Min-Max Entropy Inverse RL of Multiple Tasks

Saurabh Arora[1], Prashant Doshi[1], and Bikramjit Banerjee[2]

***Abstract*— Multi-task IRL recognizes that expert(s) could be switching between multiple ways of solving the same problem, or interleaving demonstrations of multiple tasks. The learner aims to learn the reward functions that individually guide these distinct ways. We present a new method for multi-task IRL that generalizes the well-known maximum entropy approach by combining it with a Dirichlet process based minimum entropy clustering of the observed data. This yields a single nonlinear optimization problem, called MinMaxEnt Multi-task IRL (MME-MTIRL), which can be solved using the Lagrangian relaxation and gradient descent methods. We evaluate MME-MTIRL on the robotic task of sorting onions on a processing line where the expert utilizes multiple ways of detecting and removing blemished onions. The method is able to learn the underlying reward functions to a high level of accuracy and it improves on the previous approaches.**

I. Introduction

Inverse reinforcement learning (IRL) [1]–[3] refers to the problem of ascertaining an agent's preferences from observations of its behavior while executing a task. For instance, observing a human perform a task on the factory line provides information and facilitates learning the task. This passive mode of transferring skills to a collaborative robot (cobot) is appealing because it mitigates costly human effort in not only manually programming the task in a cobot but also in actively teaching the cobot through interventions. The learned preferences can be utilized by a cobot to imitate the observed task [4], or assist the human on it [5].

Motivated by the goal of bringing robotic automation to post-harvest processing lines for vegetables, we focus on the well-defined but challenging task of sorting onions. Our observations of persons engaged in this job in a processing shed attached to a farm revealed commonly used sorting techniques. For example, in addition to the overt technique of individually picking and inspecting the onions as they pass by, we noticed that sometimes the sorters would simply roll the onions (without picking them up) to expose more of their surface. The latter technique allows more onions to be quickly assessed but less accurately. Consequently, the problem of learning the ways to sort onions requires multi-task IRL. This variant of IRL allows the possibility that the demonstrator could be switching between multiple reward functions thereby exhibiting multiple ways (preferences) of solving the given problem or performing multiple tasks. In a previous approach to multi-task Bayesian IRL, DPM-BIRL [6], a Dirichlet process model is used to perform non-parametric clustering of the trajectories, where each cluster corresponds to an underlying reward function. Different from Bayesian IRL, Babes-VRoman et al. [7] apply the iterative EM-based clustering by replacing the mixture of Gaussians with a mixture of reward functions, and a reward function of maximum likelihood is learned for each cluster. Unlike previous approaches, we present a new method for multi-task IRL that generalizes the well-known maximum entropy approach to IRL (MaxEntIRL) [8]. We call our method Min-Max Entropy Multi-Task IRL, or MME-MTIRL. Figure 1 shows the overview of using this method. We formulate the problem as a single entropy-based nonlinear program that combines the MaxEntIRL objective with the objective of finding a cluster assignment distribution having the least entropy. Ziebart et al. [8] demonstrate the advantage that MaxEntIRL brings to single-task IRL in comparison to the Bayesian technique. We expect to leverage this benefit toward multi-task IRL. Taking a different viewpoint, Gleave and Habryka [9] propose regularized MaxEnt multi-task IRL that assumes the target reward functions to lie close to the mean across all tasks, thus transferring information across tasks. Unlike [9], MME-MTIRL has been tested on real-life physical cobot. It does not need task specifications to be close to each other, and it explicitly uses minimum entropy clustering to minimize the number of reward functions needed to explain the observed behavior.

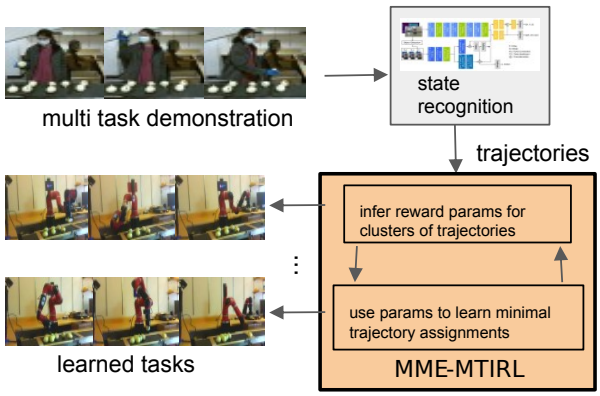


Fig. 1: States are identified using SA-Net [10] from the human demonstration of onion sorting, and the trajectories are given as input to MME-MTIRL method. Learned tasks are demonstrated using the Sawyer cobot.

