An Introduction to Contextual Bandits

Written by: Dwight Gunning (https://getstream.io/blog/author/dwight/)
Published on: August 23, 2016 at 1:49 pm.
Updated on: August 21, 2017 at 2:16 pm.
Tags: Examples (https://getstream.io/blog/tag/examples/), Feed Personalization (https://getstream.io/blog/tag/feed-personalization/), Machine Learning (https://getstream.io/blog/tag/machine-learning/)

In this post I discuss the Multi Armed Bandit problem and its applications to feed personalization. First, I will use a simple synthetic example to visualize arm selection in with bandit algorithms, I also evaluate the performance of some of the best known algorithms on a dataset for musical genre recommendations.

What is a Multi Armed Bandit?

Imagine you have a bag full of biased coins. How would you go about finding the one that gives you the highest reward in expectation? You could use your favorite statistical tool by running enough independent trials to get a good estimate of the expected reward for each coin. This of course wouldn't be realistic if you only had access to a limited amount of trials or if you had to pay a penalty every time you toss a coin and you get a bad outcome. If bias exploration is costly, you would need to be smarter about how you carry your experiments by somehow learning on the go and making sure that you explore all possibilities.

The biased coin scenario essentially captures the Multi-Armed Bandit (MAB) problem: A repeated game where the player chooses amongst a set of available arms, and at every point of the game she can only see the outcome of the action that was chosen. Although highly restrictive, the assumption that only rewards for chosen actions can be observed is much closer to many real life situations. Think of a clinical trial where there is no way to know what the results would have been if a patient had received a different treatment; the same would apply to sponsored ad placement in a website, since it is always difficult to estimate what the clickthrough-rate would have been if we had chosen a different ad.

The MAB has been successfully used to make personalized news recommendation (http://www.research.rutgers.edu/~lihong/pub/Li10Contextual.pdf), test image placement on websites (https://developer.washingtonpost.com/pb/blog/post/2016/02/08/bandito-a-multi-armed-bandit-tool-for-content-testing/) and optimizing random allocation in clinical trials (https://arxiv.org/pdf/1507.08025.pdf). In most machine learning applications, bandit algorithms are used for making smart choices on highly dynamical settings where the pool of available options is rapidly changing and the set of actions to choose has a limited lifespan.

Arm Exploration vs Arm Exploitation

I used John Myle's implementation of Bandit Algorithms (https://github.com/johnmyleswhite/BanditsBook) to illustrate how we can tackle the biased coin problem using a MAB. For illustration purposes we will use Normally distributed rewards instead of Bernoulli trials. I am assuming that we can choose between eight arms, where each arm draws from a normal distribution with the following parameters.

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Arm 6	Arm 7	Arm 8
Mean	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Variance	2	2	2	2	2	2	2	2

Search Search

Stream

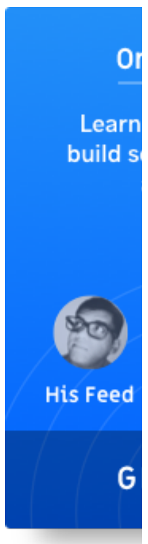
Techstars NYC 2015 - Build scalable newsfeeds & activity streams in a few hours instead of weeks.

- Get Started (https://getstream.io/get_started/#intro?fthr=blog)
- Documentation (https://getstream.io/docs/?fthr=blog)
- Features (https://getstream.io/features/?fthr=blog)
- Pricing (https://getstream.io/pricing/?fthr=blog)
- Team (https://getstream.io/team/?fthr=blog)

