

NIPS '17 Workshop: Criteo Ad Placement Challenge

Counterfactual policy learning for display advertising

Mikhail Trofimov

Структура рассказа

- Постановка задачи вообще
- Датасет
- Мой подход
- Что попробовал, но не заработало
- Заключение

Введение

Criteo: 2014, Kaggle





questwalk
1st place

3 Idiots' Solution & LIBFM

posted in [Display Advertising Challenge](#) 3 years ago

43

Dear all,

We have prepared our codes and documents.
















For the codes, please see [here](#); and for the documents, please see [here](#).

In this competition and [Amazon CTK companion](#), we find field-aware factorization machines (LFMF) are powerful models. We have implemented it in the package [LIBFM](#).

Your comments are very welcome. Thanks!

-- 3 Idiots

Details

Overview	Data	Discussion	Leaderboard	Rules	Late Submission		
#	Sub	Team Name	Kernel	Team Members	Score @	Entr...	Lat
1	—	3 Idiots			0.44463	13	3y
2	—	Michael Jahner and Icar...			0.44527	61	3y
3	—	belle			0.44610	67	3y
4	—	Julian de Wit			0.44659	63	3y
5	—	Click here to win a mill...			0.44665	85	3y
6	—	machine learner			0.44738	55	3y
7	—	Autobots			0.44760	80	3y
8	—	BigDataScience			0.44783	33	3y
9	—	brigitte			0.44810	95	3y
10	—	CworldExperiments (KL...			0.44824	42	3y
11	—	caowork			0.44867	53	3y
12	—	Owse			0.44871	25	3y
13	—	idlo_speculation			0.44882	49	3y
14	—	Silegram			0.44884	99	3y
15	—	Gioia & Luca			0.44950	89	3y

УЛУЧШАТЬ МОДЕЛЬ



УЛУЧШАТЬ ДАННЫЕ

Проблема обратной связи

- Изначальные данные определяют модель
- Модель определяет последующие данные



Неполнота информации

Бандитский сигнал

-	?	?
?	-	?
?	?	-
+	?	?

Обычная классификация

-	+	-
+	-	-
-	+	-
+	-	-

Можно придумать подходящий лосс, но проблема обратной связи?

Exploration / Exploitation

2 крайности сбора данных:

- исключительно по модели (exploitation)
- на рандоме (exploration)

Можно их балансировать (eps-greedy).

Главное — наличие рандомизации и возможность “отойти” от модели.

Но что измерять на таком датасете?

			p_i
-	?	?	0.96
?	-	?	0.83
?	?	-	0.05
+	?	?	0.63

Inverse Propensity Score

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n} \sum_{i=1}^n \delta_i \frac{\pi_w(y_i | x_i)}{\pi_0(y_i | x_i)}$$

Несмещенная оценка!

Позволяет по историческим данным оценить эффективность стратегии.

Отсюда и название — counterfactual

			p_i
0.1 2	0.3 4	0.5 4	0.96
0.0 3	0.6 7	0.3	0.83
0.2 7	0.3 5	0.3 8	0.05
0.9	0.0 2	0.0 8	0.63

Связь методов обучения с бандитским сигналом

Тип	Название задачи	Частный случай для
Онлайн обучение	Contextual bandits	On-policy RL
Оффлайн обучение	Counterfactual Learning	Off-policy RL

Связь методов обучения с бандитским сигналом

Поддержка этого есть в VW из коробки

Тип	Название задачи	Частный случай для
Онлайн обучение	Contextual bandits	On-policy RL
Оффлайн обучение	Counterfactual Learning	Off-policy RL

Вот тут находится конкурс от Criteo

Краткий итог введения

Эффект обратной связи при сборе датасета можно смягчить, если:

- добавить рандомизацию
- сохранять вероятности p_i исходов
- использовать $1 / p_i$ как вес примера при обучении

Более детально можно посмотреть в [ICML 2017 Tutorial on Real World Interactive Learning](#)

Конкурс

Данные

13M примеров в трейне, 7M в тесте

Пример:

