SMPyBandits: an Experimental Framework for Single and Multi-Players Multi-Arms Bandits Algorithms in Python

Lilian Besson

SMPyBandits: an Experimental Framework for Single and Multi-Players Multi-Arms Bandits Algorithms in Python

Lilian Besson LILIAN.BESSON@CENTRALESUPELEC.FR
CentraleSupélec (campus of Rennes), IETR, SCEE Team,
Avenue de la Boulaie – CS 47601, F-35576 Cesson-Sévigné, France

Editor: ?

Abstract

SMPyBandits is a package for numerical simulations on *single*-player and *multi*-players Multi-Armed Bandits (MAB) algorithms, written in Python (2 or 3). This library is the most complete open-source implementation of state-of-the-art algorithms tackling various kinds of sequential learning problems referred to as Multi-Armed Bandits. It is extensive, simple to use and maintain, with a clean and well documented codebase. It allows fast prototyping of experiments, with an easy configuration system and command-line options to customize experiments.

Keywords: sequential learning; multi-armed bandit; reinforcement learning; python.

1. Presentation

1.1 Single-Player MAB

Multi-Armed Bandit (MAB) problems are well-studied sequential decision making problems in which an agent repeatedly chooses an action (the "*arm*" of a one-armed bandit) in order to maximize some total reward (Robbins, 1952), (Lai and Robbins, 1985). Initial motivation for their study came from the modeling of clinical trials, as early as 1933 with the seminal work of Thompson (Thompson, 1933), where arms correspond to different treatments with unknown, random effect. Since then, MAB models have been proved useful for many more applications, that range from cognitive radio (Jouini et al., 2009) to online content optimization like news article recommendation (Li et al., 2010), online advertising (Chapelle and Li, 2011), A/B Testing (Kaufmann et al., 2014; Yang et al., 2017), or portfolio optimization (Sani et al., 2012).

More formally, a stochastic MAB is defined by $K > 1$ distributions ν_k (arms), and *i.i.d.* rewards $r_k(t) \sim \nu_k, \forall t$. An agent choose arm $A(t) \in \{1, \dots, K\}$ at time t and observes the reward $r_{A(t)}(t)$ without knowing the other (hidden) rewards. Her goal is to maximize $\sum_{t=1}^{T} r_{A(t)}(t)$ by sequentially exploring the K arms, and she essentially has to find and exploit the best one as fast as possible. This library tackles one dimensional distributions, and supports `Bernoulli`, `binomial`, `Poisson`, and a generic `discrete` distributions, as well as `exponential`, `gamma`, `Gaussian` (of known scale or variance) and `uniform` continuous distributions, which can be truncated to an interval $[a, b]$ or have unbounded support ($\mathbb{R}$).

SMPyBandits is a complete open-source implementation of single-player bandit algorithms, containing over 65 algorithms. It uses a well-designed hierarchical structure and class inheritance scheme to minimize redundancy in the codebase. For example, many existing algorithms are index-based: they compute an index $I_k(t) \in \mathbb{R}$ for each arm k and simply play $A(t) = \arg\max_k I_k(t)$ at time t. As such, it is easy to write new index-based algorithms by inheriting from the `IndexPolicy` class, and simply defining one method `computeIndex(k)` to compute the index $I_k(t)$.

1.2 Multi-Players MAB

For Cognitive Radio and other applications, a well-studied extension is to consider $M \geq 2$ players, interacting on the *same* K arms. Whenever two or more players select the same arm at the same time, they all suffer from a collision. Different collision models has been proposed, and the simplest one consists in giving a 0 reward to each colliding players. Without any centralized supervision or coordination between players, they must learn to access the M best resources (*i.e.*, arms with highest means) without collisions. *SMPyBandits* implements all collision models found in the literature, as well as all the algorithms from the last 10 years (including `rhoRand`, `MEGA`, `MusicalChair`, and our state-of-the-art algorithms `RandTopM` and `MCTopM` from Besson and Kaufmann (2018b)). For comparison, realistic or full-knowledge centralized algorithms are also implemented.