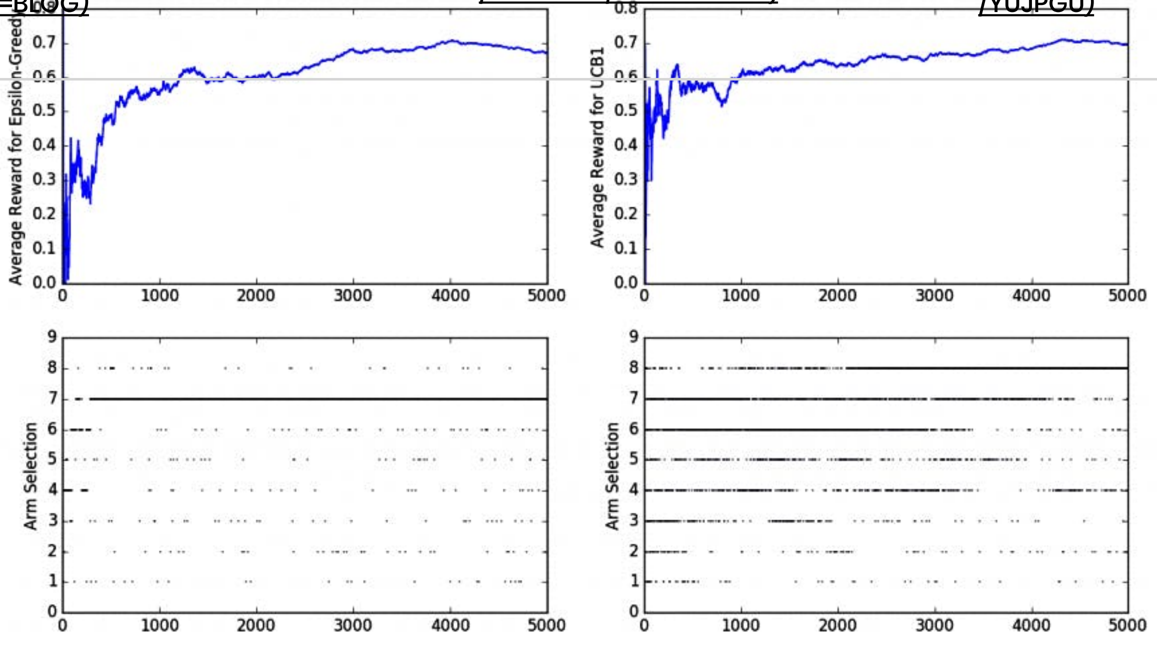
Recent Posts

- Stream raises $3 Million from Arthur Ventures (https://getstream.io/blog/stream-raises-3-million-arthur-ventures/)
- Examining Decentralized Social Networks (https://getstream.io/blog/examining-decentralized-social-networks/)
- Best Practices for Recommendation Engines (https://getstream.io/blog/best-practices-feed-personalization/)
- Based Mobile: A free UI Kit for Mobile Social Media (https://getstream.io/blog/social-media-free-ui-kit/)
- Creative Developer Marketing (https://getstream.io/blog/creative-developer-marketing/)

(https://getstream.io/get_started/)



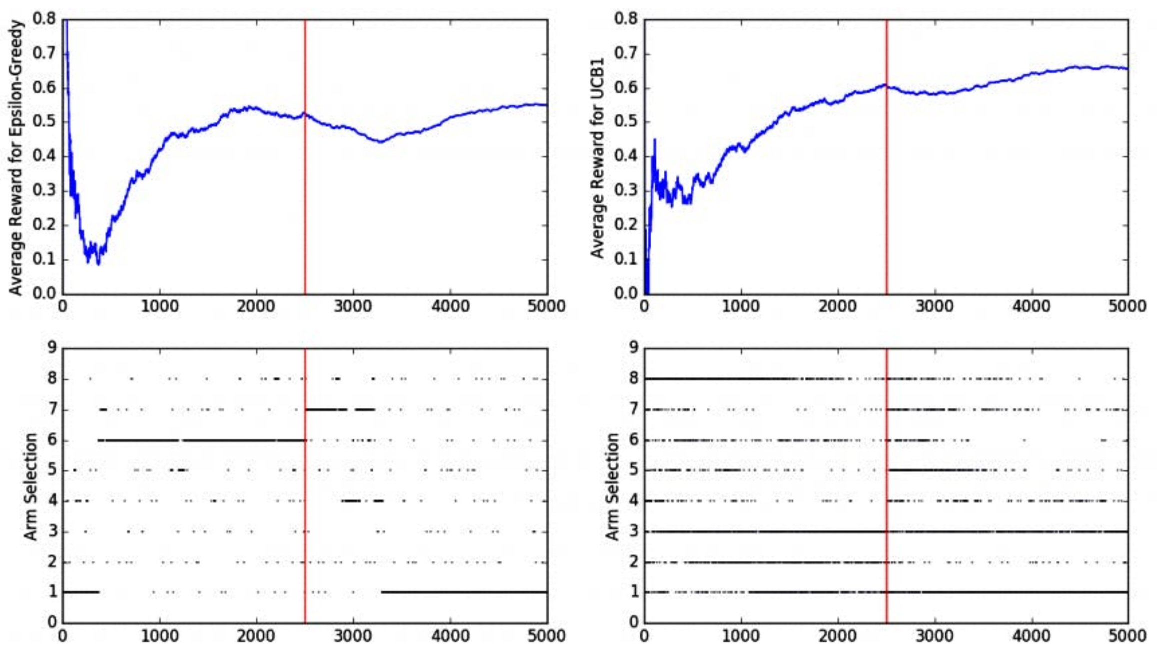
In this case, Arm 8 gives the highest reward, we want to test whether the algorithms correctly identify it. We will test the performance of two of the most well known MAB algorithms: ε-greedy and UCB1.



