AdXGame

From CS1951k Lab: Brown University

The Trading Agent Competition (TAC) is an annual tournament in which teams of programmers from around the world build autonomous agents that trade in a market simulation. Application domains for these market simulations have ranged from travel to supply chain management to ad auctions to energy markets, and most recently, to ad exchanges.

You will build an agent to play a greatly simplified version of the most recent game, Ad Exchange (AdX), in which agents play the role of ad networks, competing to procure contracts (i.e., advertising campaigns) from advertisers (e.g., retailers), and then bidding in an exchange on opportunities to exhibit those ads to Internet users as they browse the web. The game is complicated by the fact that certain demographics are more valuable to certain retailers, but user demographics are not always fully visible to the ad networks. Moreover, bidding repeats over a series of simulated days (60), and agents' reputations for successfully fulfilling past contracts impact their ability to procure future contracts.

Game Description

In this project, you will build an agent to play a very simple version of the TAC AdX game. The primary simplifications are: your agent will not compete to procure campaigns (instead your agent will be assigned one, randomly, from a known distribution), user demographics are fully visible, and the game lasts thirty days. Consequently, your agent's only job in this game is to bid on (and win) impression opportunities, which are opportunities to exhibit ads to Internet users as they browse the web. And your goal will be for your agent to do so in such a way as to fulfill its campaign, as inexpensively as possible.

The private information that your agent is given at the start of this game is a single initial campaign, which represents an advertiser coming to you and saying "This contract is for some number of ads to be shown to users in some demographic, and here is my budget." That is, each campaign is characterized by the following:

- A **market segment**: a demographic(s) to be targeted:
    - There are 26 market segments in this game, corresponding to combinations of {`Male, Female`} x {`HighIncome, LowIncome`} x {`Old, Young`}. A market segment might target only one of these attributes (for example, only `Female`) or two (`Female_Young`) or three (`Female_Old_HighIncome`).
- A **reach**: the number of ads to be shown to users in the relevant market segment.
- A **budget**: the amount of money the advertiser will pay if this contract is fulfilled

Each campaign in our version of the game lasts for only one specific day. There may be variants with different campaign lengths.

In each simulation, there will be a certain number of Internet users browsing the web, with their market segments drawn from the distribution described at the end of this document. For each user, a second price sealed-bid auction is run to determine who to allocate that user's advertising space to, and how much to charge. Ties are broken randomly per user. So if there are two winning bidders in a market segment, each will be allocated (about) half the users in that segment at the price they bid.

Here is an example of a campaign:
[Segment = `Female_Old`, Reach = 500, Budget = $40.0]

To fulfill this campaign, your agent must show at least 500 advertisements to older women. If successful, it will earn $40. Showing an advertisement to a user is equivalent to winning that user's auction. But note that winning an auction for a user who does not match a campaign's market segment does not count toward fulfilling that campaign.

At the end of each simulation, the server computes the profit earned by each agent/ad network. Profit is computed as the product of the proportion of the campaign's reach fulfilled and the campaign's budget, less total spending. The proportion of reach fulfilled only counts users won in the relevant market segment, and cannot be higher than 1 (it does not help an agent to win more users than its reach). The agent with the highest profit wins that simulation. Because of the randomness in each simulation, the game is simulated repeatedly and scores are averaged over multiple simulations to determine an overall winner.

Key here is to understand that your agent is given a random campaign on the first day with some budget. 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 number of impressions needed), and 1.38442 denotes perfect quality (in case the agent acquired many many impressions, above and beyond the required number of impressions). 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.

Another point to note here is agents can only bid in these auctions once! -- before their simulation begins. Consequently, agents must reason in advance about how events might unfold over the course of the day and perhaps make contingency plans. The AdX game provides a mechanism for making a contingency plan in the form of a spending limit: i.e., an upper bound on spending that can accompany an agent's bid in each market segment.