- набор кандидатов (2-50, 11 покрывает >70% выборки)
- какой кандидат был выбран (его индекс)
- какая у него была вероятность (p_i)
- был ли клик

Кандидат: one-hot encoded, dim = 74k, nnz = ~20

Весь трейн: ~174M строк, 25GB в формате libsvm в виде текста

Данные

[illegible]

CTR: 0.05 (но в описании указано, что негативных примеров в 10 раз меньше, чем в реальной жизни)

Метрика

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n} \sum_{i=1}^n \delta_i \frac{\pi_w(y_i | x_i)}{\pi_0(y_i | x_i)} \quad \times 10000$$

δ — был ли клик, $\{0, 1\}$

Сумма по всем объектам (даже тем, которые гипотетические!)


```

23 num_positive_instances = 0
24 num_negative_instances = 0
25
26 #For each row
27 prediction_stochastic_predictor = np.zeros(num_instances, dtype = np.float)
28 prediction_stochastic_denominator = np.zeros(num_instances, dtype = np.float)
29
30 impression_counter = 0
31 for idx, _impression in enumerate(valid_data):
32     # TODO: Add Validation
33     predictor = nonlinearpredictor
34     if _impression['id'] != predictor['id']:
35         raise Exception("prediction_id [%d] doesn't match the corresponding %d"
36
37 scores = predictor['scores']
38
39 label = _impression['act']
40 properties = _impression['properties']
41 num_candidates = len(_impression['candidates'])
42
43 rectified_label = 0
44
45 if label == pos_label:
46     rectified_label = 1
47     num_positive_instances += 1
48 elif label == neg_label:
49     num_negative_instances += 1
50 else:
51     raise Exception("Unknown test label for impression_id [%d].format(idx)
52
53 if label == pos_label:
54     log_weight = 1.0
55 elif label == neg_label:
56     log_weight = 0.8
57 else:
58     raise Exception("Unknown test label for impression_id [%d].format(idx)
59
60 #For deterministic action
61 best_score = np.max(scores)
62 best_indexes = np.argwhere(scores == best_score).flatten()
63
64 #For stochastic policy
65 error_logger_action = True
66 score_normalizer = 2.0
67
68 scores_with_offset = scores - best_score
69 prob_scores = np.exp(scores_with_offset)
70
71 score_normalizer = np.sum(prob_scores)
72
73 logged_action_index = 0
74 if split_train:
75     logged_action_index = utils.compute_integral_hash(_impression['id'], s
76
77 score_logged_action = prob_scores[logged_action_index]
78
79 prediction_stochastic_weight = 1.0 * score_logged_action / score_normaliz
80 predictor_stochastic_predictor[idx] = rectified_label * prediction_stoch
81 predictor_stochastic_denominator[idx] = prediction_stochastic_weight
82
83 impression_counter += 1 adding this as _idx is not available out of this
84 if _idx % 100 == 0:
85     if context and factory_utils:
86         factory_utils.update_progress(context, _idx=100.0/max_instance
87     if debug: print(' ', end='')
88
89 good_data.close()
90 predictions.close()
91

```

VS

x2

```

def IP_score(y, p, ctr):
    denom = len(p)*(1 + (1-ctr) / ctr +10)
    return np.sum( y / p) / denom * 10000

def IPS_from_logits(logits, ctr):
    x = logits - np.max(logits, axis=1, keepdims=True)
    x = np.exp(x)
    x = x / np.sum(x, axis=1, keepdims=True)
    sliced_probs = np.choose(ids, x.T)
    return IP_score(sliced_probs, probs, ctr)

```

Предобработка данных

- Пошардировать (16 шардов)
- Пересобирать в словарики
- Перемешать кандидатов (изначально 0й -- это тагрет)
- Пересобирать в спарс-матрицы
- Экспортировать в VW / TF / ...

