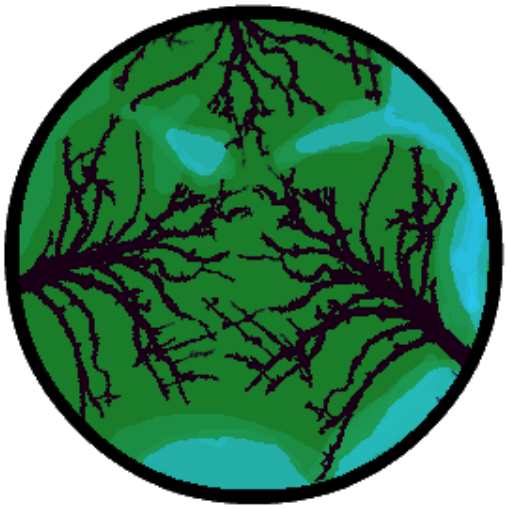
Software Testing Plan

April 3rd, 2020

Canopy - Team 11

Team Members:
Robert Plueger
Dongyu Xia
Maria Granroth
Sponsor: Dr. Patrick Jantz, Ph.D.
Mentor: Scooter Nowak



Version 1.2

Table of contents

Introduction

This project is to develop an application for better understanding forest structure and health. The client of this project is Dr. Patrick jantz. He is a member of the NAU Vegetation Structure as an Important Biodiversity Variable Project. His work is to analyse vegetation structure data from the Global Ecosystem Dynamics Survey (GEDI). The analysis can be used to improve land use decisions and protect the biodiversity in tropical landscapes. However, there are some problems with his current workflow. Analyzing the data by the R language is inconvenient, and the process is time consuming. The results are hard to understand or comprehend for most people. The application we developed will help users automatically analyze the GEDI data and will reduce the time and energy used. The result will also be easier to understand.

In order to ensure that all functions can run as expected, we will conduct software tests. In the process of software testing, we will first conduct unit testing. Unit tests make it easy to troubleshoot problems that may arise, and can also ensure that each unit is functioning as expected. We will use the data written by ourselves for the temporary testing of each module, which is simpler and faster for the unit tests. The GEDI data test will be carried out in the integration test. When we conduct integration testing, it can ensure mutual cooperation between different modules. Finally, we will do usability testing to ensure that our application is user-friendly and customers can use our products to replace our client's previous workflow.

In unit testing, we will use PyTest to test **Statistical Functions, Clustering (Classify Forest Types)**, and **Email Validation**. Because we only need to perform some small tests, PyTest is appropriate and can be successfully integrated with our existing code and systems. At the same time, we will also use valid and invalid data for testing to ensure that he will not output wrong results. In integration testing, We will test whether the interaction between each unit can work as we expect. In usability testing, we will design questionnaires to ensure that our product is user-friendly. Our plans for each testing category will be explained further in this document.

Unit Testing

The first step of our testing plan is to do unit testing. Unit testing is the process of testing the smallest possible individual components of a system. Its goal is to determine if all components are working as intended. Our general procedure will be to test each function using the PyTest framework to streamline the tests and help us with gathering metrics. For the metrics in particular, we will use PyTest's pytest-cov plugin, which tests code coverage by determining if all of the lines of code in a given unit are ran during a test. This will help us determine if our testing has covered all of the code that needs to be tested. PyTest seemed appropriate, as we only need small tests performed, and it seems that it will smoothly integrate with our existing code and system. In this document, we will be outlining tests for the following units:

- Statistical functions
    - Number of Observations
    - Time of Year
    - Density of Observations
    - Standard Deviation of the Density of Observations
    - Average Canopy Height
    - Data Quality
    - Classify Forest Types (see clustering, below)
- Clustering (Classify Forest Types)
- Email validation

Statistical Functions

We will test the statistical data that users need to use when analyzing data. Because GEDI data is large and needs a lot of time to run, it is not suitable for testing functions. We plan to write unofficial data to test. The unofficial data can reduce the testing time and it is easy to change the data to be invalid for further testing. Each statistical function will have its own equivalence partitions and boundary values.

