contextual: Simulating Contextual Multi-Armed Bandit Problems in R

Robin van Emden
JADS

Eric Postma
Tilburg University

Maurits Kaptein
Tilburg University

Abstract

Contextual bandit algorithms have been gaining in popularity due to their effectiveness and flexibility in the online evaluation of partial information sequential decision problems - from online advertising and recommender systems to clinical trial design and personalized medicine. At the same time, there are as of yet surprisingly few options that enable researchers and practitioners to simulate and compare the wealth of new and existing Bandit algorithms in a practical, standardized and extensible way. To help close this gap between analytical research and real-life application the current paper introduces the object-oriented R package **contextual**: a user-friendly and, through its clear object oriented structure, easily extensible framework that facilitates the parallel comparison of, amongst others, contextual and non-contextual Bandit policies by means of both simulation and offline analysis.

Keywords: contextual multi-armed bandits, simulation, sequential experimentation, R.

1. Introduction

There are many real-world situations in which we have to decide between a set of options but only learn about the best course of action by choosing one way or the other repeatedly, learning but one step at a time. In such situations, the basic premise stays the same for each renewed decision: do you stick to what you already know and receive an expected result ("exploit") or choose something you don't know all that much about and potentially learn something new ("explore")? As we all encounter such dilemma's on a daily basis, it is easy to come up with a great many examples - for instance:

- Do you feed your next coin to the one-armed bandit that paid out last time, or do you test your luck on another arm, on another machine?
- When going out to dinner, do you explore new restaurants, or do you exploit familiar ones?
- Do you stick to your current job, or explore and hunt around?
- Do I keep my current stocks, or change my portfolio and pick some new ones?
- As an online marketer, do you try a new ad, or keep the current one?
- As a doctor, do you treat your patients with tried and tested medication, or do you prescribe a new and promising experimental treatment?

Every one of these issues represents yet another take on the same underlying dilemma: when to explore versus when to exploit. To get a better grip on such decision problems, and to learn if and when specific strategies might be more successful than others, such explore/exploit dilemmas have been studied extensively under the title of "the Multi-Armed Bandit problem" (MAB problem).

More formally, in MAB problems, at each time step t, an algorithm or "policy" chooses one of several available options (defined as the "arms" of a "multi-armed bandit," named after the old slot machines of yore, with one arm on the side). Following its decision to make a particular choice, or play an "arm," the policy then observes the reward it receives from the "bandit" on each consecutive choice - with the goal of maximizing total reward over time. That is, a MAB policy suggests when to explore new options and when to exploit known ones, where, importantly, for each decision, at each time step t, the only information that is acquired is the reward for the latest decision. The algorithm remains in the dark about the potential rewards of the unchosen options and any other information outside of current and past rewards and choices.

Since it's inception in the 50's, the formalization of the exploit-explore dilemma has given rise a lively and active field of inquiry under the umbrella of the "Multi-Armed Bandit problem", producing both analytically proven optimal and computationally more tractable approximate policies. And building on these solid foundations, a relatively recent generalization known as the contextual Multi-Armed Bandit (cMAB) adds one crucial element to the equation. In addition to past decisions and their rewards, cMAB policies are allowed to make use of side information on the state of the world at each t, right before making their decision. In other words, an agent that follows the advice of a cMAB policy may decide differently in different contexts.

This access to side information makes cMAB algorithms even more relevant to many real-life decision problems than its MAB progenitors: do you show a particular add to returning customers, to new ones, or both? Do you prescribe a different treatment to male patients, female patients, or both? In the real world, it appears almost no choice exists without a context. So it may be no surprise that cMAB algorithms have found many applications: from recommender systems and advertising to health apps and personalized medicine - inspiring a multitude of new, often analytically derived bandit algorithms or policies, each with their strengths and weaknesses

Still, though cMAB algorithms have gained traction in both research and industry, comparisons on simulated, and, importantly, real-life, large-scale offline "partial label" data sets have relatively lagged behind. To this end, the current paper introduces the **contextual** R package. **contextual** aims to facilitate the simulation, offline comparison, and evaluation of (Contextual) Multi-Armed bandit policies. There exist a few other frameworks that enable the analysis of offline datasets in some capacity, such as Microsoft's Vowpal Wabbit, and the MAB focussed python package Striatum. But, as of yet, no extensible and widely applicable R package that can analyze and compare, respectively, K-armed, Continuum, Adversarial and Contextual Multi-Armed Bandit Algorithms on either simulated or offline data.

