kaggle

OTTO – Multi-Objective Recommender System
Kaggle Winner Presentation
1st place

mrkmakr (Akira Murakami)

Agenda

Agenda

1. Background
2. Summary
3. Feature selection & engineering
4. Training methods
5. Important findings
6. Simple model

Background

- Name : Akira Murakami
- Japanese
- My English skill
  - My reading is ok
  - I am not confident in speaking and listening
  - If you think that communication is not possible, it would be helpful if you could tell me by text.

- I work as a data scientist and ML engineer at Yahoo! JAPAN, which is an internet company and the second largest search engine used in Japan behind Google.
- I work on internet display advertising and handles data in a format similar to the OTTO competition.
  - data having item id, user id and timestamp
  - data having multiple reactions like view, click and conversion.

Summary

Summary

- My approach is a two-step strategy of recall and reranking.
    - Recall : Covisitation, revisitation and Neural network
    - Reranking : LGBMRanker
- NN was useful for clicks and carts, and covisitation and revisitation were useful for orders.
- I used Python with Pytorch, polars and lightgbm
- My final solution takes 2 weeks for training and inference

Features Selection/ Engineering

Features Selection/Engineering

- Recall strategies
  - covisitation
  - revisitation
  - neural network
- Reranking features
  - Aid * session
    - Recall scores by covisitation and NN
    - aid information in the session (when it appeared, what type it is, etc)
  - Aid
    - popularity of aid
    - ratio of types
  - Session
    - Length
    - aid duplication ratio
    - ts between the last aid and the second last aid
    - type ratio

Recall@20 by each strategy (local validation)

	recall@20 clicks
emb_clicks_v42	0.539910
emb_clicks_v27	0.537567
emb_clicks_v31	0.535343
emb_carts_v42	0.534813
emb_carts_v31	0.534624
emb_orders_v31	0.534470
emb_carts_v27	0.533615
emb_clicks_v29	0.532640
emb_orders_v27	0.532178
emb_carts_v29	0.531648
emb_orders_v29	0.531528
emb_orders_v42	0.526960
count_dic_v15_p_chain_all_20_2_all_0	0.511804
count_dic_v15_p_chain_all_100_2_all_0	0.511804
count_dic_v11_p_chain_w_100_2_all_0	0.509067
count_dic_v20_p_chain_w_20_2_all_0	0.508844
count_dic_v11_p_chain_all_100_2_all_0	0.508827
count_dic_v11_p_chain_full_100_5_3_0	0.506997
count_dic_v16_p_chain_all_100_2_all_0	0.503507
count_dic_v19_p_chain_w_20_2_all_0	0.502327

	recall@20 carts
emb_orders_v42	0.410156
emb_carts_v42	0.409639
emb_carts_v27	0.407570
emb_orders_v27	0.407157
emb_carts_v31	0.406588
emb_orders_v31	0.405916
emb_clicks_v42	0.404933
emb_clicks_v31	0.404571
emb_clicks_v27	0.403899
emb_carts_v29	0.403382
emb_orders_v29	0.403227
count_dic_v19_p_chain_w_20_2_all_0	0.402296
count_dic_v14_p_chain_w_100_2_all_0	0.401779
count_dic_v14_p_chain_all_100_2_all_0	0.401469
emb_clicks_v29	0.400796
count_dic_v18_p_chain_w_100_2_all_0	0.399038
count_dic_v16_p_chain_all_100_2_all_0	0.398211
count_dic_v19_p_chain_w_20_2_all_1	0.396401
count_dic_v14_p_chain_all_20_2_all_1	0.395522
count_dic_v20_p_chain_w_20_2_all_1	0.394126

