Coding Challenge: Agent-Based Tournament

EPA1315: Data Analysis and Visualization

Rev 21 September 2019

1. Overview

In this assignment you will submit a software agent which can independently participate in a contest with, and against, other agents. In short the problem involves an assessment of when it is better to cooperate with, or to cheat, others based on previous experiences and interactions. In the following sections the learning objectives, submission deadlines and assessment procedures are outlined. The remainder of the document provides more detailed information about various aspects of the problem.

This assignment tests your ability to use functional and object-oriented programming. You will have many continued challenges and opportunities to perfect your coding abilities. However this is, I hope, a fun opportunity to get started with a concrete project. While programming this agent you will have the opportunity to interact with data structures, to create control loops, and to think about the types of data

The remainder of this document states the high-level problem (section 2). Then a description of the agents is presented – both in general, as well as requirements for your specific agent. This is section 3. Finally a description of the tournament is presented (section 4). This describes the circumstances under which the agent can compete.

2. Statement of the Problem

A social dilemma is a situation where the pursuit of individual best interests precludes achieving favourable outcomes for the group as a whole. These dilemmas persist even though individuals could make themselves better off by cooperating with others. The first and classic statement of the problem is known as the Prisoner’s Dilemma.

In the Prisoner’s Dilemma two accused prisoners are arrested and separated by the police. The prisoners are told individually that if they confess, and also testify against the other prisoner they’ll then be let off lightly. If the police collect testimony against both prisoners, the prisoners both go to jail for a long time. If the police collect testimony against one and not the other, they both go to jail. The one that confesses goes to jail for a short time, and the other goes to jail for a long time. If both resist the police pressure, and if neither confess, the police let both of the prisoners away without charge.

2.1 Theory of the Problem

The following table structures these options. Call “testify and confess” a “defect” because one of the two prisoners failed to cooperate with the other. Call “don’t confess” a “cooperate.” In this instance of a cooperation action the prisoner stands firm, and thereby holds up their end of the bargain against the police. There are four possible outcomes – an outcome where both agents cooperate, an outcome in which both agents defect, and two outcomes where one of the agents defects against the other.

Table 2. The Prisoner’s Dilemma

|  |  |  |  |
| --- | --- | --- | --- |
|  | Prisoner 2 | | |
| Prisoner 1 |  | Cooperate | Defect |
| Cooperate | ( 4, 4 ) | ( 0, 5 ) |
| Defect | ( 5, 0 ) | ( 2, 2 ) |

Inside the table there are point totals. These are like points in a board or video game – the higher the point total the better. We will use these point totals for the tournament.

The first number in table 2 is for the first prisoner, and the second number is for the second prisoner. As an example, if the first prisoner defects, and the second prisoner cooperates, a total of five points is awarded to the prisoners. All five of the points are given to prisoner one. Prisoner one achieved the best achievable outcome, and prisoner two received the worst possible outcome.

When the game is played once, the theory is clear – both players will defect against each other. To see this consider that defection is the best answer *no matter what the other player does*. If the other player cooperates, you are better off defecting. And if the other player defects, you are certainly glad that you have defected.

However when the game is played more than once, there are multiple cooperative possibilities. The “folk theorem” of game theory says that all possibilities between complete cooperation and complete defection are possible, depending on the initial beliefs of the player. Other people have also played such tournaments together.

In these tournaments generally a “tit-for-tat” strategy does really well. That is, agents which treat the other agents as they have also been treated, generally achieve a high score. Other modifications involve a “forgiving tit-for-tat” which forgives a little bad behaviour, and prevents the agents from spiralling into distrust. So I am curious to see what beliefs you as a class hold, and what strategies you will give your agent.

2.2 Summary of the Tournament Process

In a section below a full description of the tournament is offered. You may refer there for more detailed questions. However it might be useful to briefly describe how we can set up a tournament which enables agents to meet each other in repeated interactions and to grow or evolve their strategies. Object-oriented programs enable self-contained code. Using such code we can create agents, and stage them against each other through repeated interactions. Agents may employ a range of different strategies. Agents which can consult a history of interactions may have a strategic advantage over more naïve agents.

