Data Collection Documentation (Draft)

Steven Deutekom

June 2019

Contents

1 Introduction

1.1 Purpose

This is a document describing the tools and process that we used for collecting source code from the internet for sociolinguistic research. The building of a dataset was to support this research to analyze how sociolinguistic features affect a programmers use of a programming languages. We collected source code samples in different ways from websites like GitHub and codeforces. These samples were collected for several languages and saved with information about the author that wrote them. The goal was to get a good representation of source code written by authors of different gender, experience and region. The data will be run through machine learning algorithms to determine what kinds of differences exist in the use of programming languages by these different groups of programmers.

1.2 Contents

The document first focuses on the use of GitHub to collect source code. Then it moves on to Codeforces. In each section there is a discussion of the data that supported finding and collecting source code for specific authors. Then there is a discussion of the process used to collect this data. Finally we discuss some of the issues that were encountered collecting the data, as well as some of the issues with the validity of the data.

2 GitHub

GitHub is the largest online code sharing/hosting website on the internet (proof? and perhaps an estimated size). This means that it can be a very good source of code samples. All code is associated with an author and changes to that code are tracked as well. This make it possible to find code that has a high likelihood of being from a single author. Because of GitHub's size and use by the developer community, there is code from many different demographics of programmers available.

Because it is a mature and large website it has well developed tools for accessing this data, and even public datasets that store data on what is available on the site. This makes it easy to work with in general. It has an api that can be used to get any information that is desired. It is easy to use, though it has a rate limit that can slow down collection of data depending on what is being collected or how many api calls might be needed for each piece of data.

These reasons led us to choose it as a site to collect source code from. We were able to do this in a couple different ways. Using projects and git blame to find single author files. And using commits and newly added commit code to find single author samples as well. We were also able to take advantage of the Ghtorrent database and google BigQuery to find and narrow down the projects and commits that we wanted to try and process.

2.1 GitHub Metadata

There is lots of data on GitHub and to narrow it down or find projects and commits to get source code from we needed information on these projects. The GitHub api offers lists of these things, but the size of the data means that many api calls would have to be made and filtering out entries that are not useful would have to be done after collecting the data. For this we used Ghtorrent and google big query to filter out the metadata we wanted, such as country and making sure projects were not forked, etc. Google BigQuery enable this to be done quickly and their usage limits for free accounts are more than enough to get all the data we needed. These tools and how we used them is detailed in this section.

2.1.1 Ghtorrent

Ghtorrent is an 'offline mirror of data offered through the GitHub REST API' (from website). It contains large quantities of data on GitHub users, projects, commits from as far back as 2012. This data is being collected (by whom) to support research on software repositories and is available to the public for other research on Github.

We chose to use Ghtorrent because it allowed us to filter and collect specific sets of commits and projects without having to use the GitHub api. The GitHub api has a rate limit and in order to process the amount of data we were able to with Ghtorrent would have taken a very long time. Ghtorrent allowed us to filter out data that was for specific languages as well as make sure we were only looking for projects that had a country attached to them.

2.1.2 Google BigQuery

The Ghtorrent dataset is available for download, but it is very large (100 Gb for the MySQL). To make it possible to run queries on it without downloading it the dataset has been made available on Google BigQuery. This make accessing it much easier and faster, though not without it's issues. We were able to use SQL queries to collect only the information that we needed for our research. We were then able to download these results to use them to facilitate collecting source code for valid projects and commits.

2.1.3 Technical details

First one must open BigQuery with the Ghtorrent dataset (link?). Then one can select an option to query the dataset. Choosing the fields that are desired from each database in the dataset one can filter out only the results that are relevant to their search (some examples?). If this set of results is very large it is best to save it to a new BigQuery table. Then once it is saved if it is more data than is needed a random sample can be collected with sql again and saved to another table. Once the sample is saved it can be downloaded as json files. Unfortunately, BigQuery only allows 10k rows to be downloaded at a time so one must use sql to get each set of 10k rows (more examples) and download them individually. For a 100k-500k this does not take too long. Once the files are downloaded they can be processed and combined with a script to be in one or more larger files that are lists of records and can be loaded by pythons json module. (there is a script for this). These resulting data files can then be passed to the relevant script for processing to get source code and other necessary details.

2.1.4 The Data

Our research centers on how different sociological features of a programmer affect their use of programming languages. So far we have been exploring things like the gender and the region of the programmer. We were looking for commits and projects in languages like C++, Python, and Java. We needed these to have regional data associated with them and Ghtorrent provides country, state, and city for a user. All projects have a programming language attached to them as well so we were able to filter by language easily.