2. Features

With this numerical framework, simulations can run on a single CPU or a single multi-core machine using joblib (Varoquaux, 2017), and summary plots are automatically saved as high-quality PNG, PDF and EPS, using matplotlib (Hunter, 2007) and seaborn (Waskom et al., 2017). Raw data from each simulation is also saved in a HDF5® file using h5py (Collette et al., 2018), an efficient and compressed binary format, to allow easy post-mortem exploration of simulation results. Making new simulations is very easy, one only needs to write a configuration script (`configuration.py`), without needing a complete knowledge of the internal code architecture.

A complete Sphinx documentation, for each algorithm and all parts of the codebase, even including the constants in the different configuration files, is available here: `https://SMPyBandits.GitHub.io`.

2.1 How to run experiments?

We show how to install *SMPyBandits*, and an example of how to run a simple experiment. This bash snippet[1] shows how to clone the code[2], and install the requirements for Python 3 (once):

```bash
# 1. get the code in the folder you want
$ git clone https://GitHub.com/SMPyBandits/SMPyBandits.git
$ cd SMPyBandits.git
# 2. install the requirements
$ pip install -r requirements.txt
```

Launching simulations is easy, for instance this snippet shows how to start $N = 1000$ repetitions of a simple non-Bayesian Bernoulli-distributed problem, for $K = 9$ arms, an horizon of $T = 10000$ and on 4 CPUs. Such simulation takes about 20 minutes, on a standard 4-cores 64 bits GNU/Linux laptop. Using environment variables (`N=1000` etc) in the command line is not required, but it is convenient:

```bash
# 3. run a single-player simulation
$ BAYES=False ARM_TYPE=Bernoulli N=1000 T=10000 K=9 N_JOBS=4 \
  MEANS=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9] python3 main.py configuration.py
```

1. See `SMPyBandits.GitHub.io/How_to_run_the_code.html` for more details.
2. SMPyBandits is also available on Pypi, see `pypi.org/project/SMPyBandits`. You can install it directly with `sudo pip install SMPyBandits`, or from a `virtualenv`.

2.2 Example of simulation and illustration

A small script `configuration.py` is used to import the arm classes, the policy classes and define the problems and the experiments. Choosing the algorithms is easy by customizing the `configuration["policies"]` list in the `configuration.py` file. For instance, one can compare the standard anytime `klUCB` algorithm against the non-anytime variant `klUCBPlusPlus` algorithm, and also `UCB` (with $\alpha = 1$) and `Thompson` (with Beta posterior).

```
configuration["policies"] = [
  {"archtype": klUCB, "params": {"klucb": klucbBern}},
  {"archtype": klUCBPlusPlus, "params": {"horizon": HORIZON, "klucb": klucbBern}},
  {"archtype": UCBalpha, "params": {"alpha": 1.0}},
  {"archtype": Thompson, "params": {"posterior": Beta}}
]
```

Running the simulation as shown above will save figures in a sub-folder, as well as save data (pulls, rewards and regret) in a HDF5 file[3]. Figure 1 below shows the average regret for these 4 algorithms. The regret is the difference between the cumulated rewards of the best fixed-armed strategy (which is the oracle strategy for stationary bandits), and the cumulated rewards of the considered algorithms.

