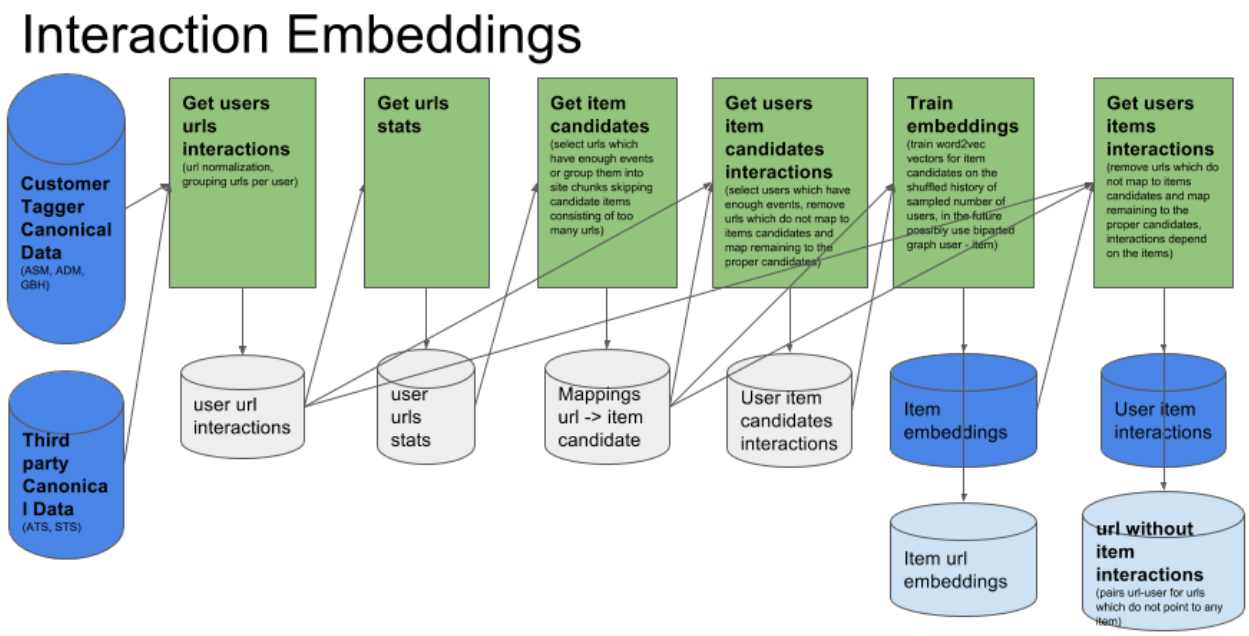
Interaction embeddings

The old and obsolete documentation of this pipeline is in Interaction module



Task: produce item embeddings and prepare user - item interaction data which is the input for the embedder producing user embeddings.

On the diagram above shaded blue are optional outputs which are currently not used but where considered to be use in the past and pipeline can produce them.

Definitions

item candidate - url or site chunk (if urls grouped to this site chunk don't have enough events) having more than given threshold events.

item - candidate item for which we generated embeddings. Items are different depending on the embedding version.

Steps

Prepare training data

Step consisting of 4 different steps:

Get user urls interactions

1. Find the data form the last run which has been run on the same **days_back** value and based on this calculate which date ranges should be calculated.
   That is:
   a. Provided that we want to run on the given **to** and **days_back** values find biggest **to'** such that **to - days_back** < **to' <= to** for which we have data with the given **days_back** value.
   b. Then we can calculate events on **[start', start)** where **start = to - days_back** and **start'** = **to' - days_back** and **[to', to)**.
   c. If no previous run data found satisfying above condition just calculate events for **[start, to)**
2. Read canonical data (e.g. from s3a://1plusx-data/canonical/v3/ADM/tagger/Interaction/) for the selected time range(s).
   a. Reduce event to user id, url and timestamp
   b. Apply urls canonicalization (removing query parameters, space normalization, etc.)
   c. Filter users having on average more than than given **max_user_events_per_day** value (in practice it's 1000). Users with bigger number of events are probably bots
   or other kind of outlier users.
3. Merge the data if needed:

    a. Read user url interaction from **[start', to')**
    b. Find the max timestamp in the data from "too old" range (**[start', start)**) and remove all interactions with the timestamp smaller equal it.
    c. Add interactions from **[to' to)**
4. Save the new user url interactions.

