강화학습을 적용하여 Secretary Problem의 최적 정책 검증

18101204 김윤성

환경 버전

unity editor : 2021.3.19.f1
unity build : windows
unity ml-agent package : 2.0.1 (release-20)

python mlagents : 0.28.0
“pip install mlagents==0.28.0”
python mlagents_envs : 0.28.0
“pip install mlagents_envs”

numpy : 1.21.5
“pip install numpy”

- unity build가 windows 환경 이므로, windows 환경에서만 실행 가능합니다.

용량 문제로 인해 전체 소스코드는 아래 링크에서 확인 가능합니다.
소스코드 : https://github.com/oeccsy/SecretaryProblem

문제 소개 - Secretary Problem

문제의 이름이 “Secretary Problem (비서 문제)” 인 이유는 비서를 고용하는 과정으로 설명할 수 있기 때문입니다.

최고의 비서를 고용하고자 하는 면접관을 상상해보겠습니다.
n명의 지원자가 있습니다. n명의 지원자는 1등부터 n등 까지 순위를 매길 수 있습니다. 면접관은 지원자 중 한명씩 무작위로 인터뷰를 진행합니다.

이때 이 사람을 고용할지 말지는 인터뷰 직후에 이루어져야 합니다.

일단 지원자를 탈락시키면 결정을 번복할 수 없습니다

면접관은 지금까지 인터뷰 한 모든 지원자들과, 현재 면접을 보고 있는 지원자에 대해서는 순위를 매길 수 있습니다.
하지만 당연하게도 아직 인터뷰하지 않은 지원자의 순위까지는 매길 수 없습니다.

이때 최고의 지원자를 선택하기 위해서는 어떻게 해야 하는가? 에 대한 전략이 Secretary Problem입니다.

접근법과 이에 대한 최적해

이 문제에 대한 접근 방법은 어떤 수 k를 정하여 k번째 지원자까지는 "무조건" 거절하는 접근법입니다.
k번째 까지 중 가장 순위가 좋았던 지원자를 기준으로 그 이후 지원자들 중 "기준보다 높은 순위"의 지원자가 나오면 바로 수락하는 방법을 통해 접근할 수 있습니다.
그리고 이 경우 가장 최적의 해가 되는 k값은 $k=\frac{n-1}{e}$에 가장 가까운 자연수입니다.

k값 증명 과정

앞서 언급한 k값의 최적해를 구하는 과정은 다음과 같이 나타낼 수 있습니다.

최고의 지원자의 인터뷰 순서를 A라 하고 최고의 지원자를 고르게 되는 사건을 B라고 하자.

최고의 지원자가 i번째에 있다고 하면 k가 i보다 크거나 같으면 $P_k(B|A=i)=0$이다.
k가 i보다 작으면, $(i-1)$번째까지의 지원자 중 최고의 지원자가 k번째까지 중에 있으면 되므로 $P_k(B|A=i)=\frac{k}{i-1}$이다.

따라서 $P_k(B|A=i)=\begin{cases}0 & (k\geq i)\\ \frac{k}{i-1} & (k<i)\end{cases}$ 로 나타낼 수 있다.

이때 $A=i\,(1\leq i\leq n)$는 동시에 일어날 수 없다. 즉, 서로 독립인 관계에 있고 이때

$$\begin{aligned}P_k(B)&=P_k(B\cap(A=1))+\cdots+P_k(B\cap(A=n))\\&=P_k(B|A=1)P_k(A=1)+\cdots+P_k(B|A=n)P_k(A=n)\\&=\sum_{i=1}^{n}P_k(B|A=i)P_k(A=i)\end{aligned}$$

따라서 k에 따른 최고의 지원자를 선택할 확률을 위와 같이 표현할 수 있다.
이때 최고의 지원자의 인터뷰 순서를 알 수 없으므로 각 순서에 최고의 지원자가 있을 확률은 $\frac{1}{n}$이다.

