Multi-Armed Bandits

Sebastian Cojocariu

Machine Learning Engineer

Table of Contents

Introduction

Multi-armed bandit problem is a very intense studied paradigm of the Reinforcement Learning Field. The main challenge is to dynamically allocate a resource/arm from a particular pool of arms (which might change across time) in a way that maximizes the overall expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources. Because of the nature of the underlying problem, a trade-off between exploration and exploitation must be established to improve the overall performance of the system.

Based on the features that influence the *RL agent's* decision, several categories (with respect to the policy nature) are distinguished:

- context-free policies:
  - *Random:* a random resource is returned at each step.
  - *Explore-Exploit*: a random resource is returned for a number of steps, and then only the resource with highest expected return (while updating their expected returns).
  - *Epsilon-Greedy:* alternating between random resources and those with the highest expected return.
  - *UCB:* a deterministic approach between exploration and exploitation that is enhanced as the agent gathers more knowledge of the environment.
  - *Thompson*: uses Beta Distribution to predict the expected value of a particular arm.
  - *and variations…*

- context-based:
  - *Contextual Explore Eploit:* same as the context-free counterpart version, but uses an oracle to predict the reward based on the context (which is updated regularly).

- *Contextual Epsilon Greedy:* same as the context-free counterpart version, but uses an oracle to predict the reward based on the context (which is updated regularly).
- *Contextual Adaptive Greedy.*
- *LinUCB:* models a linear relationship between features ([1]).
- *Bootstrapping UCB:* makes use of the boostrapping technique to improve its internal models/oracles ([2]).
- *Bootstrapping Thompson:* makes use of the boostrapping technique to improve its internal models/oracles ([3])
- *and variations …*

Several pseudocodes of these policies can be found at [4].

Experimental Results

Dataset

Online content recommendation represents an important example of interactive machine learning problems that require an efficient tradeoff between exploration and exploitation. Such problems, often formulated as various types of multi-armed bandits, have received extensive research in the machine learning and statistics literature. Due to the inherent interactive nature, creating a benchmark dataset for reliable algorithm evaluation is not as straightforward as in other fields of machine learning or recommendation, whose objects are often prediction.

In our experiments, we are going to use **R6B - Yahoo! Front Page Today Module User Click Log Dataset (v2.0)**. This dataset contains a fraction of user click log for news articles displayed in the Featured Tab of the Today Module on Yahoo!'s front page. The articles were chosen uniformly at random, which allows us to use the relatively new offline evaluation method described in Li et al ([5]) to obtain an unbiased evaluation of a bandit algorithm. There are 15 days worth of logged data from October 2 to 16, 2011 containing binary raw features for the users. For each visit, the user is associated with a binary feature vector of dimension 136 (including a constant feature with ID 1) that contains information about the user like age, gender, behavior targeting features, etc. For sensitivity and privacy reasons, feature definitions are not revealed, and browser cookies (bcookies) of the users are replaced with a constant string “user”.

Because of the intensive nature of the task, we focused our research on the largest logged file, namely **ydata-fp-td-clicks-v2_0.20111011**, constisting of 2,230,762 entries.

Exploratory Data Analysis

Before diving into assessing the end-performance of these policies from an unbiased point of view, a need to better understand the data emerged.

An initial observation is that among those 135 features that should give us some information about the users, only 116 are actually present, so we reduced the context sizes by 19 features (-14%), using remapping. We tested the performance of both these sizes and found out that the evaluation was not statistically affected, as was the overall training time. Several statistics about the events are shown in Table 1.

Number of different contexts	Total articles	Average pool of articles per event
224,246	105	41

Table 1 - General statistics around the data.

