README for Clayton Stanley's dissertation

Clayton Stanley

Rice University

Description

Directory Structure. The repository is separated into three main folders: dissProject, firstSOProject, and writeups. The firstSOProject folder contains the support code for the Stanley and Byrne (2013) research. The dissProject folder contains support code for this dissertation research. The writeups folder contains all writeup projects created during the dissertation process (e.g., the LaTeX for this dissertation, Keynote presentations).

Within the dissProject folder, there are two main files that contain the majority of the code for the dissertation research: scrapeUsers.py and preProcess.R. The python file contains the necessary code to scrape the Twitter site for tweets, import results into the Postgres database, and perform various cleanup functions that were easiest to do in Python. The preProcess.R file contains all modeling and analysis code for the project. Within the file, there are implementations for the Bayesian and random permutation models. Also there is code to test the models by running each model through the various StackOverflow and Twitter test datasets.

Model Implementations. The Bayesian model is implemented using the data.table package, so that fast joins can be performed on the table when computing the strength of association values. The random permutation model is represented as a matrix, so that correlations between the model and context vectors are fast. The matrix is not sparse, since there are currently no sparse matrix packages for R that have a vectorized function to compute correlations between a vector and a matrix.

The Bayesian model requires slightly more time to compute activation values and also more memory space to store the representation. This is because the random permutation model is compressed and only needs to store around 2048 rows in the co-occurrence matrix. However, both models are implemented efficiently enough so that all analysis runs and activation computations can be kept in memory. 64 GB of memory was required to perform the analysis, particularly on the largest Twitter (3 million) and

StackOverflow (1 million) corpus sizes.

Design Decisions. All model manipulations (e.g., using an entropy weighting metric, using a model with only the prior term) were expressed as part of a configuration object. That object is a list of configuration name, value pairs. The configuration object is passed to the analysis code, which determines the specific type of run that is performed.

This configuration design enables a large amount of code reuse and overlap. This helps minimize the likelihood of a programming mistake, since functions are reused multiple times and overall static code length is kept to a minimum.

Testing. Each analysis run produced a set of result files. Those result files were checked into the repository so that changes could be tracked. Code changes were tested by looking at differences between previously checked in output files and the output files produced after a code change was made.

A small amount of unit testing was also performed. The core behavior of each model implementation (Bayesian and random permutation) was tested in isolation to ensure that the activation values returned by the models were mathematically correct. Unit test code is located within dissProject/test.

Parallelization. None of the analysis code is finely parallelized. Most of the inner loop in R ends up running in compiled C, and execution within those function calls occurs on a single core. However, since multiple models were explored, each different model configuration was executed in parallel. This type of course-grained parallelization was achieved using R's built-in Parralel package. Also some parallelization was done at the OS level when various make targets would run in parallel.

Database. All raw tweet and post data were stored in Postgres tables. Also, after the documents were tokenized, the tokenized versions of the raw data were stored in Postgres tables. When R ran an analysis, it would query the tables to retrieve the correct set of tokenized words and tags. So there is a lot of R code that uses the RPostgreSQL package to communicate with Postgres as the analysis is running.

Running the Analysis. All analysis runs were scripted and can be run by using the appropriate target in dissProject/Makefile. The three main targets to run the models through the prior-only runs on the popular users dataset, the full models on the randomly-sampled and popular-hashtags datasets, and the full models on the popular users dataset are "parrunPriors", "parrunContext", and "parrunPUsers" respectively. Each of these targets takes tens of hours to complete. "parrunContext" is the longest and takes around 72 hours on a desktop machine with 64 GB of memory and 24 virtual cores.

Smaller test runs were also created so that model changes could be more rapidly developed. These typically use smaller Twitter and StackOverflow corpus sizes (e.g., only 10,000 StackOverflow posts) to create the co-occurrence matrix and a smaller number of posts to test the models on (e.g., 20 instead of 500). Running these functions takes around a minute to complete. To run these, the R function is called directly (i.e., not a make target) within an already-running R session. As an example, the "runContext20g1s6" function takes only a few minutes to complete, runs the models on a corpus size of 1000, with only a single run per model, and only 20 sampled posts for each run.

Configuring environment

Git Repositories. There are two main git repositories for this work. The main repository contains all of the source code but very little source data. This repo is located on chil under "srv/git/stanley-dissertation.git". The source data was separated from the source code for two reasons: First, the size of the source data is much larger than the code, so if someone wants to look at just the source, it takes much less time to clone the source repo with this method. Second, the source data for Twitter should not be made public, due to Twitter's terms of service, and to encourage replication and transparency of research, the source code is publicized and available on GitHub. So the source data cannot be included in this public repository.

The source repo is included as a submodule of the main repository, but in order to

access it, you have to have credentials on the chil server. This way access to the repository can be limited, but the repo is still properly linked to the main repository.

Setting Up Environment. First, clone the main repository:

- *git clone ssh://[chil uname]@chil.rice.edu/srv/git/stanley-dissertation*: Clone the main repository.

Then, cd to the top level of the repository and run the following commands to configure the environment:

- *./check-dev-env.sh*: Ensures that ssh keys, hostname alias, binary dependencies, etc. are all configured properly. If this isn't run first, none of the following commands will work.
- *git submodule update –init –recursive*: Initialize and update all submodules. This will pull in the source data submodule.
- *make -C submodules/dissertationData pull*: Get additional compressed source data for the source data repo. This includes large .csv text data extracted from the postgres tables. These .csv files are too large to be tracked with git.
- *make -C submodules/dissertationData extract*: Uncompress the compressed source data.

If you want this done all in one go, run the *make post-clone* command at the top level of the repository: You'll need a version of make installed to do this. *check-dev-env* checks for this binary dependency and includes a suggestion on how to install make if you don't have it.

Manual Environment Setup. Setting up the postgres tables is currently not automated. This involves creating the tables, all indexes, importing the .csv tables from the source data repository, and then updating the tables with a few additional columns. This process *was* documented in the form of a script, but that script hasn't been tested. It contains code for both creating the tables and updating columns in the tables. Those steps must be done separately, and the tables should be imported into postgres before the updates are called. Currently that script is located in "dissProject/createTables.sql". I figure that no one other than myself will want to get the database configured, since the results of all the model runs are saved in separate .csv files and the db is not required to

analyze or visualize them. If I ever have to confiure the database on another machine for this research, I'll separate the "createTables.sql" file into two separate files, automate and test the whole process, and create an additional make target to configure the db.

References

Stanley, C. & Byrne, M. D. (2013). Predicting tags for StackOverflow posts. In *The 12th international conference on cognitive modeling* (pp. 414–419). Ottawa: Carleton University. Retrieved from pdfs/StanleyByrne2013StackOverflow.pdf