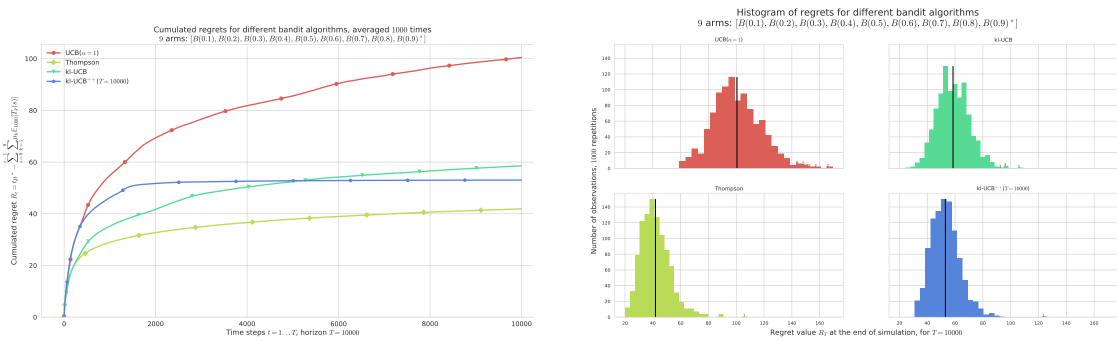

Figure 1: Example of a single-player simulation showing the average regret and histogram of regrets of 4 algorithms. They all perform very well: each algorithm is known to be order-optimal (*i.e.*, its regret is proved to match the lower-bound up-to a constant), and each but UCB is known to be optimal (*i.e.* with the constant matching the lower-bound). For instance, Thomson sampling is very efficient in average (in yellow), and UCB shows a larger variance (in red).

3. Research papers using *SMPyBandits*

SMPyBandits was used for the following research articles since 2017:

- For Besson and Kaufmann (2018b), we used *SMPyBandits* for all the simulations for multi-player bandit algorithms[4]. We designed the two `RandTopM` and `MCTopM` algorithms and proved than they enjoy logarithmic regret in the usual setting, and outperform significantly the previous state-of-the-art solutions (*i.e.*, `rhoRand`, `MEGA` and `MusicalChair`).
- In Besson et al. (2018), we used *SMPyBandits* to illustrate and compare different aggregation algorithms[5]. We designed a variant of the Exp3 algorithm for online aggregation (or boosting)

3. E.g., this simulation produces `GitHub.com/SMPyBandits/SMPyBandits/blob/master/plots/paper/example.hdf5`.
4. See the page `SMPyBandits.GitHub.io/MultiPlayers` on the documentation.
5. See the page `SMPyBandits.GitHub.io/Aggregation` on the documentation.

of experts (Bubeck and Cesa-Bianchi, 2012), called `Aggregator`. Aggregating experts is a well-studied idea in sequential learning and in machine learning in general. We showed that it can be used in practice to select on the run the best bandit algorithm for a certain problem from a fixed pool of experts. This idea and algorithm can have interesting impact for Opportunistic Spectrum Access applications (Jouini et al., 2009) that use multi-armed bandits algorithms for sequential learning and network efficiency optimization.

- In Besson and Kaufmann (2018a), we used *SMPyBandits* to illustrate and compare different "doubling trick" schemes[6]. In sequential learning, an algorithm is *anytime* if it does not need to know the horizon T of the experiments. A well-known trick for transforming any non-anytime algorithm to an anytime variant is the "Doubling Trick": start with an horizon $T_0 \in \mathbb{N}^*$, and when $t > T_i$, use $T_{i+1} = 2T_i$. We studied two generic sequences of growing horizons (geometric and exponential), and we proved two theorems that generalized previous results. A geometric sequence suffices to conserve minimax regret bounds (in $R_T = \mathcal{O}(\sqrt{T})$), with a constant multiplicative loss $\ell \leq 4$, but cannot be used to conserve a logarithmic regret bound (in $R_T = \mathcal{O}(\log(T))$). And an exponential sequence can be used to conserve logarithmic bounds, with a constant multiplicative loss also $\ell \leq 4$ in the usual setting. It is still an open question to know if a well-tuned exponential sequence can conserve minimax bounds, or even "weak" minimax bounds (in $R_T = \mathcal{O}(\sqrt{T \log(T)})$).