In section 2, this paper will continue with a more formal definition of MAB and CMAB problems and relate it to our implementation. In section 3, we will continue with an overview of **contextual**'s object-oriented structure In section 4, we list the policies that are available by default, and simulate two MAB policies and a cMAB policy. In section 5, we demonstrate how easy it is to extend contextual with a policy (RVE: NOTE TO SELF: also custom bandit?) of your own. In section 6, we replicate two papers, thereby demonstrating how to test policies on offline data sets. Finally, in section 7, we will go over some of the additional features in the package and conclude with some comments on the current state of the package and possible enhancements.

2. From formalisation to implementation

2.1. Formalisation

In formalizing the described cMAB problem, we define an algorithm or **policy** π, that seeks to maximize its total **reward** (that is, to maximize its cumulative reward $\sum_{t=1}^{T} r_t$ [1]). This **policy** observes information on the current state of the world represented as a contextual feature vector $x_t = (x_{1,t}, \ldots, x_{d,t})$. Based on earlier payoffs, the **policy** then selects one of the Bandit's arms $a_t \in \{1, \ldots, k\}$, and receives reward $r_{a_t,t}$, the expectation of which depends both the context and the reward history of that particular arm. With this observation $(x_{t,a_t}, a_t, r_{t,a_t})$, the policy now updates its arm-selection strategy. These steps are then repeated T times, where T is often defined as the Bandit's "**horizon**" T.

Schematically, for each round t= { 1, . . . , T }:

1) Policy observes state of the world as a contextual feature vector
2) Bandit generates reward vector $r_t = (r_{t,1}, \ldots, r_{t,k})$
3) Policy π selects one of the Bandit's arms $a_t \in \{1, \ldots, k\}$
4) Policy π observes reward r_{t,a_t} from Bandit and updates its arm-selection strategy with $(x_{t,a_t}, a_t, r_{t,a_t})$

Where the goal of the policy is to miminize expected cumulative regret:

- $R_T = \mathbb{E}\left[\sum_{T}^{t=1}(r_{a*,t-r})\right] = \mathbb{E}\left[\sum_{T}^{t=1}(\mu * -\mu_{a_t})\right]$

2.2. Basic Implementation

We set out to develop an implementation that stays close to the previous formalisation, while offering maximum flexibility and extendibility. As an added bonus, this kept the class structure of the package elegant and simple, with the following five classes forming the backbone of the package:

- `Bandit`: generates rewards and contexts. These can be generated synthetically, based on offline data, etc.
- `Policy`: observes the context and the rewards of a Bandit, uses these observations to update of a set of parameters θ, and decides which arm to choose next.
- `Agent`: encapsulates, and is responsible for the flow of information between and running of one Bandit/Policy pair.
- `Simulation`: the entry point of any contextual simulation. It encapsulates one or more agents, potentially clones them, runs them, and saves the log of all of the agents interactions to a History object.
- `History`: wraps a data.table a data.table that keeps a log of all interactions.

[1] or to minimize its cumulative or expected regret

- `Plot`: generates plots based on History data. It can be invoked by calling plot(x) for a History instance.

Importantly, any particular `Bandit` or `Policy` class has to inherit from, and implement the methods of, its respective abstract superclass. From this follows that in our framework, on being run by the Simulator, an Agent repeatedly executes the next few lines for every t in t=1,2,...,T (see also 2.2)

1a) Agent checks the bandit for side information that might influence the expression of its arms

1b) Bandit returns feature vector Xt

2a) Agent asks policy π which of the bandit's K arms to choose given Xt

2b) Given Xt, policy π advices action a_t based on the state of a set of parameters θ_t

3a) Agent does action a_t by playing the suggested bandit arm.