Using the dataset we were able to filter out millions of commits for each language. Using SQL we further took random samples of 500k rows for each language. We then were able to download them and use the GitHub api to get the content of the commits. We were also able to filter out 150k plus projects for each language that had not been forked. From these we took random samples of 150k projects and used some python libraries to download projects and collect source files from them.

2.1.5 Issues

Because the Ghtorrent database holds a lot of older data it is not all still available. Some of the projects are marked as deleted so we filtered these out. There are also fake users (which means what?) that we mad sure to filter out. Whe taking projects specifically we filtered out forked repositories to give ourselves a better chance of finding single author code. We also had as mentioned above to filter for users with countries and our desired programming languages. Even with our attempts to filter these things. Some projects have been made private since their data was cached so we could no longer access them. Also even though a project is in a specific language it does not mean that it won't have other types of files in it. So it is necessary when collecting source code to filter for only files that are in the desired language. This can easily be done by checking the files extension. While using Ghtorrent makes it much easier to get metadata it does not offer everything that one might want.

There were a few specific things were were not able to get. While user logins can be used to connect commits and projects to an author, there is no name information available for users on Ghtorrent. Since we are doing analysis on gender it is necessary to get names in order to determine gender. If name information is available on a user, it is obtainable through the GitHub api. There is also no source code or anything of that sort in the Ghtorrent dataset.

It only contains things like user info and project names and urls, etc. It is a great resource, but for us it is only the first step in getting what we need.

2.2 Collecting Source From GitHub Projects

With this method we were trying to find projects and files that had only one author working on them. We collected data from Ghtorrent based on the project language and filtered out all forked projects and any that did not include country data. The goal was to download all the source code from these projects that fit our needs.

2.2.1 Process

Once we had the project metadata we set up a script to iterate through all of the entries. First we check the repository to make sure that it did not have more than one contributor. (some have 0 which we still took).We then checked if we could get the users real name with the GitHub api. If they had one we would try to find their gender. If they had a name and a gender we would download the repository and check each file in the desired programming language with git blame (mention the paper that also did this). If the file only had one author that contributed to it we downloaded this file and added it to the database. The entry was given the users information as well. If once a user was fully processed we had too much code for them files were selected at random and removed until they had a manageable amount.

2.2.2 Technical details

The script that runs first loads the desired BigQuery records file. For each record in the list loaded it first checks to see if a project is valid. This involves using the GitHub api to look for a real name. If there is a name then gender is searched for with the first part of the name (we assume this is the first name). Then if there is a gender for that name we again use the GitHub api to check the number of contributors to the repository. If there are 0-1 we say the project is valid. If any of these fails we move to the next project. If the gender data is unavailable at the time the project is added to another list of projects to be processed again later.

Once the project is determined to be valid it is cloned into a temporary folder. Then the script checks each file in the repository to determine if it is valid. This involves checking files and directories to make sure they are not in our excluded list. We also check to make sure they are in the right programming language. Then if they are we check the file with git blame and make sure that there has only been one author working on the file. Also, the author of the file must be the repository owner (we only have their information). If a file meets these criteria its path is added to a list that is returned to be processed.

Each file in the valid files list is opened and it's source code is collected. The number of lines is checked to make sure it is over 10. If it is ok we add the file and it's number of lines to the record and add the source, user, and project information to the database.

Once the script is finished, either by unexpected exit or reaching the end of records or a user defined limit of records to process, it returns some information. This information tells how long it ran and how many files were collected vs how many were processed. It also tells how many projects were processed total and what the index of the first project was. This enables the process to be started again at the place it left off.

2.2.3 Issues

The main issue with collecting code this way is getting library code that authors have copied from other sources and added to their project. If they are adding a file that is not in their git project it will appear as if they are the only author. To avoid this we filtered out certain folder names like lib, ext, library, extensions. (mention paper I got this idea from). This hopefully limits some of this kind of code, but it does not guarantee it will all be caught. Anyone could add any code to their repository and not follow convention for putting this code in certain folders. Also, projects could be collaborated on not using git and then added to GitHub after they are done. We would not be able to easily tell if these files are single author either.

A bonus was that the projects do not need the git api to download and process a project so there is less waiting. Since they take some time to process using the api for contributors and names does not use up the api each hour. Though adding gender while processing the projects means that if we don't have a name in our database and the gender api is down the project must be saved for later if desired. These lists of saved projects can then be processed again when the gender api is reset.

