Optimization for CVaR IRL

March 30, 2020

1 Notation

We use the following notation:

- States $\mathcal{S} = \{1, \ldots, S\}$
- Actions $\mathcal{A} = \{1, \ldots, A\}$
- Δ^k probability simplex in k-dimensions.
- Initial distribution: $p_0 \in \Delta^S$
- Rewards: $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Policy: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ **[Double check other notation to make it work for stochastic policies]**
- Rewards for policy π: $r_\pi(s) = \mathbb{E}_{a \sim \pi(s)} r(s, a)$
- Expert's policy $\pi_E : \mathcal{S} \to \mathcal{A}$
- Transition probability for a policy π: P_π, treated as a matrix, defined as:

$$P_\pi(s, s') = \mathbb{E}_{a \sim \pi(s)} P(s, a, s') = \sum_a \pi(a|s) P(s, a, s')$$

- Occupancy frequency for a policy: $u_\pi = (I - \gamma P_\pi^\mathsf{T})^{-1} p_0$. This can be derived by noting that the Markov Chain stationary distribution over states u_π satisfies the following equation at equilibrium: $u_\pi = p_0 + \gamma P_\pi^T u_\pi$. Solving for u_π yields the above equation.
- If we want the state-action occupancies we can solve for these similarly as $u_\pi = p_0 + \gamma P_\pi^T u_\pi$ and $u(s, a) = u_\pi \cdot \pi(a|s)$.
- Linear feature matrix with rows as states and columns as features: $\Phi \in \mathbb{R}^{S \cdot A \times k}$, where k is the number of features
- Assume that the rewards are approximated as $r = \Phi w$ for some $w \in \mathbb{R}^k$
- Feature counts: $\mu_\pi = \Phi^\mathsf{T} u_\pi$. Here $\mu_\pi \in \mathbb{R}^k$
- Value function $v_\pi = (I - \gamma P_\pi)^{-1} r_\pi$. This can be derived via the bellman equation for values: $v_\pi = r_\pi + \gamma P_\pi v_\pi$ and solving for v_π.
- Return for a specific policy and rewards: $\rho(\pi, r) = p_0^\mathsf{T} v = u_\pi^\mathsf{T} r_\pi$

Now, assume that R is the random variable representing the reward. The posterior can be derived using Bayesian IRL. Let $R_1, R_2, R_3, \ldots$ be samples from the posterior distribution.

2 Value at Risk

When dealing with risk we will assume that lower values are worse (riskier), thus we will want to maximize the Value at Risk or Conditional Value at Risk since tails to the left are bad. We will define α-Value at risk as the $(1-\alpha)$ quantile worst-case outcome. Thus, the α-VaR is such that

$$\alpha\text{-VaR}[X] = \sup\{x : Pr(X \geq x) \geq \alpha\} \tag{1}$$

Given policy π, our AAAI'19 paper [**?**] focused on finding a high-confidence lower bound on:

$$\text{V@R}\left[\rho(\pi, R) - \rho(\pi^*_R, R)\right],$$

This is not quite the same as finding a lower bound on

$$\text{V@R}\left[\rho(\pi, R) - \rho(\pi_E, R)\right],$$

where R is the random variable distributed according to the posterior from the Bayesian IRL. This second form is interesting because we may be able to do better than the expert. We don't want to match the risk of the expert, rather we want to minimize our risk with the expert as the baseline. **Can we do the same thing as our AAAI paper by reusing the BIRL π^* for each policy? I think so. We just adjust the objective so instead of u_E we use u_{π^*}, right?** We will denote the posterior distribution p; and generally assume that p is a probability distribution over a finite number of samples from the posterior distribution, e.g. a uniform distribution over n samples from MCMC. In [**?**] we derive finite-sample bounds in terms of the number of samples from the posterior distribution. Unfortunately, V@R is not convex and thus is hard to optimize.

3 Average Value at Risk

Average Value at Risk (AV@R) is a convex coherent risk measure. It is also commonly referred to as Conditional Value at Risk, expected tail risk, or expected shortfall. It is convex, and is a lower bound on V@R. It can be also preferable because it does not ignore how heavy the tail of the distribution is. V@R only considers the quantile, but ignores any outcome that may be worse than that.