Modeling multi-task IRL as a single optimization problem enables the direct application of well-studied optimization algorithms (e.g. fast gradient-descent) to this problem. In

[1]Saurabh Arora and Prashant Doshi are with THINC Lab, Dept. of Computer Science, University of Georgia, Athens GA 30606, USA {sa08751,pdoshi}@uga.edu

[2] Bikramjit Banerjee is with School of Computing Sciences and Computer Engineering, University of Southern Mississippi, Hattiesburg, MS 39406, USA Bikramjit.Banerjee@usm.edu

particular, we derive the gradients of the Lagrangian relaxation of the nonlinear program, which then facilitates the use of fast gradient-descent based algorithms. We evaluate the performance of MME-MTIRL in comparison with two previous multi-task IRL techniques, on the problem sorting onions. We show that MME-MTIRL improves on both and learns the reward functions to a high level of accuracy, which allows the collaborative robot Sawyer to observe and reproduce both ways of sorting the onions while making few mistakes. However, we also observed room for improvement in one of the learned behaviors.

II. Background

In IRL, the task of a learner is to find a reward function under which the observed behavior of an expert, with dynamics modeled as an incomplete MDP $\langle S, A, T\rangle$, is optimal [1], [2]. Abbeel and Ng [11] first suggested modeling the reward function as a linear combination of K binary features, ϕ_k, each of which maps a state from the set of states S and an action from the set of expert's actions A to a value in {0,1}. The reward function is then defined as $R(s,a) = \boldsymbol{\theta}^T \phi(s,a) = \sum_{k=1}^{K} \theta_k \cdot \phi_k(s,a)$, where θ_k are the *feature weights* in vector $\boldsymbol{\theta}$. The learner's task is to find a vector $\boldsymbol{\theta}$ that completes the reward function, and thus, the MDP such that the observed behavior is optimal.

Many of the early methods for IRL bias their search to combat the ill-posed nature of IRL and the very large search space [8], [11]. Ziebart et al. [8], taking a contrasting perspective, seeks a distribution over all trajectories (sequences of state-action pairs) that exhibits the maximum entropy while being constrained to match the observed feature counts.

$$\begin{array}{ll} \max_{\Delta} & -\sum_{i=1}^{|\mathbb{Y}|} P(y_i)\ logP(y_i) \\ \textbf{subject to} & \sum_{i=1}^{|\mathbb{Y}|} P(y_i) = 1 \text{ and } E_{\mathbb{Y}}[\phi_k] = \hat{\phi}_k \qquad \forall k \end{array} \tag{1}$$

Here, Δ is the space of all distributions over the set $\mathbb{Y}$ of all trajectories, and $E_{\mathbb{Y}}[\phi_k] = \sum_{i=1}^{|\mathbb{Y}|} P(y_i)\ \sum_{(s,a)\in y_i} \phi_k(s,a)$. Let $\mathcal{Y}$ denote the set of observed trajectories. Then, the right-hand side of the second constraint above becomes, $\hat{\phi}_k = \frac{1}{|\mathcal{Y}|} \sum_{i=1}^{|\mathcal{Y}|} \sum_{(s,a)\in y_i} \phi_k(s,a)$. The problem reduces to finding $\boldsymbol{\theta}$, which parameterizes the exponential distribution that exhibits the highest likelihood.

A. Multi-Task IRL

The same job on a processing line may be performed in one of many ways, each guided by a distinct set of preferences. An expert may switch between these varied behaviors as it performs the job, or multiple experts may interleave to perform the job. If the reward functions producing these behaviors are distinct, then the traditional IRL would yield a single reward function that cannot explain the observed trajectories accurately. However, modeling it as a multi-task problem allows the possibility of learning multiple reward functions. If the number of involved reward functions is predetermined, we may view each unknown reward function as a generative model producing a cluster of trajectories among the observed set. As both, the reward weights and the set of observed trajectories generated by it, are unknown, we may utilize the iterative EM to learn both [7]. But, if the number of reward functions is not known a'priori, multi-task IRL can be viewed as *non-parametric* mixture model clustering, which is typically anchored by a Dirichlet process (DP) [12].

A DP is a stochastic process whose sample paths are drawn from functions that are distributed according to the Dirichlet distribution. DPs find application in Bayesian mixture model clustering [12] due to an interesting property exhibited by distribution $G \sim DP(\alpha, H)$, where α is the concentration parameter and H is a base distribution. Irrespective of whether H is smooth, G is a discrete distribution. Observations θ_d distributed according to G allow us to update the DP. i.i.d. draws of observation θ_d can be seen as cluster parameters. A generic DP-based Bayesian mixture model can be seen as:

$$G|\alpha, H \sim DP(\alpha, H); \quad \theta_i|G \sim G; \quad y_i|\theta_i \sim F(\theta_i)$$

where data y_i has distribution $F(\theta_i)$. Notice the lack of any bound on the number of mixture components. To utilize this mixture model for clustering observed data $\{y_i\}$, we must additionally assign each data point to its originating cluster, and these assignments are drawn from convex mixture weights $(\pi_1, \pi_2, \ldots, \pi_D)$ which are themselves distributed randomly. The number of components D may grow as large as needed. We may then obtain a cluster assignment c_i for data point y_i by sampling distribution G parameterized by event probabilities $\boldsymbol{\pi}$. Then, for each data point

$$\theta_d^*|H \sim H; \qquad c_i|\boldsymbol{\pi} \sim G; \quad y_i|c_i, \{\theta_d^*\} \sim F(\theta_{c_i}^*) \tag{2}$$

where θ_d^* denotes a unique component parameter value.

