contextual: Simulating Contextual Multi-Armed Bandit Problems in R

Robin van Emden
JADS

Eric Postma
Tilburg University

Maurits Kaptein
Tilburg University

Abstract

Contextual multi-armed bandits have been gaining ever more popularity due to their effectiveness in solving previously computationally intractable partial information sequential decision problems - from online advertising and recommender systems to clinical trial design and personalized medicine. A popularity both inspired by and inspiring an ever-growing body of predominantly analytically oriented research on ever more sophisticated Contextual Bandit algorithms. 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, though its clear object oriented structure, easily extensible framework that facilitates the comparison of, amongst others, contextual and non-contextual Bandit policies through both simulation and offline analysis. Furthermore, as the data generated in cMAB settings are often extremely large, much care has been taken to enable easy parallisation right out of the box.

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

1. Introduction

There are many real-world situations in which we repeatedly have to decide between a set of options, yet only learn about the best course of action by testing one choice after the other, one step at a time. Such problems are deceptively easy to state but have proven to have broad statistical and practical implications and applications. To get a better grip on such decision problems, and to learn why specific strategies might be more successful than others, they have been studied extensively under the title of "Multi-Armed Bandit" problems. Here, multi-armed bandits are defined as a statistical and machine learning concept in which a so-called agent follows the advice of an algorithm or "policy" to optimize the overall reward it receives in a sequential decision problem with limited information. That is, a MAB policy suggests an agent when to explore new options and when to exploit known ones – where, importantly, for each decision, at each time step t, the only new information the agent acquires is the reward for its latest decision. The agent remains in the dark about the potential rewards of the unchosen options and about any other information outside of current and past rewards and choices made.

In that respect, MAB problems reflect dilemmas we all encounter on a daily basis: do you stick to what you know and receive an expected result ("exploit") or choose something you don't know all that much about and potentially learn something new ("explore")?

- 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?

Though MAB models have already proven powerful of their own accord, a recent generalization, known as the **contextual** Multi-Armed Bandit (cMAB), adds one important element to the equation: in addition to past decisions and their rewards, cMAB agents are able to make use of side information about the state of the world at each t 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 adept to many real-life decision problems than its MAB progenitors: do you show a certain 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. A practical applicability that has led to a multitude of new, often analytically derived bandit algorithms or policies, each with their own strengths and weaknesses.

Yet though cMAB algorithms have gained much traction in both research and industry, they have mostly been studied mathematically and analytically – as of yet, comparisons on simulated, and, importantly, real-life large-scale offline "partial label" data sets have been lacking. To this end, the current paper introduces the **contextual** R package. A package that aims to facilitate the simulation, offline comparison, and evaluation of (Contextual) Multi-Armed bandit policies. Though there exists one R package for basic MAB analysis, there is, as of yet, no extensible and widely applicable R package that is able to analyze and compare, respectively, basic K-armed, Continuum, Adversarial and Contextual Multi-Armed Bandit Algorithms on either simulated or online data.

In section 2, this paper will continue with a more formal definition of MAB and a 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 polices that are available by default, and simulate some MAB 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

As stated above, a MAB problems can formaly stated with deceptive ease. First, state the number of arms k, and the horizon, or number of rounds T. Then, for each round $t = \{ 1, \ldots, T \}$:

1) An reward vector is generated $r_t = (r_{1,t}, \ldots, r_{k,t})$
2) Agent A chooses an arm at $a_t \in \{1, \ldots, k\}$
3) Agent A receives a reward $r_{a_t,t}$

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

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

On implementing this basic formaliztion, the aforementioned sequential decision maker's exploit/explore dilemma can be captured by defining a finite set (or **bandit)** of K i.i.d. options (the **arms** of the bandit) each with their own, unknown**,** reward distribution $v_1,\ldots,v_k$ with means $\mu \ldots \ \mu_k$. Next we define an **agent**, who has to decide between the exploration of unknown arms and the exploitation of known arms in K in order to maximize its total **reward** (that is, to maximize its cumulative reward $\sum_{t=1}^{T} r_t$ [1]) over a period of time T by following the advice of a **policy π** which keeps track of **parameters θ** that are updated when new information (reward r awarded by the bandit when the agent has chosen an arm) becomes available. This process is repeated T times, where T is often defined as the Bandit's "horizon".