After this step calculate user url interactions number. It will be used in some of the next steps

Get urls stats

Extract and save needed stats. In practice usually the only thing computed and returned is number of events for each url.

Get item candidates

1. Algorithm to find items candidates:
    a. Look through the url events stats from urlEventsCounts
        i. If the url has events counts >= MIN_EVENTS extract mapping url url
        ii. Otherwise create mapping (site_chunk, level) url. Assumption is the site chunk can have maximum level 20.
    b. Look through the site chunks from level 20 up to 0
        i. Group urls mapped to the given site chunk
            1. If the site chunk has events counts >= MIN_EVENTS extract mappings url site chunk for the urls mapped to the given site chunk
            2. Otherwise strip the last chunk of the given site chunk and assign it all urls previously collected for the given site chunk
        ii. On the site chunk level 0 map all site chunks which do not have enough events to domains
    c. Look through domains
        i. Group urls mapped to the given domain and output mapping url domain for the domains having enough events

Get user item candidates interactions

1. Filter users having more than given threshold events (it doesn't make sense to run word2vec on the user which has just couple events in his history)
2. Sample given number of ids.
3. Load url interactions for the sampled users
4. Join interactions with the mappings url item candidate
5. Group interactions by user id and shuffle them to make sure that their order is random
6. Save interactions

Train embeddings

1. Train Spark ML word2vec on user item candidate interactions
2. Save generated item vectors. Items are the subset of item candidates.
3. There's an option to output item url embeddings but usually it's disabled because they're not needed (item embeddings + mappings url item candidate is enough)

Get users items interactions

1. Read user url interactions
2. Count them and divide by DefaultUserUrlsEventsPerPartition (400K) to get the value of the spark.sql.shuffle.partitions argument which is increased from the default 200
   if needed so that we don't get errors during join because of too big partitions.
3. Reduce user, item interactions to their count so that we get pairs (user-item interaction, count). It's needed so that later when we group user events we don't get too much events for the single user.
   Maybe filtering too frequent users in the first pipeline step is enough but it's better to be on the safe side.
4. Get mappings url -> item split by popularity
   The idea is that sometimes mappings url -> item are too big to broadcast all of them so we split on those which are very popular (have a lot of interactions and if we join on them all those interactions would go into a single partition) and those which are less popular on which we can safely do the join
    a. Read and broadcast items set.
    b. Get previously collected counts of interactions per url (they were needed to generate item candidates).
    c. Read RDD of mappings url -> item candidate and filter only those mappings which candidates actually became items.
    d. Join mappings with their url interactions counts into triples (item, mappings strings length, url interactions number)
    e. Group triple for the given item and calculate their total length and interactions number
    f. Sort triples by decreasing interactions number
    g. Find the largest index such that the sum of lengths of mappings is below the given threshold size (so that we can broadcast them)
    h. Broadcast popular items set
    i. Mark mapping with the boolean information if their item is in popular items set or not

    j. Return dataset of popular and less popular items.
5. Broadcast the map of the mappings of popular items
6. Filter interactions with urls mapped to popular items and map these interactions to interactions with items
7. Filter interactions without urls mapped to popular items
8. Join interactions with the mappings for less popular items and map them to interactions with less popular items
9. Join item interactions for popular and less popular items, group it by user id and output each user items history

Things to fix in the future

1. Currently there's a problem that we can have a large cluster which remains idle during the phase of copying from temp directory on S3 to the final location.
2. At the beginning of the development of the pipeline there was no Luigi pipeline so the logic of constructing the paths has been put into spark jobs. Not it starts to cause some troubles so in the future it would be good to move the path construction into Luigi
3. In the past there was a concept of running multiple sources so spark jobs have config and task config, but since it's always run for the single source it's more confusing and leads to more code so could be removed in the future.
4. Still in some stages some workers die - could be investigated.
5. Adding possibility to train using non-Spark word2vec and increase the users sample size.
6. Variable user history length - keep the longer user history just for the users with a lot of events discard their events with higher probability also maybe with higher probability discard old events. Also analyze less eventers do they disappear forever.