COMP90051 Statistical Machine Learning
Project 2 Description[1]

Due date: 5:00pm Friday, 27 May 2022 **Weight:** 25%; forming combined hurdle with Proj1

Academic misconduct: You are reminded that all submitted Project 2 work is to be your own individual work. Automated similarity checking software will be used to compare all submissions. It is University policy that academic integrity be enforced. For more details, please see the policy at `http://academichonesty.unimelb.edu.au/policy.html`. Academic misconduct hearings can determine that students receive zero for an assessment, a whole subject, or are terminated from their studies. You **may not** use software libraries or code snippets found online, from friends/private tutors, or anywhere else. You can only submit your own work.

Multi-armed bandits (MABs) are a simple yet powerful framework for sequential decision-making under uncertainty. One of the many applications of MABs was pioneered by Yahoo! News, in deciding what news items to recommend to users based on article content, user profile, and the historical engagement of the user with articles. Given decision making in this setting is sequential (what do we show next?) and feedback is only available for articles shown, Yahoo! researchers observed a perfect formulation for MABs like those (ϵ-Greedy and UCB) discussed in Lecture 17 and Workshop Week 9.[2] These researchers realised that incorporating some element of user-article state requires *contextual bandits*: articles are arms; context per round incorporates information about both user and article (arm); and $\{0, 1\}$-valued rewards represent clicks. Therefore the per round cumulative reward represents click-through-rate, which is exactly what online services want to maximise to drive user engagement and advertising revenue. In this project, you will work *individually* to implement several MAB algorithms. These are a little more advanced that what was covered in class, coming from real research papers that you will have to read and understand yourself. By the end of the project you should have developed:

ILO1. A deeper understanding of the MAB setting and common MAB approaches;

ILO2. An appreciation of how MABs are applied;

ILO3. Demonstrable ability to implement ML approaches in code; and

ILO4. Ability to understanding recent ML papers, understand their focus, contributions, and algorithms enough to be able to implement and apply them. (While ignoring details not needed for your task.)

Overview

The project consists of four parts.

1. Thompson sampling MAB (Agrawal & Goyal, 2012) [4 marks]
2. Thompson sampling contextual MAB with linear payoffs (Agrawal & Goyal, 2013) [5 marks]
3. Thompson sampling MABs with fair exposure (Wang *et al.*, 2021) [8 marks]

[1]Special thanks to Neil Marchant for generously co-designing this project.
[2]Note that the skeleton for this project may deviate from the code in Workshop Week 9.

4. SquareCB contextual MAB with a regression oracle (Foster & Rakhlin, 2020) [8 marks]

All parts are to be completed in the provided Python Jupyter notebook `proj2.ipynb`.[3] Detailed instructions for each task are included in this document. These will require you to consult one or more academic papers. We provide helpful pointers in this project spec to guide your reading and to correct any ambiguities, with margin mark symbol 💡.

MAB algorithms. The project's tasks require you to implement MAB algorithms by completing provided skeleton code in `proj2.ipynb`. All of the MAB algorithms must be implemented as sub-classes of a base `MAB` class (defined for you). This ensures all MAB algorithms inherit the same interface, with the following methods:

- `__init__(self, n_arms, ..., rng)`: Initialises the MAB with the given number of arms `n_arms` and pseudo-random number generator `rng`. Arms are indexed by integers in the set $\{0, \ldots, \texttt{n\_arms} - 1\}$. All MAB algorithms take additional algorithm-specific parameters in place of '`...`'. *Hint: When instantiating a MAB, you should set* `rng` *to be the random generator initialised for you in the notebook. This will ensure your results are reproducible.*

- `play(self, context)`: Plays an arm based on the provided `context` (a multi-dimensional array that encodes user and arm features). For non-contextual bandits, this method should accept `context=None`. The method must return the integer index of the played arm in the set $\{0, \ldots, \texttt{n\_arms} - 1\}$. *Note: this method should not update the internal state of the MAB. All play methods should tie-break uniformly at random in this project. If a MAB class's* `play()` *method is undefined due to insufficient* `update()` *calls, you should by default pick an unplayed arm uniformly at random.*

- `update(self, arm, reward, context)`: Updates the internal state of the MAB after playing an arm with integer index `arm` for given `context`, receiving an real-valued `reward`. For non-contextual bandits, this method should accept `context=None`.

