Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

contextual: Simulating Contextual Multi-Armed Bandit Problems in R

Robin van Emden
JADS

Maurits Kaptein
Tilburg University

Abstract

Over the past decade, contextual bandit algorithms have been gaining in popularity due to their effectiveness and flexibility in solving sequential decision problems—from online advertising and finance 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 standardized way. To help close this gap between analytical research and practical evaluation the current paper introduces the object-oriented R package **contextual**: a user-friendly and, through its object-oriented structure, easily extensible framework that facilitates parallelized comparison of contextual and context-free bandit policies through 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 multiple options, yet are only able to learn the best course of action by testing each option sequentially. In such situations, the underlying concept remains the same for each and every renewed decision: Do you stick to what you know and receive an expected result ("exploit") or choose an option you do not know all that much about and potentially learn something new ("explore")? As we all encounter such dilemma's on a daily basis (Wilson, Geana, White, Ludvig, and Cohen 2014), it is easy to come up with examples - for instance:

- When going out to dinner, do you explore new restaurants, or choose a favorite?
- As a website editor, do you place popular or new articles at the top of your frontpage?
- As a doctor, do you prescribe tried and tested medication, or do you also provide promising experimental drugs?

- When visiting a casino, do you stay with the slot machine or "one-armed bandit" that just paid out, or do you try some of the other slot machines?

Although people seem to navigate such explore-exploit problems with relative ease, this type of decision problem has proven surprisingly difficult to solve analytically[1] and has been studied extensively since the 1930s (Bubeck, Cesa-Bianchi *et al.* 2012) under the umbrella of the "multi-armed bandit" (MAB) problem. Just like in the above casino scenario[2], the crux of a multi-armed bandit problem is that you only receive a reward for the arm you pull—you remain in the dark about what rewards the other arms might have offered. Consequently, you need some strategy or "policy" that helps you balance the exploration and exploitation of arms to optimize your rewards over repeated pulls. One option would, for instance, be to pull every available arm once and from then on exploit the arm that offered you the highest reward This reward might, however, be nothing more than a lucky fluke. On the other hand, if you decide to keep exploring other arms, you may lose out on the winnings you might have received from the arm that had been doing so well.

A recent MAB generalization known as the *contextual* multi-armed bandit (CMAB) extends on the previous by adding one crucial element: contextual information (Langford and Zhang 2008). Contextual multi-armed bandits are known by many different names in about as many different fields of research (Tewari and Murphy 2017)—for example as "bandit problems with side observations" (Wang, Kulkarni, and Poor 2005), "bandit problems with side information" (Lu, Pál, and Pál 2010), "associative reinforcement learning" (Kaelbling, Littman, and Moore 1996), "reinforcement learning with immediate reward" (Abe, Biermann, and Long 2003), "associative bandit problems" (Strehl, Mesterharm, Littman, and Hirsh 2006b), or "bandit problems with covariates" (Sarkar 1991). However, the term "contextual multi-armed bandit," as conceived by Langford and Zhang (2008), is the most used—so that is the term we will use in the current paper.

Nevertheless, however named, all CMAB policies differentiate themselves, by definition, from their MAB cousins in that they are able to make use of features that reflect the current state of the world—features that can then be mapped onto available arms or actions[3]. This access to side information makes CMAB algorithms yet more relevant to many real-life decision problems than their MAB progenitors (Langford and Zhang 2008). To follow up on our previous examples: do you choose the same type of restaurants in your hometown and when on vacation? Do you prescribe the same treatment to male and female patients? Do you place the same news story on the frontpage of your website for both young and old visitors? Probably not—in the real world, it appears no choice exists without at least some contextual information to be mined or mapped. So it may be no surprise that CMAB algorithms have found applications in many different areas: from recommendation engines (Lai and Robbins 1985) to advertising (Tang, Rosales, Singh, and Agarwal 2013) and (personalized) medicine (Tewari and Murphy 2017), healthcare Rabbi, Aung, Zhang, and Choudhury (2015), and portfolio choice (Shen, Wang, Jiang, and Zha 2015)—inspiring a multitude of new, often analytically derived bandit algorithms or policies.

However, although CMAB algorithms have found more and more applications, comparisons on both

[1] As Dr. Peter Whittle famously stated "[the problem] was formulated during the [second world] war, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of intellectual sabotage." (Whittle 1979)

[2] Though historically the problem has been defined as the multi-armed bandit problem, it might just as well have been named the "multiple one-armed bandits problem".

[3] That is, before making a choice, the learner receives information on the state of the world or "context" in the form of a d-dimensional feature vector. After making a choice the learner is then able to combine this contextual information with the reward received to make a more informed decision in the next round.

synthetic, and, importantly, real-life, large-scale offline datasets (Li, Chu, Langford, and Wang 2011) have relatively lagged behind [4]. To this end, the current paper introduces the R package **contextual**, which to facilitate the of (contextual) multi-armed bandit policies by offering an easily extensible, class-based, modular architecture.

In that respect, **contextual** differentiates itself from several other types of bandit oriented software applications and services, such as:

1) Online A/B and basic, out-of-the-box MAB test services such as **Google Analytics** (Google 2018), **Optimizely** (Optimizely 2018), **Mix Panel** (Mixpanel 2018), **AB Tasty** (ABTasty 2018), **Adobe Target** (Adobe 2018), and more.

2) More advanced online CMAB test services and software, such as the flexible online evaluation platform **StreamingBandit** (Kaptein and Kruijswijk 2016) and Microsoft's **Custom Decision Service** (Agarwal, Bird, Cozowicz, Hoang, Langford, Lee, Li, Melamed, Oshri, and Ribas 2016).

3) Predominantly context-free simulation oriented projects such as **Yelp MOE** (Yelp 2018), which runs sequential A/B tests using Bayesian optimization, and the mainly MAB focused Python packages **Striatum** (NTUCSIE-CLLab 2018) and **SMPyBandits** (Besson 2018).

4) Software that facilitates the evaluation of bandit policies on offline data, such as **Vowpal Wabbit** (Langford, Li, and Strehl 2007), **Jubatus** (Hido, Tokui, and Oda 2013), and **TensorFlow** (Abadi, Barham, Chen, Chen, Davis, Dean, Devin, Ghemawat, Irving, and Isard 2016).

Though each of these applications and services may share certain features with **contextual**, overall, **contextual** clearly distinguishes itself in several respects. First, it focusses on the evaluation of bandit policies on simulated and offline datasets, which discriminates it from the online evaluation oriented packages cited under items 1 and 2. Second, though **contextual** is perfectly capable of simulating and comparing context-free MAB policies, its emphasis lies on the simulation of contextual policies, distinguishing it from the projects cited under item 3. Finally, though **contextual** is closely related to the projects cited under item 4, it also, again, differentiates itself in several key respects:

a) **contextual** offers a diverse, open and extensible library of common MAB and CMAB policies.

b) **contextual** is developed in R, opening the door to a lively exchange of code, data, and knowledge between scientists and practitioners trained in R.

c) **contextual** focusses on ease of conversion of existing and new algorithms into clean, readable and shareable source code.

d) In building on R's **doParallel** package, **contextual**'s simulations are parallelized by default—and can easily be run on different parallel architectures, from cluster (such as on Microsoft Azure, Amazon ec2 or Hadoop backends) to GPU based.

[4] Here, a **synthetic** data generator (or `Bandit`, in **contextual** parlance) compares policies against some simulated environment, usually seeking to model or emulate some online bandit scenario—whereas an **offline** `Bandit` compares policies against a previous collected data set—generally logged with a completely different policy than the one(s) under evaluation (Li, Chu, Langford, Moon, and Wang 2012).

All in all, though there are some alternatives, there was, as of yet, no extensible and widely applicable R package to analyze and compare, respectively, basic multi-armed, continuum (Agrawal 1995) and contextual multi-armed bandit Algorithms on both simulated and offline data. In making the package publicly available, we hope to further the dissemination and evaluation of both existing and new CMAB policies—particularly where such policies were previously only available through single-use scripts or basic, isolated code packages (Gandrud 2016).

In the current paper, we will further introduce **contextual**. Section 2 presents a formal definition of the contextual multi-armed bandit problem, shows how this formalization can be transformed into a clear and concise object-oriented architecture, and describes how to set up a minimal simulation. Section 3 gives an overview of **contextual**'s predefined `Bandit` and `Policy` subclasses and demonstrates how to run a very basic `Simulation`. Section 4 delves a little deeper into the implementation of each of pkgcontextual's core superclasses. Section 5 shows how to extend **contextual**'s superclasses to create your own custom `Bandit` and `Policy` subclasses. Section 6 focusses on how a `Bandit` subclass can make use of offline datasets. Section 7 brings all of the previous sections together in a partial replication of a frequently cited contextual bandit paper. We conclude with some comments on the current state of the package and potential future enhancements.

This organization offers the reader several ways to peruse the current paper. If you have a passing knowledge of R and want to run simulations based on **contextual**'s default bandits and policies, the current introduction plus section 3 should be able to get you up and running. If you are looking for a more formal introduction, read section 2 as well. Include section 6 and possibly 7 if you also want to run one of **contextual**'s policies on an offline dataset. Finally, if you know your way around R and would like to extend **contextual** to run custom bandits and policies, it is probably best to read the whole paper—with a focus on sections 4 and 5.

2. Formalization and implementation

In the current section, we first introduce a more formal definition of the contextual multi-armed bandit problem. Next, we present our concise implementation, and demonstrate how to put together a minimal MAB simulation.

2.1. Formalization

On further formalization of the contextual bandit problem, a (k-armed) **bandit** can be defined as a set of K distributions, where each distribution is associated with the rewards generated by one of the $k \in \mathbb{N}^+$ arms based on a contextual feature vector $x_t = (x_{1,t}, \ldots, x_{d,t})$. We now define an algorithm or **policy** π, that seeks to maximize its total **reward** (that is, to maximize its cumulative reward $\sum_{t=1}^{T} r_t$ or minimize its cumulative regret—see equations 1, 2). This **policy** observes information on the current state of the world represented in d-dimensional contextual feature vector $x_t = (x_{1,t}, \ldots, x_{d,t})$. Next, the **policy** selects one of the **bandit**'s k arms by choosing an action $k_t \in \{1, \ldots, K\}$, and receives reward $r_{k_t,t}$, which expectation depends both on the context and reward history of that particular arm. With observation $(x_{t,k_t}, k_t, r_{t,k_t})$, the policy now updates its arm-selection strategy. This cycle is then repeated T times, where T is referred to as a bandit's **horizon**.

Additionally, for scalability reasons, **policies** generally use a limited set of parameters θ_t (Kaptein and Kruijswijk 2016). This set of parameters summarizes all historical interactions $D_{t'} = (x_{t,k_t}, k_t, r_{t,k_t})$ over t= {1, . . . , t'}, ensuring that the dimensionality of $\theta_{t'} << D_{t'}$.

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

1) Policy π observes state of the world as contextual feature vector $x_t = (x_{1,t}, \dots, x_{d,t})$
2) Based on x_t and θ_{t-1}, Policy π now selects one of the bandit's arms $k_t \in \{1, \dots, K\}$
3) Policy π receives a reward r_{t,k_t} from the bandit
4) Policy π updates arm-selection strategy parameters θ_t with $(x_{t,k_t}, k_t, r_{t,k_t})$

The goal of the policy π is to optimize its *cumulative reward* over t= { 1, . . . , T }

$$R_T = \sum_{t=1}^{T} (r_{k_t,x_t}) \tag{1}$$

In practice, the most popular performance measure for bandit policies is *cumulative regret* (Kuleshov and Precup 2014)—defined as the sum of rewards that would have been received by choosing optimal arm k at every t subtracted by the sum of rewards awarded to the chosen actions for every t over t= { 1, . . . , T }:

$$\mathbb{E}\left[R_T\right] = \mathbb{E}\left[\max_{\mathrm{k}=1,\dots,K} \sum_{t=1}^{T} (r_{\mathrm{k},x_t}) - \sum_{t=1}^{T} (r_{k_t,x_t})\right] \tag{2}$$

Where expectation $\mathbb{E}[\cdot]$ is taken with respect to random draws of both rewards assigned by a bandit and arms as selected by a policy (Zheng and Hua 2016).

2.2. Implementation: core classes

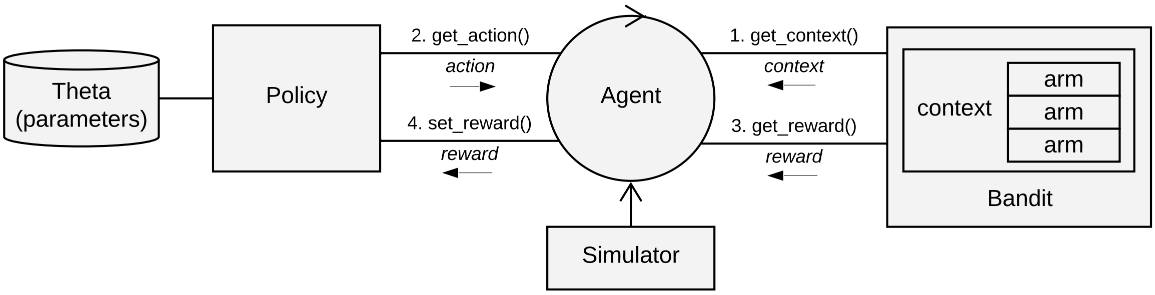


Figure 1: Diagram of **contextual**'s basic structure. The context feature matrix returned by get_context() is only taken into account by CMAB policies, and may be ignored by MAB policies.