The intuitive (but not entirely correct) definition of AV@R (the same as CVaR) is:

$$\text{AV@R}_\alpha[X] = \mathbb{E}\left[X \;\middle|\; X \leq \text{V@R}_\alpha[X]\right] .$$

This only works for atomless distributions such that no ω has a positive probability (i.e. most continuous distributions). However, we are interested in maximizing AV@R given a finite number of samples from the posterior distribution $P(R|D)$. The correct convex definition of AV@R that works for any distribution (discrete or continuous) is:

$$\max_\sigma \left(\sigma - \frac{1}{1-\alpha} p^\intercal [\sigma \cdot \mathbf{1} - x]_+\right) ,$$

where $[\cdot]_+$ is an element-wise non-negative part of the vector x: $[x]_+ = \max\{x, \mathbf{0}\}$.

A popular way to analyze and use coherent risk measures is to look at their robust representation:

$$\text{AV@R}_\alpha[X] = \min_{q \in \mathcal{Q}} \mathbb{E}_q[X] ,$$

which is the expectation with respect to a worst-case distortion of the nominal probability distribution p. For AV@R the set $\mathcal{Q}$ is defined as:

$$\mathcal{Q} = \left\{q \in \Delta^n \;\middle|\; q \leq \frac{1}{1-\alpha} p\right\} ,$$

where Δ^n is the probability simplex over $\mathbb{R}^n$ and $p \in \Delta^n$. You can think about this as follows: when alpha is 0 then $q \leq p$ and since $\sum q = \sum p = 1$ and $q \geq 0$, we must have $q = p$. Thus CVaR is just $\min_{q=p} \mathbb{E}_q[X] = \mathbb{E}_p[X]$. Just the nominal expectation since VaR with alpha = 0 is the biggest possible value of X so CVaR is just the expectation of X. As you make alpha larger it gives q more wiggle room. As alpha goes to 1 you make 1/(1-alpha) go to infinity so q can choose to put all the probability mass on very unlikely outcomes. Because it is trying to minimize expectation under this warped distribution it will put as much weight as it can on these worst outcomes and can focus more on unlikely outcomes as alpha goes to 1.

Lets say that the goal is to find the best policy and we want to minimize AV@R of the "robust baseline regret" [**?**, **?**, **?**]. We called this a robust baseline regret but this is just the standard objective in IRL:

$$\max_{\pi} \text{AV@R}_\alpha \left[\rho(\pi, R) - \rho(\pi_E, R)\right] \tag{2}$$

We can formulate (2) as a linear program following the next steps. Recall the one to one correspondence between randomized policies $\pi : \mathcal{S} \to \Delta^A$ (where A is the number of actions) and the occupancy frequencies u [**?**]. That means that $\max_\pi \rho(\pi, r)$ corresponds to the following linear program [**?**]:

$$\max_{u:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \left\{ r^\intercal u \;\mid\; \sum_{a\in\mathcal{A}} (\mathbf{I} - \gamma \cdot P_a^\intercal) u_a = p_0, u \geq \mathbf{0} \right\} .$$

I'm currently solving this via SciPy's built in LP solver as follows:

$$\begin{aligned}
\min_{u:\mathcal{S}\times\mathcal{A}\to\mathbb{R}} \quad & -r^\intercal u && (3)\\
\text{s.t.} \quad & \left[(I - \gamma P_{a_1}^\intercal), \ldots, (I - \gamma P_{a_m}^\intercal)\right] \begin{bmatrix} u_{a_1} \\ \vdots \\ u_{a_n} \end{bmatrix} = p_0 && (4)\\
& u \geq \mathbf{0} && (5)
\end{aligned}$$

where $u_a = [u_{(s_1,a)}, u_{(s_2,a)}, \ldots u_{(s_n,a)}]^\intercal$.

Using the same approach as the linear program above, we can formulate (2) as a linear program following these steps. Let R be a matrix $(S \cdot A) \times n$ of all sampled posterior rewards R. That is, each column of R represents one sample of the vector over rewards for each state and action pair. (2) becomes:

$$\max_{u,\sigma} \left\{ \sigma - \frac{1}{1-\alpha} p^\intercal \left[\sigma \cdot \mathbf{1} - R^\intercal u + R^\intercal u_E\right]_+ \;\mid\; \sum_{a\in\mathcal{A}} (\mathbf{I} - \gamma \cdot P_a^\intercal) u_a = p_0, u \geq \mathbf{0} \right\} . \tag{6}$$

