Matthew Owens

CS410 Final Project Documentation

12/16/2017

GitHub repository link: <https://github.com/MattOwens/twitter_recommender>

**Introduction**

This project is a recommendation system for Twitter that takes a list of usernames and hashtags (called the seed) as input, and uses the recent tweets from those users and tweets containing those hashtags, along with new tweets received during execution, to recommend users and hashtags to follow that will show similar content. The recommendations refresh at a configurable period and feed back into the tweet-collection library to bring in more tweets to base recommendations on. The program can pass data directly to other components or through Kafka as part of a distributed system. Data is retrieved from Twitter through the tweepy wrapper of the Twitter API.

**Recommendation Algorithm**

The recommendation algorithm takes tweets and separates them into groups based on the user that posted them and any hashtags used. Tweets are duplicated in both the user’s and the hashtag’s bucket. This is to create a stronger association between a user and the hashtags she uses. For each tweet collected, the text is tokenized, stemmed, lowercased, stopwords are removed, and the tokens are split into unigrams. This project would benefit from using TF-IDF weighting as well, but it is currently not included for reasons I will explain later in the documentation. These groups of tweets are then divided based on whether they came from the seed or if they are new tweets. Each group of new tweets is compared to every group of tweets from the seed and given a score. The total score for a group of tweets (corresponding to a user or hashtag) is the sum of the scores that result from comparing it to each of the seed groups. Then the groups of tweets are ranked based on these scores at the top k are output as recommendations.

Groups of tweets are scored based on the text-similarity of the tweets and the amount of time that has passed since the tweet was posted. To do this, a score is generated for each pair of tweets in the cartesian product of the groups of tweets. Text similarity is measured based on the cosine similarity of the vectors representing the unigrams from the text of the tweet. This value is then multiplied by a time multiplier to get the score for the pair. The time multiplier is equal to one over the product of the natural logs of the time since each tweet was posted. This is done to penalize older tweets because they may be less relevant. The score for the two groups together is the sum of these tweet scores divided by the natural log of the number of pairs of tweets compared. This is done to help prevent groups with large numbers of tweets from dominating the recommendations. I considered using a true normalization to eliminate the effect of the number of tweets, but there is some value in recognizing users and hashtags that produce larger numbers of relevant tweets, that is these groups are rewarded for bringing large numbers of tweets into the recommendations in the first place.

**Code Overview**

The following is a list of the packages and modules included in this project along with a short description of what is contained in each.

* db – this package is used to store received tweets in MongoDb
  + tweet\_recorder.py – listens to tweets coming on a Kafka subject and records them to the database
* recommender – this package is where the actual scoring and recommendation takes place
  + analyzer.py – the module that actually scores tweets
  + direct\_result\_sender.py – a module used for passing results of recommendations directly back to the module that gets data from Twitter
  + kafka\_result\_sender.py – a module used for sending results on a Kafka stream
  + kafka\_tweet\_receiver.py – a module used to take input tweets from a Kafka stream
  + new\_tweet\_loader.py – a common module used to collect input tweets, whether they are from Kafka or directly passed from the module that got them from Twitter
  + recommender\_controller.py – a module that sets up the analyzer and controls how snapshots are taken to create a new set of recommendations
* result\_handlers – a package for storing modules used to listen to recommendations
  + kafka\_result\_listener.py – a module that listens to results on a Kafka stream and passes them back to the module that gets Twitter data
* twitterdata – a package used to get data from Twitter
  + batch\_loader.py – a module that gets a user or hashtag’s most recent tweets
  + direct\_tweet\_sender.py – a module that passes incoming tweet data on to the recommender directly
  + kafka\_tweet\_sender.py – a module that sends incoming tweets on a Kafka stream
  + keys.py – a module for storing access tokens for the Twitter API. This file needs to be modified as described in the Usage section for the program to run
  + stream\_loader.py – a module that listens for incoming tweets for subscribed users/hashtags
  + test\_tweet\_consumer.py – a module I used in early testing to make sure Kafka was working correctly.
* run\_config.py – a module used to represent the configuration options including what users/hashtags are used as a seed
* twitter\_recommender.py – the main module that starts the project.

