Spark Coding Challenge

We have designed a coding challenge that:

1. Tests real software engineering skills, based on a real code base.
2. Allows for the use of AI (a very important real engineering skill)

This take-home test is more time consuming than the traditional algorithmic puzzles, but much better at assessing your true skills. It is also necessarily ambiguous, as AI can one shot clearly specified challenges. Deciding how to deal with ambiguity is part of the challenge. A strong engineer with AI should be able to solve this challenge within 3 hours. We know this time commitment is not for everyone, but if you do decide to participate, we thank you graciously for taking the effort! Furthermore we promise to:

- Assess your solution in detail
- Weigh the quality of your solution heavily in our hiring decision
- Interview you if you have a performant working solution.

Despite all of that, some senior / experienced candidates may feel they would prefer a face to face online coding interview. If you would prefer that, please reply and tell us this. For experienced candidates, who fit the role, we are willing to do that instead! This assignment is all about finding those candidates who are excited by the opportunity to prove themselves.

Please submit your solution within **7 days** from receiving this email**.** If you cannot attempt it this week that is no problem, just let us know when you will be attempting the solution[1].

Please note, the problem has been designed such that an AI on its own won't do a good job or won't write a good README. You'll need to work with AI to produce a good solution!

Submission Details

Please create a git repository **pyspark-coding-challenge** and make it public so we can review it. Include a README, and all code and tests relating to the task.

Raw Input

You are a data engineer in a machine learning team and our data scientists need to help prepare the data to train a transformer model. The base of the dataset are sequences of impressions. Each row represents a list of items that were shown to the customer in the app / website. Each impression comes from the website's "Our choices for you" carousel, at the top of the home page. Each row of the impression data set looks like this:

[1] If you do delay your solution beyond a week, there is a chance the role may have closed. Just let us know the date you intend to submit instead, and we can tell you how likely it is.

Impressions

dt: str - The date partition, of Hive format: dt=YYYY-MM-DD
ranking_id: str
customer_id: integer
impressions: Array[Item]

Where an **Item** object has the following properties:

- **item_id**: integer
- **is_order:** bool - a flag indicating whether or not the customer ordered the item

The impressions are generated once per day per customer, and only exist if the customer logged in to the product (and therefore saw the carousel). For this data set, there are about a million rows a day, where each list of impressions has 10 items. Each single impression forms an individual training example for our transformer model. The items themselves will be mapped to embeddings in the transformer, via their item_id. The data scientists will set up the model with embeddings, you just need to provide the article ID and their corresponding is_order target.

So far the only input feature we have is the embeddings derived from the item_d. The model also needs some information about the actions taken by the customer. There are 3 kinds of actions, clicks, add to carts and previous orders[2]. The raw data for the first two actions are stored in two different Kafka topics, shown below. In a 7 day period, we have about 150 million clicks and 15 million add to carts, and 2 million orders. You can assume that we have 10 million active users, in any given month.

Clicks

dt: str - The date partition, of Hive format: dt=YYYY-MM-DD
customer_id: integer
item_id: integer
click_time: timestamp - when the click occurred

Add to Carts

dt: str - The date partition, of Hive format: dt=YYYY-MM-DD
customer_id: integer
config_id: integer (same exact field as item_id)
simple_id: integer - the specific size of the item being added to cart.
occurred_at: timestamp - when the add to cart occurred

Previous Orders

These are instead stored in a table in our data warehouse
order_date: date - The date of the order
customer_id: integer
config_id: integer (same exact field as item_id)

---

[2] These are all historical actions, actions at least one day **before** the current impression (using dt fields). Otherwise we will have a data leak.

simple_id: integer - the specific size of the item being added to cart.
occurred_at: timestamp - when the add to cart occurred

Training Input

Our Data scientists are essentially developing a new algorithm for the “Our top choices carousel”, but for now we are just in a training / experimentation phase. You won’t be writing the training code, but this section describes our data scientists intent so you can write a Spark pipeline which produces the input for that data set. Exactly what that input is like, and the high level of how the data scientist will use it, is up to you.

In particular, the training will run on 14 days of data. The training will run in Pytorch on a single large GPU and iterate over the full training data set one day at a time, iterating over the same dataset multiple times (epochs). Within a single day must be randomly sampled (i.e. the data scientists will explode the impressions list and sample from all impressions across that day).

Notice each row in the impression data set contains a customer id. For a given day, we want to know the customer's **most recent** 1000 actions from all days before, in descending order[3], **going back at least 1 year**. The customers list of most recent actions will change every day (assuming the customer logs in). For each impression, we need to have this most recent sequence of actions for the corresponding customer id, as part of the model's input.

They will build a Pytorch model with the following forward method signature:

```
def forward(self,
            impressions: Tensor,
            actions: Tensor,
            action_types: Tensor) -> Tensor:
    pass
```

- **impressions** shape: [batch_size] A list of integers, the batch index corresponds to a single impression. The integer value itself corresponds to the embedding index of the item in the impression.

- **actions**: shape: [batch_size, 1000] - Each batch index also corresponds to a particular single impression, the sequence of 1000 actions at each batch index, corresponds to the customer id associated with that impression (on that dt). The integer values themselves correspond to the embedding index of the clicked / add to cart item in the action. (0 for missing actions, padding for when a customer has less than 1000 actions in their history)

- **action_types**: shape: [batch_size, 1000] - Just like actions above except each integer value can be one of four values. (1 for clicks, 2 for add to carts, 3 for previous orders and 0 for missing actions).

[3] Don’t worry about duplicates, that is in fact useful information!

Your Requirements

In this work, you do not need to write the Pytorch code. You can assume that the work to:

- load the data you produce
- Map the item IDs to embedding index
- Write the sampling / training code
- The model itself

Will be done in a future iteration. However, as the technical leader, you do need to provide training input data for this part, and provide a logical high level description for how it will work, just a few paragraphs, no more than one page in the README.

You need to implement a Spark Data pipeline that takes the raw data as inputs and produce the training inputs. It is up to you to determine how that data is structured, but it should be structured to make the training part above run fast, **ideally the GPU would be fully utilized**.

Here is specifically what we are looking for:

1. You should describe a high level proposed structure for the training inputs, and a high level description of how the training will work in the README.

2. You need to write one or more PySpark pipelines to produce the training input data sets. You pipelines should accept Dataframes as inputs, so it can be run easily by us, with our own version of the raw inputs[4].

3. Those training inputs should be correct, in accordance with the specs above. For example, each action sequence really needs to be the customer's last 1000 actions (and the action types need to be correct too).

4. The PySpark should be performant, in the sense it should be written in such a way that it would be fast at scale and use compute resources effectively. It would be helpful to you to at least write comments about this, or explain what you are doing in relation to performance in the README.

5. We would like to see comprehensive test cases written, as if this were a production application, please use pytest. At least for the unit tests, we would like to be able to run these locally.

6. The README should also describe the training inputs you are producing, the pipelines you created to produce them, and why you are producing those pipelines.

[4] Given the open nature of this question, we accept we might have to do a bit of work to make sure we can run your solution on our Databricks clusters, but the idea should be to write your transformations in standard PySpark.