Recognition of Agents based on Observation of Their Sequential Behavior

Qifeng Qiao and Peter A. Beling

Department of Systems Engineering
University of Virginia, VA, USA
{qq2r, pb3a}@virginia.edu

Abstract. We study the use of inverse reinforcement learning (IRL) as a tool for recognition of agents on the basis of observation of their sequential decision behavior. We model the problem faced by the agents as a Markov decision process (MDP) and model the observed behavior of an agent in terms of forward planning for the MDP. The reality of the agent's decision problem and process may not be expressed by the MDP and its policy, but we interpret the observation as optimal actions in the MDP. We use IRL to learn reward functions for the MDP and then use these reward functions as the basis for clustering or classification models. Experimental studies with *GridWorld*, a navigation problem, and the *secretary problem*, an optimal stopping problem, show algorithms' performance in different learning scenarios for agent recognition where the agents' underlying decision strategy may be expressed by the MDP policy or not. Empirical comparisons of our method with several existing IRL algorithms and with direct methods that use feature statistics observed in state-action space suggest it may be superior for agent recognition problems, particularly when the state space is large but the length of the observed decision trajectory is small.

1 Introduction

The availability of sensing technologies, such as digital cameras, global position system, infrared sensors, web technology and others, makes the computer easily access varieties of data recording the interaction between agents and the environment. As summarized in Figure 1, research in learning from the observed behavior has seen the development of approaches to activity recognition (It may be referred as different terms within the published literature, including plan recognition and goal recognition) and learning from demonstrations (It may be referred as imitation learning in other fields):

– Activity recognition: an activity can be described as a specific event or a combination of events, e.g. "go to bed", "cook a breakfast", "read a book" for the study of human activity recognition. The goal in activity recognition is a special event so that some optimal plan for a goal is compatible with the observations. A plan represents a mapping between state of a decision problem and action of an agent. The goal may change or it may consist of several

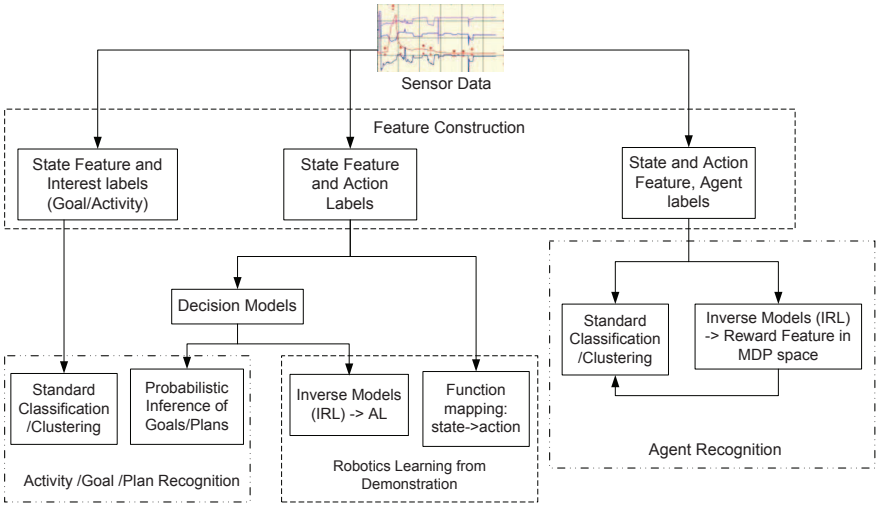


Fig. 1: Overview and Categorization of problems of learning from observation of decision-making behavior, including the widely studied problems that infer the goals of an agent and that learn how to make decisions, as well as our proposed new problem of recognizing agents based on their decision behavior.

sub-activities or sub-goals. One may recognize an activity by applying classification/clustering algorithms directly to the feature vectors constructed from the observation data [22]. Alternatively, given a plan library or a set of goals as a prior, the entire trace of actions can be recognized and matched against a plan library or a set of possible goals [16]. Despite of the success of these methods, they assume that the plan library, a set of possible goals or some behavior model are known beforehand and provided as an input. Goal information is often completely unknown in practice, however, and so it is difficult to model the goal precisely.

– Learning from demonstrations: Much of work is focused on approximating the function mapping from observed experts' states to actions [2]. Alternatively, one may use demonstration data to inverse a decision model and a policy is then derived using this model, e.g. apprenticeship learning [4].

However, in practical applications we may not only be interested in reverse engineering of a decision-making process or imitating a behavior (identification), but also in determining whether two agents correspond to the same behavior pattern (clustering), or which decision-making pattern is being observed in an agent (classification). In this paper, we propose a new problem, termed *Behavior-based Agent Recognition*(BAR), that involves recognizing agents based on observation of their sequential behavior, instead of recognizing activities or actions.

This new problem is also motivated by varieties of applications in the real-world. E.g., we may find a way to train drivers by classifying the observed drivers

into defensive driving and aggressive driving, even those drivers may have similar activities or goals, such as avoid driving over curbs or a collision. In the e-commerce market, if the web site can automatically categorize users based on observation of their on-line behavior, such as which buttons have been clicked by an user on which web page with what advertisements, a similar marketing strategy may work successfully on people in the same group. Another motivation comes from domains like high frequency trading of stocks and commodities, where there is considerable interest in identifying new market players and algorithms based on observations of trading actions, but little hope in learning the precise strategies employed by these agents [24][14].