따라서 $\sum_{i=1}^{n}P_k(B|A=i)P_k(A=i)=\sum_{i=1}^{k}0\times\frac{1}{n}+\sum_{i=k+1}^{n}\frac{k}{i-1}\times\frac{1}{n}$이므로 최고의 지원자를 선택할 확률은 $P_k(B)=\frac{k}{n}\sum_{i=k+1}^{n}\frac{1}{i-1}$ 이다.

이때, n이 충분히 클 때 $P_k(B)=\frac{k}{n}\sum_{i=k+1}^{n}\frac{1}{i-1}$는 $\frac{k}{n}\int_{k}^{n-1}\frac{1}{x}dx$에 근사한다고 볼 수 있다.

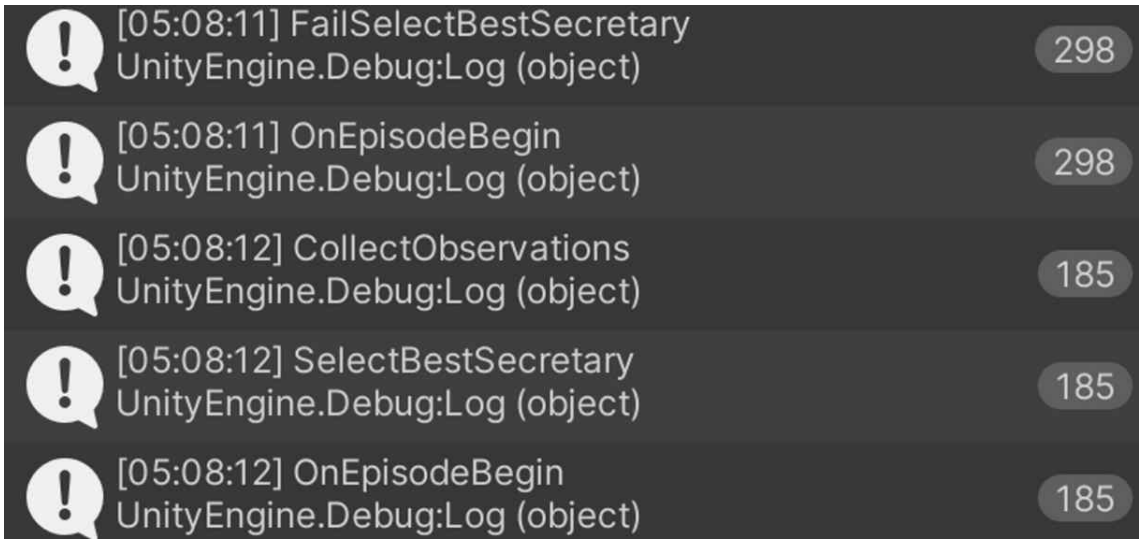
이때 $\frac{k}{n}\int_{k}^{n-1}\frac{1}{x}dx=\frac{k}{n}(\ln(n-1)-\ln k)$ 이고 $P_k(B)$가 최대가 되는 k값을 찾기 위해 $P_k(B)$를 미분하면 $\frac{d}{dk}P_k(B)=\frac{1}{n}(\ln(n-1)-\ln k)-\frac{1}{n}=\frac{1}{n}(\ln(\frac{n-1}{k})-1)$ (n,k는 자연수)

이고 $k<\frac{n-1}{e}$ 일 때, $\frac{d}{dk}P_k(B)>0$이고, $k>\frac{n-1}{e}$일 때, $\frac{d}{dk}P_k(B)<0$ 이므로 $P_k(B)$는 $k=\frac{n-1}{e}$일 때 최댓값을 갖는다.

이때 $k=\frac{n-1}{e}$은 자연수가 아니므로 $k=\frac{n-1}{e}$에 가장 가까운 자연수에서 최댓값을 가진다.

그리고 이때, 이를 이용하여 최적의 선택을 할 확률은 약 36.8%가 된다.

해당 결과로부터 Unity에서 제작한 환경에서 100명의 면접자 중 37번째까지 무조건 거절하면서 가장 뛰어난 비서를 고용하려고 했을 때 실제로 가장 뛰어난 비서를 고용하게된 확률을 확인한 결과입니다.