In summary, an agent repeats the following lines one at a time at each time step t in t=1,2,...,T:

1a) Agent asks policy π which of the bandit's K arms to choose
1b) Policy π advices action a_t based on the state of a set of parameters θ_t
2a) Agent does action a_t by playing the suggested bandit arm.
2b) Bandit rewards the agent with reward r_t for action a_t,
3a) Agent sends the reward r_t to policy π
3b) Policy π uses r_t to update the policy's set of parameters θ_t

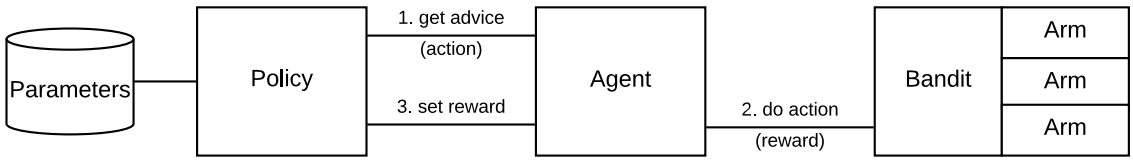

Figure 1: Overview MAB formalization towards **contextual**'s implementation

To allow for side information, that is, to generalize this formalization to a *contextual* Multi-Armed Bandit model, we need to state the number of feature d, and add one first step to our model:

1) A contextual feature vector is observed, $x_t = (x_{1,t}, \ldots, x_{d,t})$

[1] or to minimize its cumulative or expected regret

2) An reward vector is generated $r_t = (r_{1,t}, \ldots, r_{k,t})$

3) Agent A chooses an arm at $a_t \in \{1, \ldots, k\}$

4) Agent A receives a reward $r_{a_t,t}$

From which follows that in our implementation, an agent repeats the following lines for each time step t in t=1,2,...,T:

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 given Xt

Figure 2: Overview of cMAB formalization towards **contextual**'s implementation

As a matter of fact, when excluding a contextual feature, the suggested cMAB model perfectly emulates a non-contextual MAB model, easing the comparison and implementation of both MAB and cMAB substantially.

3. Object-oriented setup of the package

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.

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 3 on page 7.

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")
```

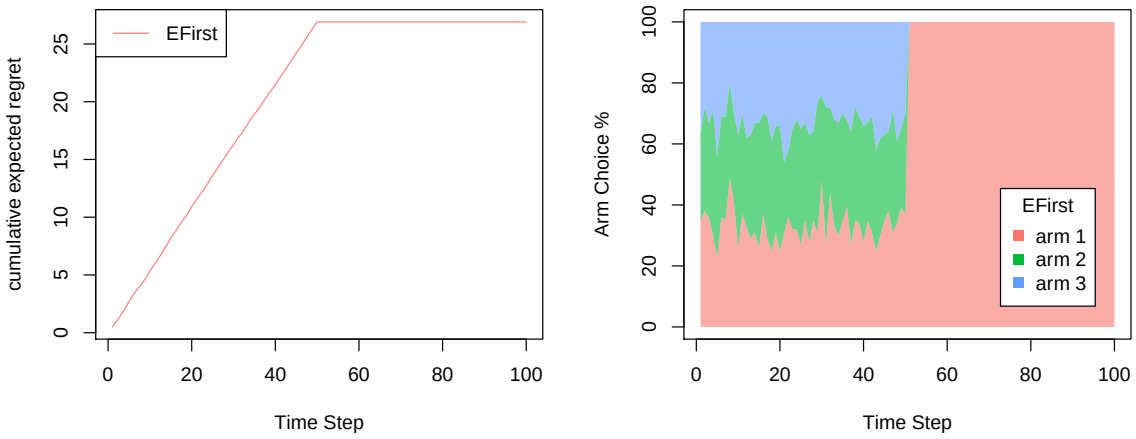

Figure 3: Epsilon First

4.2. Epsilon Greedy

This is an algorithm for continuously balancing exploration with exploitation. (In 'greedy' experi-

ments, the lever with highest known payout is always pulled except when a random action is taken). A randomly chosen arm is pulled a fraction $\epsilon of the time. The other 1-\epsilon of the time, the arm with highest known payout is pulled$

For figures, see Figure 4 on page 9.

The algorithm:

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

Translated to the EpsilonGreedyPolicy class:

```
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")
```

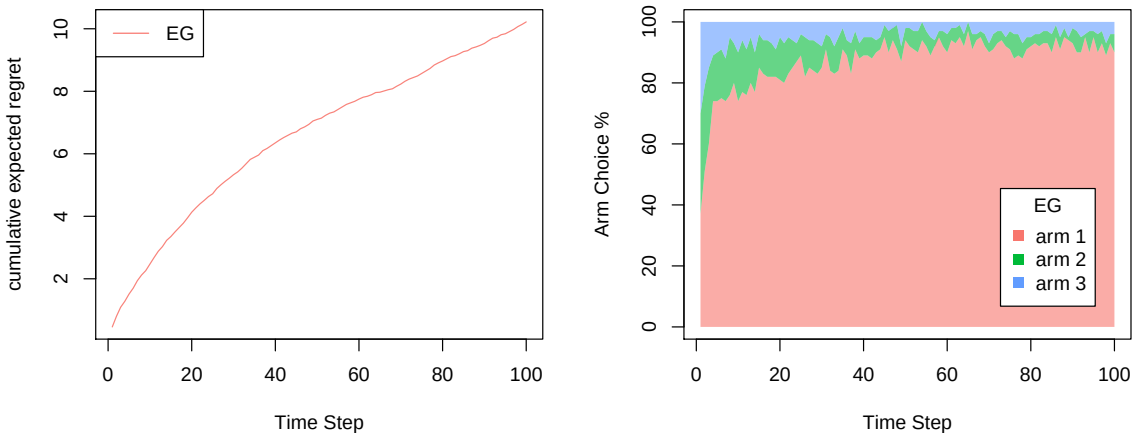

Figure 4: Epsilon Greedy

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 5 on page 12.