A direct approach to BAR problem is to program some heuristic rules to recognize agents by decomposing complex behavior into a series of simple events and then evaluating them to reach a conclusion. However, programming the rules is hard. Alternatively, we can construct a feature vector to characterize agents based on the observed behavior, and then categorize the agents using these feature vectors. Consider image recognition as an illustration of this method. A computer learns to categorize images by representing every image as a multi-dimensional feature vector that consists of the components such as RGB color, texture, or other metrics. Here, an agent is an object like an image, and the feature vector needs to be constructed from the observed behavior. The key point in this process is how to find a high-level vector that can represent the underlying decision-making process. If the decision problem can be cast in the MDP framework, we propose to represent the agents with the reward functions of MDP models because they effectively influence the forward planning process.

IRL [13] addresses the task of finding a reward function that explains the observed behavior via the forward planning of a MDP. The observed behavior is assumed to maximize the long-term accumulative reward for that MDP. Most of recent work in IRL is focused on apprenticeship learning (AL), in which IRL is used as the core method for finding decision policies consistent with observed behavior [1] [21]. A number of IRL algorithms, being designed for apprenticeship learning or imitation learning, includes Max-margin planning [17], gradient tuning methods [12], linear solvable MDP [10], bootstrap learning [5], Gaussian process IRL [15] and Bayesian inference [6].

Our main contribution is to propose a new problem called BAR, and present a method based on IRL that models the problem faced by the agents as a MDP and assumes the reward function of the MDP model as a high-level abstraction of the decision behavior. The motivation is that even when the agent's behavior is not rational and we hardly learn the precise goals/plans, we still may categorize agents by using reward functions learned from the observed behavior.

On two well-known sequential decision-making problems, we compare our method with several existing IRL algorithms and with direct methods that use feature statistics observed in state-action space. The results show that our method using reward functions provides a formal way to solve the agent recognition problem and performs superior to other methods.

2 Preliminaries

We define the input of BAR problem as a tuple $\mathbf{B} = (D_1, D_2, \dots D_N)$, where $D_n, n \in \{1, 2, \dots, N\}$ is the observation of the n-th agent. For a classification problem, $D_n = (\mathcal{O}_n, y_n)$, where $\mathcal{O}_n$ is a set of observed decision trajectories and y_n is the class label for the n-th agent. The agents, who have the same decision-making pattern, are given the same class label. Similarly, for a clustering problem, D_n only consists of the observed decision trajectories $\mathcal{O}_n$.

We define the set of decision trajectories $\mathcal{O}_n = \{h_n^j\},\ j = 1, 2, \dots, |\mathcal{O}_n|$, where each trajectory h_n^j is defined as a series of state and action pairs: $\{(s, a)_n^t\}, t = 1, 2, \dots, |h_n^j|$. Here, the s denotes the state for the decision problem and the a means the action selected by the agent at state s. Below are two definitions about the agent recognition problem.

Definition 1. *In general, an classification problem is: given a decision problem where the observed behavior $\mathcal{O}_n$ comes from, a universe $\mathcal{F}$ where the examples come from(the observed behavior is expressed in this space), a fixed set of classes $\mathcal{Y}$, and a training set X of labeled agents whose element $x \in \mathcal{F} \times \mathcal{Y}$, we use a learning algorithm to find a function $g : \mathcal{F} \to \mathcal{Y}$.*

Definition 2. *Given the observed behavior $\{\mathcal{O}_n\}_{n=1}^N$, a universe $\mathcal{F}$ where the examples come from, and a training set $X = \{f_1, f_2, \dots, f_N\}$, where $f_n \in \mathcal{F}, n \in \{1, \dots, N\}$, the clustering of N agents is the partitioning of X into K clusters $\{C_1, C_2, \dots, C_K\}$ that satisfies (1) $\cup_{k=1}^K C_k = X$; (2) $C_k \neq \phi, k = 1, 2, \dots, K$; (3) $C_k \cap C_{k'} = \phi,\ \forall k \neq k'$ and $k, k' \in \{1, 2, \dots, K\}$.*

Next, we present the approaches to BAR problem in Section 3 and Section 4, and review the related IRL algorithms that have been used within our model in Section 5.

3 Two Direct Agent Recognition Models

In this section, we describe two approaches to agent recognition problem that construct feature vectors directly with the raw observation data.

The first method is called feature trajectory (FT). Consider a decision trajectory h_n^j. The vector to characterize the behavior in j-th decision trajectory is written as follows.

$$f(h_n^j) = [s_1, a_1, s_2, a_2, \dots, s_{|h_n^j|}, a_{|h_n^j|}],$$

where $s_i, i \in \{1, 2, \dots, |h_n^j|\}$ is a discrete random variable meaning the state index at i-th decision stage, and a_i represents the action selected at state s_i. E.g., we have a problem that can be defined by 3 states and 2 actions. Then $s_i \in \{1, 2, 3\}$ and $a_i \in \{1, 2\}$. In the observation, every trajectory starts from the same initial state. Given the observation set $\mathcal{O}_n$ for n-th agent, the feature vector f_n is obtained by computing this equation: $f_n = \frac{1}{|\mathcal{O}_n|} \sum_{j=1}^{|\mathcal{O}_n|} f(h_n^j)$, where the vector $f(h_n^j)$ is preprocessed by scale-normalization before averaging.

Then, the n-th agent is represented by a feature vector f_n. Consider a supervised learning problem. Given a real valued input vector $f_n \in \mathcal{F}$ and a category label $y_n \in \mathcal{Y}$, we aim to learn a function $g : \mathcal{F} \to \mathcal{Y}$.

The second method is called feature expectation (FE), which has been widely used by apprenticeship learning as a representation of the averaged long-term performance. Assume a basis function $\phi : \mathcal{S} \to [0,1]^d$, where $\mathcal{S}$ denotes the state space. The feature expectation $f_n = \frac{1}{|\mathcal{O}_n|} \sum_{j=1}^{|\mathcal{O}_n|} \sum_{s_t \in h_n^j} \gamma^t \phi(s_t)$, where $\gamma \in (0,1)$ is a discount factor. The associated apprenticeship learning algorithms aim to find a policy that performs as well as demonstrations by minimizing the distance between their feature expectations. Here, we only use the observed state sequence to compute the feature expectation vector for an agent, where the γ is manually defined constant, e.g. 0.95. Then, the n-th agent can be represented by the vector f_n that is built on $\mathcal{O}_n$.