Your implementations *must* conform to this interface. You may implement some functionality in the base `MAB` class if you desire—*e.g.*, to avoid duplicating common functionality in each sub-class. Your classes may also use additional private methods to better structure your code. And you may decide how to use class inheritance.

Evaluating MAB classes. Each task directs you to evaluate your implementations in some way. You will need to use the `offline_eval()` function and dataset file provided. See Appendix A for details.

Python environment. You must use the Python environment used in workshops to ensure markers can reproduce your results if required. We assume you are using Python $\geq$ 3.8, numpy $\geq$ 1.19.0, scikit-learn $\geq$ 0.23.0 and matplotlib $\geq$ 3.2.0.

Other constraints. You may not use functionality from external libraries/packages, beyond what is imported in the provided Jupyter notebook highlighted here with margin marking ☁. You must preserve the structure of the skeleton code—please only insert your own code where specified. You should not add new cells to the notebook. You may discuss the bandit learning slide deck or Python at a high-level with others, including via Piazza, but do not collaborate with anyone on direct solutions. You may consult resources to understand bandits conceptually, but do not make any use of online code *whatsoever.* (We will run code comparisons against online partial implementations to enforce these rules. See 'academic misconduct' statement above.)

[3] We appreciate that while some have entered COMP90051 feeling less confident with Python, many workshops so far have exercised and built up basic Python and Jupyter knowledge. Both are industry standard tools for the data sciences.

Submission Checklist

You must complete all your work in the provided `proj2.ipynb` Jupyter notebook. When you are ready to submit, follow these steps. You may submit multiple times. We will mark your last attempt. *Hint: it is a good idea to submit early as a backup. Try to complete Part 1 in the first week and submit it; it will help you understand other tasks and be a fantastic start!*

1. Restart your Jupyter kernel and run all cells consecutively.
2. Ensure outputs are saved in the `ipynb` file, as we may choose not to run your notebook when grading.
3. Rename your completed notebook from `proj2.ipynb` to `username.ipynb` where `username` is your university central username[4].
4. Upload your submission to the Project 2 Canvas page.

Marking

Projects will be marked out of 25. Overall approximately 50%, 30%, 20% of available marks will come from correctness, code structure & style, and experimentation. Markers will perform code reviews of your submissions with **indicative** focus on the following. We will endeavour to provide (indicative not exhaustive) feedback.

1. *Correctness:* Faithful implementation of the algorithm as specified in the reference or clarified in the specification with possible updates in the Canvas changelog. It is important that your code performs other basic functions such as: raising errors if the input is incorrect, working for any dataset that meets the requirements (*i.e.*, not hard-coded).
2. *Code structure and style:* Efficient code (*e.g.*, making use of vectorised functions, avoiding recalculation of expensive results); self-documenting variable names and/or comments; avoiding inappropriate data structures, duplicated code and illegal package imports.
3. *Experimentation:* Each task you choose to complete directs you to perform some experimentation with your implementation, such as evaluation, tuning, or comparison. You will need to choose a reasonable approach to your experimentation, based on your acquired understanding of the MAB learners.

Late submission policy. Late submissions will be accepted to 4 days at -2.5 penalty per day or part day.

Part Descriptions

Part 1: Thompson sampling MAB [4 marks total]

Your first task is to implement a Thompson sampling MAB by completing the `TS` Python class. *Thompson sampling* is named after William R. Thompson who discovered the idea in 1933, before the advent of machine learning. It went unnoticed by the machine learning community until relatively recently, and is now regarded as a leading MAB technique. When applied to MABs, Thompson sampling requires a Bayesian model of the rewards. Binary rewards in $\{0, 1\}$ can be modelled as Bernoulli draws with different parameters per arm, each starting with a common Beta prior. This is described in Algorithm 1 "Thompson Sampling for Bernoulli bandits" of the following paper:

[4]LMS/UniMelb usernames look like `brubinstein`, not to be confused with email such as `benjamin.rubinstein`.