Choi and Kim [13] utilize this Bayesian mixture model application of a DP toward multi-task IRL. The data points in DPM-BIRL $\{y_i\}$ are the observed trajectories, $\{\theta_d^*\}$ parameterizes the D distinct reward functions, and $F(\theta_{c_i}^*)$ corresponds to $\frac{1}{Z(\theta_{c_i}^*)}\ e^{\sum_{t=1}^{T} Q(s_t, a_t; \theta_{c_i}^*)}$, where T is the fixed length of the trajectory, $Z(\theta_{c_i}^*)$ is the partition function, and H is taken as the Gaussian distribution. Neal [14] discusses several MCMC algorithms for posterior inference on DP-based mixture models, and Choi and Kim select the Metropolis-Hastings.

III. Min-Max Entropy Multi-Task IRL

Ziebart et al. [8] notes a key benefit of the MaxEnt distribution over trajectories over the distribution $F(\theta_{c_i}^*)$ mentioned in Section II-A (which is the prior over trajectories utilized in the Bayesian formulation for IRL [15]), despite their initial similarities. Specifically, the latter formulation, which decomposes the trajectory into its constituent state-action pairs and obtains the probability of each state-action as proportional to the exponentiated Q-function, is vulnerable to the *label bias*. Due to the locality of the action probability computation, the distribution over trajectories is impacted by the number of action choice points (branching) encountered by a trajectory. On the other hand, the MaxEnt distribution does not suffer from this bias. A major consequence of this bias is that Bayesian methods may not assign higher likelihoods to trajectories that have higher rewards, but MaxEnt does. To illustrate this distinction, we conducted experiments in a 10×10 Object World [16] with two desirable objects at

corners, and randomly placed walls. We noted the correlation coefficients (ρ) between the variables X (log-likelihood of a trajectory as assigned by the method) and Y (total reward of a trajectory) for 75 expert trajectories. While ρ_{XY} is 1 for MaxEnt due to a strictly linear relation between X and Y, it is 0.1771 (p-value 0.2914) for Bayesian IRL, showing no significant correlation between X and Y.

This important observation motivates a new method that combines the non-parametric clustering of trajectories and the learning of multiple reward functions by finding trajectory distributions of maximum entropy. This method has the benefit of avoiding label bias.

A. Unified Optimization

A straightforward approach to the combination would be to replace the parametric distribution in MaxEntIRL with $F(\theta^*_{c_i})$ of DP-based mixture model, and the distribution over the trajectories with cluster assignment value c_i. Solving the nonlinear program will yield parameter $\theta^*_{c_i}$ that maximizes the entropy of $F(\theta^*_{c_i})$. Though simple, this approach is inefficient because it requires solving the MaxEnt program repeatedly – each time the DP-based mixture model is updated. As an analytical solution of MaxEnt is not available, the optimization is performed numerically by using either gradient descent [8] or L-BFGS [17].

Instead, we pursue an approach that adds key elements of the DP-based mixture modeling to the nonlinear program of MaxEnt optimization. MaxEnt can learn component parameters $\{\theta_d\}$ (these are the Lagrangian multipliers), which maximize the entropy of the distribution $F(\theta_d)$ over those trajectories whose cluster assignment $c_i = d$. Subsequently, each component distribution F assumes the form of an exponential-family distribution parameterized by θ_d, which is known to exhibit the maximum entropy. For our DPM model, the distribution G is the mixture $\sum_{d=1}^{D} \pi_d \delta_{\theta^*_d}$. To our multi-task max-entropy objective, we add a second objective of finding component weights $\boldsymbol{\pi}$ that exhibit a minimal entropy. The effect of this objective is to learn a minimal number of distinct clusters. More formally, the objective function is

$$\max_{P(y_i|c_i)\in\Delta, \boldsymbol{\pi}\in\Delta} -\sum_{d=1}^{D}\sum_{i=1}^{|\mathbb{Y}|} P(y_i|c_i=d) \log(P(y_i|c_i=d)) + \sum_{d=1}^{D} \pi_d \log(\pi_d).$$

Here $P(y_i|c_i = d)$ can be written as $\delta_d(c_i)Pr_d(y_i)$ where $\delta_d(c_i)$ is the Kronecker delta taking a value of 1 when $c_i = d$, and 0 otherwise, and $Pr_d(y_i)$ is the distribution over all the trajectories for cluster d. The unified nonlinear optimization problem is shown below.

$$\max_{Pr_d(y_i)\in\Delta^{\mathcal{D}}, \boldsymbol{\pi}\in\Delta} -\sum_{d}\sum_{i=1}^{|\mathbb{Y}|} \delta_d(c_i)Pr_d(y_i) \log(\delta_d(c_i)Pr_d(y_i)) + \sum_{d} \pi_d \log(\pi_d) \quad (3)$$

subject to

$$\sum_{d}\sum_{y_i\in\mathbb{Y}} P(y_i, c_i = d) = 1$$
$$E_{\mathbb{Y}}[\phi_k|c_i = d] = \hat{\phi}_{d,k} \qquad \forall d \in D, \forall k \in K$$
$$\sum_{d=1}^{\mathcal{D}} \pi_d = 1 \quad (4)$$