	recall@20 orders
count_dic_v19_p_chain_w_20_2_all_0	0.640493
count_dic_v14_p_chain_w_20_2_all_0	0.640403
count_dic_v14_p_chain_w_100_2_all_0	0.640403
count_dic_v14_p_chain_all_100_2_all_0	0.640040
count_dic_v18_p_chain_w_100_2_all_0	0.637138
count_dic_v11_p_chain_w_100_2_all_0	0.634781
count_dic_v16_p_chain_all_100_2_all_0	0.633240
count_dic_v19_p_chain_w_20_2_all_1	0.631426
count_dic_v14_p_chain_w_20_2_all_1	0.631064
count_dic_v14_p_chain_all_20_2_all_1	0.630701
count_dic_v12_p_chain_all_100_2_all_0	0.628162
count_dic_v17_p_chain_all_100_2_all_0	0.627618
count_dic_v13_p_chain_all_100_2_all_0	0.625442
count_dic_v18_p_chain_w_20_2_all_1	0.625261
count_dic_v20_p_chain_w_20_2_all_1	0.623810
count_dic_v11_14_chain_2_2dic_all_2	0.616738
count_dic_v13_p_chain_all_20_2_all_1	0.609484
count_dic_v16_p_chain_all_20_2_all_1	0.602956
isin_0_carts	0.594614
emb_orders_v42	0.565962

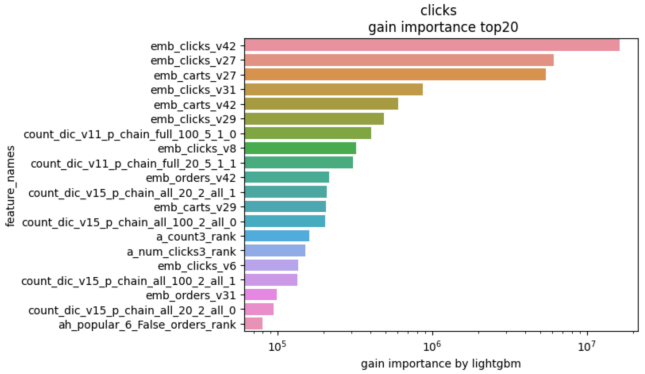
emb_*
 = NN prediction

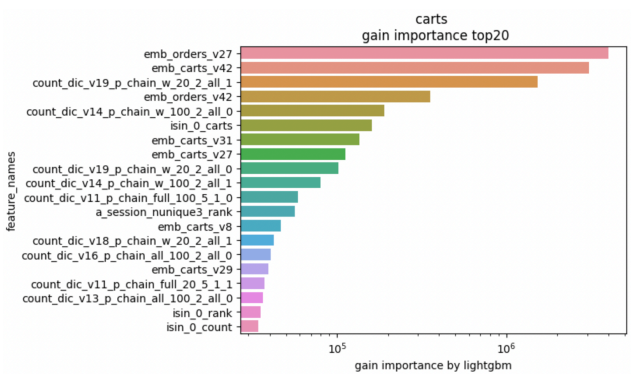
count_dic_*
 = covisitation

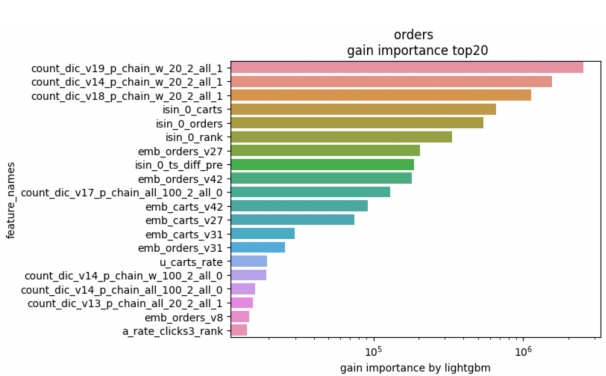
isin_*
 = revisitation

- For clicks and carts, NN is important.
- For orders, covisitation is important

Gain importance of Reranker







Similar to recall results. NN is important for clicks and carts, covisitation is important for orders.

Some session features(u_*) and aid features(a_*) appear, but less important than interaction features.