3b) Bandit rewards the agent with reward r_t for action a_t,

4a) The agent sends the reward r_t to policy π

4b) Policy π uses r_t to update the policy's set of parameters θ_t g iven Xt

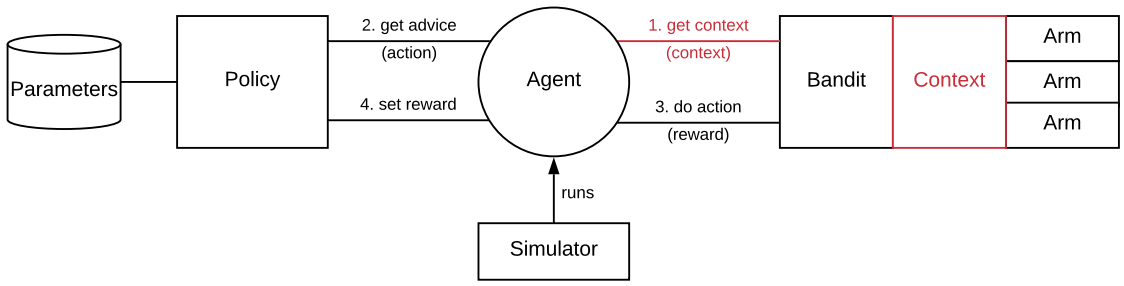

Figure 1: Basic overview **contextual**'s structure

3. Object-oriented setup of the package

3.1. The R6 Class System

Statistical computational methods, in R or otherwise, are often made available through single-use scripts. Usually, these scripts are meant to give a basic idea of a statistical method, technique or algorithm in the context of a scientific paper. This is of no direct consequence within that particular setting. But when a set of well-researched interrelated algorithms find growing academic, practical and commercial adoption, it becomes crucial to offer a more standardized and more accessible way to compare different methods and algorithms.

A concern has become particularly pressing in the cMAB literature, where there is a tendency to publish analytical formalizations without any readily available script or implementation - while there is, at the same time, an ever growing interest in the practical application of cMAB algorithms.

The contextual package means to address this by making available an easily extendible framework, together with a library containing clear example implementations of several of the best known and most popular Contextual Bandit algorithms. For us, it made the most sense to create such a package in R. Firstly, as R is currently the de facto language for the dissemination of new statistical methods, techniques, and algorithms, while also being widely used in industry to simulate and test both new and existing algorithms. This makes it a sensible arena to bring together the interests, source code and data of both research and industry.

At the same time, it was clear to us that it would be of critical importance to make our R based framework as clear and as easily extensible as possible. We, therefore, chose to build our Object Oriented package on the R6 Object system. In contrast to the older S3 and S4 object systems, R6 methods are mutable and belong to their objects. That means that R6 objects behave, feel and look more like objects in computer languages like Python and Java. Together with its speed, simplicity, and clarity, we think contextual's use of R6 has indeed to enabled to achieve all of the aforemetioned goals.

The R6 package allows the creation of classes with reference semantics, similar to R's built-in reference classes. Yet compared to reference classes, R6 classes are simpler and lighter-weight, and they are not built on S4 classes, so they do not require the methods package. At the same time, classes do allow public and private members, and they support inheritance, even when the classes are defined in different packages. One R6 class can inherit from another. In other words, you can have super- and sub-classes. Subclasses can have additional methods, and they can also have methods that override the superclass methods.

This enabled us to translate the basic cMAB formalization from section 2 almost one on one to a clear object oriented structure. To clarify how our objects hang together, we created two UML diagrams. The UML class diagram shown in figure X visualizes the structure of our package by showcasing contextual's classes, attributes, and relationships between classes. The UML sequence diagram in figure X, on the other hand, shows how contextuals's classes behave over time. It depicts the objects and classes involved over one time step t, and it displays a basic version of the sequence of messages exchanged between all of contextual's basic objects.

3.2. Main classes

Main classes1

Main classes2

4. Basic use of the package

Here, we show how to simulate some bandits, with their current implementation.

4.1. Epsilon First

In this algorithm, also known as AB(C) testing, a pure exploration phase is followed by a pure exploitation phase. The Epsilon First policy is equivalant to the setup of a randomized controlled trial (RCT): a study design where people are allocated at random to receive one of several clinical interventions. One of these interventions is the control. This control may be a standard practice, a placebo, or no intervention at all. On completion of the RCT, the best solution at that point is then suggested to be the superior "evidence based" option for everyone, at all times.

For figures, see Figure **??** on page **??**.

