Contents

1 Introduction

I am not sure if there should be much here outside of the subsections, but perhaps a kind of thesis like paragraph and standard introduction could come here. Or everything could be confined to subsections.

1.1 Purpose

It would be informative here for me not only to describe the purpose of this document, but also the reasons for the project. I am trying to collect data for graduate students that are doing sociolinguistic research on source code. They have specific avenues of research and so there are specific things like gender, country, and experience that are important to include in our data. Perhaps some references to past work and the reason that we are trying to come up with a larger and more comprehensive database. things like more programming languages, more samples, and hopefully more professional work. Though some of this has it's challenges for certain.

1.2 Organization

Here is where I should detail the layout and organization of the paper. Perhaps give a very brief idea of what each section covers. I won't bother until I actually have an idea of what it will turn out as.

2 Data Collection

If we keep this section some of the intro and overview could go here before all the subsections. Keep in mind that this should not be about the specific methods as it might be about the data and the collection itself.??? Note somewhere that we most certainly did not always get all the data that was available to us, But we tried to make sure that what we did get was useable. So we may have filtered out some useable data without knowing it, but thats ok.

2.1 Overview of Process

The process for gathering data was similar in all counts. It involved finding websites that have freely available source code that we could use in our research. Then the goal was to find some sort of list of users or projects etc that we could download and filter for our needs. For the most part we filtered it to make sure that the records we used had things like gender and country as those are important aspects of our research. After gathering and filtering these records scripts were used to collect valid source code and programmer information for each one. The ones that we found enough information for were then put into our database.

2.2 Creating The Dataset

This might really be the same as the overview above. I should say what kind of data we were looking for. What kind of constraints we put on the data. In fact maybe the whole data collection thing really is what the whole document is about and it should be part of each subsection on the specific websites. Maybe there can just be a quick introduction of what we are doing in the introduction and then put some of this more specific detail into each section. Ie) github could have a breakdown of the process and what kinds of data we wanted to collect. Then we could list the issues with collecting data from github and what steps we took to deal with these issues. We did try 2 different ways of collecting data. So it would be valuable to talk about why we did it and what we hoped to gain by doing so. I do think it will go a long way toward being more informative though if we have a section in each specifically about the collection of data and not about the other parts of it.

2.3 Issues and Challenges

Think about what it is that is hard about collecting a dataset like this. Not only from a technical standpoint, but also from an availability standpoint. What makes good data hard to collect. What threatens the validity of the data we collect. Are people begin truthful? Are certain groups more likely to not fill in the required fields?

2.3.1 Accessing The Data

Api limits. Unreliable websites or no api. Hard to anticipate errors that stop the code from running.

2.3.2 Availability of Required Information

Lots of people are not giving all the information that they can. We are then unable to get all the information we want on every user. We question why people are not filling in all these fields and wonder if certain groups are underrepresented because they are not filling in fields rather than just not coding. Things like names are not required and they are needed for us to get gender since code platforms do not ask for gender. We also are doing research with region so if programmers are not putting their country in we cannot use their data. There could be many reasons why people choose to put this data in, from a feeling of privacy to perhaps social or legal pressure to not be coding?

2.3.3 Validity of Data

There are several issues here. First is that people may not be writing their own code. On codeforces this is not as big a problem because a person is usually solving the problems on their own. Especially if it is a contest and there is limited time to collaborate. And perhaps if they collaborate the solution might not be their own but the code probably is. However, we cannot guarantee in spite of efforts to eliminate team code that all submissions are single author code. For github a file may have been obtained from any source and then added to github. This could be most problematic with library code which virtually all programmers will use on certain projects. (as in an article) we tried to filter out directories that may be obviously library code, but cannot guarantee all programmers follow these structures. Also there may be times where a programmer gets code from another source and adds it to their project and modifies it very little. All code was vetted with git blame so anything that was created with git should have everything we need to call it single author code. But again we cannot guarantee this. Second there is nothing saying that people have to fill in the fields for their profile accurately. One could just as easily put in a fake name or a fake country and we would never know it. We hope that most people will fill in the fields accurately or not at all or at least with very obvious fake names. But if someone wants to call themselves bruce willis or gal gadot we will classify them as male or female with their names, but not necessarily know that they are fake names unless we inspect every name in the dataset. Thirdly, the resource we used to collect gender data with may not classify names accurately. We can and will make sure only to use names that have a higher probability of being the right gender, but we cannot be sure they are always accurate. This means that our machine learning algorithms will learn to classify gender code according to our api not reality. Though again this should be a relatively decent way of finding gender. Other companies (list some) use this service. Of course there is always the fact that a name is no guarantee of gender in the first place. It would be better to ask participants in a study to give their gender and use that, but this is much more difficult when trying to collect this much data.