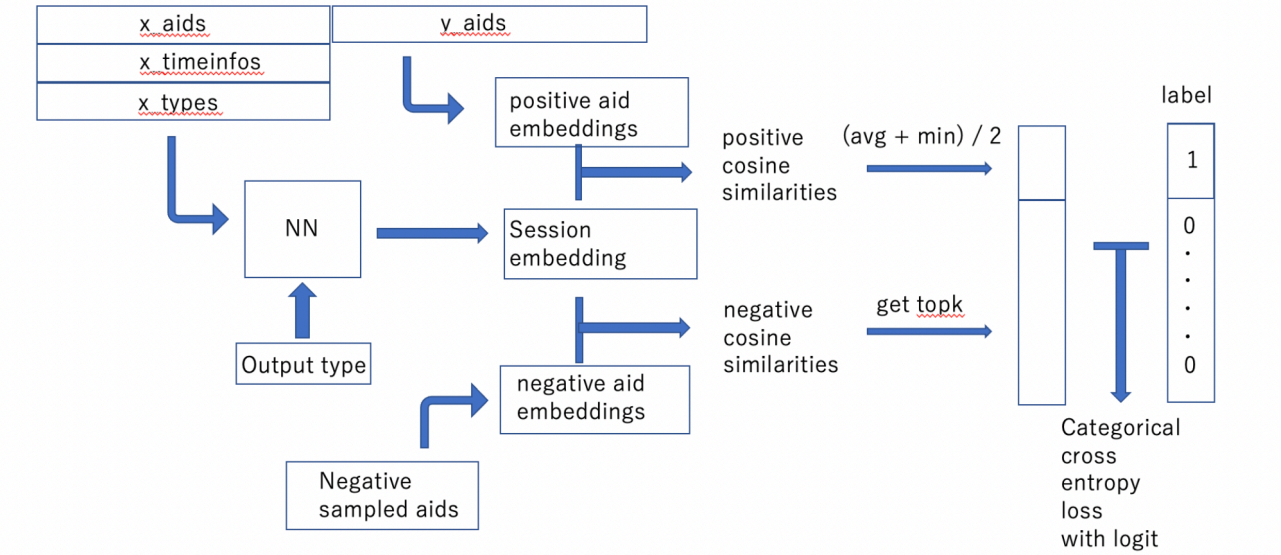
Training Methods

Training Methods

- About Neural network
  - What training methods did you use?
    - Two tower for NN used for candidate scoring. (I will explain in an after slide in detail)
  - Did you ensemble the models?
    - 8 NNs for both of recall and reranking features
  - If you did ensemble, how did you weight the different models?
    - Recall : Use all top200 candidates calculated from each NN
    - Reranking features : Use all scores from each NN as independent features like stacking.

