Propensity Weighted Click Prediction

Nayun Xu

***Abstract*—This work focuses on proposing and evaluating a new method to solve click prediction problems based on logged bandit feedback. Basically the idea is to use propensity weighted regularization terms to reduce the biasness of the training objective loss resulted from the biasness of logged bandit feedback.**

I. Introduction

Traditional supervised learning is based on the assumption that the training set used to train the supervised model is identically distributed as the problem space. And also, it requires the input to be full-information, i.e, each input feature should have a corresponding ground truth label. However, in many cases, like clicking data collected from search engines and recommendation systems, such data naturally is not full-information, and human annotations are not necessarily accurate. Such data will also be affected by the policy used to present results to users, i.e collected data is not identically distributed as the problem space but affected by the policy. Such data is called logged bandit feedback. How to address problems based on such feedback is worth investigating. Under the direction of Professor Thorsten Joachims, and with the help of Jinning Li, I proposed a simple method to use propensity-weighted techniques to make the objective loss more unbiased. The method has several variants and is evaluated on a Criteo ad-placement challenge launched on CrowdAI [1], and the forest cover type dataset [2]. This work is based on several previous work [3] [4] [5].

II. Problem Description

The problem is based on logged bandit feedback. To formalize the logged bandit feedback, there is a policy π_0 which takes inputs from a set of input features $x \sim X$, makes an action $y \sim \pi_0(Y|x)$, and receives a feedback $\delta(x, y)$. The propensity p is equal to the probability $\pi_0(y|x)$. Then the dataset is $\{x_i, y_i, p_i, \delta_i\}$. And in this click prediction setting, each input x_i is called an impression. x_i consists of several candidates, and each candidate is a blended feature vector consists of document content, user information and the context. Each action y_i is to choose one of the candidates in the given impression x_i to present to the user. Feedback(Reward) δ_i is whether the presented action y_i is clicked or not, if clicked, $\delta_i = 1$ otherwise 0. p_i is the propensity that pi_0 takes y_i as the action given the impression x_i.

The objective is to derive a model f that takes an impression x_i, and can predict how likely each candidate in this impression will be clicked. This model can also be viewed as a policy π, where $\pi(Y|x_i)$ is calibrated by $f(Y|x_i)$. A simple way to calibrate $f(Y|x_i)$ is to use a softmax function over all the feasible actions.

Because the data is biased and the feedback is not full-information, the evaluation cannot simply apply traditional metrics. The evaluation metric used in this work is the IPS estimator. For each impression, $f(\cdot)$ will give predictions to each candidates. For impression i, take the predictions and calibrate them into propensity $\pi(Y|x_i)$. Then the propensity $\hat{p}_i$for the action y_i of the policy to be evaluated will be $\pi(y_i|x_i)$ .Then the IPS score for this impression is $\frac{\hat{p}_i}{p_i}\delta_i$. The total IPS score of the evaluation set is $\frac{1}{n}\sum_i^n \frac{\hat{p}_i}{p_i}\delta_i$. This estimator is an unbiased estimator the the reward/loss.

Advisor: Professor Thorsten Joachims; Co-worker: Jining Li

III. Methodology

A. Overview

The model is based on multiple-layer perceptrons. Such simple structure makes it easier to analyze the experiments. The idea is to involve the logging policy's propensity and the current policy's propensity we are trying to optimize, and make the objective loss closer to the expectation loss on the problem space.

B. Notations

x_i is the input blended features of impression i. x_i contains n_i candidates. Each candidate is a m-dimensional vector.

y_i is the action logged by the logging policy.

p_i is logging policy's propensity of the action.

δ_i is the feedback $\delta(x_i, y_i)$ logged by the logging policy.

M is the clipping hyper parameter. Determined by grid search.

$f(y|x)$ is not a distribution. $f(\cdot)$ is a score function for each candidate given x_i

CE is the cross entropy loss function.

C. Baseline and Strong Baseline

Network: 2-layer MLP(dense-relu-dense-sigmoid)

Objective function: $\sum_i \min(M, \frac{1}{p_i})CE(f(y_i|x_i), label_i)$

Note: For baseline model M equals 1. For strong baseline model, do grid search to pick M.

