Twitter Analysis DB Details

The main page for the project is **Twitter Analysis DB - OpenCircuits**

Contents

General

Look at the GUI Twitter Analysis DB GUI and just try it out. This page of documentation is both operational, that is using the app, and theoretical, as the ultimate guide to the app is to read the code. Module, class, and method names are fairly stable, but subject to change.

Performance

I currently run the db on a ram disk, in any case put on your fastest drive.

I have not tuned the db for performance. Got suggestions (that are more than just guesses) Let me know. I am sure that more indexing might help, will try in time.

Code is not optimized either. For most of it this does not matter. Looks like DB performance is the main restriction. Not an issue for me now.

> Concordance for 4 years of data with count of each use of about 10K words, grouped and sorted requires about 5 seconds. This is a three way join and is probably one of the most taxing.

Debugging

Initial debugging will probably consist of

- tweaking the parameter file for your computer
- Installing missing modules / packages.
- Check your console for messages
- Check the Python Log File (see button in the GUI)

The program does not come with an installer You should drop the code into your Python 3.6 development environment.

see: Twitter Analysis DB Configure to Run.

The Database

To understand the functionality of the application you should have some understanding of the data that it draws on: the database. That is what I will I will do in this section. If you wish skip over the more technical info and go right to the table descriptions.

The application is designed so that using it does not require writing any SQL (Structured Query Language), however, some familiarity with SQL and relational databases may be useful. This is not an introduction to SQL but is a description of some of the ways it has been applied in this application.

First the database is really a database, it is not a csv file or a spreadsheet. To make best use of this tool it really helps to have some understanding of the database, so read this. It is an sql db in sql lite that could be upgraded to a more robust db if worth while.

Tables can of course be joined, tweets to concord on tweet_id, and concord to words on word, the latter join may of course fail (produce a null in one table of the other so outer joins should be used.

Standard sql tools outside of the app can be used to view the data, Sqllite Studio, for example.

Table Structure

Currently it consists of three tables:

Tweets

This is data drawn directly from twitter and restructured to be loaded into a database table. You want to select against this table if you want to see the tweet text. This is also the only table with date information.

tweets -- the actual tweet and supporting columns:

- tweet_id -- identifies the tweet
- tweet_datetime -- when tweeted, has both a date and a time. I have extracted an extra column from it:
- time_of_day -- the time of day of the tweet. For both date and time I am uncertain how these relate to the timezone of the tweet record and of the tweeter.
- tweet -- text of the tweet
- who -- who tweeted -- so far no real support for multiple tweeters, but not too hard to add to the rest of the application. Is populated in the database load.
- tweet_type -- a tweet by the author or a retweet
- is_covid -- and indicator that suggest the tweet is covid related or not. Not in source data, generated by the table load routine, for now see the code.

Concordance

This is a concordance (for background a concordance is an index of all words used in a body of text, here tweets, cross referenced to the tweets text, perhaps first produced as a reference to the bible. For more information see: ***Concordance (publishing) - Wikipedia**) A concordance gives insight into the vocabulary of the author, and since tweets are dated we can also see how he vocabulary changes over time. While the concordance table is only generated at database load time each select against the table can be viewed as creating a new concordance.

Commonly abbreviated in the app as concord, it is a table generated from the tweets by extracting each "word" of the tweet. "Word" is in quotes because quite a bit of processing is involved in determining what is a word. The simple way would be to just break the tweets up on blank spaces, but what is actually done is a bit more complicated as described in see **Building a Database** section below.

concord, or concordance -- this is every "word" from all the tweets. A concord.word has be sanitized from the raw tweet, it is always lower case and "dirt" like punctuation has been removed.

- word -- the cleaned up word
- tweet_id -- link to the tweet
- word_type -- just a word, or a "@reference" or a hashtag
- is_ascii -- indicator if the entire word is encoded in ascii, this help identify words that are possibly non-words like emogies.

Words

This is a table of words taken from common usage and then loaded into a database table "words". It may help you see how the usage of vocabulary in the tweets corresponds/differs from the more general use of the language.

words -- a list (currently about 300 K) of words and information on the amount of usage.

- word -- the word (again all lower case and normalized)
- word_count -- count of the uses in the analyzed body of text.
- word_rank -- index starting at 1 and ascending from the highest_word_count to the lowest.