If your agent is allocated a campaign whose market segment is very specific (e.g., Female_Old_HighIncome), then it won't have a choice about which users to bid on; it has to bid for users in precisely that market segment, or it cannot earn a positive profit. However, if its market segment is less specific (e.g., `Female`), it can bid different amounts in the `Female_Old` and `Female_Young` markets, for example, based on how much competition it thinks there will be in each. Keep in mind, though, that the order in which users arrive is

random. So if it bids more on `Female_Old` than `Female_Young`, but then if all `Female_Old` users arrive before any `Female_Young`, it may end up spending its entire budget for that campaign on `Female_Old` users. For this reason, when bidding on a market segment, your agent might want to specify a spending limit specifically for that segment.

An agent can also specify an overall spending limit to ensure that it does not spend more than some fixed amount on all users. This feature may or may not be useful, depending on your strategy.

Quality Score

Let R be the total reach of a campaign C . The quality score $Q_C(x)$ of the agent owning C as a function of the number of procured impressions, x , is computed as follow:

$$Q_C(x) = (2 / 4.08577)\,(arctan(4.08577)(x / R) - 3.08577) - arctan(-3.08577))$$

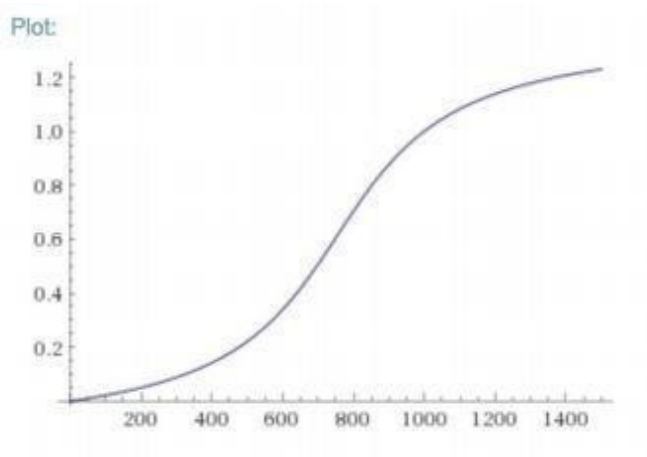
Note that $Q_C(0) = 0$, and $Q_C(R) = 1.0$ and $lim_{x \to infinity}\, Q_c = 1.38442$

If your agent procures no impressions, your quality drops to zero which means the budget of your second campaign is zero (you have no reward for fulfilling that campaign!). On the other hand, if you attain a quality score greater than 0.0, then your second campaign will have a positive budget, which could potentially lead to positive profit.

Let C1 and C2 be the first and second campaigns assigned to an agent in the first two days. The reach and budget of the first campaign are randomly drawn. The reach and budget of the second campaign are also randomly drawn, but the budget is discounted by $Q_{C1}(x)$, where *x* is the number of impressions acquired for the first campaign.

Quality Score Example

The following plot depicts the quality score function $Q_C(x)$ of a campaign with a reach of 1000.



This plot shows that the value of obtaining the first few impressions to fulfill a campaign is relatively low compared to the value of obtaining the final impressions.

API for Adx Thirty-Days Games

`ThirtyDaysBidBundle` Object

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 ThirtyDaysBidBundle object to communicate all their bids to the server at once.

The constructor for this **`ThirtyDaysBidBundle`** object takes 4 parameters:

1. **Day**: the day for which the bid is placed, ranging from 1 to 30.
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 collection of SimpleBidEntry objects, which specify how much to bid in each market segment.

A **`SimpleBidEntry`** object has 3 parameters:

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

For example, say your agent decides to bid 1.0 in all market segments in your campaign, and it wants to limit spending in total and in each market segment to the campaign's budget. First it would create a new `SimpleBidEntry` for the campaign's market segment that bids 1.0 and limits spending in the market segment to the campaign's budget:

```
SimpleBidEntry bidEntry = new
      SimpleBidEntry (
      this.myCampaign.getMarketSegment(),
      1.0,
      this.myCampaign.getBudget()));
```

Note: if your agent's market segment is more general, this entry will bid in all the specific market segments that are subsets of its market segment at the given price using the given limit. For example, if the campaign's market segment is `Female_Old`, this bid entry will bid 1.0 with a spending limit of the campaign's budget in both `Female_Old_HighIncome` and `Female_Old_LowIncome`.