Our class structure builds on the previous formalisation and offers a clean class structure with an easily programmable interface. The following six classes form the backbone of the package (see also Figure 1):

- `Bandit`: R6 class `Bandit` is the parent class of all **contextual** `Bandit` subclasses. It is responsible for the generation of `contexts` and `rewards`.
- `Policy`: R6 class `Policy` is the parent class of all **contextual**'s `Policy` implementations. For each $t = \{1, \dots, \mathrm{T}\}$ it has to choose one of a `Bandit`'s `k` arms, and update its parameters `theta` in response to the resulting reward.

- `Agent`: R6 class `Agent` is responsible for the running of one `Bandit/Policy` pair. As such, multiple `Agents` can be run in parallel with each separate Agent keeping track of `t` for its assigned `Policy` and `Bandit` pair. To be able to fairly evaluate and compare each agent's performance, and to make sure that simulations are replicable, seeds are set equally and deterministically for each agent over all `horizon x simulations` time steps of each agent's simulation.
- `Simulator`: R6 class `Simulator` is the entry point of any **contextual** simulation. It encapsulates one or more `Agents` (running in parallel to each other, by default), creates a clone (each with its own deterministic seed) for each to be repeated simulation, runs the `Agents`, and saves the log of all `Agent's` interactions to a `History` object.
- `History`: R6 class `History` keeps a log of all `Simulator` interactions. It allows several ways to interact with the data, provides summaries of the data, and can save and load simulation data in several different (`data.table`, `data.frame` and CSV) formats.
- `Plot`: R6 class `Plot` generates plots from `History` logs. It is usually invoked by calling the generic `plot(h)` function, where `h` is an `History` class instance.

2.3. Putting it together: a first MAB simulation

Setting up a simulation

The six building blocks introduced in the previous section are essentially all that is needed to put together, for example, the following five line context-free MAB simulation:

```
library(contextual)

policy    <- ThompsonSamplingPolicy$new()
bandit    <- BasicBernoulliBandit$new(p_per_arm = c(0.9, 0.1, 0.1))
agent     <- Agent$new(policy,bandit)
simulator <- Simulator$new(agents = agent, simulations = 100, horizon = 50)
history   <- simulator$run()
```

Here we start out by instantiating `Policy` subclass `EpsilonGreedyPolicy` (covered in section 5.3) as object `policy`, with its parameter `epsilon` set to `0.1`. Next, we instantiate the `Bandit` subclass `BasicBernoulliBandit` as `bandit`, with three Bernoulli arms, each offering a reward of one with probability p, and otherwise a reward of zero. For the current simulation, we have set the `bandit`'s probability of reward to respectively 0.9, 0.1 and 0.1 per arm through a `weight` parameter. We then assign both our `bandit` and our `policy` to `Agent` instance `agent`. This `agent` is added to a `Simulator` that is set to one hundred `simulations`, each with a `horizon` of fifty—that is, `simulator` runs one hundred `simulations`, each with a different (deterministic) random seed, for fifty time steps `t`.

Running the simulation

On running the `Simulator` it starts several (by default, the number of CPU cores minus one) worker processes, splitting simulations as efficiently as possible over each parallel worker. For each simulation, for every time step `t`, `agents` runs through each of the four function calls that constitute their main loop.

A main loop that, not accidentally, relates one on one to the four steps defined in our CMAB formalization from section 2.1:

1) agent calls bandit$get_context(t). The bandit returns a named list that contains the current d x k dimensional feature matrix context$X, the number of arms context$k and the number of features per arm context$d.

2) agent calls policy$get_action(t, context). The policy computes which arm to play based on the current values in named lists theta and context. Returns a named list containing action$choice, which holds the index of the arm to play.

3) agent calls bandit$get_reward(t, context, action). The bandit returns a named list containing the reward for the action chosen in [2] and, optionally, an optimal_reward—when computable.

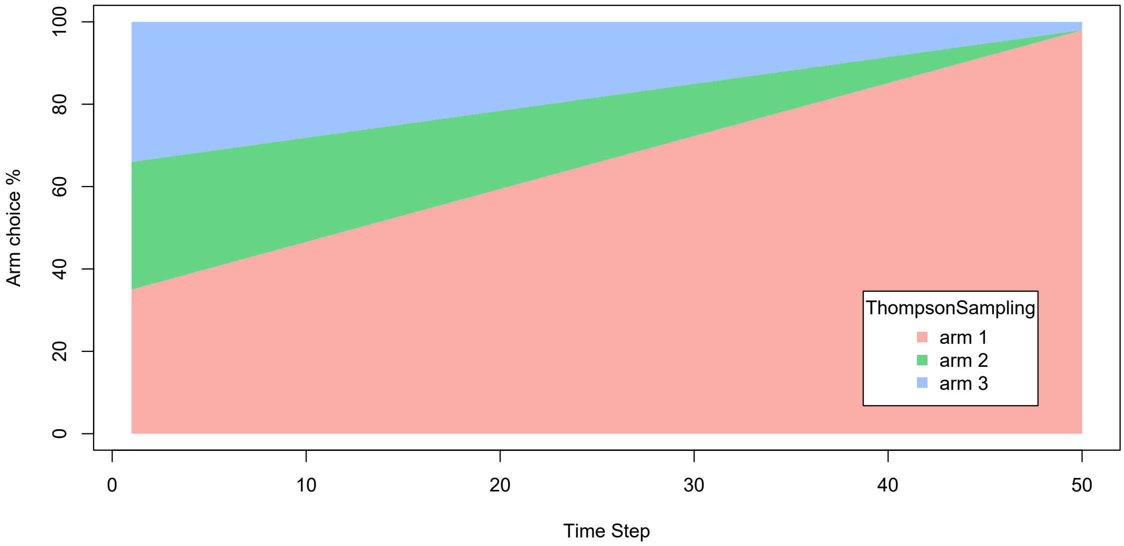
4) agent calls policy$set_reward(t, context, action, reward). The policy uses the action taken, the reward received, and the current context to update its set of parameter values in theta.

Results of the simulation

On completion of all of its agents' simulation runs, Simulator returns a history object which contains a complete log of all interactions. This history log can then, for example, be summarized and plotted:

```
> summary(history)

Agents:

  ThompsonSampling

Cumulative regret:

            agent  t sims cum_regret cum_regret_var cum_regret_sd
 ThompsonSampling 50  100       4.84       4.519596      2.125934


Cumulative reward:

            agent  t sims cum_reward cum_reward_var cum_reward_sd
 ThompsonSampling 50  100      40.68       9.452121       3.07443


Cumulative reward rate:

            agent  t sims cur_reward cur_reward_var cur_reward_sd
 ThompsonSampling 50  100     0.8136      0.1890424     0.0614886
```

```
> plot(history, type = "arms", interval = 50)
```



3. Basic Usage

The current section offers an overview of **contextual**'s predefined bandits and policies and further demonstrates how to run them.

3.1. Implemented Policies and Bandits

Though contextual is at its most useful when developing and evaluating custom bandit and policy classes, as demonstrated in the previous section, it is also possible to run basic simulations with just its built-in bandits and policies. See Table 1 for an overview of all available policies and Table 2 for an overview of all currently implemented bandits—where possible with references to their original papers.

	ϵ-Greedy	**UCB**	**Thomspon Sampling**	**Other**	**Special**
MAB	ϵ-Greedy[1] ϵ-First	UCB1[2] UCB-tuned[3]	Thompson Sampling[4] BootstrapTS[5]	Softmax[6] Gittins[7]	Random Oracle LiF[8]
CMAB	Epoch-Greedy[9]	LinUCB[10] COFIBA[11]	LinTS[12] LogitBTS[13]		

Table 1: An overview of **contextual**'s predefined policy classes.

MAB	**CMAB**	**Offline**	**Continuous**
BasicBernoulliBandit BasicGaussianBandit	ContextualBernoulli ContextualLogit ContextualHybrid ContextualLinear ContextualWheel[14]	OfflinePolicyEvaluator[15] DoublyRobust[16]	ContinuumBandit

Table 2: An overview of **contextual**'s predefined bandit classes.

[1] Sutton and Barto (1998) [2] Auer *et al.* (2002) [3] Auer *et al.* (2002) [4] Agrawal and Goyal (2011) [5] Eckles and Kaptein (2014) [6] Vermorel and Mohri (2005) [7] Brezzi and Lai (2002) [8] Kaptein *et al.* (2016) [9] Langford and Zhang (2008) [10] Li *et al.* (2010) [11] Li (2016) [12] Agrawal and Goyal (2012) [13] Eckles and Kaptein (2014) [14] Riquelme *et al.* (2018) [15] Li *et al.* (2011) [16] Dudík *et al.* (2011)

3.2. Running basic simulations

In the current subsection we demonstrate how run simulations with **contextual**'s predefined bandit and policy classes on the basis of a familiar bandit scenario.

The scenario

As online advertising is one of the areas where bandit policies have found widespread application, we will use it as a backdrop to our basic bandit tutorial. Generally, the goal in online advertising is to determine which out of several ads to serve a visitor to a particular web page. Translated to a bandit setting, in online advertising:

- The context is usually determined by visitor and web page characteristics.
- Arms are represented by the pool of available ads.
- An action is the displayed add as chosen by a policy.
- The reward is a visitor's click (a reward of 1) or non-click (a reward of 0) on an ad.

In the case of the simulations in the current section, we limit the number of advertisements in the pool to 3, and set ourselves the goal of finding the policy that offers the highest overall click-through rate within 400 impressions—with the higher level goal of choosing the best policy to implement on a live website.

Comparing context-free policies

Before we are able to evaluate any policies, we first have to model the three ads—with each a different probability of generating a click—as the arms of a bandit. We choose to model the ads with the weight-based `SyntheticBandit` as this enabled us to easily set uniformly distributed weights determining the probability of the rewards for each arm. As can be observed in the source code below, for the current simulation, we set the weights of the arms to respectively $p = 0.8$, $p = 0.4$ and $p = 0.2$.

We also choose two context-free policies to evaluate and compare:

- EpsilonFirstPolicy: explores all three ads at uniformly at random for a preset period and from thereon exploits the ad with the best click-through rate [5]. For our current scenario, we set the exploration period to 50 impressions. A formal definition and an implementation of the algorithm can be found in section 5.2.

- EpsilonGreedyPolicy: explores one of the ads at uniformly at random ϵ of the time and exploits the ad with the best current click-through rate $1-\epsilon$ of the time. For our current scenario, we set $\epsilon = 0.2$. For a formal definition and implementation see section 5.3.

Next, we assign the one bandit instance and two policy instances to two agents, ensuring that both policies are run on the same set of generated data. Finally, we assign a list holding both agents to a Simulator instance, set the simulator's horizon to 400 and the number of repeats to 1000, run the simulation, and plot() its results:

```
# Load and attach the contextual package.
library(contextual)
# Define for how long the simulation will run.
horizon            <- 400
# Define how many times to repeat the simulation.
simulations        <- 1000
# Define the probability that each ad will be clicked.
click_probabilities <- c(0.8, 0.4, 0.2)
# Initialize a SyntheticBandit, which takes probabilites per arm for an argument.
bandit             <- SyntheticBandit$new(weights = click_probabilities)
# Initialize an EpsilonGreedyPolicy with a 20% exploiration rate.
eg_policy          <- EpsilonGreedyPolicy$new(epsilon = 0.2)
# Initialize an EpsilonFirstPolicy with a 50 step exploration period.
ef_policy          <- EpsilonFirstPolicy$new(first = 50)
# Initialize two Agents, binding each policy to a bandit.
ef_agent           <- Agent$new(ef_policy, bandit)
eg_agent           <- Agent$new(eg_policy, bandit)
# Assign both agents to a list.
agents             <- list(ef_agent, eg_agent)
# Initialize a Simulator with the agent list, horizon, and number of simulations.
simulator          <- Simulator$new(agents, horizon, simulations)
# Now run the simulator.
history            <- simulator$run()
# And plot the cumulative reward rate (equals Click Through Rate)
plot(history, type = "cumulative", regret = FALSE, rate = TRUE)
```

[5] A type of policy also known as an A/B test Kohavi, Henne, and Sommerfield (2007).

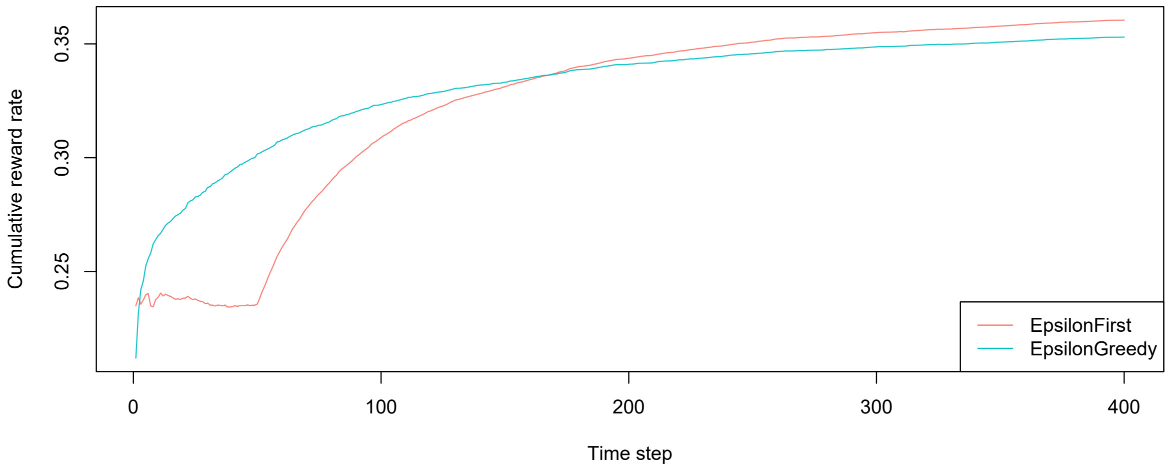


Figure 2: Cumulative reward rate (equals click-through rate) of a context-free bandit scenario comparing ϵ-first and ϵ-greedy policies.