This is a linear program which works in the tabular case. This can be written more explicitly as

$$\begin{aligned}
\max_{\sigma,u} \quad & \sigma - \frac{1}{1-\alpha} p^\intercal z && (7)\\
\text{s.t.} \quad & z \geq \sigma \mathbf{1} - R^\intercal (u - u_E) && (8)\\
& \left[(I - \gamma P_{a_1}^\intercal), \ldots, (I - \gamma P_{a_m}^\intercal)\right] \begin{bmatrix} u_{a_1} \\ \vdots \\ u_{a_n} \end{bmatrix} = p_0 && (9)\\
& u \geq \mathbf{0} && (10)\\
& z \geq \mathbf{0} && (11)
\end{aligned}$$

I'm using SciPy to solve this LP in the following form:

$$\min_{\sigma,u} \quad -\sigma + \frac{1}{1-\alpha} p^\mathsf{T} z \tag{12}$$

$$\text{s.t.} \quad -R^\mathsf{T} u + \sigma \mathbf{1} - z \leq -R^\mathsf{T} u_E \tag{13}$$

$$\left[(I - \gamma P_{a_1}^\mathsf{T}), \ldots, (I - \gamma P_{a_m}^\mathsf{T})\right] \begin{bmatrix} u_{a_1} \\ \vdots \\ u_{a_n} \end{bmatrix} = p_0 \tag{14}$$

$$u \geq \mathbf{0} \tag{15}$$

$$z \geq \mathbf{0} \tag{16}$$

The optimal risk-averse IRL policy $\pi^\star$ can be constructed from an optimal $u^\star$ solution to (6) as:

$$\pi^\star(s,a) = \frac{u^\star(s,a)}{\sum_{a' \in \mathcal{A}} u^\star(s,a')} \ . \tag{17}$$

The linear program can be easily extended to linear approximation by assuming that $r = \Phi w$ is which case the return becomes $u^\mathsf{T} r = u^\mathsf{T} \Phi w = \mu^\mathsf{T} w$.

This formulation can be extended to non-linear approximation using a similar approach as GAIL, I think. The key is the dual representation of AV@R naturally maps to an adversarial algorithm.

3.1 Recovering Rewards

The question is how to recover the reward vector the would generate the AVaR return. One possibility is to use the dual representation of AVaR. Let $\pi^\star$ be the optimal solution to (6) as in (17). Let:

$$\mathcal{Q} = \left\{ q \in \Delta^n \ \mid \ q \leq \frac{1}{1-\alpha} p \right\} \ .$$

Then to get the reward, let $\pi^\star$ be the optimal solution to (6). Then one needs to solve:

$$\text{AV@R}_\alpha \left[\rho(\pi^\star, R) - \rho(\pi_E, R)\right] = \min_{q \in \mathcal{Q}} \mathbb{E}_{R \sim q} \left[\rho(\pi^\star, R) - \rho(\pi_E, R)\right] \ . \tag{18}$$

It seems like this should be minimize, right? I'm changing it for now. Another way to write the expectation would be as follows:

$$\mathbb{E}_{R \sim q} \left[\rho(\pi^\star, R) - \rho(\pi_E, R)\right] = \sum_{i=1}^{n} q_i \left(\rho(\pi^\star, r_i) - \rho(\pi_E, r_i)\right) \ ,$$

where r_i is the i-th posterior sample.

Let $q^\star$ be the optimal solution to the linear program in (18). Using the fact that ρ is linear in r, we get:

$$\mathbb{E}_{R \sim q^\star} \left[\rho(\pi^\star, R) - \rho(\pi_E, R)\right] = \rho\left(\pi^\star, \mathbb{E}_{R \sim q^\star}[R]\right) - \rho\left(\pi_E, \mathbb{E}_{R \sim q^\star}[R]\right) \ .$$

That means that $\mathbb{E}_{R \sim q^\star}[R] = \sum_i q_i R_i$ is the worst-case reward.

So does this mean that π^*_{cvar} is optimal wrt $\mathbb{E}_{R \sim q^\star}[R]$ or that it is optimal wrt maximizing the CVaR robust baseline regret wrt this reward? It must be the latter, right? Actually, since $R_{cvar}^\mathsf{T} u_E$ is constant shouldn't maximizing one be equivalant to maximizing the other? Why doesn't it work out in code? The learned policy doesn't match what you would get by just maximizing policy for R_{cvar}.

