EECS4313 Software Engineering Testing
Project (100 points), Version 1
Building Defect Detection Models for Source Code Changes

Instructors: Song Wang
Release Date: March 12, 2020

Due: 11:59 PM, Monday, April 8, 2020

In this project, you are expected to build a defect prediction model for software source code changes from scratch. You are required to apply machine learning techniques, e.g., classification, sampling, feature engineering, etc., to build more accurate prediction models with the dataset provided.

Policies

- You are required to work as a team of 1 or 2 for this lab. Group members will get the same marks.
- There will be 5 bonus points for groups who can submit their work on time (i.e., **11:59 PM, Monday, April 8, 2020**), which will be distributed based on the contribution of each member in a group. If you work alone, you will get all the 5 bonus points.
- When you submit your work, you claim that it is solely the work of you or your group. Therefore, it is considered as an violation of academic integrity if you copy or share any parts of your code during any stages of your development.
- Your (submitted or un-submitted) solution to this assignment (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.
- Emailing your solutions to the instruction or TAs will not be acceptable.

Submitting Your Work

- create a folder named "EECS4313_Proj".
- put all the reports/result analysis from each part into one PDF file, named **EECS4313_proj_report.pdf**.
- create subfolders with names "**part_I**", "**part_II**", "**part_III**", "**part_IV**" and "**part_V**". Put the source code from each part into each folder, e.g., your code from `part I` should be in "**part_I**".
- create a subfolder name "**data**", put your experiment data in this folder.
- make sure the Instructor/TAs can run your code based on the above file structure without any manual modification.

```
zip -r EECS4313_Proj.zip EECS4313_Proj
submit 4313 proj EECS4313_Proj.zip
```

Background

Change-level defect prediction can predict whether a change is buggy at the time of the commit (a commit could contains many different changes) so that it allows developers to act on the prediction results as soon as a commit is made. In addition, since a change is typically smaller than a file, developers have much less code to examine in order to identify defects. However, for the same reason, it is more difficult to predict buggy changes accurately.

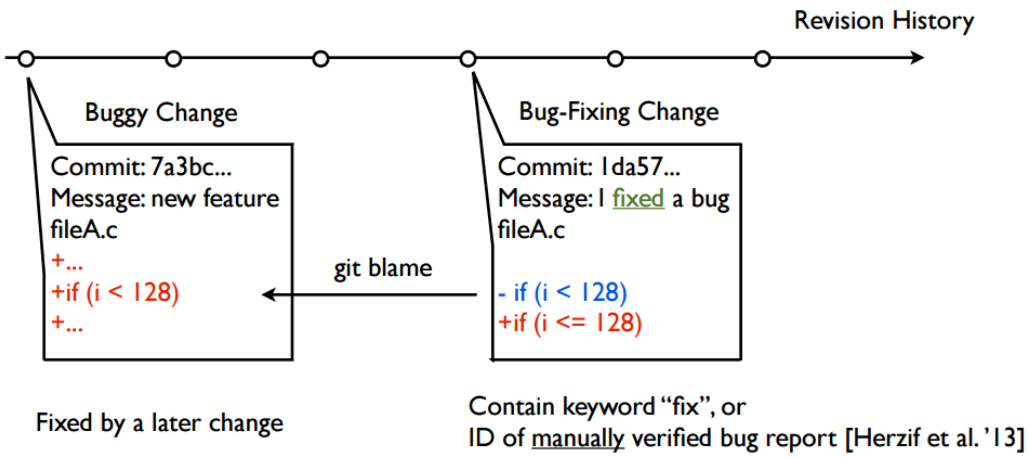

Figure 1: The process of labelling change-level data.

A typical change-level defect prediction process consists of the following steps:

- **Data labeling:** Labeling each change as buggy or clean to indicate whether the change contains bugs. Labeling change-level defect data requires to further link bug-fixing changes to bug-introducing changes. A line that is deleted or changed by a bug-fixing change is a faulty line, and the most recent change that introduced the faulty line is considered a bug-introducing change. We could identify the bug-introducing changes by a *blame* technique provided by a Version Control System (VCS), e.g., git. The bug-introducing changes are considered as buggy, other changes are labeled as clean. Figure 1 presents the process of collecting change-level data.
- **Feature extraction:** Extracting the features to represent the changes. Features are attributes extracted from a commit which describe the characteristics of the source code, such as LOC (line of code).
- **Model building/training:** Using the features and labels to build and train machine learning based prediction models. The widely used classifiers include Decision Tree, Naive Bayes, Supporting Vector Machine (SVM), Random Forest, and Logistic Regression, etc. All these machine learning algorithms are available in Scikit-learn[1].
- **Fault prediction:** with the well trained prediction models we can predict new changes as buggy or clean.

