Online Contextual Bandits in e-commerce domain

Jakub Bahyl

Exponea AI Department

Problem definition

Implement **contextual bandit** in production quality in order to **choose the best content** for given user browsing on the website.

Introduction

Personalization of user experience in the e-commerce domain can be boosted using several techniques including recommender systems, user-based web optimizations or multi-armed bandits for choosing the best content to display.

Since classical bandit picks the best option based on historical rewards, it can't use contextual data and thus personalize for a user. Therefore we approach the problem with a contextual bandit (CB) with an ability to learn from the state (context) of a given environment. Additionally, by implementing a policy layer for exploration-exploitation trade-off, we not only train CB to optimize decision based on previous observations but also to personalize chosen content for every situation.

Approach

Inspired by Microsoft's published white-paper [1] we used an open sourced machine learning platform *Vowpal Wabbit*, developed by Yahoo Research [2] that implements online training algorithms for CB. We designed the architecture as a simplified reinforcement learning problem so that our model incrementally learns on past decisions and their corresponding rewards.

Formally, the goal of CB learning can be defined as a minimization problem of cumulative regret:

$$R(T) \triangleq \arg\min_{a_t \in \mathcal{A}} \sum_{t=1}^{T} \left(E[r_{a_t^*}] - E[r_{a_t}] \right)$$

Within our implementation, we use a multi-policy learning algorithm for online learning [3]. As an exploration strategy, we took the ε-greedy approach. Rewards are considered linear to the context vector:

$$r_{a_t} = \vec{\mathbf{x}}_{a_t} \theta^{\intercal} + \varepsilon_t \,,$$

where θ represents the user's preference.

Implementation

We used Google Cloud Platform as a leading framework for our ecosystem. The high-level architecture of our service depicted in Figure 1 implements three steps of the cycle:

1. **Client interface** - exposed API with deployed current best learner that makes decisions, performs exploration and serves back recommended actions based on requests containing context.
2. **Rewards Joiner** - component responsible for joining served decisions with corresponding rewards.
3. **Online Learner** - regular job performing policy learning and re-deploying newest model back to the app.

After request with a context has been sent to our API, we transform it to structural feature vector and let CB calculate expected values for each possible decision. Since we set ε-greedy exploration factor to $\varepsilon = 0.05$ we serve sorted predictions in 95% cases and shuffled in 5% cases. In order to train on joined decision-reward pairs, we store each context request, decision ever made and most importantly, the rewards streamed by the client after a user interacted with the recommended content.

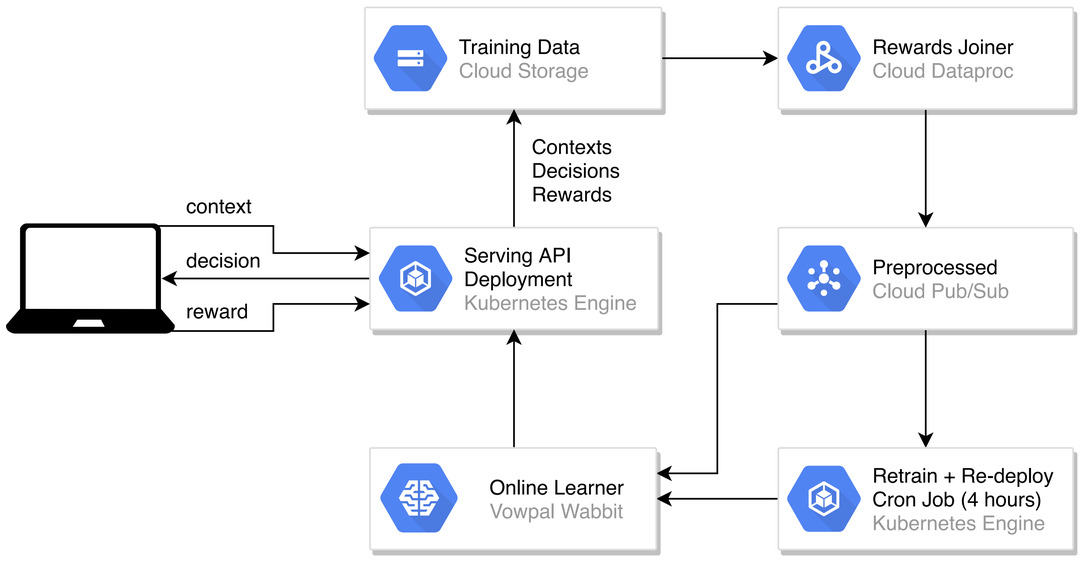

Figure 1: High-level skeleton of implemented architecture developed within Google Cloud Platform

Every four hours we take the latest model and train it on new joined data - in form of input tuples: $(\vec{\mathbf{x}}, a, c, p)$ meaning context vector, action, cost, and probability. After that, we perform the rolling update.

Business use-case

Our chosen client has a motivation to serve personalized offers to users browsing the website after querying their search engine. Client's back-end system immediately request our API with the context (cookie and query field) in the body. After we serve back the ordered list of recommendations, the client filter out not relevant offers (time-limited, etc.) and renders the best one to the user's screen. After the user finishes the session, the client will send us a post request with the reward for given rendered offer. The reward is binary and tracks whether the user clicked on given offer or not.

Using this pipeline we want to deliver real-time recommendations that are personalized and can change in time as users naturally change their preference. Our main expectation is to achieve higher conversion rates and stronger retention of customers than other common recommender systems could provide.

Conclusions

We believe contextual bandits can be applied as one of the best commercial tools for web personalization. Our successful implementation of this live system gave us many learnings on how to maintain such an ecosystem. Moreover, by tracking the behaviour of CB we raised many open questions we would like to address:

- Is our architecture flexible enough on preference change of users?
- How often should we completely re-train CB offline? Using which algorithm?
- Which feature engineering steps would lead to the higher precision of predictions?

References

[1] Microsoft Research.
Multiworld Testing Decision Service: A System for Experimentation, Learning, And Decision-Making.
October 2016.

[2] Yahoo! Labs.
Vowpal Wabbit - fast machine learning framework.
https://github.com/VowpalWabbit/vowpal_wabbit.

[3] John Langford Miroslav DudÃık.
Doubly Robust Policy Evaluation and Learning.
arXiv, October 2011.

Contact Information

Exponea webpage: https://exponea.com/
Personal email: jakub.bahyl@exponea.com

VOWPAL WABBIT

Google Cloud Platform