Motivation

PubMed publications are always increasing. Citation analysis often requires the latest available data so an easy way to update PubMed records is valuable. Additionally, the citation networks analyzed may require all the features of interest (i.e. Publication year, MeSH list, titles and abstracts) to be present in order for an analysis. This is why this software was developed. It gets a directory of baseline tar.gz files and uploads a table to Postgres indicating if these features are present or not for a given DOI and PMID.

Steps to run this pipeline

These are the steps to run this pipeline. As they could be RAM/CPU intensive, they are better to be run separately (followed by some manual checks) rather than altogether. The code is stored in `/shared/pubmed_etl` .Below are the steps:

Step 1: Download the whole baseline tar.gz files

Each tar.gz file holds 30,000 records. They are nearly 1600 of such files when this file is created, but they will keep increasing. PubMed shares all these files on their website. They all shall be downloaded and placed in a directory. Let's call it `/shared/pubmed_baseline` for example.

An easier way is to construct a local copy of PubMed using EDirect:
https://www.nlm.nih.gov/dataguide/edirect/archive.html

Step 2: Extracting all the tar.gz files into xml files

a) cd to `/shared/pubmed_etl`
b) Run `python3 extractor.py -gz /shared/pubmed_baseline -cores 24`

The number of cores is arbitrary but 24 is a conservative value on Valhalla. No venv shall be required for this step. It modifies the tar.gz files in place, converting them to xml files. A good sanity check is to inspect the directory to make sure that the number of files is constant.

Step 3: Create a venv with required packages

Install these packages to run next steps:
-pyspark
-xmltodict
-pandas

Step 4: Extracting the fields of interest from xml files

So far we only care about the features we discussed. This step creates a parquet file corresponding to each xml files. In that it stores the raw features of interest (year, MeSH, title, abstract) into a parquet file, which later we can also extract the presence of each features. All these parquet files will be stored in an already existing directory which I will call `/shared/parquets` . Later I will use Pyspark to aggregate all these parquet files and generate the required table for the presence of features.

If new metadata are to be added (say, grants for instance), this is the step that needs to be changed. Don't forget to activate the virtual environment with the packages mentioned above.

a) cd to `/shared/pubmed_etl`
b) Run `python3 parallel.py -xml /shared/pubmed_baseline -parquet /shared/parquets -cores 48 -wrap 0`
c) Run `python3 parallel.py -xml /shared/pubmed_baseline -parquet /shared/parquets -cores 48 -wrap 1`

The wrap 1 flag makes sure that all the xml files are mapped to a parquet files. So initially run the wrap 0 command and then followed by a wrap 1. Make sure machine is not under load when running this. 48 cores is a conservative number in this scenario. Once b) is run, it may take a minute or two since the first parquet file shows up in the `/shared/parquets` directory, and every new parquet file will take 1.5 seconds to be generated. A very important sanity check is to make sure that the number of files here match the number of xml files in `/shared/pubmed_baseline` . Note that a file in `/shared/pubmed_baseline` may not be xml file but just metadata about the directory, so don't count that.

Step 5: Aggregating the parquet files and loading the result into Postgres
This step aggregates all the parquet files, checks for presence of features of interest, de-duplicates the records and only holds the latest record of duplicate records and loads them into Postgres as a table. Do this if you want to have the results as a table in Postgres:

a) cd to `/shared/pubmed_etl`
b) Run `spark-submit --master local[18] --jars './postgresql-42.5.2.jar' --driver-memory 180g  --conf "spark.local.dir=./logs" pyspark_parquet.py --tname "public.pubmed_etl" --user "your_own_username" --pas "your_own_password" --path "/shared/parquet_summer/"`

If you like to dump results in form of parquet rather than loading into Postgres, run this, where `/shared/parquet_dump` is the path you want to store your resulting parquet file:

c) cd to `/shared/pubmed_etl`
d) Run `spark-submit --master local[18] --jars './postgresql-42.5.2.jar' --driver-memory 180g  --conf "spark.local.dir=./logs" pyspark_parquet.py --path "/shared/parquet_summer/" --parquet --dump_path '/shared/parquet_dump'`

This step could take 30 minutes. The 18 is the number of cores and 180g is the driver memory. Make sure you clean the logs in `./logs` once the process is done. Not that the user_name and password shall be replaced by your Postgres credentials. Once the process starts running, ignore the logs or warnings on the terminal. The process has failed only if you see an exception thrown in the terminal logs.
Once the process is complete, you should be able to access the table in `public.pubmed_etl`

Developed by Hossein.
Questions can be directed to hm31@illinois.edu
Link to GitHub: https://github.com/illinois-or-research-analytics/pubmed_etl/tree/main