As can be observed in Figure 3.2.2, within our horizon of $T = 400$, `EpsilonFirstPolicy` has accumulated more rewards than `EpsilonGreedytPolicy`. It is easy to see why: The winning arm is consistently the best - by a margin. So `EpsilonFirstPolicy` has no difficulty in finding the optimal arm within its exploration period of 50 impressions. Up to that point, `EpsilonGreedyPolicy` had the advantage of a headstart—from the moment it established which arm was best, it was able to exploit it for $1 - \epsilon$ or 80 percent of the time. However, in contrast to `EpsilonFirstPolicy`, it is not able to exploit it consistently from 50 impressions on—the `EpsilonGreedyPolicy` is obliged to keep exploring lesser arms for 20 percent of the time.

Adding context

If that is all we know of our visitors, we expect the results to be stationary over time, and these are the only policies available, the choice is clear: we will use `EpsilonFirstPolicy` on our live website[6]. However, if we have contextual information on our visitors—for instance, their age—we might be able to do better. Let us suggest, for instance, that we expect that some of our ads are more effective for older visitors, and other ads more effective for younger visitors.

To incorporate this expectation in our simulation, we need to change the way our bandit generates its rewards. Fortunately, in the case of our `SyntheticBandit`, the introduction of two contextual features only requires the addition of a single row to its weight matrix—as `SyntheticBandit` parses each of the d rows of its weight matrix as a binary feature generated with a probability of $p_{d,t} = 1/d$.

As can be seen in the source code below, in choosing our weights, we have taken care to keep the average rewards per arm over the two features equal to the rewards per arm in the previous simulation. So we do not expect a substantial difference with the previous simulation's outcome for context-free policies `EpsilonFirstPolicy` and `EpsilonGreedyPolicy`.

We therefore also include the contextual `LinUCBDisjointPolicy` (Li *et al.* 2010), which, in assuming its reward function is a linear function of the context, is able to incorporate our new contextual information into its decision-making process. See 5.4 for a detailed description and implementation details of this policy. Now let us rerun the simulation:

[6] Also: if our bandit represents our visitors' click behavior realistically, if our policies' parameters are optimal, etcetera.

```
#                                    +-----+----+-----------> ads: k = 3
#                                    |     |    |
click_probabilities <- matrix(  c(  0.2,  0.3, 0.1,     # --> d1: old   (p=.5)
                                    0.6,  0.1, 0.1   ), # --> d2: young (p=.5)
                                                        #     features: d = 2

                                  nrow = 2, ncol = 3, byrow = TRUE)

# Initialize a SyntheticBandit with contextual weights
context_bandit      <- SyntheticBandit$new(weights = click_probabilities)
# Initialize LinUCBDisjointPolicy
lu_policy           <- LinUCBDisjointPolicy$new(0.6)
# Initialize three Agents, binding each policy to a bandit.
ef_agent            <- Agent$new(ef_policy, context_bandit)
eg_agent            <- Agent$new(eg_policy, context_bandit)
lu_agent            <- Agent$new(lu_policy, context_bandit)
# Assign all agents to a list.
agents              <- list(ef_agent, eg_agent, lu_agent)
# Initialize a Simulator with the agent list, horizon, and number of simulations.
simulator           <- Simulator$new(agents, horizon, simulations)
# Now run the simulator.
history             <- simulator$run()
# And plot the cumulative reward rate again.
plot(history, type = "cumulative", regret = FALSE, rate = TRUE)
```

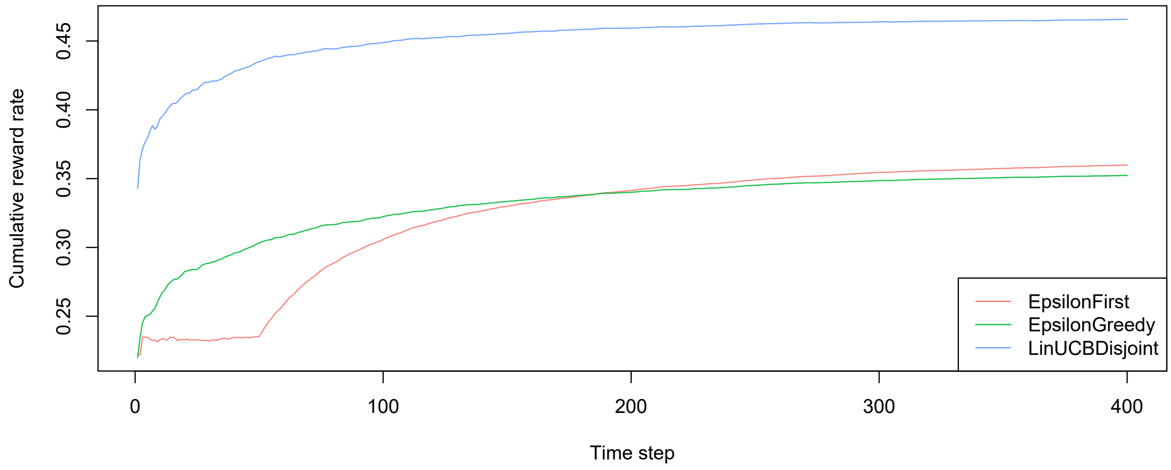


Figure 3: Cumulative reward rate (equals click-through rate) of a contextual bandit scenario comparing LinUCB, ϵ-first and ϵ-greedy policies.

As can be observed in Figure 3.2.3, both context-free bandit's results do indeed about as well as before. On the other hand, `LinUCBDisjointPolicy` does very well, as this contextual policy is able to map its rewards to the observed contextual features. All in all, this time around, you would probably choose the `LinUCBDisjointPolicy` for your website—with the added advantage of being able to incorporate other contextual features to optimize results further.

Of course, neither simulation in the current section is very realistic. One way to do better would be to write a Bandit subclass with a more complex generative model. How to create such a bandit subclass is delineated in section 5. Another option would, for example, be to evaluate policies on an offline dataset—more on that in section 6.

4. Class structure

Since it is **contextual**'s explicit goal to offer researchers and developers an easily extensible framework to develop, evaluate and compare their own `Policy` and `Bandit` implementations, the current section offers additional background information on **contextual**'s class structure—both on the R6 class system Chang (2017) and on each of the six previously introduced core **contextual** classes.

4.1. Choice for the R6 Class System

Though widely used as a procedural language, R offers several Object Oriented (OO) systems, which can significantly help in structuring the development of more complex packages. Out of the OO systems available (S3, S4, R5 and R6), we settled on R6, as it offered several advantages compared to the other options. Firstly, it implements a mature object-oriented [7] design when compared to S3. Secondly, its classes can be accessed and modified by reference—which offers the added advantage that R6 classes are instantly recognizable for developers with a background in programming languages such as Java or C++. Finally, when compared to the older R5 reference class system, R6 classes are lighter-weight and (as they do not make use of S4 classes) do not require the methods package—which makes **contextual** substantially less resource-hungry than it would otherwise have been.

4.2. Main Classes

In this section, we go over each of **contextual**'s six main classes in some more detail—with an emphasis on the `Bandit` and `Policy` classes. To clarify **contextual**'s class structure, we also include two UML diagrams (UML, or "unified modeling language" presents a standardized way to visualize the overall class structure and general design of a software application or framework (Rumbaugh, Jacobson, and Booch 2004)). The UML class diagram shown in Figure 7 on page 43 visualizes **contextual**'s static object model, showing how its classes inherit from, and interface with, each other. The UML sequence diagram in figure Figure 8 on page 44, on the other hand, illustrates how **contextual**'s classes interact dynamically over time.

Bandit

In **contextual**, any bandit implementation is expected to subclass and extend the `Bandit` superclass. It is then up to these subclasses themselves to provide an implementation for each of its abstract methods. In practice, this entials that `Bandit` subclasses are, firstly, responsible for setting the instance variable `self$k` to the number of arms, and instance variable `self$d` to the number of context features. Next, any `Bandit` siblings are expected to implement at least `get_context()` and `do_action()`:

```
#' @export
```

[7] In object-oriented programming, the developer compartmentalizes data into objects, whose behavior and contents are described through the declaration of classes. Its benefits include reusability, refactoring, extensibility, ease of maintenance and efficiency. See, for instance, Wirfs-Brock, Wilkerson, and Wiener (1990) for a general introduction to the princples of Object Oriented software design, and Wickham (2014) for more information of the use of OOP in R.

```
Bandit <- R6::R6Class(
  portable = TRUE,
  class    = FALSE,
  public   = list(
    k           = NULL,     # Number of arms (integer)
    d           = NULL,     # Dimension of context feature vectors (integer)
    ...
    precaching  = FALSE,    # Pregenerate context & reward matrices? (boolean)
    class_name  = "Bandit", # Bandit name - required (character)
    initialize  = function() {
      # Initialize Bandit. Set self$d and self$k here.
    },
    ...
    get_context = function(t) {
      stop("Bandit subclass needs to implement bandit$get_context()", call. = FALSE)
      # Return a list with self$k, self$d and, where applicable, a context matrix X.
      list(X = context, k = arms, d = features)
    },
    get_reward = function(t, context, action) {
      stop("Bandit subclass needs to implement bandit$get_reward()", call. = FALSE)
      # Return a list with the reward and, if known, the reward of the best arm.
      list(reward = reward_for_choice_made, optimal = optimal_reward_value)
    },
    ...
  )
)
```

The main `Bandit` functions can be futher detailed as follows:

- `new()` Generates and initializes a new `Bandit` object.
- `pre_calculate()` Called right after `Simulator` sets its seed, but before it starts iterating over all time steps `t` in T. If you need to initialize random values in a `Policy`, this is the place to do so.
- `get_context(t)` Returns a named list `list(k = n_arms, d = n_features, X = context)` with the current `d x k` dimensional `context` feature matrix `X` together with the number of arms `k`.
- `get_reward(t, context, action)` Returns the named list `list(reward = reward_for_choice_made, optimal = optimal_reward_value)` containing the `reward` for the `action` previously returned by `policy` and, optionally, the `optimal` reward at the current time `t`.
- `generate_bandit_data()` A helper function that is called before `Simulator` starts iterating over all time steps `t` in T. This function is called when `bandit$precaching` has been set to `TRUE`. Pregenerate `contexts` and `rewards` here.

As already previously indicated in Table 2 from Section 3.1 Bandit, **contextual** already contains several predefined Bandits. For each of these Bandits, the package offers at least one example script, to be found in the package's demo directory:

- `BasicBandit`: this basic (context-free) k-armed bandit synthetically generates rewards based on a weight vector. It returns a unit vector for context matrix X.
- `SyntheticBandit`: an example of a more complex and versatile synthetic bandit. It pregenerates both a randomized context matrix and reward vectors
- `ContextualBandit`: a contextual bandit that synthetically generates contextual rewards based on randomly set weights. It can simulate mixed user (cross-arm) and article (arm) feature vectors, generated from parameters k, d and num_users.
- `ContinuumBandit`: a basic example of a continuum bandit.
- `LiBandit`: a basic example of a bandit that makes use of offline data - here, an implementation of Li's [reference].

Each of these bandits can be deployed to run policies without further ado. They can, however, also be used as either examples or templates for your own custom `Bandit` implementation(s), or as superclasses for sub-subclass implementations.

Policy

`Policy` is the second often subclassed contexual superclass. Just like the `Bandit` superclass, `Policy` is an abstract class that declares methods without itself offering an implementation. Any `Policy` subclass is therefore expected to implement `get_action()` and `set_reward()`. Also, any parameters that keep track or summarize `context`, `action` and `reward` values are required to be saved to `Policy`'s *named list* `theta`.

```
#' @export
Policy <- R6::R6Class(
  portable = FALSE,
  class = FALSE,
  public = list(
    action        = NULL,      # action results (list)
    theta         = NULL,      # policy parameters theta (list)
    theta_to_arms = NULL,      # theta to arms "helper" (list)
    class_name    = "Policy",  # policy name - required (character)
    initialize = function() {
      self$theta  <- list()    # initializes theta list
      self$action <- list()    # initializes action list
    },
    get_action = function(t, context) {
      # Selects an arm based on self$theta and context, returns it in action$choice
      stop("Policy$get_action() has not been implemented.", call. = FALSE)
    },
    set_reward = function(t, context, action, reward) {
      # Updates parameters in theta based on reward awarded by bandit
      stop("Policy$set_reward() has not been implemented.", call. = FALSE)
    },
    set_parameters = function(context_params) {
      # Policy parameter (not theta!) initialisation happens here
      stop("Policy$set_parameters() has not been implemented.", call. = FALSE)
    },
    initialize_theta = function(k) {
      # Called during contextual's initialisation.
      # Copies theta_to_arms k times, makes the copies available through theta.
      ...
    }
  )
)
```

Policy's main functions can be futher detailed as follows:

- set_parameters() This helper function, called during a Policy's initialisation, assigns the values it finds in list self$theta_to_arms to each of the Policy's k arms. The parameters defined here can then be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name. The source code in section 5.2 illustrates the mechanism further.
- get_action(t, context) Calculates which arm to play based on the current values in named list theta and the current context. Returns a named list list(choice = arm_chosen_by_policy) that holds the index of the arm to play.
- set_reward(t, context, action, reward) Returns the named list list(reward = reward_for_choice_made, optimal = optimal_reward_value) containing the reward for the action previously returned by policy and, optionally, the optimal reward at the current time t.

