Aboutcode/ScanCode_Toolkit

Project Proposal
GSoC 2020

Improve ScanCode License detection accuracy, by leveraging the ClearlyDefined dataset of Scans

Ayan Sinha Mahapatra

Abstract

ScanCode license detection is using multiple techniques to accurately detect licenses based on automatons, inverted indexes and multiple sequence alignments. The detection is not always accurate enough. The goal of this project is to improve the accuracy of license detection leveraging the ClearlyDefined data set, where ScanCode is used to massively scan millions of packages.

Some of the cases (not limited to) where this project proposes to improve license detection accuracy are:

- when multiple licenses are detected with a low score and some detections are incorrect.
- when some unknown licenses may not be detected correctly.
- text/code identical to license tags resulting in false-positives
- when license references such as "see license in file LICENSE.txt" are reported as unknown license references.

This project aims to write tools and create models to massively analyze the accuracy of license detection and detect areas where the accuracy could be improved. These tools and models would be reusable to assist in the semi-automated reviews of scan results. It will also create new license detection rules semi-automatically to fix the detected anomalies.

Goals to Achieve

1. **Improved License Detection Accuracy:** This is the main goal of the project, i.e. to improve the license detection accuracy, and not just on the ClearlyDefined dataset (adding rules corresponding to the errors in that dataset), but as a whole by gaining more insight into the Scan Results.
2. **Reusable Tools/Models to assist in Semi-automated review of Scan Results:** In the course of analyzing the dataset, the tools and models created have to be re-usable and robust so that it can be used by the community in future easily, to review more scan results. These can be deployed to GCP, i.e. support will be added so these can be deployed and used via GCP very easily.
3. **Semi-Automatic Creation of License Detection Rules:** From the “matched_text” attribute of detected licenses, to create automatic license detections rules (wherever possible), and even try semi-automatic, basic .yml file generation.
4. **Statistical Analysis to gain more insight into Scan Anomalies:** Statistical Analysis of all license detection scans (in terms of scores, marched rules, rule lengths, matchers etc), and even analysis of existing rules and license texts, to gain insights into anomalies, helping with further ScanCode roadmap.

About Me

Name	:	Ayan Sinha Mahapatra
Website	:	https://ayansinha.dev
Proposal Link	:	Uploaded at my Website Google Doc Draft (Will be updated here)
Link to Resume	:	Resume Uploaded at my Website
Proof of Enrollment	:	Proof of Enrollment required for GSoC 2020
Timezone	:	Asia/Kolkata (UTC+05:30)
Email	:	ayansmahapatra@gmail.com
Course	:	Bachelors, Electronics and Tele-Communications Engg.
University	:	Jadavpur University, Kolkata
Year	:	4th Year (Senior Year)
Country	:	India
Obligations	:	None
Links	:	Github LinkedIn Twitter Portfolio

Introducing Myself

I'm Ayan Sinha Mahapatra, 4th-year Undergrad at Jadavpur University, Kolkata. Contributing to open source software has been a dream to me because of the impact it has and what it means to us as a society. I'm highly motivated to pursue research and remain in academia after graduating, specifically studying computational intelligence and how to achieve intelligent behavior. As a Deep Learning/Neuroscience enthusiast, all of my work uses Open-Source libraries ranging from Tensorflow, Scikit-Learn, Numpy for Machine Learning and Data Science to MNE for EEG data processing.

I'm very comfortable with Python and C in general, as I have coding/project experience in both of them. I'm very comfortable with Linux, Scripting, Git/GitHub, and also very familiar with ScanCode as I completed my GSoD '19 project in the same.

My machine learning experience includes building a prototype signature verification system, deployed as a service in an AWS instance written in Tensorflow/Flask, EEG signal processing and Classification using Deep Learning written in Tensorflow/MNE-python. I've also participated in several beginner Kaggle competitions, with topics ranging in Anomaly Detection, Text Classification etc.

I have also written code to develop meaningful projects myself and with friends in a collaborative environment that are available in my GitHub account. Alongside, as an effort to better equip my juniors in my university with trends in Machine Learning, I host sessions on Image/Data Processing. I'm also a part of the university's Code-Club (promotes Competitive Coding and Open Source Contributions) and Kaggle Club (promotes Data Science and Machine Learning), where I regularly host sessions and volunteer.