The policy:

Algorithm 1 Epsilon First

Require: $\eta \in \mathbb{Z}^+$, number of time steps t in the exploration phase
  $n_a \leftarrow 0$ for all arms a $\in \{1, \ldots, k\}$ (count how many times an arm has been chosen)
  $\hat{\mu}_a \leftarrow 0$ for all arms a $\in \{1, \ldots, k\}$ (estimate of expected reward per arm)
  for $t = 1, \ldots, T$ **do**
    if $t \leq \eta$ **then**
      play a random arm out of all arms a $\in \{1, \ldots, k\}$
    else
      play arm $a_t = \arg\max_a \hat{\mu}_{t=\eta,a}$ with ties broken arbitrarily
    end if
    observe real-valued payoff r_t
    $n_{a_t} \leftarrow n_{a_{t-1}} + 1$
    $\hat{\mu}_{t,a_t} \leftarrow \dfrac{r_t - \hat{\mu}_{t-1,a_t}}{n_{a_t}}$
  end for

The EpsilonFirstPolicy class:

```
EpsilonFirstPolicy <- R6::R6Class(
  public = list(
    first = NULL,
    initialize = function(first = 100, name = "EpsilonFirst") {
      super$initialize(name)
      self$first <- first
    },
    set_parameters = function() {
      self$theta_to_arms <- list('n' = 0, 'mean' = 0)
    },
    get_action = function(context, t) {
      if (sum_of(theta$n) < first) {
        action$choice          <- sample.int(context$k, 1, replace = TRUE)
        action$propensity   <- (1/context$k)
      } else {
        action$choice          <- max_in(theta$mean, equal_is_random = FALSE)
        action$propensity   <- 1
      }
      action
    },
```

```
    set_reward = function(context, action, reward, t) {
      arm      <- action$choice
      reward   <- reward$reward

      inc(theta$n[[arm]]) <- 1
      if (sum_of(theta$n) < first - 1)
        inc(theta$mean[[arm]] ) <- (reward - theta$mean[[arm]]) / theta$n[[arm]]

      theta
    }
  )
)
```

Running the policy:

```
library("contextual")

horizon            <- 100
simulations        <- 100
arm_weights        <- c(0.9, 0.1, 0.1)

policy             <- EpsilonFirstPolicy$new(first = 50, name = "EFirst")
bandit             <- SyntheticBandit$new(arm_weights = arm_weights)

agent              <- Agent$new(policy,bandit)

simulator          <- Simulator$new(agents = agent,
                                    horizon = horizon,
                                    simulations = simulations)

history            <- simulator$run()


par(mfrow = c(1, 2),mar = c(5, 5, 1, 1))
plot(history, type = "cumulative")
plot(history, type = "arms")
```

4.2. Epsilon Greedy

This is an algorithm for continuously balancing exploration with exploitation. A randomly chosen arm is pulled a fraction ϵ of the time. The other 1-ϵ of the time, the arm with highest known payout is pulled.

For figures, see Figure **??** on page **??**.

The algorithm:

Translated to the EpsilonGreedyPolicy class:

Algorithm 2 Epsilon Greedy

Require: $\epsilon \in [0, 1]$ - exploration tuning parameter

$n_a \leftarrow 0$ for all arms a $\in \{1, \ldots, k\}$ (count how many times an arm has been chosen)

$\hat{\mu}_a \leftarrow 0$ for all arms a $\in \{1, \ldots, k\}$ (estimate of expected reward per arm)

for $t = 1, \ldots, T$ **do**

  if sample from $\mathcal{N}(0, 1) > \epsilon$ **then**

    play arm $a_t = \arg\max_a \hat{\mu}_{t-1,a}$ with ties broken arbitrarily

  else

    play a random arm out of all arms a $\in \{1, \ldots, k\}$

  end if

  observe real-valued payoff r_t

  $n_{a_t} \leftarrow n_{a_{t-1}} + 1$

  $\hat{\mu}_{t,a_t} \leftarrow \frac{r_t - \hat{\mu}_{t-1,a_t}}{n_{a_t}}$

end for

```
EpsilonGreedyPolicy <- R6::R6Class(
  public = list(
    epsilon = NULL,
    initialize = function(epsilon = 0.1, name = "EGreedy") {
      super$initialize(name)
      self$epsilon <- epsilon
    },
    set_parameters = function() {
      self$theta_to_arms <- list('n' = 0, 'mean' = 0)
    },
    get_action = function(context, t) {
      if (runif(1) > epsilon) {
        action$choice <- max_in(theta$mean)
        action$propensity <- 1 - self$epsilon
      } else {
        action$choice <- sample.int(context$k, 1, replace = TRUE)
        action$propensity <- epsilon*(1/context$k)
      }
      action
    },
    set_reward = function(context, action, reward, t) {
      arm <- action$choice
      reward <- reward$reward
      inc(theta$n[[arm]])    <- 1
      inc(theta$mean[[arm]]) <- (reward - theta$mean[[arm]]) / theta$n[[arm]]
      theta
    }
  )
)
```