Shipra Agrawal and Navin Goyal, 'Analysis of Thompson sampling for the multi-armed bandit problem', in *Proceedings of the Conference on Learning Theory (COLT 2012)*, 2012.
http://proceedings.mlr.press/v23/agrawal12/agrawal12.pdf

While the COLT'2012 Algorithm 1 only considers a uniform Beta$(1, 1)$ prior, you are to implement a more flexible Beta(α_0, β_0) prior for any given $\alpha_0, \beta_0 > 0$. A point that may be missed on first reading, is that while each arm begins with the same prior, each arm updates its own posterior. Note that tie-breaking in `play` should be done uniformly-at-random among value-maximising arms.

Experiments. Once your `TS` class is implemented, it is time to perform some basic experimentation.

(a) Include and run an evaluation on the given dataset where column 1 forms arms, column 2 forms rewards, and columns 3–102 form contexts:

```
mab = TS(10, 10, alpha0=1.0, beta0=1.0, rng=rng)
TS_rewards, TS_ids = offline_eval(mab, arms, rewards, contexts, 800)
print("TS average reward ", np.mean(TS_rewards))
```

(b) Run offline evaluation as above, but now plot the per-round cumulative reward *i.e.*, $T^{-1}\sum_{t=1}^{T} r_{t,a}$ for $T = 1..800$ as a function of round T. We have imported `matplotlib.pyplot` to help here.

(c) Can you tune your MAB's hyperparameters? Devise and run a grid-search based strategy to select `alpha0` and `beta0` for `TS` as Python code in your notebook. To simplify the task, you may set `alpha0`=`beta0`=`a` and vary `a`. Output the result of this strategy—which could be a graph, number, etc. of your choice.

Part 2: Thompson sampling contextual MAB with linear payoffs [5 marks total]

In this part, you are to implement a contextual MAB—likely the first you've seen in detail—which is a direct mashup of Thompson sampling and Bayesian linear regression (covered in Lecture 18). Your implementation should be completed in the `LinTS` skeleton class, based on Algorithm 1 of the following paper:

Shipra Agrawal and Navin Goyal, 'Thompson sampling for contextual bandits with linear payoffs', in *Proc. International Conference on Machine Learning (ICML 2013)*, pp. 127-135. 2013.
http://proceedings.mlr.press/v28/agrawal13.pdf

While the ICML'2013 intro does re-introduce the Thompson sampling framework explored in Part 1, it can be very much skimmed. Section 2.1 introduces the setting formally—information on regret and the assumptions[5] are not important for simple experimentation. Section 2.2 and Algorithm 1 'Thompson Sampling for Contextual bandits' is the key place to find the described algorithm to be implemented. Note that the first 14 lines of Section 2.3 explains the hyperparameters ϵ, δ further: the latter controls our confidence of regret being provably low (and so we might imagine taking it to be 0.05 as in typical confidence intervals); while advice is given for setting ϵ when you know the total number of rounds to be played. All that said, R, ϵ, δ only feature in the expression v, while we wouldn't really know R. And so you should just use v as a hyperparameter to control explore-exploit balance.

[5]Sub-Gaussianity is a generalisation of Normally distributed rewards. I.e., while they don't assume the rewards are Normal, they assume something Normal-like in order to obtain theoretical guarantees. R takes the role of $1/\sigma$ and controls how fast likelihood of extreme rewards decays. You can ignore all this—phew!

Experiments. Repeat the same set of experiments from Part 1 here, except for (respectively):

(a) Here run `mab = LinTS(10, 10, v=1.0, rng=rng)` and save the results in `LinTS_rewards` and `LinTS_ids` instead.

(b) Plot the per-round cumulative reward for `TS` and `LinTS` on the same set of axes.

