1 Introduction

This lab is a continuation of our exploration of the TAC Ad Exchange (AdX) game. Once again, you will be building an agent for the AdX Game. However, you will be playing a different, slightly more complicated, variant of the game than in the previous lab.

In this version of this game, your agent will compete in AdX auctions over two consecutive days, and will receive a different campaign each day such that the budget campaign your agent receives on the second day depends on its success fulfilling its campaign on the first day. This success will be measured by a metric called **quality score**. Your agent will still aim to maximize its profit.

The complexity we introduce here will help you gain a better understanding of the AdX game, and the kinds of agent strategies that can be successful. You will be given the opportunity to apply the knowledge you gain in this lab in the AdX final project, which will consist of a still more complicated AdX game.

2 Game Description

This game is very similar to last week's One-Day AdX game (Lab 8). The main difference introduced in the two-day variant is the following: In the Two-Day AdX game, agents will be given an initial campaign that lasts for just one day (just like in the one-day variant), and then, contingent upon the agent's performance in fulfilling its initial campaign, which will be measured by its **quality score**, it will be given a second campaign, whose budget will be discounted by this quality score. The quality score is a number between 0 and 1.38442, where 0 denotes very low quality (in case the agent acquired 0 impressions for the first campaign), 1 denotes very good quality (in case the agent acquired the campaign's reach in its entirely), and 1.38442 denotes perfect quality (in case the agent acquired many many impressions, above and beyond the campaign's reach). Your agent is thus faced with the usual task of fulfilling its first campaign profitably, but at the same time in such a way as to earn a valuable second campaign!

This two-day game has a built-in trade off: bid too high on the first day and your agent won't be profitable on the first day, but it will earn a high quality score, giving it a chance to be very profitable the second day; bid too low on the first day and your agent can be more profitable on the first day, but if it does not fulfill its campaign's reach, it will earn a lower quality score on the first day, and hence a lower budget for its campaign on the second day.

3 Quality Score

Let R be the reach associated with a campaign C. The quality score $Q_C(x)$ earned by an agent seeking to satisfy C who procures x impressions suitable for C is computed as follows:

$$Q_C(x) = \frac{2}{a}\left(\arctan\left(a\left(\frac{x}{R}\right) - b\right) - \arctan(-b)\right),$$

where $a = 4.08577$, $b = 3.08577$, x is the amount of impressions achieved, and R is your campaign's reach.

Figure 1 plots the effective reach function $\rho(C)$ of two campaigns, one with with reach $R = 500$ (blue), and the other with reach $R = 1000$ (red). Note that $\rho(0) = 0$, $\rho(R) = 1.0$, and $\lim_{x\to\infty} \rho(C) = 1.38442$. The plot shows that the value of obtaining the first few impressions on the road to fulfilling a campaign is relatively low, compared to the value of obtaining the middle and final impressions.

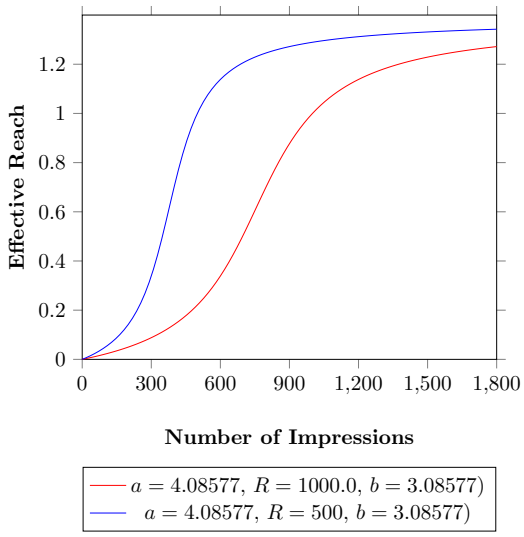

Figure 1: Effective Reach when Reach, R, equals 500 (blue) and 1000 (red).

If your agent procures zero impressions on its first campaign, then its quality score drops to zero, which means the budget of its second campaign will be identically zero (it will gain zero revenue for fulfilling that campaign!). On the other hand, if it attains a quality score greater than 0.0, then its second campaign will have a positive budget, which will hopefully lead to a positive profit.

The campaign distribution in the Two-Day, Two-Campaign game is the same as in the One-Day game. That is, the reach, target market segment, and budget of each campaign are randomly drawn as in the One-Day variant (see Lab 8, Appendix A). On the second day, however, the second campaign's budget is multiplied by $Q_{C_1}(x)$, where x is the number of impressions acquired by the first campaign, C_1.