How to run it:

```
library("contextual")

horizon            <- 100
simulations        <- 100
arm_weights        <- c(0.9, 0.1, 0.1)

policy             <- EpsilonGreedyPolicy$new(epsilon = 0.1, name = "EG")
bandit             <- SyntheticBandit$new(arm_weights = arm_weights)

agent              <- Agent$new(policy,bandit)

simulator          <- Simulator$new(agents = agent,
                                    horizon = horizon,
                                    simulations = simulations)

history            <- simulator$run()


par(mfrow = c(1, 2),mar = c(5, 5, 1, 1))
plot(history, type = "cumulative")
plot(history, type = "arms")
```

4.3. Contextual Bandit: LinUCB with Linear Disjoint Models

The algorithm:

Algorithm 3 LinUCB with linear disjoint models

Require: $\alpha \in \mathbb{R}^+$, exploration tuning parameter
  for $t = 1, \ldots, T$ **do**
    Observe features of all arms $a \in \mathcal{A}_t : x_{t,a} \in \mathbb{R}^d$
    for $a \in \mathcal{A}_t$ **do**
      if a is new **then**
        $A_a \leftarrow I_d$ (d-dimensional identity matrix)
        $b_a \leftarrow 0_{d \times 1}$ (d-dimensional zero vector)
      end if
      $\hat{\theta}_a \leftarrow A_a^{-1} b_a$
      $p_{t,a} \leftarrow \hat{\theta}_a^T + \alpha \sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}$
    end for
    Play arm $a_t = \arg\max_a p_{t,a}$ with ties broken arbitrarily and observe real-valued payoff r_t
    $A_{a_t} \leftarrow A_{a_t} + x_{t,a_t} x_{t,a_t}^T$
    $b_{a_t} \leftarrow b_{a_t} + r_t x_{t,a_t}$
  end for

This is how the algorithm works: at each step, we run a linear regression with the data we have collected so far such that we have a coefficient for each context feature. We then observe our new

context, and generate a predicted payoff using our model. We also generate a confidence interval for that predicted payoff for each of the three arms. We then choose the arm with the highest upper confidence bound.

For figures, see Figure **??** on page **??**.

```
#' @export
LinUCBDisjointPolicy <- R6::R6Class(
  public = list(
    alpha = NULL,
    initialize = function(alpha = 1.0, name = "LinUCBDisjoint") {
      super$initialize(name)
      self$alpha <- alpha
    },
    set_parameters = function() {
      self$theta_to_arms <- list( 'A' = diag(1,self$d,self$d), 'b' = rep(0,self$d))
    },
    get_action = function(context, t) {
      expected_rewards <- rep(0.0, context$k)
      for (arm in 1:self$k) {
        X          <-  context$X[,arm]
        A          <-  theta$A[[arm]]
        b          <-  theta$b[[arm]]
        A_inv      <-  solve(A)

        theta_hat  <-  A_inv %*% b
        mean       <-  X %*% theta_hat
        sd         <-  sqrt(tcrossprod(X %*% A_inv, X))
        expected_rewards[arm] <- mean + alpha * sd
      }
      action$choice  <- max_in(expected_rewards)
      action
    },
    set_reward = function(context, action, reward, t) {
      arm <- action$choice
      reward <- reward$reward
      Xa <- context$X[,arm]

      inc(theta$A[[arm]]) <- outer(Xa, Xa)
      inc(theta$b[[arm]]) <- reward * Xa

      theta
    }
  )
)
```

```
horizon          <- 100L
simulations      <- 300L
                              # k=1  k=2  k=3           columns represent arms
context_weights  <- matrix(  c( 0.9, 0.1, 0.1,  # d=1
                                0.1, 0.9, 0.1,  # d=2
                                0.1, 0.1, 0.9), # d=3   rows for context features
```

```
                                nrow = 3, ncol = 3, byrow = TRUE)

bandit            <- SyntheticBandit$new(context_weights = context_weights)

agents            <- list(Agent$new(EpsilonGreedyPolicy$new(0.1, "Egreedy"), bandit),
                          Agent$new(OraclePolicy$new("Oracle"), bandit),
                          Agent$new(LinUCBDisjointPolicy$new(1.0, "LinUCB"), bandit))

simulation        <- Simulator$new(agents, horizon, simulations)
history           <- simulation$run()

par(mfrow = c(1, 2),mar = c(5, 5, 1, 1))
plot(history, type = "cumulative")
plot(history, type = "cumulative", regret = FALSE)
```

