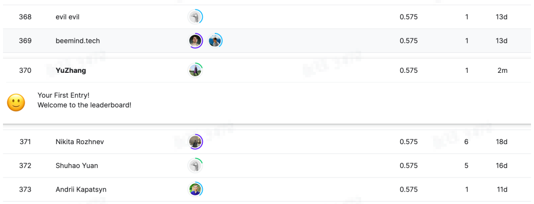
OTTO竞赛

开发&提交记录

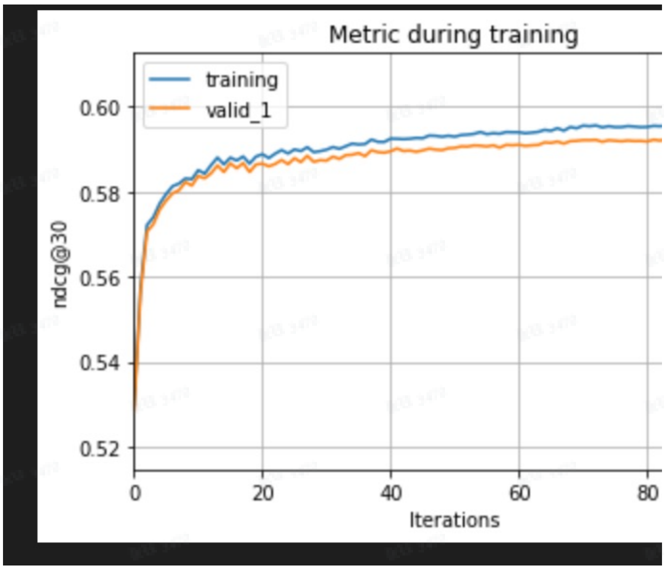
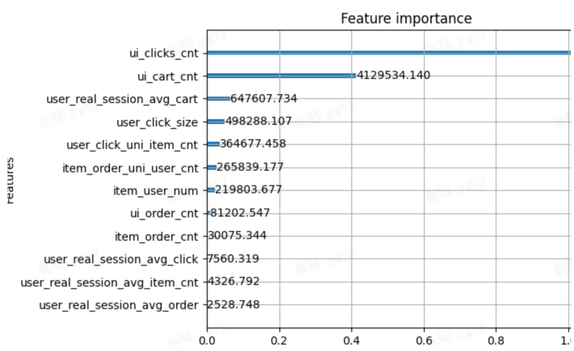
天梯 https://www.kaggle.com/competitions/otto-recommender-system/leaderboard

争取进top50

<table>
<tr><th>日期</th><th>成绩</th><th>改动说明</th></tr>
<tr><td>12/03</td><td>

0.575</td><td>直接把baseline在kaggle上运行一下就行了
基本没有任何改动
https://www.kaggle.com/code/cdeotte/candidat
model-lb-0-575</td></tr>
<tr><td>1/21</td><td>0.397</td><td>Click 50个召回直接取top20
buy&cart训练三个feature的lgbm，加上50%的负采样
test时item特征需要考虑train+val+test全部特征，ズ
train+valA
https://www.kaggle.com/yuzhang0422/train-predi</td></tr>
<tr><td>1/22</td><td>0.397</td><td>修复了test特征，没有改善
跳过精排，debug下recall部分单独提交的分数
参考https://www.kaggle.com/code/adaubas/otto-fa
handcrafted-model-recall-20
<pre>[11]: # 计算recall rate和总分数
_ = otto_metric(val_clicks, val_orders, val_

clicks recall = 0.52547 (vs 0.52556 in benchmark)
carts recall = 0.40856 (vs 0.40933 in benchmark)
orders recall = 0.63482 (vs 0.64879 in benchmark)
=============
Overall Recall = 0.55601 (vs 0.56463 in benchmark
=============</pre>排查显示recall的问题并不大
单独提交一版基于recall的test，0.568分。基本可以断
了。</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

1/24	0.567	单独提交recall，0.567
1/26		发现ndcg≈1可能是由于负例过多导致 降低负例采样率 ~~看看ndcg如何处理target全0的user。发现全0的话lgb~~ ~~试试把click=1的作为负例。平均每个用户只有2个点击~~
1/27	0.321	调整了负采样，过滤无cart/order用户  训练看起来没有问题 但是LB和本地Metric还是很低
1/28		再次检查label拼接。没有问题 过滤所有无交互session后，carts NDCG@20≈0.27，接近0.27，说明目前的**LGBM学习能力不足** 之前虚高的NDCG是由于过多的负例，无交互session
1/29		使用word2vec构建相似度特征 [https://www.kaggle.com/competitions/otto-recom system/discussion/367234](https://www.kaggle.com/competitions/otto-recom)
1/31	带上ui特征0.51 带上recall序特征0.555 调大数据0.558 最终1474/2616  	ui特征非常强 