```
#' @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)
```

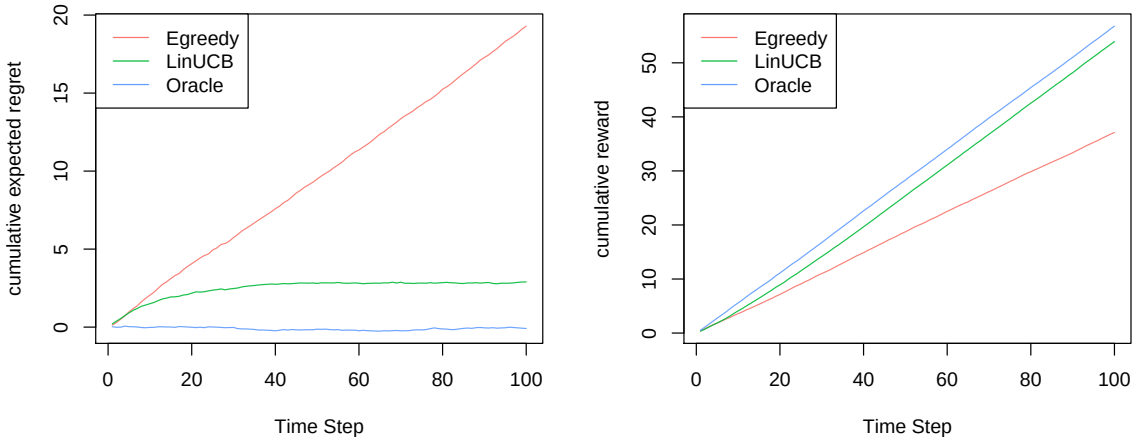

Figure 5: LinUCB algorithm with linear disjoint models, following Li et al. (2010)

5. Extending the package

Here, we show how to extend contextual to implement a basic Contextual Bandit algoriithm.

6. Replications with offline data

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

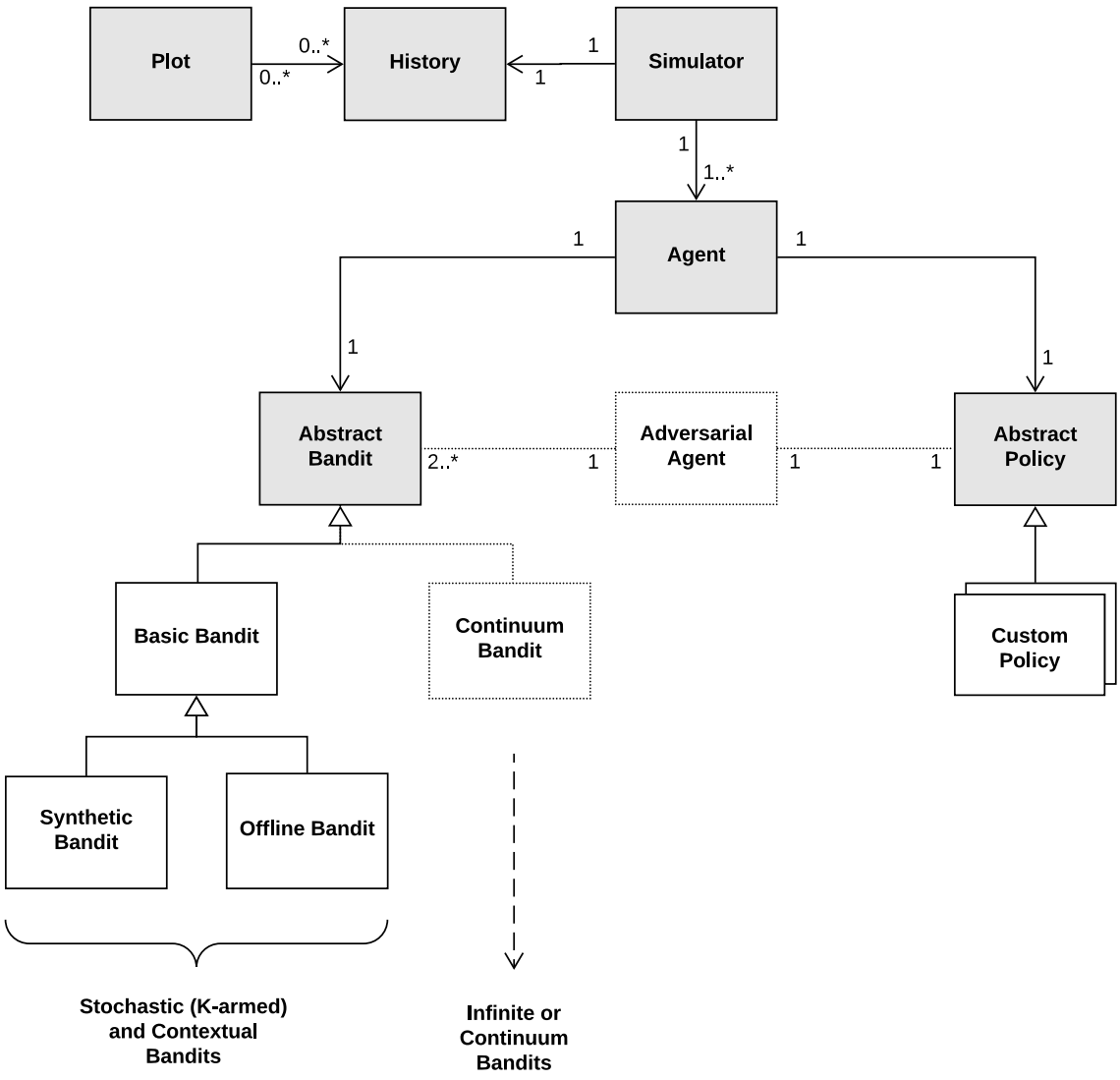