One possible issue that I've noticed is that there might be multiple optimal solutions for R_{cvar}. This is because if there are two different reward functions that lead to the same policy loss then q^* can blend them any way. However, this seems to be able to lead to different optimal policies. It would be nice to have a reward function that when optimized would recover the solution to the CVaR policy, but maybe that is not possible?

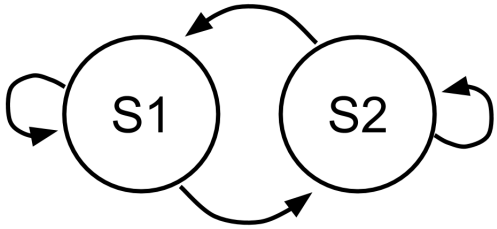

Figure 1:

3.2 Dual

So, after talking with Marek it turns out that to get the reward that, when optimized, results in the CVaR optimal policy, we need to solve the dual problem:

$$\min_{v,q} \quad p_0^T v - u_E^T R q \tag{19}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} q = 1 \tag{20}$$

$$q \leq \frac{1}{1-\alpha} p \tag{21}$$

$$\begin{bmatrix} (I - \gamma P_{a_1}) \\ \vdots \\ (I - \gamma P_{a_m}) \end{bmatrix} v \geq Rq \tag{22}$$

$$q \geq \mathbf{0} \tag{23}$$

$$v \in \mathbb{R}^{|S|} \tag{24}$$

4 Example

Consider a determinisitc two state chain MDP with actions left and right as shown in Figure 1 with two actions: left and right.

Let's assume that $\gamma = 0.5$ and that we saw a demonstration that started in state S2 and took action left to travel to S1. Given this information let's assume that our posterior distribution has three possibilities with non-zero probability: R_1, R_2, and R_3, where

$$R_1 = (0, 1, 0, 0) \tag{25}$$

$$R_2 = (0.5, 0, 0.5, 0) \tag{26}$$

$$R_3 = (0.5, 0.5, 0, 0) \tag{27}$$

$$\tag{28}$$

where each reward vector is of the form $\big(r(S1, \leftarrow), r(S2, \leftarrow), r(S1, \rightarrow), r(S2, \rightarrow)\big)$. R_1 assumes going left in state 2 is the only thing that matters. R_2 assumes that being in state S1 is the only thing that matters. R_3 assumes that always taking the left action is the only thing that matters.

Let's assume for now that we have a uniform posterior over these possibilities, thus $p = (1/3, 1/3, 1/3)$ and

$$R = \begin{bmatrix} 0. & 0.5 & 0.5 \\ 1. & 0. & 0.5 \\ 0. & 0.5 & 0. \\ 0. & 0. & 0. \end{bmatrix} \tag{29}$$

We will also assume for now that $u_E = \mathbf{0}$. Thus we just want to find the robust solution that maximizes the AV@R of $\rho(\pi, R)$.

4.1 Solutions for different α

Let's try solving for the optimal cvar policy for different values of α.

α	CVaR	$P(\leftarrow \mid S1)$	$P(\rightarrow \mid S1)$	$P(\leftarrow \mid S2)$	$P(\rightarrow \mid S2)$
0	0.75	1	0	1	0
0.2	0.70	0.6	0.4	1	0
0.5	0.66	0.6	0.4	1	0
0.8	0.66	0.75	0.25	1	0
0.99	0.66	0.75	0.25	1	0

As expected the CVaR decreases as α increases. However, I was surprised that it plateaued at 2/3 so quickly.

4.2 q^* for different α

I also calculated q^* for different α values.

α	CVaR	$r(S1, \leftarrow)$	$r(S2, \leftarrow)$	$r(S1, \rightarrow)$	$r(S2, \rightarrow)$	q_1	q_2	q_3
0	0.75	1/3	1/2	1/6	0	1/3	1/3	1/3
0.2	0.70	0.35	0.44	0.21	0.0	0.29	0.42	0.29
0.5	0.66	0.42	0.25	0.33	0.0	0.17	0.67	0.17
0.8	0.66	0.25	0.5	0.25	0.0	0.5	0.5	0
0.99	0.66	0.25	0.5	0.25	0.0	0.5	0.5	0

4.3 Numerical issues

I've found that depending on the solver I use I sometimes get different answers. In particular, I ran with $\alpha - 0.2$ using either simplex or interior point and got two different results. The CVaR values are the same for both, and the CVaR maximizing policies are the same, but the values for q^* are different. I think this is because the policy returns over the posterior are $\rho(\pi^*_{cvar}, R) = [0.75, 0.625, 0.75]$ for R_1, R_2, and R_3, respectively.