From the beginning, we understood that the task would be a difficult one, for (at least) two reasons:

- Multi-label multi-class predictions: A particular context can get a reward equal to 1 for multiple articles (which was expected because users may like more than one article), so it could be associated with a multi-class multi-label paradigm from Supervised Learning (which would be an extremely hard task given that the average pool size of available articles is around 41).
- Added noise / features that are not descriptive enough: The same context, when given the same article, return different rewards. This would imply that either the context is not descriptive enough (and so we might need more features), or that some users accessed a recommended article by mistake / for other reasons. Maybe the exact bcookies that are summarized as “user” could offer more information, but this approach could not be tested with the data that we had at our disposal.

#(context, arm) pairs with different both types of rewards	#contexts with multiple articles with positive rewards	#contexts with only one article with positive reward
10,187	8,761	215,485

Table 2 - Multi-class multi-label statistics .

These statistics are highlighted in Table 2. Another important observation is that the available pool of arms per logged event changes dinamically throughout the dataset, which makes the task at hand even harder.

Implementation Details

In order to implement these policies, an Oriented Object Programming style was chosen:

- The dataset is loaded using YahooReader class (which returns a generator for the events). This class also offers some methods for statistics.
- A base class BaseBandit that is inherited by ContextualFreeBandit and ContextualBandit classes.
- RandomPolicy, ExploreExploit, EpsilonGreedy, UCB, Thompson policies inherit each from ContextualFreeBandit.
- ContextualExploreExploit, ContextualEpsilonGreedy, ContextualAdaptiveGreedy, BootstrappedUCB, BootstrappedThompson, LinUCB policies inherit each from ContextualBandit.

Their internal properties, as well as their dependencies are displayed in the Diagram 1.

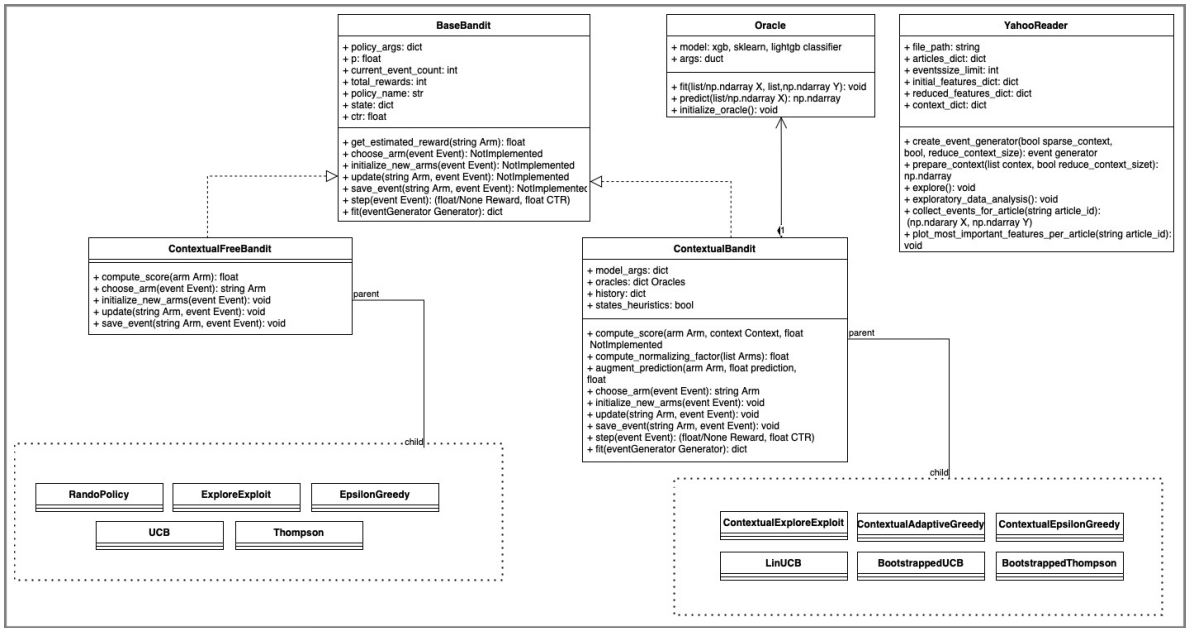

Diagram 1 - Implementation Overview.