4 A Behavior-based Agent Recognition Model

First, we briefly review some background about MDP necessary for the next proposed method.

A finite MDP model $M = (\mathcal{S}, \mathcal{A}, R, \gamma, \mathcal{P})$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, R is the reward function, γ is the discount factor, and $\mathcal{P} = \{\mathbf{P}_a\}_{a \in \mathcal{A}}$ is a set of transition probability matrices. The entries of $\mathbf{P}_a$, written as $\mathbf{P}_a(s, s')$, give the probability of transitioning to state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ given the action is a. The rows of $\mathbf{P}_a$, denoted $\mathbf{P}_a(s, :)$, give a probability vector of transitioning from state s to all the states in $\mathcal{S}$. In a finite state space the reward function R may be considered as a vector, r, whose elements give the reward in each state.

In the MDP, a stationary policy is a function $\pi : \mathcal{S} \to A$. The value function for a policy π is $V^\pi(s_0) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | p(s_0), \pi]$ where $p(s_0)$ is the distribution of the initial state and the action at state s_t is determined by policy π. Similarly, the Q function is defined as $Q(s,a) = R(s) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}_a(s, s') V^\pi(s')$. At state s, an optimal action is selected by $a^* = \max_{a \in \mathcal{A}} Q(s, a)$.

Then, an instance of the IRL problem is written as a triplet $B = (M \setminus r, p(r), \mathcal{O})$, where $M \setminus r$ is a MDP model without the reward function and $p(r)$ is prior knowledge on the reward. The vector $p(r)$ can be a non-informative prior if we have no knowledge about the reward, or a Gaussian or other distribution if we model the reward as a specific stochastic process. Later in Section 5, we present the details for Bayesian IRL that has been used in our experiments.

Our behavior-based agent recognition method proceeds as follows.

1. Given the BAR problem with input $\mathbf{B}$, we use the set $\{\mathcal{O}_n\}, n \in \{1, 2, \ldots, N\}$ to construct the state space $\mathcal{S}$ and action space $\mathcal{A}$ for the decision-making problem. The $\mathcal{P}$ can be modelled using prior knowledge of the problem or estimated from the observed decision trajectories.
2. For every agent, we construct a MDP model, no matter whether the optimal policy of this MDP can match the observed behavior.

3. Apply IRL algorithms to learn the reward vector r_n for n-th agent.
4. Given the training set $\{r_1, r_2, \ldots\}$, where $r_n \in \mathcal{F}$ and the corresponding category label $y_n \in \mathcal{Y}$, we aim to train a classifier $g : \mathcal{F} \to \mathcal{Y}$.
5. Given a new agent, we repeat step 1-3 to get the reward vector for the agent and then predict the label for the behavior pattern using function $g : \mathcal{F} \to \mathcal{Y}$.

We use a MDP to model the decision problem faced by an agent under observation. The reality of the agent's decision problem and process may differ from the MDP model, but we interpret every observed decision of the agent as the choice of an action in the MDP. The dynamics of the environment in the MDP are described by the transition probabilities $\mathcal{P}$. These probabilities may be interpreted as being a prior, if known in advance, or as an estimation of the agent's beliefs of the dynamics. Next, we will show how to learn the reward functions by employing some exiting IRL algorithms.

5 Bayesian Framework for IRL

Most existing IRL algorithms have some assumption about the form of the reward function. Prominent examples include the model in [13], which we term linear IRL (*LIRL*) because of its linear nature, *WMAL* in [21], and *PROJ* in [1]. In these algorithms, the reward function is written linearly in terms of features as $R(s) = \sum_{i=1}^{d} \omega_i \phi_i(s) = \omega^T \phi(s)$, where $\phi : \mathcal{S} \to [0,1]^d$ and $\omega^T = [\omega_1, \omega_2, \cdots, \omega_d]$.

Our computational framework uses Bayesian IRL to estimate the reward vectors in a MDP, which was initially proposed in [8]. The posterior over reward function for n-th agent is written as

$$p(r_n|\mathcal{O}_n) = p(\mathcal{O}_n|r_n)p(r_n) \propto \prod_{j=1}^{|\mathcal{O}_n|} \prod_{(s,a)\in h_n^j} p(a|s, r_n).$$

Then, the IRL problem is written as $\max_{r_n} \log p(\mathcal{O}_n|r_n) + \log p(r_n)$. For many problems, however, the computation of $p(r_n|\mathcal{O}_n)$ may be complicated and some algorithms use Markov chain Monte Carlo (MCMC) to sample the posterior probability. Considering the computation complexity to deal with a large number of IRL problems, we choose two IRL algorithms that have well defined likelihood function to reduce the computation cost, which are shown in the following subsections. The first algorithm in Section 5.1 has two assumptions on the reward functions: (1) it can be written linearly in terms of the state features; (2) it only depends on state. The second algorithm in Section 5.2 doesn't have these restrictions, and it not only can model the reward functions in nonlinear form but also consider the reward affected by both state and action.

5.1 IRL with Boltzmann Distribution

The IRL algorithm in [3], which we call maximum likelihood IRL (*MLIRL*), uses Boltzmann distribution to model likelihood function using $p(a|s, r_n) = \frac{e^{\beta Q(s,a)}}{\sum_{a\in\mathcal{A}} e^{\beta Q(s,a)}}$, where β denotes the degree of decision-making confidence.