The image above shows the how the average reward and arm selection progresses over time for two MAB algorithms: ε-greedy and UCB1. There is an initial period of exploration and after a while the algorithms converge to their preferred strategy. In this case, ε-Greedy quickly converges to the second-best solution (Arm 7) and UCB1 slowly converges to the optimal strategy. Note how both algorithms tend to concentrate on a small subset of available arms but they never stop exploring all possible options. By keeping the exploration step alive we make sure that we are still choosing the best action but in the process we fail to exploit the optimal strategy at its fullest, this captures the *Exploration-vs-Exploitation* dilemma that is often found in bandit problems.

Why is exploration useful even when we have identified an optimal arm? In the next experiment I replicated the same scenario as before, but I have included a distributional shift at *t=2500* where the optimal choice becomes Arm 1. The shift in distributions is summarized in the following table:

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Arm 6	Arm 7	Arm 8
Mean	.8	.6	.7	.4	.3	.2	.1	.5
Variance	2	2	2	2	2	2	2	2



The graph shows how both ε-greedy and UCB1 adapt to a change in reward distribution. For this specific simulation UCB1 quickly adapted to the new optimal arm, whereas ε-greedy was a bit slower to adapt (note how the Average Reward curve dips a little bit after the change in distribution takes place). As always, we should take these results with a grain of salt and not try to draw too many conclusions from one simulation, if you are interested in experimenting with different parameters feel free to tweak the script (https://github.com/kevvzub/BanditsBook/blob/master/python/normal_demo.py) that I used to generate these results.

Contextual Bandits

In most real life applications, we have access to information than can be used to make a better decision when choosing amongst all actions in a MAB setting, this extra information, or context, is what gives Contextual Bandits their name. For the ad-placement example, having access to historical data about the user's buying habits can be highly informative of what type of products or promotions they will engage with in the future.

For really large datasets, the highly optimized Contextual Bandit Algorithms in Vowpal Wabbit are the way to go. There are a four of bandit algorithms that are ready to use in VW:

- ε-greedy: Exploit the best strategy with probability 1-epsilon, keep exploring uniformly over all the other actions with probability epsilon.
- Explore-first: Starts with a phase of pure exploration in the first *K* trials, after the exploration phase the algorithm exploits the leading strategy.
- Bagging: Different policies are trained using bootstrapping (sampling with replacement).
- Online cover: Explores all available actions by keeping only a small subset of policies active, this approach is fully described in the 2014 paper by Agarwal et. al (http://jmlr.org/proceedings/papers/v32/agarwalb14.pdf).

Bandit datasets are highly proprietary and hard to come by, in order to test some of the bandit algorithms that are implemented in VW we can use the *–cbify* option to transform any multiclass classification training set into a contextual bandit training set.

I will use a dataset that was created using the Stream Analytics (https://getstream.io/docs_analytics/) client. Each line in stream_labaled_data.vw (https://stream-machinelearning.s3.amazonaws.com/vw-files/stream_labeled_data.vw) is a VW format labeled sample with one hundred different features:

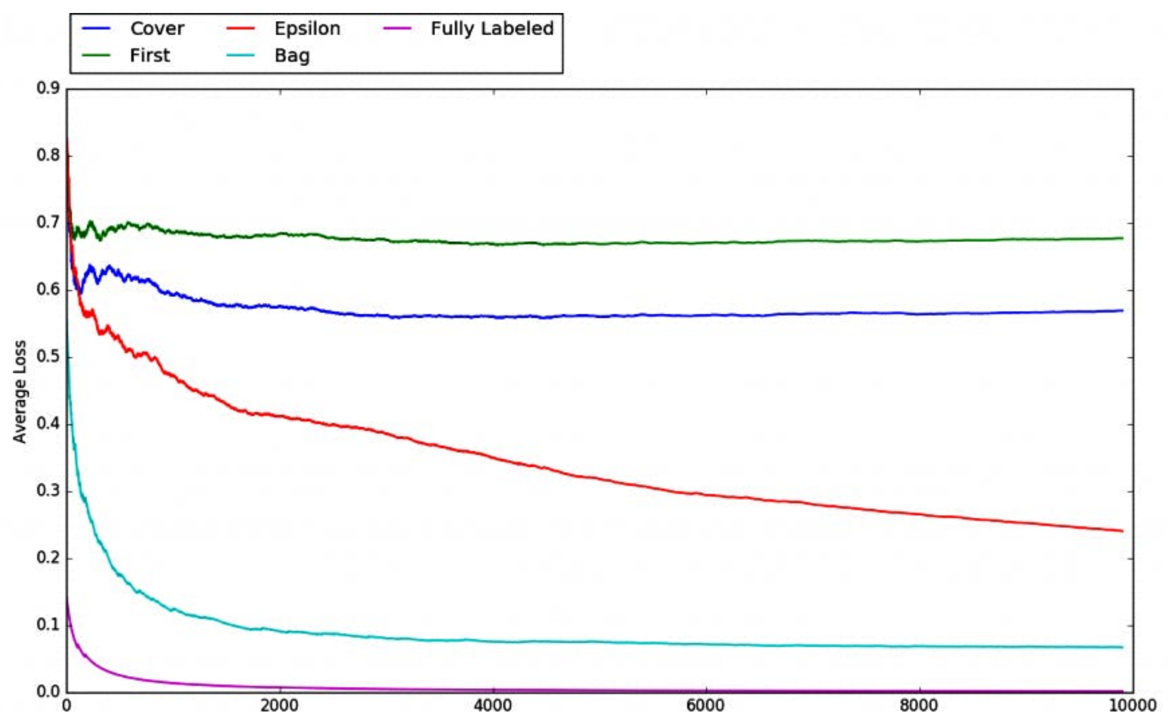
5 |features w1:5 w2:0 w3:0 w4:37 w5:6 w6:0 w7:0 w8:0 w9:0 w10:25 w11:0 w12:0 ...

These features were extracted over a large and sparse dataset using a Principal Component Analysis, where the original features were related to individual user's listening habits, but are not directly related to a genre. We have also preprocessed the dataset to maintain privacy by tokenizing each of the features. In the setting of a Contextual Bandit, we can think of features of each sample as the context and rewards are either 1 or 0 depending in wether we predict the class label correctly or not.

I have added the option *–progress 1* to all my VW calls in order to save the average loss into a file. I also trained a model using the original dataset with VW's multiclass classifier (*–oaa*) to see how the performance compares to a full information setting.

```
vw -d stream_labeled_data.vw --cbify 10 --cb_explore 10  --cover 6 --progress 1  > first.progressive  2>&1
vw -d stream_labeled_data.vw --cbify 10 --cb_explore 10 --first 100 --progress 1  > first.progressive  2>&
vw -d stream_labeled_data.vw --cbify 10 --cb_explore 10 --epsilon 0.05 --progress 1  > epsilon.progressive
vw -d stream_labeled_data.vw --cbify 10 --cb_explore 10 --bag 7 --progress 1  > first.progressive  2>&1
vw -d stream_labeled_data.vw --oaa 10 --progress 1 > labeled.progressive 2>&1
```

I made the following graph using the average validation error reported by VW:



Unsurprisingly, under full information we get a really good predictor with an average error rate of 1.5% (remember, in this case we can observe the reward associated with each arm at each step). On the other hand, —*first K* performs rather poorly but it is still an improvement from a predictor that outputs random labels (the expected error that comes from randomly choosing among ten equally likely choices should be around 0.9 if we have a training set with an equal proportion of labels). Cover converges to an error rate of 56.91% and ε-greedy gets the average loss down to 24.06%; Bag has the best performance overall with a 6.71% average loss, this is very impressive considering the fact that compared to the full information scenario it only can see one arm-reward value per trial, as opposed to the 10 arm-reward values that can be observed under the full information setting.

If you are interested in experimenting with Contextual Bandits algorithms to personalize your feeds, feel free to download the dataset and try your own models. You could do some feature engineering or create a model where each arm is associated with a different bandit classifier.

Feel free to contact us (https://getstream.io/contact/) if you are interested in learning more about how personalization (https://getstream.io/personalization/) can help improve retention and conversion in your app.