(c) This time consider tuning `v`.

Part 3: Thompson sampling MABs with fair exposure [8 marks total]

You may have noticed that the MABs covered so far exhibit a *winner-takes-all* behaviour. They quickly learn which arm yields the highest reward, and keep playing that arm, even if other arms are almost equally as good. This is not desirable for applications like content recommendation, where the MAB's recommendations become too narrow, not giving fair exposure to other quality content. In this part, you will implement two MABs from the following ICML 2021 paper, which are designed to achieve *merit-based fairness of exposure* while still optimising utility to users:

> Lequn Wang, Yiwei Bai, Wen Sun, and Thorsten Joachims. 'Fairness of Exposure in Stochastic Bandits', in *Proc. International Conference on Machine Learning (ICML 2021)*, pp. 10686–10696. 2021. http://proceedings.mlr.press/v139/wang21b/wang21b.pdf

You may skim the first few pages, up to and including Section 3.1, for motivation and a formulation of the fairness of exposure (FairX) MAB setting. The first MAB you are to implement is an extension of the basic Thompson sampling MAB from Part 1 to the FairX setting, called FairX-TS. It is described in Section 3.3 and Algorithm 2. Your implementation should be completed in the `FairXTS` skeleton class using the same Beta-Bernoulli model for the rewards as Part 1.

Experiments. Repeat the same set of experiments from Part 1, except for (respectively):

(a) Here run `mab = FairXTS(10, 10, ?, rng=rng)` and save the results in `FairXTS_rewards` and `FairXTS_ids` instead.

(b) Plot the per-round cumulative reward for `TS` and `FairXTS` on the same set of axes.

(c) Plot the number of pulls for each arm for `TS` and `FairXTS` and comment on any differences you observe.

The second MAB you are to implement is FairX-LinTS described in Sections 4.3 and Algorithm 4. It extends the contextual Thompson sampling MAB from Part 2 to the FairX setting. You may like to skim Section 4.1 for background on contextual FairX MABs. Your implementation should use the same Bayesian linear regression model for the rewards as Part 2.

Experiments. Repeat the same set of experiments from Part 1, except for (respectively):

(d) Here run `mab = FairXLinTS(10, 10, ?, rng=rng)` and save the results in `FairXLinTS_rewards` and `FairXLinTS_ids` instead.

(e) Plot the per-round cumulative reward for `LinTS` and `FairXLinTS` on the same set of axes.

Part 4: SquareCB contextual MAB with a regression oracle [8 marks total]

In this part, you will implement a final learner for the contextual MAB setting, for 'general purpose' contextual MABs. Where `LinTS` is designed to work only with Bayesian linear regression estimating arm rewards (the Bayesian version of ridge regression), general-purpose MABs aim to be able to use *any supervised learner* instead. The following ICML'2020 paper describes a state-of-the-art approach to general-purpose MABs. (Note in this paper 'oracle' or 'online regression oracle' refers to this user-defined supervised learning algorithm. It is an oracle because the bandit can call it for answers without knowing how the oracle works.)

> Dylan Foster and Alexander Rakhlin. 'Beyond UCB: Optimal and Efficient Contextual Bandits with Regression Oracles.' In *Proceedings of the 37th International Conference on Machine Learning (ICML'2020)*, pp. 3199-3210, PMLR, 2020. `http://proceedings.mlr.press/v119/foster20a/foster20a.pdf`

You are to read this paper up to and not including Theorem 1, then the first paragraph of Section 2.2 and Algorithm 1 (*i.e.*, you might skip Theorem 1 to Section 2.1, and most of Section 2.2 after the algorithm is described; these skipped discussions are interesting theoretical developments but not needed for the task).

You should implement the paper's Algorithm 1 within the provided skeleton code for the `SquareCB` class. As before, you will need to implement the `__init__`, `play`, and `update` methods. The parameters and return values for each method are specified in docstrings in `proj2.ipynb`. While the paper allows for any 'online regression oracle' to be input into Algorithm 1 as parameter 'SqAlg', you should hard-code your oracle to be this logistic regression implementation using `scikit-learn` which is imported for you in the `proj2.ipynb` skeleton. Use the logistic regression class's defaults. We have picked logistic regression since the data in this project has binary rewards for clicks.