The likelihood function is optimized via gradient ascent method as follows.

1. Initialize: Choose random set of reward weights ω_1.
2. Iterate for $t = 1$ to M do: Compute $Q(s,a)$ and $p(a|s,r_n)$ using ω_t; $L = \sum_{j=1}^{|\mathcal{O}_n|} \sum_{(s,a)\in h_n^j} \log p(a|s,r_n)$; $\omega_{t+1} \leftarrow \omega_t + \alpha_t \bigtriangledown_\omega L$.
3. Output reward $r_n = \omega_M^T \phi(s)$ for n-th agent.

The parameters β and M need to be defined as constants.

5.2 IRL with Gaussian Process

IRL algorithm, which is called *GPIRL* in [15], uses preference relations to model the likelihood function $P(\mathcal{O}_n|r_n)$ and assumes the r_n is generated by Gaussian process for n-th observed agent.

Given a state, we assume an optimal action is selected according to Bellman optimality. At state s, $\forall \hat{a}, \check{a} \in \mathcal{A}$, we define the *action preference relation* as:

- Action $\hat{a}$ is weakly preferred to $\check{a}$, denoted as $\hat{a} \succeq_s \check{a}$, if $Q(s,\hat{a}) \geq Q(s,\check{a})$;
- Action $\hat{a}$ is strictly preferred to $\check{a}$, denoted as $\hat{a} \succ_s \check{a}$, if $Q(s,\hat{a}) > Q(s,\check{a})$;
- Action $\hat{a}$ is equivalent to $\check{a}$, denoted as $\hat{a} \sim_s \check{a}$, if and only if $\hat{a} \succeq_s \check{a}$ and $\check{a} \succeq_s \hat{a}$.

Given the observation set $\mathcal{O}_n$, we have a group of preference relations at each state s, which is written as

$$\mathcal{E} \equiv \left\{(\hat{a} \succ_s \check{a}),\ \hat{a} \in \hat{\mathcal{A}},\ \check{a} \in \mathcal{A} \setminus \hat{\mathcal{A}}\right\} \cup \left\{(\hat{a} \sim_s \hat{a}'),\ \hat{a}, \hat{a}' \in \hat{\mathcal{A}}\right\},$$

where $\hat{\mathcal{A}}$ is the action subspace from observation $\mathcal{O}_n$.

Then, the likelihood function $p(\mathcal{O}_n|r_n) = \prod p(\hat{a} \succ_s \check{a}) \prod p(\hat{a} \sim_s \hat{a}')$. The models of $p(\hat{a} \succ_s \check{a})$ and $p(\hat{a} \sim_s \hat{a}')$ are defined in [15].

Let $\mathbf{r}$ be the vector of r_n containing the reward for $M = |\mathcal{A}|$ possible actions at T observed states. We have

$$\begin{aligned}\mathbf{r} &= (\underbrace{\mathbf{r}_1(s_1), ..., \mathbf{r}_1(s_T)}, \ldots, \underbrace{\mathbf{r}_M(s_1), \ldots, \mathbf{r}_M(s_T)}) \\ &= (\qquad\quad \mathbf{r}_1, \qquad \cdots, \qquad\quad \mathbf{r}_M),\end{aligned}$$

where $T = |\mathcal{S}|$ and $\mathbf{r}_m, \forall m \in \{1, 2, \ldots, M\}$, denotes the reward for action a_m.

Consider $\mathbf{r}_m$ as a Gaussian process. We denote by $k_m(s_i, s_j)$ the function generating the value of entry (i,j) for covariance matrix $\mathbf{K}_m$, which leads to $\mathbf{r}_m \sim N(0, \mathbf{K}_m)$. Then the joint prior probability of the reward is a product of multivariate Gaussian, namely $p(\mathbf{r}|\mathcal{S}) = \prod_{m=1}^{M} p(\mathbf{r}_m|\mathcal{S})$ and $\mathbf{r} \sim N(0, \mathbf{K})$. Note that $\mathbf{r}$ is completely specified by the positive definite covariance matrix $\mathbf{K}$, which is block diagonal in the covariance matrices $\{\mathbf{K}_1, \mathbf{K}_2 ..., \mathbf{K}_M\}$ based on the assumption that the reward latent processes are uncorrelated . In practice, we use a squared exponential kernel function, written as:

$$k_m(s_i, s_j) = e^{\frac{1}{2}(s_i - s_j)\mathbf{M}_m(s_i - s_j)} + \sigma_m^2 \delta(s_i, s_j),$$

where $\mathbf{M}_m = \kappa_m \mathbf{I}$ and $\mathbf{I}$ is an identity matrix. The function $\delta(s_i, s_j) = 1$, when $s_i = s_j$; otherwise $\delta(s_i, s_j) = 0$. Under this definition the covariance is almost unity between variables whose inputs are very close in the Euclidean space, and decreases as their distance increases.

Then, the *GPIRL* algorithm estimates the reward function by iteratively conducting the following two main steps:

1. Get estimation of $\mathbf{r}_{MAP}$ by maximizing the posterior $p(r_n|\mathcal{O}_n)$, which is equal to minimize $-\log p(\mathcal{O}_n|r_n) - \log p(r_n|\theta)$, where $\theta = (\kappa_m, \sigma_m)_{m=1}^{M}$ is the hyper-parameter controlling the Gaussian process. Above optimization problem has been proved to be convex programming in [15].
2. Optimize the hyper-parameters by using gradient decent method to maximize $\log p(\mathcal{O}_n|\theta, \mathbf{r}_{MAP})$, which is the Laplace approximation of $p(\theta|\mathcal{O}_n)$.