3. Specification of the Agent

In order to participate in the tournament your agent must adhere to some requirements. The requirements are as follows:

1. You should create an R6 object. The name of the object should be “Agent.”
2. You need to be able to receive a round number. This should be through a function called set\_round(). You do not need to use this information.
3. You need to be able to receive your own id, as well as that of your opponent. These functions should be called set\_id(), and set\_opponent\_id(). You do not need to use this information.
4. You need to have a greeting. This should be a property of the agent which I can access. This should be called greeting.
5. You need to receive a greeting from the agent. This is a function where I can pass you a string. This function must be called set\_response(). (You do not have to use this information.)
6. You need to receive the book. This is a comprehensive history of the tournament. This function should be called set\_book(). I will pass you a data frame.
7. You need to accept a request to formulate your bid. This function call needs to be publicly accessible, so that I can call it. This function should be called get\_bid().
8. If you make additional functions for your own purposes, they must be inside the scope of your agent.
9. You need to have an object property, which I can then access, called bid. This contains your move. Your move should be a character variable. The permissible values are “ cooperate” or “defect.”

Your agent must be in the R language. This is for two reasons – our learning goals are to use R, and further to allow your agent to interact with others. So it’s important to specify a common language.

You may use whatever R packages you wish in programming your agent. You must use the R6 package to create your objects. I will load the necessary packages to ensure your agent runs in the tournament environment.

3.1 Example Agent

The following R agent is a fully-functional agent, which would be acceptable as a submission. The agent has several object attributes, including round, my\_id, opponent\_id, my\_greeting, greeting\_response, and bid. There are a number of set functions, which can be used to pass the agent important information. These functions are used to set the round, the opponent id, the greeting response, and the book. The agent will formulate a bid using the get\_bid() function.

library(R6)

Agent <- R6Class("Agent",

public = list(

bid = NULL,

book = NULL,

greeting = "Hi!",

id = NULL,

opponent\_id = NULL,

round = NULL,

response = NULL,

set\_book = function(book=NA) {

self$book <- book

},

set\_id = function(id=NA) {

self$id = id

},

set\_opponent\_id = function(opponent\_id=NA) {

self$opponent\_id = opponent\_id

},

set\_response = function(response=NA) {

self$ response <-response

},

set\_round = function(round=NA) {

self$round <- round

},

get\_bid = function() {

bid\_vector <- c("cooperate","defect")

self$bid <- sample(bid\_vector,1)

}

)

)

This particular agent is not very smart. It cooperates and defects at random. Perhaps you can do better, using information such as the round number, the book, or the greeting\_response! To do this you will want to add your own functions in the agent, and utilize them through the get\_bid() function.

3.2 Available Information

If you do choose to use the book, it looks like the following. As you can see there are five columns, containing the agents ids – id1, and id2. It also includes the trade number. And then it shows the bids of the two agents.

The bidding process is always simultaneous. There is no particular meaning in which agent is placed in the id1 field, or the id2 field. The individual agent numbers always remain the same however. So for instance when we next see agent 5, it will be the same agent 5 which participated in the first trade of the tournament.