2.3.4 Permission

All code that we have gathered is made publicly available. Github for instance has a user agreement that allows anyone to take publicly available code and use it. We can assume that if it is permissible to use and modify the code for our own work then it is permissible to use it in our research. As for codeforces info was not readily available, but again all code is freely available for anyone to view and we could easily have done parts of our research by viewing this code. We have just chosen to collect it automatically. We are using names etc. but we (should/will) remove any names and identifying features. We can also just explain how to reproduce our dataset without actually sharing any of the code with anyone.

3 GitHub

An overview of what we were doing collecting from github in the first place. Explain what to expect here.

3.1 GhTorrent and Google BigQuery

This should be about what is involved using GhTorrent and Google BigQuery for collecting our data. What are they, what did we user them for. ght is a collection of data on github that has been going on for a long time (link to their website). It is accessible through google BigQuery and we were able to filter out commits ad projects that had information and programming languages that we wanted. Some issues with it is the extraction of this data. If you save it in a google cloud account it is easy to keep, but not so easy to get the results if there are many of them. So we had to manually download them 10k rows at a time into json files. these rows were combined to created larger files that our scripts could run on. There may be ways to download large amounts of data, but I did not find them (should probably look this up again to make sure).

3.2 Using Commits

The first method we tried was to collect commit data from ght and then use the git api to get all the information for a commit. Included in this information are all the changes made to each file. since we could not be sure of a file that was modified being worked on by the given programmer before this modification, and the changes may not have reflected the work they did, or even work that was on nearby lines we decided not to take modified changes. We indtead restricted our search to added files. This way we could at least be certain that all the work was uploaded by the author even if we could not be sure they wrote it before uploading it. An idea was to only take files of a certain link imagining that library code would be larger. Some issues with the files though were that it was naturally hard to tell what condition the file was in and that added files are often a first commit of that file and many times have not been finished. So it is difficult to use this method to find adequate samples without manually inspecting them to see if they are valid code or not.

3.3 Using Projects

The next method we tried attempted to be a lot pickier about what code we took. The goal was to get a good number of files from a user and ensure that the code was single author code to the best of our ability. We employed several techniques here. first we only took projects that met our needs. Then when proceesing a project we validated it by checking to make sure it did not have more than 1 contributor. We also checked for a user name using the git api to find gender with. If there was no gender we did not process it either. If it passed all these tests we cloned the repo and ran git blame on all the files we were interested in. (.cpp .cc .py .java). once git blame was run we counted up all the individual authors and if the author of the file was not the author of the repository and the only one we did not take the file. If everything worked out we took it and added it to the db with the use and project info. Once all this was done the repository was deleted locally and we moved on to the next project.

The database had a project and file id as a way of determining duplicates. This way if for some reason we processed a project again it would not add duplicate code to the database. Since we did not take forked projects we also reduced the likelyhood of duplicate code. If for some reason an author decided to copy the code into a new repo and start fresh it is possible that it could make it into our database. Sometimes authors did have files with the same names in their repos (don't know why) and only one was added.

3.4 Statistics

4 Codeforces

4.1 Collecting User Data

4.2 Using Direct Links

4.3 Using Selenium

4.4 Statistics

5 Gender

5.1 genderize.io

5.2 Adding gender

5.3 Integrating Gender Collection Into Other Scripts

5.4 Statistics

6 Scripts

6.1 Github

6.2 Codeforces

6.3 Gender

6.4 Database Access

7 Extending The Code

8 Conclusion