Agent

To ease the encapsulation of parallel Bandit and Policy simulations, Agent is responsibe for the flow of information between and the running of one Bandit and Policy pair, for example:

```
policy             <- EpsilonGreedyPolicy$new(epsilon = 0.1, name = "EG")
bandit             <- SyntheticBandit$new(weights = c(0.9, 0.1, 0.1))
agent              <- Agent$new(policy,bandit)
```

It does this by keeping track of t through its private named list variable state and by making sure that, at each time step t, all four main Bandit and Policy CMAB methods are called in correct order, one after the other:

```
Agent <- R6::R6Class(
  public = list(
    #...
    do_step = function() {
      t <- t + 1
      context = bandit$get_context(t)
      action  = policy$get_action (t, context)
      reward  = bandit$get_reward (t, context, action)
      theta   = policy$set_reward (t, context, action, reward)
      list(context = context, action = action, reward = reward,theta = theta)
    }
    #...
  )
)
```

Its main function is `do_step()`, generally called by a `Simulator` object (or, more specifically, by the `Simulator`-started parallel worker that is repsonsible for this particular `Agent`):

- `do_step()` Completes one time step `t` by consecutively calling `bandit$get_context()`, `policy$get_action()`, `bandit$get_reward()` and `policy$set_reward()`.

Simulator

A `Policy` instance is the entry point of any **contextual** simulation. It encapsulates one or more `Agents`, clones them if necessary, runs the `Agents` (in parallel, by default), and saves the log of all of the `Agents` interactions to a `History` object:

```
history <- Simulator$new(agents = agent, horizon = 10, simulations = 10)$run()
```

By default, for performance reasons, a `Simulator` does not save `context` matrices and the (potentially deeply nested) `theta` list to its `History log`—though this can be changed by setting either `save_context` and `save_theta` arguments set to `TRUE`.

To specify how to run a simulation and which data is to be saved to a `Simulator` instance's `History` log, a `Policy` object can be configured through the following parameters:

- `agents` An `Agent` instance, or a `list` of `Agent` instances to be run by the instantiated `Simulator`.
- `horizon` The T time steps to run the instantiated `Simulator`.
- `simulations` How many times to repeat each agent's simulation with a new seed on each repeat (itself deterministically derived from set_seed).
- `save_context` Save context matrices `X` to the `History` log during a simulation?
- `save_theta` Save the parameter list `theta` to the `History` log during a simulation?
- `do_parallel` Run `Simulator` processes in parallel?

- worker_max Specifies how many parallel workers are to be used, when do_parallel is TRUE. If unspecified, the amount of workers defaults to max(workers_available)-1.
- continuous_counter Of use to offline Bandits. If continuous_counter is set to TRUE, the current Simulator iterates over all rows in a dataset for each repeated simulation. If FALSE, it splits the data into simulations parts, and a different subset of the data for each repeat of an agent's simulation.
- set_seed Sets the seed of R's random number generator for the current Simulator.
- write_progress_file If TRUE, Simulator writes progress.log and doparallel.log files to the current working directory, allowing you to keep track of workers, iterations, and potential errors when running a Simulator in parallel.
- include_packages List of packages that (one of) the policies depend on. If a Policy requires an R package to be loaded, this option can be used to load that package on each of the workers. Ignored if do_parallel is FALSE.
- reindex If TRUE, removes empty rows from the History log, re-indexes the t column, and truncates the resulting data to the shortest simulation grouped by agent and simulation.

History

A Simulator aggregates the data acquired during a simulation in a History object's private data.table log. It also calculates per agent average cumulative reward, and, when the optimal outcome per t is known, per agent average cumulative regret. It is furthermore possible to plot() a History object, summarize() it, or obtain, for example, a data.frame() or a data.table() from any History instance:

```
history             <- Simulator$new(agent)$run()
dt                  <- history$get_data_table()
df                  <- history$get_data_frame()
cumulative_regret   <- history$cumulative(regret = TRUE)
```

Some other History functions:

- set(index, t, action, reward, policy_name, simulation_index, context_value = NA, theta_value = NA) Stores one row of simulation data. Generally not called directly, but rather through a Simulator instance.
- save(filename = NA) Writes History to a file with name filename.
- load(filename, interval = 0) Reads a History log file with name filename. If interval is larger than 0, every other interval row of data is read instead of the full data file.
- reindex(truncate = TRUE) Removes empty rows from the History log, reindexes the t column, and, when truncate is TRUE, truncates the resulting data to the number of rows of the shortest simulation.

Plot

The Plot class takes an History object and offers several ways to plot it, each optimized to be able to plot gigabytes worth of data, quickly:

- average: plots the average reward or regret over all simulations per Agent (that is, each Bandit and Policy combo) over time.
- cumulative: plots the average reward or regret over all simulations per Agent over time.
- arms: plots ratio of arms chosen on average at each time step, in percentages, totaling 100

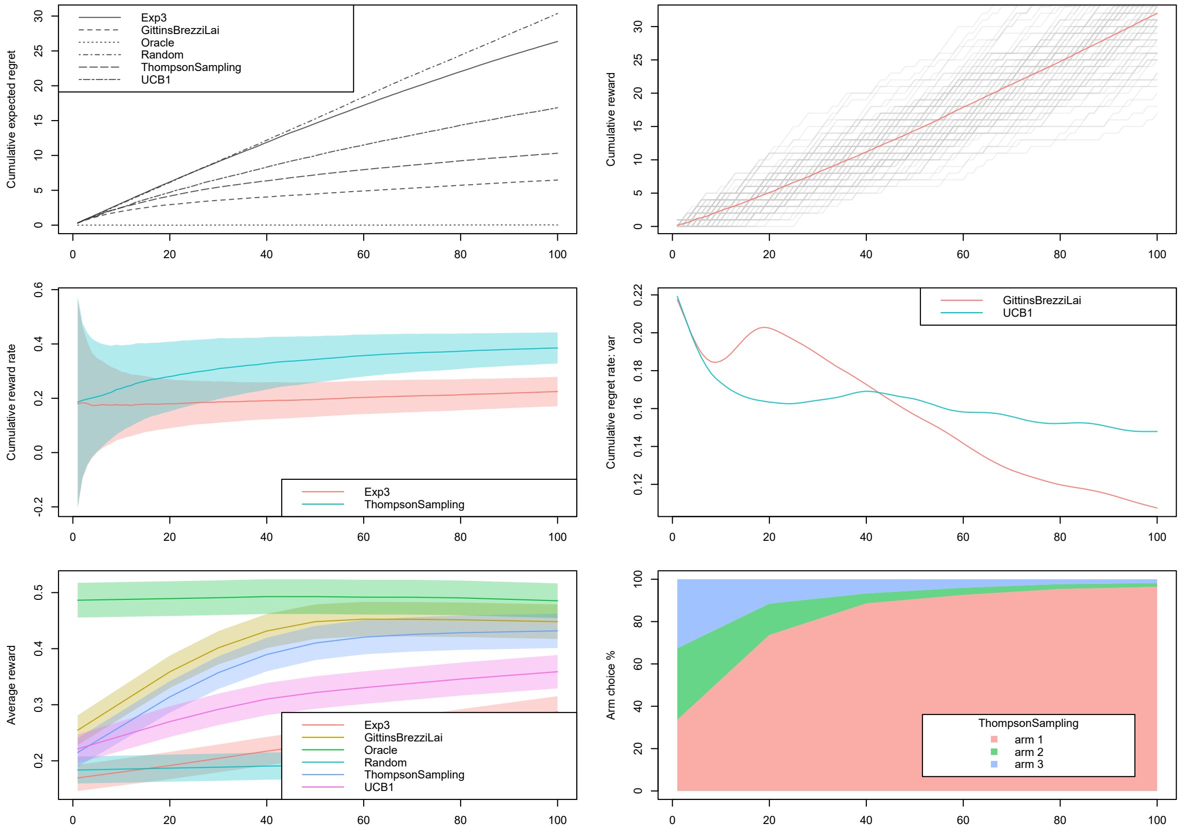
Plot objects can be instantiated directly, or, more commonly, by calling the plot() function. In either case, make sure to specify a History instance and one of the plot types specified above:

```
# plot a history object through default generic plot() function
plot(history, type = "arms")

# or call the Plot() directly
p1 <- Plot$new()$cumulative(history)
p2 <- Plot$new()$average(history)
```

Multiple agents can be combined within one Plot, and multiple plots can themselves again be combined into one graph by turning off the default par formatting of the Plot class and redefining par through, for example par(mfrow, mar). Some example plots that illustrate many of Plot()'s features:

```
bandit             <- SyntheticBandit$new(weights = c(0.9, 0.1, 0.1))
agents             <- list(Agent$new(RandomPolicy$new(), bandit),
                                Agent$new(OraclePolicy$new(), bandit),
                                Agent$new(ThompsonSamplingPolicy$new(1.0, 1.0), bandit),
                                Agent$new(Exp3Policy$new(0.1), bandit),
                                Agent$new(GittinsBrezziLaiPolicy$new(), bandit),
                                Agent$new(UCB1Policy$new(), bandit))
history            <- Simulator$new(agents, horizon = 100, simulations = 1000)$run()
par(mfrow = c(3, 2), mar = c(5, 5, 1, 1))
plot(history, type = "cumulative", use_colors = FALSE, no_par = TRUE)

plot(history, type = "cumulative", regret = FALSE, legend = FALSE,
     limit_agents = c("UCB1"), traces = TRUE, no_par = TRUE)

plot(history, type = "cumulative", regret = FALSE, rate = TRUE, ci = "sd",
     limit_agents = c("Exp3", "ThompsonSampling"),
     legend_position = "bottomright", no_par = TRUE)

plot(history, type = "cumulative", rate = TRUE, plot_only_ci = TRUE,
     ci = "var", smooth = TRUE, limit_agents = c("UCB1", "GittinsBrezziLai"),
     legend_position = "topright", no_par = TRUE)
```

```
plot(history, type = "average", ci = "ci", regret = FALSE, interval = 10,
     smooth = TRUE, legend_position = "bottomright", no_par = TRUE)

plot(history, limit_agents = c("ThompsonSampling"), type = "arms",
     interval = 20, no_par = TRUE)
par(mfrow = c(1, 1))
```



5. Implementing and extending Policy and Bandit subclasses

Though section 3 provides a first introduction to all of **contextual**'s main classes, in practice, researchers will mostly focus on subclassing `Policies` and `Bandits`. The current section therefore first demonstrates how to implement some well-known bandit algorithms, and, secondly, how to create `Policy` and `Bandit` sub-subclasses.

5.1. BasicBandit: a Minimal Bernoulli Bandit

Where not otherwise noted, all `Bandit` implementations in the current paper refer to (or will be configured as) multi-armed `Bandits` with Bernoulli rewards. For Bernoulli `Bandits`, the reward received is either a zero or a one: on each t they offer either a reward of 1 with probability p or a reward of 0 with probability $1 - p$. In other words, a Bernoulli bandit has a finite set of arms a $\in \{1, \ldots, k\}$ where the rewards for each arm a is distributed Bernoulli with parameter p_a, the expected reward of the arm.

One example of a very simple context-free Bernoulli bandit is **contextual**'s minimal Bandit implementation, BasicBandit:

```
BasicBandit <- R6::R6Class(
  inherit = Bandit,
  public = list(
    initialize   = function(weights = NULL) {
      self$set_weights(weights)                         # arm weight vector
      private$X <- array(1, dim = c(self$d, self$k, 1)) # context matrix of ones
    },
    # ...
    get_weights = function() {
      private$W
    },
    set_weights = function(local_W) {
      private$W <- matrix(local_W, nrow = 1L)           # arm weight vector
      self$d    <- as.integer(dim(private$W)[1])        # context dimensions
      self$k    <- as.integer(dim(private$W)[2])        # arms
      private$W
    },
    get_context = function(t) {
      contextlist <- list(k = self$k, d = self$d, X = private$X)
      contextlist
    },
    get_reward = function(t, context, action) {
      private$R  <- as.double(matrix(runif(self$k) < get_weights(), self$k, self$d))
      rewardlist <- list(
        reward                   = private$R[action$choice],
        optimal_reward_value     = private$R[which.max(private$R)]
      )
      rewardlist
    }
    # ...
  )
)
```

BasicBandit expects a weight vector of probabilities, where every element in weight represents the probability of BasicBandit returning a reward of 1 for one of its k arms. Also, observe that, at every t, BasicBandit sets private context matrix X to an unchanging, neutral d features times k arms unit matrix, which alludes to the fact that BasicBandit does not generate any covaried contextual cues related to its arms.

