Readme

Part 1. Inner Product Model

High level idea:

- user i represented as a known vector $u_i \in \mathbb{R}^{15}$
- card j represented as an unknown vector $v_j \mathbb{R}^{15}$. This is what we need to learn.
- when card j assigned to user i: random reward $\sim$ i.i.d. Bernoulli($\langle u_i, v_j \rangle$)

To be more precise, we embed each user into $\mathbb{R}^{16}$ as following:

1. for each category[1] k, if its number of appearances is at least 5, then let $\tilde{u}^k$ be its average binge time.
2. normalize by L_1 norm, i.e. $\tilde{u} \leftarrow \tilde{u}/\|\tilde{u}\|_1$.
3. rescale by the "intensity" $\lambda(u) \in [0, 1]$ of this user:

$$u = \lambda(u)\tilde{u}$$

Part 2. Instructions of My Code

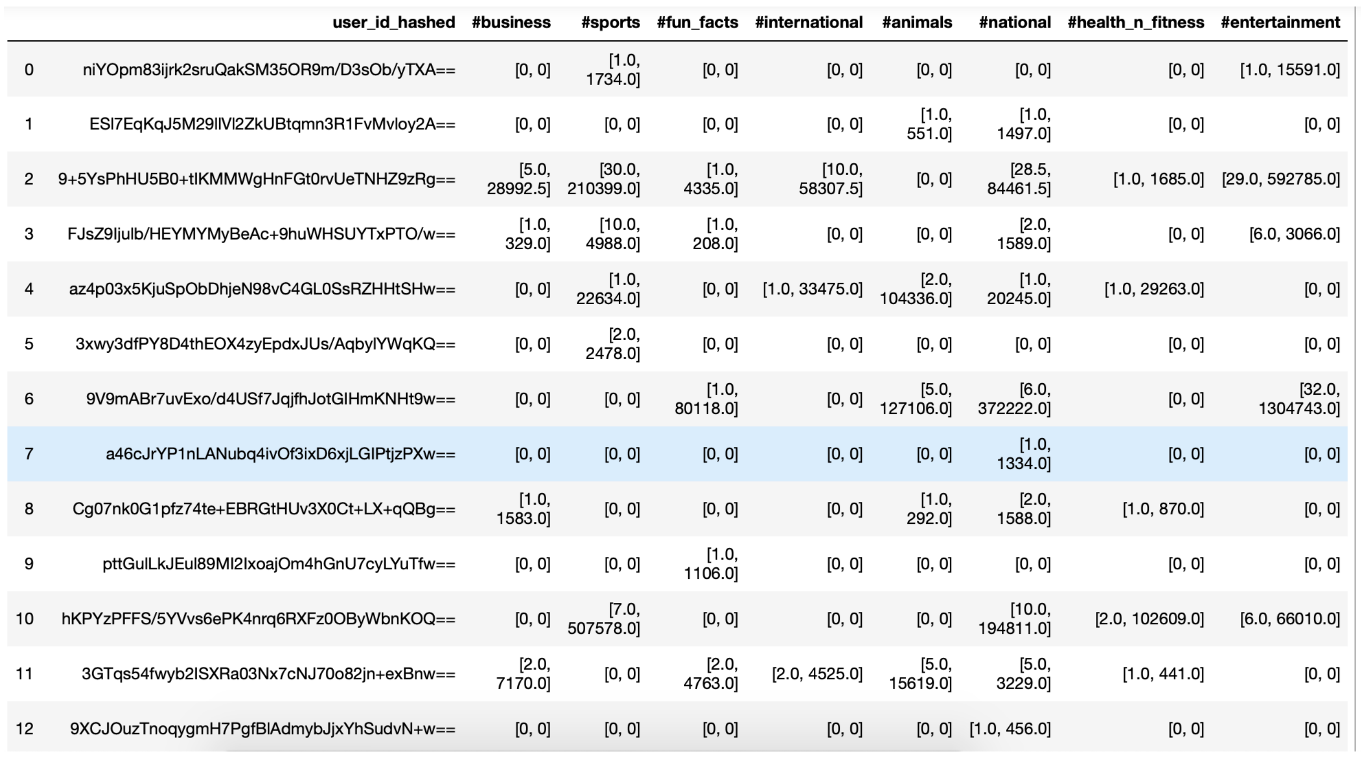
We will view the interaction data as a bipartite graph whose left nodes are users and right nodes are cards. Note that we will only consider binge edges, and **remove** all default edges. This is because we think the user behavior within binge sessions can better reflect the taste of the user and the popularity of the card.

My code is run using the first 10 parquets of interaction data (a million users and about 2.77 million edges).

Compute User Embedding. We first illustrate show how to embed any fixed user into $\mathbb{R}^{15}$.