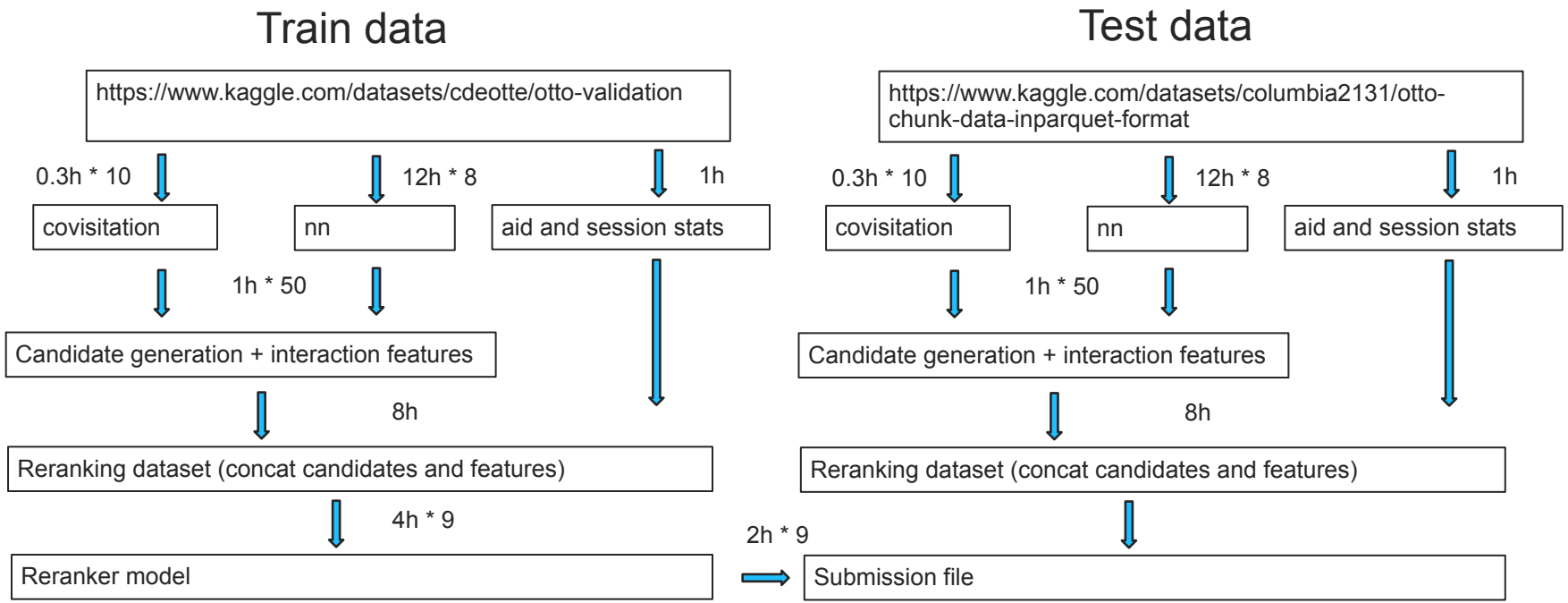
- About LGBMRanker
  - What training methods did you use?
    - lambdarank for LGBMRanker as reranking model
  - Did you ensemble the models?
    - 9 LGBMRankers with different hyperparameters
  - If you did ensemble, how did you weight the different models?
    - The LGBMRanker ensemble was performed by simple averaging of output scores.
      - I didn’t test other ensemble methods like voting or rank averaging