The first constraint above simply ensures that the joint probability distribution sums to 1. The second constraint makes the analogous constraint in MaxEntIRL more specific to matching expectations of feature functions that belong to the reward function of cluster d. Here,

$$E_{\mathbb{Y}}[\phi_k|c_i = d] = \sum_{i=1}^{|\mathbb{Y}|} P(y_i, c_i = d) \sum_{(s,a)\in y_i} \phi_k(s,a)$$
$$= \sum_{i=1}^{|\mathbb{Y}|} P(y_i|c_i = d)\, P(c_i = d) \sum_{(s,a)\in y_i} \phi_k(s,a)$$
$$= \pi_d \sum_{i=1}^{|\mathbb{Y}|} \delta_d(c_i) Pr_d(y_i) \sum_{(s,a)\in y_i} \phi_k(s,a),$$

and $$\hat{\phi}_{d,k} = \frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} \delta_d(c_i) \sum_{(s,a)\in y_i} \phi_k(s,a).$$

Constraint 3 of the program in (4) ensures that the mixture weights are convex. Recall that the DP-based mixture model obtains cluster assignment c_i from mixture weights $\boldsymbol{\pi}$. We may approximate this simulation of c_i simply as $\pi_d = \frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} \delta_d(c_i)$, which is the proportion of observed trajectories currently assigned to cluster c_i. For notational convenience, let us denote $\delta_d(c_i)$ as indicator $v_{d,i}$. We may then rewrite the first constraint as

$$\sum_d \sum_{i=1}^{|\mathbb{Y}|} P(y_i, c_i = d) = 1$$
$$\Leftrightarrow \sum_d \pi_d \sum_{i=1}^{|\mathbb{Y}|} P(y_i|c_i = d) = 1$$
$$\Leftrightarrow \sum_d \pi_d \sum_{i=1}^{|\mathbb{Y}|} \delta_d(c_i) Pr_d(y_i) = 1$$
$$\Leftrightarrow \sum_d \frac{1}{|\mathcal{Y}|} \sum_{i=1}^{|\mathcal{Y}|} v_{d,i} \sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) = 1$$

and the second constraint is rewritten as,

$$E_{\mathbb{Y},d}[\phi_k] = \frac{\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}}{|\mathcal{Y}|} \sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) \sum_{(s,a)\in y_i} \phi_k(s,a) \quad (5)$$

while

$$\hat{\phi}_{d,k} = \frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i} \sum_{(s,a)\in y_i} \phi_k(s,a). \quad (6)$$

Furthermore, we may expand the third constraint of the nonlinear program as follows:

$$\sum_{d=1}^{\mathcal{D}} \pi_d = 1 \Leftrightarrow \sum_{d=1}^{\mathcal{D}} \frac{1}{|\mathcal{Y}|} \sum_{i=1}^{|\mathcal{Y}|} v_{d,i} = 1$$
$$\Leftrightarrow \sum_{i=1}^{|\mathcal{Y}|} \sum_{d=1}^{\mathcal{D}} v_{d,i} = |\mathcal{Y}|$$
$$\Leftrightarrow \sum_{d=1}^{\mathcal{D}} v_{d,i} = 1, \ \forall i$$

The last equivalence follows from the fact that every observed trajectory must belong to a single cluster, and $v_{d,i} \in \{0, 1\}$. The final form of the NLP of (4) is as follows.

$$\max_{Pr_d(y_i)\in\Delta^{\mathcal{D}}, \boldsymbol{v}_d\in\{0,1\}^{|\mathbb{Y}|}} -\sum_{d}\sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) \log(v_{d,i} Pr_d(y_i)) + \frac{1}{|\mathcal{Y}|}\sum_{d}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i} \log\left(\frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}\right)$$

subject to

$$\sum_d \frac{1}{|\mathcal{Y}|} \sum_{i=1}^{|\mathcal{Y}|} v_{d,i} \sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) = 1$$
$$E_{\mathbb{Y},d}[\phi_k] = \hat{\phi}_{d,k} \qquad \forall d \in D, \forall k \in K$$
$$\sum_d v_{d,i} = 1, \forall i \in \{1, \ldots, |\mathbb{Y}|\} \quad (7)$$

where $E_{\mathbb{Y},d}[\phi_k], \hat{\phi}_{d,k}$ are defined in Eqs. 5 and 6.