When choosing q^* it will put as much mass on R_2 as possible, but then has a choice between R_1 and R_3 and it doesn't matter in terms of the robust representation. However, it does matter in terms of the actual reward function $R_{cvar} = \mathbb{E}_{R \sim q^*}[R]$.

For simplex I get $R_{cvar} = (0.29, 0.5, 0.21, 0)$, but for interior point I get $R_{cvar} = (0.35, 0.44, 0.21, 0)$. Maximizing the first leads to always going right in S1 and left in S2. Optimizing the second leads to always going left in S1 and S2. Also the expected returns of each of the policies are different from the value of the CVaR objective and also different from the value of the robust representation LP objective with π^* fixed. This seems problematic, if we can't recover the CVaR optimal policy from the CVaR reward since for $\alpha = 0.2$ the solution of the CVaR LP was $\pi^* = (0.6, 0.4, 1, 0)$.

5 Bayesian IRL

We solve for the feature expectations using the following linear program [**?**]:

$$\max_{u: \mathcal{S} \times \mathcal{A} \to \mathbb{R}} \left\{ r^\mathsf{T} u \;\mid\; \sum_{a \in \mathcal{A}} (\mathbf{I} - \gamma \cdot P_a^\mathsf{T}) u_a = p_0, u \geq \mathbf{0} \right\} .$$

For vanilla Bayesian IRL we need to get the Q-values for all states in the demonstrations. However, for optimizing CVaR and finding something robust to the best policy under each reward function we need the occupancy frequencies for each mdp in the chain.

How do we do this? We take a hypothesis reward function r and then run the above LP to get u. We then calculate the Q-values how? What's the fastest way with matrix vector notation? We can solve for values with LP, but then need to solve the dual for occupancy frequencies or can we recover them?

Can we efficiently go backwards? If we have the state value function, we can easily solve for the Q-values using an expectimax over next states. The Q-values give a greedy optimal policy. We can then solve for the feature expectations or state occupancies via a policy evaluation step. Actually, can't we just solve for the successor features via value iteration? This gives everything we need, right? Rather than using a scalar as the reward, just use a vector of state occupancies and dot with reward weights when computing max? This way seems better since we can get expected occupancies by taking initial distribution and multiplying by successor features matrix.

First TODO: Get slow version of BIRL working with running an LP for occupancy frequencies (dual). Then get policy, and policy rewards. Then we can solve for the value function as

$$v_\pi = (I - \gamma P_\pi)^{-1} r_\pi \tag{30}$$

and for the Q-values as

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s,a,s') v_\pi(s') \tag{31}$$

Alternatively, we can vectorize and solve as

$$q_\pi = r + \gamma \begin{bmatrix} P_{a_1} \\ \vdots \\ P_{a_n} \end{bmatrix} v_\pi = r + \gamma \begin{bmatrix} P_{a_1} \\ \vdots \\ P_{a_n} \end{bmatrix} (I - \gamma P_\pi)^{-1} r_\pi \tag{32}$$

Optimization TODO: Test this. Write a value iteration method that computes successor representations [Dayan, DeepMind] and check if I get the same thing as the LP.

Can probably use sparse matrices for transitions and make things faster!

Might want to transition to C++ if things are working nicely but too slow...

6 Questions

What to do about reward scaling? Can we add a L1 constraint to the LP? Taking a convex combo of R_i could lead to a reward with $\|R\| < 1$. I'm not sure if that is a problem or not.

What if we don't have good expert occupancies? Can we do what we did in the AAAI'18 paper and have things relative to optimal policies for each posterior sample? Yes. this should work. What if the demonstrator is risky? Can we do better than the demonstrator but still stay safe? **I think this would be very compelling since other work like RS-GAIL just tries to match the safety of the demonstrator.**

Can we combine this with T-REX? We can get fast MCMC done with TREX maybe even add stein variational stuff. Then we have posterior. Can we make the demonstrations the baseline? Could take the empirical feature counts of the demos and try to maximize CVaR with respect to demonstrator.

7 TODO

Extend this to linear feature approximation.

Extend to non-linear feature approximation via GAIL-like approach.