Report on Mining Git Hub Repository -

Repo name : deteket , document author : sheik shameer S
https://github.com/arturbosch/detekt

Knowing the repo

I started knowing the repository by visiting the github link. Initial understanding were as follows.

- The repository is about code smelling and analysis.
- It is confined to a specific language of *Kotlin.*
- The primary owner was arturbosch.
- It has around 2500 commits.
- It has around 700 issues.

Setting up tools

The repository size, number of issues and commits were relatively small, so I decided to store the research data in a local machine and not access it via the github api. Using the github api was slow since it contacts the server each time. So i felt that storing the data in a **mysql** database was the best choice.

My primary entities to research around were the commits, issues, modified files, issue labels. So I started deciding on the database models.

- Entities table - (Commits, Issues, subscribers)
- Mapping tables - (CommitwithModifiedfiles, Issuewithlabels, CommitwithIssue)

The next questions to be answered were as follows.

1. Languages to be used
2. Fetching and populating data

3. Utilities necessary for my research

1. Languages to be used

Since mining was the task, i used **python3** for my needs. The main reasons for choosing this language was its availability of features in its package manager and its versatility in usage. Since my needs were not finalised, i needed a language that can handle changes smoothly, easily configurable and can integrate easily with packages. Spyder IDE was used to develop the coding repository.

I choose **mysql** as my database language because of its rich query library and the availability of workbench (IDES) and its easy setup.

Operating system used is **Ubuntu 18.04.1** LTS.

2. Fetching and populating data

I used **GitHub3** python module (https://github3py.readthedocs.io) for mining github using apis to fetch the data. The mining was slow due to the api calls , i was able to mine the local cloned repository for commits using **pydriller ,** a local repo mining python module (https://pydriller.readthedocs.io/en/latest/). This was fast since there were no api calls involved. However for fetching issues and subscribers i had to rely on github's api.

Once i populated my database , i was able to mine the issues and subscribers data as well without relying on api.

Please refer populatedb.py for more details on populating databases by mining github repository.

3. Utilities necessary for my research

I required to show data in formatted tables for which i used **tabulate,** a python module (https://pypi.org/project/tabulate/) . please refer CommitModifiedFilesUtil.py for more info on this.

Another requirement was to export data in excel format. **Xlsxwriter** (https://xlsxwriter.readthedocs.io/) was the python module i used. Please refer exportexcel.py for more information on this.

For connecting with myql db i used **pymysql** , a python module (https://pymysql.readthedocs.io/en/latest/) . please refer initializeconnections.py for more information on this.

Mining the repo :

The first question was to "Can you mine the commits and issues from this repo: https://github.com/arturbosch/detekt" . I was able to mine the repository using pydriller. I grabbed all the commit related information (hash, email of the commited user, committeddatea, is this a merge commit, branch in which is committed, number of modified files).

I was also able to mine the filenames of the files that were committed in a commit , number of lines added and removed as well.

The next question was to "Then can you link the commits with the issues - which issue was fixed in which commit?"

The problem was to know relation between the commits and the issues. After some digging in the repository and reading the github documentation best practises, I came to realize that the issues are mentioned in the commits message by prefixing with a # symbol. So i mined for the commits message and used regular expression to extract the issues concerned with the message. I populated these in a mysql table (CommitIssueMap).

The final specific question was to "Then based on that can you do a count of how many issues (bugs and features seperately) were fixed in each file of the project? "

The next challenge was to mine the issues and segregate them into different labels so that i can classify them as bugs, features etc.

I was able to get all the labels by mining the issue details using the github's api and populated them in a column prioritized table (IssueLabelMapping).

Now since i already had a map between commits and issues, I mined for Issues with labels as "bug" and "features" from my database. Once i got the result for that i linked the issues with the hash values of a commit. I already got the modified files for a commit in my previous mapping . So i was able to find the number of issues that were fixed in each file of the projects. The result is as follows.

Files touched for bugs :

FILENAME	TOTALBUGS	BUGNUMBERS
README.md	6	41,51,58,83,89,1039
default-detekt-config.yml	5	10,83,280,552,716
DetektExtension.kt	5	116,166,349,545,980
Junk.kt	5	18,41,69,1030,1107
KT.kt	5	41,115,428,442,534
build.gradle	4	41,58,69,87
ExpressionBodySyntax.kt	4	58,69,87,976
gradle.properties	4	41,58,87,373
Indentation.kt	4	41,69,89,1039
MagicNumberSpec.kt	4	276,280,659,992

The complete list can be found in the file filestouchedbybugs.xls in the results folder.

Files touched by features:

FILENAME	TOTAL	FEATURE
build.gradle	14	3,4,5,8,21,24,30,40,45,48,143,438,496,664
default-detekt-config.yml	12	4,30,36,45,48,53,56,66,129,552,664,844
Junk.kt	9	4,36,38,48,66,141,171,664,844
DetektPlugin.kt	8	16,21,24,48,62,67,143,542
DetektExtension.kt	7	16,24,48,62,67,92,542
Main.kt	6	3,4,24,48,92,171
KT.kt	5	6,16,36,48,129
KtCompiler.kt	5	6,16,45,66,171
Constants.kt	4	21,92,143,542
Detekt.kt	4	3,7,53,66

The complete list can be found in filestouchedbyfeatures.xls in the results folder.

Please refer populatedb.py , CommitModifiedFilesUtil.py for the complete code.

Some Findings based on the above mine bugs

- They had a practise of adding the contributors to their readme.md file. It is a good way to give credit to contributors.
- They default configurations seems to be adjusted around a lot depending on the bugs raised. For example , they tried to set the threshold for find Magic Numbers to be of greater length.

- The repository seems to have a lot of coding problems, where variables are not properly read from the configuration file. My guess would be that they did not have a default configuration file in the beginning, but later developed one. But there seems to be a lot of bugs in replacing the hard coded configuration value with the ones in the configuration files.
- They had a lot of bugs depending on the client environment.

Some FIndings based on the above feature mining

- Gradle is a file that is responsible for building a project. Seems they failed to provide a build config file for a sample project.
- They seem to develop features and put them in the default configuration files so that it has more visibility.
- They seems to be trouble in understanding how the user is willing to configure the project.
- Since most of the bugs and feature number indicate old numbers , they seem to have progressed well in fixing the projects.

Some Additional FIndings based on self mining

I used the same concept and found files that is touched for each label that is related to an issue and found the following.

- Again the top most hit was the configuration, the coders seems to know what they are doing in a code level. But since different consumers have different needs , they are having difficulty in allowing provisions for configurations. (filestouchedbyapi.xls)
- On checking the files touched by block labels , seems they have a baseline.xml file that marks the thresholds for the analysis (filestouchedbyblocked.xls)
- Since most of the documentation issues revolves around default-detekt-config.yml file , it seems the project is not versatile enough and they need to work on that. (filestouchedbydocumentation.xls)

- Most of the issues marked easy were relatively asking for examples , it would mean there is not enough documentation and examples available (filestouchedbyeasy.xls)
- They had a problem with merging. Seems some unknown code comes via merge (filestouchedbyfalse-positve.xls)
- Most of the formatting issues revolved around Expressionbodysyntax (filestouchedbyformatting.xls)
- They seems to be difficulty keeping up with kotlin coding conventions upgrade. (filestouchedbyhelpwanted.xls)
- There seems to be a lot of regression bugs but the label is not properly updated in the issues mostly due to merge conflicts (filestouchedbyregression.xls)
- Seems they did a code revamp depending on this issue https://github.com/arturbosch/detekt/issues/326 (filestouchedbysonar.xls)
- Getting a total count of the different issues corresponding to different labels revealed that the issues related to rules were more. The particular file used for checkin is the default-detekt-config.yml . This proves that there was a problem in understanding the specific requirements of a customer. (LabelstoTotalIssues.xls)
- I tried mining the frequency of creation of issue with respect to the individual labels. Most of the issues were created in the later half of the year 2017. This marked the time where the project was active at its peak. (filenames prefixed with frequencyofcreation*.xls)
- Around 71 features were closed on the 6th and 7th month of 2017, which inidcate their peak time as well. (frequencyofclosedfeature.xls)
- Next I went on to mine the user related activites in the repository.On mining the commits in the repository , i found that the project repo is mostly build by 3 people (arturbosch,shalkms and mauin). A count of total commits revealed this. (UserCommits.xls).
- A user named vanniktech seems to have created a lot of relevant bugs in the repository.(Usercreated*.xls) Depending on how he uses the repository we might be able to get a good understanding of the repository.

- Another important aspect was the subscribers , the subscribers peaked at the 10th month of 2016 amounting to around 53. And it seems they were the initial subscribers and were not propelled by a particular feature. There were not any features that were closed in 2016.
- On mining the number of issues created by each user with respect to each label , i was able to identify the users concern regarding a particular part of the repository . All the issues mostly revolved around 4 users (arturbosch, vanniktech,schalkms,Mauin).
- Since i was not able to find any data regarding security , I feel that there was not any proper consideration for the security of the project. They could atleast mention that they have run security tests which may improve their project usage.

Some Future mining that can be done :

Problem - The issues are not properly labeled . For example, it is mentioned in a comment body that this issue happened due to regression. But the issue was not labeled as regression.

Solution - We could apply natural language processing to find if a comment is about regression and automatically select the regression label.

Methodology -

1. mine all issues comments using github3 module.
2. Store all messages in a database.
3. Find token of word using nltk python library. (https://www.nltk.org/)
4. Store them in a column prioritized table.
5. Find all distinct label in a repository. (refer getDistinctLabels() in CommitModifiedFilesUtil.py)
6. Check if the issue has those labels.
7. If not update the label using the github3 api.