B. Gradient Descent

The Lagrangian dual for the nonlinear program in (7) is optimized as $\max_{Pr_d, \boldsymbol{v}_d} \min_{\eta, \boldsymbol{\theta}_d, \boldsymbol{\lambda}} \mathcal{L}$ with

$$\begin{aligned}\mathcal{L} = &\left(-\sum_d \sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i)\ \log(v_{d,i} Pr_d(y_i))\right) \\ &+ \left(\frac{1}{|\mathcal{Y}|}\sum_d \sum_{i=1}^{|\mathcal{Y}|} v_{d,i}\ \log\left(\frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}\right)\right) \\ &+ \eta\left(\sum_d \left(\frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}\right)\sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) - 1\right) \\ &+ \sum_{d,k} \boldsymbol{\theta}_{d,k}\Bigg(\left(\frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}\right)\sum_{i=1}^{|\mathbb{Y}|} v_{d,i} Pr_d(y_i) \\ &\sum_{(s,a)\in y_i} \phi_k(s,a) - \frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|} v_{d,i} \sum_{(s,a)\in y_i} \phi_k(s,a)\Bigg) \\ &+ \sum_{i=1}^{|\mathcal{Y}|} \lambda_i \left(\sum_d v_{d,i} - 1\right) \end{aligned} \tag{8}$$

where the multipliers $\eta, \{\lambda_i\}_{y_i \in \mathbb{Y}}$ can be substituted by using relations derived from equating the derivatives of $\mathcal{L}$ w.r.t. the variables of optimization to 0. The target is to learn the multipliers $\boldsymbol{\theta}_d$ (weights for the linear reward function for each learned cluster d) and the variables $v_{d,i}$ (for each trajectory $y_i \in \mathbb{Y}$) that achieve $\max_{Pr_d, \boldsymbol{v}_{d,i}} \min_{\theta_{d,k}} \mathcal{L}$. We achieve the target via gradient ascent for $v_{d,i}$ and descent for $\theta_{d,k}$ using the following partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial \theta_{d,k}} = E_{\mathbb{Y},d}[\phi_k] - \hat{\phi}_{d,k}$$

$$\frac{\partial \mathcal{L}}{\partial v_{d,i}} = \frac{\left(\sum_{i=1}^{|\mathbb{Y}|} P(y_i|c_i = d) + 1 + \frac{|\mathcal{Y}|}{v_{d,i}}(1 - \log Z(\theta_{d,k})\right)}{\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}}$$

where $P(y_i|c_i = d) = \frac{\exp\left(\pi_d \sum_{k=1}^{K} \theta_{d,k} \sum_{(s,a)\in y_i} \phi_k(s,a)\right)}{Z(\theta_{d,k})}$ and $Z(\theta_{d,k}) = \sum_{d\in\mathcal{D}} \pi_d \sum_{i=1}^{|\mathbb{Y}|} \exp\left(\pi_d \sum_{k=1}^{K} \theta_{d,k} \sum_{(s,a)\in y_i} \phi_k(s,a)\right)$. The first derivative is the same as that used for single-task MaxEntIRL. The second derivative indicates that the chances of change in assignment is less if a cluster has many trajectories assigned to it (inversely proportional to $\sum_{i=1}^{|\mathcal{Y}|} v_{d,i}$) and has a higher likelihood of generating trajectories. Due to lack of space, we show the derivations of these gradients in an online appendix located at `https://tinyurl.com/y5l8s2ua`. We approximate $E_{\mathbb{Y},d}[\phi_k]$ as a running average over feature expectations of the trajectories generated by the policy computed using $\boldsymbol{\theta}_d$, which are the reward weights learned for cluster d in the current iteration of gradient descent.

IV. Domain: Robotic Sorting of Onions

Our broader vision is to make it easy to deploy robotic arms on complex processing lines involving manipulation tasks, using IRL. With this vision, we seek to deploy the robotic arm Sawyer for sorting vegetables in processing sheds. Our setup involves a learner robot observing an expert sort onions in a post-harvest processing facility. The expert aims to identify and remove onions with blemishes from the collection of onions present on a static conveyor belt. Blemished onions are dropped in a bin while others are left on the table.

In a visit to a real-world onion processing line attached to a farm, we observed that two distinct sorting techniques were in common use and would be interleaved by the human sorters. Subsequently, we model the expert as acting according to the output of two MDPs both of which share the state and action sets, the transition function and the reward feature functions. They differ in the weights assigned to the features, which yields different behaviors. The specific task is to learn the reward functions underlying the two MDPs.

The state of a sorter is perfectly observed and composed of four factors: *onion* and *gripper location*, *quality prediction*, and *multiple predictions*. Here, an onion's location can be on the sorting table, picked up, under inspection (involves taking it closer to the head), inside the blemished-onion bin, or the onion has been returned to the table post inspection. *Gripper location* is similar but does not include the return back to the table. *Quality prediction* of the onion can be blemished, unblemished, or unknown. The simultaneous predictions for multiple onions is either available or not.