Number of Observations

This function will count how many observations have been made in this data. In brief, this function will count how many pieces of data are in the files. Here, we divide the input file to three equivalence partitions: a file with no data, one with only 1 piece of data, and one with 10 pieces of data. If the result of the function is 0, it means the file is invalid, and the function shouldn't work. The boundary value is 1. If the result of the function is 1, it should be output as 1. The same with 10 pieces of data file.

Time of Year

This function will output the observation time of this data. GEDI is an Earth Ventures mission that started in December 2018 and takes two years. The time of year should be between 2018 and 2020. The result out of this range should be invaild.

Density of Observations

This function will output the observation density of the area, It needs to call the **Number of Observations** function, and read the area data in the data, then the calculation result is output. The area should be positive, so the input data will be divided into two equivalence partitions: positive value and non-positive value. If the value is non-positive, the function shouldn't work.

Standard Deviation of the Density of Observations

This function will input the standard deviation of the observation. According to the formula of standard deviation, there should not be less than 2 pieces of data to calculate. We divide the input into three equivalence partitions: no-data file, 1 piece of data file and 2 pieces of data file. The boundary value is 2 pieces of data file. The non-data file and 1 piece of data file should return NAN. The file with 2 pieces of data will output normally.

Average Canopy Height

This function will input the average height. Because it is height, all the data should be positive. So the input data will be divided to negative, zero and positive. The boundary value is 1. The zero and negative should be invaild.

Data Quality

This function will input the average of the quality flags. According to GEDI data, flags here are 0 and 1. So we will divide the input data to 0-1 data and non-0-1 data. When the function meets non-0-1 data, it shouldn't work.

Clustering (Classify Forest Types)

We will be using clustering to sort and classify the various forest types found in a given area. Our goal is to implement three clustering algorithms: HDBScan, kmeans, and hierarchical. All three algorithms would be performed on the same data, and the best performing algorithm for that dataset would be chosen and

presented to the user. Since they will be run on the same data when our application is used, the algorithms will be tested the same way. The tests will still be run individually, but the methods and inputs chosen follow the exact same logic. Each of these algorithms will be implemented using clustering modules provided by the scikit-learn machine learning library. As a result, we can use the datasets provided in scikit-learn examples as our inputs. These 3 datasets will act as valid inputs for their respective algorithms only, because we want to use a guaranteed 100% correct input when testing validity.

Most clustering algorithms can technically cluster any dataset, even if the data does not have natural groupings. This makes it difficult to test most inputs, because there is usually not a clear “right/wrong” answer. In the case of these tests, we can use the adjusted Rand index function from the scikit-learn library. This function compares the similarity between cluster results and the “truth” values, which are both lists of labels that define how the clusters are laid out. If the rand index function returns a negative number or 0, then the results are considered “bad”. If the function returns a 1, then the results are perfect. The function can return a decimal between 0 and 1. Because we will be using preset datasets as the inputs, we will expect a perfect score of 1 from this function to consider the test as passed. In order to use this testing method, we will need to use the same datasets, which already have set truth values, and adjust the data in the set so that the clustering results will not end up with the expected values. There is no clear way to decide boundary values for these tests without testing all possibilities for the datasets, so the datasets will be adjusted semi-randomly. Since we are using the adjusted Rand index, there are only 2 ranges to consider for equivalence partitioning: generally “incorrect” results and “correct” results. With this function, there is no specific limit of what can and can’t be correct, so it is simplest to divide it clearly between 0 and 1.

Email Validation

The user will be entering an email address in order to receive their requested data back. We will have to make sure the email address they enter is valid. Thus, we will be implementing a test to ensure the user doesn’t enter anything that is unreasonable. Given that the user input is non-numerical, there are no boundary values, and we will be focusing on what inputs are considered valid.