		lgbm无法用gpu做predict，所以尽可能减少n_estima 少feature数目，来加快predict过程

总结

关键上分点：

1.co-visitation矩阵

2.ui交互特征

一些经验：

gpu加速lgbm训练速度约x30

lgbm超参对于结果影响很小

ranker对于无交互的负例直接全部过滤

内存不足用pandas分chunk处理

题意

github链接 https://github.com/otto-de/recsys-dataset

Kaggle赛题链接 OTTO – Multi-Objective Recommender System

学习用户历史交互，推荐可能下一次点击、加车、购买的物品

每一个type可以预估20个，最终以召回率作为评判标准

train：20w个用户的历史行为序列。

test：填充每一个test uid之后的行为，有大约一半的数据用于冷启

评估：三类行为**召回率**的**加权**和

$$score = 0.10 \cdot R_{clicks} + 0.30 \cdot R_{carts} + 0.60 \cdot R_{orders}$$

$$R_{type} = \frac{\sum\limits_{i=1}^{N} |\{\text{predicted aids}\}_{i,type} \cap \{\text{ground truth aids}\}_{i,type}|}{\sum\limits_{i=1}^{N} \min\left(20, |\{\text{ground truth aids}\}_{i,type}|\right)}$$

数据

数据包含用户的浏览行为

session-id可以视为user-id，aid可以视为item-id

一共有三类行为：click，carts，order

下面这个用户行为

```
{
    "session": 42,
    "events": [
        { "aid": 0, "ts": 1661200010000, "type": "clicks" },
        { "aid": 1, "ts": 1661200020000, "type": "clicks" },
        { "aid": 2, "ts": 1661200030000, "type": "clicks" },
        { "aid": 2, "ts": 1661200040000, "type": "carts"  },
        { "aid": 3, "ts": 1661200050000, "type": "clicks" },
        { "aid": 3, "ts": 1661200060000, "type": "carts"  },
        { "aid": 4, "ts": 1661200070000, "type": "clicks" },
        { "aid": 2, "ts": 1661200080000, "type": "orders" },
        { "aid": 3, "ts": 1661200080000, "type": "orders" }
    ]
}
```

可以被如下展示

> 横向为时间序

click	0	1	2		3		4		
add2cart				2		3			
order								2	3

数据理解

原始的数据是jsonl数据格式的

test.jsonl (402.09 MB)

This preview is truncated due to the large file size. The number of JSON items
be might be truncated. Create a Notebook or download this file to see the full c


```
"root" : { 2 items
  "session" : int 12899779
  "events" : [ 1 item
    0 : { 3 items
      "aid" : int 59625
      "ts" : float 1661724000278
      "type" : string "clicks"
    }
  ]
}
```

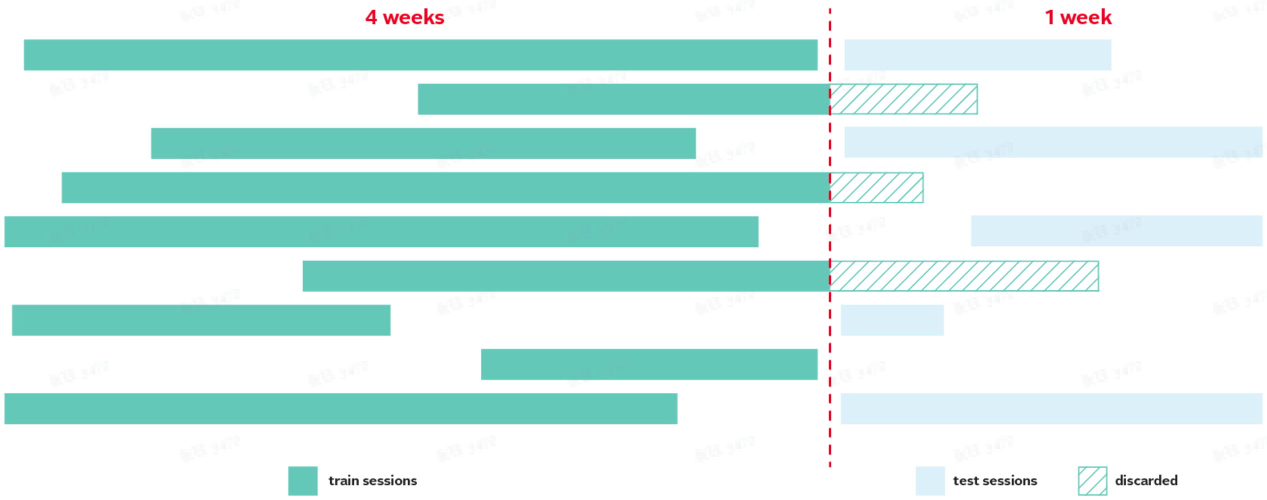

Recommendation Systems for Large Datasets

原始数据规模如下：

- 12,899,779 sessions，**12M**用户
- 1,855,603 items，**1.8M**物品
- 216,716,096 events，2.1亿次交互
- 194,720,954 clicks
- 16,896,191 carts
- 5,098,951 orders