4. Dependencies

This library is written in Python (Foundation, 2017), for versions *2.7+* or *3.4+*, using `matplotlib` (Hunter, 2007) for 2D plotting, `numpy` (van der Walt et al., 2011) for data storing, random number generations and operations on arrays, `scipy` (Jones et al., 2001) for statistical and special functions, and `seaborn` (Waskom et al., 2017) for pretty plotting and colorblind-aware colormaps.

Optional dependencies include `joblib` (Varoquaux, 2017) for parallel simulations, `numba` (Inc. et al., 2017) for automatic speed-up on small functions, `sphinx` (Brandl et al., 2018) for generating the documentation, and `jupyter` (Kluyver et al., 2016) used with `ipython` (Pérez and Granger, 2007) to experiment with the code.

References

Lilian Besson and Emilie Kaufmann. What Doubling Trick Can and Can't Do for Multi-Armed Bandits. Preprint, February 2018a. URL `hal.inria.fr/hal-01736357`.

Lilian Besson and Emilie Kaufmann. Multi-Player Bandits Revisited. In *Algorithmic Learning Theory*, Lanzarote, Spain, April 2018b. URL `hal.inria.fr/hal-01629733`.

Lilian Besson, Emilie Kaufmann, and Christophe Moy. Aggregation of Multi-Armed Bandits Learning Algorithms for Opportunistic Spectrum Access. In *IEEE WCNC - IEEE Wireless Communications and Networking Conference*, Barcelona, Spain, April 2018. URL `hal.inria.fr/hal-01705292`.

Georg Brandl et al. Sphinx: Python documentation generator, 2018. URL `www.sphinx-doc.org`.

Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret Analysis of Stochastic and Non-Stochastic Multi-Armed Bandit Problems. *Foundations and Trends® in Machine Learning*, 5(1), 2012.

Olivier Chapelle and Lihong Li. An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems*, pages 2249–2257. Curran Associates, Inc., 2011.

6. See the page `SMPyBandits.GitHub.io/DoublingTrick` on the documentation.

Andrew Collette et al. h5py: HDF5 for Python, 2018. URL `www.h5py.org`.

Python Software Foundation. Python language reference, version 3.6, October 2017. URL `www.python.org`.

John D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9 (3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Anaconda Inc. et al. Numba, NumPy aware dynamic Python compiler using LLVM, 2017. URL `numba.pydata.org`.

Eric Jones, Travis E. Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL `www.scipy.org`.

Wassim Jouini, Daniel Ernst, Christophe Moy, and Jacques Palicot. Multi-Armed Bandit Based Policies for Cognitive Radio's Decision Making Issues. In *International Conference Signals, Circuits and Systems*. IEEE, 2009.

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the Complexity of A/B Testing. In *Conference on Learning Theory*, pages 461–481. PMLR, 2014.

Thomas Kluyver et al. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

T. L. Lai and Herbert Robbins. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *International Conference on World Wide Web*, pages 661–670. ACM, 2010.

Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. URL `ipython.org`.

Herbert Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

Amir Sani, Alessandro Lazaric, and Rémi Munos. Risk-Aversion In Multi-Armed Bandits. In *Advances in Neural Information Processing Systems*, pages 3275–3283, 2012.

William R. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25, 1933.

Stéfan van der Walt, Chris S. Colbert, and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30, March 2011. doi: 10.1109/mcse.2011.37.

Gaël Varoquaux. Joblib: running Python functions as pipeline jobs, March 2017. URL `pythonhosted.org/joblib`.

Michael Waskom et al. Seaborn: statistical data visualization, September 2017. URL `seaborn.pydata.org`.

Fanny Yang, Aaditya Ramdas, Kevin Jamieson, and Martin J. Wainwright. A framework for Multi-A(rmed)/B(andit) Testing with Online FDR Control. In *Advances in Neural Information Processing Systems*, pages 5957–5966. Curran Associates, Inc., 2017.