- (Select Best Secretary) / (Fail Select Best Secretary + Select Best Secretary) * 100 = 185 / (298+185) * 100 = 38.302277. . .

강화학습을 적용하여 최적해 확인하기

제가 제시한 정책은 k명의 까지는 무조건 고용을 거절하는 정책입니다.
여기서 k값의 최적해를 찾기 위해, [1,n]의 n개의 값 중 하나로 k값을 지정하여 보상을 확인하는 과정으로 모델링 할 수 있습니다.

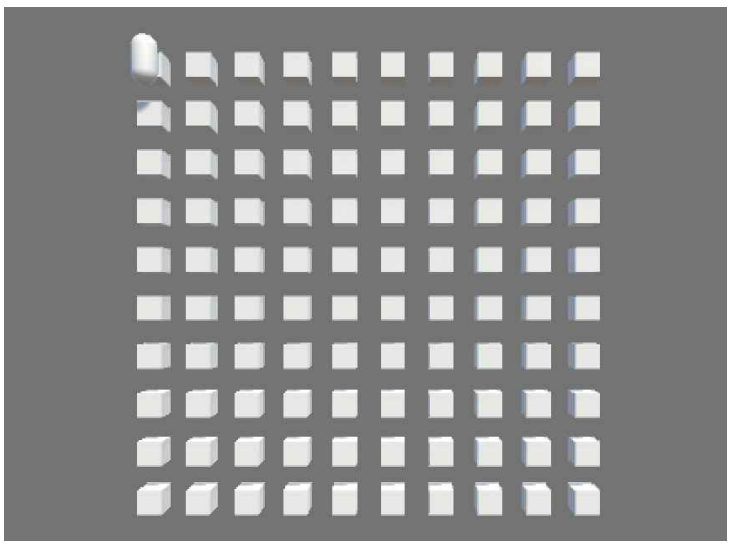
그리고 이 전략을 사용하는 경우 Multi-Armed-Bandit Problem과 그 문제가 유사합니다. 특별한 state의 정의 없이 1~n 까지의 슬롯머신 중 어떤 슬롯머신을 선택하는게 가장 현명한지 학습하는 과정으로 대응할 수 있기 때문입니다.

따라서 Action이 n가지, State는 1가지(terminate state를 제외했을 때)인 문제로
Unity 환경을 제작하였습니다.
여기서 n의 값은 지원자 수와 같으며 그 값은 100으로 지정하였습니다.

Agent는 다음 100가지 Action중 하나를 결정하여 진행합니다.
- 1~n 까지 값 중 k 값을 결정하여 k번째 지원자까지 무조건 고용을 거절하며 이후 지원자 중 지금까지의 지원자 중 최고로 판단될 때 해당 지원자를 고용한다.

해당 Action 이후 terminate state로 상태가 전이되며, Reward를 확인하게 됩니다.

Environment는 직접 제작하여 Custom하는 과정이 필요하기 때문에
Unity의 ML-Agent환경에서 환경을 제작하고, python api를 통해 python script로 해당 환경을 이용하는 방법을 진행하였습니다.



- 마련한 환경의 이미지

해당 환경에서 학습을 진행하기 위한 소스코드는
"SecretaryProblem_UnityEnv\Assets\Scripts\Secretary_Problem_Case_MAB_Agent.cs" 에서 확인할 수 있습니다.

해당 환경에 MAB 방법으로 최적해를 찾았습니다.
여기서 action을 결정하는 방식은 총 3가지로 나눠서 소스코드를 작성하여 문제를 해결하였습니다.

[Epsilon-Greedy 방식]
Explore와 Exploit 사이의 Trade-Off로 인하여 항상 Exploit하지 않고, 일정 확률로 Explore 하여 0.5의 확률로 선택 가능한 모든 Action중 무작위로 하나를 선택하는 방식을 적용하였습니다.