train内含有12M用户，test含有1.6M

train是前4周，test是后面1周。并且为了避免特征穿越，train内不含最后1周。



Week 5的数据被切分成两部分，一部分放进test data（用于模型的输入创建submission.csv），另外一部分相当于ground-truth计算leaderboard

关键insight

- 不同session id要视为独立的交互事件，也即每一次session id都是一个**新用户**
- 每一个session都被切分到了指定的时间片，下面数据集切分方法是train用前三周，val是第4周，test是第五周。三个集合**没有重复**
- 每一个session内部有**多个**real_session，每一个real_session内反映了用户的一些消费习惯
- 任务目标是给定一批session的**前半段**，预测**后半段**。
  - 数据中val切分成两段，valA是前半段交互（用于构建model input），valB是后半段（就是label）
  - test其实是第五周的前半段，后半段在leadboard上，用于评分，我们不可见

	A	B	C	D	E
1	用户群	**用户数**	数据切分	usage	~~aid~~
2	train	**11,098,528**	train	train+valA提item特征	~~1,825,499~~
3	val	**1,801,251**	valA	train+valA提item特征； 提val的user特征	~~874,852~~
4			valB	作为模型label	
5	test	**1,671,803**	test	提取test的user特征；item特征 训练好的模型eval，输出的prediction作为submission	
6			不可见	计算submission的评分	

数据转换

- 历史变更
  - Colum2131帮忙把原始的train，test转换成分片的parquet，OTTO Chunk Data in Parquet Format
  - Radek帮忙转换成了parquet格式，并且区分了train，val，test
  - local validation tracks public LB perfecty，OTTO train and validation (extracted from train)
  - 这里的test是train中最后1week的数据，并且还要按时间切成AB两部分，valA等价于val-train，valB等价于val-label

Data Explorer

Version 1 (1.35 GB)

- id2type.pkl
- test.parquet
- test_labels.parquet
- train.parquet
- type2id.pkl

- otto-validation
  - ts已经乘回了1000（单位秒），所以需要先除1000再用datetime进行转换
  - Chris帮忙把这个数据切碎成100块，避免遇到内存不足的问题，**后续主要就用这个数据集**。test.parquet就是val-train，test_labels.parquet是val-labels
  - train_partquet（train）：
    - 含义：每一行是一个交互，包含用户在前三周的行为
    - 作用：参与构建Item之间的共现矩阵，以及各类特征

[4]:

	session	aid	ts	type
0	11098528	11830	1661119200000	clicks
1	11098529	1105029	1661119200000	clicks
2	11098530	264500	1661119200000	clicks
3	11098530	264500	1661119288000	clicks
4	11098530	409236	1661119369000	clicks
...	...	...	...	...
450996	11188589	641024	1661165822000	clicks
450997	11188589	1627743	1661631490000	clicks
450998	11188590	1689436	1661165822000	clicks
450999	11188590	607328	1661165932000	clicks
451000	11188590	607328	1661165944000	clicks

451001 rows × 4 columns

  - test_parquet（validA）：
    - 含义：每一行是一个交互，时间是第四周前半段
    - 作用：用于validation阶段的预测的Trigger，参与构建特征
  - test_labels（validB）：
    - 含义：代表每一个uid在不同type下与哪些aid发生了交互，时间是第四周后半段

- 作用：用于validation阶段的验证

```
train_labels
```

[17…

	session	type	ground_truth
0	11098528	clicks	[1679529]
1	11098528	carts	[1199737]
2	11098528	orders	[990658, 950341, 1462506, 1561739, 907564, 369...
3	11098529	clicks	[1105029]
4	11098530	orders	[409236]
...	...	...	...
2212687	12899774	clicks	[1399483]
2212688	12899775	clicks	[1760714]
2212689	12899776	clicks	[1737908]
2212690	12899777	clicks	[384045]
2212691	12899778	clicks	[32070]