5.2. EpsilonFirstPolicy

An important feature of **contextual** is that it eases the conversion from formal and pseudocode policy descriptions to clean R6 classes. We will give several examples of such conversions in the current paper, starting with the implementation of the ϵ-first algorithm. In this context-free algorithm, also known as AB(C) testing, a pure exploration phase is followed by a pure exploitation phase.

In that respect, the ϵ-first algorithm is equivalent to a randomized controlled trial (RCT). An RCT, generally refered to as the gold standard clinical research paradigm, is a study design where subjects are allocated at random to receive one of several clinical interventions. On completion of an RCT, the most succesful intervention up till that point in time is suggested to be the superior "evidence-based" option from then on.

A more formal pseudocode description of this ϵ-first 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
```

And the above pseudocode converted to an `EpsilonFirstPolicy` class:

```
EpsilonFirstPolicy <- R6::R6Class(
  public = list(
    first = NULL,
    initialize = function(first = 100) {
      super$initialize(name)
      self$first <- first
    },
    set_parameters = function() {
      self$theta_to_arms <- list('n' = 0, 'mean' = 0)

      # Here, we define a list with 'n' and 'mean' theta parameters to each arm
      # through helper variable self$theta_to_arms. That is, when the number of
      # arms is 'k', the above would equal:

      # self$theta <- list(n = rep(list(0,k)), 'mean' = rep(list(0,k)))

      # ... which would also work just fine, but is much less concise.

      # When assigning both to self$theta directly and via self$theta_to_arms,
      # do make sure to do it in that order.
    },
    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
    }
  )
)
```

To evaluate this policy, instantiate both an `EpsilonFirstPolicy` and a `SyntheticBandit` (a contextual and more versatile `BasicBandit` subclass). Then add the `Bandit/Policy` pair to an `Agent`. Next, add the `Agent` to a `Simulator`. Finally, run the `Simulator`, and `plot()` the its `History` log:

```
horizon            <- 100
simulations        <- 1000
weights            <- c(0.6, 0.3, 0.3)

policy             <- EpsilonFirstPolicy$new(first = 50)
bandit             <- SyntheticBandit$new(weights = weights)

agent              <- Agent$new(policy,bandit)

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

history            <- simulator$run()

par(mfrow = c(1, 2), mar = c(2,4,1,1))
plot(history, type = "cumulative", no_par = TRUE)
plot(history, type = "arms", no_par = TRUE)
```

5.3. EpsilonGreedyPolicy

Contrary to the previously introduced ϵ-first policy, an ϵ-greedy algorithm Sutton and Barto (1998) does not divide exploitation and exploration into two strictly separate phases—it explores with a probability of *epsilon* and exploits with a probability of $1 - epsilon$, right from the start. That is, an ϵ-greedy policy with an *epsilon* of 0.1 explores arms at random 10% of the time. The other $1 - epsilon$, or 90% of the time, the policy "greedily" exploits the currently best-known arm.

This can be formalized in pseudocode as follows:

```
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 $unif(0, 1) > \epsilon$ 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
```

Converted to an EpsilonGreedyPolicy class:

```
EpsilonGreedyPolicy <- R6::R6Class(
  public = list(
    epsilon = NULL,
    initialize = function(epsilon = 0.1) {
      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
    }
  )
)
```

Assign the new class, together with SyntheticBandit, to an Agent. Again, assign the Agent to a Simulator. Then run the Simulator and plot():

```
horizon            <- 100
simulations        <- 1000
weights            <- c(0.6, 0.3, 0.3)

policy             <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit             <- SyntheticBandit$new(weights = weights)

agent              <- Agent$new(policy,bandit)

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

history            <- simulator$run()

par(mfrow = c(1, 2), mar = c(2,4,1,1))
plot(history, type = "cumulative", no_par = TRUE)
plot(history, type = "arms", no_par = TRUE)
```

5.4. Contextual Bandit: LinUCB with Linear Disjoint Models

As a final example of how to subclass **contextual**'s Bandit superclass, we move from context-free algorithms to a contextual one. As described in section 1, contextual bandits make use of side information to help them choose the current best arm to play. For example, contextual information such as a website visitors' location may be related to which article's headline (or arm) on the frontpage of the website will be clicked on most.

Here, we show how to implement and evaluate probably one of the most cited out of all contextual policies, the LinUCB algorithm with Linear Disjoint Models Li *et al.* (2010). The policy is more complicated than the previous two bandits, but when following its pseudocode description to the letter, it translates nicely to yet another Bandit subclass.

The LinUCBDisjoint algorithm works by running a linear regression with coefficients for each of d contextual features on the available historical data. Then the algorithm observes the new context and uses this context to generate a predicted reward based on the regression model. Importantly, the algorithm also generates a confidence interval for the predicted payoff for each of k arms. The policy then chooses the arm with the highest upper confidence bound. In pseudocode, following Algorithm 1 from Li *et al.* (2010):

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
```

Next, translating the above pseudocode into a well organized `Bandit` subclass:

```
#' @export
LinUCBDisjointPolicy <- R6::R6Class(
  public = list(
    alpha = NULL,
    initialize = function(alpha = 1.0) {
      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
    }
  )
)
```

Now it is possible to evaluate the LinUCBDisjointPolicy using a Bernoulli SyntheticBandit with three arms and three context features. In the code below we define each of SyntheticBandit's arms to be, on average, equally probable to return a reward. However, at the same time, the presence of a random context feature vector exercises a strong influence on the distribution of rewards over the arms per time step t: in the presence of a specific feature, one of the arms becomes much more likely to offer a reward. In this setting, the EpsilonGreedyPolicy does not do better than chance. But the LinUCBDisjointPolicy is able to learn the relationships between arms, rewards, and features without much difficulty:

```
horizon       <- 100L
simulations   <- 300L

                      # k=1  k=2  k=3              -> columns represent arms

weights      <- matrix(c(0.8, 0.1, 0.1,      # d=1   -> rows represent
                        0.1, 0.8, 0.1,      # d=2      context features
                        0.1, 0.1, 0.8),     # d=3

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

bandit       <- SyntheticBandit$new(weights = weights, precaching = TRUE)

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

simulation   <- Simulator$new(agents, horizon, simulations, do_parallel = FALSE)
history      <- simulation$run()
```

```
par(mfrow = c(1, 2), mar = c(2,4,1,1))
plot(history, type = "cumulative", regret = FALSE, no_par = TRUE)
plot(history, type = "cumulative", no_par = TRUE)
```

5.5. Subclassing Policies and Bandits

contextual's extensibility does not limit itself to the subclassing of `Policy` classes. Through its R6 based object system it is easy to extend and override any **contextual** super- or subclass. Below, we demonstrate how to apply that extensibility to sub-subclass one `Bandit` and one `Policy` subclass. First, we extend `BasicBandit`'s codePoissonRewardBandit, replacing `BasicBandit`'s Bernoulli based reward function with a Poisson based one (Presman 1991). Next, we implement an `EpsilonGreedyAnnealingPolicy` version of the ϵ-greedy policy introduced in section 5.3—where its `EpsilonGreedyAnnealingPolicy` subclass introduces a gradual reduction ("annealing") of the policy's *epsilon* parameter over T (Cesa-Bianchi and Fischer 1998; Kirkpatrick, Gelatt, and Vecchi 1983), in effect moving the policy from more explorative to a more exploitative over time.

```
PoissonRewardBandit <- R6::R6Class(
  # Class extends BasicBandit
  inherit = BasicBandit,
  public = list(
    class_name = "PoissonRewardBandit",
    initialize = function(weights) {
      super$initialize(weights)
    },
    # Overrides BasicBandit's get_reward to generate Poisson based rewards
    get_reward = function(t, context, action) {
      reward_means = c(2,2,2)
      rpm <- rpois(3, reward_means)
      private$R <- matrix(rpm < self$get_weights(), self$k, self$d)*1
      list(
        reward                  = private$R[action$choice],
        optimal_reward_value    = private$R[which.max(private$R)]
      )
    }
  )
)
EpsilonGreedyAnnealingPolicy <- R6::R6Class(
  # Class extends EpsilonGreedyPolicy
  inherit = EpsilonGreedyPolicy,
  portable = FALSE,
  public = list(
    class_name = "EpsilonGreedyAnnealingPolicy",
    # Override EpsilonGreedyPolicy's get_action, use annealing epsilon
    get_action = function(t, context) {
      self$epsilon <- 1 / log(t + 0.0000001)
      super$get_action(t, context)
    }
  )
)
weights     <- c(7,1,2)
horizon     <- 200
simulations <- 100
bandit      <- PoissonRewardBandit$new(weights)
agents      <- list(Agent$new(EpsilonGreedyPolicy$new(0.1), bandit, "EG Annealing"),
                    Agent$new(EpsilonGreedyAnnealingPolicy$new(0.1), bandit, "EG"))
simulation  <- Simulator$new(agents, horizon, simulations, do_parallel = FALSE)

history     <- simulation$run()

par(mfrow = c(1, 2), mar = c(2,4,1,1))
plot(history, type = "cumulative", no_par = TRUE)
plot(history, type = "average", regret = FALSE, no_par = TRUE)
```

6. Offline evaluation

Though it is, as demonstrated in the previous section, relatively easy to create basic synthetic Bandits to evaluate simple MAB and CMAB policies, the creation of more elaborate 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 intricate than the analysis and implementation of the policies themselves (Strehl, Li, Wiewiora, Langford, and Littman 2006a). Moreover, even when succeeding in surpassing these technical challenges, it remains an open question if an evaluation based on simulated data reflects real-world applications since modeling by definition introduces bias (Li *et al.* 2012, 2011).

It would, of course, be possible to evaluate policies by running them in a live setting. Such live evaluations would deliver unbiased, realistic estimates of a policy's effectiveness. However, the use of live data makes it more difficult to compare multiple policies at the same, as it is not possible to evaluate multiple policies at the same time with for same user (Mandel, Liu, Brunskill, and Popovic 2016). Using live data is usually also much slower than an offline evaluation, as online evaluations are dependent on active user interventions (Tewari and Murphy 2017). Furthermore, the testing of policies on a live target audience, such as patients or customers, with potentially suboptimal policies, could become either dangerous or very expensive (Bastani and Bayati 2015).

Another unbiased approach to testing MAB and CMAB policies would be to make use of offline historical data or logs. Such a data source does need to contain observed contexts and rewards, and any actions or arms must have been selected either at random or with a known probability per arm $D = (p_1, p_2, p_3, ..., p_k)$. That is, such datasets contain at least $D = (x_{t,a_t}, a_t, r_{t,a_t})$, or, in the case of know probabilities per arm $D = (x_{t,a_t}, a_t, r_{t,a_t}, p_a)$. Not only does such offline data pre-empt the issues of bias and model complexity, but it also offers the advantage that such data is widely available, as historical logs, as benchmark datasets for supervised learning, and more (Li *et al.* 2011).

There is a catch though; when we make use of offline data, we miss out on user feedback every time a policy under evaluation suggests a different arm from the one that was initially selected and saved to the offline dataset. In other words, offline data is "partially labeled" in respect to evaluated `Bandit` policies (Strehl, Langford, Li, and Kakade 2010). However, as shown in the following subsections, it is possible to get around this partial labeling problem by discarding part of the data, and by making the most of any additional information in offline datasets.

6.1. Offline Evaluation of Policies through LiSamplingBandit

The first, and most important, step in using offline data in policy evaluation is to recognize that we need to limit our evaluation to those rows of data where the arm selected is the same as the one that is suggested by the policy under evaluation (Li *et al.* 2012, 2011). In pseudocode, following Algorithm 2 from (Li *et al.* 2011):

Algorithm 4 Li Policy Evaluator

```
Require: Policy π
  Data stream of events $S$ of length $T$
  $h_0 \leftarrow \emptyset$ An initially empty history log
  $R_\pi \leftarrow 0$ An initially zero total cumulative reward
  $L \leftarrow 0$ An initially zero length counter of valid events
  for $t = 1, \ldots, T$ do
    Get the $t$-th event $(x_{t,a_t}, a_t, r_{t,a_t})$ from $S$
    if $\pi(h_{t-1}, x_{t,a_t}) = a_t$ then
      $h_t \leftarrow \text{CONCATENATE}(h_{t-1}, (x_{t,a_t}, a_t, r_{t,a_t}))$
      $R_\pi = R_\pi + r_{t,a_t}$
      $L = L + 1$
    else
      $h_t \leftarrow h_{t-1}$
    end if
  end for
  Output: rate of cumulative regret $R_\pi / L$
```

Below, an example of Algorithm 4 converted to a basic version of **contextual**'s LiSamplingOfflineBandit run on an offline simulation. The offline dataset in this example,as used before in Kaptein, McFarland, and Parvinen (2018), has been provided by a web store. Each of the dataset's 570061 rows represents a page view of a user browsing the site's products. For each view, the users were shown one out of four strategies to persuade them to buy the product on the page: no strategy (control group), authority (e.g., "recommended product"), social proof (e.g., "bestseller"), and scarcity (e.g., "almost out of stock"). Adding the product on a page to a shopping basked is counted as a success (rewarded by 1), and not adding a product as a failure (rewarded by 0).