6 Experimentation

Our experiments simulate agent recognition problems and compare several IRL algorithms against the methods that construct feature vectors from raw observation data. We study two problems, *GridWorld* and the *secretary problem*. *GridWorld* sheds light on the task of recognizing agents whose underlying decision strategy can be matched by a policy of the MDP model. The secretary problem provides a more practical environment in which agents' true decision strategy may not be explained or expressed by any policy of the MDP that is used to model the decision-making problem. Agents in the secretary problem employ heuristic decision rules derived from experimental study of human behavior in psychology and economics.

To evaluate the recognition performance, we use the following algorithms: (1) Clustering: Kmeans [9]; (2) Classification: Support vector machine (SVM), K-nearest neighbours (KNN), Fisher discriminant analysis (FDA) and logistic regression (LR) [9]. We use clustering accuracy [23] and Normalized Mutual Information (NMI) [20] to compare clustering results.

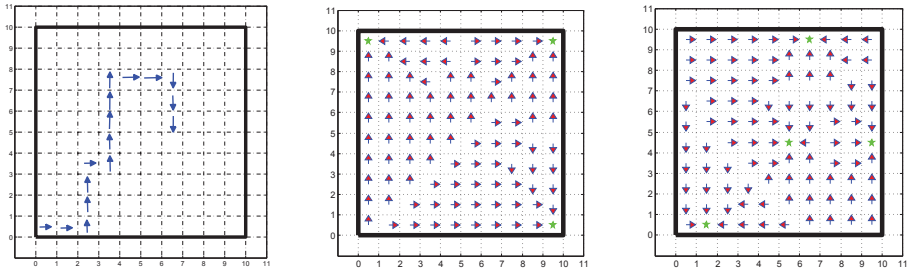
6.1 *GridWorld* Problem

In the *GridWorld* problem, which is used as a benchmark experiment by Ng and Russell in [13], an agent starts from a given square and moves towards a destination square. The agent has five actions to take: moving in the four cardinal directions or staying put. With probability 0.65 the agent moves to its chosen location, with probability 0.15 it stays in the same location regardless of chosen action, and with probability 0.2 it moves in a random cardinal direction.

The small *GridWorld* has been widely used as a test domain by most of IRL algorithms. The observation data is collected when an agent is moving in the grid world. From the observation, the reward is learned to make the optimal policy of a MDP match the observed behavior. We investigate the agent recognition problem in terms of clustering and classification on a 10×10 *GridWorld* problem. Experiments are conducted according to the steps in Algorithm 1.

Algorithm 1 *GridWorld* experimentation steps

1: Input the variables $\mathcal{S}, \mathcal{A}, \mathcal{P}$, and two policies π_1 and π_2.
2: **for** $i = 1 \rightarrow 2$ **do**
3: **for** $j = 1 \rightarrow 200$ **do**
4: Model an agent's action selection using π_i + random Gaussian noise. With probability 0.65 the agent executes the selected action.
5: Sample decision trajectories $\mathcal{O}_{ij}$, and make the ground truth label $y_{ij} = 0$, if $i = 1$; $y_{ij} = 1$, if $i = 2$.
6: IRL has access to the problem $B = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{O}_{ij})$ for this agent, and then infers the reward r_{ij}.
7: **end for**
8: **end for**
9: Recognize these agents based on the reward r_{ij}.



(a) An observed path (b) Policy π_1 for Class 1 (c) Policy π_2 for Class 2

Fig. 2: The (a) shows an observed decision trajectory. The (b) and (c) illustrate underlying decision policy for two classes of agents. Colored arrow denotes the observed action.

For the input of the experimentation, we simulate two groups of agents, and make each group have 200 agents who adopt similar decision strategy moving in the grid world. Figure 2 (b) and (c) display the underlying policy used by two groups. Each policy represents a decision-making pattern, e.g. a group of agents may prefer the routes close to the border where the scenery is more attractive, while the other group may like passing by the center to avoid traffic. In each group, an agent's decision is simulated by adding Gaussian noise to his/her group's underlying policy. Here, agents may have multiple destinations to visit. Though these agents may have the same goal such as arriving at the destination in the shortest time, their decision patterns can still be different.

In the experiments, we find that a small number of short decision trajectories tends to present challenges to action feature methods, which is an observation of particular interest. Additionally, the length of trajectories may have a substantial impact on performance. If the length is so long that the observed agent reaches the destination in every trajectory, the problem can be easily solved based on

observations. Thus, we evaluate and compare performance by making the length of decision trajectory small.

Table 1: NMI scores

$\|\mathcal{O}_n\|$	FE	FT	PROJ	GPIRL
4	0.0077	0.0012	0.0068	0.0078
8	0.0114	0.0016	0.0130	0.0932
16	0.0177	0.0014	0.0165	0.7751
20	0.0340	0.0573	0.0243	0.8113
30	0.0321	0.0273	0.0365	0.8119
40	0.0361	0.0459	0.0389	0.8123
60	0.0387	0.0467	0.0388	0.8149
80	0.0441	0.1079	0.0421	0.8095
100	0.0434	0.1277	0.0478	0.8149
200	0.0502	0.1649	0.0498	0.8149