The expert's actions involve focusing attention on a new onion on the table at random, picking it up, bringing the grasped onion closer and inspecting it, placing it in the bin, placing it back on the table, roll its gripper over the onions, and attend to the next onion among those whose quality has been predicted. Reward features are following predicates:

- *ClaimNewOnion*(s,a) action a in state s considers a new onion on table;
- *PickUnknown*(s,a) action a picks an onion with unknown prediction;
- *MakeMultiplePredictions*(s,a) a makes predictions for multiple onions simultaneously by rolling onions;
- *AvoidNoOp*(s,a) the action a changes the state;
- *InspectNewOnion*(s,a) onion is inspected for the first time and a prediction is made for it;
- *GoodOnTable*(s,a) the considered onion is predicted to be unblemished and is placed on the table;
- *BlemishedNotOnTable*(s,a) onion is predicted to be blemished and is not placed on the table;
- *GoodNotInBin*(s,a) onion is predicted to be unblemished and is not placed in the bin;
- *BlemishedInBin*(s,a) onion is predicted to be blemished and is placed in the bin;
- *PickBlemished*(s,a) onion predicted blemished is picked;
- *EmptyList*(s,a) empty the list of predictions by removing blemished onions from table;

Two distinct vectors of real-valued weights on these feature functions yield two distinct reward functions. The MDP with one of these solves to obtain a policy that makes the expert randomly pick an onion from the table, inspect it closely, and place it in the bin if it appears blemished, otherwise place it back on the table. The second reward function yields a policy that has the expert robot roll its gripper over the onions, quickly identify blemished onions, pick only those and place them in the bin. In the MDP, we model classification of blemished and unblemished onions by using a distribution over prediction values. For careful inspection, our distribution assigns higher mass to the correct prediction. For rolling, this mass is lower than that of inspection. This makes the accuracy of careful inspection higher.

V. Experiments

We test the learning performance of different methods by using simulated trajectory data as input, and we compare their sorting performances using human demonstration as input. For the former, we generate the two weight vectors of expert as follows: we collect expert trajectories by executing the behaviors from multiple start states, run an IRL algorithm on the two sets of trajectories to generate the respective weights, and finally verify that using these weights indeed yields the desired behaviors. These "true" weights are denoted $\boldsymbol{\theta}^*_{c_i}$.

Metrics A known metric for evaluating the performance of multi-taskIRL is the *expected value difference averaged over the trajectories* [18], which gives the loss of value if the learner uses the policy obtained by solving the expert's MDP with the learned reward function (parameterized by $\boldsymbol{\theta}^L$) instead of the expert's true policy obtained by solving its MDP with its actual reward function,

$$EVD = \frac{1}{|\mathcal{Y}|}\sum_{i=1}^{|\mathcal{Y}|}\sum_{(s,a)\in y_i} \|V^{\sigma^{\boldsymbol{\theta}^*_{c_i}}}(s) - V^{\sigma^{\boldsymbol{\theta}^L_{c_i}}}(s)\|_1.$$

$\boldsymbol{\theta}^*_{c_i}$ and $\boldsymbol{\theta}^L_{c_i}$ are the true and learned reward weights (component parameters) for the cluster assigned to the observed trajectory y_i, and $\sigma^{\{\cdot\}}$ denotes the corresponding policy.

Another pair of metrics is used to measure the performance of Sawyer on the onion sorting task using the learned reward functions. *Precision* is the ratio of the number of onions placed in the bin that are actually blemished to the total number of onions placed in the bin. *Recall* is the ratio of the number of onions placed in the bin that are actually blemished to the number of onions that are actually blemished (including both in bin and on table). As careful inspection tends to be more accurate than simply rolling over the onions, we expect the behavior of pick-inspect-place to exhibit a higher precision compared to the alternative. On the other hand, it is slower compared to rolling and placing, hence its recall is expected to be lower.

A. Performance Evaluation

We use the metric of EVD as defined previously to measure the performance of MME-MTIRL. Figure 2 (top) shows the average EVDs as the number of input trajectories is increased for MME-MTIRL as well as existing MTIRL baselines, DPM-BIRL [6] and EM-MLIRL [7]. Each data point is the average of 10 runs on the set of trajectories generated as described previously. MME-MTIRL correctly learned two clusters (starting with initial D of 4) in most runs, though a few yielded just one cluster. On the other hand, DPM-BIRL mostly learned three clusters while EM-MLIRL learned two predominantly. For a preliminary demonstration of scalability of MME-MTIRL in the number of learned tasks, we add a (meaningless) third behavior where the sorter keeps considering new onions but does not inspect them, preferring to do nothing. Figure 2 (bottom) shows the average EVDs for learning three tasks. Notice that the MaxEnt based MME-MTIRL exhibits EVDs that are consistently and significantly lower than those of the Bayesian DPM-BIRL. The former correctly learns three clusters in two-thirds of the trajectory sets (otherwise four), while DPM-BIRL varies between three and four. While we do not investigate the reason for DPM-BIRL's relatively poor performance, we note that its EVDs seem to worsen with expanding basis of the MDP's initial state distribution. Unlike the two DP methods, EM-MLIRL performs worse throughout, converging to incorrect reward functions that are likely local optima. In terms of speed of learning, DPM-BIRL is faster than MME-MTIRL, and EM-MLIRL is significantly slower than both. For 96 input trajectories, on average, they take 11.94 seconds, 49.06 seconds, and 265.43 seconds respectively.