Training Methods



NN is trained with a two tower scheme with negative sampling.

final solution



Total time for serial execution is about 2weeks

Important and Interesting Findings

- NN is more effective than the covisitation for the prediction of clicks and carts.
  - Interaction information involving multiple aids may be important for predicting clicks and carts.
  - Or time information may be important.

- On the other hand, in predicting orders, the covisitation and revisitation information were more effective than NN.
  - Most part of ordering accuracy may be determined systematically, such that the aid that has already been put into carts is ordered. If so, complex models may not be needed in the current task setting.

Comparing with the second-place team's score by types

Private	clicks	carts	Orders	total
mine	**0.05705**	**0.13787**	0.4101	**0.60503**
2nd	0.05653	0.13749	**0.41044**	0.60446

Comparing the second-place team‘s score by type, I win at clicks and carts but lose at orders.
I think the reason for my victory was that I improved the accuracy for clicks and carts with NN.

Simple Model

Simple Model

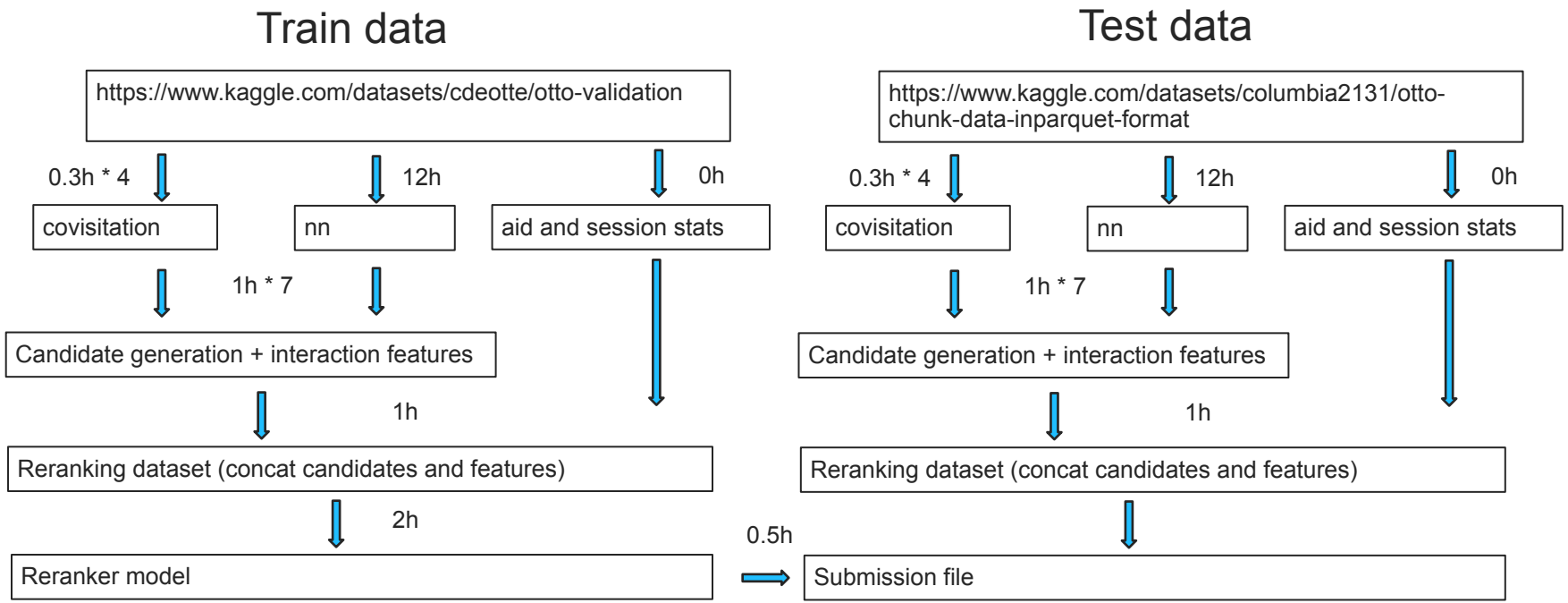
Private	Clicks	Carts	Orders	Total
final	0.05705	0.13787	0.4101	0.60503
simple	0.05533	0.13384	0.4048	0.59398

```
feats_clicks = ["emb_clicks_v42",
                "count_dic_v15_p_chain_all_20_2_all_0",
                "isin_0_rank",
                "count_dic_v15_p_chain_all_20_2_all_1",
                "isin_0_carts"
               ]
feats_carts = ["emb_carts_v42",
               "count_dic_v19_p_chain_w_20_2_all_0",
               "count_dic_v16_p_chain_all_100_2_all_0",
               "count_dic_v14_p_chain_all_20_2_all_1",
               "isin_0_carts"
              ]
feats_orders = ["isin_0_carts",
                "emb_orders_v42",
                "count_dic_v19_p_chain_w_20_2_all_0",
                "count_dic_v14_p_chain_w_20_2_all_1",
                "count_dic_v16_p_chain_all_100_2_all_0",
               ]
```