Next, you would create a set of bid entries, and add this particular `bidEntry` to the set:

```
Set<SimpleBidEntry> bidEntries = new HashSet<SimpleBidEntry>();
bidEntries.add(bidEntry);
```

Finally, you would create a `ThirtyDaysBidBundle` for your campaign that includes these bid entries and limits total spending to the budget.

```
BidBundle bidBundle = new
 ThirtyDaysBidBundle ( day,
 this.myCampaign.getId(),
 this.myCampaign.getBudget(),
 bidEntries);
```

ThirtyDaysThirtyCampaignsAgent Class

Your job is to extend the abstract class **`ThirtyDaysThirtyCampaignsAgent`** by implementing the **`getBidBundle(int day)`** method. This method takes as input a day, and should return a ThirtyDaysBidBundle containing all the agent's bids for the given day. The class has one attribute: a primitive array of **`Campaign`** objects of size 30. First index i.e. **`Campaign[0]`** contains the campaign assigned to the agent for the first day of the game.
This campaign lasts for a day. **`Campaign[1]`**, contains the campaign assigned to the agent for the second day of the game, and so on. These subsequent objects are only available on the specific days they're indexed by (i.e. **`Campaign[1]`** is only available for Day 2, and not before), and are null otherwise.

SimpleThirtyDaysThirtyCampaignsAgent.java is already a sample Thirty Day Thirty Campaigns Agent given to you, off which you can build upon. But as iterated before, make a new file for your agent, and name it as per the instructions mentioned below.

Main Method

Your agent needs a main method. For this game, the main method should create an agent object which in this case takes your teamnumber.

```
public static void main(String [] args) {
MyAgent myAgent = new MyAgent("localhost" ,
9898); myAgent.connect("teamnumber");
}
```

Helper Functions

Your agent might want to iterate over market segments. The `adx.jar` file contains support code for this. Concretely, each **`MarketSegment`** is implemented as an Enum, so to iterate through them you can do something like:

```
for (MarketSegment m : MarketSegment.values()) { … }
```

The static function `marketSegmentSubset(MarketSegment m1, MarketSegment m2)` returns a boolean indicating whether m2 is a subset of m1 (this is NOT a strict subset, so segments are subsets of themselves). You may wish to use this functionality as well, to construct a more sophisticated strategy.

Including the adx.jar File

The `adx.jar` file can be found in the dependencies folder posted on Moodle. To include it in your Eclipse project: `Right-click -> Build Path -> Configure Build Path -> Libraries -> Add External JARs`

Instructions and Submission Format

1. Name your Agent filename as “teamnumberAgent.java”. So if your teamnumber is “1”, then name the agent file as “1Agent.java”
2. Follow the upper level directory structure as initially posted on Moodle while zipping and submitting (Two folders - `dependencies`, and `adxsource`, along with a README).
3. For submission, zip the contents, and name it - teamnumber_roll1_roll2_roll3.zip
4. **You are free to use whatever strategies and libraries you want, but make sure to place its JAR file in the `dependencies` folder.**
5. This is a competition style project.
6. Your team and your team number is the same as for the Reading Assignment.

Appendix

Segment	**Average Number of Users**
Male_Young_LowIncome	1,836
Male_Young_HighIncome	517
Male_Old_LowIncome	1,795
Male_Old_HighIncome	808
Female_Young_LowIncome	1,980

Female_Young_HighIncome	256
Female_Old_LowIncome	2,401
Female_Old_HighIncome	407
Total	10,000

User Distributions: An Alternative View

	Young	**Old**	**Total**
Male	2,353	2,603	4,956
Female	2,236	2,808	5,044
Total	4,589	5,411	10,000

	Low Income	**HighIncome**	**Total**
Male	3,631	1,325	4,956
Female	4,381	663	5,044
Total	8,012	1,988	10,000

	Young	**Old**	**Total**
Low Income	3,816	4,196	8,012
High Income	773	1,215	1,988
Total	4,589	5,411	10,000

Campaigns Distribution

A campaign targets one of the 26 possible market segments at random. The reach of the campaign is given by the average number of users in the selected segment (see table above) times a random discount reach factor selected from $\{_1, _2, _3\}$, where $0 \leq i \leq 1$, for all i. The budget is always $1 per impression. The exact value of each discount factor depends on the number of people playing the game and thus, will be announced after all the teams are registered.