We then proceed to run a context-free ThompsonSamplingPolicy (ignoring "time of day" and repeat visits as potential contextual side information) over the offline data set, with the three different persuasion strategies plus the strategy free control as the four to be selected arms. See Figure 5 for the policy's simulated resulting click-through rate over time.

```
library(contextual)
library(data.table)
library(RCurl)
library(foreign)

LiSamplingOfflineBandit <- R6::R6Class(
  inherit = BasicBandit,
  portable = TRUE,
  class = FALSE,
  private = list(
    S = NULL
  ),
  public = list(
    class_name = "LiSamplingOfflineBandit",
    randomize = NULL,
    initialize   = function(data_stream, k, d) {
      self$k <- k               # Number of arms (integer)
      self$d <- d               # Dimension of context feature vectors (integer)
      private$S <- data_stream  # Data stream, as a data.table
    },
    post_initialization = function() {
      private$S <- private$S[sample(nrow(private$S))]
    },
    get_context = function(index) {
      contextlist <- list(
        k = self$k,
        d = self$d,
        X = matrix(private$S$day_part, self$d, self$k)
      )
      contextlist
    },
    get_reward = function(index, context, action) {
      reward_at_index  <- as.double(private$S$reward[[index]])
      if (private$S$choice[[index]] == action$choice) {
        list(
          reward = reward_at_index
        )
      } else {
        NULL
      }
    }
  )
)

url <- "https://raw.githubusercontent.com/Nth-iteration-labs/contextual_data/"
url <- paste0(url,"master/data_persuasion_api/persuasion_api_daypart.csv")
website_data   <- getURL(setDT(read.csv(textConnection(website_data))))

horizon        <- nrow(website_data)
simulations    <- 10L
```

```
bandit        <- LiSamplingOfflineBandit$new(website_data, k = 4, d = 1)
agents        <- list(Agent$new(LinUCBHybridPolicy$new(0.6), bandit))

history       <- Simulator$new(agents, horizon, simulations, reindex = TRUE)$run()

plot(history, type = "cumulative", regret = FALSE, smooth = TRUE, traces = TRUE,
     rate = TRUE, ylim = c(0.0105, 0.014), legend_position = "topright")
```

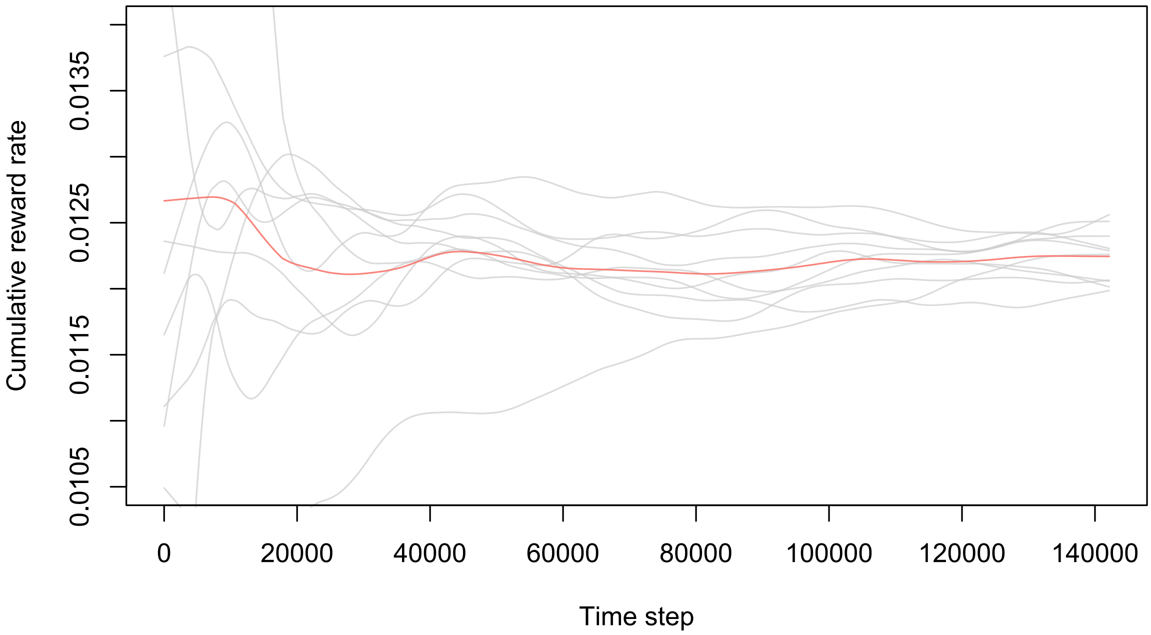


Figure 4: LinUCBHybridPolicy evaluated with LiSamplingOfflineBandit. The offline bandit samples from 570061 rows with clicks for rewards and the display of one of four "persuasive strategies" to users of an online store representing the offline bandit's four arms. The context is a one dimensional context feature representing whether the page is shown during night or day.

7. Replication of Li et al 2010

In the current section, we demonstrate how **contextual** facilitates the comparison of bandit policies on big offline datasets by running a partial replication of Li *et al.* (2010)'s frequently cited "A Contextual-Bandit Approach to Personalized News Article Recommendation.". The paper describes how the authors made use of offline Yahoo! click-through rate data to evaluate and compare the effectiveness of several context-free and contextual policies—therein introducing both the offline policy evaluator outlined in the previous section and the LinUCB algorithm introduced in section 5.4.

7.1. Description of the data

The dataset used in the Li *et al.* (2010) paper has been made available at the Yahoo! lab's website[8]. It

[8] At https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=49

contains the click-through rate from the Today news module on Yahoo!'s homepage over the course of several days in May 2009, totaling 45,811,883 separate, random and unbiased events.

Each row in the dataset describes an interaction event (click or no click) of users shown a randomly chosen article. Each of these events contains information on (I) six features for each of a varying subset of 19 to 25 articles shown below the the story position drawn from a pool of 217 articles by human editors (II) the id ofthe articles that is randomly chosen to be positioned at the story position at the top of the Today module on Yahoo!'s website (III) six user features, where each event contains another, distinct user (IV) information on whether the user clicked on the article at the story position, or not.

That is, for each event t an article represents one of k arms (that is, one of the 271 articles observed within the course of the 10 days covered by the dataset) with $\mathbb{R}^6$ features $X_{t,a}$ per arm, and another $\mathbb{R}^6$ features $X_{t,u}$ per unique visitor. Together, the flattened outer product of the user and article feature vector creates a $\mathbb{R}^{36}$ feature vector X_t for each user and article pair with the outcome value or reward r_t) click (1) or no click (0). For the further details on the data structure and the general setup of the experiment, we refer the reader to Chu, Park, Beaupre, Motgi, Phadke, Chakraborty, and Zachariah (2009) and to the original Li *et al.* (2010) paper.

7.2. Data import

As the Yahoo data is too large to fit into memory, we imported most[9] of the dataset's CSV files into a MonetDB (Idreos, Groffen, Nes, Manegold, Mullender, and Kersten 2012) instance—a fast, open source column-oriented database management system with excellent R support[10]. The import script, example import scripts for several other databases (MySQL, SQLite, Postgresql) and all other source code related to this replication can be found in the package's `demo/replication_li_2010 directory`.

7.3. Custom bandit and policies

With the Yahoo! data imported into our MonetDB server, our next step was the creation of a `Bandit` subclass to query the data and the seven `Policy` subclasses implementing the policies described in the Li *et al.* (2010) paper. Though most of these Policies were already implemented in **contextual**, the fact that only a subset of all 271 articles or arms is shown to a visitor at a time meant we needed to make some minor adaptations to all of the pre-implemented policies to make our policies run smoothly on this continually shifting pool of available arms.

In effect, first, our YahooBandit receives a `self$arm_lookup` table with an overview of 271 all arms. The bandit then proceeds to look up the article id's in the lookup table at each time step t, and includes the indexes of the currently active arms in the context list:

```
get_context = function(index) {
  ...
  # Retrieve the index of all arms this row/event.
  arm_indices_this_event   <- seq(10, 184, by = 7)
  article_ids              <- row[arm_indices_this_event]
```

[9] The first two CSV files of the Yahoo! dataset are somewhat irregular, as they contain articles with more than six features. We, therefore, decided to leave these two CSV files out of our import, resulting in 37,450,196 imported events, instead of the 45,811,883 events used in the original paper.

[10] MonetDB can be downloaded at https://www.monetdb.org/

```
      article_ids              <- article_ids[!is.na(article_ids)]
      article_ids              <- match(article_ids,self$arm_lookup)
      ...
      contextlist <- list(
        k = self$k,
        d = self$d,
        unique = self$unique, # <-- Indexes of all disjoint arms (user features)
        shared = self$shared, # <-- Indexes of all shared arms (article features)
        arms = article_ids,   # <-- Indexes of arms this event.
        X = X
      )
    }
```

These indexes are then picked up by the policy classes, which use the article id's to select and update only the currently active subset of arms. For instance, in YahooEpsilonGreedyPolicy's `get_action()`:

```
    get_action = function(t, context) {
      if (runif(1) > self$epsilon) {
        # get the max of context$arms *currently in play*
        max_index            <- context$arms[max_in(theta$mean[context$arms])]
        self$action$choice   <- max_index
      } else {
        # sample from the arms *currently in play*
        self$action$choice   <- sample(context$arms, 1)
      }
      self$action
    }
```

Next, we initiated the seven customized policy subclasses (Random, EGreedy, EGreedySeg, LinUCB Dis, LinUCB Hyb, UCB1 and the UCB1Seg[11]) for each of the six (0, 30, 20, 10, 5 and 1 percent) levels of sparsity defined in the original paper. Resulting in 7×6 = 42 Agents, each in their turn added to one Simulation instance which is then run on the offline dataset:

```
simulations             <- 1
horizon                 <- 37.45e6
...
con <- DBI::dbConnect(MonetDB.R(), host=monetdb_host, dbname=monetdb_dbname,
                                   user=monetdb_user, password=monetdb_password)

message(paste0("MonetDB: connection to '",dbListTables(con),"' succesful!"))

arm_lookup_table <-
  as.matrix(DBI::dbGetQuery(con, "SELECT DISTINCT article_id FROM yahoo"))
```

[11] EGreedy Dis and EGreedy Hyb were too summarily described for us to be able to replicate them with confidence.

```
arm_lookup_table <- rev(as.vector(arm_lookup_table))

bandit <- YahooBandit$new(k = 217L, unique = c(1:6), shared = c(7:12),
                          arm_lookup = arm_lookup_table, host = monetdb_host,
                          dbname = monetdb_dbname, user = monetdb_user,
                          password = monetdb_password, buffer_size = buffer_size)

agents <-
  list (Agent$new(YahooLinUCBDisjointPolicy$new(0.2),
                  bandit, name = "LinUCB Dis",  sparse = 0.99),
        Agent$new(YahooLinUCBHybridPolicy$new(0.2),
                  bandit, name = "LinUCB Hyb",  sparse = 0.99),
        Agent$new(YahooEpsilonGreedyPolicy$new(0.3),
                  bandit, name = "EGreedy",     sparse = 0.99),
        Agent$new(YahooEpsilonGreedySegPolicy$new(0.3),
                  bandit, name = "EGreedySeg",  sparse = 0.99),
        Agent$new(YahooUCB1AlphaPolicy$new(0.4),
                  bandit, name = "UCB1",        sparse = 0.99),
        Agent$new(YahooUCB1AlphaSegPolicy$new(0.4),
                  bandit, name = "UCB1Seg",     sparse = 0.99),
        ...
        Agent$new(YahooRandomPolicy$new(),
                  bandit, name = "Random"))

simulation <- Simulator$new(
    agents,
    simulations = simulations,
    horizon = horizon,
    do_parallel = TRUE,
    worker_max = worker_max,
    reindex = TRUE,
    progress_file = TRUE,
    include_packages = c("MonetDB.R"))

history  <- simulation$run()
...
```

7.4. Results

We were able to complete a $7 \times 6 = 42$ Agent simulation learning over all of the 37,450,196 events in our database within 22 hours on a 64 core Intel Xeon Unbuntu server with 256GB of memory. We then proceeded to use the results of the first 4.7 million events (following the original paper, representing about a day worth of events) to reproduce Li *et al.* (2010)'s Figure 4b: "CTRs in evaluation data with varying data sizes in the learning bucket.". Just as the original paper, the replicated Figure **??** reports each algorithm's relative CTR for all of the defined data sparsity levels, that is, each algorithm's CTR divided by the random policy's CTR.

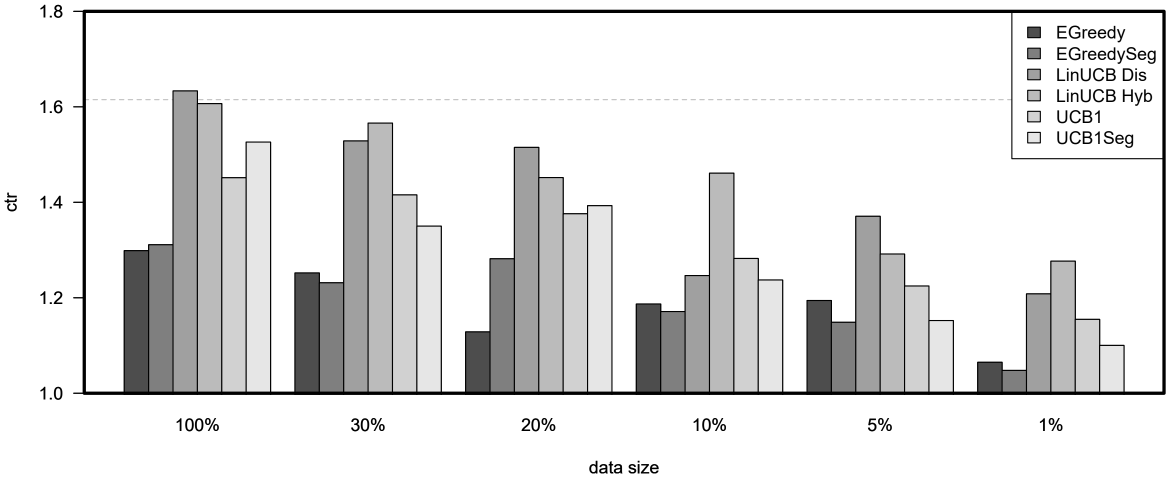


Figure 5: Replication of Figure 4b on Yahoo! dataset's day three: "CTRs in evaluation data with varying data sizes in the learning bucket." from Li *et al.* (2010).