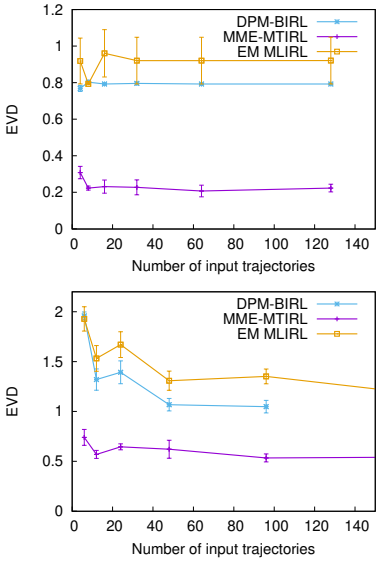
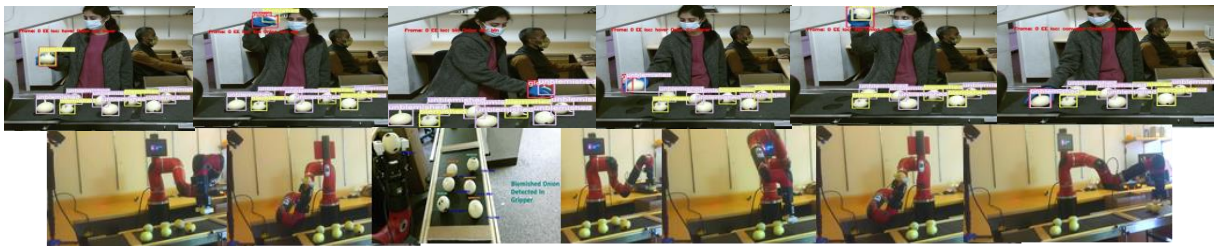


Fig. 2: Average EVDs of MME-MTIRL, DPM-BIRL and EM-MLIRL as the number of trajectories increases, with two demonstrated behaviors (or tasks) **(top)** and three demonstrated behaviors **(bottom)**. Vertical bars are the standard deviations. Note that DPM-BIRL did not successfully terminate for the last data point.

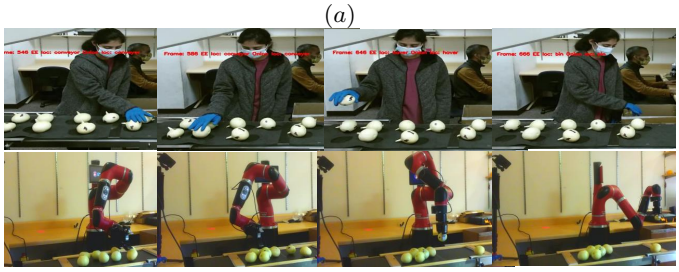
B. Evaluating Sort Performance Using Human Demonstration

Next, we evaluate the performance of the three methods using human demonstration data. We create the domain described in Section IV in our laboratory, and use as training data the videos of both sorting behaviors executed by human sorters (Fig. 3). We utilize SA-Net [10] to process the human demonstrations by identifying the sequences of states from the image frames. The network identifies sequences of states with prediction and listStatus received directly from the object recognition network YOLO [19]. We derive the actions in a trajectory from the state sequence. We use the Sawyer robot, a cobot arm with 7 degrees of freedom and a range of about 1.25m, as the learner executing learned policies. We partially simulate a moving conveyor belt by repeatedly making a collection of onions – some of these are blemished – appear on the table for a fixed amount of time after which the onions disappear. We task Sawyer with sorting as many onions as possible from each collection before it disappears. We utilize the MoveIt motion planner to plan Sawyer's actions.

Does the improvement in learning translate to improved performance in the sorting task? In Table I, we show the average precision and recall of the expert engaged in using



(a)



(b)

Fig. 3: Human demonstration and learned behaviors executed by Sawyer of the two sorting techniques: (a) pick-inspect-place (picks each onion, inspect it closely), (b) roll-pick-place (roll them, expose hidden surfaces, to classify many onions simultaneously). The process of identification of states in demonstration involves tracking the locations of claimed onion and hand (referred as EE), prediction of claimed onion, and list of blemished onions. First two can be at four locations: conveyor, hover, front of eyes, and bin; and last two are derived directly from YOLO's output.

		(TP,FP,TN,FN)	**P%, R%**
Expert	P-I-P	(7,0,12,5)	100.00, 58.33
	R-P-P	(9,4,8,3)	69.23, 75.00
Learned (MME-MTIRL)	P-I-P	(7,1,11,5)	87.50, 58.33
	R-P-P	(9,4,8,3)	69.23, 75.00
Learned (DPM-BIRL)	P-I-P	(7,3,9,5)	70.00, 58.33
	R-P-P	(9,4,8,3)	69.23, 75.00
Learned (EM-MLIRL)	P-I-P	(6,3,9,6)	66.67, 50.00
	R-P-P	(6,4,8,6)	60.00, 50.00

TABLE I: P-I-P and R-P-P stand for Pick-inspect-place and Roll-pick-place resp. Column labels TP denotes true positive (# blemished onions in bin), FP denotes false positive (# good onions in bin), TN denotes true negatives (# good onions remaining on table), and FN denotes false negatives (# blemished onions remaining on table). P and R denote precision (= TP/(TP+FP)) and recall (= TP/(TP+FN)) in %, respectively.