Adopting an OOP approach helped in the development of the package, by reusing similar logic, while being easier to debug. At the same time, having a configurable API, more bandit policies could be easily implemented in the future.

Another important feature is that the *Oracle* class support classifiers from sklearn, xgboost and lightgbm libraries (with more libraries that could be added with small modifications to *Oracle* class only).

Comparison

In order to established some baselines, all the context-free policies were ran against the available data. For the context-base policies, we used as oracles *SVCs* and *LGBMClassifiers,* each with default parameters.

As it can be seen, by running the offline policy described in ([5]), the number of events that are actually used in updating the policies is drastically reduced (from around 2M to just 50k, or even less, depending on the policy). Given the difficulty of the task (and its challenges described in the Exploratory Data Analysis Section), updating on such a reduced dataset does not allow the oracles to achieve their true power (which is not the case of the context-free policies that use smaller updates each time). Moreover, because of the high imbalance between the dataset available to each oracle (rewards of 1 are fairly sparse), we used a weighting scheme to guide the training.

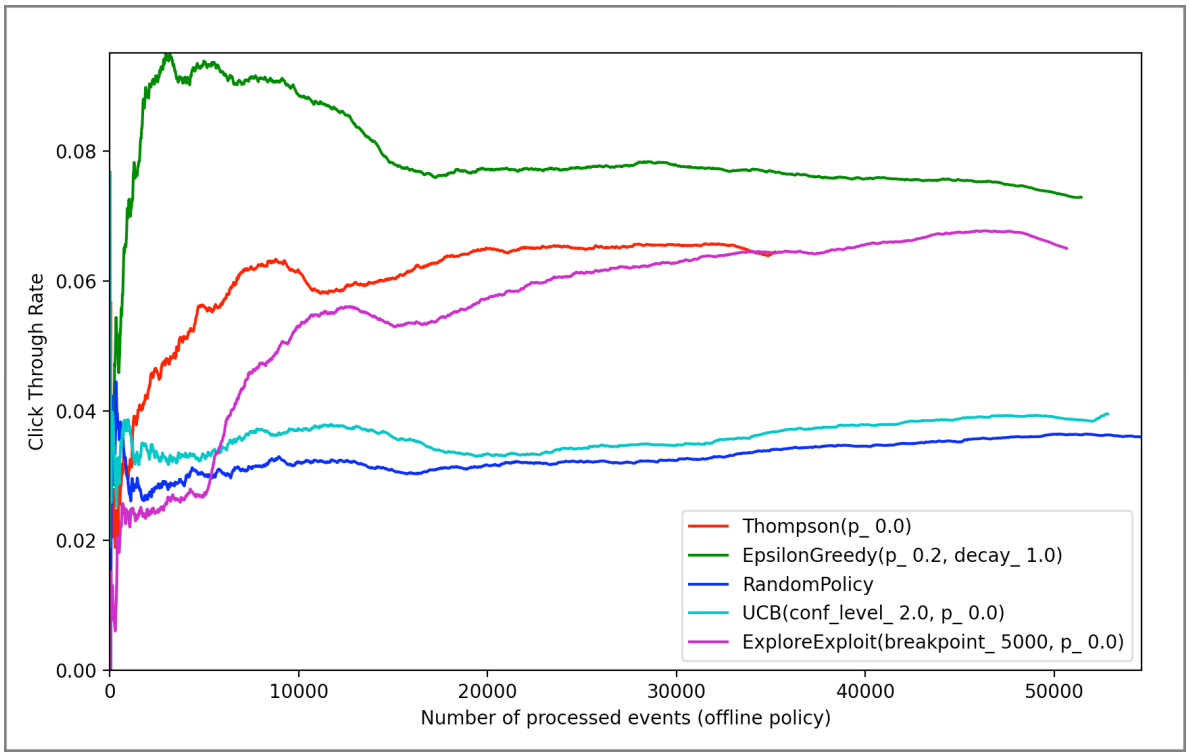

Figure 1 - Context-Free Policies Comparison.