I may not be an exceptional individual, but I'm highly motivated to take on challenges and learn from them as I complete my objectives.

Why I’m applying for the Google Summer of Code @ AboutCode

The concept of people from all backgrounds coming together as a team from all parts of the world to write impactful code, that anyone can use for their own cause, amazes me. Also, as I use open-source software extensively in my projects/research work, I know by heart the value of an open-source project having an amazing community around it, making it better every day collectively. I’ve worked for Aboudcode before through GSoD, to improve the documentation experience, and now I’m looking to contribute code/analysis-pipelines more directly.

Why I’m Perfect for This Project

This project, Improve Scancode License Detection Accuracy, is to write reusable tools and create models to massively analyze and to assist in the semi-automated reviews of scan results. Thus in its essence, it has certain knowledge/skills requirements like:

- ScanCode License Detection System
- ScanCode Licenses/Rules
- Statistical Analysis
- Machine Learning Models (Anomaly/Patterns Detection)
- Machine Learning Models (Natural Language Processing)
- GCP/AWS cloud compute instances
- Data formatting/analysis at scale
- ssh and other remote access tech
- Linux, Scripting
- Git/GitHub

I have all the Statistical-Analysis/Machine-Learning/Cloud-Compute knowledge required, i’ve worked with ScanCode license detection bugs and adding new licenses, and I’m very familiar with Git/Github, Linux as demonstrated in my previous GSoD project with Aboutcode. I can work

under guidance and produce results timely. As these requirements perfectly overlap with my skills, I think I'd be perfect for this project.

This is why I decided that I'll apply for GSoC@AboutCode and I am very excited about the work ahead.

Contributions to Aboutcode [WIP]

License Detection Bugs

PR - https://github.com/nexB/scancode-toolkit/pull/1963

Fixes #1907 #1910 #1911 #1912 #1914

Also I've reproduced and checked other bugs #1933 #1932 #1931 #1930 #1929 #1928 #1927 #1926 #1925 #1924 #1923 #1922 #1921 #1920 #1919 #1918 #1917 #1915.

New License Additions

Adds new licenses and related license detection rules/scripts.

Solves nexB/scancode-toolkit#863 and some more related issues.

Scancode Cookiecutter-Plugin

Adds support for cookiecutter plugins for ScanCode Toolkit. Simple post-scan plugin added, more cookiecutter plugins and support docs to be added.

Repository - https://github.com/nexB/scancode-toolkit-plugin-cookiecutter

Google Season of Docs 2019 Contributions

All contributions can be found at https://ayansinha.dev/assets/gsod-report.pdf or GSoD Report. Added scancode-toolkit.readthedocs.io and aboutcode.readthedocs.io.

Project in Details

The project aim is to improve the license detection accuracy of ScanCode. Presently, if we look into incorrect scan results across a wide variety of bugs reported using the issue tracker we see that these are the main reasons where scancode could have given better results (i.e. where there is a scope for improvement)

- Wrong Detections (Wrong, often multiple licenses are detected)
- False Positives (The correct license was detected with a low score, and there were more license detections which shouldn't have been there)
- When some unknown licenses and license referances/notices may not be detected correctly

Now, presently these bugs have to be solved by looking at them individually, figuring out what was wrong, and in most cases, craft appropriate rules to be added, which solves that particular bug.

As this project aims to leverage the ClearlyDefined dataset which has 10 Million+ scans, it obviously is impossible to look into scans individually. So the goal is to create a system which can help review scan results in a semi-automated manner. Since we can map license detection errors to license scan attributes (like in most cases, if the license scan score of an individual file is low, the detections are incorrect), we can segregate these large numbers of scans into smaller groups/types of scans to review and address these problems better.

Case-Specific Segregation, Processing and Eventually Automatic Rule Generation for License Detection Accuracy

1. Data Formatting, Assertion, Stripping and Loading

Goals:

1. **Loading** Scan Result Data from how it is collected from the ClearlyDefined dataset, and into Pandas DataFrame Objects, one for each license detection, which can be multiple per file. It will have both file-level and license-detection level attributes necessary for further analysis.
2. **Asserting** that the Scan Results data follow a particular format, as using different scan options may result in different data format (like "--summary" causes an additional summary section, and older scancode versions may have differently structured output data). Implement proper error-handling/assertions to eliminate load-time errors.
3. **Stripping** unnecessary attributes from the Scan Data, to make the dataframe objects smaller, and only have essential data for analysis. Example - Stripping copyright/package data.
4. **Formatting** these data into Pandas DataFrames, in batches, with proper "input_from_file", and "output_to_file" functions, which would be used throughout the project in every step, as it's much better to divide the whole pipeline into parts rather than running one script. So the whole analysis is divided into stages, and these stages into batches of data as necessary(smaller batches at first to make sure everything is working properly and later scale up).

ScanCode scan results are JSON files with a predefined structure, which can be loaded into Python objects having the same schema or into Pandas DataFrames with some careful formatting, which we will use.

Now, as the goal of this project is license detection, we might ignore other information such as “copyrights”, ”packages”, “holders”, “authors”, “emails”, “urls”, “dirs_count”, “size_count”, “scan_errors”, “name”, “base_name”, “size”, “date”.

The “licenses” attribute often has multiple objects, for multiple detections in a file, and for each “licenses” object, i.e. each license detections, we’ll have a single dataframe object (one row in a table like object), having these features:

- license-object level attributes like “key”, “score”, “start_line”, “end_line”, “matcher”, “match_coverage” etc. (Here only url attributes will be left out, like “homepage_url”, “text_url”, “referance_url”, “spdx_url” as they will not be required, all other attributes important for the analysis will be kept intact)
- file-level attributes attributes like “sha1”, “extension” “is_text”, “is_source” etc added to each of them, as these hold information required later.
- As the primary key a combination of the “sha1” value and a numerical identifier like 1, 2 .. for all the detections in a single file. I.e. for 3 license detections in a file their primary keys will be “sha1_value”+”1”, “sha1_value”+”2” …

2. Level 1 Case Specific Classification

Goal:

Divide into 3 Parts, each of them a specific case

- Unknown License texts (entire file with a license text, not detected accurately)
- License Tag Detections (For False Positives among them)
- And the most Important, when multiple licenses are detected with a low score and some detections are incorrect.

Multiple Licenses with Low Score

There's a license detection score associated with each license detection, which is often multiple per file, with some/a lot of those scores not perfect (not 100).

These scores again depends on factors such as "match_coverage" (matched_len/rule_len) and "rule_relevance" and rule relevance is either given explicitly with each rule in it's .yml file or computed dynamically based on it's length (a rule of length smaller than a threshold has a relevance of 100/threshold, rounded down, where threshold is `18`).

Now it can be any of these cases -

- Entirely new license which is a mixture of already known ones [WITH exceptions, OR, AND cases]
- Multiple False Positives(some with low scores), even when the correct(similar) license rule match score is 100, here there might be more than one area of interest location wise, one is detected properly, another is not.
- Multiple False Positives, the correct(similar) license rule is weak, suggest stronger rules.
- Multiple False Positives, a similar license (the correct one) rule is missing
- New previously unseen License Notice, triggering False Positives
- Some detections are matched with "2-aho" (usually smaller rules) and some detected by "3-seq" where some words are common, detected by set-matching.

Now, if there's a perfect rule matching (by exact matching automatons), it exits and there doesn't exist multiple rule detections, so in most of these cases, if there's multiple detections in a single location inside the file (this location part will be detailed later) and one or more of these detections have low scores, they are bugs/wrong detections where the accuracy could be improved.

In this part we'd have to exactly segregate this class of errors, I propose some attribute based segregation:

- One or more detections with matcher: "3-seq"
- One or more detections with low score (say equal to or less than 90) [ToDo Select Value based on Rule Score and other Analysis]
- One or more detections with match_coverage less than 95 (ToDo select value)
- conflicting license detections (like gpl 2.0 vs gpl 2.0 plus or bsd-new vs gpl 2.0 plus etc)

Here, we can't select all scores less than 100, because in some cases when license texts are accompanied by some introductory words about the code/script itself, or some very minor variations which does not mean a lot in terms of what the license actually is. These cases, (i.e. maybe between scores 100 - 90) might be/might not be an inaccuracy and so this can be another separate class and could be analysed further. This is seperated, so the obvious inaccuracies can be addressed properly and surely. This is also a case where linear regression based mapping can be used later to train accurate/inaccurate detections.

License Tag Detections (For False Positives among them)

A lot of small license tags like texts (like a lot of texts with “gpl” or “bsd” or other words in them) get detected as ScanCode detects license tags based on 1 word (or other very small) rules.

Also there’s a lot of instances of structured license information being present in a class of projects (like MODULE_LICENSE in linux kernels), which normally gets picked up by license tag rules.

Now there’s a lot of false positives like these, and generally ScanCode has a lot of negative rules to weed out non-license tag instances, keeping correct detections accurate.

Here to detect we can use these attributes, but stricter assertion may be needed.

- is_license_tag : True
- is_license_text, is_license_notice : False
- (mostly) is_source/script: true
- rule_len : 1 (ToDo Look Into all license tag rules if)

Unknown License Texts

Normally license texts gets detected by "matcher": "1-hash" as these texts are mostly copy pasted into files when used, but in case it’s a new license text which we haven’t seen before, it might get detected as other general rules having other unknown-like tags, (i.e. “unknown”, “other-copyleft” etc)

Now all detections which have these attributes can be segregated as a different class and treated.

- has extensions like .txt, .rst or has LICENSE, COPYING etc in their filenames
- "is_license_text": true, (license-level)
- "is_text": true, (file-level)
- Non-perfect scores (not 100)
- Multiple license detections per file

3. [Case 2] Unknown (New, previously unseen) License Text

This case deals with how unknown license texts are dealt with, links to already existing descriptions are as follow:

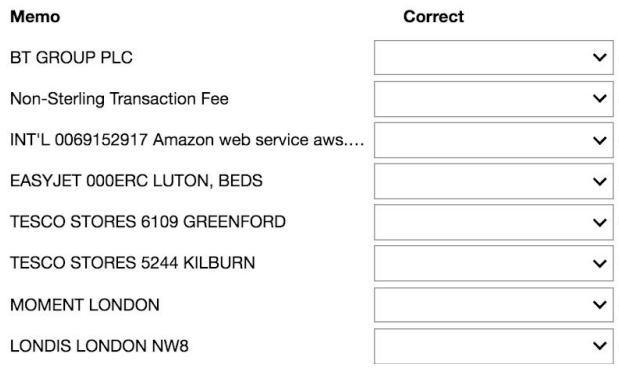
- How to Detect/Segregate
- Adding new Licenses Summed up here - Comment 1 Comment 2
- Adding new tests for these licenses - Comment 3

4. [Case 3] All License Tags Detections, for possible False Positives

After segregation, all these License tag Detections have to be reviewed semi-autonomously, as it's entirely dependent on the context whether it is a correct detection or a False Positive. Now, training a Sentence/Phrase Classification Neural Network (Trained on License Corpuses) would make sense here, but would require a well labeled dataset to be further trained on to detect accurately.

Examples of Such False Positives -
https://github.com/nexB/scancode-toolkit/issues/1914
https://github.com/nexB/scancode-toolkit/issues/1928

So, it makes sense to implement a **GUI based interactive review framework**, where the matched text is shown, and there's correct/False Positive Buttons to review and mark these cases. Example -
https://github.com/pbugnion/jupyter-widgets-for-data-science-guis



Here, in place of “memo”, there’ll be matched texts, and the “correct” drop-down list will have options whether it’s correct, False Positive where matched text is a non-generic negative rule, and where you’d have to actually download the scanned file and add lines from before/after to make it a non-generic effective negative rule. Here generic means rules that are very similar to actual license tags and would misclassify license tags as wrong. [Example]

Then results from this marked dataset can be used for further crafting Negative Rules, in cases of False Positives. Care should be taken here, as these rules should not be very generic, as then it could match with actual license tags. This is in addition to the already existing Negative rules in the ScanCode rules, which can also be used as negative rule examples.

Once a large number of tag detections are classified semi-autonomously, this creates a dataset for the network to be trained with. If the accuracy is reasonable, this can be deployed in the pipeline, reviewing only doubtful cases.

5. [Case 1] Removing Identical Cases across the same Package

Across one same package, there can be similar faulty license detection results, as there might exist same license notices/pieces of text throughout the package files, referring to

- a same license
- some specific exception/variation of the same license

Example - https://github.com/nexB/scancode-toolkit/issues/1920 where the same bug was seen across files in a same package. This happens quite often and thus there’s a lot of opportunity here to detect cases like these and eliminate all of them but one, to basically eliminate duplicates, but also being careful enough not to eliminate useful information.

It’s safe in that case to delete exact sets of license detections (and matched texts/rules).

Also analyze the matched texts for structured and tagged license notices, i.e. this issue - https://github.com/nexB/scancode-toolkit/issues/707, and as this information has to do with package information in some cases, explore possibilities.

6. [Case 1] Level 2 Case Specific Classification

Some of these cases can be better handled if they are segregated and looked into individually, as:

- there might be specific structure to the problems that require special attention.
- They might also vary in terms of Rule/.yml File Generation, so it's easier to automate those processes.
    - ❖ In cases "see license in file LICENSE.txt", "referanced_filename" attribute has to be added
    - ❖ In case of license tag false positives, "is_negative" is added.
    - ❖ Segregation by License text/notice/tag/reference attributes.

Also, there might be more insight into these as the project starts and data is reviewed, i.e. it's likely there will be more sections here then.

In some cases, unique structures in rules/target texts trigger incorrect license detections, which are largely case specific. Like in Example 2 of this comment, presence of HTML tags in a RULE results in license detections which are wrong, as there's no mention of the specific license tags, but just the structure of the target texts, having those html tags, trigger the match.

Some of these classes could be -

- **“Unknown” License Detection**
- **Improve Known License Detection**
- **License tags having “AND”/”OR” in their names**
- **"see license in file LICENSE.txt" reported as unknown license references**

More groups of cases like these are to be detected from the license detection datasets. Clustering the matched texts (after converting to vectors i.e. Word2Vec) might be a good approach here, using dimensionality reduction methods like t-sne.

7. [Case 1] Group License Detections by location in file and delete Correct Detections

In license detection, in a file, there’s often several groups of text which could be license related texts, i.e. after the whole file is converted from words to integer values, and then these values (common license text words have low value, less common, more discriminant words have a higher value) are judged against the threshold. Now after files are converted to queries, these queries are converted to slices using these id values and heuristics.

So, there can be multiple slices of important texts, in multiple locations of the file. And not all of these has an incorrect license detection, it can be the case that

- All of these slices are detected correctly
- Some detected correctly, some not detected correctly.
- All of these were detected incorrectly.

Example Issue, where

- the first GPL detection is for this line range -

  "start_line": 8, "end_line": 19,

- the second is for the free-unknown for this line range:

  "start_line": 349, "end_line": 350,

Now, for a single file, we group all the detections by locations, and then delete the ones which are correctly detected, and keep the wrong detections.

Correct/Wrong Detections here are defined in the same way as in **Level 1 Case Specific Classification -> Multiple Detections with Low Scores**.

As, we have line number information, i.e. starting and end line number information for all the license detections, grouping them would follow this algorithm:

Algorithm

- Find the Longest Match, i.e. Highest value of (endline-startline), these are new boundaries of a group.
- For all other license detections :-
    - ❖ If entirely inside the boundary, drop.
    - ❖ If partly inside the boundary, extend boundaries to include this and then drop.
    - ❖ If very close to boundary, i.e. less than a threshold, extend boundaries to include, then drop.
    - ❖ Else, skip.
- Repeat until there's no groups left.

As there's never too many detections in a file, and there's almost always detections which have almost all of the matched texts, this will be efficient enough.

8. [Case 1] Automatic Rule Generation by Stitching License Texts together

The last part of this pipeline is creating new rules (automatically) to add to the existing repository of rules, and even attempts semi-automated .yml file generation.

Rule Generation

This Algorithm groups the license detections by location, and essentially at last these are the boundaries (start and end) of the matched text, and by stitching all the matched texts together from all these license detections we get the whole text “query”, which is almost always the Rule text to be added.

So, in the algorithm, along with keeping track of the boundaries, we also stitch the matched texts together one by one, in order to generate the final Rule text.

.yml Generation

Important .yml fields are

- license_expression:
- is_license_text:
- is_license_notice:
- is_license_reference:
- is_license_tag:
- is_negative:
- relevance:
- notes:
- Referenced_filenames
- ignorable_urls:
- Ignorable_copyrights:
- ignorable_holders:
- only_known_words:

We handle “is_negative” in this case. Almost always “license_expression” has to be entered manually, as it is complicated and requires a lot of context. These tasks can be sped up significantly by using a **GUI based interactive review framework**, like this one.

Statistical Analysis of Rules, License texts and the Scan Dataset

Statistical Analysis of the Scan Dataset will mainly mean how all the scan attributes relate to all the anomalies/inaccuracies in the ClearlyDefined Scan Dataset.

Examples of Attributes based on which statistical analysis could be performed to understand how these scans/results vary/compare in these fields:

1. It terms of matchers (stages of matching “1-hash”, “2-aho”, “3-seq”)
2. Rule Length, Match Coverages
3. Across License Texts/Notices/References/Tags.
4. File Level Differences (is_text, is_source, is_script,)
5. Project level Differences (programming_language, type of projects)
6. Rules, i.e. which rules are detected how much, in which cases

Outcomes -

- detect areas where the accuracy could be improved
- standardizing relevance scores and minimum coverages across all the rules, after performing statistics on false positive/unknown matches
- review of how we can improve the way we deal with multiple "low score" detected licenses in a single file

Regression/Gradient based Attributes to Case Mapping

Creating a machine learning model is helpful in segregating/reviewing, but in more complicated problems, where there doesn't exist simple mappings from the Scan Attributes to the separate cases, but there's some context/complex combinations of these attributes.

As this project progresses, we'll have a perfectly "labeled" dataset, as we are reviewing and segregating these scans into groups of inaccuracies/similar areas where accuracy could be improved. These cases are multiple labels which we can train simpler traditional machine learning techniques like regression/SVMs/GBMs/RFs to train multi-label classifiers which can be used to directly segregate future Scan datasets into these cases and get insights/create Rules.

Machine Learning (Natural Language Processing) specific Modelling

One of the major tasks which can be used in multiple cases and stages of this project is sentence based multi-label classification.

Cases where this can be used:

- Assigning License specific labels like "copyleft", "permissive", "proprietary", "public-domain" etc
- Classify License Tag detections whether Correct/False Positive
- Deciding between cases based on matched texts (as Context/wording is important)
- Assigning License Text/Notice/Reference labels (or other labels) to .yml files

For these tasks, BERT-Based Sentence Classification is proposed, which is well-researched, SOTA, and has extensive open-source libraries to facilitate the same without a lot of effort. Essentially these large models are trained on large corpus texts (like Wikipedia) to complete sentences, generate next sentences etc, which are tasks requiring extensive context awareness. These models then can be deployed to other tasks like sentence classification (Input length is not fixed obviously) by modifying the output layers, and fine-tuning according to Target datasets/problems.

Packages like ernie support all these functions and can be fine-tuned on -

- License Corpus texts (Say all rules/license texts in scancode)
- Fine Tuned for How long/short the input output is.

As BERT is very large and memory-intensive, DistilBERT or other shorter versions can also be used, without losing any accuracy, as the license corpus text to be fine tuned doesn't have the large variation of words/contexts of other more universal tasks.

Project Timeline

[Link to GSoC 2020 Timeline]

Before the Community Bonding Period

- Setting Up ClearlyDefined services locally to play with scan data
- Getting more familiar with ScanCode license detection
- Solving more License Detection Related Bugs
- Add more Unknown licenses and more tests/rules for better detection
- Discuss the project proposal in more details with my Mentors

Community Bonding Period [May 4 - June 1, 2020]

- Get smaller parts of the actual ClearlyDefined data to get familiar
- More Detailed Research/Literature Survey on ML models and Training Parameters
- Brush up on Statistical-Analysis/Anomaly-Detection for Specific Use cases
- Finalize the Project in a more detailed manner, i.e. make more mockups
- Discuss my project in details with the mentors, and the community as a whole to integrate their feedback.

Coding Begins [June 1 - August 31, 2020]

Coding Part 1 (June 1 - June 28) Coding the pipeline and Testing with ~5% Data locally	
June 1 - June 5	Loading, Formatting, Assertion, Stripping

June 6 - June 10	Level 1 Case Specific Classification
June 11 - June 15	Case 2 Unknown License Texts
June 16 - June 20	Case 3 All License Tags Detections, for possible False Positives
June 20 - June 24	Case 1 - Removing Identical Cases And Level 2 Case Specific Classification
June 25 - June 28	Case 1 - Group by Location, Stitching Matched Texts for Automatic Rule Generation
Evaluation 1 (June 29 - July 3)	
Coding Part 2 (July 4 - July 26) Scaling Up to the entire data (GCP) and Generating Rules Start Analysis and Crafting ML Models	
July 4 - July 7	Execute Pipelines On Data Batches
July 8 - July 11	Start Statistical Analysis/ML Models and Continue in background for Part 2
July 12 - July 15	Execute Pipelines On Data Batches
July 16 - July 19	Execute Pipelines On Data Batches
July 20 - July 23	Finalize Rules/.yml files to Add
July 24 - July 26	Inspect License Detection complexities
Evaluation 2 (July 27 - July 31)	
Coding Part 3 (August 1 - August 23) Generate Statistical Analysis Reports and Finalize ML Model Uses	
August 1 - August 6	Test and Use ML Models
August 7 - August 12	Statistical Analysis on License Detections
August 13 - August 18	Buffer Period*
August 19 - August 23	Working on submitting the final analysis pipelines, review reports, project reports and mentor evaluations.
Final Evaluation and Code Submission (August 24 -31)	

* This Buffer Period is reserved for unplanned events or any other part taking more time

Commitments

During the GSoC 2020 timeline, i.e. June 1 to August 31, I have no other prior commitments and can work for a full 45 hours per week in a regular work pattern. I will be available in all the communication channels (Gitter, Mail, UberConference as required) throughout, even in the days after proposal submission, i.e. Application Review and Community Bonding Period. As I'm in my last year of college, my final exams will be over by the second week of May latest. And after college I'd be working part-time for the same research lab I interned at, which will start late into August - September, after my Final Year Exam Results are out. So I'll have no other commitments in this period, whatsoever. Thus I can fully focus on Google Summer Of Code and quality coding, research and analysis to increase license detection accuracy of ScanCode.

Expectations from Mentors

During the GSoC 2020 period, my expectations from the mentors will be as follows:

- Helping me Understand the nuances of License Detection in ScanCode. (I'm already familiar to an extent, but in case any complications arise)
- Helping me Understand the Codebase when required if I'm unable to understand on my own. (I'm already familiar but in case any complications arise)
- Provide me with necessary pointers/links if I need to pick up extra concepts regarding my work when required if I'm unable to find the same on my own.

- To formulate more broadly the area of work in accordance with the existing plans of the organization, and have discussions whenever important decisions are taken.
- To be straightforward about the direction/quality of work and give feedback for path correction whenever necessary, throughout the work period.
- Procurement of the ClearlyDefined Scan Database, and help in issues on the same.
- With GCP credits, as discussed, in the 2nd part of the project where we scale up and analyse the entire scan dataset (in batches as necessary).
- Take time to review my work and integrate it into the Organization Workflow/Repositories.

Long Term Goals after GSoC

GSoC ‘20 is only a stepping stone for me into Open Source Contributions, which started with GSoD ‘19. My GSoC project will improve the license detection accuracy of scancode-toolkit, but this is really a continuous process as a whole and this project will open up many more areas of work/improvement. There will be more data to scan and as both Scancode and ClearlyDefined evolves, there’d be adjustments to make and incremental improvements, even new methods/tools for semi-automated review could be introduced, for more insight.

I would also love to expand and make contributions to more OSS projects. I’ll continue other community endeavors at my university to mentor younger minds into open source.