그 결과 k값 38이라는 최적해를 얻으면서 성공적으로 적용 결과를 확인할 수 있었습니다.

```
total reward for ep 19974 is action : 25 and reward : -1.0
total reward for ep 19975 is action : 38 and reward : -1.0
total reward for ep 19995 is action : 38 and reward : -1.0
total reward for ep 19996 is action : 71 and reward : 1.0
total reward for ep 19997 is action : 80 and reward : -1.0
total reward for ep 19998 is action : 90 and reward : -1.0
total reward for ep 19999 is action : 38 and reward : -1.0
The optimal arm is 38
```

소스코드는
“SecretaryProblem_PythonScripts\secretary_problem_case_mab_epsilon_greedy.py” 에서 확인할 수 있습니다.

[Softmax 방식]
Softmax 방식을 적용하여 가치가 높은 행동들에 대해 더 자주 탐색을 진행하는 과정을 확인하였습니다.

결과값 38이라는 최적해를 얻으면서 성공적으로 적용 결과를 확인할 수 있었습니다.

```
total reward for ep 19974 is action : 38 and reward : -1.0
total reward for ep 19975 is action : 38 and reward : -1.0
total reward for ep 19976 is action : 38 and reward : 1.0
total reward for ep 19995 is action : 38 and reward : -1.0
total reward for ep 19996 is action : 38 and reward : 1.0
total reward for ep 19997 is action : 38 and reward : -1.0
total reward for ep 19998 is action : 38 and reward : -1.0
total reward for ep 19999 is action : 38 and reward : -1.0
The optimal arm is 38
```

소스코드는
“SecretaryProblem_PythonScripts\secretary_problem_case_mab_softmax.py”
에서 확인할 수 있습니다.

[Upper Confidence Bound 방식]

불확실성을 고려하여 신뢰구간에 따라 action을 선택하도록 했습니다. UCB를 적용한 방식은 action의 가치를 잘 모르는 초기 학습 단계에서 특히 유용 했습니다.

```
total reward for ep 19992 is action : 52 and reward : -1.0
total reward for ep 19993 is action : 83 and reward : 1.0
total reward for ep 19994 is action : 83 and reward : -1.0
total reward for ep 19995 is action : 83 and reward : -1.0
total reward for ep 19996 is action : 83 and reward : 1.0
total reward for ep 19997 is action : 83 and reward : -1.0
total reward for ep 19998 is action : 83 and reward : -1.0
total reward for ep 19999 is action : 83 and reward : -1.0
The optimal arm is 37
```

결과값 37이라는 최적해를 얻으면서 성공적으로 적용 결과를 확인할 수 있었습니다.

소스코드는
“SecretaryProblem_PythonScripts\secretary_problem_case_mab_ucb.py” 에서 확인할 수 있습니다.

문제 모델링

이렇게 제가 제시한 전략으로부터 강화학습을 적용하여 최적의 선택을 하는 과정을 확인하였지만, 이미 수학적으로 증명한 전략을 적용하였기 때문에 실제 면접관이 마주하는 state와 action과는 거리가 있었습니다. 따라서 이번에는 해당 전략을 적용하지 않고, state와 action만이 주어진 상황에서 강화학습을 적용해 보았습니다. 먼저 해당 과정을 진행하기 위해서는 Environment가 필요합니다.

마찬가지로 Environment는 직접 제작하여 Custom하는 과정이 필요하기 때문에 Unity의 ML-Agent환경에서 환경을 제작하고, python api를 통해 python script로 해당 환경을 이용하는 방법을 진행하였습니다.

그리고 해당 환경을 제작하기 위해 문제를 모델링 하는 과정을 진행했습니다.

우리는 다음과 같이 State를 정의하여 한명씩 순서대로 면접을 진행하는 환경을 마련할 수 있습니다.

1. 지원자 총원(n)
2. 현재 면접을 보는 지원자를 포함하여 지금까지 면접을 진행한 인원 [1,n]
3. 지금까지의 면접을 반영한 현재 지원자의 순위 [1,n]

이때 지원자 총원은 한번의 비서 고용에서 하나의 값으로 고정되기 때문에 실질적인 state의 종류는 n x n = n ^ 2 가지 입니다.