Разреженные матрицы в TensorFlow

```
def scipy_sparse_to_tf_sparse(X):  
    X_coo = X.tocoo()  
    indices = np.mat([X_coo.row, X_coo.col]).transpose()  
    return tf.SparseTensorValue(indices, X_coo.data, X_coo.shape)
```

```
place_X = tf.sparse_placeholder(dtype=tf.float32, shape=(None, 74000))  
W = tf.Variable(tf.random_normal(shape=(74000, 1)))  
output = tf.sparse_tensor_dense_matmul(place_X, W)
```

Подходы к решению

$$L = \sum_{i=0}^{N_c} \text{LogLoss}(y_{pred}^{(i)}, y_{true}^{(i)})$$

$$L = \sum_{i=0}^{N_c} \frac{\text{LogLoss}(y_{pred}^{(i)}, y_{true}^{(i)})}{p^{(i)}}$$

$$L = \text{mean}(u), \quad u^{(i)} = \delta^{(i)} \frac{\pi_w^{(i)}}{\pi_0^{(i)}}$$

$$L = \text{mean}(u) + \text{std}(u), \quad u^{(i)} = \delta^{(i)} \frac{\pi_w^{(i)}}{\pi_0^{(i)}}$$

CLICKS

**CLICKS WITH
WEIGHTS**

IPS

IPS LOWER BOUND



Ожидание

Approach	$\hat{R}(\pi_\epsilon) \times 10^4$
Random	44.676 ± 2.112
π_0	53.540 ± 0.224
Regression	48.353 ± 3.253
IPS	54.125 ± 2.517
DRO	57.356 ± 14.008
POEM	58.040 ± 3.407

Реальность

Random	~44
pi_0	?
Clicks	~48
IPS	~54
DRO (VW)	~54
POEM (IPS_LB)	~54

Reward hacking

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n} \sum_{i=1}^n \delta_i \frac{\pi_w(y_i | x_i)}{\pi_0(y_i | x_i)}$$

$$\delta_0 = 0.1, \delta_1 = -1$$

Учится предсказывать клики

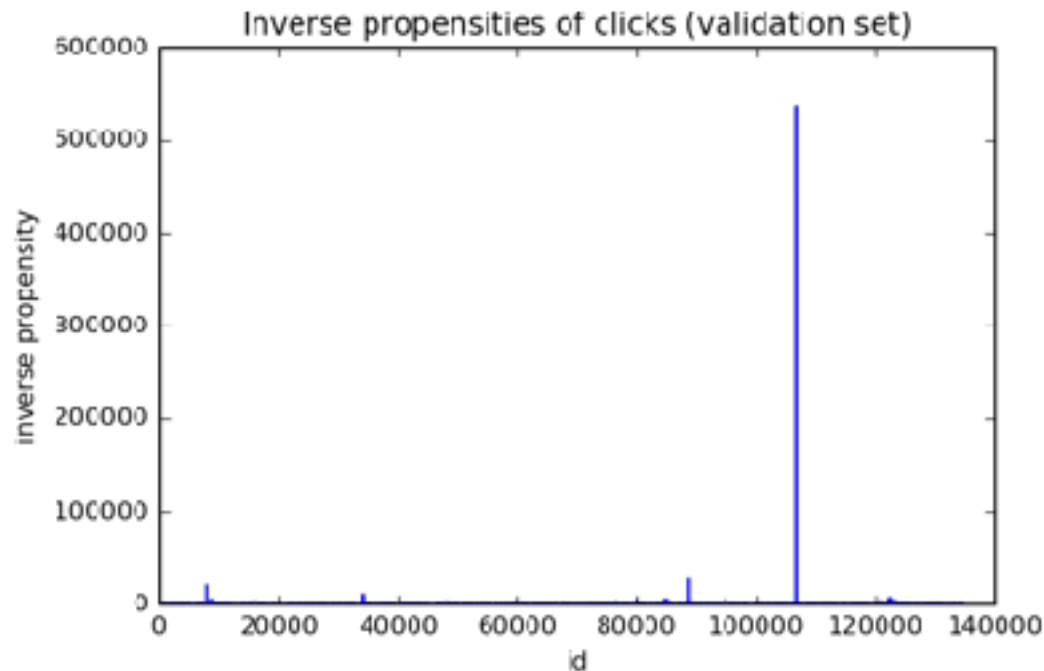
$$\delta_0 = -0.1, \delta_1 = -1$$

Учится предсказывать выбранного кандидата

В пределе, оба эти варианта идеальны с точки зрения используемой метрики.
На практике, у меня чуть лучше работал второй.