Figure 6: **contextual** UML Class Diagram

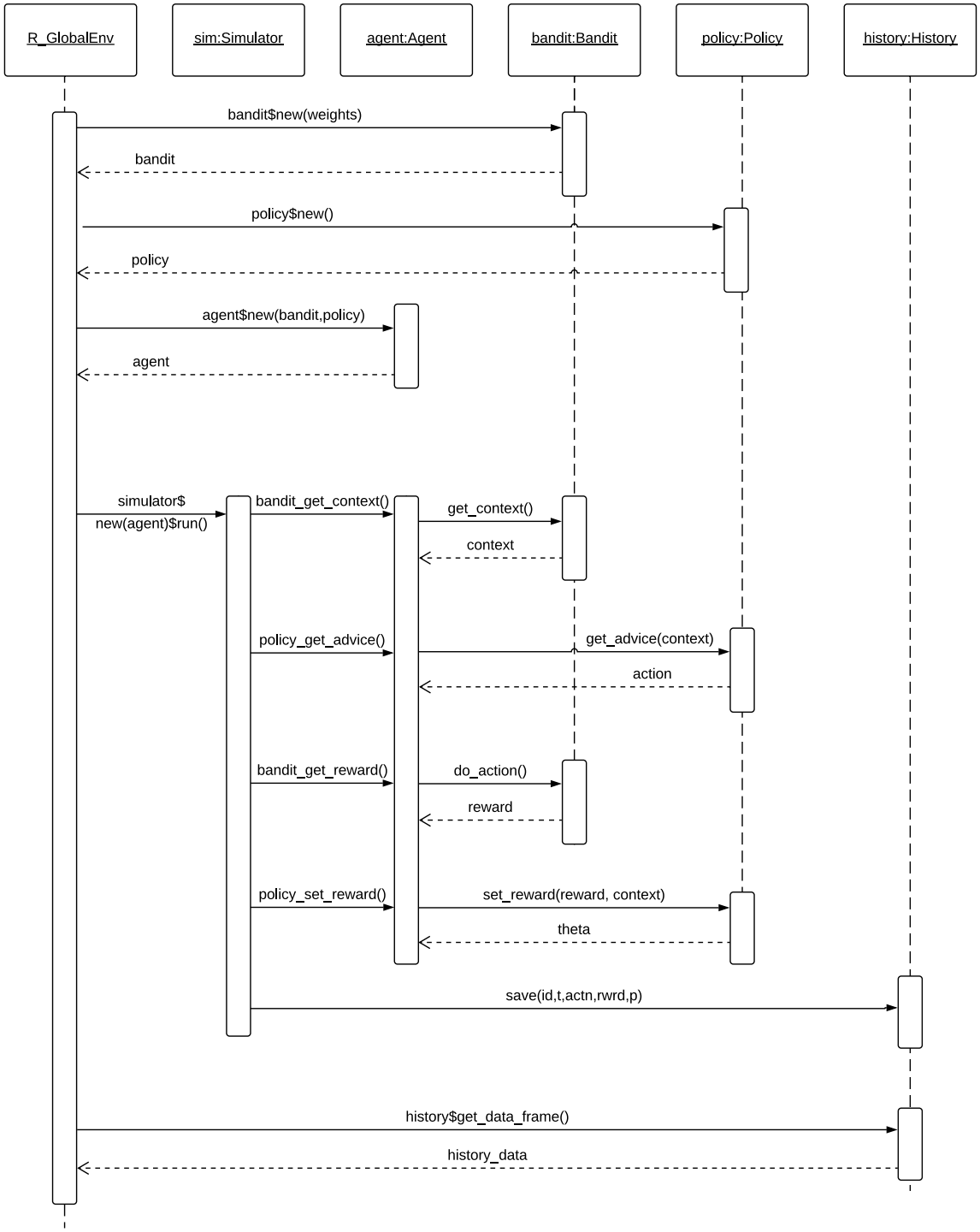

Figure 7: **contextual** UML Sequence Diagram

7. Special features

For instance, quantifying variance..

8. 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

9. Extra greedy UCB

Ladila bladibla.

10. 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.

11. 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