While this paper appears to be free of errors, it is sophisticated so we offer pointers to help you get started. *Hint: this paper deals with losses not rewards. To fit this into our framework, it might help to convert a loss to a reward by using its negative value. Hint: Line 6 of the algorithm defines $p_{t,a}$ of a distribution but in general this might not sum to one to form a proper distribution. You should hard-code parameter μ to be the number of arms to fix this.* Finally, you should be sure to use a separate model per arm.

Experiments. Repeat the same set of experiments as above, except for (respectively):

(a) Here run `mab = SquareCB(10, 10, gamma=18.0, rng=rng)` with all rewards, saving results in `SquareCB_rewards` and `SquareCB_ids` instead. *Hint: Where does $\gamma = 18$ come from? It was examined by a reference, Abe & Long.*

(b) When plotting the results on all data, include `LinTS`'s curve too.

(c) This time consider tuning `gamma`.

A Appendix: Details for Off-Policy Evaluation and Dataset

You are directed to experiment with your bandits in each project task. To help with this, we have provided in the skeleton notebook a useful `offline_eval()` function, and a dataset. This section describes both.

Off-policy evaluation: The `offline_eval()` function evaluates bandit classes on previously collected data. The parameters and return value of the `offline_eval` function—its interface—are specified in the function's docstring. Recall that bandits are interactive methods: to train, they must interact with their environment. This is true even for evaluating a bandit, as it must be trained at the same time as it is evaluated. But this requirement to let a bandit learner loose on real data, was a major practical challenge for industry deployments of MAB. Typically bandits begin with little knowledge about arm reward structure, and so a bandit must necessarily suffer poor rewards in beginning rounds. For a company trying out and evaluating dozens of bandits in their data science groups, this would be potentially prohibitively expensive. A breakthrough was made when it was realised that MABs can be evaluated *offline* or *off-policy*. ('Policy' is the function outputting an arm given a context.) With off-policy evaluation: you collect just one dataset of uniformly-random arm pulls and resulting rewards; then you evaluate any interesting future bandit learners on that one historical dataset! We have provided this functionality in `offline_eval()` for you, which implements Algorithm 3 "Policy_Evaluator" of the following paper:

> Lihong Li, Wei Chu, John Langford, Robert E. Schapire, 'A Contextual-Bandit Approach to Personalized News Article Recommendation', in *Proceedings of the Nineteenth International Conference on World Wide Web (WWW 2010)*, Raleigh, NC, USA, 2010.
> https://arxiv.org/pdf/1003.0146.pdf

In order to understand this algorithm, you could read Section 4 of WWW'2010 if you want to.[6]

Dataset. Alongside the skeleton notebook, we provide a 2 MB `dataset.txt` suitable for validating contextual MAB implementations. You should download this file and place it in the same directory as your local `proj2.ipynb`. It is formatted as follows:

- 10,000 lines (*i.e.*, rows) corresponding to distinct site visits by users—events in the language of this task;
- Each row comprises 103 space-delimited columns of integers:
  - Column 1: The arm played by a uniformly-random policy out of 10 arms (news articles);
  - Column 2: The reward received from the arm played—1 if the user clicked 0 otherwise;
  - Columns 3–102: The 100-dim flattened context: 10 features per arm (incorporating the content of the article and its match with the visiting user), first the features for arm 1, then arm 2, etc. up to arm 10.

[6]What might not be clear in the pseudo-code of Algorithm 3 of WWW'2010, is that after the MAB plays (policy written as π) an arm that matches the next logged record, off-policy evaluation notes down the reward as if the MAB really received this reward—for the purposes of the actual evaluation calculation; it also runs `update()` of the MAB with the played arm a, reward r_a, and the context $\mathbf{x}_1, \ldots, \mathbf{x}_K$ over the K arms.