As can be observed in Figure 5, after one day of learning, the conclusions of the original paper still stand. First, features again prove to be of use at all levels of sparsity, as LinUCB policies outperform the others consistently. Second, UCB policies generally outperform ϵ-greedy ones. And third, Hybrid LinUCB again shows benefits when the data is small, as can be deduced from it doing better in the 1% bucket. Still, as we started our simulation on the third day instead of the first, our results are close to, but not quite the same as those reported in the original paper. Particularly the third conclusion, that of the relative advantage of Hybrid LinUCB with sparse data, seems to be less convincing in our Figure 5.

So we decided to run a simulation that would continue to learn beyond the first day on sparse (1%) data to test whether Hybrid LinUCB's relative advantage would prove stable over time.

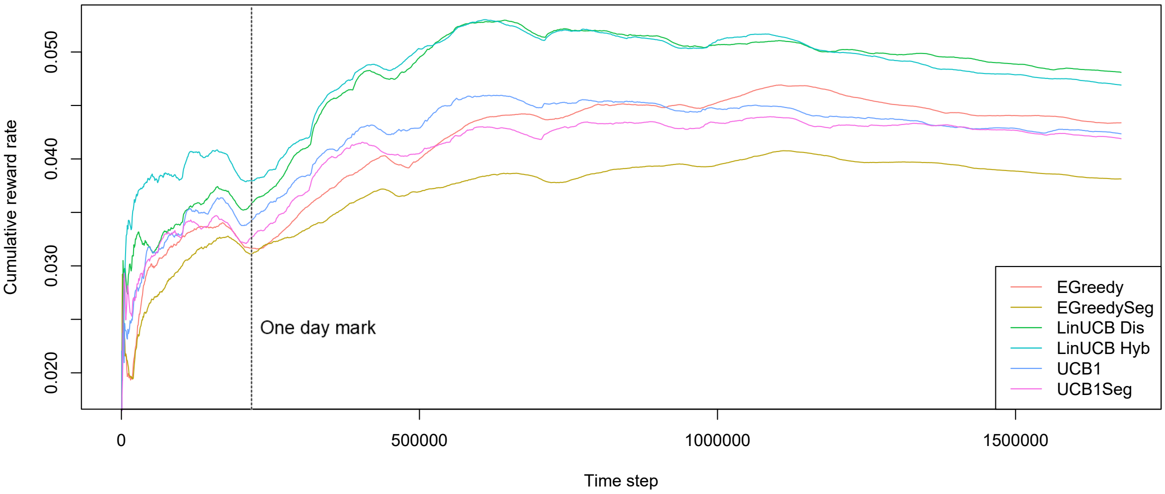


Figure 6: A plot of the cummulative reward rate (equals click-through rate) for EGreedy, EGreedySeg, LinUCB Dis, LinUCB Hyb, UCB1 and UCB1Seg policies over eight days of events from the Yahoo dataset.

From this new Figure 5, it seems that after one day of training the policies did not settle yet. Looking at the full span of about eight days of learning under 1% sparse conditions, the advantage of Hybrid LinUCB over Disjoint LinUCB seems to melt away and change into an advantage of the Disjoint version. Also, surprisingly, over the full eight days, the ϵ-greedy policy goes from the worst to third best policy overall, and to best context-free policy—outperforming both context-free UCB policies. Though we intend to further analyze this discrepancy in the near future, for now, these results seem to contradict two out of three conclusions drawn from the original figure—leaving only the first outcome, the superiority of the contextual LinUCB in comparison to several context-free ones, standing. Underlining the benefits of **contextual**, as it enabled us to replicate and confirm the original Li *et al.* (2010) paper, and then explore it further with ease. All within about two days of work: extending `Bandit` and `Policy` classes, simulating all policy classes in parallel on an offline dataset, and plotting and analyzing the results.

8. Discussion and Future Work

Statistical computational methods, in R or otherwise, are regularly made available through single-use scripts or basic, isolated code packages (Gandrud 2016). Usually, such code examples are meant to give a basic idea of a statistical method, technique or algorithm in the context of a scientific paper (Stodden, Guo, and Ma 2013). Such code examples offer their scientific audience a first inroad towards the comparison and further implementation of their underlying methods (Buckheit and Donoho 1995). However, when a set of well-researched interrelated algorithms, such as MAB and CMAB policies, find growing academic, practical and commercial adoption, it becomes crucial to offer a more standardized and more accessible way to compare such methods and algorithms (Mesirov 2010).

It is on that premise that we decided to develop the **contextual** R package—a package that would offer an open bandit framework with easily extensible bandit and policy libraries. To us, it made the most sense to create such a package in R (R Core Team 2018), as R is currently the de facto language for the dissemination of new statistical methods, techniques, and algorithms (Tippmann 2015)—while it is at the same time finding ever-growing adoption in industry (Muenchen 2012). The resulting lively exchange of R related code, data, and knowledge between scientists and practitioners offers precisely the kind of cross-pollination that **contextual** hopes to facilitate.

In making the package publicly available, we hope to further the dissemination and evaluation of both existing and new CMAB policies. By developing the package R, we furthermore hope to offer an arena for the lively exchange of policies between scientists and practitioners—as R has proven itself to be the de facto domain specific computer language in both academia and industry, from the social and medical sciences to statistics and business analytics (Tippmann 2015).

As the package is intended to be usable by practitioners, scientists and students alike, we started our paper with a general introduction to the (contextual) multi-armed bandit problem, followed by a compact formalization. We then demonstrated how our implementation flows naturally from this formalisation, with Agents that cycle Bandits and Policies through four function calls: `get_context()`, `get_action()`, `get_reward()` and `set_reward()`. Next, we evaluated some of **contextual**'s built-in Policies, delved deeper into **contextual**'s class structure, extended **contextual**'s `Bandit` and `Policy` superclasses, demonstrated how to evaluate Policies on offline datasets, and, finally replicated a frequently cited CMAB paper.

Though the package is fully functional and we expect no more changes to its core architecture and API, there is ample room to further improve and extend **contextual**. We intend to expand **contextual**'s documentation and tests. We may include more bandit paradigms, such as dueling and combinatorial bandits. We are thinking about replicating some more oft-cited CMAB papers. We expect to add a doubly robust bandit (Dudík *et al.* 2011). We are interested in growing our policy library—possibly by creating a separate repository where both existing and new CMAB policies are shared, evaluated and compared. Finally, we hope that the package will find an active community of users and developers, organically growing the package in new and unexpected ways and introducing an ever-growing group of people, from business and marketing to the social and medical sciences, to the refined sequential decision strategies offered by contextual bandit policies and algorithms.

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M (2016). "Tensorflow: A System for Large-Scale Machine Learning." In *OSDI*, volume 16, pp. 265–283.

Abe N, Biermann AW, Long PM (2003). "Reinforcement Learning with Immediate Rewards and Linear Hypotheses." *Algorithmica*, **37**(4), 263–293. doi:10.1007/s00453-003-1038-1.

ABTasty (2018). "AB Tasty." [Online; accessed 27. Aug. 2018], URL https://www.abtasty.com.

Adobe (2018). "Adobe Target." [Online; accessed 27. Aug. 2018], URL https://www.adobe.com/marketing/target.html.

Agarwal A, Bird S, Cozowicz M, Hoang L, Langford J, Lee S, Li J, Melamed D, Oshri G, Ribas O (2016). "Making Contextual Decisions with Low Technical Debt." *arXiv preprint arXiv:1606.03966*.

Agrawal R (1995). "The Continuum-Armed Bandit Problem." *SIAM journal on control and optimization*, **33**(6), 1926–1951. doi:10.1137/s0363012992237273.

Agrawal S, Goyal N (2011). "Analysis of Thompson Sampling for the Multi-armed Bandit Problem." *arXiv*. http://arxiv.org/abs/1111.1797v3.

Agrawal S, Goyal N (2012). "Thompson Sampling for Contextual Bandits with Linear Payoffs." *arXiv*. http://arxiv.org/abs/1209.3352v4.

Auer P, Cesa-Bianchi N, Fischer P (2002). "Finite-time Analysis of the Multiarmed Bandit Problem." *Machine learning*, **47**(2-3), 235–256.

Bastani H, Bayati M (2015). "Online Decision-Making with High-Dimensional Covariates." *SSRN*.

Besson L (2018). "Smpybandits: An Open-source Research Framework for Single and Multi-players Multi-arms Bandits (mab) Algorithms in Python." Online at: github.com/SMPyBandits/SMPyBandits. Code at https://github.com/SMPyBandits/SMPyBandits/, documentation at https://smpybandits.github.io/, URL https://github.com/SMPyBandits/SMPyBandits/.

Brezzi M, Lai TL (2002). "Optimal Learning and Experimentation in Bandit Problems." *Journal of Economic Dynamics and Control*, **27**(1), 87–108. doi:10.1016/s0165-1889(01)00028-8.

Bubeck S, Cesa-Bianchi N, *et al.* (2012). "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems." *Foundations and Trends® in Machine Learning*, **5**(1), 1–122. doi:10.1561/2200000024.

Buckheit JB, Donoho DL (1995). "Wavelab and Reproducible Research." In *Wavelets and statistics*, pp. 55–81. Springer. doi:10.1007/978-1-4612-2544-7_5.

Cesa-Bianchi N, Fischer P (1998). "Finite-Time Regret Bounds for the Multiarmed Bandit Problem." In *ICML*, pp. 100–108. Citeseer.

Chang W (2017). *R6: Classes with Reference Semantics*. R package version 2.2.2, URL https://CRAN.R-project.org/package=R6.

Chu W, Park ST, Beaupre T, Motgi N, Phadke A, Chakraborty S, Zachariah J (2009). "A Case Study of Behavior-driven Conjoint Analysis on Yahoo!: Front Page Today Module." In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1097–1104. ACM. doi:10.1145/1557019.1557138.

Dudík M, Langford J, Li L (2011). "Doubly Robust Policy Evaluation and Learning." *arXiv preprint arXiv:1103.4601*.

Eckles D, Kaptein M (2014). "Thompson Sampling with the Online Bootstrap." *arXiv*. http://arxiv.org/abs/1410.4009v1.

Gandrud C (2016). *Reproducible Research with R and R Studio*. Chapman and Hall/CRC.

Google (2018). "Google Analytics." [Online; accessed 26. Aug. 2018], URL https://marketingplatform.google.com/about/analytics.

Hido S, Tokui S, Oda S (2013). "Jubatus: An open source platform for distributed online machine learning." In *NIPS 2013 Workshop on Big Learning, Lake Tahoe*.

Idreos S, Groffen F, Nes N, Manegold S, Mullender KS, Kersten ML (2012). "Monetdb: Two Decades of Research in Column-oriented Database Architectures." *IEEE Data Engineering Bulletin*, **35**(1), 40–45.

Kaelbling LP, Littman ML, Moore AW (1996). "Reinforcement Learning: A Survey." *Journal of artificial intelligence research*, **4**, 237–285. doi:10.1613/jair.301.

Kaptein M, McFarland R, Parvinen P (2018). "Automated Adaptive Selling." *European Journal of Marketing*, **52**(5/6), 1037–1059.

Kaptein MC, Kruijswijk JMA (2016). "Streamingbandit: A Platform for Developing Adaptive Persuasive Systems." *JSS*.

Kaptein MC, Van Emden R, Iannuzzi D (2016). "Tracking the Decoy: Maximizing the Decoy Effect through Sequential Experimentation." *Palgrave Communications*, **2**, 16082. doi:10.1057/palcomms.2016.82.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by Simulated Annealing." *science*, **220**(4598), 671–680.

Kohavi R, Henne RM, Sommerfield D (2007). "Practical Guide to Controlled Experiments on the Web." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 959–967. ACM.

Kuleshov V, Precup D (2014). "Algorithms for Multi-Armed Bandit Problems." *arXiv preprint arXiv:1402.6028*.

Lai TL, Robbins H (1985). "Asymptotically Efficient Adaptive Allocation Rules." *Advances in applied mathematics*, **6**(1), 4–22. doi:10.1016/0196-8858(85)90002-8.

Langford J, Li L, Strehl A (2007). "Vowpal {W}abbit."

Langford J, Zhang T (2008). "The Epoch-greedy Algorithm for Multi-armed Bandits with Side Information." In *Advances in neural information processing systems*, pp. 817–824.

Li L, Chu W, Langford J, Moon T, Wang X (2012). "An Unbiased Offline Evaluation of Contextual Bandit Algorithms with Generalized Linear Models." In *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation 2*, pp. 19–36.