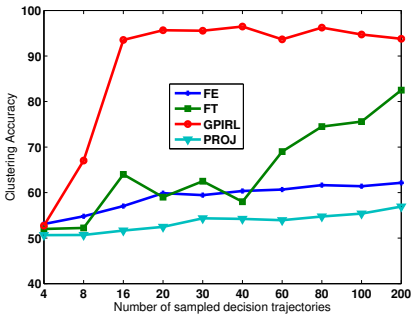
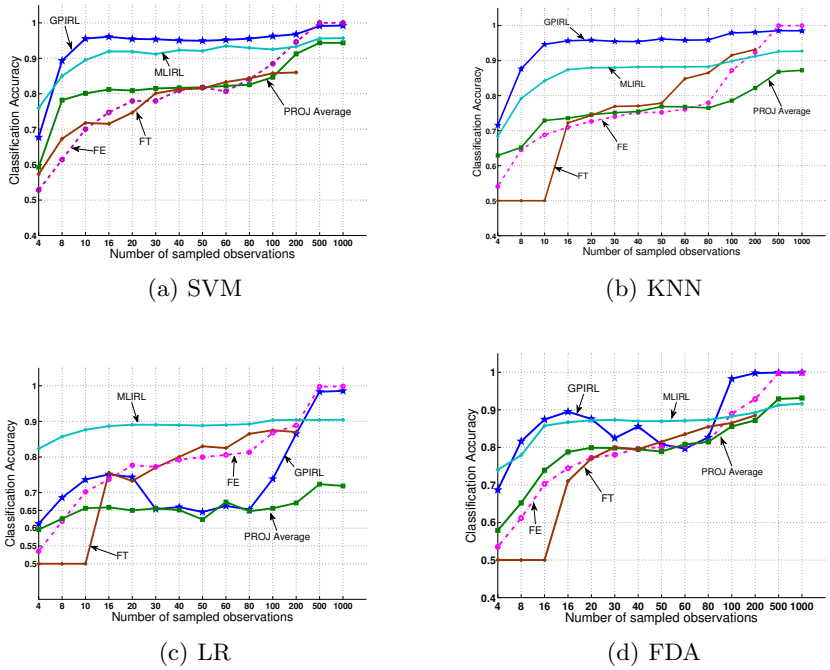


Fig. 3: Clustering accuracy

Table 6.1 displays NMI scores and Fig.3 shows clustering accuracy. The length of the trajectory is limited to six steps, as we assume the observation is incomplete and the learner does not have sufficient information to differentiate behavior directly. Results are averaged over 100 replications. Clustering performance improves with increasing number of observations. When the number of observations is small, *GPIRL* method achieves high clustering accuracy and NMI scores due to the advantage of finding more accurate reward functions that can well characterize the decision behavior. The IRL algorithms based on feature expectation vectors, such as *PROJ*, are not effective in this problem because the length of the observed decision trajectory is too small to accumulate enough observations that correctly approximate the long-term goal.

Considering the utilization of feature learning algorithms to improve the simple feature representations, we also run experiments with PCA-based features where the projection sub-space is spanned by those eigenvectors that correspond to the principal components $c = 10, 20, \ldots, 90$ for *FE* and $c = 2, 4, 6, 8, 10$ for *FT*. No significant changes in the clustering NMI scores and accuracy scores are observed. Therefore, we do not show the performance of PCA-based features in Table 6.1and Figure 3.

Fig.4 displays classification accuracy for a binary classification problem in which there are four hundred agents coming from two groups of decision strategies. The results are averaged over 100 replications with tenfold cross-validations. Four popular classifiers (SVM, KNN, FDA and LR) are employed to evaluate the classification performance. Results suggest that the classifiers based on IRL perform better than the simple methods, such as *FT* and *FE*, particularly when the number of observed trajectories and the length of the trajectory are small. The results support our hypothesis that recovered reward functions constitute an effective and robust feature space for clustering or classifying the agents abased on observation of their decision behavior.



(a) SVM (b) KNN

(c) LR (d) FDA

Fig. 4: Classification results with respect to different classifier.

6.2 Secretary Problem

The secretary problem is a sequential decision-making problem in which the binary decision to either stop or continue a search is made on the basis of objects already seen. As suggested by the name, the problem is usually cast in the context of interviewing applicants for a secretarial position. The decision maker interviews a randomly-ordered sequence of applicants one at a time. The applicant pool is such that the interviewer can unambiguously rank each applicant in terms of quality relative to the others seen up to that point. After each interview, the decision maker chooses either to move on to the next applicant, forgoing any opportunity to hire the current applicant, or to hire the current applicant, which terminates the process. If the process goes as far as the final applicant, he or she must be hired. Thus the decision maker chooses one and only one applicant. The objective is to maximize the probability that the accepted applicant is, in fact, the best in the pool.

To test our hypotheses on BAR, an ideal experiment would involve recognizing individual human decision makers on the basis of observations of hiring decisions that they make in secretary problem simulations. Experiments with human decision making for the secretary problem are reported on in [19][18], but raw data consisting of decision maker action trajectories is not available.

Algorithm 2 Experimentation with Secretary Problem

1: Given a heuristic rule with a parameter h, k or ℓ.
2: Add random Gaussian noise to the parameter, which is written as $\hat{p}$.
3: Generate new secretary problem with N applications and let $n-th$ agent solve these problems using this heuristic rule with its own parameter $\hat{p}$. Save the observed decision trajectories into $\mathcal{O}_n$.
4: Model the secretary problem in terms of an MDP consisting of the following components:

   1. State space $\mathcal{S} = \{1, 2, \ldots, N\}$, where $s \in \mathcal{S}$ means that at time s the current applicant is a candidate.
   2. Action space $\mathcal{A}$ consisting of two actions: reject and accept.
   3. Transition probability $\mathcal{P}$, computed as follows: given the reject action, the probability of transitioning from state s_i to s_j, $p(s_j|s_i)$, is $\frac{s_i}{s_j(s_j-1)}$ if $s_j \geq s_i$, and 0 otherwise; given the accept action, the probability of transitioning from state s_i to s_j, $p(s_j|s_i)$, is 1 if $s_i = s_j$, and 0 otherwise.
   4. The discount factor γ is a selected constant.
   5. The reward function is unknown.