count_dic (= covisitation)
v15 : clicks weighted
v14, v19 : carts weighted
v16 : balanced weighting

- 1 NN, 4 covisitations, revisitation and 1 lgbm reranker as a simple solution.
- Total time : 2weeks -> 2days
- Clicks and carts are 3% worse and orders are 1% worse.
    - My complex modeling in my final solution seems to improve clicks and carts than orders.

Simple solution



Total time for serial execution is about 2days

Question and Answer

- Did you try non-GBM rankers? What was the performance/speed?
  - No, I didn't try.

- What would be your MVM (Minimal viable model for an optimal tradeoff of performance and complexity)
  - Covisitation and revisitation is a simple and enough strong baseline in the current metric.
  - Statistics of aid is not very strong, but I think it will be easy to adopt.
  - Neural network is strong, but I don't know if the cost is worth the KPI improvement.

- What were the trade-offs for choosing the final NN architecture? (e.g. Transformers vs. MLP)
  - AdamW vs SparseAdam. AdamW is slower but better than SparseAdam.
  - MLP vs Transformer. MLP is faster and slightly better than Transformer. Transformer may contribute to model diversity.
  - The dimension of embedding, the size of negative samples and the length of input sequence is trade-off between accuracy and computational cost.

- What loss function was used for NNs?
  - Categorical cross entropy loss. I used aids visited in future as positive examples and aids randomly sampled as negative examples.

```
# sampling negative aids
neg_aids = torch.randperm(CFG.n_aid, device = xy_batch.get_device())[:CFG.n_neg_coef * len(xy_batch)]

# calculate session embedding
x_em = self._calc_query_emb(x_batch) # (batch_size, CFG.emb_dim)
label_type_em = self.emb_label_type(label_type) # (batch_size, CFG.n_label, CFG.emb_dim)
x_em = x_em.unsqueeze(1) + label_type_em # (batch_size, CFG.n_label, CFG.emb_dim)

# calculate target aid embeddings
y_em = self.do_emb(y_batch) # (batch_size, CFG.n_label, CFG.emb_dim)
neg_em = self.do_emb(neg_aids) # (n_negative_sample, CFG.emb_dim)

# normalize emebddings
x_em = x_em/torch.norm(x_em, dim=2, keepdim = True)
y_em = y_em/torch.norm(y_em, dim=2, keepdim = True)
neg_em = neg_em/torch.norm(neg_em, dim=1, keepdim = True)

# cosine similarity with positive aids
cossim_pos_all = torch.sum(x_em * y_em, dim = 2) # (batch_size, CFG.n_label)
y_mask = y_batch.eq(CFG.n_aid - 1)
cossim_pos_min = torch.min(cossim_pos_all + y_mask, dim = 1)[0] # cosine similarity of most difficult positive sample
cossim_pos_mean = torch.sum(cossim_pos_all * label_type, dim = 1) / torch.sum(label_type, dim = 1)
cossim_pos = (cossim_pos_mean + cossim_pos_min) / 2 # (batch_size, )

# cosine similarity with negative aids
cossim_neg = torch.matmul(x_em[:,0], neg_em.T) # (batch_size, n_negative_sample)
cossim_neg = torch.topk(cossim_neg, CFG.neg_topk, dim = 1)[0] # get difficult negative samples

# calculate loss and accuracy
cossim = torch.cat([cossim_pos.unsqueeze(1), cossim_neg], dim = 1) / CFG.temperature
p = torch.softmax(cossim,dim=1)
loss = - torch.mean(torch.log(p[:,0])) # first sample in each batch is positive
acc = torch.mean((torch.max(p,dim=1)[1] == 0).to(torch.float))
return loss, acc
```

- How are the input features encoded?
  - aid : a categorical feature
  - ts : assuming hh = ts // 3600 // 1000,
    - hh (absolute hour as a categorical feature)
      - Average between pre and post hh embedding to make smooth embedding
    - sin(hh % 24 / 24 * np.pi * 2) (a numerical feature indicating time of a day)
    - cos(hh % 24 / 24 * np.pi * 2) (a numerical feature indicating time of a day)
  - type : numerical feature
  - Categorical features are embedded by a lookup table

```
aids = x_batch[:,:,0] # aid
hour = x_batch[:,:,1:2] # = ts // 3600 // 1000
types = x_batch[:,:,2:3] # clicks -> 1, carts -> 3, orders -> 6

# embedding from aids
em = self.emb(aids)

# other features like time in a day and type of aid
x_other = torch.cat([
    torch.sin(hour % 24 / 24 * np.pi * 2),
    torch.cos(hour % 24 / 24 * np.pi * 2),
    types,
], dim=2)

# embedding from hour
emh =   self.emb_h(hour) + \
        self.emb_h(torch.clip(hour+1, max = CFG.h_max-CFG.h_min+3)) + \
        self.emb_h(torch.clip(hour+2, max = CFG.h_max-CFG.h_min+3))

# concat all embeddings
x = torch.cat([em, x_other, emh], dim = 2)

# masking where no history aid
mask = ~x_batch[:,:,0].eq(CFG.n_aid-1).unsqueeze(2)
x = x * mask
x = x.reshape(len(x), -1) # (batch_size, CFG.n_length * sum_dim_embeddings_and_features)

# mlp to create session embedding
return self.mlp(x)  ## (batch_size, CFG.emb_dim)
```

• You wrote, "apply covisitation matrix at multiple times like beam search". Would you explain this in more detail, please?

When visited aid sequence is A -> B

aid	score
A	1
B	1
C	0
D	0

×

covisitation calculated in advance

	A	B	C	D
A	1	0	1	0
B	0	2	1	0
C	0	0	0	2
D	0	0	0	0

=

aid	score
A	1
B	2
C	2
D	0

Select topk aids (here k = 2)