Input: $\{x_{i,y_i}, y_i, label_i\}$ ($label_i$ is derived from δ_i)

D. Advanced version 1

Network: 2-layer MLP(dense-relu-dense-sigmoid)

Objective function: $\sum_i \min(M, \frac{\hat{p}_i}{p_i})CE(f(y_i|x_i), label_i)$

Intermediate result: $\hat{p}_i = softmax(f(y|x_i))_{y_i}$

Input: $\{x_i, y_i, p_i, label_i\}$ ($label_i$ is derived from δ_i)

E. *Advanced version 2*

Network: 2-layer MLP(dense-relu-dense-sigmoid)
Objective function: $\sum_i \min(M, \frac{\hat{p}_i}{p_i}) CE(f(y_i|x_i), label_i)$
Intermediate result: $\hat{p}_i = softmax(f(y|x_i))_{y_i}$
Note: This $softmax(\cdot)$ is detached from the computation graph, which means $\hat{p}_i$ will be treated as constant rather than a function in the objective function.
Input: $\{x_i, y_i, p_i, label_i\}$ ($label_i$ is derived from δ_i)

F. *Advanced version 3*

Network: 2-layer MLP(dense-relu-dense-sigmoid)
Objective function: $\sum_i \min(M, \frac{\hat{p}_i}{p_i}) CE(f(y_i|x_i), label_i)$
Intermediate result: $\hat{p}_i = softmax(f(y|x_i))_{y_i}$
Note: This $softmax(\cdot)$ is calculated outside the computation graph after each training epoch, which means $\hat{p}_i$ will be treated as constant rather than a function in the objective function.
Input: $\{x_i, y_i, p_i, label_i\}$ ($label_i$ is derived from δ_i)

G. *Differences among three advanced methods*

The three methods are described using pseudo codes in Algorithm 1, 2, 3.

Algorithm 1 Advanced version 1
```
for $i$ in range($epochs$) do
    for $j$ in range($batches$) do
        $X, y, p, \delta$ = $DataBatch[j]$
        $logits = \text{forward}(X)$
        $\hat{p} = softmax(logits)[y]$
        $loss = \text{Loss}(\hat{p}, p, logits, \delta)$
        Backward($loss$)
    end for
end for
```

Algorithm 2 Advanced version 2
```
for $i$ in range($epochs$) do
    for $j$ in range($batches$) do
        $X, y, p, \delta$ = $DataBatch[j]$
        $logits = \text{forward}(X)$
        $\hat{p} = softmax(logits)[y]$
        Detach($\hat{p}$)
        $loss = \text{Loss}(\hat{p}, p, logits, \delta)$
        Backward($loss$)
    end for
end for
```

IV. Experiments and Results

All the experiment codes are written with tensorflow [6]. The code of covertype experiments can be accessed on https://github.com/Nerer/Propensity-weighted-click-prediction.

Algorithm 3 Advanced version 3
```
Initialize $\hat{p}$ in $DataBatch$ to 1.
for $i$ in range($epochs$) do
    for $j$ in range($batches$) do
        $X, y, p, \hat{p}, \delta$ = $DataBatch[j]$
        $logits = \text{forward}(X)$
        $loss = \text{Loss}(\hat{p}, p, logits, \delta)$
        Backward($loss$)
    end for
    update $\hat{p}$ with current model.
end for
```

A. *Criteo dataset*

In the dataset, each impression is represented by M lines where M is the number of candidate ads. Each line has feature information for every other candidate ad. The feature consists of user, context and candidate information. The logged feedback is either clicked or non-clicked.
Each feature is a 74000-dimensional one-hot vector. Training set contains about 14M impressions. Test set contains about 7M impressions.
Training and testing are both time-consuming. The result shows some unreasonable properties. So I build a new criteo-like dataset based on covertype dataset to do further experiments and evaluations.

B. *Original Covertype dataset*

A dataset for forest cover type prediction. Each feature contains 12 measures, but 54 columns of data(10 quantitative variables, 4 binary, wilderness areas and 40 binary, soil type variables).There are 7 categories of cover type. Total instance number is 581,012.

It's a alternative for Criteo because it has smaller size, fewer dimensions of input features and the category number is appropriate.

C. *Crieo-like Covertype dataset*

Train a weak classifier $f(y|x)$ using 1% of the full information data.

Using the weak classifier as a logging policy, go through the full information data, to give prediction likelihood for every category given a input feature, i.e., given a input feature, predict the probability that it belongs to category 0, category 1, ..., category 6(These are propensities). Then use the predicted probability to sample actions. That is for x_i, calibrate $f(y|x_i)$ using some function $h(\cdot)$, then sample actions by $y \sim h(f(y|x_i))$. Denote the action as $\hat{y}$, the propensity for the action is $h(f(y|x_i))_{\hat{y}}$. For each (feature, label, action) tuple, construct an impression by blending each category and the feature to construct candidates. My blending method is to multiply the feature length by number of categories. And put the action candidate on the first row. For example, if the feature is [1, 2], label is 1(0 based), action is 2, propensity is 0.5 then the impression would be:

candidates x_i: [[0,0,0,0,1,2],[1,2,0,0,0,0],[0,0,1,2,0,0]]
propensity p_i: 0.5

reward δ_i: 0

In experiments, I tried two different $h(\cdot)$ function, a softmax function and a normalized $\frac{1}{rank}$ weight, in which the rank is a descending order by $f(y|x)$. Finally I choose the softmax over normalized $\frac{1}{rank}$ weight, because it makes the propensity distribution smoother.