2.3 Collecting Source From GitHub Commits

With this method we were collecting code by using authors commits. It is a little easier to be sure that a commit is made by one author. So using Ghtorrent we collected a random sample of commits for a desired programming language. We then used this to find the source code that was added with each commit. In order to get code that was all together we took only commits where a file was newly added.

2.3.1 Process

The GitHub commit data was processed one commit at a time. For each commit we used the GitHub api to get the contents of the commit. Each file change in the commit was checked to see if the file was in the desired programming language. It was also screened to make sure that it had a minimum number of lines and was a newly added file instead of a modified file. If the file met all our criteria it was added along with project and user data to our database.

2.3.2 Technical Details

The script loads the file of commit records into a list. Each one of these records is processed by collecting the commit details with the GitHub api. The commit details contains all the changes that were made to files for the commit. So each of these sets of changes to a file is processed individually. We first check the status of the file, only taking ones that are set as added. The filename is then checked to make sure it has the desired extension. Then the number of changes is checked to make sure that it is over 10. If the file satisfies these checks then the source code is collected. It must be cleaned a little first because the commit text puts + and - in front of every line. Since we only take added files there is only a +. This + is removed along with the first line that contains some stats on the changes to the file. Then the remaining text is returned as the source. The source and the user, project and commit information is then added to the database. The process is repeated until the file is exhausted or the max number of commits is processed. Like the projects script there is information returned about the details of the processing.

2.3.3 Issues

There are some larger issues with this method. Though the commit ensures that the file was added by one author and not worked on again by another author, it still does not guarantee that the code is not library code or collaborative code.

Also, because many first commits on a file are just for setting up it's structure many files obtained this way do not have working code.

Because most commits are small the git api is used up in about 30min on average. This means that it was not possible to collect gender at the same time as the commits as to get the name would add an api call every commit. It might be possible to query users from our database to get names rather than looking for them with the api, but it would still slow things down. Instead we added gender to these after they were in the database. Then a separate script could find names and genders. Updating the database when it is large takes a while so the api was never used up this way and the other scripts could keep running.

It could be modified easily to take other information commits.

3 Codeforces

Codeforces is a competitive programming website. It allows users to participate in contests and attempt to solve computer programming problems. Users who solve problems during rated contests are given a rating and a ranking. Large portions of the submission code is freely available to be viewed on the website on the users individual page. There is a high probability that each submission was worked on by only one user and there are large numbers of submissions saved for most of the users.

Codeforces is one of the major online platforms for this kind of programming so they have many users and many problems to solve and contests to participate in. They also have an api that allows collection of some kinds of user and submission data that makes it possible to find users that have desired information so their code can be collected.

3.1 Codeforces User Data

Before being able to collect any source code from codeforces we first needed to find user information. The codeforces api has a way to collect a list of all users that have ever participated in a rated contest. At the time of collection this was around 180k users. Collecting this list of users and their information was the first step in the process. From this user data we were able to have access to the users unique handle, rating, rank, and when the users was registered. Also, there was some voluntary data like name, country, and organization. Because some information is voluntary we filtered out only the users that had all the fields we wanted. Since much of the research focuses on gender and region we could not use any user that did not have country or first name. Once we had filtered these users out we also tried to add gender to them based on their first name. If no gender data was available for their name we removed them from our list. This gave us a much smaller pool of users, but they all had the data we needed. (give a number?). Any information that we wanted to add like gender was added to the records at this point so that it would be available when using the records and adding information to the database for each submission. For each user codeforces also has an api call which can get a list of the users submission for problems. This data has things like programming language, author(s), problem name, time submitted, a contest id, and sometimes a difficulty. It does not contain the source code though. We were able to use this data to filter out submissions that had more than one author to make sure that it was a single author piece of code. This is the norm for most submissions. Once a submissions information is retrieved it can be used to get the source code.

3.2 Collecting Source From Submissions

3.2.1 Process

In order to collect the source code we had to do a number of things. First the list of users is iterated and for each user a portion of their latest submissions is collected. Then for each submission the scripts we ran would attempt to collect the source by scraping it from the source code page for the submission. If the code could be scraped it was collected together with some user information and submission information and added to the database. (should I put in any actual information about how the scripts accomplished this?)

3.2.2 Technical Details

The first thing to do is to collect the list of users from the codeforces api. This can be done with a single api call. the records are returned in json form and need to be put into a list so that python can load them. Once they are in a list it is possible to pre-process them. A script is run to add gender to records if it is available. This script collects all the users that have gender and any other required fields and writes them to a new data file. Users that have all fields except gender when the api is unavailable are added to another file that can be reprocessed for gender later. Once this list of users is complete it can be used by the next script to collect submission source code for the users.