the two sorting techniques and the analogous metrics for the learned behaviors using all three IRL approaches. For each of the three IRL methods, we compute the learned policies for two behaviors by using the feature weights averaged over 10 runs of learning. Then we execute each policy in the physical domain for three moving sets of eight onions per set – four blemished and four good – giving as output the precision and the recall corresponding to each learned behavior. The performance of the behaviors learned by MME-MTIRL is closer to that of the expert's than those learned by the two baseline methods. Learned pick-inspect-place behavior shows high precision but leaves many onions on the table leading to worse recall. This may happen because of incorrect weights learned for *PickUnKnown(s,a) PickBlemished(s,a)* features. The former feature should have the higher weight for pick-inspect-place and latter feature should be dominant for roll-pick-place. Due to inaccurate learning, the cobot sometimes repeats picking and placing a blemished onion without inspecting it. On the other hand, the roll-pick-place behavior is learned satisfactorily and exhibits precision and recall close to those of the true behavior. Finally, we also show in Fig. 3 and the associated video Sawyer executing both of the onion-sorting behaviors autonomously. It uses Kinect-v2 point-cloud to track the onion locations and YOLO to classify onions held in the gripper.

VI. Concluding Remarks

An expert may solve a problem in multiple distinct ways, each of which optimizes a different reward function while still sharing the features. For IRL to remain relevant, it should generalize to not only learn how many distinct reward functions are present in the demonstration, but also to learn the parameters of each. We presented a new multi-task IRL method that combines MaxEnt IRL – a key IRL technique – with elements of the DP-based Bayesian mixture model. While minimizing the number of behaviors learned to explain the observations, it leverages the advantages of MaxEnt IRL and facilitates solving the generalization as a single unified optimization problem. On a real-world inspired domain, we showed that it improves on previous multi-task IRL methods. The behaviors induced by the learned reward functions imitated the observed ones for the most part. Having established the value of combining MaxEnt with multi-task IRL in this paper, a next step could be to make multi-task IRL online. On the theoretical front, a future contribution could be sample complexity bounds for the method.

Acknowledgments: We thank Farah Saeed and Kenneth Bogert for help in the experimentation. Our work was enabled in part by NSF grants IIS-1830421 (to PD), IIS-1526813 (to BB), and a Phase 1 grant from the GA Research Alliance.

REFERENCES

[1] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Seventeenth International Conference on Machine Learning*, 2000, pp. 663–670.

[2] S. Russell, "Learning agents for uncertain environments (extended abstract)," in *Eleventh Annual Conference on Computational Learning Theory*, 1998, pp. 101–103.

[3] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *CoRR*, vol. abs/1806.06877, 2018.

[4] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[5] M. Trivedi and P. Doshi, "Inverse learning of robot behavior for collaborative planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[6] J. Choi and K.-E. Kim, "Nonparametric bayesian inverse reinforcement learning for multiple reward functions," in *25th International Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 305–313.

[7] M. Babes-Vroman, V. Marivate, K. Subramanian, and M. Littman, "Apprenticeship learning about multiple intentions," in *28th International Conference on Machine Learning (ICML)*, 2011, pp. 897–904.

[8] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *23rd National Conference on Artificial Intelligence - Volume 3*, 2008, pp. 1433–1438.

[9] A. Gleave and O. Habryka, "Multi-task maximum entropy inverse reinforcement learning," *arXiv preprint*, no. arXiv:1805.08882, 2018.

[10] N. Soans, E. Asali, Y. Hong, and P. Doshi, "Sa-net: Robust state-action recognition for learning from observations," in *2020 IEEE International Conference on Robotics and Automation, ICRA*, 2020, pp. 2153–2159.

[11] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Twenty-first International Conference on Machine Learning (ICML)*, 2004, pp. 1–8.

[12] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin, *Bayesian Data Analysis*, 3rd ed. CRC Press, 2013.

[13] J. Choi and K.-E. Kim, "Bayesian nonparametric feature construction for inverse reinforcement learning," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13. AAAI Press, 2013, pp. 1287–1293.

[14] R. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, 2000.

[15] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *20th International Joint Conference on Artifical Intelligence (IJCAI)*, 2007, pp. 2586–2591.

[16] S. Levine, Z. Popović, and V. Koltun, "Nonlinear inverse reinforcement learning with gaussian processes," in *24th International Conference on Neural Information Processing Systems (NIPS)*, 2011, pp. 19–27.

[17] K. Bogert and P. Doshi, "Multi-robot inverse reinforcement learning under occlusion with interactions," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '14, 2014, pp. 173–180.

[18] J. Choi and K.-E. Kim, "Inverse reinforcement learning in partially observable environments," *J. Mach. Learn. Res.*, vol. 12, pp. 691–730, 2011.

[19] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Yu, changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, "ultralytics/yolov5: v3.1," Oct. 2020.