steps. Then, if*

$$Q^{\pi^*}_{T-t+1}(x, a) - Q^{\pi^*}_{T-t+1}(x, \pi^*) \leq u$$

for all actions $a \in A$ and all time points $t \in [T]$, then the error is bounded by $uT\epsilon$.

Specifically, ϵ is defined as the expected loss over the states selected by policy π, where the randomness is due to the world:

$$\epsilon = \mathop{\mathbb{E}}_{\substack{r.i.w. \\ \text{over states } x \text{ reached by policy } \pi}} \Big[\ell(x, \pi)\Big]$$

Note that the absence of L, the upper bound on the loss functions ℓ_t, is due to the fact that ϵ is defined in terms of the loss functions.

We can interpret this theorem as giving us a potentially better interpolation between the supervised learning scenario we saw previously and something better, that no longer has regret on order T^2. The bound on the difference between the Q-functions can be interpreted as follows. We have arrived at state x after t timesteps. We want to know what will happen if, for just this current time step, we use some action a, and then use the expert policy to guide our steps. Compare the resulting losses to the situation where we just use the expert policy forever after and see how different the two cases are. The difference in cost between these two settings is bounded by u. The value of u is at most T (since could lose at every following single step, setting $t = 1$). Thus in the worst case, this bound is as bad as the supervised learning setting. We get a much better bound if u is constant. Generally, if the expert policy is able to get back on track after the learner policy makes some mistakes within a constant number of turns, u is constant since we will only pay for that constant number of turns where the expert was turning things around.

The quantity that u bounds is of interest mainly because we can write the loss of the learner's policy as a telescoping sum of the expected value of the difference between the $Q(x, \pi)$ and $Q(x, \pi^*)$.

2.2.2 SMILe

A tractable alternative to the forward algorithm is **stochastic mixed iterative learning**. The essential idea here is to, instead of training T different policies sequentially, learning a mixture of n policies plus querying the expert at iteration n. Thus, we use only a single mixture of policies, rather than T different policies as before. For the first iteration, the initial policy always chooses the expert. It is then updated by mixing the policy trained in the current iteration with the previous policy, using some parameter α. The algorithm can stop at any point by normalizing over the learned policies, so that the expert is no longer necessary. The algorithm maintains the property that it is trained on states which are likely to be reached by the learner's policy since it is trained to mimic the expert on the states that the learner is likely to reach.

The regret bound for SMILe is the same as for Forward Training, the difficulty is now due to the instability of the mixture of policies, since the policy is now itself stochastic.

2.3 DAGGER

Both of the regret bounds in the previous section had bounds that were at best linear in T, the number of time steps. As T increases, the average regret would end up being a constant: We would always keep making mistakes. Is there an algorithm for which average regret will go to 0 as $T \to \infty$? Another issue of the previous methods is the potential stochasticity of the resulting output policy. We would like to be able to return a deterministic policy if possible. Finally, we would prefer to not train multiple policies, as we did in the forward algorithm setting.

The answer to all of these issues is the DAGGER algorithm, which stands for "Dataset Aggregation". The principle is simple: We will iteratively create a dataset of ever-increasing size which we will apply supervised learning to in order to learn the next policy. We construct the dataset as follows: Initially, we ask the expert to solve the task (for instance, driving a car), and record the actions and state-sequence they encountered. Then, we learn a policy based off of this sequence. Now, we take that policy and set it to work — but, at the same time, we will also take the expert along for the ride, and record the actions the expert *would have made* at each decision point! A modification of this algorithm is, at every iteration, to allow the actual guiding policy to be a mixture between the previous learned policy and the expert, for which the weight placed on the expert decays rapidly over time. In this manner, we will collect another dataset of expert actions, yet on a new set of states. We then combine the datasets and learn a new supervised policy on this set of data. Iterating this process, we return a single policy at the end, after we have learned on all the generated data. We can also select a policy by evaluating all policies produced by DAGGER on some heldout dataset, and then choosing the best one. In practice, this method is safer.

To make the algorithm concrete, we provide pseudocode. Let τ denote a sequence of length T of (x_t, a_t, ℓ_t) tuples (state, action, loss). We also refer to these as trajectories. Let $\mathcal{A}$ be a no-regret online algorithm, m be the number of trajectories of length T per round, and N be the total number of rounds. We also have oracle access to an expert.