D. *Result on Criteo*

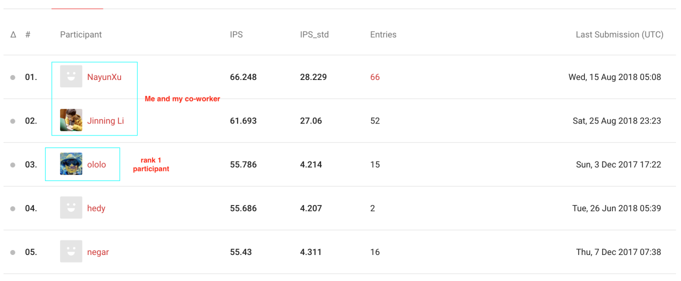
The model is evaluated is evaluated by the crowdai challenge 'NIPS '17 Workshop: Criteo Ad Placement Challenge'.

I tried baseline, strong baseline and advanced version 3 on this dataset. I haven't tried all the methods due to limited time.

The result(Fig 1)seems good considering it achieves a much higher score than the rank 1 participant.

I record the strong baseline model's score on the challenge in Fig 2. It achieves the highest point of 66.25.

And the advanced version 3 get a result of 60.069.



Δ	#	Participant	IPS	IPS_std	Entries	Last Submission (UTC)
●	01.	NayunXu	66.248	28.229	66	Wed, 15 Aug 2018 05:08
●	02.	Jinning Li	61.693	27.06	52	Sat, 25 Aug 2018 23:23
●	03.	ololo	55.786	4.214	15	Sun, 3 Dec 2017 17:22
●	04.	hedy	55.686	4.207	2	Tue, 26 Jun 2018 05:39
●	05.	negar	55.43	4.311	16	Thu, 7 Dec 2017 07:38

Me and my co-worker

rank 1 participant

Fig. 1. Leader board on crowdai

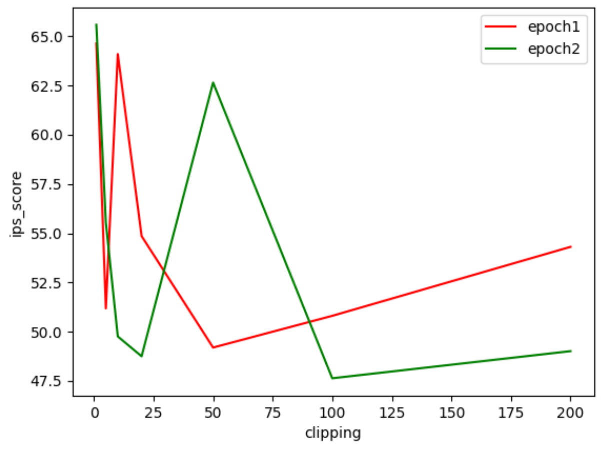


Fig. 2. Result of strong baseline

E. *Problems of Criteo dataset experiments*

During the experiments, some situations and results are strange.

- The order of data batches influences the evaluation result largely. Same model can have different performance from 66 to 55 when the data feeding order is different.
- The model gets easily overfitting in 2 epochs. Validation set and test set show very different evaluation performance. IPS score can be 100 in the validation set but under 60 in the test set.
- $\frac{1}{n}\sum_i \frac{\hat{p}_i}{p_i} > 1$ on validation set. In my experiment, training data is randomly divided into 1/2 training, 1/4 validation A and 1/4 validation B. The result for two validation sets are 1.99 and 1.97. And the expectation of $\frac{1}{n}\sum_i \frac{\hat{p}_i}{p_i}$ is 1(See [3]) if the training set and validation sets are independent. So the dataset may have some bugs.
- The test set is not public. So no further study can be done in the test data. So I tried to build a new Criteo-like dataset based on the cover type dataset.

subsectionResult on Covertype I tried baseline, strong baseline, and all three advanced methods.

After some experiments, the $h(\cdot)$ function used in the data construction is determined to be normalized $\frac{1}{rank}$. It's smoother and can be used to sample more passes easily.

The propensity distribution of the new dataset is in Fig 3. My logging policy has ips score of 0.608.

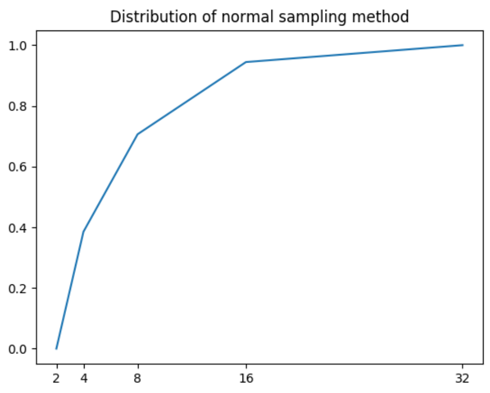


Fig. 3. Propensity distribution

Using the original cover type dataset, my MLP classifier gets 0.776.

