# Scoring

This folder contains the sql files that we'll use for scoring the real graph edges in BQ. We have 4 steps that take place:

- check to make sure that our models are in place. the feature importance query should return 20 rows in total: 10 rows per model, 1 for each feature.

- follow graph feature generation. this is to ensure that we have features for all users regardless if they have had any recent activity.

- candidate generation. this query combines the candidates from the follow graph and the activity graph, and the features from both.

- scoring. this query scores with 2 of our prod models and saves the scores to a table, with an additional field that distinguishes if an edge in in/out of network.

## 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\_scoring src/scala/com/twitter/interaction\_graph/bqe/scoring && \

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

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

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

```

# candidates.sql

This BigQuery (BQ) query does the following:

1. Declares two variables, date\_start and date\_end, which are both of type DATE.

2. Sets the date\_end variable to the maximum partition ID of the interaction\_graph\_labels\_daily table, using the PARSE\_DATE() function to convert the partition ID to a date format.

3. Sets the date\_start variable to 30 days prior to the date\_end variable, using the DATE\_SUB() function.

4. Creates a new table called candidates in the realgraph dataset, partitioned by ds.

5. The query uses three common table expressions (T1, T2, and T3) to join data from two tables (interaction\_graph\_labels\_daily and tweeting\_follows) to generate a table containing candidate information and features.

6. The table T3 is the result of a full outer join between T1 and T2, grouping by source\_id and destination\_id, and aggregating values such as num\_tweets, label\_types, and the counts of different types of labels (e.g. num\_follows, num\_favorites, etc.).

7. The T4 table ranks each source\_id by the number of num\_days and num\_tweets, and selects the top 2000 rows for each source\_id.

8. Finally, the query selects all columns from the T4 table and appends the date\_end variable as a new column named ds.

Overall, the query generates a table of candidates and their associated features for a particular date range, using data from two tables in the twttr-bq-cassowary-prod and twttr-recos-ml-prod datasets.

# follow\_graph\_features.sql

This BigQuery script creates a table twttr-recos-ml-prod.realgraph.tweeting\_follows that includes features for Twitter user interactions, specifically tweet counts and follows.

First, it sets two variables date\_latest\_tweet and date\_latest\_follows to the most recent dates available in two separate tables: twttr-bq-tweetsource-pub-prod.user.public\_tweets and twttr-recos-ml-prod.user\_events.valid\_user\_follows, respectively.

Then, it creates the tweet\_count and all\_follows CTEs.

The tweet\_count CTE counts the number of tweets made by each user within the last 3 days prior to date\_latest\_tweet.

The all\_follows CTE retrieves all the follows from the valid\_user\_follows table that happened on date\_latest\_follows and left joins it with the tweet\_count CTE. It also adds a row number that partitions by the source user ID and orders by the number of tweets in descending order. The final output is filtered to keep only the top 2000 follows per user based on the row number.

The final SELECT statement combines the all\_follows CTE with the date\_latest\_tweet variable and inserts the results into the twttr-recos-ml-prod.realgraph.tweeting\_follows table partitioned by date.

# scoring.sql

This BQ code performs operations on a BigQuery table called twttr-recos-ml-prod.realgraph.scores. Here is a step-by-step breakdown of what the code does:

Declare two variables, date\_end and date\_latest\_follows, and set their values based on the latest partitions in the twttr-bq-cassowary-prod.user.INFORMATION\_SCHEMA.PARTITIONS and twttr-recos-ml-prod.user\_events.INFORMATION\_SCHEMA.PARTITIONS tables that correspond to specific tables, respectively. The PARSE\_DATE() function is used to convert the partition IDs to date format.

Delete rows from the twttr-recos-ml-prod.realgraph.scores table where the value of the ds column is equal to date\_end.

Insert rows into the twttr-recos-ml-prod.realgraph.scores table based on a query that generates predicted scores for pairs of user IDs using two machine learning models. Specifically, the query uses the ML.PREDICT() function to apply two machine learning models (twttr-recos-ml-prod.realgraph.prod and twttr-recos-ml-prod.realgraph.prod\_explicit) to the twttr-recos-ml-prod.realgraph.candidates table. The resulting predicted scores are joined with the twttr-recos-ml-prod.realgraph.tweeting\_follows table, which contains information about the number of tweets made by users and their follow relationships, using a full outer join. The final result includes columns for the source ID, destination ID, predicted score (prob), explicit predicted score (prob\_explicit), a binary variable indicating whether the destination ID is followed by the source ID (followed), and the value of date\_end for the ds column. If there is no match in the predicted\_scores table for a given pair of user IDs, the COALESCE() function is used to return the corresponding values from the tweeting\_follows table, with default values of 0.0 for the predicted scores.