2212692 rows × 3 columns

架构设计

召回

> 目标
>
> ```
> val_clicks
> ```
>
> [13…
>
> ```
> 11098528    [11830, 588923, 1732105, 571762, 884502, 11578...
> 11098529    [1105029, 459126, 1339838, 1544564, 217742, 16...
> 11098530    [409236, 264500, 1603001, 963957, 254154, 5830...
> 11098531    [396199, 1271998, 452188, 1728212, 1365569, 62...
> 11098532    [876469, 7651, 108125, 1202618, 1159379, 77906...
>                                   ...
> 12899774    [33035, 1539309, 819288, 95488, 771913, 270852...
> 12899775    [1743151, 1760714, 1255910, 1163166, 832192, 2...
> 12899776    [548599, 1401030, 1440959, 1607333, 1144446, 3...
> 12899777    [384045, 1308634, 1688215, 703474, 395762, 148...
> 12899778    [561560, 1167224, 32070, 1175618, 13942, 56604...
> Length: 1801251, dtype: object
> ```
>
> + Code   + Markdown

- 基线版本 https://www.kaggle.com/code/yuzhang0422/candidate-rerank-model-lb-0-575/edit
- 加速版本 https://www.kaggle.com/code/adaubas/otto-fast-handcrafted-model-recall-20
  - 不要启动GPU，否则会减少内存以及core数，导致超内存

Co-visitation

想法来源 Co-visitation Matrix

- 数据 otto-validation

- 使用其中的test_parquet+train_parquet（train+validA），提取出共现矩阵

[2]:

	session	aid	ts	type
0	12089221	700554	1661448002	0
1	12089221	619488	1661448024	0
2	12089221	579241	1661449547	0
3	12089221	619488	1661449585	0
4	12089221	619488	1661456661	0

- 预存co-visitation matrix，数据 OTTO co visitation matrices
  - Co-visitation matrix of click/cart/order to cart/order with type weighting——用于建模各个type之间的共现关系，并且不同的行为共现权重不同，click1/cart6/order3。
  - Co-visitation matrix of cart/order to cart/order called buy2buy——主要建模购买之间的共现关系，只考虑共现次数（权重都是1）
  - Co-visitation matrix of click/cart/order to clicks with time weighting——考虑不同行为对于点击的共现关系

共现矩阵最后提取出来的是每一个aid top 20共现的aid，`top_20_clicks\buys\buy2buy`

```
[7]: top_20_clicks
```

```
[7]: {0: [532042,
  643097,
  1848174,
  1735605,
  1363081,
  1012453,
  251581,
  1256426,
  994505,
  150507,
  39551,
  66456,
  215649,
  285817,
  319593,
  353304,
  435875,
  472412,
  505086,
  878565],
 1: [28092,
  1533875,
  190430,
  645003,
  1234826,
  1815894,
  590833,
```

读取test_parquet，提取**测试期间**的高热，top_clicks\orders

构造测试用例，`valid_bysession_list`

每一个格式是 `[uid,[aid_list],[action_list]]`

[10]:
```
valid_bysession_list
```

```
[[11098528, [11830], [0]],
 [11098529, [1105029], [0]],
 [11098530,
  [264500, 264500, 409236, 409236, 409236, 409236],
  [0, 0, 0, 0, 0, 1]],
 [11098531,
  [452188,
   1239060,
   1557766,
   452188,
   396199,
   1309633,
   1449555,
   1557766,
   1728212,
   1365569,
   1728212,
   1271998,
```

召回核心部分

suggest_clicks/buys/orders函数。

执行过程，并行对于 `valid_bysession_list` 中的每一个uid样例进行预测

代码逻辑

- 如果该用户已经交互过超过20个aid，那么按照交互时间先后继续0~1插值，加权取出top 20，返回
- 否则，通过共现矩阵，每个交互过的aid都找到其共现最高频的20个aid。比如总共交互N个aid，召回 `N*20` 个aid，然后从 `N*20` 个aid中中找到频次最高的top20个aid返回。

召回结果如下，存放在val_clicks，每一个uid都有一个长度20的序列

```
val_clicks
```

```
11098528      [11830, 588923, 1732105, 571762, 884502, 11578...
11098529      [1105029, 459126, 1339838, 1544564, 217742, 16...
11098530      [409236, 264500, 1603001, 963957, 254154, 5830...
11098531      [396199, 1271998, 452188, 1728212, 1365569, 62...
11098532      [876469, 7651, 108125, 1202618, 1159379, 77906...
                                    ...
12899774      [33035, 1539309, 819288, 95488, 771913, 270852...
12899775      [1743151, 1760714, 1255910, 1163166, 832192, 2...
12899776      [548599, 1401030, 1440959, 1607333, 1144446, 3...
12899777      [384045, 1308634, 1688215, 703474, 395762, 148...
12899778      [561560, 1167224, 32070, 1175618, 13942, 56604...
Length: 1801251, dtype: object
```