그리고 면접관이 한 명의 면접자와 면접을 진행하는 것을 1 step이라고 하면
1번의 step에서 선택할 수 있는 선택지는 다음과 같은 2가지로 정의할 수 있습니다.

1. Select : 해당 면접자를 고용하기로 결정합니다.
2. Pass : 해당 면접자를 고용하지 않기로 결정합니다.

이때 Select를 선택 하는 경우 이후의 면접은 진행하지 않으므로 terminate state로 상태가 전이됩니다. 지원자중 가장 뛰어난 비서를 고용하게 된 경우 Reward : +1, 아닌 경우 Reward : -1이 부여됩니다.

Pass를 선택하는 경우 다음 면접자의 면접을 진행합니다. Pass 한 면접자를 고용하지 않은 것은 번복할 수 없습니다. 만약 마지막 지원자까지 Pass 하는 경우 가장 뛰어난 비서를 고용하는데 실패했기 때문에 Reward :-1 이 부여됩니다.

해당 환경에 대한 소스코드는
“SecretaryProblem_UnityEnv\Assets\Scripts\Secretary_Problem_Case_1_Agent.cs"” 에서 확인할 수 있습니다.

이후, 해당 환경에서 어떠한 강화학습 알고리즘을 적용할 것인지에 대한 고민이 이뤄졌습니다.

[Markov Decision Process]

앞서 State와 Action에 대한 정의가 이뤄졌습니다.
하지만 우리는 보상(reward)함수와 전이확률(transition probability)를 정의할 수 없습니다. 왜냐하면 같은 State라도 이후 면접자들에 따라 현재 State에서 면접을 진행하는 지원자가 최고의 지원자일수도, 아닐 수 도 있기 때문입니다. 따라서 같은 State에서 Select를 진행하더라도 받을 수 있는 보상이 달라질 수 있습니다.

또한, 우리는 앞으로 면접을 진행하게 될 지원자가 얼마나 좋은 지원자인지 알 수 없습니다. 따라서 면접관이 면접을 진행하면서 알 수 있는 “지금까지 진행한 지원자 중 현재 지원자의 순위” 에 대한 값을 action을 수행하는 시점에서 미리 일반화하기 어렵습니다. 따라서 전이확률 또한 정확히 정의할 수 없기 때문에 이 문제는 MDP를 모르는 문제입니다.

따라서 MDP로 문제를 해결할 수 없습니다.

[Monte Carlo Prediction]

앞에서 State, Action, Reward로 구성된 샘플이 존재했습니다.
그리고 100명의 지원자에 대한 면접이라는 100번의 step으로 하나의 episode가 이뤄지는 episodic한 문제입니다.

하지만 여기서 MDP를 모르는 상황이기 때문에 샘플링을 이용하는 Monte Carlo 방법론을 고려할 수 있습니다.

하지만 Monte Carlo Prediction을 적용하는 것은 무의미 합니다.
왜냐하면 “지금까지의 면접을 반영한 현재 지원자의 순위” 는 1인 상태에서만 보상을 받을 수 있는 점은 자명하고,

“현재 면접을 보는 지원자를 포함하여 지금까지 면접을 진행한 인원” 은 많을수록 현재 지원자가 높은 확률로 가장 뛰어난 지원자 라는 것도 자명하기 때문입니다.

그래서 우리는 “지금까지의 면접을 반영한 현재 지원자의 순위” 가 높고, “현재 면접을 보는 지원자를 포함하여 지금까지 면접을 진행한 인원” 일수록 상태의 가치가 높다는 것을 알 수 있습니다.

하지만 해당 state가 등장하는 확률은 다른 얘기입니다. 마지막 지원자가 가장 뛰어난 지원자라면 우리는 항상 해당 문제를 해결할 수 있습니다.

하지만 그렇지 않기 때문에 주어진 상태에서 어떤 Action을 취해야 하는 지를 고민해야 합니다.

따라서 Monte Carlo Control을 적용합니다.

[Monte Carlo Control]

Monte Carlo Control을 적용하기 위해 다음의 과정을 고려하였습니다.

q 함수를 나타내는 q 테이블을 만들어서 운용합니다.