To be a valid email address, there may not be any special characters within the address, i.e. !, #, $, etc. Additionally, the address must contain exactly one “at” symbol (@). There must be some amount of letters or numbers on the left hand side of the “at” symbol. To the right of the “at” symbol, there must be a valid domain and domain extension. The domain can only contain alphabetical characters. There should be a single period (.) separating the domain and the

valid domain extension. This domain extension can be .com, .net, .edu, etc. We will pass in strings that do and do not satisfy these conditions to determine whether or not they're valid. All of these criteria must be met for the test to pass.

Integration Testing

After unit testing is performed to determine if the individual components perform as intended, integration testing is used to test the interactions between the units. In other words, it tests whether different parts of the system pass the necessary information correctly between each other.

We split the modules which interact with each other into the following parts:

- User selection
- User email input
- Data retrieval
- Perform statistical functions
- Perform clustering
- Graph clustering results
- Generate statistical summary
- Email results

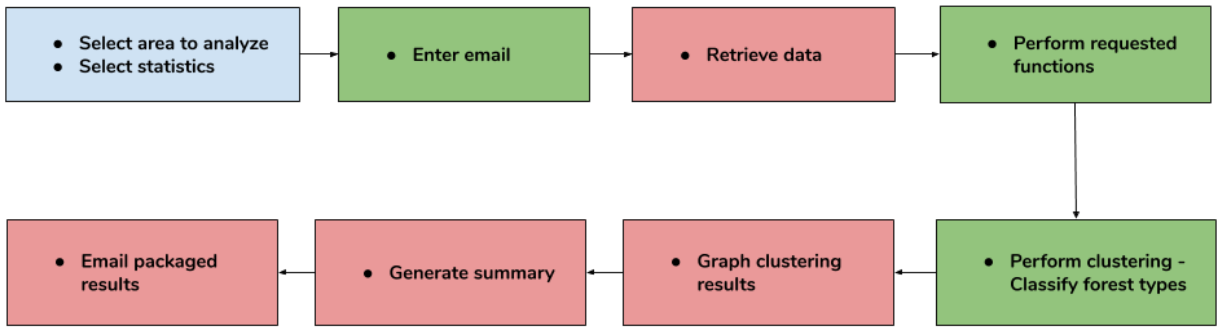


Diagram 1 - System workflow displaying the modules and how they are linked to other modules.

Diagram 1 shows that there are eight modules that integrate with each other, and seven of them require testing. The red colored boxes show the modules that will be tested with integration testing. The green colored boxes show the modules that are being tested with unit testing. Since these modules are already being tested, we will not discuss them in this section. The blue colored box shows the module that does not require testing. This module is based on predetermined inputs that the user selects without external variability, so we will not discuss them in this section, either.

Retrieve Data

Once the user submits their request, we will have to make sure they retrieve the correct information from the correct region. We will create tests that ensure that each country that is selected corresponds to the correct set of GEDI data. Similarly, we will create tests that ensure each statistical function that is selected corresponds to the correct mathematical computation. The final potential issue is when a user inputs their own shapefile. We will create tests to determine whether the data we pull from the database intersects the boundaries of the user's shapefile (as in, lies within the region specified by the user).

Graph Clustering Results

We will be outputting a graphical representation of the clustering algorithms performed. Given that the clustering algorithms are being unit tested, that dramatically increases the probability that the graphical output will also be correct. Additionally, automated testing is difficult with graphs, so we will be manually testing this module. We will be looking at the graphical outputs to determine whether any errors occurred. Any graphical misinformation should be glaringly obvious given that they are a direct result of the clustering algorithms.

Generate Statistical Summary

Similarly to graphing the results, generating the summary is a direct result of another module. In particular, it is a direct result of the statistical function operations. Since we will just be rewriting the results of the statistical functions into a .csv file, and since the statistical functions are being tested with unit testing, there is an unlikelihood for errors. Thus, we will be manually looking at the .csv file and comparing the results to the respective statistical functions to determine if there are any incorrect numerical values or any misplaced values.

