`StreamingBandit`: A Platform for Developing Adaptive Persuasive Systems.

Maurits Kaptein and Jules Kruijswijk

Tilburg University, Tilburg, the Netherlands
Assistant Professor, Statistics and Research Methods
m.c.kaptein@uvt.nl

Researchers in the persuasive technology field have demonstrated the effectiveness and utility of persuasive applications in diverse domains such as healthcare, energy reduction, and interactive marketing (see, e.g., 4; 2). However, there is an aspect of persuasive technologies that has long been advocated by scholars, but has by-and-large not left the research realm; this is the basic notion that persuasive technologies should "deliver the right message, at the right time, to the right user" (see, e.g., 3). Although there is a common understanding that persuasive technologies *should* be made both adaptive and personalized, creating such systems is challenging. The successful development of adaptive persuasive systems requires a combination of design, social science, and technology. With this poster we contribute to the latter: we introduce `StreamingBandit`, a platform that supports the "logic and reasoning" of adaptive or personalized persuasive systems. `StreamingBandit` is available open-source to those wishing to create adaptive persuasive systems.

1 Introducing `StreamingBandit`

`StreamingBandit` is designed using the following formalization of a personalized persuasive system:

- We assume index of the interactions $t = 1, \ldots, t = T$. (T is likely undefined at design time).
- The *context* $x_t \in \mathcal{X}_t$ where $\mathcal{X}$ is a set of variables describing the current state of the application or user.
- The *action* $a_t \in \mathcal{A}_t$ where $\mathcal{A}$ is a set of possible actions the application can take.
- The *reward* r_t is a (function of the) measured response at that point in time.
- A *policy* $\Pi\colon x_1, \ldots, x_{t'}, a_1, \ldots, a_{t'-1}, r_1, \ldots, r_{t'-1} \rightarrow a_{t'}$, is a mapping from all possible interactions (their contexts, actions, and rewards) up to some point in time $t = t'$ to the next action $a_{t'}$.

In short, `StreamingBandit`, provides a platform that allows users to implement different policies: different "rules" to assign new actions based on the historical interactions and the current context.[1]

[1] Note that the above formalization of the adaptive persuasive system in terms of context, actions, and reward, is known as a contextual multi-armed bandit problem (cMAB, or MAB for the simpler version without a context) (1).

`StreamingBandit` formalizes the challenge of designing adaptive persuasive systems as a contextual Multi-Armed Bandit problem, and allows designers to implement a policy Π on a webserver. The implementation of a policy is split into two steps:

1. The *summary* step: In each summary step a set of parameters $\theta_{t'-1}$ is updated by the new information $\{x_{t'}, a_{t'}, r_{t'}\}$. Thus, $\theta_{t'} = g(\theta_{t'-1}, x_{t'}, a_{t'}, r_{t'})$ where $g()$ is some update function. Effectively, all the prior data, $x_1, \ldots, x'_t$, $a_1, \ldots, a_{t'}$, $r_1, \ldots, r_{t'}$ are *summarized* into $\theta_{t'}$.
2. The *decision* step: In the decision step, the model $r = f(a, x_{t'}; \theta_{t'})$ is evaluated for the current context and the recommended action at time t' is selected.[2]

Figure 1 presents an overview of the platform. `StreamingBandit` is a python 3 application that runs a Tornado webserver (see `http://www.tornadoweb.org/en/stable/`) and which discloses a REST API that facilitates the implementation of the *summary* and *decision* steps as described above. The two main REST calls are:

- The *decision* call:

```
http://HOST/EXPID/getaction.json?key=EXPKEY&context={}
```

  where EXPID and EXPKEY are the ID and key of the current application and the variable context contains a JSON object encoding the context x_t. The call returns a JSON formatted object containing the selected action given the policy.
- The *summarize* call:

```
http://HOST/EXPID/setreward.json?key=EXPKEY&context={}
    &action={}&reward={}
```

  where the context x_t, action a_t and reward r_t at a point in time are used to update θ_t.

`StreamingBandit` allows users to create a new "experiment" to enable the above two calls, and allows users to write custom `python 3` scripts that implement the *summary* and *decision* steps. For the Persuasive 2016 conference we have made an instance of `StreamingBandit` available at `http://131.174.75.205`.

1.1 Using `StreamingBandit`

Configuring `StreamingBandit` consists of three steps; first, one creates a new experiment which initializes up the associated REST calls. Second, one implements the *summary* step logic in a custom `python` script which can be done using the web-based front-end of `StreamingBandit`. Finally, one implements the *decision* step.