Algorithm 1 DAGGER

procedure DAGGER($\mathcal{A}, m, N,$ expert)
  Run the expert m times to produce $\{\tau_i^0\}_{i=1}^m$.
  Initialize the dataset $D_0 = \{(x, a) : \forall i \in [m], \forall (x, a, \ell) \in \tau_i^0\}$.
  Initialize policy $\pi_0 = \mathcal{A}(D_0)$.
  for $j = 1, \cdots, N$ **do**
    Run policy π_{j-1} m times to produce $\{\tau_i^j\}_{i=1}^m$.
    Update $D_j = D_{j-1} \cup \left\{(x, \text{expert}(x) : \forall i \in [m], \forall (x, a, \ell) \in \tau_i^j\right\}$.
    Update policy $\pi_j = \mathcal{A}\left(D_j\right)$.
  end for
  return policies $\{\pi_0, \cdots, \pi_N\}$
end procedure

We can then select the final policy to output by checking each of them on a validation set, and returning the one which performs the best.

2.4 Why and How Well Does DAGGER Work?

2.4.1 Error bounds for DAGGER

We can get error bounds by (after some other steps) applying the Azuma-Hoeffding inequality.

Theorem 3. *Error bound for DAGGER (1).*
If $N = \mathcal{O}(T^2 \log(1/\delta))$ (number of total iterations of DAGGER) and $m = \mathcal{O}(1)$ (number of sequences per iteration that we sample, and recall that T is the length of a sequence), then with probability $1 - \delta$ there exists a policy $\hat{\pi}$ of the ones we have discovered so far such that

$$\mathop{\mathbb{E}}_{x \sim d_{\hat{\pi}}} \left[\ell(x, \hat{\pi}) \right] \leq \hat{\epsilon}_N + \mathcal{O}(1/T)$$

where the expectation means we sample states empirically by running the policy $\hat{\pi}$ and average the loss of the policy over those states. $\hat{\epsilon}_N$ means the best empirical policy over the training data. Thus, as $T \to \infty$, we will match the best empirical policy in hindsight over the aggregated data.

It is possible to use only order $(uT)^2$ iterations of the algorithm to get a bound matching the error bounds of the previous section's algorithms.

2.4.2 Why DAGGER Works

DAGGER can essentially be viewed as a Follow-the-Leader algorithm from online learning: We simply take the best possible policy given everything we have seen so far! The fact that the average regret of such an algorithm goes to zero as the number of iterations goes to infinity is the only property needed in the analysis. This property is called the **no regret property**.

The second ingredient we need is a bound on the difference between the distribution generated by the policy that continues to use the expert and the distribution generated by the policy that does not. Supposing that the weight on the policy which continues to use the expert is β_i for iteration i, and there are T steps in the sequence, we can bound the total variation distance between these distributions to be $\leq 2T\beta_i$. Note that when we say "distribution" here, the randomness is really coming from the randomness in the world. In other words, we are assuming that running the same policy from the same starting state may result in seeing different states upon each run.

Then, one of the N policies we learn has the property that its regret on the aggregated data is bounded by the Follow-the-Leader regret plus a term due to the difference in distributions mentioned above. This bound follows by essentially re-writing the aggregated data regret using the definition of the Follow-the-Leader regret bound.

We can also get a finite sample bound by essentially applying Azuma-Hoeffding's inequality to the differences between the expected per-step losses of the learned policy at each iteration. We are trying to replace the total variation bound from before with an estimate which depends on the number of sample sequences in each iteration. The first expected per-step loss is taken with respect to the distribution over the states if the expert is sometimes allowed to be in control, over all of the m trajectories. The second expected per-step loss is taken with respect to a specific trajectory. These differences form a martingale which allow the bound over the difference in distributions to be made concrete: Azuma-Hoeffding makes the difference concrete.

References

[1] H. Daumé III. A Course in Machine Learning. Chapter 18: Imitation Learning Self-published, http://ciml.info/. 2017.

[2] S. Ross, G. Gordon, and J. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 15:627–635, 2011.