5: Infer the reward function by solving an IRL problem $B = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, \mathcal{O}_n)$.

However, a major conclusion of these studies is that the decisions made by the humans largely can be explained in terms of three decision strategies, each of which uses the concept of a *candidate*. An applicant is said to a candidate he or she is the best applicant seen so far. The decision strategies of interest are the:

1. *Cutoff rule (CR)* with cutoff value h, in which the agent will reject the first $h-1$ applicants and accept the next candidate;
2. *Successive non-candidate counting rule (SNCCR)* with parameter value k, in which the agent will accept the first candidate who follows k successive non-candidate applicants since the last candidate; and
3. *Candidate counting rule (CCR)* with parameter value ℓ, in which the agent selects the next candidate once ℓ candidates have been seen.

The optimal decision strategy for the secretary problem is to use CR with a parameter that can be computed using dynamic programming for any value of N, the number of secretaries. As N grows, the optimal parameter converges to N/e and yields a probability of successfully choosing the best applicant that converges to $1/e$. Thus only one of the three decision strategies enumerated above can be viewed as optimal, and that only for a single parameter value out of the continuum of possible values. Human actions are usually suboptimal and tend to look like mixtures of CR (with a non-optimal parameter), SNCCR, and CCR [19]. As a surrogate for the action trajectories of humans, we use agents that we generate action trajectories for randomly sampled secretary problems using CR, SNCCR, and CCR. For a given decision rule (CR, SNCCR, CCR), we simulate a group of agents that adopt this rule, differentiating individuals in a group by adding Gaussian noise to the rule's parameter. The details of the process are

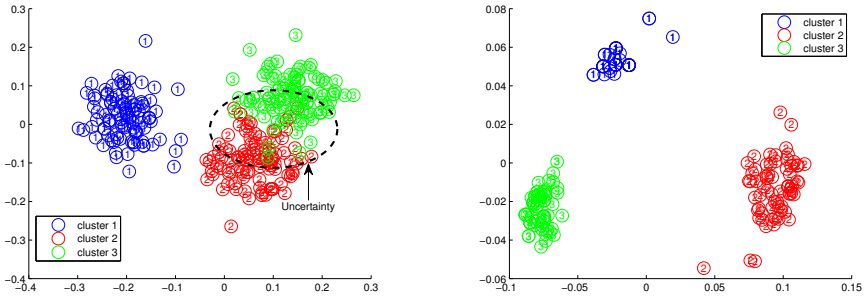


Fig. 5: Each circle denotes the feature vector for an agent, which is projected into 2D space by using PCA. The feature vectors provided by *FE* method are shown on the left. The reward vectors estimated by IRL are shown on the right.

given in Algorithm 2. We use IRL and observed actions to learn reward functions for the MDP model given in Algorithm 2. It is critical to understand that the state space for this MDP model captures nothing of the history of candidates, and as a consequence is wholly inadequate for the purposes of modeling SNCCR and CCR. In other words, for general parameters, neither SNCCR nor CCR can be expressed as a policy for the MDP in Algorithm 2. (There does exist an MDP in which all three of the decision rules can be expressed as policies, but the state space for this model is exponentially larger.) Hence, for two of the rules, the processes that we use to generate data and the processes we use to learn are distinct.

As an initial set of experiments, we generated an equal number of agents from each rule. All the heuristic rules use the same parameter value. We have compared the method using statistical feature representations obtained from the raw decision trajectories and our IRL model-based method. We employ 10 fold cross-validation to obtain the average accuracy, and it is always 100% .

Given that perfect classification performance was achieved by all algorithms, the problem of recognizing across decision rules appears to be quite easy. A more challenging problem is to recognize variations in strategy within a single decision rule. For each rule, we conducted recognition experiments in which 300 agents were simulated, 100 each for three distinct values of the rule parameter. Individuals were differentiated by adding random noise to the parameter. Here, we show the comparison of the clustering performance between the simple method called *FE* and our MDP model-based method. In Fig.5, the left figure displays an area marked "uncertainty" for the method called *FE*, while the right figure shows that the reward vectors have lower variance in the same group and higher variance between different groups.Fig.5 intuitively demonstrates that when the agents' behavior is represented in the reward space, the recognition problem becomes easier to solve.

Table 2: NMI score for Secretary Problem

H	CR		SNCCR		CCR	
	Action	BayesIRL	Action	BayesIRL	Action	BayesIRL
1	0.0557	**0.5497**	0.0551	**0.1325**	0.0229	**0.2081**
11	0.3852	**0.6893**	0.2916	**0.7190**	0.1844	**0.4974**
21	0.6017	**0.7898**	0.4305	**0.8179**	0.2806	**0.5181**
31	0.7654	**0.8483**	0.5504	**0.8641**	0.4053	**0.6171**
41	0.8356	**0.9676**	0.5682	**0.9218**	0.4524	**0.6533**
51	0.8781	**0.9739**	0.5894	**0.9423**	0.5464	**0.6507**
61	0.9102	**0.9913**	0.5984	**0.9518**	0.5492	**0.6513**
71	0.9115	**0.9915**	0.6460	**0.9639**	0.6024	**0.6512**
81	0.9532	**1.0000**	0.6541	**0.9721**	**0.6708**	0.6563
91	0.9707	**1.0000**	0.6494	**0.9864**	**0.6884**	0.6544

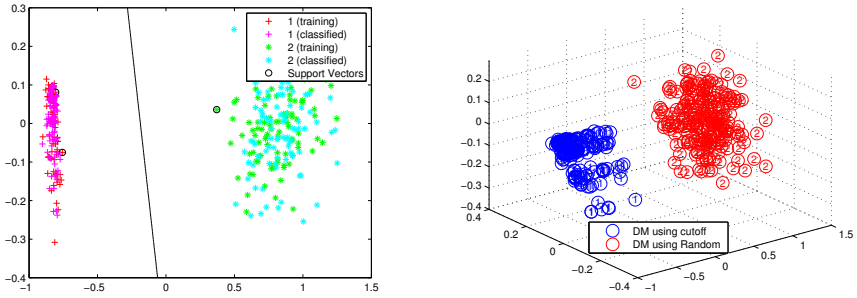


Fig. 6: Visualization of a binary classification problem for subjects using cutoff rule and random rules. The reward vectors are projected into 2(left)/3(right) dimensional subspace, which are spanned by the first 2/3 principal components.