As it can be observed, the best non-context policy with respect to Click-Trough Rate metric is the EpsilonGreedy approach, with a score just below 0.08. This is a consistent boost over the Random Policy, which shows a CTR of around 0.033. Explore Eploit method seems to perform well, displaying a CTR of 0.63. Thompson sampling shows strong strength when compared to the Explore Exploit method, but is updated only for 35k rounds (and so we cannot exactly compare its true performance). Totally unexpected, UCB policy slighly manages to beat the Random approach (Figure 1).

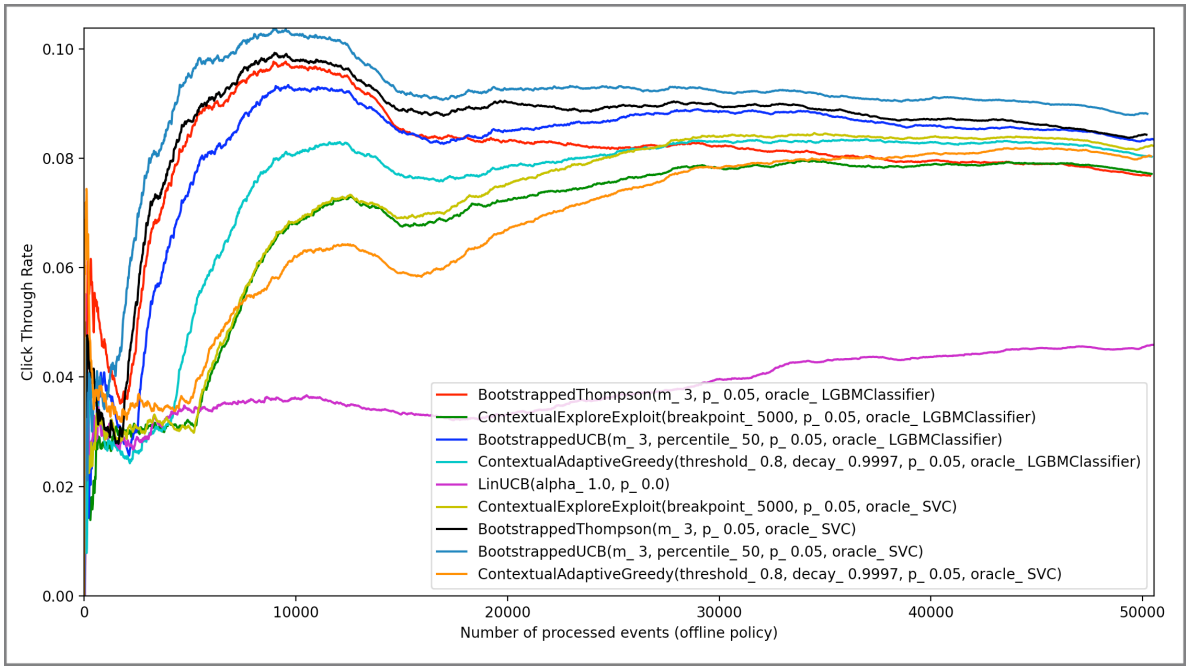

Figure 2 - Context-Based Policies Comparison (SVC vs LGBM).

Turning our attention to the contextual policies, we can observe a slight increase in performance over the EpsilonGreedy approach. The best policies in this category are the ones that use bootstrapping, with a CTR between 0.085-0.09 each. The other policies follow the same ascending trend, with a CTR of about 0.08, still slightly better than the best context-free policies. Surprisingly, LinUCB shows only a CTR of 0.044 (0.004 more than UCB policy score), a performance that is definitely beaten by the best three context-free policies. An explanation for this situation would be the nature of the involved features (which are binary and thus discrete, not continuous) which cannot be modeled via a linear relationship.

A final comparison of all the policies, as well as a comparison between context-base policies that use SVC and LGBM are given in Figure 2 and Figure 3.