Email packaged results

The user will be receiving via email either the graphical clustering results, the generated statistical summary, or both in a ZIP file. To test this, we will simply review the emails generated with all three possible options. We will be making sure the email is sent properly, and that any requested information is correctly attached. To further ensure the validity of the email sending process, we will extract the ZIP file and make sure that the included files contain correct and accurate information.

Usability Testing

If the integration testing was successful, then usability testing is next. Traditional usability testing has real people interacting with your application. There are variations on how they are observed and if/how the moderator asks questions (if there is one at all), but its general goal is to determine if the application is usable for a general audience. “Usable” would mean that the user can navigate the application efficiently and that the application is functional for them.

Our project is meant to be used by a wide variety of people interested in GEDI data, which may include both scientists and people who do not have a scientific background. If the website is poorly designed, then someone may choose the wrong area or the wrong function for analysis, which could result in a decision-maker choosing to cut down the wrong forest or someone’s reputation being ruined. Because of this, we should have a range of testers. The categories of testers would be: scientists familiar with GEDI data, scientists not familiar with GEDI data, and non-scientists. These categories will provide variance in tester experiences, which will help us determine all of the possible ways a user could interpret our application. Ideally, we would have 3 or more testers in each category to have more rounded testing and even more variance in experience.

To choose testers, we will mostly find people via networking. We will ask our capstone mentors and our sponsor, Dr. Jantz, for people that might fall under either of the categories related to scientists. Dr. Jantz will not be a tester in this scenario, as he helped us design the application and knows how it should be used.

Our testing will be loosely moderated, meaning that one of us from the team will present the application to a user, ask them to do a specific task, then ask them follow up questions. Due to recent events, it would be irresponsible to perform the testing in person unless the subject is part of the moderator’s household. The majority of subjects will not be household members, so the test will be conducted via Zoom and the subject will share their screen so the moderator can follow the cursor movements and take notes. The subject may ask any questions during the test, and the moderator will answer them and note the question. The moderator will also ask the subject why they asked the question.

Since our user flow is straightforward without much variance in paths, the difference in tasks is mainly through the area and the type(s) of analysis chosen. The follow up questions will be posed through a Google survey consisting of both closed and open-ended questions. The survey will be emailed to the subject immediately after the test, and the subject will be directed to fill it out right away. The questions included will be:

1. Are you a professional or student researcher?
    a. Professional
    b. Student
    c. Not a researcher
2. If you are a researcher, what is your background?
3. What do you know about GEDI?
4. Who could you see using the website?
5. How easy was it to navigate the website?
    a. Very easy
    b. A little easy
    c. A little hard
    d. Very hard
6. Why did you choose that answer for the last question?
7. Were you able to understand the results?
    a. Yes
    b. No
    c. Other:
8. Were you frustrated with anything while using the website? If so, what?
9. Any suggestions for us?

Moderator notes and survey answers will be considered manually, since the resulting data is largely qualitative. We plan to use the data to improve usability and reassess website layout if needed. While testing typically continues throughout the lifespan of an application, this usability testing portion will wrap up our testing plan and prepare us to present our final application before handing it off to our sponsor.

Conclusion

This project is to develop an application for better understanding forest structure and health. The application will help users automatically analyze the GEDI data and will reduce the required time. The result will also be easy to understand. To meet all of our client's requirements, we designed a software testing plan. Our software testing plan is to do the unit testing first, using PyTest to ensure that the functions of the **Statistical Functions**, **Clustering (classification forest types)**, and **Email Validation** are implemented as expected. We will use valid and invalid data to test to make sure the function will not output the wrong result. Then we will do the integration test, to make sure the eight modules can cooperate with each other. The final step is usability testing. We will select testers to use our products, and then let them do a questionnaire survey to judge whether our application is user-friendly. We will also let our customers use it to judge whether this can replace our client's previous workflow.

So far, we have completed a large portion of the functions our client expects our application to perform. We have also made corresponding test plans to ensure that the functions we have completed can run as we expect. After completing the test, our application can be considered complete. Next, we will continue to improve our project and strengthen documentation and communication with our client in order to pass on our project to him.