**Usage**

The easiest way to run the project is to grab the code from github and use virtualenv and pip to install dependencies. Commands to run are in bold (assuming this is a linux environment. In windows the equivalent commands must be run to get the code as a zip file and extract it).

* Make sure you have virtualenv installed: **pip install virtualenv**
* Get the code from Github:
  + **wget https://github.com/mattowens/twitter\_recommender/archive/master.tar.gz**
  + **tar xpvf master.tar.gz**
  + **cd twitter\_recommender-master**
  + **virtualenv twitter\_recommender**
  + **source twitter\_recommender/bin/activate**
  + **pip install -r dependencies.txt**
* Follow the instructions at <https://www.slickremix.com/docs/how-to-get-api-keys-and-tokens-for-twitter/> to set up access keys for the Twitter API (For the purposes of grading the project, I’ve included my keys at the end of this file so the instructors don’t need to go through this step).
* Once keys are generated, copy those values into the appropriate variables in twitterdata/keys.py
* The easiest way to start the program for testing is to run the program with only direct data transfer:
  + **python twitter\_recommender.py local\_config.ini**
* Running with kafka\_config.ini will use Kafka to pass data between parts of the system and store tweets in MongoDB, but it requires Kafka and MongoDB to be set up locally, which is pretty difficult and more environment dependent than just setting up python dependencies. You can modify one of these files or create your own config file as I describe in the next section.
* You will see logging messages that tell you the program is running. After the initial data is collected and the time has passed to do a set of recommendations, the top k (as defined in the configuration) results will be printed to the console. If Kafka is in use, the results are also send on a stream so other applications could use them.

**Configuration**

The program takes a configuration file as a command line argument. Several sections of the configuration are required. Here are the configuration options and their possible values:

* SEED (Required)
  + SeedUsers (Required) – comma separated list of usernames to include in the seed
  + SeedHashtags (Required) – comma separated list of hashtags to include in the seed, must include ‘#’ because tweets are retrieved differently
* CONTROLLER (Required)
  + RefreshPeriod (Required) – how frequently recommendations will be recomputed, in seconds
  + FeedBackNum (Required) – the number of top results to send back to the module retrieving data from twitter as pseudo-relevance feedback
* DELIVERY (Required)
  + Sender (Optional) – Either Kafka or Direct, defaults to Direct
  + Recorder (Optional) – Mongo is the only option, no recording if it isn’t specified
  + Results (Optional) – Either Kafka or Direct, defaults to Direct

**Known Issues**

* Users/hashtags are scored only on similarity of tweets. I did not have time to include any topic mining or sentiment analysis because it took much more time to get the tweet data than I thought it would.
* With the feedback set up the way it is, the system hits Twitter’s API limits within 3-5 sets of recommendations depending on the number of users mentioned. The program could handle hitting the limits better, but waiting would still be necessary because of the incredibly low limits on the free tier of the Twitter API.
* I do not have TF-IDF weighting in the recommendation algorithm because I was unable to create a document corpus and score it the way I wanted to, including the discount for timestamp, using metapy. I attempted it, but was unable to do so and I thought it was better to show the effect of the time and implementing my own algorithm than it would be to use a built-in algorithm that didn’t do what I wanted it to.

Key.py content:

**consumer\_key = '3IzCTyLv7ib1L3D5fsPr7bGiv'**

**consumer\_secret = 'wmTdoOfiIuRhY4LGy11ncgOPpCALd6s06CurZS3T4LahnqCQ2q'**

**access\_token = '2437437204-DTQsqgXQmFxDAtXJR9giKa03li8MKmTm5CoZ6VJ'**

**access\_secret = 'c7T3Up3BluwjcQBn2iCGFw7f2McHvkjhQYaW8gmxbg2El'**