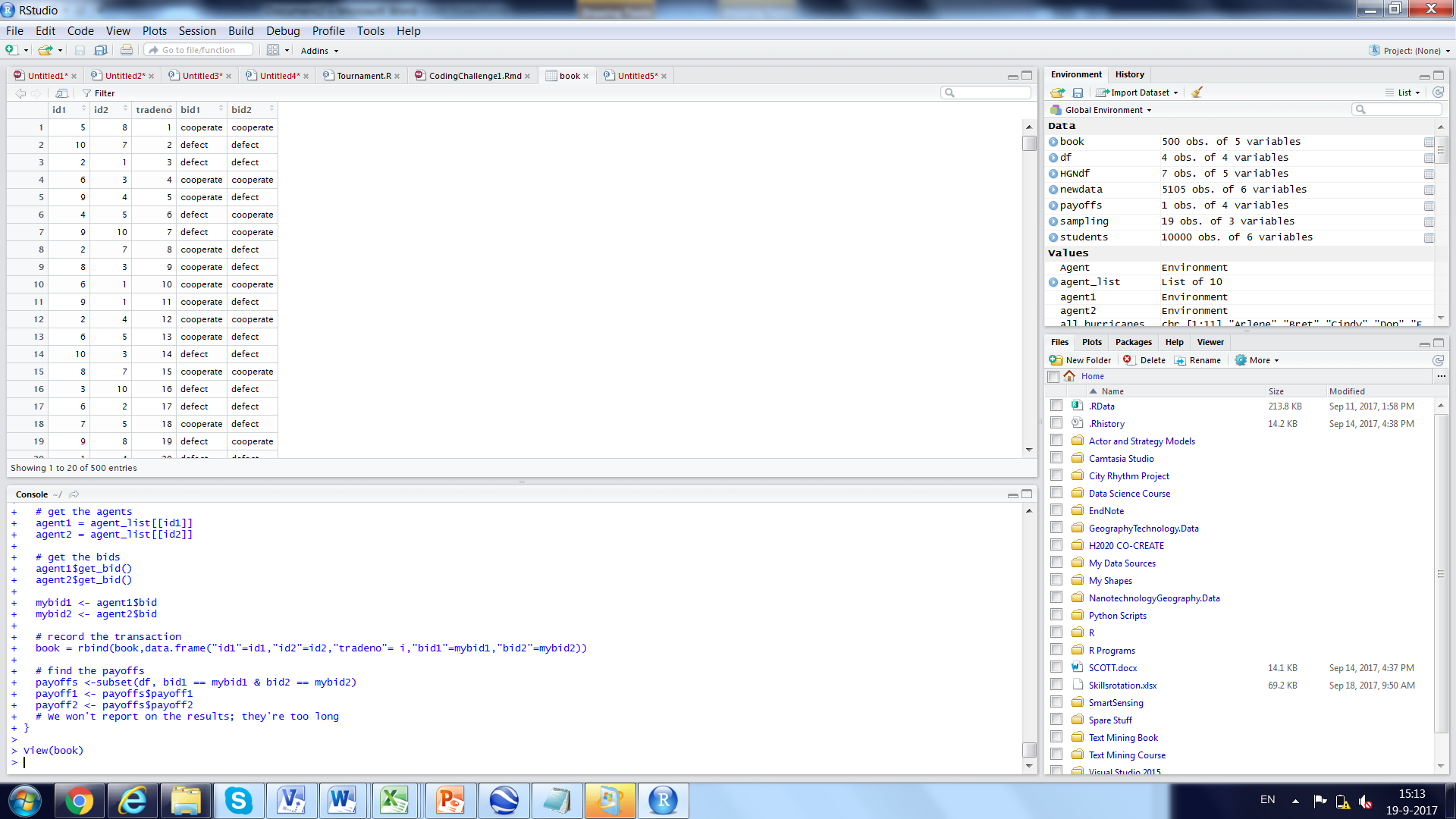


Figure 1. Sample Book

This table was generated by the “dumb” agents which choose at random. But the during the real tournaments there may be agents which calculate a smart strategy. Therefore there is probably a lot of valuable information in this table. See if you can use it to your advantage!

3.4 Agent Population

There will be one agent for each group in class – perhaps 28 agents in total. I will add four agents of my own to the population. These agents will greet by you saying “Lemon!”. They will always cheat you.

4. Description of the Tournament

4.1 Detailed Process Description

Here is a description of the tournament process.

1. All agents are assigned an anonymous, but unique, identity number.
2. Your agent will be invited to participate against other agents up to 1000 times. All agents will get an equal number of plays.
3. I will shut down the tournament early to prevent individuals gaming the end rounds of the tournament. I will randomly choose a number of rounds from 700-1000. This will be the actual number of times the game is played. No agent will know the actual end round.
4. Agents will be paired against each other randomly. As an accident of chance you may meet some agents more often than others.
5. At the beginning of each round your agent will be told the round number.
6. At the beginning of each round your agent will be given the full history of the tournament up till the current round. You will be given the identity number of your opponent for this round. With this information you can look up their trading history.
7. Before making a move, agents will both send and receive a greeting, which will be a short phrase in characters.
8. Both agents will make their move. When they move they can only “cooperate” or “defect.” Of course they may make a different move each time, and the actual move made can be conditioned on the history of the tournament and the history of the other agent.
9. The tournament keeper will award the points for the round according to the point scheme given above. This information will be recorded in a private ledger. No agents can write to this ledger, but all will be allowed to read it.
10. The sum total of earned points during the game, across multiple rounds, is the indicator of a winning agent.
11. Agents which take an excessive amount of time to process their information may be banned from the tournament. Agents which hang, or fail to make a bid at all, will certainly be banned from the tournament. Banned agents will receive some score lower than the participating agents.

4.2 Example Tournament

I implement the tournament on my side. You don’t need to do anything with this information, but you may want to see the details so as to best prepare your agent. Setting up the tournament has four parts.

Part 1. Serializing the Agents. I will put your R code in a directory. Then I will load each of your scripts and initialize your agents. Your agent will be assigned a unique number, and will be added to a list to access later, during the tournament.

Part 2. Creating the Contest. I will pair up the agents so that each agent gets to play against an opponent up to 1000 times. The code for this looks like the following:

# each agent gets to play 1000 times

# matches are paired opportunities for agents to compete

matches <- c()

for (i in 1:1000) {

# There will be an even number of agents, up to about 40

# each agent has a unique id

id\_list<-1:40

# shuffle the list in place

shuffle <- sample(id\_list)

while (length(shuffle) > 0) {

# get two agents off the list

el1 <- shuffle[[1]]

el2 <- shuffle[[2]]

# remove two agents off the front of the list

shuffle <- shuffle[-1]

shuffle <- shuffle[-1]

# contestants shows the pairing

contestants <- c(el1,el2)

# add your contest to the end of the match list

matches[[length(matches)+1]] <- contestants

}

}

Part 3. Running the Tournament. I then call the agents to determine their bid in the tournament. The code for this looks like the following.

# Run the Tournament

x <- length(contestants)

# Contestants is a list of list

# The outer list is a list of contests

# The inner list is a list of two agents

for (i in 1:x) {

# find the first contest

# get the list, not the slice

contest <- contestants[[i]]

# get the ideas

id1 <- contest[[1]]

id2 <- contest[[2]]

# get the agents

agent1 = agent\_list[[id1]]

agent2 = agent\_list[[id2]]

# set the agent ids

agent1$set\_id(id1)

agent2$set\_id(id2)

# set the opponent number

agent1$set\_opponent\_id(id2)

agent2$set\_opponent\_id(id1)

# get the greetings

greeting1 = agent1$greeting

greeting2 = agent2$greeting

#send the greetings

agent1$ set\_response (greeting2)

agent2$ set\_response (greeting1)

# set the round number

agent1$set\_round(i)

agent2$set\_round(i)

# get the bids

agent1$get\_bid()

agent2$get\_bid()

mybid1 <- agent1$bid

mybid2 <- agent2$bid

# Run the Tournament

# record the transaction

book = rbind(book,data.frame("id1"=id1,"id2"=id2,"tradeno"= i,"bid1"=mybid1,"bid2"=mybid2))

# find the payoffs

payoffs <-subset(df, bid1 == mybid1 & bid2 == mybid2)

payoff1 <- payoffs$payoff1

payoff2 <- payoffs$payoff2

}

So this concludes the handbook for the submission. Good luck with your efforts. I look forward to running the tournament, and sharing the results with you.