5. Extending the package

Through its R6 based object system, it's relatively easy to extend contextual. Below, we demonstrate how to make use of that extensibility through the implementation of a PoissonRewardBandit extending contextual's BasicBandit class, and of an PoissonRewardBandit version of the Epsilon Greedy policy presented above.

```
PoissonRewardBandit <- R6::R6Class(
  "PoissonRewardBandit",
  # Class extends BasicBandit
  inherit = BasicBandit,
  public = list(
    initialize   = function(weights) {
      super$initialize(weights)
    },
    # Overrides BasicBandit's do_action to generate Poisson based rewards
    do_action = function(action, t) {
      reward_means = c(2,2,2)
      private$R <- matrix(rpois(3, reward_means) < self$get_weights(), self$k, self$d)*1
      private$reward_to_list(action, t)
    }
  )
)

EpsilonGreedyAnnealingPolicy <- R6::R6Class(
  "EpsilonGreedyAnnealingPolicy",
  # Class extends EpsilonGreedyPolicy
  inherit = EpsilonGreedyPolicy,
  portable = FALSE,
  class = FALSE,
  public = list(
```

```
    get_action = function(context, t) {
      # Override get_action to make annealing
      epsilon = 1 / log(t + 0.0000001)
      if (runif(1) > epsilon) {
        action$choice <- max_in(theta$mean)
        action$propensity <- 1 - self$epsilon
      } else {
        action$choice <- sample.int(context$k, 1, replace = TRUE)
        action$propensity <- epsilon*(1/context$k)
      }
      action
    }
  )
)

weights     <- c(7,1,2)
bandit      <- PoissonRewardBandit$new(weights)
agents      <- list( Agent$new(EpsilonGreedyPolicy$new(0.1, "EG Annealing"), bandit),
                     Agent$new(EpsilonGreedyAnnealingPolicy$new(0.1, "EG"), bandit) )
simulation  <- Simulator$new(agents, horizon = 200L, simulations = 100L)

history     <- simulation$run()

par(mfrow = c(1, 2),mar = c(5, 5, 1, 1))
plot(history, type = "cumulative")
plot(history, type = "average", regret = FALSE)
```

6. Simulation and Offline evaluation Bandits

6.1. Simulation

Some info on the implemented simulating Bandits, inc strengths and weakenesses.

*** Basic very simple ***

*** Based on modeling ***

6.2. Offline evaluation

Offline evaluation through LiLogBandit

Though it is, as demonstrated in the previous section, relatively easy to create basic simulators to test simple MAB and cMAB policies, the creation of more complex simulations that generate more complex contexts for more demanding policies can become very complicated very fast. So much so, that the implementation of such simulators regularly becomes more complex than the analysis and implementation of the policies themselves. More seriously, even when succeeding in surpassing these

technical challenges, it remains an open question if an evaluation based on simulated data reflects real-world applications, as modeling by definition introduces bias.

But there exists another, unbiased approach to testing MAB and cMAB policies. This approach makes use of widely available offline sources of data and can pre-empt the issues of bias and model complexity. It also offers the secondary advantages that offline data is both widely available and reflective of real-world online interactions. But there is one catch, that is particular to the evaluation of MAB problems: when we seek to make use of offline data, we miss out on user feedback when a policy advices a different arm than the one the user selected. In other words, offline data is only "partially labeled" with respect to any Bandit policies, as bandit evaluations only contain user feedback for arms that were displayed to the agent but include no information on other arms.

*** explain how li log algorithm helps here***

*** insert algorithm ***

*** insert code ***

Offline evaluation through DoublyRobustBandit

*** insert algorithm ***

*** insert code ***

7. Replications with offline data

Here we replicate some papers with a huge offline dataset..