Table 2 summarizes the NMI scores for using K-means clustering algorithm to recognize variations in strategy within one heuristic decision rule. We conduct experiments on three rules separately. The column called H in Table 2 records the number of decision trajectories that have been sampled for training. Table 2 indicates that the feature representation in reward space is almost always better than the representation with statistical features computed from the raw observation data. Moreover, the reward space can particularly better characterize the behavior when the scale of the observation data is small. Note that although none of the MDP policies can match the SNCCR and CCR rules, the reward vectors, which are recovered by IRL for the MDP model, still make the clustering problem easier to solve.

Fig.6 shows a binary classification result of using *PROJ* algorithm to learn the reward functions for the agents in Secretary problem and then categorize the agents into two groups. In this classification experiment, the users' ground

truth label is either cutoff decision rule or random strategy that makes random decisions.

7 Conclusions

We have proposed the use of IRL to solve the agent recognition problem. The observed agent is not required to be rational in the decision-making process. However, we model the agent's behavior in an MDP environment and estimate the reward function by making the MDP policy match the observed behavior. Numerical experiments on *GridWorld* and the secretary problem suggest that the advantage that IRL enjoys over simple methods is more pronounced when observations are limited and incomplete. We also note that there seems to be a positive correlation between the success of IRL algorithms in apprenticeship learning (cf. [15]) and their success in the agent recognition problem. To some degree, this relationship parallels results from [11] [7], where apprenticeship learning benefits from a learning structure that based on sophisticated methods for task decomposition or hierarchical identification of skill trees. Exploration of IRL algorithms that consider subgoals and which of the algorithmic choices can help agent recognition is an avenue of future work.

Validation of the ideas proposed here can come only through experimentation with more difficult problems. Of particular importance would be problems involving human decision makers or other real-world scenarios, such as periodic investment, gambling, or stock trading.

References

1. Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.
2. Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009.
3. Monica Babes-Vroman, Vukosi Marivate, Kaushik Subramanian, and Michael Litman. Apprenticeship learning about multiple intentions. In *the 28th International Conference on Machine learning*, WA, USA, 2011.
4. Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. Action understanding as inverse planning. *Cognition*, 113:329–349, 2009.
5. Abdeslam Boularias and Brahim Chaib-draa. Bootstrapping apprenticeship learning. In *Advances in Neural Information Processing Systems 24*. MIT press, 2010.
6. Jaedeug Choi and Kee-Eung Kim. Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing System*, pages 1989–1997, 2011.
7. Luis C. Cobo, Charles Lee Isbell Jr., and Andrea Lockerd Thomaz. Automatic task decomposition and state abstraction from demonstration. In *AAMAS*, pages 483–490, 2012.
8. Ramachandran Deepak and Amir Eyal. Bayesian inverse reinforcement learning. In *Proc. 20th International Joint Conf. on Artificial Intelligence*, 2007.

9. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, 2001.
10. Krishnamurthy Dvijotham and Emanuel Todorov. Inverse optimal control with linearly-solvable mdps. In *Proc. 27th International Conf. on Machine learning*. ACM, 2010.
11. George D. Konidaris, Scott R. Kuindersma, Roderic A. Grupen, and Andrew G. Barto. Robot learning from demonstration by constructing skill trees. *International Journal of Robotics Research*, 31(3):360–375, March 2012.
12. Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. Uncertainty in Artificial Intelligence*, 2007.
13. Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.
14. Mark Paddrik, Roy Hayes, Andrew Todd, Steve Yang, Peter Beling, and William Scherer. An agent based model of the e-mini s&p 500: Applied to flash crash analysis. In *2012 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr 2012)*, 2012.
15. Qifeng Qiao and Peter A. Beling. Inverse reinforcement learning via convex programming. In *Americon Control Conference*, 2011.
16. Miquel Ramirez and Hector Geffner. Plan recognition as planing. In *21st Int'l Joint Conf. on Artificial Intelligence*, pages 1778–1783, 2009.
17. Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *In Proceedings of the 23rd International Conference on Machine Learning*, 2006.
18. Daniel Schunk and Joachim Winter. The relationship between risk attitudes and heuristics in search tasks: A laboratory experiment. *Journal of Economic Behavior and Organization*, 71:347–360, 2009.
19. Darryl A. Seale. Sequential decision making with relative ranks: An experimental investigation of the 'secretary problem'. *Organizational Behavior and Human Decision Process*, 69:221–236, March 1997.
20. Alexander Strehl and Joydeep Ghosh. Cluster ensembles? a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.
21. Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems*, pages 1449–1456. MIT Press, 2008.
22. Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In *In Pervasive*, pages 158–175, 2004.
23. Linli Xu, James Neufeld, Bryce Larson, and Dale Schuurmans. Maximum margin clustering. In *Advanced Neural Information Process Systems*, pages 1537–1544, 2005.
24. Steve Yang, Mark Paddrik, Roy Hayes, Andrew Todd, Andrei Kirilenko, Peter Beling, and William Scherer. Behavior based learning in identifying high frequency trading strategies. In *2012 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr 2012)*, 2012.