Decription of 2nd place solution for Criteo Ad Placement Challenge

Mikhail Trofimov

December 3, 2017

1 Introduction

This document describes the solution for the Criteo Ad Placement challenge hosted on crowdAI platform. A dataset consists of 10+M examples, each example is a set of candidates (number of candidates may vary, the mode is 11), observation for a chosen index (click or no-click) and the probability of this choice. The task was to construct a policy that will maximize IPS (Inverse Propensity Score).

In short, my solution is just a linear model trained over subsampled one-hot encoded data.

2 Data preprocessing

2.1 Sharding

Firstly, data is sharded (split into chunks for simplifying further processing).

The train set is sharded over ID to 16 shards (using ID%16 as shard number).

The test set is split into 8 quasi-equal shards sequentially (since it's important to save the order in submission file).

A number of shards can be increased in order to reduce memory footprint.

I use pypy for this processing stage since it's much faster than pure python.

This stage takes a few minutes to complete.

2.2 Parsing to dicts

At this stage, I parse samples using the code provided in a starter kit. Each shard can be processed independently (in parallel), but this will increase memory footprint. Using my hardware I was able to process 2 chunks at the same time. This can be controlled as $-j$ option of **parallel** utility.

This stage takes about an hour to complete.

2.3 Converting to sparse matrices

In order to use TensorFlow, we need to convert samples in sparse matrices. I wrote a simple converter which transforms an array of pythons' dictionary into a sparse matrix in COO format. Also, in order to prevent data leakage, candidates are shuffled inside a single sample.

Again, this stage was applied to every shard. Internally, it uses parallelization at the level of single shard – that's why I use $-j1$ option.

This stage takes about an hour to complete.

3 Model description

3.1 Preparing batches and subsampling data

I use shards 0–7 for training, 8–11 for validation and 12–15 for a holdout.

For training, I subsample filter only samples with 11 candidates since they cover more than half of all dataset and it's convenient to work with a fixed number of candidates.

All click and 3 times more non-clicked objects were sampled (note that this approach effectively uses only 20% of data and ignore 80% of non-clicks). Objects are shuffled and split into batches of size 512.

For validation and holdout, I select only clicks with 11 candidates, since for calculating IPS we need only information of clicks and a total number of objects (or just **ctr**).

This stage takes a few minutes to complete.

3.2 Model

The underlying model is a linear. There are 2 main points: probability clipping and costs.

I restore probabilities for non-clicked samples (they are 10 times smaller than for clicks) and manually set cost -1.0 for clicks and -0.1 for non-clicks. Also, all probabilities are clipped to $[0.3, 1]$.

As a loss function, I use IPS with manual costs and L2 penalty for output logits (with coefficient $1e-4$).

As optimizator I use Adam with **learning_rate** $= 0.003$ for 50 epoches.

This approach gives $IPS : 54.37$ on the leaderboard.

With post-processing heuristic $\text{logits}_{\text{scaled}} = \text{logits} * 20$ (which is quite similar to assign probability 1.0 to the argmax) the score goes to $IPS : 54.55$.

This stage takes about 15-20 to complete.

4 Hardware

This code was executed on a machine with 64G RAM, i7-6800K core and NVIDIA GTX 1080 running under Linux Mint 18.1.

It should also work run without GPU and with less amount of RAM (probably number of shards should be increased in this case).

5 Software

- pypy (5.1.2)
- python (3.5.2)
- notebook (5.2.1)
- numpy (1.13.3)
- scipy (1.0.0)
- tensorflow-gpu (1.4.0)
- tensorflow-tensorboard (0.4.0rc2)
- tqdm (4.19.4)

6 Problems and what didn't work

In this section, I briefly describe ideas and approaches which I tried and they didn't work. I don't know the exact reason for that – maybe it's a property of task itself, maybe just a bug in my code.

6.1 Propensity overfitting and validation

The problem is that some objects have very low probability (less than $1e-5$) so they are contributing extremely high in total IPS. Because of this problem, 1 right prediction over 1M samples can give you 20% of total score ($53- > 62$). This this the reason why cross-validation approach doesn't work for me without propensity clipping. I decided to use $1e-3$ as threshold – after this, my local scores gets close to the leaderboard.

6.2 Costs and 2 perfect solutions

I noticed that costs for clicks/non-clicks $\{-1, -0.1\}$ produces almost the same IPS as $\{-1, 0.1\}$ and $\{-1, -1\}$. If we think carefully about the task, we can realize that there are 2 perfect solutions:

- one perfectly selects the candidate **which will be clicked**.
- second perfectly selects the candidate **which will be chosen to show to the user**.

They solve very different tasks, but both will produce nice IPS. My hypothesis is that this is the reason for such "robustness" to costs and, probably, learning for the second solution is more benefitical for IPS (but, obviously, not for real-life applications)

6.3 More complicated models

I tries several models, like

- mixture of linear experts
- factorization machines
- MLP

- Boosted trees over SVD-transformed data

Despite some of them produce cross-validation score slightly better then linear model, they perform worse on the leaderboard.

6.4 More complicated loss functions

I tried to use self-normalization approach (via normalizing to S estimated over a batch of 512 samples) and POEM-like (via adding std of r (in term of my code) to the loss function). The latest help with convergence and slightly improve local cross-validation score but doesn't improve the leaderboard score.

6.5 Black-box optimization

I tried to optimize IPS directly with an evolution strategy optimization technique. It tends to overfit hardly, but even with propensity clipping works worse than SGD-based learning in my case.

6.6 Baseline from the reference paper

I tried to reproduce result from the reference paper ("Large-scale Validation of Counterfactual Learning Methods: A Test-Bed") using **vw**, and it produces a good local IPS (before propensity clipping). But on the leaderboard, it performs worse with IPS around 53.