~~但是不清楚为什么buy和order使用的是同样的预测~~

问了原文作者：答案是为了省事，使得notebook运行的更快。

可以自己尝试做独立的逻辑，这样可能可以提升LB分数

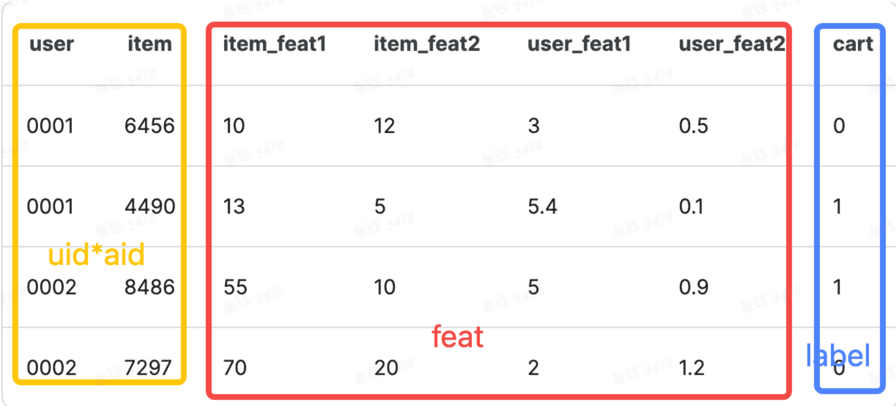
```
clicks_pred_df = pd.DataFrame(pred_df_clicks.add_suffix("_clicks"), columns=["labels"]).reset_index()
orders_pred_df = pd.DataFrame(pred_df_buys.add_suffix("_orders"), columns=["labels"]).reset_index()
carts_pred_df = pd.DataFrame(pred_df_buys.add_suffix("_carts"), columns=["labels"]).reset_index()
```

特征工程

参考天池推荐赛 https://tianchi.aliyun.com/notebook/144453

数据处理

> 目标拿到这样的结果



user	item	item_feat1	item_feat2	user_feat1	user_feat2	cart
0001	6456	10	12	3	0.5	0
0001	4490	13	5	5.4	0.1	1
0002	8486	55	10	5	0.9	1
0002	7297	70	20	2	1.2	0

可能的特征 https://www.kaggle.com/code/cdeotte/candidate-rerank-model-lb-0-575/comments#2030893

Example user features

- how many items has user already clicked
- how many items has user already ordered
- what is average hour that user clicks
- what is average hour that user orders
- how many real sessions does user have (real session define by time gap between activity)
- what is average number of items in each user real session
- what is last day of week user made activity (i.e. monday, tuesday)
- what is first day of week user made activity
- what is average time between clicks

Example item features

- has this item already been clicked by user
- has this item already been added to cart by user
- if already clicked, what is its relative order? 1 means last clicked, 2 means second to last clicked etc
- has user clicked this item multiple times already? how many
- how many items (that user has already clicked) have recommended this item with their co-visitation matrix
- when was date that this item was first seen in train
- how many times what this item clicked in train
- what is the average hour of day that this item is clicked
- what is the average hour of day that this item is ordered
- how popular is this item on monday (i.e. what percentage of monday clicks are this item)
- how popular is this item on tuesday
- what is the most common day of week this item is clicked
- count up all unique items that were clicked immediately before and after. How many unique items have been cli immediately before and after. (For example, maybe item only has 10 unique items that get clicked before and after another item has 1000 unique items clicked before and after)
- what percentage of users click this item more than once
- has this item ever been bought in train data

- 在**召回后的item**上分别构建item feature和user feature
    - 每人召回20条click，100条cart，100条order
    - 构建成(User,Item,Label)形式
    - 汇总User_set，以及召回的Item_set
    - 仅保留User_set和Item_set同时满足的交互记录
- 负采样
    - 对负例进行下采样5%，正例不采样<==想法来源https://www.kaggle.com/competitions/otto-recommender-system/discussion/370116
    - 负采样之后，保证**所有的用户和文章**仍然出现在采样之后的数据中

```python
# 负采样函数，这里可以控制负采样时的比例，这里给了一个默认的值
def neg_sample_recall_data(recall_items_df, sample_rate=0.05):
    '''
    recall_items_df ['user_id', 'sim_item', 'score']
    '''
    pos_data = recall_items_df[recall_items_df['label'] == 1]
    neg_data = recall_items_df[recall_items_df['label'] == 0]

    print('pos_data_num:', len(pos_data), 'neg_data_num:', len(neg_data), 'pc

    # 分组采样函数
    def neg_sample_func(group_df):
        neg_num = len(group_df)
        sample_num = max(int(neg_num * sample_rate), 1) # 保证最少有一个
        sample_num = min(sample_num, 5) # 保证最多不超过5个，这里可以根据实际情况
        return group_df.sample(n=sample_num, replace=True)

    # 对用户进行负采样，保证所有用户都在采样后的数据中
    neg_data_user_sample = neg_data.groupby('user_id', group_keys=False).appl
    # 对文章进行负采样，保证所有文章都在采样后的数据中
    neg_data_item_sample = neg_data.groupby('sim_item', group_keys=False).app

    # 将上述两种情况下的采样数据合并
    neg_data_new = neg_data_user_sample.append(neg_data_item_sample)
    # 由于上述两个操作是分开的，可能将两个相同的数据给重复选择了，所以需要对合并后的数
    neg_data_new = neg_data_new.sort_values(['user_id', 'score']).drop_duplic

    # 将正样本数据合并
    data_new = pd.concat([pos_data, neg_data_new], ignore_index=True)

    return data_new
```