First, I run the logging policy for one pass, to construct a criteo-like dataset, and conduct experiments on it.

Then I run the logging policy for 5 passes, to construct a larger dataset, and conduct experiments on it.

The results of 1-pass data and 5-pass data are in Fig 4 and 5. The version 2 is not evaluated on 1-pass data. In the 5-pass result, label 'v3h' means to use 'hardmax' to calibrate current policy logits into propensities, i.e the max value is calibrated to 1 and others become 0.

F. *Problems of Covertype dataset experiments*

There exist minor categories in the dataset. According to my data construction method, the category information is included in the features. And this causes problem to advanced version 1.

The version 1 is not included in the result because it goes away from the logging actions, which means the model tends to give most propensity to the minor category when the true

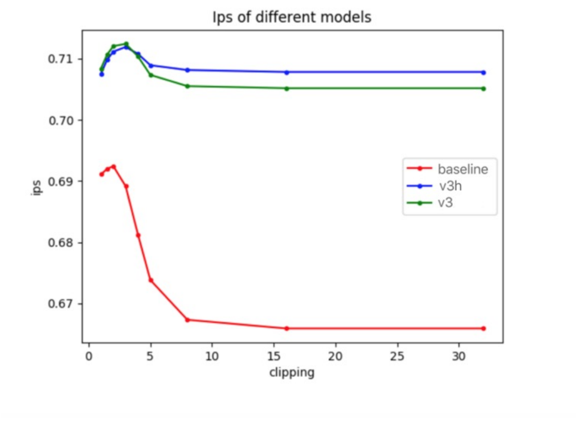


Fig. 4. Result on 1-pass data

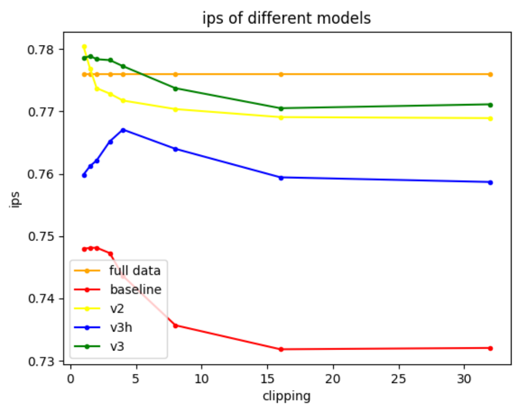


Fig. 5. Result on 5-pass data

label is not the minor one, otherwise it gives propensity to other candidates. Then in most impressions $\hat{p}_i$ goes to nearly 0, which results in that objective loss keeps shrinking but the evaluation result is very bad.

V. Tricks and skills

I have learned some tricks and skills during this project.

- Tensorflow supports GPU acceleration of sparse tensor operations. But PyTorch doesn't support the feature now. The sparse tensor can reduce the memory usage and speed up the calculation significantly when input is sparse.
- PyTorchs multi-thread dataloader is not very stable when the memory consumption is close to the limit.
- Splitting files into pieces is a good solution for large input file. And the experience of dealing with large scale data.
- Use matrix dot to sum over one axis with a list of different step lengths(It cannot be done by reshape because the step lenghts are different).
- Advanced numpy skills.
- A better taste for probability and expectation in the machine learning area.
- Better ability to express ideas in English.
- Making use of the validation set.

VI. Conclusion and Future work

My work focuses on the modifying the objective loss to approximate the expectation loss in the problem space. And it shows a good performance compared to baselines and the challenge leader board.

Because of some unreliable factors of the Criteo dataset, the result evaluated on the Covertype dataset is more solid. However, the data construction method causes the problem in advanced version 1. And there remain some techniques like variance regularization that I haven't well combined into my experiments.

The evaluation on the Criteo dataset can be launched again in a more well-organized way - fix the dataloader order, tune the weight decay parameter smaller, and see the result.

The data construction method may be improved in some way that can avoid the problem in advanced version1 method.

References

[1] D. Lefortier, A. Swaminathan, X. Gu, T. Joachims, and M. de Rijke, "Large-scale validation of counterfactual learning methods: A test-bed," *arXiv preprint arXiv:1612.00367*, 2016.
[2] C. Blake, "Uci repository of machine learning databases," *http://www. ics. uci. edu/˜ mlearn/MLRepository. html*, 1998.
[3] T. Joachims, A. Swaminathan, and M. de Rijke, "Deep learning with logged bandit feedback," 2018.
[4] A. Swaminathan and T. Joachims, "Counterfactual risk minimization: Learning from logged bandit feedback," in *International Conference on Machine Learning*, 2015, pp. 814–823.
[5] G. W. Imbens and D. B. Rubin, *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.
[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.