Data mining on detekt repo

Lixuan Luo

1. Introduction

With the developments in the field of computer science and data science, big data has been paid more and more attention now. There are many useful tools emerged to make predictions or do the classification, and almost all of these tools are built with large and clear datasets. However, the fundamental basis for these developments is the data mining. Web scraping (also called Web harvesting or Web data extraction) is a software technique aimed at extracting information from websites[1]. Web scraping is an efficient method to transform the unstructured web data into structured data which can be used in data analysis. This report deals with the data mining on the detekt repository by a web scraping tool implemented in Python.

2. Background

Github, is a web-based open source hosting service for version control using Git. It provides access control and many collaboration features like bug tracing, feature request, task management, and documentations for every project[6].

For every project, the changes like add, edit or delete a file are recorded by commits. Commits create a transparent history of developer's work that others can follow to understand what have been done and why[7]. Meanwhile, each commit has a message that describes the particular changes.

Pull requests initiate discussion about user's commits. Users can open a Pull Request during the development process: when they want to share general ideas, are stuck and need help or advice, or are ready for others to review the work. Pull Requests help to start code review and conversation about proposed changes before they're merged into the master branch[7].

Issues are a tool to keep track of tasks, enhancements, and bugs for the projects. Each issue will have a title, an unique issue number, different labels categorizing and filtering issues, comments allowing others with access to the repository to provide feedback. Furthermore, it also has notifications, mentions and references. Mentions and notifications are used to notify other users and teams. References are used to connect the dependent issues and commits[8].

3. Data

The data that I crawled is from the detekt git repository, a static code analysis tool for the Kotlin programming language, which operates on the abstract syntax tree provided by the Kotlin compiler[9]. Here are the statistics as follows:

Related information	Numbers
Branches	5
Commits	2367
Closed issues	614
Open issues	113
Close pull requests	808
Open pull requests	9
Files	875

Form 1. Statistics about detekt repo

My goal is to mine the above Github repository, then:

1) Match the commit with the issues it solved.
2) Match the file with the issues that is solved in it.
3) Extract the bug issues and feature issues that are solved in each file respectively.

4. Method

4.1 Beautiful Soup

Beautiful Soup is a python library to pull data out of HTML or XML files which sits atop a parser to provide idiomatic ways of navigating, searching, and modifying the parse tree[10].

4.2 Urllib

The urllib is a python module which provides a high-level interface for fetching data across the World Wide Web[11]. Urllib.request is one part of the urllib. Urllib.request module defines functions and classes that help in opening URLs in real world.

4.3 Data scraping

To scrape the data from the Github repo, I combine Beautiful soup and urllib together to do the work.

The start link of the data scraping is the link of the commit page of the repository. Before the program goes into the next page of the commit page, it would do several things.

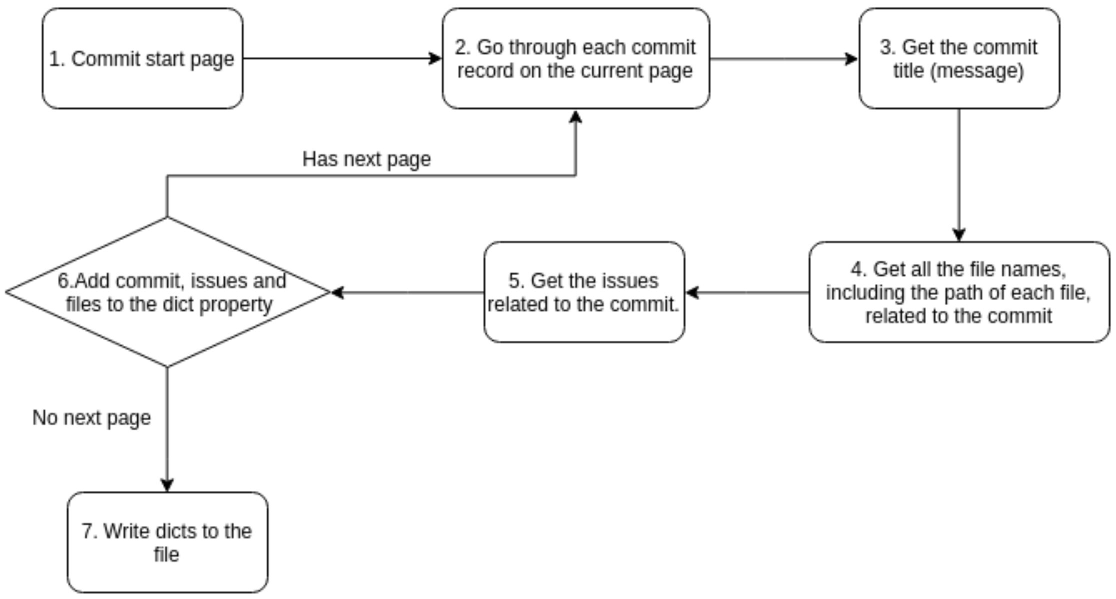
1. Commit start page
2. Go through each commit record on the current page
3. Get the commit title (message)
4. Get all the file names, including the path of each file, related to the commit
5. Get the issues related to the commit.
6.Add commit, issues and files to the dict property
Has next page
No next page
7. Write dicts to the file

Figure 1. Mining flow chart

For both step 4 and step 5, the program needs to request more link. Step 4 is relatively simpler comparing to the step 5, since it requires max one more request to get all the files related to the commit. As for the step 5, the program first looks into the pull request of the commit to see if there are any relevant keywords to the issue (e.g. “Fixes”, “Closes”) and go to the issue page to get the issue information like tabs to identify the issue as bug or feature. If the issue has a tag “bug”, we classify it as bug issue. And if the issue has a tag of “feature” or “improvement”, we identify it as a feature issue. Then the program continues to the next page and repeats step 2 - 6 until there is no more next page left.

5. Results & discussion

The experiment platform is python 3.7 on MacOS operating system. Based on the goal of my study, I output the results into 3 json files for further inspection.

5.1 Commits and issues

For this part, I store the data into a dictionary with key as the issue number/id and value as commit messages representing each commit. There are total 305 issues with corresponding commits. Here are the statistics:

Number of commits corresponding to one issue	Number of issues
1	216
2	38
3	18

4	11
5	6
6	4
7	4
8	2
10	3
11	1
12	1
14	1

Form 2. Issues and commits

By going through the json file, I found that issues fixed in one commit are often some minor changes in add or edit a test case, add an interface, add a corner case, update library version, etc. Issues fixed with more commits are usually add some individual changes in different commits, add test cases for these changes then finally combine together. From my perspective, the number of commits to an issue is related to the complexity of the issue itself. When issue is simple, less one commit can fix, otherwise, more commits might be needed for fixing by changing from different parts of the project.

5.2 Files and issues

For this part, I category the issues into two kinds, bug issue and feature issue. Then I build two dictionaries, the first one with file path as key and its corresponding bug issue number(id) as value, the second one with file path as key and its corresponding feature issue number(id) as value. We will go into detail separately for these two kinds.

5.2.1 Files and bugs

There are total 254 files I found with corresponding bug issues.
Firstly, most of files are Kotlin files under ‘src’ directory since the project is wrote in Kotlin and is a static code analysis tool for the *Kotlin* programming language. There are also some files related to environment building which are used to facilitate the bug fixing. Secondly, the match between files and commits are not bijective. There are files that fix more than one bug, and also there are bugs which are fixed in more than one file:

Number of bugs fixed in one file	Number of files
1	181
2	42
3	17
4	11
5	1

6	0
7	2

Form 3. Files and bug issues

5.2.2 Files and features

There are total 612 files I found with corresponding feature issues.

First, similarly as bug issues, most of files are Kotlin files since the project is wrote in Kotlin and new features should also be written in Kotlin. There are also some gradle files that relate to the environment setting for new features. Secondly, the match between files and features are not bijective. But most of the files match to one feature which is plausible since developers usually would build new feature in a file rather than continue in the previous features' files. Following are the statistics:

Number of features added in one file	Number of files
1	389
2	121
3	46
4	20
5	11
6	12
7	3
8	1
9	4
10	1
13	3
29	1

Form 4. Files and feature issues

Comparing the bugs with features, we can find that more feature files are added than bug files are fixed, which means the project is being improved to rich functionalities and stability gradually. However, in this report, I did not deal with the time which can also be researched. We can scrape the start time and close time for each bug issue and feature issue to investigate the responding/processing time for bug fixing, feature implementing, and during which time period the team focus more on bugs or features.

6. Challenges & Future work

Based on the current work so far, I can come up with 2 challenges with their future work.

The first challenge is requesting to the server as the web mining requires many requests to the same server and the request rate is too high to the server. The server wants to protect itself from spending too much resource on the web mining program, so that it sends the status code 429 to kindly remind the program not sending too much requests. The current solution is to make the program wait for several seconds and send the request again and if the same status code returns, the program will wait and try again until the correct response is given.

The second challenge is mining all the possible issues related to a commit. Since in the pull request page of a commit, it may contain links of other pull requests which may have the issue contained. In the future, the program can be perfected to support the multi-layer of pull request pages search to get all possible issues.

7. Reference

[1] Schrenk, M. Webbots, spiders, and screen scrapers: a guide to developing Internet agents with PHP/CURL. No Starch Press, 2007.

[2] Brett, M. Accessing Online Data: Web-Crawling and Information-ScrapingTechniques to Automate the Assembly of Research Data. Journal of Business Logistics, 2016, 37(1): 34–42.
Retrieved from:
https://onlinelibrary-wiley-com.myaccess.library.utoronto.ca/doi/epdf/10.1111/jbl.12120

[2] Eloisa, V. Mirko, U. Exploiting web scraping in a collaborative filtering- based approach to web advertising. Artificial Intelligence Research, 2013.
Retrieved from:
https://pdfs.semanticscholar.org/25cf/21117f60d80b32c6d2868defc39e39f74109.pdf?_ga=2.205333655.642143531.1552503263-1199444762.1552503263

[3] Claudia,V. Yehia, E. Dominik,R. Christopher J.A.M, Wouter, B. Web technologies for environmental Big Data. Environmental Modelling & Software Volume 63, January 2015, Pages 185-198.
Retrieved from:
https://www-sciencedirect-com.myaccess.library.utoronto.ca/science/article/pii/S1364815214002965

[5] Christoph,T. Larissa,L. Maurício, A. Unusual events in GitHub repositories. Journal of Systems and Software Volume 142, August 2018, Pages 237-247.
Retrived from:
https://www-sciencedirect-com.myaccess.library.utoronto.ca/science/article/pii/S0164121218300876

[6] Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/GitHub

[7] Github flow. Retrieved from: https://guides.github.com/introduction/flow/

[8] Github issues. Retrieved from: https://guides.github.com/features/issues/

[9] Detekt repository. Retrieved from:
https://github.com/arturbosch/detekt

[10] Beautiful Soup 4.4.0 documentation. Retrieved from:
https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[11] Urllib documentation. Retrieved from:
https://docs.python.org/2/library/urllib.html