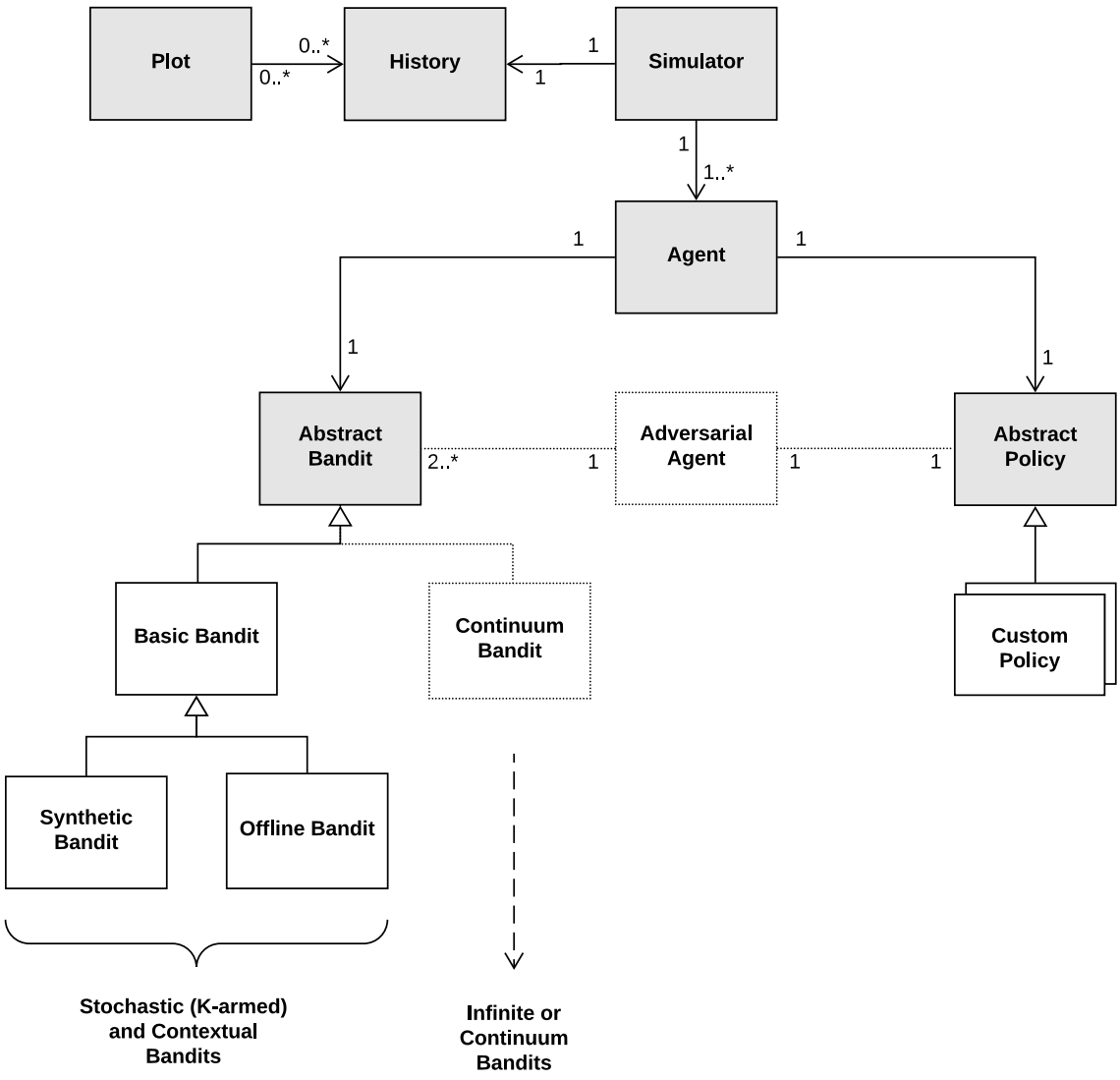

Figure 2: **contextual** UML Class Diagram

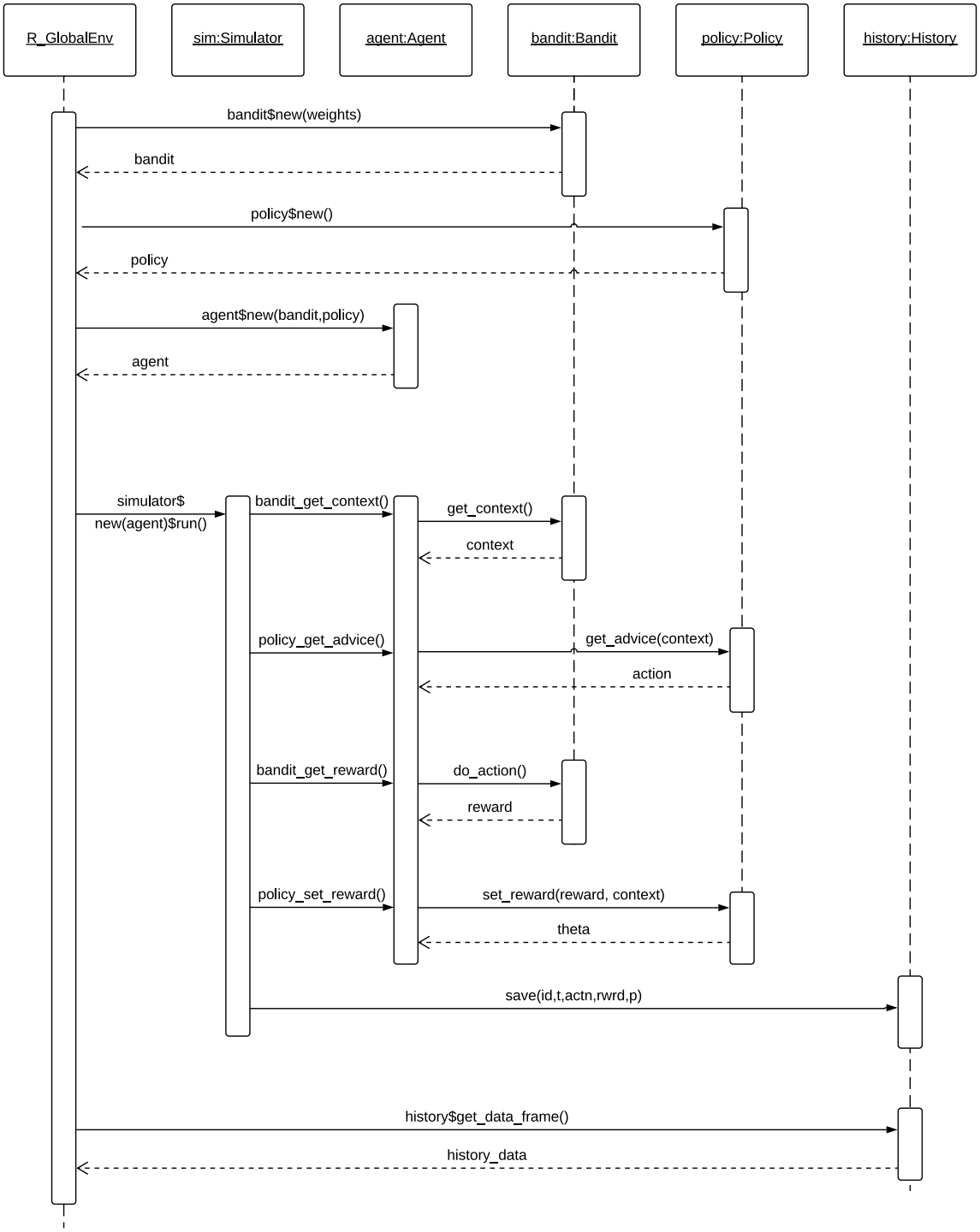

Figure 3: **contextual** UML Sequence Diagram

8. Special features

For instance, quantifying variance..

9. The art of optimal parallelisation

There is a very intersting trade of between the amount of parallelisation (how many cores, nodes used) the resources needed to compute a certain model, and the amount of data going to and fro the cores.

PERFORMANCE DATA ————————————————————————

on 58 cores: k3*d3 * 5 policies * 300 * 10000 –> 132 seconds

on 120 cores: k3*d3 * 5 policies * 300 * 10000 –> 390 seconds

—

on 58 cores: k3*d3 * 5 policies * 3000 * 10000 –> 930 seconds

on 120 cores: k3*d3 * 5 policies * 3000 * 10000 –> 691 seconds

10. Extra greedy UCB

Ladila bladibla.

11. Conclusions

Placeholder... the goal of a data analysis is not only to answer a research question based on data but also to collect findings that support that answer. These findings usually take the form of a table, plot or regression/classification model and are usually presented in articles or reports.

12. Acknowledgments

Thanks go to CCC.

Affiliation:

Robin van Emden
Jheronimus Academy of Data Science
Den Bosch, the Netherlands
E-mail: robin@pwy.nl
URL: pavlov.tech

Eric O. Postma
Tilburg University
Communication and Information Sciences
Tilburg, the Netherlands
E-mail: e.o.postma@tilburguniversity.edu

Maurits C. Kaptein
Tilburg University
Statistics and Research Methods
Tilburg, the Netherlands
E-mail: m.c.kaptein@uvt.nl
URL: www.mauritskaptein.com