Goals

We want to maximize the long term monetary gain to produce a stable source of return.

Deliverables

Produce a simulation on never before tested data and produce a graph of the returns long term (multiple years). It should produce a strong return (minimum 5% target 10%, with a stability comparable to the stock market).

Formalization

g_t is the money in the pot, g_0 is the initial investment, r_t is the percent return between g_t and g_{t+1} such that $g_{t+1} = r_t g_t$.

$$\max \lim_{t \to \infty} g_t$$

We can then define the amount we invest in each team, where we have teams A and B. An investment in team A would be represented as A and a investment in no team is $\emptyset$. Thus $A_t + B_t + \emptyset_t = g_t$.

Let's assume we have a model that provides us with both a probability of team A winning, $P[W_t = A]$, and its confidence on that probability, ϵ_t.

Below is the Hoeffding bound

$$P[|v - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$$

There is also a given gambling odds O_A and O_B such that $O_A = 1 - O_B$.

$$g_{t+1} = \emptyset_t +$$

Implementation

Architecture

1. **Probability estimation**: Generates $P[W_t = A]$ and ϵ. Should create a game environment similar to a sport where I can measure and re-run to manually calculate $P[W_t = A]$ and calculate the correctness of ϵ to test is the model is constructed properly prior to using real data

2. **Gambler model**: Uses $P[W_t = A]$ and ϵ generated by the probability estimation model to find the optimal gambling strategy. Note that this model should be re-trained for each sport that is played as the sampling distribution will vary from sport to sport

Milestones

1. Train gambler reinforcment model with sim that gives $P[W_t = A]$ and ϵ see if we can profit maximize in this environment, this way we don't need any data yet its just a game agnostic model that can be applied in any gambling situation
2. Create a sports environment with a similar setup to baseball and provides a similar data
3. Write and test model until epsilon is correct within a certain bound to be defined
4. Create data API for use in training the probability estimation model
5. Train the model
6. Use end to end and test
7. Write front end and ship

Concerns to consider

There will be a tradeoff between achieving a good return and losing a large amount of pot. How accurate are betting odds int terms of expected value.

Updates

After running the gambler model that just takes in g, O, team_a, team_b and date, it returned an average earnings with an episode length of 300 of $\mu = 9.944\,04\text{e}{-1}$ and $\sigma = 6.815\,31\text{e}{-3}$. This was run with the TD3 algorithm with $\lambda_{\text{loss}} = -1.1$ and $\lambda_{\text{gain}} = 1$.

PPO with $\lambda_{\text{loss}} = -1$ had final result of $\mu = 2.195\,57\text{e}{-13}$ and $std = 2.157\,27\text{e}{-12}$, very poor. Ran A2C with the same λ performed the same.

Changing the λ value to -1.3 to see in an increase will help

None of the above worked. This is because the discount factor $\gamma = 0.99$ which throws off the model completely as it thinks that its current actions affect the future rewards. When I changed episode length to 1 the final results were $\mu = 1.683\,95$ $\sigma = 1.139\,96$. I have not tried testing large length episodes so I am unsure if this actually produces stable results. However, after simulating it with just random variables I get that it sometimes returns very large amounts but most of the time it returns 0. Out of 50 trials of episode length of 50 only 2 returned any money (it was in the billions though). Question: can i divide both μ and σ by the same amount to bolster the stability?

I will now try removing any possibility of cheating by removing all duplicates. Additionally, I will save the model so I can do future testing.

In the future try removing the `/ self.pot` in reward calc for the env.

Resulted in μ =9.803 33e−1 σ = 9.721 42e−1 sem = 3.074 18e−2. Somehow the reward was that of the original experiment where I reached a μ of 1.6. I'll have to identify all the differences.