**Apprenticeship Learning via Inverse Reinforcement Learning**
Pieter Abbeel and Andrew Y. Ng [ICML 2004]

CS 6784
March 9, 2010
(Presented by Dane & Vasu)

---

**Markov Decision Processes**
- Used for modeling sequential decision problems.
- Set of states $S$
- Set of available actions $A$.
- Transition probabilities $P_{s,a}$
  - Give probabilities for arriving in a new state after performing action $a$ while in state $s$.
- Reward functions $R(s)$
  - The ‘value’ of being in state $s$. Assume to be bounded in the absolute value by 1.

---

**Example**

**Gridworld**
- States given by grid cells
  - Additionally, specified start and end states
- At each cell, action is given by direction of movement
- Transition follows the specified action with 80% probability, else move to an adjacent cell randomly
- A transition to a given cell is accompanied by an immediate reward
- A policy maps each state to an action

---

**Policies**
- $\pi$ gives a function from states to distributions over actions.
- The value of a policy is given by:
  \[
  E_{w\sim n}[\gamma^t(s_t) - E_{s\sim \pi}^\infty \gamma^t R(s_t)] = w \cdot E_{s\sim \pi}^\infty \gamma^t \cdot \phi(s_t) \cdot \pi
  \]
- $D$ gives the distribution of starting states
- $\gamma$ is a discount factor – earlier rewards are given more weight.

---

**Computing Optimal Policies**
- Reinforcement Learning
  - For instance, Q-learning
- $Q: S \times A \to \mathbb{R}$ is a function that gives the ‘quality’ of an action from a certain state
- The agent uses $Q$ to explore the state space, and updates the function at each transition based on the experienced reward
- We know the reward function $R(s)$, but not the transition probabilities.

---

**The Problem**
- It is often difficult to specify the reward function, even if you are capable of making good decisions.
- E.g. You might be a perfectly good driver, but describing a reward function for good driving isn't so obvious.
- The solution: Apprenticeship learning.
  - Observe expert behavior, and assuming their actions to be optimal, derive the reward function.
Assumptions

- There is some feature vector over states \( \phi: S \rightarrow [0,1]^k \)
- The unknown reward function \( R(s) \) can be given by \( w^T \phi(s) \) for some \( w \in \mathbb{R}^k, \|w\|_1 \leq 1 \)

The Expert

- We have access to some expert policy \( \pi_E \)
- More accurately, we have examples of state sequences generated by said policy.
- We are also able to estimate the feature expectations \( \mu_E \)
  - Given a set of \( m \) state sequences \( s^1, s^2, \ldots, s^m \)
  - Calculate an estimate:
    \[
    \hat{\mu}_E = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s^i_t)
    \]

Algorithm (max-margin)

- Given an MDP, a feature mapping \( \phi \) and the expert’s feature expectations \( \mu_E \), find a policy whose performance is close to that of the expert’s, on the unknown reward function \( R^* = w^T \phi \).
- To accomplish this, we find a policy that induces feature expectations close to the expert policy.

Algorithm Termination

- Algorithm terminates with \( t \leq \varepsilon \). For any \( w \) (and in particular the expert’s \( w_E \)) there is at least one \( w \) whose performance under \( R^* \) is at least as good as the expert’s performance minus \( \varepsilon \).
- Ask the agent designer to manually test/examine the policies found by the algorithm, and pick one with acceptable performance.
- OR, solve: \( \arg \min \sum \lambda \mu \geq \sum \lambda \mu^i + \varepsilon \) and check if all \( w \) are within \( \varepsilon \) of \( w_E \).
- Note: the algorithm does not necessarily recover the underlying reward function correctly – it only (approximately) matches the feature expectations.

Compare with Structural SVM

\[
\arg \min \frac{1}{2} w^T \Sigma w + C \sum \xi \quad \forall \forall i j, w^T \phi(x_i, y_i) \geq w^T \phi(x_j, \hat{y}_j) + 1 - \frac{\xi}{\Delta(y_i, \hat{y}_j)}
\]

SVM Training examples:
\( \{(x_i, y_i), \ldots, (x_n, y_n)\} \)

IRL (iteration)

Constraints:
\( \forall \forall i j, w^T \phi(x_i, y_i) \geq w^T \phi(x_j, \hat{y}_j) + 1 - \frac{\xi}{\Delta(y_i, \hat{y}_j)} \)

\[
\min \sum \mu \geq \sum w^T \mu + \varepsilon
\]
**Projection Method**

- Instead of keeping all prior feature expectations, just look at the most recent expectations, and an orthogonal projection of the expert expectations
  - Set $\hat{\mu}^{(t-1)} = \hat{\mu}^{(t-2)} + \frac{(\hat{\mu}^{(t-1)} - \hat{\mu}^{(t-2)})^T (\hat{\mu}^{(t-1)} - \hat{\mu}^{(t-2)})}{(\hat{\mu}^{(t-1)} - \hat{\mu}^{(t-2)})^T (\mu_E - \hat{\mu}^{(t-2)})} (\mu_E - \hat{\mu}^{(t-2)})$
  - Set $w^{(t)} = \mu_E - \hat{\mu}^{(t-1)}$

- We no longer have to solve a QP, so no SVMs here.
  - In case you’re just not an SVM kind of guy/gal.

**Experimental Results**

- Projection method converges slightly faster than max-margin
- In general, IRL performs better than more naïve alternatives even with a small amount of training data

**Conclusions**

- Assumed access to demonstrations by an expert maximizing a reward function linear in known features
  - (How reasonable is this? Quite, for rich feature spaces.)
- Algorithm based on inverse reinforcement learning
  - Terminates in a small number of iterations
  - Guarantees policy with performance comparable to or better than expert on the expert’s unknown reward function (but without recovering the reward function!)
- Open problems:
  - Non-linear reward functions
  - Automatic feature construction and feature selection

**More Results**

- Gridworld:
  - Projection method converges slightly faster than max-margin
  - In general, IRL performs better than more naïve alternatives even with a small amount of training data

**Driving task:**

- (Please turn off your cell phones during the movie)