Features

This project provides a basic feature set, i.e., Metadata, to help you start this project.

These data are in **/data** folder.

Metadata: This feature set contains the commit time of a change, source code file/path names, and file age in days, the LOC (line of code) of the file that are involved, the added line count per change, the deleted line count per change, the changed line count per change, the added chunk count per change, the deleted chunk count per change, and the changed chunk count per change, etc.

[1] `https://scikit-learn.org/stable`

Evaluation Metrics

Metrics, i.e., *Precision*, *Recall*, and $F1$, are widely used to measure the performance of defect prediction models. Here is a brief introduction:

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{1}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{2}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

Precision and recall are composed of three numbers in terms of *true positive*, *false positive*, and *false negative*. True positive is the number of predicted defective files (or changes) that are truly defective, while false positive is the number of predicted defective ones that are actually not defective. False negative records the number of predicted non-defective files (of changes) that are actually defective. F1 is the weighted average of precision and recall.

In this project, you are expected to compare the performance of different defect prediction models by using F1.

(0)- Team Members and Their Contributions (0 points)

In the first part of your report, you are expected to list your team members and the contributions from each member. Please use percentage to quantify a team member's contribution. This part is **mandatory** in your report. If missing, **your group will not get the bonus points**.

(I)- Using Machine Learning Libraries to build a simple model (5 points)

In this project, you are excepted to build defect prediction models by using Python machine learning libraries. All the data we provided use the "csv" data format. The first line of each "csv" data file contains the id of a change, features to represent the change, and the label of the change. The provided features we provided looks like this:

```
change_id numeric // *Not a feature* This is the id of a change
401_lines_added numeric
402_lines_deleted numeric
403_lines_changed numeric
404_loc numeric
405_file_lines numeric
406_time_of_day numeric
407_day_of_week numeric
408_previous_patches numeric
409_previous_buggy_patches numeric
410_file_age numeric
411_commit_time,
412_full_path,
413_patch_lines numeric
414_lines_inserted numeric
415_lines_removed numeric
500_Buggy? {0,1} // *Not a feature* This is label, 0 means clean, 1 means buggy
```

The data of a change looks like the following:

```
change_id, 401_lines_added numeric,402_lines_deleted numeric,403_lines_changed numeric,404_loc
    numeric,405_file_lines numeric,406_time_of_day numeric,407_day_of_week numeric,408
    _previous_patches numeric,409_previous_buggy_patches numeric,410_file_age numeric,411_commit_time
    , 500_Buggy?412_full_path ,413_patch_lines numeric,414_lines_inserted numeric,415_lines_removed
    numeric, 500_Buggy?
9007,0,0,8,10,0,21,5,8,1,184,'2010-09-17 21:10:50+00:00',querycomponent.java,81,2,8,0
9016,0,5,10,20,0,21,5,10,5,184,'2010-09-17 21:10:50+00:00',solrindexsearcher.java,0,3,17,0
9002,1,0,4,6,1603,10,5,15,13,183,'2010-09-17 10:30:03+00:00',documentswriter.java,0,4,2,0
9011,0,0,2,2,0,21,5,9,1,184,'2010-09-17 21:33:48+00:00',querycomponent.java,0,1,1,0
9035,5,1,4,11,296,11,6,3,1,164,'2010-09-18 11:50:31+00:00',packedints.java,215,8,3,0
9046,28,1,13,47,84,11,6,1,1,164,'2010-09-18 11:50:31+00:00',packedreaderiterator.java,215,37,10,0
9355,6,8,807,874,0,19,6,10,3,191,'2010-09-25 19:32:37+00:00',fieldcacheimpl.java,3317,123,751,1
```

Your first task is to build a simple **Naive Bayes** based defect prediction models with all the defect prediction features provided. Specifically, for each of the projects, you are required to build and train the **Naive Bayes** based defect prediction models with the provided training and test datasets.

An example of using Scikit libs to build a **Naive Bayes** classifier for files (in a release) is available here: **example/nb_predictor.py**. We also provide example data to drive this bug prediction model, which are also in **example/**. Note that, the example file-level bug prediction model uses different features from this project.

There are multiple Naive Bayes models in Scikit (different models have different assumptions about the distribution of the data), we focus on Gaussian Naive Bayes: `https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes`.