Item feature

> 用train + valid data

- norm(1/点击总次数)+norm(时间差)，越小热度越高
- click，cart，order覆盖用户率
- 总click数，cart数，order数
- 平均click率，cart率，order率
- 被click、cart，order的DoW，HoD的众数

User feature

> **仅仅用valid data**，因为test时很多用户也是新用户

- norm(1/交互总次数)+norm(时间差)，越小活跃度越高
- 总click数，cart数，order数
- 平均click率，cart率，order率
- click、cart、order的DoW和HoD的众数
- 平均click、cart、order的交互间隔时间

U*I feature

> 仅用Valid A

- 当前item与用户最近三个交互的item的共现权重的（最大值，最小值，均值，时间距离加权和）

已开发特征

- 红色字体为新增特征

	A	B	C
1	**type**	**name**	**descp**
2	Item feature	item_avg_DoW	item平均出现时间DoW
3	Item feature	item_avg_HoD	item平均出现时间HoD
4	Item feature	item_click_cnt	item被click行为次数
5	Item feature	item_cart_cnt	item被cart行为次数
6	Item feature	item_order_cnt	item被order行为次数
7	Item feature	item_cart_ratio	item点击后cart概率
8	Item feature	item_order_ratio	item cart后order概率
9	Item feature	item_order_click_ratio	item点击后order概率
10	Item feature	item_click_uni_user_cnt	item被click用户数
11	Item feature	item_cart_uni_user_cnt	item被cart用户数
12	Item feature	item_order_uni_user_cnt	item被order用户数
13	Item feature	user_num	item被交互总用户数（归一化0到1）
14	Item feature	time_diff_mean	item被交互行为时间间隔的均值
15	Item feature	item_interaction_time_diff_mean	item被交互行为时间间隔的均值（归一化）
16	Item feature	item_hotness	item热度值（user_num+item_interaction_time_diff_
17	user feature	user_avg_DoW	user活跃的DoW
18	user feature	user_avg_HoD	user活跃的HoD

18	user feature	user_avg_HoD	user活跃的HoD
19	user feature	user_last_DoW	user最后一次活跃DoW
20	user feature	user_first_DoW	user第一次活跃DoW
21	user feature	user_active_seconds	user活跃时长s
22	user feature	user_active_minutes	user活跃时长min
23	user feature	user_active_hours	user活跃时长hour
24	user feature	user_active_days	user活跃时长day
25	user feature	click_size	user交互行为次数（归一化）
26	user feature	user_action_time_diff_mean	user交互行为的时间间隔均值
27	user feature	user_activeness	user活跃度（click_size+user_action_time_diff_mean
28	user feature	user_click_cnt	user 点击次数
29	user feature	user_cart_cnt	user cart次数
30	user feature	user_order_cnt	user order次数
31	user feature	user_cart_ratio	user 点击后加购物车概率
32	user feature	user_order_ratio	user 加购物车后order概率
33	user feature	user_order_click_ratio	user 点击后order概率
34	user feature	user_click_item_cnt	user click了几种item
35	user feature	user_cart_item_cnt	user cart了几种item
36	user feature	user_order_item_cnt	user order了几种item
37	user feature	real_session	real session
38	user feature	user_real_session_avg_item_cnt	real session平均交互item数
39	user feature	user_real_session_avg_click	real session平均click item数
40	user feature	user_real_session_avg_cart	real session平均avg item数
41	user feature	user_real_session_avg_order	real session平均order item数
42	user feature	user_real_session_avg_duration	real session平均活跃时长(s)
43	user feature	user_real_session_avg_click_diff	real session平均活跃间隔(s)
44	user*item feature	user_item_click_cnt	user和item的click次数
45	user*item feature	user_item_cart_cnt	user和item的cart次数
46	user*item feature	user_item_order_cnt	user和item的click次数
47			

精排

> 给每一个session预估20个

	session_type	labels
0	12899779_clicks	59625 1253524 737445 438191 731692 1790770 942...
1	12899780_clicks	1142000 736515 973453 582732 1502122 889686 48...
2	12899781_clicks	918667 199008 194067 57315 141736 1460571 7594...
3	12899782_clicks	834354 595994 740494 889671 987399 779477 1344...
4	12899783_clicks	1817895 607638 1754419 1216820 1729553 300127 ...