Tags: Examples (https://getstream.io/blog/tag/examples/), Feed Personalization (https://getstream.io/blog/tag/feed-personalization/), Machine Learning (https://getstream.io/blog/tag/machine-learning/)

The Activity Streams and Personalization of Google Music (https://getstream.io/blog/activity-streams-personalization-google-music/)

The Stream Firehose with SQS, Lambda, and Redis (https://getstream.io/blog/stream-firehose-sqs-lambda-redis/)

5 Comments **The Stream Blog** 1 **Login**

Recommend 2 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

djf • a month ago

Hey Dwight! I like the article, especially showing the option to turn a multi-class formatted VW line into a CB problem, evaluating many different algorithms performance to determine a winner, and providing sample data.

I'd just like to note that VW offers a lot of tuning parameters that can drastically improve performance on these algorithms (I was able to have Online Cover competitive with Bagging in your example). As it's learning online, doing multiple passes over the data is especially critical, but check out all the options here: https://github.com/JohnLang...

1 • Reply • Share ›

Kwame djf • 2 days ago

Great article, I want to point out that it may be possible to automate parameter searching through `vw-hyperopt`. See: https://github.com/JohnLang... for more details and apply to historical, recent data.

1 • Reply • Share ›

Dwight Gunning Mod djf • 18 hours ago

Absolutely. Thanks for the link - definitely a must-see for anybody taking these approaches further.

• Reply • Share ›

Ian Clarke • a year ago

Recommend you check out Thompson Sampling as an approach to MAB. It's very efficient and fairly simple to implement.

1 • Reply • Share ›

Sepehr Sadighpour • 3 months ago

Kevin, thanks for this excellent intro! Do you happen to know if --cbify would work similarly for a cost-sensitive multi-class dataset?

• Reply • Share ›

ALSO ON **THE STREAM BLOG**

Aggregated Feeds – Demystified
1 comment • a year ago•
Siddhartha R — Link this page in the docs.

Factorization Machines for Recommendation Systems
4 comments • 7 months ago•
Jianmo — Hi Kevin, thank you for the post.I have one question about why you choose get_dummies for user and item encoding? Since in vanilla matrix factorization, user …

Cabin – React & Redux Example App – Redux
1 comment • a year ago•
johnbeynon — FYI - GitHub URL for source should be https://github.com/GetStrea...

Examining Decentralized Social Networks
2 comments • 4 months ago•
Vince Hodges — Scuttlebutt has recently come to light that is not only decentralized but also works with network outages.https://www.scuttlebutt.nz/

FEATURES
- Get started (https://getstream.io/get_started/)
- Overview (https://getstream.io/features/)
- Personalization NEW (https://getstream.io

QUICK LINKS
- Blog (https://getstream.io/blog/)
- Activity Feed Design (https://getstream.io/activity-feed-design/)
- Pricing (https://getstream.io/pricing/)

COMPANY
- Team (https://getstream.io/team/)
- Careers (https://getstream.io/careers/)
- Contact Us (https://getstream.io/contact/)

BUILD SOCIAL NETWORKS (https://getstream.io/build/social-networks/) WITH
- Node JS (https://getstream.io/build/social-networks/node/)
- Ruby (https://getstream.io/build/social-networks/ruby/)
- Golang (https://getstream.io/build/social-networks/golang/)
- Python (https://getstream.io/build/social-networks/python/)
- PHP (https://getstream.io/build/social-networks/php/)
- Scala (https://getstream.io/build/social-networks/scala/)

/personalization/)

- Customers (https://getstream.io/case_studies/)
- Enterprise (https://getstream.io/enterprise/)

- Docs (https://getstream.io/docs/)
- Based Feeds UI Kit (https://getstream.io/based-feed-ui-kit-sketch/)
- Based Mobile App UI Kit (https://getstream.io/based-mobile-ui-kit-sketch/)

- System Status (http://status.getstream.io/)
- Policies (https://getstream.io/legal/)

- Java (https://getstream.io/build/social-networks/java/)
- Django (https://getstream.io/build/social-networks/django/)

- Laravel (https://getstream.io/build/social-networks/laravel/)
- Rails (https://getstream.io/build/social-networks/rails/)

- Meteor JS (https://getstream.io/build/social-networks/meteorjs/)
- .NET (https://getstream.io/build/social-networks/net/)

(https://twitter.com/getstream_io)
(https://github.com/GetStream)
(https://plus.google.com/107168613155747146780)