The next script opens the user file and for each user an api call is made to get a list of the users most recent 50 submissions. Each submission is checked to see if it was an accepted solution, to make sure that it was not a team solution and that we have not already collected a solution to the same problem this pass. It is also checked to make sure that it is in one of the desired programming languages. If it satisfies these requirements it is processed.

The script uses two methods to try and collect the source code. The main method is using the selenium web driver. This opens a web browser while the script runs. For a user the web page is opened to the users submissions page. Then for each valid submission it attempts to open the popup that contains the source for the submission. Once open the page is scraped and the html that contains the source code is collected and the source pulled from it. Then the popup is closed and the next submission is processed.

The second method takes over if the first fails. It uses some information on the submission to construct a url that will lead to a page containing only the source. This can be collected with requests and does not require the web browser. However, it is often not available and that is why it is a last resort. Once the html of the page is received it is scraped and the source code pulled from the html.

If it was possible to collect the source from a submission it is added along with user and submission information to the database. The process is repeated for each of the 50 submissions that are valid. The reason for collecting only 50 is partly that we don't need that many for each user, and also because that is how many submissions are available on one page of the users submissions list. It simplifies things to not have to try to open new pages with selenium, but it could be done if desired.

Like the GitHub scripts at the end of the scripts run it displays the details of the collection. These can again be used to restart the script at a desired index and see what was collected.

3.2.3 Issues

The collection process was made a bit harder by the setup of the codeforces website. It stores all of the source code in popups. These cannot be scraped unless they are opened. So it was necessary to use a library that actually opened and interacted with the website in order to scrape the source. As a fallback it is sometimes possible to open a direct link to the source in it's own page, but these are often unavailable for some reason. So this method was only used when there was some issue with opening the popups.

Since much of the data we are focused on is given voluntarily there are many users that do not have it all filled in. This thins the available users for us to use. It also presents issues with why certain people don't provide that information. It is possible that certain groups are less likely to give their personal information and therefore we may not be getting an accurate picture of the demographics involved. It is also always possible that people are not being truthful with the information they give which could skew our results a bit.

As always with code it is not possible to always tell if one author has done their work alone. It is possible for people to work together and submit on one account. It is also possible to get other peoples solutions off of GitHub and other code sharing sites. Given the competitive nature of the site it does not benefit people too much to collaborate without everyone getting credit. However, it could also entice people to cheat by collaborating or finding solutions to problems online. We imagine that these issues should only affect a small percentage of the submissions we collect, but it must be mentioned.

4 Gender

To collect gender we use an api called genderize.io. It is a free service to find gender for first names. In addition to returning the gender it also returns a probability that the gender is correct. This probability seems to be based on the number of samples they have for a name and the male vs female. The calls to this api are done with a simple REST api call. It has a rate limit of 1000 per day. To deal with this we cached names that we checked in our database so that it would reduce the calls needed byt first checking locally. Since there are a lot of more common names this seems to work quite well. The downside is that lots of names entered are unusual or have extra characters and will not have gender information so they waste a lot of api calls.

For GitHub commits name and gender was added to the database after the commit data was collected. This way we could use one GitHub accounts authentication to collect names and add genders to one language, while collecting commits with another GitHub account in a different language. Github projects had there gender collected when the name was found because it did not run the GitHub api out faster to get names and gender during the script run. Codeforces it was possible to just run the gender script on the user results and collect it all at once. This is because codeforces users have a first name field if they enter it. Also, this way there is no rate limit other than the gender api to worry about.

The code to do this was all very simple. If a name is in the gender database we setup use it Otherwise check the api. If the api has a gender or says it has no gender info for that name we add the name to our database and return the gender to the script that called it. If the api is up then Null is returned for gender and probability. If this happens the scripts either ignore the entry for now or they save it to be processed again later.

Because the api has a probability it is possible when deciding what data to use to check this and filter records that are above a certain percentage. It would not be very good to try to predict gender when the gender of your tagged data was already only 50%. It is also interesting to note (add some figures later). That the probability is much more likely to be low for female names. Most male names seem to fall into the 100% category. However, for female names there can be an almost even number of names in each 10% category. It does still get higher the closer to 100%. This makes one wonder if the gender data is somehow biased towards male names, maybe it is easier to collect data on male names? Or perhaps it is more likely for female names to be closer to unisex names.