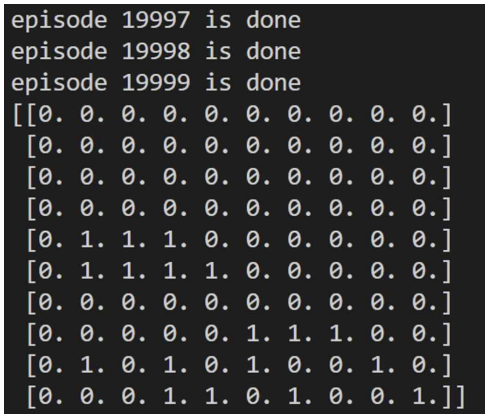
q 테이블은 다음과 같이 표현합니다.

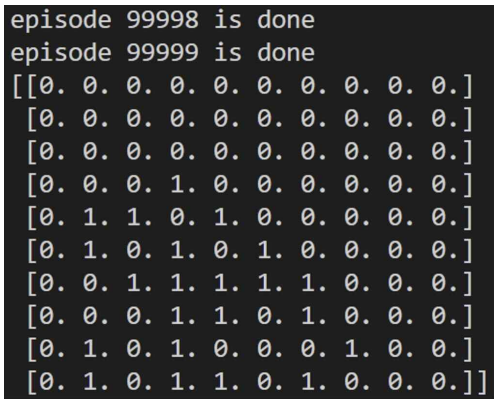
Q = np.zeros((10,10,2))
(현재 면접자 순서, 현재 면접자의 지금까지 중 순위, action - pass or select)

한 에피소드의 경험을 쌓고, 경험한 데이터로 q(s,a) 테이블의 값을 업데이트 하는 정책 평가가 이뤄집니다.

이후 업데이트 된 q(s,a) 테이블을 이용하여 epsilon greedy 정책을 만들어서 정책을 개선합니다.

해당 과정의 결과로 얻은 q테이블의 결과물은 다음과 같습니다.





해당 결과로부터 행렬의 주대각선을 기준으로 주대각선에는 1이, 상부 삼각 행렬에는 0이 분포하고, 하부 삼각 행렬에는 다시 1이 분포하는 것을 확인할 수 있었습니다.

주대각선에 1이 분포하는 것은 n번째 지원자가 n번째 순위 인 경우 반드시 Pass 해야 한다는 것이 반영된 것으로 보이며,

상부 삼각 행렬에 0이 분포하는 것은 n번째 지원자가 n번째 보다 뒷순위일 수 없가 때문에 초기화 값이 그대로 반영되어 있는 것으로 보이며,

하부 삼각 행렬에서 1열은 모두 0 (Select) 인것으로부터 지금까지 면접을 본 지원자 중 1순위인 지원자는 Select하는 것이 현명하다는 것이 반영된 것으로 보입니다.

state의 규모에 비해 iterate의 수가 비교적 부족하여 나타난 과정으로 보이지만,
충분한 iterate의 과정이 이뤄진다면

하부 삼각 행렬의 대부분에 1이 분포하며,
우리가 원하는 최적 정책을 찾아서 1열의 일부 행을 제외하고 0으로 덮이는 상태가 될 것으로 예상합니다.

소스코드는
“SecretaryProblem_PythonScripts\secretary_problem_case_monte_carlo_control.py” 에서 확인할 수 있습니다.

참고문헌

- 파이썬 예제와 함께하는 강화학습 입문
- 바닥부터 배우는 강화학습
- 따라 하면서 배우는 유니티 ML-Agents
- 파이토치와 유니티 ML-Agents로 배우는 강화학습
- https://github.com/Unity-Technologies/ml-agents
- https://github.com/Unity-Technologies/ml-agents/blob/release_20/docs/Python-LLAPI.md
- https://github.com/Unity-Technologies/ml-agents/blob/release_20/docs/Python-LLAPI-Documentation.md
- https://github.com/Unity-Technologies/ml-agents/blob/release_12_docs/docs/Learning-Environment-Create-New.md
- https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md