Propensity Overfitting

$$\hat{R}_{IPS}(\pi_w) = \frac{1}{n} \sum_{i=1}^n \delta_i \frac{\pi_w(y_i | x_i)}{\pi_0(y_i | x_i)}$$



Propensity Overfitting

```
uniform_solution = np.ones(len(vP)) / 11
n_clicks = len(uniform_solution)
print('num of clicks: {}, total samples: {}'.format(n_clicks, n_clicks / vCTR))
print('uniform policy as is, IPS: \t', IP_score(uniform_solution, vP, vCTR))

print('clipped propensities, IPS: \t', IP_score(uniform_solution, np.clip(vP, 1e-3, 1), vCTR))

uniform_solution[106663] = 0.0
print('miss single object, IPS: \t', IP_score(uniform_solution, vP, vCTR))

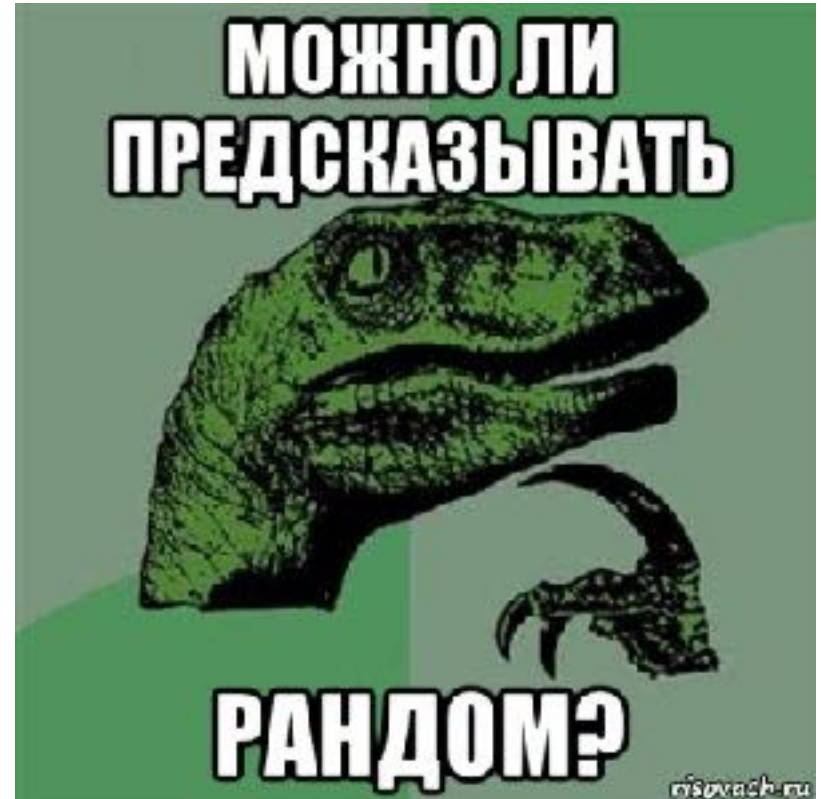
uniform_solution[106663] = 1.0
print('guess single object, IPS: \t', IP_score(uniform_solution, vP, vCTR))
```



```
num of clicks: 134868, total samples: 2572168.0
uniform policy as is, IPS:      67.714754575
clipped propensities, IPS:      41.9452974882
miss single object, IPS:        47.8217340716
guess single object, IPS:       266.644959609
```


Propensity Overfitting: безумная идея

- Предсказывать логарифм вероятности, с которой пронаблюдают кандидата
- Иными словами -- оценка “полезности” примера
- Попытка угадать, как разыграется случайная величина
- Немного получилось :)
- Но недостаточно, чтобы улучшить результат.



Evolution Strategies

- Легко реализовать
- Работает!
- Переобучается

```
class EvolutionStrategy(object):

    def __init__(self, weights, get_reward_func, population_size=10, sigma=0.2, learning_rate=0.1):
        np.random.seed(0)
        self.weights = weights
        self.get_reward = get_reward_func
        self.POPULATION_SIZE = population_size
        self.SIGMA = sigma
        self.LEARNING_RATE = learning_rate

    def _get_weights_try(self, w, p):
        weights_try = []
        for index, i in enumerate(p):
            jittered = self.SIGMA * i
            weights_try.append(w[index] + jittered)
        return weights_try

    def get_weights(self):
        return self.weights

    def run(self, iterations, print_step=10, seed=42):
        np.random.seed(seed)
        for iteration in tqdm.tqdm.notebook(range(iterations), total=iterations):

            if iteration % print_step == 0:
                global_reward = self.get_reward(self.weights)
                print('iter %d, rewards %f' % (iteration, global_reward))

            population = []
            rewards = np.zeros(self.POPULATION_SIZE)
            for i in range(self.POPULATION_SIZE):
                w = []
                for w in self.weights:
                    x.append(np.random.randn(*w.shape))
                population.append(x)

            for i in range(self.POPULATION_SIZE):
                weights_try = self._get_weights_try(self.weights, population[i])
                rewards[i] = self.get_reward(weights_try)

            rewards = (rewards - np.mean(rewards)) / np.std(rewards)

            for index, w in enumerate(self.weights):
                R = np.array([p[index] for p in population])
                self.weights[index] = w + self.LEARNING_RATE / (self.POPULATION_SIZE * self.SIGMA)
```

TensorFlow, 2K17

TensorFlow, 2K17



0



There is a hard limit of 2GB for serializing individual tensors because of the 32bit signed size in protobuf.

<https://github.com/tensorflow/tensorflow/issues/4291>

share edit

answered Feb 22 at 20:35



fabrizioM

25.1k ● 10 ● 54 ● 80

add a comment

```
from tensorflow.contrib.eager.python import tfe
tfe.enable_eager_execution()
```

Заклучение

Итог: процессинг

```
tX, tI, tC, tP = batches[i]
click_mask = tC < 0.5
```

```
# revert transformed by /10 negative probs and do propensity clipping
```

```
tP_ = tP.copy()
```

```
tP_[~click_mask] *= 10
```

```
tP_ = np.clip(tP_, 0.3, 1)
```

```
# set manual rewards (costs)
```

```
tC_ = tC.copy()
```

```
tC_[click_mask] = -1
```

```
tC_[~click_mask] = -0.1
```

```
# do optimization step
```

```
fd = {model.place_X: tX, model.place_I: tI, model.place_C: tC_, model.place_P: tP_}
```

```
_ = model.session.run(model.adam3, feed_dict=fd)
```

```
model.step += 1
```

Итог: модель

```
self.place_X = tf.sparse_placeholder(dtype=tf.float32, shape=(None, 74000), name="input_X")
self.place_I = tf.placeholder(dtype=tf.int32, shape=(None, 2), name="input_Indices")
self.place_C = tf.placeholder(dtype=tf.float32, shape=(None, ), name="input_Cost")
self.place_P = tf.placeholder(dtype=tf.float32, shape=(None, ), name="input_Propensity")

self.W = tf.Variable(tf.random_normal(shape=(74000, 1)), name="weights")

self.o_linear = tf.sparse_tensor_dense_matmul(self.place_X, self.W)
x = tf.reshape(self.o_linear, shape=[-1, 1], name='reshape')
self.probs = tf.nn.softmax(x)

self.sliced_probs = tf.gather_nd(self.probs, self.place_I)
self.ratio = self.sliced_probs / self.place_P
self.r = self.place_C * self.ratio
r_mean, r_var = tf.nn.moments(self.r, axes=[0])

self.loss = tf.reduce_mean(self.r) + 0.0001*tf.sqrt(tf.nn.l2_loss(self.o_linear))
self.adam3 = tf.train.AdamOptimizer(learning_rate=0.003).minimize(self.loss)
```

Заключение

- Интересная задачка и потенциальная область применения
- Странная метрика (имхо, нужно было ограничивать вероятности)
- Работала линейная модель
- Простая идея: добавлять рандомизацию + сохранять вероятности + IPS