Some improvements that could be applied to increase the gap between context-free and context-base policies even more would be:

- Finetuning the oracles for each particular policy (instead of using the base arguments).
- Adaptively changing the hyperparameters as the agent learns about the environment.
- Use a hybrid pool of models (in the case of BootstrappedUCB and BootstrappedThompson).
- Train on a larger dataset (which would be possible in an online simulation).
- Find better values/heuristics to deal with class imbalances.

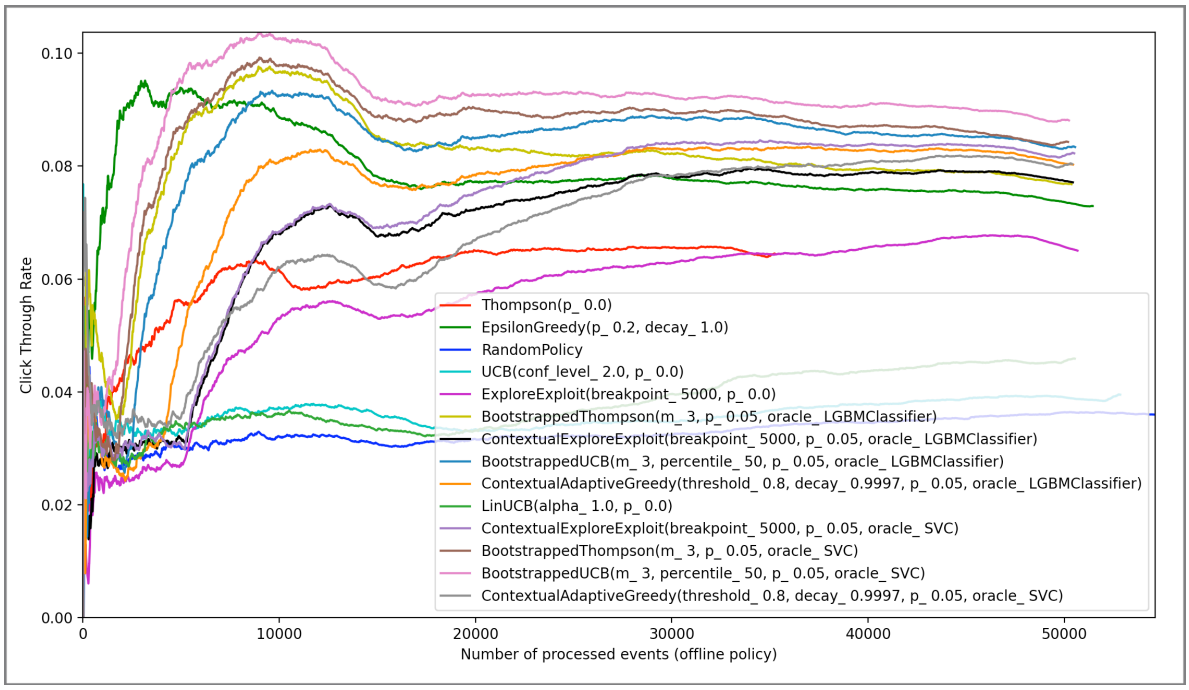

Figure 3 - Final Comparison.

Feature Importance

In order to understand which feature have a direct influence over recommending a specific article, we used a OneVSRest approach. In achieving this, several steps were involved (see *collect_events_for_article()* method from YahooReader):

- All the *inconsistencies* were trimmed: pairs (context, arm) that got two different rewards types were considered noise, and so they were discarded from any future decision.
- All the contexts that received a reward equal to 1 against our article of interest were added. Since it was also possible that the same context to predict a reward of 1 against other articles as well, we allowed multi-labeling.
- Finally, the contexts that predicted the reward 1 against our target article were given higher weights.
- The contexts that did not have a reward of 1 on neither articles were discarded too (since they will be of no use).

The collected data was used to train an XGBClassifier. On the fitted model we then applied the permutation feature technique to assess each feature importance.

