# Training

This folder contains the sql files that we'll use for training the prod real graph models:

- prod (predicts any interactions the next day)

- prod\_explicit (predicts any explicit interactions the next day)

We have 3 steps that take place:

- candidate generation + feature hydration. this query samples 1% of edges from the `twttr-recos-ml-prod.realgraph.candidates` table which is already produced daily and saves it to `twttr-recos-ml-prod.realgraph.candidates\_sampled`. we save each day's data according to the statebird batch run date and hence require checks to make sure that the data exists to begin with.

- label candidates. we join day T's candidates with day T+1's labels while filtering out any negative interactions to get our labeled dataset. we append an additional day's worth of segments for each day. we finally generate the training dataset which uses all day's labeled data for training, performing negative downsampling to get a roughly 50-50 split of positive to negative labels.

- training. we use bqml for training our xgboost models.

## Instructions

For deploying the job, you would need to create a zip file, upload to packer, and then schedule it with aurora.

```

zip -jr real\_graph\_training src/scala/com/twitter/interaction\_graph/bqe/training && \

packer add\_version --cluster=atla cassowary real\_graph\_training real\_graph\_training.zip

aurora cron schedule atla/cassowary/prod/real\_graph\_training src/scala/com/twitter/interaction\_graph/bqe/training/training.aurora && \

aurora cron start atla/cassowary/prod/real\_graph\_training

```

# candidates.sql

1. Sets the value of the variable date\_candidates to the date of the latest partition of the candidates\_for\_training table.

2. Creates a new table candidates\_sampled if it does not exist already, which will contain a sample of 100 rows from the candidates\_for\_training table.

3. Deletes any existing rows from the candidates\_sampled table where the ds column matches the date\_candidates value, to avoid double-writing.

4. Inserts a sample of rows into the candidates\_sampled table from the candidates\_for\_training table, where the modulo of the absolute value of the FARM\_FINGERPRINT of the concatenation of source\_id and destination\_id is equal to the value of the $mod\_remainder$ variable, and where the ds column matches the date\_candidates value.

# check\_candidates\_exist.sql

This BigQuery prepares a table of candidates for training a machine learning model. It does the following:

1. Declares two variables date\_start and date\_end that are 30 days apart, and date\_end is set to the value of $start\_time$ parameter (which is a Unix timestamp).

2. Creates a table candidates\_for\_training that is partitioned by ds (date) and populated with data from several other tables in the database. It joins information from tables of user interactions, tweeting, and interaction graph aggregates, filters out negative edge snapshots, calculates some statistics and aggregates them by source\_id and destination\_id. Then, it ranks each source\_id by the number of days and tweets, selects top 2000, and adds date\_end as a new column ds.

3. Finally, it selects the ds column from candidates\_for\_training where ds equals date\_end.

Overall, this script prepares a table of 2000 candidate pairs of user interactions with statistics and labels, which can be used to train a machine learning model for recommendation purposes.

# labeled\_candidates.sql

The BQ does the following:

1. Defines two variables date\_candidates and date\_labels as dates based on the $start\_time$ parameter.

2. Creates a new table twttr-recos-ml-prod.realgraph.labeled\_candidates$table\_suffix$ with default values.

3. Deletes any prior data in the twttr-recos-ml-prod.realgraph.labeled\_candidates$table\_suffix$ table for the current date\_candidates.

4. Joins the twttr-recos-ml-prod.realgraph.candidates\_sampled table with the twttr-bq-cassowary-prod.user.interaction\_graph\_labels\_daily table and the twttr-bq-cassowary-prod.user.interaction\_graph\_agg\_negative\_edge\_snapshot table. It assigns a label of 1 for positive interactions and 0 for negative interactions, and selects only the rows where there is no negative interaction.

5. Inserts the joined data into the twttr-recos-ml-prod.realgraph.labeled\_candidates$table\_suffix$ table.

6. Calculates the positive rate by counting the number of positive labels and dividing it by the total number of labels.

7. Creates a new table twttr-recos-ml-prod.realgraph.train$table\_suffix$ by sampling from the twttr-recos-ml-prod.realgraph.labeled\_candidates$table\_suffix$ table, with a downsampling of negative examples to balance the number of positive and negative examples, based on the positive rate calculated in step 6.

The resulting twttr-recos-ml-prod.realgraph.train$table\_suffix$ table is used as a training dataset for a machine learning model.

# train\_model.sql

This BQ command creates or replaces a machine learning model called twttr-recos-ml-prod.realgraph.prod$table\_suffix$. The model is a boosted tree classifier, which is used for binary classification problems.

The options provided in the command configure the specific settings for the model, such as the number of parallel trees, the maximum number of iterations, and the data split method. The DATA\_SPLIT\_METHOD parameter is set to CUSTOM, and DATA\_SPLIT\_COL is set to if\_eval, which means the data will be split into training and evaluation sets based on the if\_eval column. The IF function is used to assign a boolean value of true or false to if\_eval based on the modulo operation performed on source\_id.

The SELECT statement specifies the input data for the model. The columns selected include label (the target variable to be predicted), as well as various features such as num\_days, num\_tweets, and num\_follows that are used to predict the target variable.