Advanced SQL

Advanced SQL has been avoided to keep the application fairly independent of the database system used. Some extra "magic" has been use in the Python to provide similar features. Avoiding too many joins also helps to keep the db up to speed.

Pseodo Columns and Output Transforms

The application contains some pseodo columns and some "output transforms" (not in the orcale sense, but a bit unique to this application) to provide some additional functionality. So for example word_type is stored in the database as an integer, but is decoded to strings like "Regular Word", "hashtag"..... There is quite a bit more to this that I should document in the future.

Building a Database

DB building facilities are available from the GUI. Since this is sensitive to the input sources it only works with the type of input sources I have used. Building involves several steps:

- Create a db
- Define the tables
- Load the tables

Parameters

First to enable the GUI features you need to adjust the parameter file so

    self.show_db_def = True

Then you also need to point to the input files: (these are in the github repo)

- self.tweet_input_file_name = r"./input/all_tweets_may_16_for_2020.txt" # where tweets are
- self.word_input_file_name = r"./input/english-word-frequency/unigram_freq.csv" # word frequency data from kaggal

Then some processing options:

- self.who_tweets = "djt" # id for who tweets, not much used yet
- self.use_spacy = True # processing words to lemmas

Name a database, I would start with a file name that does not exist, as defining and loading the db will erase any current content. Or if you are only defining some of the tables, or adding to a table start with an existing db.

    self.self.database_name = "tweet_big_words.db" # database with lots of words loaded.

Run the GUI

- <Show Load Parameters> will show you the values of some of the parameters used to load the db. If you do not like what you get edit the parameter file. (and there is a button for that too)
- <Define Tweets Concord> will clear the tweets and concord tables, and initialize the columns in them.
- <Load Tweets File> will load the tweet input file and populate both tweets and concord tables.
- <Define Words> will clear the words table, and initialize the columns in it.
- <Load Words> will load the words input file and populate the words table.

- as loaded the words table does not populate the words.words_rank column, there is another utility to do that that still needs to be added to the GUI

Some Code and Theory

Tweets/Concordance

These tables are loaded at the same time since the tweets table actually contains all the information in the concord table.

load_tweets_data.py contains the code for defining and loading these tables. define_table_tweets() and define table_concord methods ... do the definition work.... read the code.

A class TweetFileProcessor performs the table loads. Again read the code. The general idea is:

- Kick off from the main app from the method cb_load_tweets()
- Create TweetFileProcessor and its helpers TweetTableWriter and ConcordTableWriter, open the db, and input file
- Start processing with TweetFileProcessor.read_process_lines
- Open the file and connect to the db
- Read the lines in the file one by one to process them
- ..... parse the line ... split on tabs to get the individual source fields
- .... create parse_tweet (class ParsedTweet) from the line. ParsedTweet will parse the tweet (just the main text, the datetime.... have already been split off and saved
- .... in the parsing the follow take place.
    - tweet data is shifted to lower case.
    - certain characters are replaced
    - words are categorized (like url's @words, ...)
    - if certain words are present is_covid is set to true.
    - if parameters.use_spacy is True then spacy is used to convert words to lemmas
- * .... Parsed/extracted data is written to the respective tables
- files and the DB are closed.

I have tried to make the above accurate, but read the code.

TweetTableWriter and ConcordTableWriter are helper classes doing the actual table writing

Concord Words

The simplest way to get a word list is simply to break up all words on whitespace. Here things are a bit more complicated. Some of this is to "normalize" the words. A simple example is to make all the words lower case. But there is a fuller story:

- Lower case everything.
- There are some odd non ascii characters in the tweets that are first converted to more normal characters or white space.
- Classify words (english words do not start with numerals, but "tweet words" do. So we classify the words. We also have words starting with @, # for hashtags, htttp for urls)
- Linguistis have a way of normalizing words to what they call lemmas, I do this using a library calld spacy. Amoung other things it converts plurals to singulars.
- Note that many words in the concordance are in some sense not words at all, (even excluding hashtags....). When we try to match them up to a table of word usage (words) these match ups fail.

Words

This is pretty much a straight import of the words usage csv file. One addition is that the count column in that data is used to generate a rank column where the highest count is converted to the lowest rank. #1 rank ("the") is the most used word. It appears that a lot of the words are not words in the dictionary sense, common misspelling are common. But that is how the language is used There are over 300K words in this table.