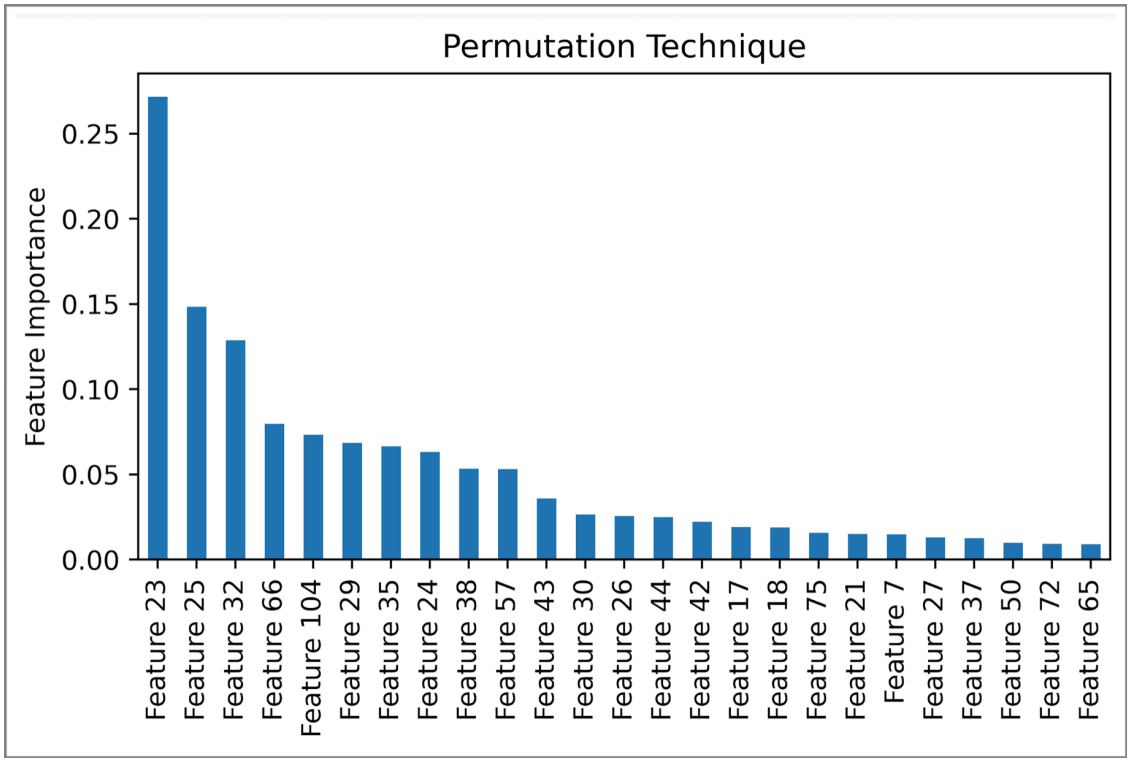

Figure 4 - Feature Importances (article *id-590109*).

Optimisations and future work

- Added an epsilon parameter for each policy (context-free and context-base) for more exploration.
- The oracle(s) predictions are enhanced by using an euristică around the estimated return per arm as well. Different heuristics can be added in the future.
- The oracles are retrained only after the dataset available to them increased by at least threshold% from the last fit.
- The oracles trainings take into account the classes imbalances.
- Article features might help (clustering them might give an insight about what users tend to like, since usually they like topics that are related).
- Weight sharing across oracles from different arms.
- Larger corpus (using an offline policy drastically decreased the dataset used in updating the oracles).
- Apply custom feature selection per arm to help the training and remove noisy features.
- Use Shapley values to assess features importances ([7]).

Bibliography

[1] LinUCB

[2] Bootstrapping Upper Confidence Bound

[3] Bootstrapping Thompson

[4] Pseudocode for various policies

[5] Offline Evaluation for Contextual Multi-Armed Bandits

[6] Taming the Monster

[7] Shapley Framework