1. Find the edges incident to this user.
2. Merge edges: there may be multiple edges between this user and a card, we will merge them into just one super-edge, and set its binge time to be the max binge time over the edges we merged. Subsequently we will use 'edge' to refer to super-edge.
   Function for this step: "merge_edges"
3. Compute user statistics ("user_stat"): loop over all edges. For each edge, if the corresponding card has k categories, then split the binge time evenly across these k categories. Remember to remove "daily digest", it does not count as a category. If the card has no real category, then we assign the binge time to the artificial class called "other". An example of the output "user_stat" looks like Fig 5.
   Functions for this step:
   - "compute_user_stat": compute the user statistics for just one user, return a dataframe.
   - "compute_user_stat_all": Loop over all users and invoke "compute_user_stat". Returns a dictionary whose keys are user_ids and values are dataframes.
4. Finally we compute the users activity intensity $\lambda(u)$ using "user_stat". $\lambda(u)$ is simply the average binge time per card, truncated and normalized so that it is within $[0, 1]$ for all users.

[1]except category "daily digest".



	user_id_hashed	#business	#sports	#fun_facts	#international	#animals	#national	#health_n_fitness	#entertainment
0	niYOpm83ijrk2sruQakSM35OR9m/D3sOb/yTXA==	[0, 0]	[1.0, 1734.0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[1.0, 15591.0]
1	ESl7EqKqJ5M29llVl2ZkUBtqmn3R1FvMvloy2A==	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[1.0, 551.0]	[1.0, 1497.0]	[0, 0]	[0, 0]
2	9+5YsPhHU5B0+tIKMMWgHnFGt0rvUeTNHZ9zRg==	[5.0, 28992.5]	[30.0, 210399.0]	[1.0, 4335.0]	[10.0, 58307.5]	[0, 0]	[28.5, 84461.5]	[1.0, 1685.0]	[29.0, 592785.0]
3	FJsZ9Ijulb/HEYMYMyBeAc+9huWHSUYTxPTO/w==	[1.0, 329.0]	[10.0, 4988.0]	[1.0, 208.0]	[0, 0]	[0, 0]	[2.0, 1589.0]	[0, 0]	[6.0, 3066.0]
4	az4p03x5KjuSpObDhjeN98vC4GL0SsRZHHtSHw==	[0, 0]	[1.0, 22634.0]	[0, 0]	[1.0, 33475.0]	[2.0, 104336.0]	[1.0, 20245.0]	[1.0, 29263.0]	[0, 0]
5	3xwy3dfPY8D4thEOX4zyEpdxJUs/AqbylYWqKQ==	[0, 0]	[2.0, 2478.0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
6	9V9mABr7uvExo/d4USf7JqjfhJotGIHmKNHt9w==	[0, 0]	[0, 0]	[1.0, 80118.0]	[0, 0]	[5.0, 127106.0]	[6.0, 372222.0]	[0, 0]	[32.0, 1304743.0]
7	a46cJrYP1nLANubq4ivOf3ixD6xjLGIPtjzPXw==	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[1.0, 1334.0]	[0, 0]	[0, 0]
8	Cg07nk0G1pfz74te+EBRGtHUv3X0Ct+LX+qQBg==	[1.0, 1583.0]	[0, 0]	[0, 0]	[0, 0]	[1.0, 292.0]	[2.0, 1588.0]	[1.0, 870.0]	[0, 0]
9	pttGulLkJEul89MI2IxoajOm4hGnU7cyLYuTfw==	[0, 0]	[0, 0]	[1.0, 1106.0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
10	hKPYzPFFS/5YVvs6ePK4nrq6RXFz0OByWbnKOQ==	[0, 0]	[7.0, 507578.0]	[0, 0]	[0, 0]	[0, 0]	[10.0, 194811.0]	[2.0, 102609.0]	[6.0, 66010.0]
11	3GTqs54fwyb2ISXRa03Nx7cNJ70o82jn+exBnw==	[2.0, 7170.0]	[0, 0]	[2.0, 4763.0]	[2.0, 4525.0]	[5.0, 15619.0]	[5.0, 3229.0]	[1.0, 441.0]	[0, 0]
12	9XCJOuzTnoqygmH7PgfBlAdmybJjxYhSudvN+w==	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[1.0, 456.0]	[0, 0]	[0, 0]

Figure 5. First 13 rows of the

Compute Card Embedding. We next show how to find the most likely embedding for any fixed card.

1. As before, we first invoke the function "merge_edges" to merge its incident edges.
   Functions: "merge_edges"

2. Embed Users: Then we embed the users into the Euclidean space, according to the model described before.
   Functions:

   - "find_U": will convert the user statistics into vectors in $\mathbb{R}^{15}$.
   - "remove_quote": each entry in Fig 5 is a string, e.g. '[a,b]'. this function removes the quote and returns a list [a,b]

3. Learn most likely embedding of the card. We will solve an MLE using projected gradient descent (due to the L_∞ norm constraint on v)
   Functions:

   - "embed_card": performs projected gradient descent
   - "log_likelihood": compute log likelihood

$$L(v|\{u_i, b_i\}) = \sum_i [\log(\langle u_i, v_i \rangle)^{b_i} + \log(1 - \langle u_i, v_i \rangle)^{1-b_i}].$$