4 API for AdX Two-Days Games

4.1 `TwoDaysBidBundle` Object

Once again, to avoid the communication overhead required to conduct each ad auction in real time (each day there are 10,000 simulated users!), the agents use a `TwoDaysBidBundle` object to communicate all their bids to the server at once.

The constructor for this `TwoDaysBidBundle` object relies on four parameters:

1. **Day**: the day for which the bid is placed, either 1 or 2.
2. **Campaign ID**: the ID for the campaign you are bidding on.
3. **Day Limit**: a limit on how much you want to spend in total on that day.
4. **Bid Entries**: a list of `SimpleBidEntry` objects, which specify how much to bid in each market segment.

A `SimpleBidEntry` object is constructed using three parameters:

1. **Market Segment**: there are a total of 26 possible market segments.
2. **Bid**: a `float`.
3. **Spending Limit**: a `float` that represents the maximum value the agent is willing to spend in this market segment.

Assume you have created a list of `SimpleBidEntry` objects called `bid_entries`. You can then create a `TwoDaysBidBundle` for your campaign on the first day as follows:

```
bid_bundle = TwoDaysBidBundle(
    day=1,
    campaign_id=self.get_first_campaign().id,
    day_limit=self.get_first_campaign().budget,
    bid_entries=bid_entries
)
```

Likewise, on the second day, you can do the same with `day=2` and `self.get_second_campaign()`.

4.2 `MyAdXAgent` Class

You should implement your agent strategy in `MyAdXAgent.py`.

Your agent class should inherit from `BaseAgent` and implement the `get_action()` method. This method receives an observation dictionary containing the current day and campaign information, and should return a `TwoDaysBidBundle` with all the agent's bids for that day. This has the following methods:

1. `get_first_campaign()` returns the campaign assigned for the first day.
2. `get_second_campaign()` returns the campaign assigned for the second day (available only on day 2).

4.2.1 Agent Naming

As in previous labs, your agent needs a name. You can give it a name by setting the `name` parameter in the main block of `MyAdXAgent.py`.

4.2.2 Helper Functions

The `core.game.market_segment` module provides helper functions to work with market segments. To iterate over all possible segments, use:

```
for segment in MarketSegment.all_segments():
    ...
```

To check if one market segment is a subset of another (i.e., whether users in `segment2` also belong to `segment1`), use:

```
if MarketSegment.is_subset(segment1, segment2):
    ...
```

(**N.B.** Market segments are subsets of themselves.)

5 Testing

To test your agent locally, run:

```
python3 MyAdXAgent.py
```

This will launch your agent, one `AggressiveAdXAgent`, and several `RandomAdXAgent` opponents in a local two-day AdX simulation using the `LocalArena` class. You can adjust the number of rounds, agents, or verbosity in the configuration near the bottom of the file.

If this test succeeds, you should see how your agent performs against the sample bots. Use this setup to debug and refine your bidding strategy before submitting your final version.

6 Competition

To connect your agent to the competition server, modify the configuration variables at the bottom of `MyAdXAgent.py`:

```
server = True
host = "insert_ip_address_provided_by_the_TA"
port = 8080
```

Then run:

```
python3 MyAdXAgent.py
```

This will connect your agent asynchronously using `connect_agent_to_server()`. Your agent will automatically submit bids and play against other competitors in each round.

A Campaign and User Distributions

Each campaign targets one of 20 possible market segments, a combination of at least two of the three attributes (chosen uniformly at random). A campaign's reach is given by the average number of users in the selected segment (listed in the tables below) times a random reach factor, selected from the set $\{\delta_1, \delta_2, \delta_3\}$, where $0 \leq \delta_i \leq 1$, for all i. The exact values of these factors are tailored to the number of agents in the game. In particular, for the ten-agent games we plan to run, we will use $\delta_1 = 0.3$, $\delta_2 = 0.5$, and $\delta_3 = 0.7$. The budget is always \$1 per impression.

Table 1: User Frequencies

Segment	**Average Number of Users**
`Male_Young_LowIncome`	1836
`Male_Young_HighIncome`	517
`Male_Old_LowIncome`	1795
`Male_Old_HighIncome`	808
`Female_Young_LowIncome`	1980
`Female_Young_HighIncome`	256
`Female_Old_LowIncome`	2401
`Female_Old_HighIncome`	407
Total	**10000**

Table 2: User Frequencies: An Alternative View

	Young	**Old**	**Total**
Male	2353	2603	4956
Female	2236	2808	5044
Total	4589	5411	10000

	Low Income	**High Income**	**Total**
Male	3631	1325	4956
Female	4381	663	5044
Total	8012	1988	10000

	Young	**Old**	**Total**
Low Income	3816	4196	8012
High Income	773	1215	1988
Total	4589	5411	10000