aid	score
A	0
B	1
C	1
D	0

×

	A	B	C	D
A	1	0	1	0
B	0	2	1	0
C	0	0	0	2
D	0	0	0	0

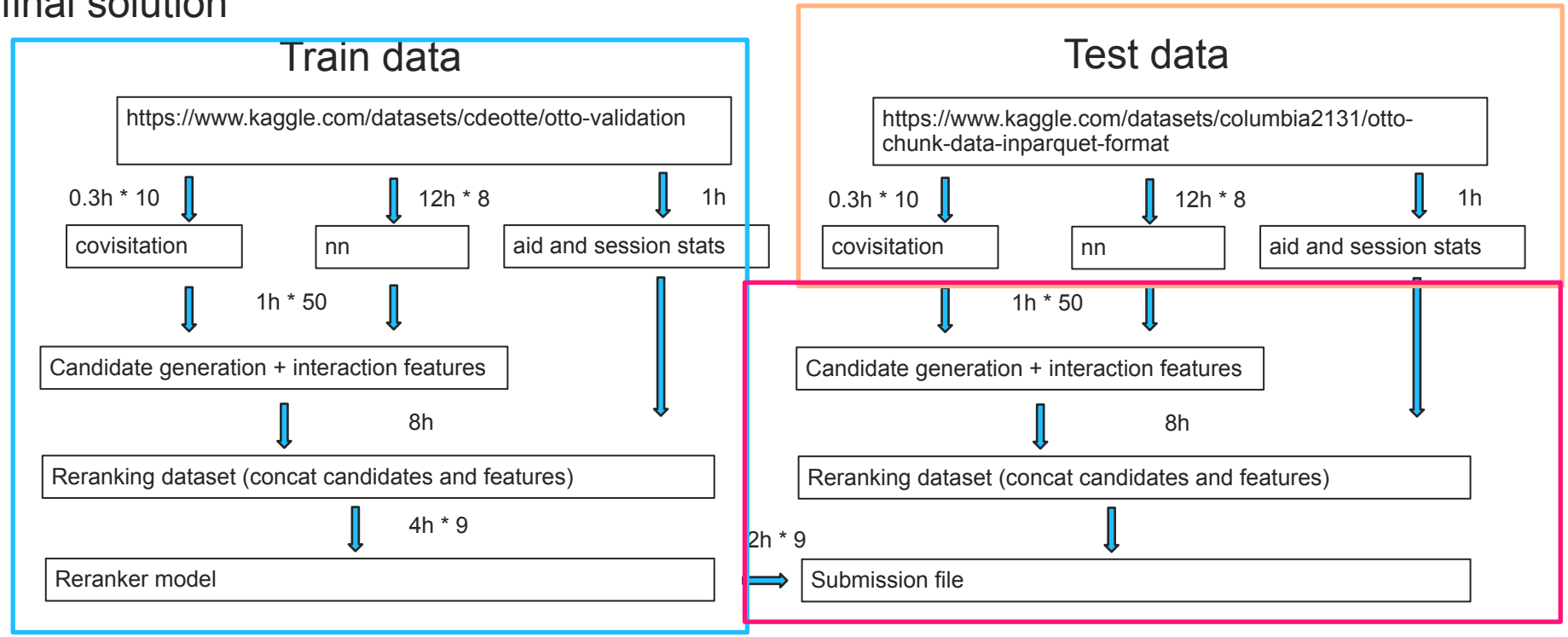
=

aid	score
A	0
B	2
C	1
D	2

kaggle

- You wrote that the total time for serial execution is about two weeks. How much of this time was spent on training, and how much on inference?

final solution



- 7 days for training on train data
- 4 days for training on test data (I retrained NN and recalculated covisitation and stats features by using test data)
- 3 days for inference (candidate generation, concatenation and lgbm inference)

kaggle

- You wrote, "I used multiple aids in future as positive targets." Did you apply this just for clicks?
  - I didn't aim for any particular type
  - I don’t know which type is improved.
  - Processing does not change depending on type.

- You wrote, "I tried to focus on samples that are not predicted well." How did you detect those?
  - I should have written ‘aids’ instead of ‘samples’.
  - I didn’t focus on difficult sessions.
  - About aids, I select and weight difficult aids by cosine similarity with a session embedding
    - regarding positive samples, aid whose cosine similarity is low
    - regarding negative samples, aids whose cosine similarity are high

- How large is k in your negative sampling part of the NN?
- How many negative items did you sample as input for the top-k?
  - The number is hyperparameter and varies from model to model.
  - In my best setting,
    - randomly sample 60000 aids for each batch (batch size = 2000)
    - calculate cosine similarity with session embeddings
    - select top 1000 aids for each session as negative samples

- Did you use the allowed test data „leakage“ and if so, how did you use it?
  - Yes
  - retrain NN on test data after training on train data
    - Especially, I use a absolute time feature. This cannot be used in a real situation.
  - calculate covisitation and statistics of aids by using test data
    - I use popularity of aid in a periods including future


kaggle