More useful information about classifiers in the Scikit libs:

Scikit-learn: `https://scikit-learn.org/stable/supervised_learning.html#supervised-learning`

In you report of this part, you are expected to show the performance of **Naive Bayes** based classifiers on different projects by using figures or tables.

For the source code of this part, your code are expected to be executable. The results presented in your report are expected to be replicated by running your code.

(II)- Improving the model by using different ML algorithms (35 points)

Naive Bayes is a simple machine learning classifier, which may deliver poor prediction performance on the change-level defect prediction dataset. You are expected to try and find a better machine learning classifier. Note that, each classification algorithm may have several parameters that may affect its performance, you are also expected to tune the parameters for improving the performance of a specific classification algorithm. **You should at least examine four different classifiers**, e.g., Decision Tree (e.g., `DecisionTreeClassifier` in Scikit-learn), Supporting Vector Machine (SVM), Random Forest, and Logistic Regression, etc.

In your report of this part, you should show which parameters you have tuned, what are the values you have tried, how the performance of a classifier changes with different parameter values, and the best parameters for each classifier by using figures or tables.

For the source code of this part, your code are expected to be executable. The results presented in your report are expected to be replicated by running your code.

(III)- Improving bug prediction models with more features (35 points)

The initialized features only contain the basic statistically information of the changes and have not considered the context of every single change, which might be not enough to build an accurate prediction model. You are expected to design new features to improve the prediction models you built in the first question.

Some hints for designing new features:

Bag-of-Words: The Bag-of-Words feature set is a vector of the count of occurrences of each word in the text. You may want to remove the keywords from a specific program language, e.g., `int, double, boolean, class, public, void, private`, etc., as these tokens often are not related to program errors could be noises. Meanwhile, we keep the control flow related keywords, e.g, `if, else, for, while`, etc. After that, you can obtain the bag-of-words features from the source code.

More details of Bag-of-Words can be found here: `https://en.wikipedia.org/wiki/Bag-of-words_model`

You are expected to add at least the above features to your defect prediction models, you can also design and obtain other features. See details in `/data/description.pdf` about how to collect the above features.

In your report of this part, you should describe the new features you have collected. Detailed step how you extracted these features (**interacted with your code**). You are expected to rebuild and tune the **FIVE**

classifiers in Part (I) and (II). You are also expected to compare your new features with the original features we provided.

For the source code of this part, your code are expected to be executable. The new features and the prediction results presented in your report are expected to be replicated by running your code.

(IV)- Improving the training set by resampling (25 points)

As you may have noticed, the change-level defect prediction data is imbalanced, i.e., there are few buggy instances in training dataset. The imbalanced data can lead to poor prediction performance. You are expected to improve the imbalanced data issue by using resampling techniques.

There are many alternative resampling techniques that you can use, e.g., SMOTE, spread subsample, and resampling with/without replacement.

SMOTE: first selects instances from the minority class and finds k nearest neighbors for each instance, where k is a given number. It then creates new instances using the selected instances and their neighbors.

Spread Subsample: eliminates instances in the majority class until the ratio of majority instances over minority instances is equal to a given ratio. Weight is a property of the instance that reflects the level of impact this instance has on its class. Spread subsample also updates the weight of each instance in order to maintain the overall weight of the two classes.

Resampling with/without Replacement: randomly picks instances for either eliminating or duplicating until the buggy rate reaches a given value. The instances may be used multiple times for resampling with replacement whereas for resampling without replacement the instances are used only once.

Some of the resampling techniques are available here:

imbalanced-learn: `https://imbalanced-learn.readthedocs.io/en/stable/`

If you cannot find an available libraries, you should implement your own version.

You are expected to try at least three different resampling techniques (**can be different from the three listed above**) and find the best one. Note that, resampling techniques often have parameters that may affect their performance, you are also expected to tune the parameters for improving the performance of a specific resampling algorithm.

In your report of this part, you should describe which resampling techniques you have tried. You are expected to rebuild and tune the classifiers in Part (III) with different resampling techniques. You are also expected to compare your bug prediction models with and without adopting the resampling techniques.

For the source code of this part, your code are expected to be executable. The results presented in your report are expected to be replicated by running your code.

(V)- Further improvements (5 bonus points)

You are also highly encouraged to improve the defect prediction models by using other techniques beyond we recommended. For example, **using powerful deep learning classifiers or hyper parameter tuning etc.**

In your report of this part, you should describe which techniques you have tried and how much the applied techniques can improve the best classifiers from part (IV).

For the source code of this part, your code are expected to be executable. The results presented in your report are expected to be replicated by running your code.