Li L, Chu W, Langford J, Schapire RE (2010). "A Contextual-bandit Approach to Personalized News Article Recommendation." In *Proceedings of the 19^{th} international conference on World wide web*, pp. 661–670. ACM. doi:10.1145/1772690.1772758.

Li L, Chu W, Langford J, Wang X (2011). "Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms." In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pp. 297–306. ACM, New York, NY, USA. ISBN 9781450304931. doi:10.1145/1935826.1935878.

Li S (2016). *The Art of Clustering Bandits*. Ph.D. thesis, Università degli Studi dell'Insubria. URL http://insubriaspace.cineca.it/handle/10277/729.

Lu T, Pál D, Pál M (2010). "Contextual Multi-armed Bandits." In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pp. 485–492.

Mandel T, Liu YE, Brunskill E, Popovic Z (2016). "Offline Evaluation of Online Reinforcement Learning Algorithms." In *AAAI*, pp. 1926–1933.

Mesirov JP (2010). "Accessible Reproducible Research." *Science*, **327**(5964), 415–416. doi:10.1126/science.1179653.

Mixpanel (2018). "Mixpanel." [Online; accessed 26. Aug. 2018], URL https://mixpanel.com.

Muenchen RA (2012). "The Popularity of Data Science Software." URL http://r4stats.com/articles/popularity/.

NTUCSIE-CLLab (2018). "Striatum: Contextual Bandit in Python." URL https://github.com/ntucllab/striatum.

Optimizely (2018). "Optimizely." [Online; accessed 26. Aug. 2018], URL https://www.optimizely.com.

Presman EL (1991). "Poisson Version of the Two-Armed Bandit Problem with Discounting." *Theory of Probability & Its Applications*, **35**(2), 307–317.

R Core Team (2018). "R: A Language and Environment for Statistical Computing." URL https://www.R-project.org.

Rabbi M, Aung MH, Zhang M, Choudhury T (2015). "MyBehavior: Automatic Personalized Health Feedback From User Behaviors and Preferences Using Smartphones." In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 707–718. ACM.

Riquelme C, Tucker G, Snoek J (2018). "Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling." *arXiv preprint arXiv:1802.09127*.

Rumbaugh J, Jacobson I, Booch G (2004). *Unified Modeling Language Reference Manual*. Pearson Higher Education.

Sarkar J (1991). "One-armed Bandit Problems with Covariates." *The Annals of Statistics*, pp. 1978–2002. doi:10.1214/aos/1176348382.

Shen W, Wang J, Jiang YG, Zha H (2015). "Portfolio Choices with Orthogonal Bandit Learning." In *IJCAI*, volume 15, pp. 974–980.

Stodden V, Guo P, Ma Z (2013). "Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals." *PloS one*, **8**(6), e67111. doi:10.1371/journal.pone.0067111.

Strehl A, Langford J, Li L, Kakade SM (2010). "Learning From Logged Implicit Exploration Data." In *Advances in Neural Information Processing Systems*, pp. 2217–2225.

Strehl AL, Li L, Wiewiora E, Langford J, Littman ML (2006a). "PAC Model-Free Reinforcement Learning." In *Proceedings of the 23rd international conference on Machine learning*, pp. 881–888. ACM.

Strehl AL, Mesterharm C, Littman ML, Hirsh H (2006b). "Experience-efficient Learning in Associative Bandit Problems." In *Proceedings of the 23rd international conference on Machine learning*, pp. 889–896. ACM. doi:10.1145/1143844.1143956.

Sutton RS, Barto AG (1998). "Reinforcement Learning: An Introduction." *IEEE Transactions on Neural Networks*, **9**(5), 1054. ISSN 1045-9227. doi:10.1109/TNN.1998.712192.

Tang L, Rosales R, Singh A, Agarwal D (2013). "Automatic Ad Format Selection Via Contextual Bandits." In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pp. 1587–1594. ACM.

Tewari A, Murphy SA (2017). "From Ads to Interventions: Contextual Bandits in Mobile Health." In *Mobile Health*, pp. 495–517. Springer. doi:10.1007/978-3-319-51394-2_25.

Tippmann S (2015). "Programming Tools: Adventures with R." *Nature News*, **517**(7532), 109.

Vermorel J, Mohri M (2005). "Multi-armed Bandit Algorithms and Empirical Evaluation." In *European conference on machine learning*, pp. 437–448. Springer. doi:10.1007/11564096_42.

Wang CC, Kulkarni SR, Poor HV (2005). "Arbitrary Side Observations in Bandit Problems." *Advances in Applied Mathematics*, **34**(4), 903–938. doi:10.1016/j.aam.2004.10.004.

Whittle P (1979). “Discussion On:“bandit Processes and Dynamic Allocation Indices”[jr Statist. Soc. B, 41, 148–177, 1979] by Jc Gittins.” *J. Roy. Statist. Soc. Ser. B*, **41**, 165.

Wickham H (2014). *Advanced R*. Chapman and Hall/CRC. doi:10.1201/b17487.

Wilson RC, Geana A, White JM, Ludvig EA, Cohen JD (2014). “Humans Use Directed and Random Exploration to Solve the Explore–exploit Dilemma.” *Journal of Experimental Psychology: General*, **143**(6), 2074. doi:10.1037/a0038199.

Wirfs-Brock R, Wilkerson B, Wiener L (1990). *Designing Object-Oriented Software*. CUMINCAD.

Yelp (2018). “MOE: A Global, Black Box Optimization Engine For Real World Metric Optimization.” Original-date: 2014-02-24T21:55:56Z, URL https://github.com/Yelp/MOE.

Zheng R, Hua C (2016). “Stochastic Multi-Armed Bandit.” In *Sequential Learning and Decision-Making in Wireless Resource Management*, pp. 9–25. Springer. doi:10.1007/978-3-319-50502-2_2.

9. Appendix A: UML Diagrams

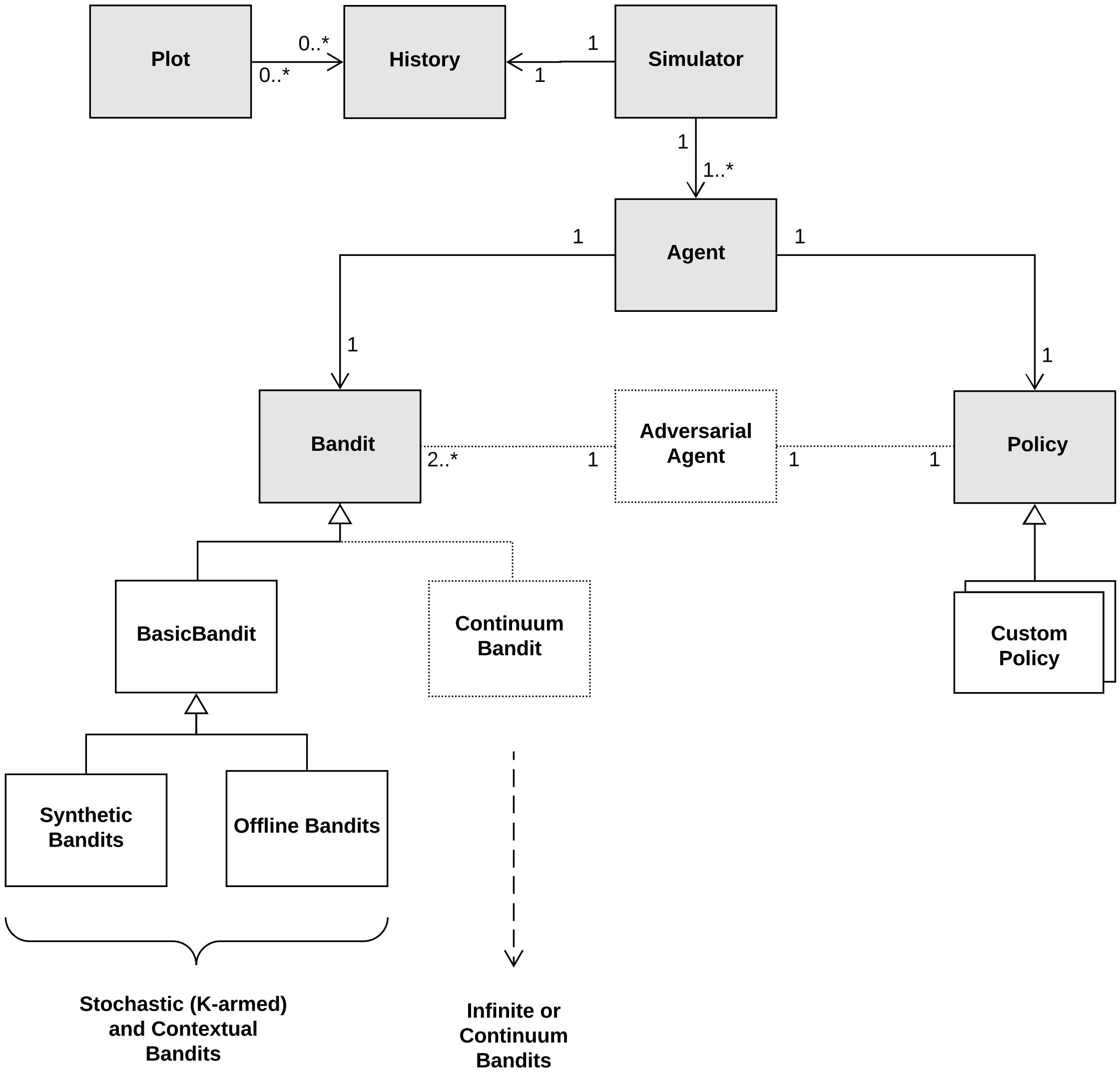


Figure 7: **contextual** UML Class Diagram

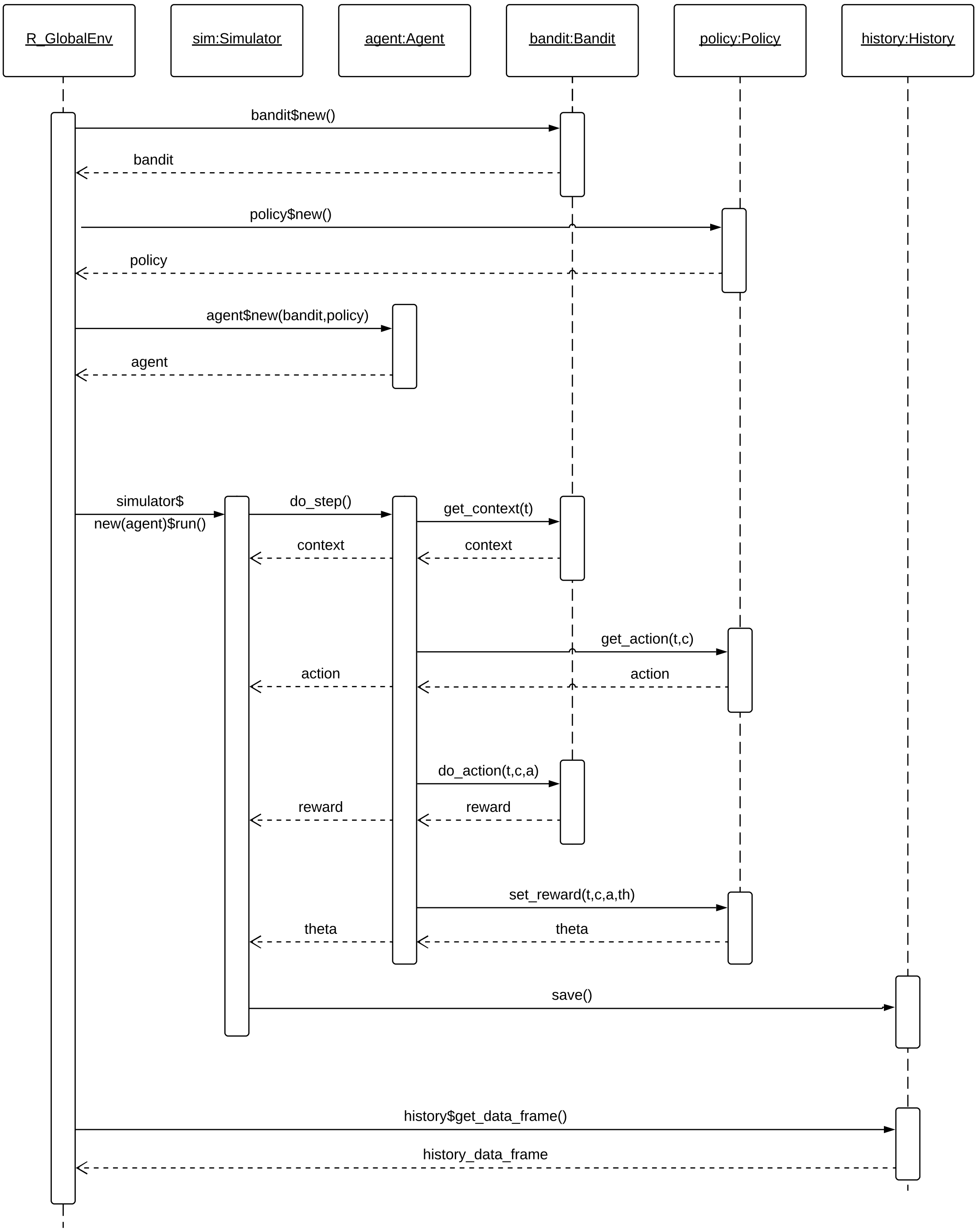


Figure 8: **contextual** UML Sequence Diagram

Affiliation:

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

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd