Sharing Code Among Academic Researchers: Lessons Learned

[Extended Abstract]

Carol Schmitz  
Illinois Institute of Technology  
Chicago, IL  
cschmitz@hawk.iit.edu

Ameena Khan  
Illinois Institute of Technology  
Chicago, IL  
akhan57@hawk.iit.edu

Libby Hemphill  
Illinois Institute of Technology  
Chicago, IL  
libby.hemphill@iit.edu

ABSTRACT

Academic researchers have been collecting data and programming scripts to process and analyze them for years. Researchers have studied the difficulty in sharing data alone, but sharing the scripts required to reproduce results has been discussed less often. At the Collective Action and Social Media (CASM) Lab at the Illinois Institute of Technology, we study how people use social media to engage with their communities. Our interdisciplinary team consists of students with various technical backgrounds. Since everyone in the lab needs to run code, we have developed a standard repository structure. We will share the structure definition and explain the reasoning behind our design decisions. We aim to make our data and code accessible to social scientists not trained in information retrieval, so we frame this paper from that perspective. By publicizing our approach we invite researchers with similar goals to build on our work, collaborate on the design and implementation of modern tools to share code and data, and to suggest improvements to our process.

CCS Concepts

- Software and its engineering → Reusability;

Keywords

open source, code sharing, replication, data reuse

1. INTRODUCTION

The challenges encountered when sharing data sets have been explored by researchers for many years and are well documented (see, e.g., [3] for an overview). One common problem data sharing efforts face is a disparity in its costs and benefits—a problem common in groupware [7]. As Birkholtz and Bietz [2] found, for instance, the burden of formatting and releasing data sets for sharing falls on the original researcher, but the extra effort required to format data in a way that promotes sharing does not benefit the original researcher. The benefits are gained by the researcher who is reusing the data, decreasing the likelihood of adequate documentation being produced by the original researcher. They also indicate that data alone is insufficient for reuse because metadata and other factors provide necessary context to the data set. In this paper we expand that claim to argue that scripts and other code associated with data sets are equally important in reuse scenarios. We also provide an overview of our lab’s practices for addressing these challenges.

While sharing data has long been a pain point for researchers who value collaboration, sharing code is not discussed as often. Neches et al. [9] wrote about the necessity of designing technology to share data and complex systems 25 years ago. While some tools have been introduced to the landscape in the years since, researchers today struggle with many of the same problems when attempting to share research. They propose an artificial intelligence approach to automating knowledge sharing and argue against "start[ing] from scratch each time" [9].

In addition to the technical and organizational challenges encountered when sharing data and code, there is also the problem of identifying the audience for the shared tools and information. Ensuring a simple process to setup shared code environments is even more important today, since the scenarios where non-technical researchers may wish to use data will become more and more frequent. In social media studies in particular, computing skills are not necessary to ask and answer relevant research questions, but the nature of the subject requires manipulating data with code. We also agree with King [8] that "unless we are content to let data sharing work only within disciplinary silos, which of course makes little sense in an era where social science research is more interdisciplinary than ever, we need to develop solutions that operate, or at least interoperate, across scholarly fields." However, while some advances have been made in standardizing the process of sharing data, as Bechofer et al. [1] explain, data alone is not enough to validate prior research.

How are we to ensure that data sharing, especially social media data, can work across disciplines and at many levels of coding expertise?

In the CASM Lab, our approach is three-fold. First, we release our code as open source repositories via GitHub so that the code, and our conversations about it, remain visible. Second, we provide easy to navigate user interfaces through Jupyter notebooks, so the code remains available to those
who are curious and usable for those who are more interested in the results of running the code than in the code itself. Third, we use Zenodo to archive and identify code used in individual papers so that our process for each publication is transparent and reproducible. We describe these steps here in an effort to capture and continue the social media research communities’ work to share data and methods.

2. EXISTING EFFORTS AT SHARING WORKFLOWS

Though we avoid a complete overview of all efforts at sharing code for research in the interest of brevity for this poster, we do want recognized one effort: Experiment. De Roure et al. [5] attempted to solve the problem of sharing at scale by developing a tool called Experiment. This tool was designed to share entire scientific workflows, including data, code, environments and other elements of the scientific process. One of the key elements of this tool is the ability of new users to add features to existing workflows. Experiment was used frequently (by 750 registered users from 78 countries [5]), confirming a true need for this type of solution in the scientific community. It is built on Ruby on Rails, which also has the advantage of being a very human-readable language. However, the open source Taverna workflow management software on which it is built supports R but not Python, so the scripts are not as accessible to those who can’t read code.

3. THE CASM LAB APPROACH TO SHARING CODE

Publishing open-source code is a mission of CASM Lab. The first step in sharing code is organizing it in a way that will make sense to others. Consistency helps outside developers know what to expect when they come to our code. By taking the time to both develop and document our internal standards, we can continue to maintain code despite turnover in the lab.

We have developed a standard repository structure and documented the reasoning behind that structure. By adopting this structure for all our public repositories, we make it easier for newcomers to understand and use the code for themselves. Once they become familiar with one repository, they can more easily learn how to use another repository published by our lab because of the similar structure. Being transparent about our process is also better for science, since researchers can hold us accountable or alert us to mistakes. Researchers can also build on our work more easily.

Nearly all of our public code is written in Python, and all of our public repositories use Jupyter notebooks to document and run the code.

Python is a programming language that is especially conducive to sharing code since the syntax is easily human-readable for English speakers. Additionally, many packages or libraries have been written by Python developers to add standard data analysis capabilities to the built-in Python packages (e.g., pandas, numpy). This allows researchers to pick up where previous research ended and focus on new problems and solutions, rather than coding new solutions to problems that have already been solved.

Jupyter notebooks make it possible for people who do not code to use our programs, and it is popular among data scientists already. Using Jupyter allows us to store the documentation of the code, including the reasoning behind various algorithms, and the code in a single location, and it is becoming standard practice in other disciplines [10]. The approach also creates “containers” of code and data that are easily shared.

3.1 Workflow

We try to follow a standard approach to social media data in all our projects (no matter where we are getting data). It has 4 basic steps:

1. Collect
2. Cache
3. Parse
4. Analyze

This workflow allows us to collect data once for use in multiple projects. It also allows us flexibility in analysis and redundancy in data storage.

Under the collect step live scripts for getting data in “raw” form. Here, raw means whatever default format for the data is. Usually this means JSON dumped by an API, but for scrapers it’s whatever data structure and format we decided to use. We are greedy in collection meaning we pull whatever data the API will let us have. For instance, in the Twitter projects, it means data returned by the ever-changing Twitter API.

Once we have raw data, we cache it by storing a read-only copy somewhere accessible to the whole team, often a local server. Usually this storage step is handled by the collection script and not by a standalone script. We treat this as a separate step because it’s conceptually important - social media data changes all the time, and caching lets us keep track of what the data looked like at the time of collection (e.g., what was returned, what structure was standard then).

Next, we parse. Parsing scripts pull data from the read-only caches and put them in formats that are appropriate for analysis or whatever comes next. For instance, some of our Twitter tools collect data from the search API, cache it, then parse it into a human-readable CSV for import to Excel or another stats program. This leaves us with two related, but not identical, copies of the data - one in JSON from Twitter, and one in CSV. Parsing scripts also do any data transformations that are necessary for analysis (e.g., converting timestamps, calculating user stats).

Finally, we get to analyze the data. Often analysis is included in the same script as parsing, but sometimes analysis steps will live on their own. Some of the analysis will involve machine learning (e.g., automated classification) or natural language processing (e.g., topic modeling), but some will be simple word clouds or descriptive statistics.

3.2 Repository Structure

At CASM Lab, all of our GitHub repositories for social media data collection and analysis have a similar structure. We use a single Jupyter notebook to explain and run the code within the repository. This notebook calls scripts within the /scripts folder, and stores any files generated in

1https://github.com/casmlab
4. OPEN CHALLENGES IN SHARING CODE

Designing and developing data analysis scripts while prioritizing potential reuse introduces some complexity to the development process. We have encountered compatibility issues between operating systems and various Python libraries. After encountering roadblocks as a result of conflicting dependencies, we started using separate environments for each project. The following sections explain our experiences.

4.1 Technical Challenges

At CASM Lab, we primarily code on Mac OS X. Our servers run Linux though, and it’s sometimes difficult to run code on the servers that was developed on our local machines.

One of the problems we’ve encountered running code developed on OS X on a Linux server is that not all Python package developers provide Linux versions of packages for OS X. Or they might provide a Linux version for certain Linux distributions, but not the particular distribution we are using. Either way, it results in a roadblock that cannot be resolved without modifying the code.

If you’re using the same operating system as your collaborators, one way to share code is by setting up environments. Environments allow you to easily switch between multiple projects without introducing conflicting package dependencies. By replicating environments using package versions, we avoid situations where one version of a package may conflict with other system configurations when another version of the same package has no conflicts.

Separate environments can be created for each Python project using Anaconda³, an open sourced Python distribution. Users can create a new environment at the start of a project, then activate the environment when they are working on that project. At CASM Lab we provide an environment.yml file in our repositories. This file allows users of our code to replicate our development environment on their local machines with a simple install command, rather than having to download and install the correct version of each package manually. There are limitations to this environment file, as it primarily tracks Python packages used in the code. Various operating system packages necessary to run the code are not always included in environment.yml, so this doesn’t eliminate all dependency issues between projects, but it greatly reduces them. Anaconda does have some limitations. For instance, it includes only distributions of Python and R, and it is dependent on collaborators using compatible operating systems.

4.2 Sharing at Scale

We know that we are not the only research lab to take steps toward solving this problem. For instance, there is a desire for a central repository where academic researchers can share data sets and scripts associated with those data sets⁴, but such a platform does not yet exist. Building a consortium of researchers who prioritize open access to data and code can begin the process of developing standards around such sharing, and these standards can eventually be developed into a sharing platform. Ideally, such a platform will

³https://www.continuum.io/downloads
⁴https://github.com/ocdx

the /files folder. The notebook also contains a section to help users create a settings (i.e., configuration) file, where the user may modify a predefined set of configurable options used by the code, such as API credentials and file names. Our standard repository structure contains several elements.

/data_samples We store sample files produced by the code after each stage of the collect, cache, analyze, parse workflow, including raw data from the source, here.
/files This directory contains any files needed to run or generated by the code. Some of our scripts download or generate extremely large files (e.g., GIS-files from the U.S. Census), so we don’t store these files in the cloud. If you run our code locally, this is where our scripts will store such files on your computer.
/scripts This directory contains the meat and potatoes of our code. A single repository may contain several scripts which can be run independently of each other as users work through the collect, cache, analyze, parse workflow.

[repo_name].ipynb A single Jupyter notebook explains how the various pieces of the code in each repository work. At a minimum, it contains the following sections: Setup, Collect, Cache, Parse, and Analyze. The notebook contains both Markdown blocks, which explain the goals of each workflow phase, and Code blocks, which run the code contained in the repository. Code blocks call scripts stored in the /scripts directory and save files in the /files directory.

settings-example.cfg This file contains a sample version of all configurable options used by the code, except for API keys and other private credentials.
/environment.yml This file can be used by a package manager, such as Anaconda, to automatically install the packages and libraries needed to run the code in this repository. Operating system requirements may apply.

requirements.txt - A pip-compatible version of the necessary packages list is included for those who use Python but don’t use Anaconda.

README.md - This file uses markdown syntax and explains what users will find in the repository. It includes operating system requirements as well as contact information for the lab director in case of any questions.

This method of organizing code has served our lab well, but keeping internal repositories organized before publishing to the open-source community is just the first step in sharing code. Since the utility in sharing code actually comes from running it, we turn now to some of the difficulties in releasing code after each stage of the collect, cache, analyze, parse workflow, including raw data from the source, here.

4.3 Archiving Repositories

Because we’re always working to improve our code, it changes frequently. In order to capture relationships between certain states of the code and the publications we produce, we use Zenodo² to archive our repositories and assign them digital object identifiers (DOIs).

To do so, we use GitHub’s release feature and name the release after the publication associated with the code in that state. Then Zenodo maps that particular release, the code that accompanies it, and generates a DOI for the archive. We can then cite the code easily and refer users to complete archives to use in replication efforts.

²https://zenodo.org/
make it easy for those with varying levels of technical expertise to access and actually use the data and code provided.

In order for programmers to effectively use shared code, it is critical that the local system used for reuse is the same as or at least compatible with the system on which the code was originally developed and tested. This may appear to be a simple task for a user who can install the needed packages manually. However, with the passing of time between the creation and reuse of code, necessary packages may be updated where the latest version breaks the code. Even if a user was provided information on which package versions were needed, upgrading or downgrading packages in the same space within a system could present conflicts with packages needed for the user’s other projects.

Sharing at scale introduces additional complexity to the general problem of sharing code. Researchers have personal preferences for operating system and programming language decisions, and if those decisions are not initially made with sharing in mind, many difficult problems may arise later in the development process which may hinder the ability to share widely and seamlessly.

For instance, when initially developing a solution to a problem, a researcher working on OS X may choose to use a familiar Python package. Maybe two packages available with similar functionality are available, but the researcher has used Package A before and is comfortable with it. If the researcher is only concerned with her own work, she may select Package A and move on. Later, when the researcher wants to share her code on a Linux server, she may discover that Package A isn’t available for Linux, but both Linux and OS X versions exist for Package B. Now the researcher must either choose not to share the code with Linux users, or to go back and replace all references in the code to Package A to the corresponding Package B reference. The code must also be tested to ensure that Package B functionality is similar enough to Package A that the code output does not change. Updating the code can be time consuming, but can be necessary in code maintenance.

Making code sharing a priority at the beginning of a new project, by slowing down and developing consistent standards to organize the code, is one step that helps make the code more easily reusable. CASM Lab develops code using this approach. We also develop new code on widely used operating systems, in popular languages, with tools other researchers already use. However, we are just one lab and not everyone shares our commitment to open source research. As De Roure et al. states, "Enabling incentive models for sharing within a community of practice and supporting an emergent model of sharing is a challenge."

4.3 Replication vs. Reuse

If the goal in sharing code is to encourage others to replicate your own results, providing code that can easily be executed by anyone is important. Everett and Earp [6] offer a number of reasons replication studies are uncommon, including that they are difficult to publish, are not awarded the same recognition are original work, and are time-consuming to conduct. By providing the code in a format that is easily reusable, researchers can more quickly replicate the study and may, as a result, be more likely to do so.

If the goal is to answer new questions that build on original research, one could argue that a certain level of programming skill is required to write scripts that can answer those new questions, so it is less critical that the original code be accessible to those who don’t code. We disagree. The technical barriers to social media data have created new hierarchies that create economic and gendered divisions between those who have insight to offer and those who have the money and skills to access data [4].

5. CONCLUSIONS

We described our data collection and analysis workflow, repository structure, and repository archiving practices in order to contribute to the broader discussion on sharing data and code for social media research. Part of our mission in the CASM Lab is to make social media data accessible to a broader research community, and that goal motivates our efforts to share both data and code.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant no. IIS-1525662.

7. REFERENCES

[1] BECHHOFER, S., BUCHAN, I., DE ROURE, D., MISSIER, P., AINSWORTH, J., BHAGAT, J., COUCH, P., CRUICKSHANK, D., DELDERFIELD, M., DUNLOP, I., GAMBLE, M., MICHAELIDES, D., OWEN, S., NEWMAN, D., SUFFI, S., AND GOBLE, C. Why linked data is not enough for scientists. Future Generation Computer Systems 29, 2 (Feb. 2013), 599–611.
[2] BRHNHOLTZ, J. P., AND BIETZ, M. J. Data at Work: Supporting Sharing in Science and Engineering. In Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work (New York, NY, USA, 2003), GROUP ’03, ACM, pp. 339–348.
[3] BORGMAN, C. L. Big data, little data, no data: scholarship in the networked world. The MIT Press, Cambridge, Massachusetts, 2015.
[4] BOYD, D., AND CRAWFORD, K. Critical Questions for Big Data. Information, Communication & Society 15, 5 (June 2012), 662–679. 00344.
[5] DE ROURE, D., GOBLE, C., AND STEVENS, R. The design and realisation of the Virtual Research Environment for social sharing of workflows. Future Generation Computer Systems 25, 5 (May 2009), 561–567.
[6] EVERETT, J. A. C., AND EARP, B. D. A tragedy of the (academic) commons: interpreting the replication crisis in psychology as a social dilemma for early-career researchers. Personality and Social Psychology (2015), 1152.
[7] GRUDIN, J. Groupware and social dynamics: eight challenges for developers. Communications of the ACM 37, 1 (Jan. 1994), 92–105.
[8] KING, G. Ensuring the Data-Rich Future of the Social Sciences. Science 331, 6018 (Feb. 2011), 719–721.
[9] NECHES, R., FIKES, R. E., FINN, T., GRUBER, T., PATIL, R., SENATOR, T., AND SWARTOUT, W. R. Enabling Technology for Knowledge Sharing. AI Magazine 12, 3 (Sept. 1991), 36.
[10] SHEN, H. Interactive notebooks: Sharing the code. Nature 515, 7525 (Nov. 2014), 151–152.