GBM Ranker整体思路 https://www.kaggle.com/competitions/otto-recommender-system/discussion/370210

几种排序算法的介绍 LTR排序算法LambdaRank原理详解

改用pandas的LGBM ranker https://www.kaggle.com/code/yuzhang0422/cudf-pandas-proof-of-concept-lgbm-ranker/edit

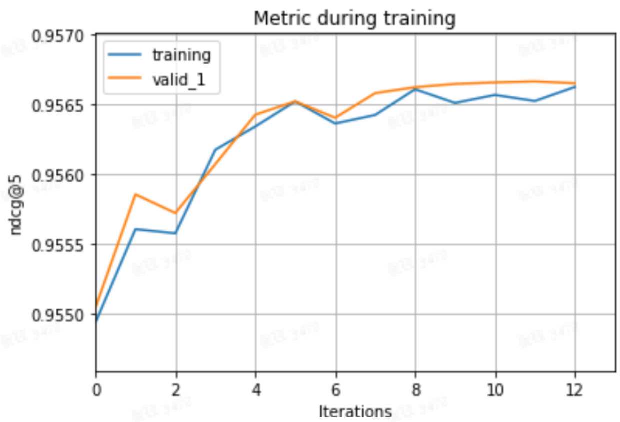
LGBM Ranker的使用方法 How to implement learning to rank using lightgbm?

- `label`：用于表示用户是否点击，或者一个数值，越大代表越应该排在前面。预测结果也是这个**排序权重**
- `group` 参数：用于标识训练数据中每一个请求的长度，然后ranker在请求内进行排序。在本次比赛中，group应该被设定为每一个user的候选物品数目。

预测的结果是概率，排在第一个

各种尝试

```
params = {
    'device':'gpu',
    'learning_rate': 0.02,
    'max_depth': 8,
    'early_stopping_round':3,
    'objective':"lambdarank",
    'metric':"ndcg",
    'boosting_type':"gbdt",
    'n_estimators':20,
    'importance_type':'gain',
}
```



Submission

```
session_type,labels
12906577_clicks,135193 129431 119318 ...
12906577_carts,135193 129431 119318 ...
12906577_orders,135193 129431 119318 ...
12906578_clicks, 135193 129431 119318 ...
etc.
```

注意召回量不能太小，否则ndcg会虚高 https://www.kaggle.com/competitions/otto-recommender-system/discussion/377442