[2] Note that one could naively think that $a_{max} = argmax_a f(a, x_{t'}; \theta_{t'})$ would be the best action to choose. However, this ignores the uncertainty in both $f()$ and θ.

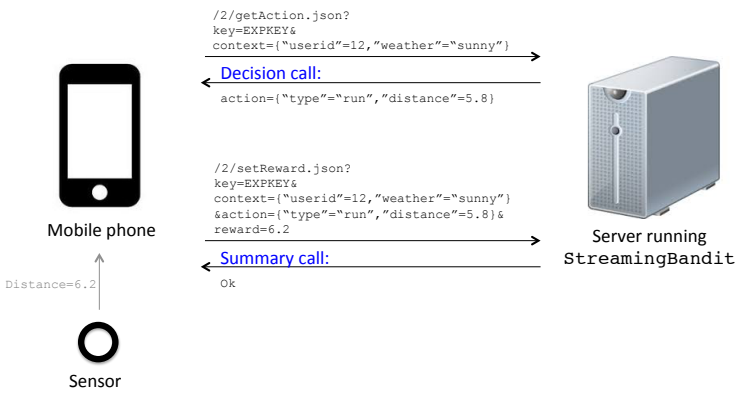

Fig. 1. Schematic overview of the core functionality of `StreamingBandit`.

Here we provide the implementation of an AB-test using `StreamingBandit`. In this simple example there is no context (thus, this is a MAB problem, not cMAB), there are only two possible actions $a_t \in \{A, B\}$, and the reward is a click on a webpage ($r_t \in \{0, 1\}$). The summarize step can be implemented as follows:

```
import libs.base as base
prop = base.Proportion(self.get_theta(key="version",
    value=self.action["version"]))
prop.update(self.reward["click"])
self.set_theta(prop, key="version", value=self.action["version"])
```

This code updates and stores the click proportion of the different versions (the actions) of the application. The decision step is given by:

```
import libs.base as base
propl = base.List(self.get_theta(key="version"),
  base.Proportion, ["A", "B"])
if propl.count() > 1000:
  self.action["version"] = propl.max()
else
  self.action["version"] = propl.random()
```

Which implements that up to 1000 interactions ($t = 1000$) randomly version A and B are suggested. After 1000 interactions, the version with the highest click through proportion is suggested.

Note that this very simple implementation of an AB-test merely touches the surface of the functionalities of `StreamingBandit`: The platform has already been used to implement personalized pricing of consumer loans for a consumer bank, persuasion profiling on a large e-commerce store, etc. `StreamingBandit` comes with a number of different libraries (such as `libs.base` mentioned above) to enable streaming processing of data. `StreamingBandit` also allows logging of all the calls that are made and all the data that is collected. We have

tried to make `StreamingBandit` both scalable and secure: the management console can easily be protected using signed cookies, and the REST calls to the core *summary* and *decision* steps are protected using the experiment ID and key combination. Scalability is ensured by a) using state-of-the art technologies in the development of `StreamingBandit` such as Tornado (as a base for the webserver), and Redis (an extremely fast, in-memory data-base system), and b) by forcing, by design, the use of online (streaming) analysis. Finally, `StreamingBandit` allows "nested" experiments: hence, the summary or decision steps of an experiment can call, using `self.run_experiment(id)`, the summary or decision code of *another* experiment. This extremely powerful feature allows one to implement complex logic to compare multiple methods of adaptation or personalization. The full documentation of `StreamingBandit` can be found at: `http://mkaptein.github.io/streamingbandit/index.html`

2 Conclusions

`StreamingBandit` is still work in progress: Although we have recently released the first stable version of the application on GitHub, we are still developing additional toolkits, examples, and documentation. Also, we are currently using `StreamingBandit` in the evaluation of adaptive persuasive technologies; we hope to be able to report on actual field trials powered by `StreamingBandit` in the near future. However, we think the current version is mature enough to share with the Persuasive Technology community, and to encourage others to use the application. We are actively seeking feedback to improve the application and maximize its use.

References

[1] Berry, D.A., Fristedt, B.: Bandit Problems: Sequential Allocation of Experiments. Springer (1985)

[2] Fogg, B.J.: Persuasive Technology: Using Computers to Change What We Think and Do. Morgan Kaufmann (2002)

[3] Kaptein, M.C., de Ruyter, B., Markopoulos, P., Aarts, E.: Tailored Persuasive Text Messages to Reduce Snacking. Transactions on Interactive Intelligent Systems 2(2), 10—-35 (2012)

[4] Tuomas, L., Oinas-Kukkonen, H.: The persuasiveness of Web-based alcohol interventions. In: 9th IFIP Conference on e-Business, e-Services, and e-Society. pp. 164–176 (2009)