Trick

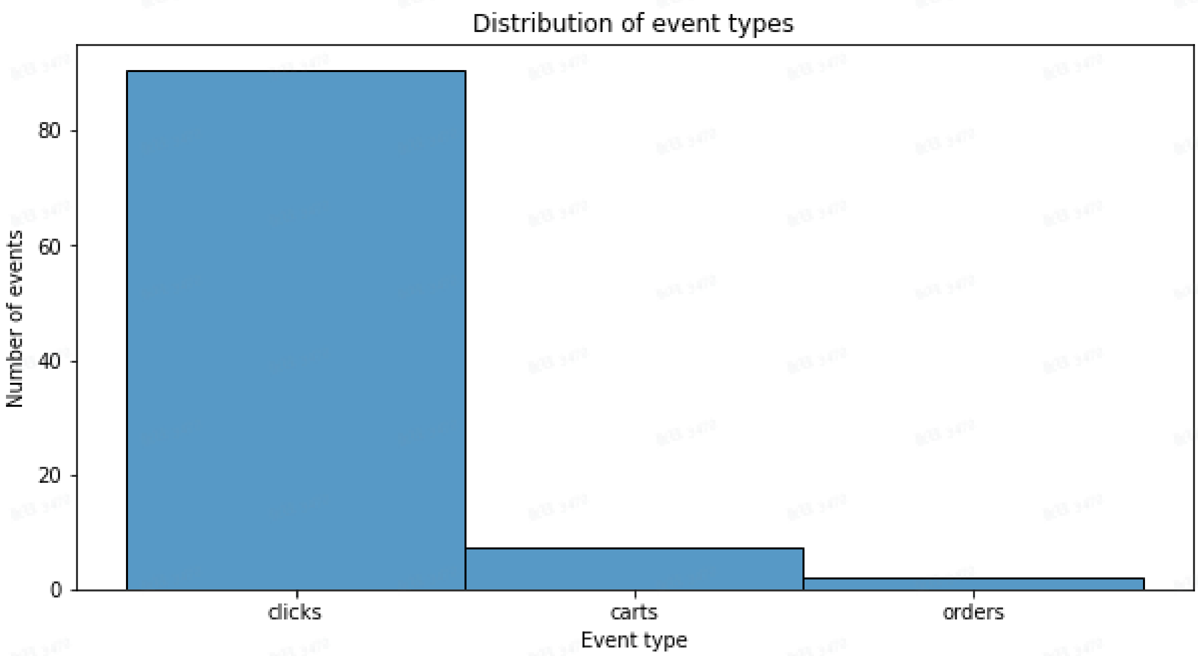
Model stacking

思路

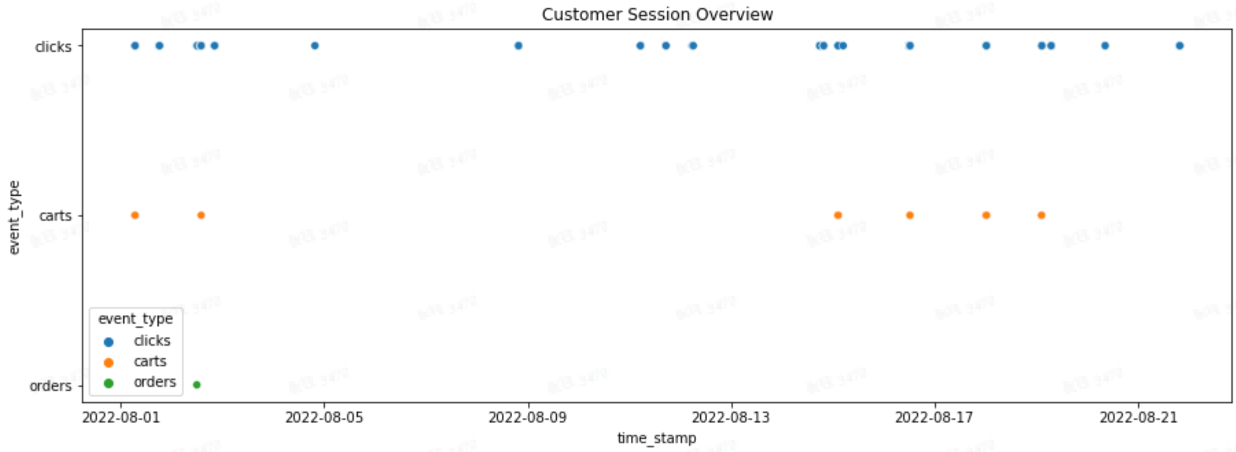
可以把session_id视为uid来用 [OTTO: Rename Columns](#) 📝

数据分析结果 [OTTO EDA - Understanding users and events](#)

200000个uid，？？ 个aid



可视化用户的行为记录



统计均值，P(click|click)=83.64%, P(cart|click)=5.89%, P(order|cart)=0.63%

发现order行为极其稀疏，但是召回权重比较大。

session内event均值51，方差75，max=495，min=2，长尾分布

参考 https://www.kaggle.com/code/cdeotte/candidate-rerank-model-lb-0-575

上手 https://www.kaggle.com/competitions/otto-recommender-system/discussion/367058

精排特征 https://www.kaggle.com/code/cdeotte/candidate-rerank-model-lb-0-575/comments#2030893

可以高度参考这个 推荐算法 入门赛 阿里天池

工具

本地测试分数与LB一致 https://www.kaggle.com/competitions/otto-recommender-system/discussion/364991

本地自测code 💡A robust local validation framework 🚀🚀🚀

省内存 https://www.kaggle.com/competitions/otto-recommender-system/discussion/368170

切分数据为Parquet Format格式https://www.kaggle.com/code/radek1/howto-full-dataset-as-parquet-csv-files

LGBM GPU加速 https://lightgbm.readthedocs.io/en/latest/GPU-Tutorial.html

```
git clone --recursive https://github.com/microsoft/LightGBM
cd LightGBM
```

```
mkdir build
cd build
cmake -DUSE_GPU=1 ..
cmake -DUSE_GPU=1 -DOpenCL_LIBRARY=/usr/local/cuda/lib64/libOpenCL.so -DOpenC
make -j$(nproc)
cd ..
cd python-package/
conda activate <your env>
python setup.py install --precompile
```