Team:

Muhammad Khokhar (213765813): Part 3 and 4 along with bag of words (bow.py) program.

Kishan Patel (215028814): Part 1 and 2 along with the programs.

How to run the programs:

First, you must install the following packages (if not already installed):

- pandas
- sklearn
- python-docx (just to make it easy to copy the tables to the report, but the results are still output to the console, a file test.docx is included as well just to output those results in the file, please leave that file there)
- imbalanced-learn (requires numpy, scipy, sklearn)
- nltk (Natural Language Toolkit)
- keras
- tensorflow (required for keras)

After installing the above packages, you can run a specific script as follows:

- python <script-name> for example: python part1.py

For Part 4 when running the script you will be prompted to choose between three options: press "1" for SMOTE, "2" for ADASYN, and "3" for random under-sampler. When selecting option "3", you will then be prompted to select yes (press "y") or no (press "n") to sample with or without replacement respectively. The appropriate results will then be output to the console.

Part 1: A simple Naïve Bayes model

Using a very simples gaussian based Naïve Bayes classifier provided by sklearn, we ran the model on four different projects ('jackrabbit', 'jdt', 'lucene', and 'xorg') and obtained the average precision, recall, and f1-score of each project. From the results we obtained, we concluded that this simple Naïve Bayes does not perform that well on the change-level defect datasets provided. This can be seen in Table 1.

Datasets	Precision	Recall	F1-Score
jackrabbit	0.60	0.56	0.52
jdt	0.70	0.64	0.61
lucene	0.71	0.70	0.68
xorg	0.68	0.55	0.50
Average	**0.67**	**0.62**	**0.58**

Table 1: Average precision, recall, and f1-score across all the data sets

The values in the table above were obtained using the metrics defined in Part 2, that is by accumulating the true positives, true negatives, false positives, and false negatives across the folders of a project and then calculating the precision, recall, and f1-score. As can be seen above, the f1-score is 0.58 across all the projects which implies a relatively poor performance on the data. The best f1-score is achieved on the "lucene" project, a score of 0.68. In the next section, we will compare four different classifiers and find the best performer on the different projects.

Part 2: Improving the model by exploring different classifiers

In this section we experiment with many different classifiers to see which ones perform best (i.e., identify the most buggy sequences) on the datasets. The methodology used to evaluate the performance of a classifier is the F1-Score which is calculated by gathering all the false positives, false negatives, true positives, and true negatives across the folders of a project. Then:

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

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

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

The following classifiers were used to build our models: Decision Tree, Random Forests, Logistic Regression, and Support Vector Machine. With each classifier we tried various values for each parameter in order to determine the best settings that will achieve the highest F1-Score across all the projects. Each row in the tables below represents the F1-Score achieved across all folders of the labeled project, i.e., for decision tree classifier, a max depth of 1 achieves an F1-Score of 0.69 across all folders (0 to 5).

Decision Tree

For the Decision Tree classifier, the parameters we chose to tune are **Criterion**, **Splitter**, **Max Depth**, and **Max features**. Criterion refers to the function used to measure the quality of a split – "gini" for Gini impurity and "entropy" for information gain. Splitter (in the above table we refer to it as 'Split') is the strategy used to choose the split at each node – included are "best" for best split (split on the most relevant feature), and "random" (takes a random feature and splits it) [1]. In our tests we excluded the "random" split just because the results are very random each time the program is run. Max depth is simply the maximum depth of the tree – **None** in the above table means the nodes are expanded until all leaves are pure [2]. Max features is the number of features to consider when looking for the best split [2].

We kept Split = "best" the same on all runs as explained earlier and tried both Criterion = "entropy" and Criterion = "gini" on every run. For Max Depth we tried values 1,2,5,8,10,20,30,50,80, and 100 using both Criterion = "gini" and "entropy". The results for both "gini" and "entropy" can be seen in the tables below.

Using: Criterion = "gini"

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.89	0.56	0.69
2.0	0.97	0.55	0.7
5.0	0.9	0.57	0.7
8.0	0.87	0.57	0.69
10.0	0.83	0.58	0.68
20.0	0.74	0.55	0.63

30.0	0.74	0.55	0.63
50.0	0.74	0.55	0.63
80.0	0.74	0.55	0.63
100.0	0.74	0.55	0.63

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.99	0.82	0.9
5.0	0.97	0.83	0.89
8.0	0.94	0.83	0.88
10.0	0.92	0.84	0.88
20.0	0.9	0.84	0.87
30.0	0.9	0.84	0.87
50.0	0.9	0.84	0.87
80.0	0.9	0.84	0.87
100.0	0.9	0.84	0.87

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.98	0.77	0.86
2.0	0.99	0.78	0.87
5.0	0.95	0.78	0.86
8.0	0.89	0.79	0.84
10.0	0.86	0.79	0.83
20.0	0.86	0.79	0.83
30.0	0.86	0.79	0.83
50.0	0.86	0.79	0.83
80.0	0.86	0.79	0.83
100.0	0.86	0.79	0.83

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.82	0.9
2.0	0.99	0.82	0.9
5.0	0.97	0.82	0.89
8.0	0.94	0.83	0.88
10.0	0.89	0.83	0.86
20.0	0.88	0.83	0.86
30.0	0.88	0.83	0.86
50.0	0.88	0.83	0.86
80.0	0.88	0.83	0.86
100.0	0.88	0.83	0.86

Using: Criterion = “entropy”

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.93	0.55	0.69
2.0	0.93	0.56	0.7
5.0	0.85	0.6	0.7
8.0	0.73	0.6	0.66
10.0	0.74	0.6	0.66
20.0	0.69	0.59	0.63
30.0	0.7	0.59	0.64
50.0	0.7	0.59	0.64
80.0	0.7	0.59	0.64
100.0	0.7	0.59	0.64

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.99	0.82	0.9
5.0	0.97	0.83	0.89
8.0	0.94	0.83	0.88
10.0	0.94	0.84	0.88
20.0	0.93	0.84	0.88
30.0	0.93	0.84	0.88
50.0	0.93	0.84	0.88
80.0	0.93	0.84	0.88
100.0	0.93	0.84	0.88

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.98	0.77	0.86
2.0	0.97	0.78	0.86
5.0	0.95	0.78	0.86
8.0	0.95	0.79	0.86
10.0	0.93	0.79	0.85
20.0	0.92	0.79	0.85
30.0	0.92	0.79	0.85
50.0	0.92	0.79	0.85
80.0	0.92	0.79	0.85
100.0	0.92	0.79	0.85

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	1.0	0.82	0.9
5.0	0.97	0.82	0.89
8.0	0.95	0.83	0.89
10.0	0.93	0.83	0.88

20.0	0.92	0.83	0.87
30.0	0.92	0.83	0.87
50.0	0.92	0.83	0.87
80.0	0.92	0.83	0.87
100.0	0.92	0.83	0.87

Our results show that for both **Criterion = "gini"** and **Criterion = "entropy"**, at **max_depth > 5**, the F1-Scores of the different projects starts decreasing slightly. For both, the "jackrabbit" project sees the most significant drop in F1-Score after max_depth = 5, dropping all the way down to 0.63 and 0.64 for "gini" and "entropy" respectively. For the other projects, the F1-Score does not change much for different values of max_depth for both "gini" and "entropy". Thus, max_depth = 5 is chosen as the optimal parameter value since it achieves the best F1-Score across all the projects

Next, we must determine the optimal parameter value for max_features. The data for this is presented below.

Using: Criterion = "gini" and max_depth = 5 to find best value for max_features.

jackrabbit

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.93	0.55	0.69
2.0	0.85	0.56	0.68
3.0	0.91	0.56	0.7
4.0	0.87	0.56	0.68
5.0	0.83	0.6	0.69
6.0	0.9	0.57	0.7
7.0	0.88	0.58	0.7
8.0	0.88	0.57	0.69
9.0	0.88	0.58	0.7
10.0	0.88	0.57	0.69
11.0	0.85	0.57	0.68
12.0	0.91	0.57	0.7
13.0	0.9	0.57	0.7

jdt

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.82	0.9
2.0	0.97	0.83	0.89
3.0	0.97	0.82	0.89
4.0	0.97	0.83	0.89
5.0	0.97	0.82	0.89
6.0	0.97	0.83	0.89
7.0	0.97	0.82	0.89
8.0	0.97	0.82	0.89
9.0	0.97	0.83	0.89
10.0	0.97	0.83	0.89
11.0	0.97	0.83	0.89
12.0	0.96	0.83	0.89

13.0	0.97	0.83	0.89

lucene

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.9	0.76	0.82
2.0	0.95	0.78	0.85
3.0	0.95	0.78	0.86
4.0	0.95	0.78	0.86
5.0	0.98	0.78	0.87
6.0	0.96	0.78	0.86
7.0	0.98	0.78	0.87
8.0	0.95	0.78	0.86
9.0	0.9	0.77	0.83
10.0	0.92	0.78	0.84
11.0	0.93	0.78	0.85
12.0	0.93	0.78	0.85
13.0	0.95	0.78	0.86

xorg

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.98	0.81	0.89
2.0	0.98	0.82	0.89
3.0	0.94	0.82	0.88
4.0	0.95	0.83	0.88
5.0	0.67	0.78	0.72
6.0	0.95	0.82	0.88
7.0	0.95	0.82	0.88
8.0	0.97	0.82	0.89
9.0	0.96	0.82	0.88
10.0	0.67	0.78	0.72
11.0	0.96	0.82	0.88
12.0	0.97	0.82	0.89
13.0	0.97	0.82	0.89

Determining the optimal value for max_features was quite difficult because the F1-Scores of the various projects for different values of max_features varies quite a lot. For example, results from the “xorg” project show that for max_features = 1 we get an F1-Score of 0.89 but then at max_features = 5 we get an F1-Score of 0.72 – the main problem being, there is no defined pattern observed for different values of max_features. Further analysis revealed however, that the best F1-Scores come from a max_features value of 13 and this can be seen in the above tables for all the projects. For “jackrabbit” we observe an F1-Score of 0.70 at max_features = 13 and for “lucene” an F1-Score of 0.86 – quite the increase from 0.82 at max_features = 1.

Thus, the best parameter settings for the Decision Tree classifier are Criterion = “gini”, splitter = “best”, max_depth = 5, and max_features = 13.

Random Forests

For Random Forests classifier we tuned the parameters **Criterion**, **n_estimators**, **Max features**, and **Max depth**. Criterion is described the same as in Decision Tree classifier above and n_estimators refers to the number of trees you want to include in the forest, typically the larger the better. Max features and Max depth are also described similarly as above.

Using: Criterion = gini

Jackrabbit

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.72	0.58	0.64
2.0	0.9	0.56	0.69
5.0	0.84	0.59	0.69
8.0	0.9	0.57	0.7
10.0	0.89	0.57	0.69
20.0	0.9	0.57	0.7
30.0	0.89	0.58	0.7
50.0	0.9	0.58	0.7
80.0	0.89	0.58	0.7
100.0	0.89	0.58	0.7

jdt

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.91	0.84	0.87
2.0	0.97	0.82	0.89
5.0	0.96	0.83	0.89
8.0	0.97	0.83	0.89
10.0	0.98	0.83	0.9
20.0	0.98	0.83	0.9
30.0	0.98	0.83	0.9
50.0	0.98	0.83	0.9
80.0	0.98	0.83	0.9
100.0	0.98	0.83	0.9

Lucene

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.89	0.79	0.83
2.0	0.95	0.78	0.86
5.0	0.92	0.78	0.84
8.0	0.89	0.77	0.83
10.0	0.89	0.77	0.82
20.0	0.94	0.78	0.85
30.0	0.95	0.78	0.85

50.0	0.94	0.78	0.85
80.0	0.93	0.78	0.85
100.0	0.93	0.78	0.85

xorg

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.89	0.83	0.86
2.0	0.98	0.82	0.89
5.0	0.94	0.82	0.88
8.0	0.97	0.82	0.89
10.0	0.97	0.82	0.89
20.0	0.98	0.82	0.89
30.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89

Using: Criterion = “entropy”

jackrabbit

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.7	0.59	0.64
2.0	0.86	0.58	0.69
5.0	0.78	0.59	0.68
8.0	0.87	0.59	0.7
10.0	0.87	0.59	0.7
20.0	0.88	0.59	0.71
30.0	0.88	0.59	0.71
50.0	0.88	0.6	0.71
80.0	0.88	0.6	0.71
100.0	0.87	0.59	0.71

jdt

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.92	0.84	0.88
2.0	0.98	0.82	0.89
5.0	0.97	0.83	0.89
8.0	0.98	0.82	0.89
10.0	0.98	0.82	0.9
20.0	0.99	0.82	0.9
30.0	0.99	0.82	0.9
50.0	0.99	0.82	0.9
80.0	0.99	0.82	0.9
100.0	0.99	0.82	0.9

Lucene

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.91	0.79	0.85
2.0	0.97	0.78	0.86
5.0	0.95	0.79	0.86
8.0	0.97	0.78	0.86
10.0	0.97	0.78	0.87
20.0	0.97	0.78	0.86
30.0	0.96	0.78	0.86
50.0	0.98	0.78	0.87
80.0	0.97	0.78	0.87
100.0	0.97	0.78	0.86

xorg

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.87	0.82	0.85
2.0	0.98	0.82	0.89
5.0	0.96	0.82	0.89
8.0	0.98	0.82	0.89
10.0	0.98	0.82	0.89
20.0	0.98	0.82	0.89
30.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89

Based on our analysis we deduced that for Random Forests there is very little difference in performance when choosing **entropy** or **gini** as seen in the above tables. However, for both "entropy" and "gini", the F1-Scores across the different projects seems to increase as we increase the value of **n_estimators** and begins to stabilize at around **n_estimators >= 20**. In particular, looking at the "lucene" project under Criterion = "gini", only for **n_estimators > 10** does the F1-Score begin to stabilize and thus we choose **Criterion = "gini"** and **n_estimators = 20** as the optimal selections at this stage.

Now, using **Criterion = "gini"** and **n_estimators = 20** we try to determine the best value for **max_depth**.

Using: Criterion = "gini" and n_estimators = 20

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.93	0.55	0.7
2.0	0.97	0.55	0.7
5.0	0.93	0.57	0.71
8.0	0.9	0.58	0.7
10.0	0.89	0.58	0.7
20.0	0.9	0.57	0.7
30.0	0.9	0.57	0.7

50.0	0.9	0.57	0.7
80.0	0.9	0.57	0.7
100.0	0.9	0.57	0.7

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	1.0	0.82	0.9
5.0	0.99	0.82	0.9
8.0	0.98	0.83	0.9
10.0	0.98	0.83	0.9
20.0	0.98	0.83	0.9
30.0	0.98	0.83	0.9
50.0	0.98	0.83	0.9
80.0	0.98	0.83	0.9
100.0	0.98	0.83	0.9

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.77	0.87
2.0	0.99	0.78	0.87
5.0	0.96	0.78	0.86
8.0	0.95	0.78	0.85
10.0	0.94	0.78	0.85
20.0	0.94	0.78	0.85
30.0	0.94	0.78	0.85
50.0	0.94	0.78	0.85
80.0	0.94	0.78	0.85
100.0	0.94	0.78	0.85

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	1.0	0.81	0.9
5.0	0.98	0.81	0.89
8.0	0.97	0.82	0.89
10.0	0.98	0.82	0.89
20.0	0.98	0.82	0.89
30.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89

The data collected on max_depth is quite interesting because it seems that increasing the values does little to impact the performance of the different projects – in particular, the F1-Scores are quite stable for

different values of max_depth. Consider project “jdt” above for example, the F1-Score is 0.90 across all the values of max_depth, suggesting that for this particular dataset, max_depth may not be such an important parameter after all. Thus, we choose max_depth = 10 (although any value seems to be fine) as the optimal parameter value.

Next, we must determine what number of features will achieve the best F1-Scores across the different projects.

Using: Criterion = “gini”, n_estimators = 20, and max_depth = 10 to find max features.

jackrabbit

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.94	0.57	0.71
2.0	0.92	0.58	0.71
3.0	0.91	0.59	0.72
4.0	0.9	0.59	0.71
5.0	0.9	0.58	0.7
6.0	0.88	0.6	0.72
7.0	0.91	0.58	0.71
8.0	0.9	0.58	0.7
9.0	0.9	0.58	0.7
10.0	0.9	0.58	0.71
11.0	0.89	0.58	0.7
12.0	0.9	0.57	0.7
13.0	0.89	0.58	0.7

jdt

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.82	0.9
2.0	0.99	0.82	0.9
3.0	0.99	0.82	0.9
4.0	0.99	0.82	0.9
5.0	0.98	0.82	0.9
6.0	0.98	0.83	0.9
7.0	0.98	0.83	0.9
8.0	0.98	0.83	0.9
9.0	0.98	0.83	0.9
10.0	0.98	0.83	0.9
11.0	0.98	0.83	0.9
12.0	0.98	0.83	0.9
13.0	0.98	0.83	0.9

lucene

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.78	0.87
2.0	0.98	0.78	0.87
3.0	0.99	0.78	0.87
4.0	0.98	0.78	0.87

5.0	0.98	0.78	0.87
6.0	0.97	0.78	0.87
7.0	0.95	0.78	0.85
8.0	0.94	0.78	0.85
9.0	0.95	0.78	0.86
10.0	0.95	0.78	0.85
11.0	0.88	0.77	0.82
12.0	0.94	0.78	0.85
13.0	0.94	0.78	0.85

xorg

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.98	0.81	0.89
3.0	0.99	0.82	0.89
4.0	0.98	0.82	0.89
5.0	0.98	0.82	0.89
6.0	0.98	0.82	0.89
7.0	0.97	0.82	0.89
8.0	0.98	0.82	0.89
9.0	0.98	0.82	0.89
10.0	0.98	0.82	0.89
11.0	0.97	0.82	0.89
12.0	0.97	0.82	0.89
13.0	0.98	0.82	0.89

We determined based on the above results that the best parameter value for **max_features** is anywhere between 1 and 6. This decision was based on the results of the "lucene" project where we observed that for max_features > 6, the F1-Score beings to decrease, down to 0.85 at max_features = 13. This perhaps suggests that not all features are relevant to the Random Forests classifier. For the "jackrabbit", "jdt", and "xorg" projects, the F1-Scores remain mostly the same for the different values of max_features. Thus, we select max_features = 6 as the optimal parameter value. The best parameters for this classifier are then Criterion = "gini", n_estimators = 20, max_depth = 10, and max_features = 6.

Logistic Regression

Next, we look at the logistic regression classifier. The only parameter chosen for tuning was the **C** value which is a regularization parameter used in the different loss functions available. For the loss function we chose the default "l2" loss and the **solver** we decided on was "liblinear" because it converges very fast for smaller data sets. As can be seen from the tables below, the differing values for C have little affect on the performance on this particular data set. By analyzing the F1-Scores obtained on the "lucene" data set, we find the best C value to be 0.0001 as for C > 0.0001, the F1-Scores begin to decrease all the way down to 0.84 for C = 1000. Thus, the optimal value for C was determined to be 0.0001.

Using solver = "liblinear" and penalty = "l2"

jackrabbit

C	PRECISION	RECALL	F1-SCORE

0.0001	0.88	0.58	0.7
0.001	0.89	0.58	0.7
0.01	0.89	0.58	0.7
0.1	0.88	0.58	0.7
1.0	0.88	0.58	0.7
10.0	0.88	0.58	0.7
50.0	0.88	0.58	0.7
100.0	0.87	0.58	0.7
500.0	0.89	0.58	0.7
1000.0	0.88	0.58	0.7

jdt

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.82	0.9
0.001	0.99	0.82	0.9
0.01	0.99	0.82	0.9
0.1	0.99	0.82	0.9
1.0	0.99	0.82	0.9
10.0	0.99	0.82	0.9
50.0	0.99	0.82	0.9
100.0	0.99	0.82	0.9
500.0	0.99	0.82	0.9
1000.0	0.99	0.82	0.9

lucene

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.77	0.87
0.001	0.97	0.77	0.86
0.01	0.97	0.78	0.86
0.1	0.94	0.77	0.85
1.0	0.93	0.77	0.84
10.0	0.93	0.77	0.84
50.0	0.93	0.77	0.84
100.0	0.93	0.77	0.84
500.0	0.93	0.77	0.84
1000.0	0.93	0.77	0.84

xorg

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.81	0.89
0.001	0.99	0.81	0.89
0.01	0.99	0.82	0.89
0.1	0.99	0.82	0.89
1.0	0.99	0.82	0.89
10.0	0.99	0.82	0.89
50.0	0.99	0.82	0.89

100.0	0.99	0.82	0.89
500.0	0.98	0.82	0.89
1000.0	0.99	0.82	0.89

The best parameters for Logistic Regression are then solver = “liblinear”, penalty = “l2” and C = 0.0001 for the reasons described above.

Support Vector Machine

The next classifier we try on the data set is the Support Vector Machine. The SVM is a classification algorithm that works well when the data is linearly separable. When working with SVMs, the main parameter usually chosen for tuning is the **C** parameter. Like in Logistic Regression, the C value acts as a regularization parameter which tells the SVM algorithm how much to avoid misclassifying a training example. In the tables below we try various values for C across the different projects.

jackrabbit

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.54	0.7
0.001	1.0	0.54	0.7
0.01	1.0	0.54	0.7
0.1	0.97	0.54	0.69
1.0	0.97	0.54	0.69
10.0	0.92	0.56	0.69
50.0	0.92	0.56	0.69
100.0	0.92	0.56	0.69
1000.0	0.88	0.57	0.69
10000.0	0.84	0.58	0.68

jdt

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.81	0.9
0.001	1.0	0.81	0.9
0.01	1.0	0.81	0.9
0.1	1.0	0.81	0.9
1.0	1.0	0.81	0.9
10.0	1.0	0.81	0.9
50.0	1.0	0.81	0.9
100.0	1.0	0.82	0.9
1000.0	0.99	0.82	0.9
10000.0	0.97	0.83	0.89

lucene

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.77	0.87
0.001	1.0	0.77	0.87
0.01	1.0	0.77	0.87
0.1	1.0	0.77	0.87

1.0	1.0	0.77	0.87
10.0	1.0	0.77	0.87
50.0	0.98	0.78	0.87
100.0	0.93	0.77	0.84
1000.0	0.92	0.77	0.84
10000.0	0.97	0.78	0.87

xorg

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.81	0.9
0.001	1.0	0.81	0.9
0.01	1.0	0.81	0.9
0.1	1.0	0.81	0.9
1.0	0.99	0.81	0.89
10.0	0.99	0.81	0.89
50.0	0.99	0.81	0.89
100.0	0.99	0.81	0.89
1000.0	0.98	0.81	0.89
10000.0	0.96	0.81	0.88

From data present in the above tables, notice how the F1-Score does not change much with different values of C but does start to decrease after a certain value such as C = 0.1 for the "xorg" project. Since the best F1-Scores are achieved with a C value of 0.0001 across all the projects, we determine it to be the optimal value.

Our analysis showed that a well tuned Random Forests classifier is the best performer on this bug prediction dataset simply because of its robustness and ability to handle large datasets very well in addition to its powerful capability to grow *n* number of trees.

Part 3: Improving our models by adding more features

In the previous section our results were quite good with each classifier, but even with powerful classifiers like Random Forests and Logistic Regression, we were not able to achieve high F1-Scores on the "jackrabbit" project in particular and other areas where improvements can be made. Considering this, we adapt a bag-of-words approach to generate new features for our training and testing data sets in hope of finding further improvements in our models.

The process used to extract the bag-of-words features is as follows:

1. We define a function *load_file* which will extract the lines of code from the patch files where changes have been made. Changes are indicated in the file with a "+" or "-" and we are only interested in these specific lines.
2. Once we obtain the lines of code for each patch file where a change has been made, we pass this new text to the *clean_file* function which will remove stop words from the text which in this case are unnecessary Java keywords (int, double, Boolean, class, public, etc.) which add noise to our analysis. The stop words are removed with the help of the NLTK (Natural Language Toolkit) python library. *Clean_file* returns a tokenized list of words in the patch file.

3. Step (1) and (2) are encompassed in the function *process_docs* which reads in the original training and testing files for each project and matches the "change_id" with the patch number to make sure we are able to process the files in the order they appear within the training and testing datasets. *Process_docs* calls another helper function *doc_to_line* which takes as input the file name (i.e., jackrabbit-35621.patch) and a set of words which we would like to search for within each file (we look for the Java keywords "if", "else", "for", "while", and "switch"). This information is passed to *clean_file* in (2) which returns us a tokenized list of the occurrences of the Java control flow related keywords. *Process_docs* is applied to each projects training and testing datasets.
4. We then utilize the Keras machine learning library which allows us to generate a matrix (numpy array) of the counts of the control flow keywords outputted from step (3), where column 0 is just an index, column 1 is count of "if" statements in the file, column 2 is the count of "for" statements, column 3 is the count of "else" statements, column 4 is the count of "while" statements, and column 5 is the count of "switch" statements. Note: we only collected the frequent tokens (i.e., those whose frequency is larger than 3).
5. Next, we read the training and testing files of each project and store this information in a unique variable for each file.
6. Then, using a function *add_to_csv*, we append the data from the matrix in (4) to new columns which we define in the original training and testing datasets of each project. We define these columns using the pandas library and are named "num_if", "num_for", "num_else", "num_while", and "num_switch". Each row will then have the counts of the control flow keyword for that "change_id" which matches the patch file it came from.
7. Finally, we output these modified training and testing files which include the new features as "train_bow.csv" and "test_bow.csv" and are included in each projects split folders.

We will now retune the classifiers from Part 2 using these new training and testing datasets we have compiled.

Naïve Bayes (GaussianNB)

Datasets	Precision	Recall	F1-Score
jackrabbit	0.53	0.58	0.55
jdt	0.64	0.86	0.73
lucene	0.85	0.80	0.83
xorg	0.52	0.86	0.65

The above results for GaussianNB show that the new features had a significant impact on the F1-Scores of the four projects. The "jackrabbit" project now has an F1-Score of 0.55 compared to 0.52 in part (1). "jdt" now has an F1-Score of 0.73 compared to 0.61 before, "lucene" has an F1-Score of 0.83 compared to 0.68 before, and "xorg" has an F1-Score of 0.65 compared to 0.58 before.

Decision Tree

Again, we are going to tune our decision tree using the criterion, max_depth, and max_features parameters.

Using Gini

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.89	0.56	0.69
2.0	0.97	0.55	0.7
5.0	0.88	0.57	0.7
8.0	0.84	0.57	0.68
10.0	0.79	0.57	0.66
20.0	0.77	0.57	0.65
30.0	0.77	0.57	0.65
50.0	0.77	0.57	0.65
80.0	0.77	0.57	0.65
100.0	0.77	0.57	0.65

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.99	0.82	0.9
5.0	0.96	0.83	0.89
8.0	0.95	0.83	0.89
10.0	0.91	0.84	0.87
20.0	0.89	0.84	0.86
30.0	0.89	0.84	0.86
50.0	0.89	0.84	0.86
80.0	0.89	0.84	0.86
100.0	0.89	0.84	0.86

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.98	0.77	0.86
2.0	0.99	0.78	0.87
5.0	0.96	0.78	0.86
8.0	0.85	0.78	0.81
10.0	0.87	0.79	0.83
20.0	0.85	0.78	0.82
30.0	0.85	0.78	0.82
50.0	0.85	0.78	0.82
80.0	0.85	0.78	0.82
100.0	0.85	0.78	0.82

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.82	0.9
2.0	0.99	0.82	0.9
5.0	0.96	0.82	0.88
8.0	0.93	0.83	0.88
10.0	0.92	0.83	0.88
20.0	0.91	0.84	0.87

30.0	0.91	0.84	0.87
50.0	0.91	0.84	0.87
80.0	0.91	0.84	0.87
100.0	0.91	0.84	0.87

Using Entropy

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.93	0.55	0.69
2.0	0.93	0.56	0.7
5.0	0.85	0.6	0.7
8.0	0.74	0.6	0.67
10.0	0.75	0.6	0.66
20.0	0.7	0.59	0.64
30.0	0.7	0.59	0.64
50.0	0.7	0.59	0.64
80.0	0.7	0.59	0.64
100.0	0.7	0.59	0.64

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.99	0.82	0.9
5.0	0.97	0.83	0.89
8.0	0.94	0.83	0.88
10.0	0.94	0.84	0.88
20.0	0.93	0.84	0.88
30.0	0.93	0.84	0.88
50.0	0.93	0.84	0.88
80.0	0.93	0.84	0.88
100.0	0.93	0.84	0.88

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.98	0.77	0.86
2.0	0.97	0.78	0.86
5.0	0.95	0.78	0.85
8.0	0.89	0.78	0.83
10.0	0.93	0.79	0.85
20.0	0.88	0.79	0.83
30.0	0.88	0.79	0.83
50.0	0.88	0.79	0.83
80.0	0.88	0.79	0.83
100.0	0.88	0.79	0.83

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	1.0	0.82	0.9
5.0	0.97	0.82	0.89
8.0	0.92	0.83	0.87
10.0	0.91	0.83	0.87
20.0	0.92	0.84	0.87
30.0	0.92	0.84	0.87
50.0	0.92	0.84	0.87
80.0	0.92	0.84	0.87
100.0	0.92	0.84	0.87

Like before with the original datasets, the F1-Scores begin to decrease after a certain max_depth value, in this case that value is 5 and this result is consistent across all the projects with slight variations seen only in the “lucene” project where the F1-score increases slightly at max_depth = 10 for both the “entropy” and “gini” criteria. Also like before, the results for “gini” and “entropy” are very similar, so performance is not affected by this parameter but because the max_depth starts to decrease after max_depth = 5, we choose this value as the best setting. The results of the “jackrabbit” project show that for a max_depth value between 10 and 100, we get slightly better F1-Scores on average (0.65) versus before (0.63). This means the new features are giving us a slight increase in predictive performance, specifically of the buggy class.

Using Criterion = 'gini' and max_depth = 5 to find max features

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.91	0.56	0.7
3.0	0.92	0.58	0.71
5.0	0.81	0.59	0.68
8.0	0.92	0.57	0.7
10.0	0.84	0.6	0.7
12.0	0.91	0.58	0.71
13.0	0.89	0.58	0.7
14.0	0.89	0.57	0.69
15.0	0.88	0.57	0.7
16.0	0.89	0.57	0.7

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.82	0.89
3.0	0.98	0.82	0.89
5.0	0.98	0.83	0.9
8.0	0.97	0.83	0.89
10.0	0.97	0.83	0.89
12.0	0.97	0.83	0.89
13.0	0.97	0.83	0.89
14.0	0.96	0.83	0.89
15.0	0.95	0.83	0.89
16.0	0.96	0.83	0.89

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.95	0.77	0.85
3.0	0.99	0.78	0.87
5.0	0.91	0.77	0.83
8.0	0.97	0.78	0.86
10.0	0.94	0.78	0.85
12.0	0.94	0.78	0.85
13.0	0.94	0.78	0.85
14.0	0.93	0.78	0.85
15.0	0.95	0.78	0.86
16.0	0.95	0.78	0.86

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.81	0.89
3.0	0.93	0.82	0.87
5.0	0.97	0.82	0.89
8.0	0.95	0.82	0.88
10.0	0.95	0.82	0.88
12.0	0.96	0.82	0.88
13.0	0.95	0.82	0.88
14.0	0.96	0.82	0.89
15.0	0.97	0.82	0.89
16.0	0.96	0.82	0.89

When tuning the max_feat parameter with the new features, we notice there is a slight increase in performance across all the projects. Looking at the “xorg” project in particular, we notice that the F1-Score does not fluctuate as much with different values for max_feat as compared to before where the score would drop to 0.72 at max_feat of 5 and 10 but now the result is much more consistent at an F1-Score of 0.88. Since the overall results are quite consistent across all the projects, the specific value of max_feat chosen does not matter but we decide on the value 16, as this considers all the features in our new dataset. Therefore, the best parameter settings for the decision tree classifier are criterion=”gini”, max_depth=5, and max_feat=16.

Random Forests

Here we will be returning the criterion, n_estimators, max_depth, and max_feat parameters of the random forests classifier provided by scikit-learn.

Using Gini

jackrabbit

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.69	0.56	0.62
2.0	0.85	0.53	0.66
5.0	0.82	0.58	0.68
8.0	0.91	0.57	0.7
10.0	0.91	0.58	0.7

20.0	0.91	0.58	0.71
30.0	0.91	0.58	0.71
50.0	0.91	0.58	0.71
80.0	0.9	0.58	0.71
100.0	0.9	0.58	0.71

jdt

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.91	0.84	0.88
2.0	0.98	0.82	0.89
5.0	0.95	0.83	0.89
8.0	0.98	0.83	0.9
10.0	0.98	0.83	0.9
20.0	0.98	0.83	0.9
30.0	0.98	0.82	0.9
50.0	0.98	0.83	0.9
80.0	0.98	0.82	0.9
100.0	0.98	0.83	0.9

lucene

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.88	0.79	0.83
2.0	0.95	0.78	0.86
5.0	0.92	0.79	0.85
8.0	0.95	0.78	0.86
10.0	0.94	0.78	0.85
20.0	0.94	0.78	0.85
30.0	0.95	0.78	0.86
50.0	0.94	0.78	0.85
80.0	0.93	0.78	0.85
100.0	0.93	0.78	0.84

xorg

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.89	0.83	0.85
2.0	0.98	0.82	0.89
5.0	0.95	0.82	0.88
8.0	0.97	0.82	0.89
10.0	0.97	0.82	0.89
20.0	0.97	0.82	0.89
30.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.97	0.82	0.89

Using Entropy

jackrabbit

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.74	0.61	0.67
2.0	0.89	0.59	0.71
5.0	0.82	0.6	0.69
8.0	0.88	0.59	0.7
10.0	0.88	0.59	0.71
20.0	0.88	0.59	0.71
30.0	0.88	0.59	0.71
50.0	0.89	0.6	0.71
80.0	0.88	0.6	0.71
100.0	0.88	0.6	0.71

jdt

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.92	0.84	0.88
2.0	0.98	0.82	0.89
5.0	0.97	0.83	0.9
8.0	0.98	0.83	0.9
10.0	0.99	0.82	0.9
20.0	0.99	0.82	0.9
30.0	0.99	0.82	0.9
50.0	0.99	0.82	0.9
80.0	0.99	0.82	0.9
100.0	0.99	0.82	0.9

lucene

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.86	0.78	0.82
2.0	0.96	0.78	0.86
5.0	0.94	0.78	0.85
8.0	0.96	0.78	0.86
10.0	0.97	0.78	0.86
20.0	0.97	0.78	0.86
30.0	0.97	0.78	0.86
50.0	0.98	0.78	0.87
80.0	0.97	0.78	0.87
100.0	0.97	0.78	0.86

xorg

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.91	0.83	0.87
2.0	0.97	0.82	0.89
5.0	0.96	0.82	0.89
8.0	0.98	0.82	0.89
10.0	0.98	0.82	0.89

20.0	0.97	0.82	0.89
30.0	0.97	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89

To see whether the new features had a positive impact on the performance of random forests, we compared the results of the "jackrabbit" project before and after the new features were added. Under the "gini" criterion, before adding the new features, we see that the highest F1-Score (0.70) is achieved with n_estimators = 20 and after adding the new features the F1-Score increased slightly, to 0.71. Like before, however, the F1-Score increases as we increase the n_estimators value. The more trees in the forest, the higher the F1-Score, which agrees with the documentation. We also see an improvement with the "entropy" criterion where the minimum F1-Score increased to 0.67 (with n_estimators = 1) whereas before this was 0.64 (with n_estimators = 1). The maximum is the same as with the "gini" criterion. This same result for "entropy" is observed in the "xorg" project as well, where we see the minimum F1-Score has increased to 0.87, whereas before it was 0.85. Therefore, we decide to choose the "entropy" parameter and an n_estimators value of 80 as we do not want to overfit our data with higher values and with lower values we may not learn enough from our training data.

Using criterion = "entropy" and n_estimators = 80 to tune max_depth.

jackrabbit

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.97	0.54	0.7
2.0	0.93	0.57	0.7
5.0	0.92	0.59	0.72
8.0	0.89	0.59	0.71
10.0	0.89	0.6	0.71
20.0	0.88	0.6	0.71
30.0	0.88	0.6	0.71
50.0	0.88	0.6	0.71
80.0	0.88	0.6	0.71
100.0	0.88	0.6	0.71

jdt

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	0.99	0.82	0.9
5.0	0.99	0.82	0.9
8.0	0.99	0.82	0.9
10.0	0.99	0.82	0.9
20.0	0.99	0.82	0.9
30.0	0.99	0.82	0.9
50.0	0.99	0.82	0.9
80.0	0.99	0.82	0.9
100.0	0.99	0.82	0.9

lucene

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.77	0.87
2.0	0.99	0.78	0.87
5.0	0.99	0.78	0.87
8.0	0.98	0.78	0.87
10.0	0.98	0.78	0.87
20.0	0.97	0.78	0.87
30.0	0.97	0.78	0.87
50.0	0.97	0.78	0.87
80.0	0.97	0.78	0.87
100.0	0.97	0.78	0.87

xorg

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9
2.0	1.0	0.81	0.9
5.0	0.99	0.81	0.89
8.0	0.98	0.82	0.89
10.0	0.98	0.82	0.89
20.0	0.98	0.82	0.89
30.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
80.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89

From the above results we can see that a max_depth of 5 gives the best F1-Score on the “jackrabbit” project whereas for the others, the score seems to remain the same for all values of max_depth. Compared to before there is slight improvement in both the “jackrabbit” and “lucene” projects. In “lucene”, before we saw an F1-Score of 0.85 for max_depth from 8 to 100 and now with the new features we see an F1-Score of 0.87 for the same values. We decide that a max_depth of 30 (realistically any value could work here because the results are very consistent for all values of max_depth across the different projects) to be optimal for random forests. Now we will tune the max_feat parameter.

Using criterion = “entropy”, n_estimators = 80 and max_depth = 30 to tune max_feat.

jackrabbit

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.96	0.57	0.71
2.0	0.95	0.58	0.72
3.0	0.93	0.58	0.72
4.0	0.92	0.59	0.72
5.0	0.92	0.6	0.73

6.0	0.91	0.6	0.72
7.0	0.91	0.6	0.72
8.0	0.91	0.61	0.73
9.0	0.91	0.6	0.72
10.0	0.9	0.61	0.72
11.0	0.9	0.6	0.72
12.0	0.9	0.61	0.72
13.0	0.9	0.6	0.72

jdt

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.82	0.9
2.0	0.99	0.82	0.9
3.0	0.99	0.82	0.9
4.0	0.99	0.82	0.9
5.0	0.99	0.82	0.9
6.0	0.99	0.82	0.9
7.0	0.99	0.82	0.9
8.0	0.99	0.82	0.9
9.0	0.99	0.82	0.9
10.0	0.99	0.82	0.9
11.0	0.99	0.82	0.9
12.0	0.99	0.82	0.9
13.0	0.99	0.82	0.9

lucene

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.99	0.77	0.87
2.0	0.99	0.78	0.87
3.0	0.99	0.78	0.87
4.0	0.99	0.78	0.87
5.0	0.99	0.78	0.87
6.0	0.99	0.78	0.87
7.0	0.99	0.78	0.87
8.0	0.99	0.78	0.87
9.0	0.99	0.78	0.87
10.0	0.98	0.78	0.87
11.0	0.98	0.78	0.87
12.0	0.98	0.78	0.87
13.0	0.98	0.78	0.87

xorg

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.81	0.9

2.0	0.99	0.81	0.89
3.0	1.0	0.81	0.9
4.0	0.99	0.81	0.89
5.0	0.99	0.81	0.89
6.0	0.99	0.81	0.89
7.0	0.99	0.82	0.89
8.0	0.99	0.82	0.9
9.0	0.99	0.82	0.89
10.0	0.98	0.82	0.89
11.0	0.98	0.82	0.89
12.0	0.98	0.82	0.89
13.0	0.98	0.82	0.89

The best F1-Scores achieved (0.73) are using max_feat = 5 and 8 for the "jackrabbit" project. For "jdt" we get a sore of 0.9 across all values and similarly for "lucene". With the previous features the highest F1-Score we achieved for "jackrabbit" was 0.72 with max_feat = 3 and 6, but now it is 0.73 for max_feat = 5 and 8. Here we choose max_feat = 8 as the best value because it considers more features of the dataset and lower or higher values do not necessarily improve performance.

Therefore, the best settings for the random forests classifier with the new features is criterion = "entropy", n_estimators = 80, max_depth = 30, and max_feat = 8.

Logsitic Regression

jackrabbit

C	PRECISION	RECALL	F1-SCORE
0.0001	0.88	0.58	0.7
0.001	0.89	0.58	0.7
0.01	0.89	0.58	0.7
0.1	0.88	0.58	0.7
1.0	0.88	0.58	0.7
10.0	0.87	0.58	0.7
50.0	0.87	0.58	0.7
100.0	0.88	0.58	0.7
500.0	0.87	0.58	0.7
1000.0	0.87	0.58	0.7

jdt

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.82	0.9
0.001	0.99	0.82	0.9
0.01	0.99	0.82	0.9
0.1	0.99	0.82	0.9
1.0	0.99	0.82	0.9

10.0	0.99	0.82	0.9
50.0	0.99	0.82	0.9
100.0	0.99	0.82	0.9
500.0	0.99	0.82	0.9
1000.0	0.99	0.82	0.9

lucene

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.77	0.87
0.001	0.97	0.77	0.86
0.01	0.97	0.78	0.86
0.1	0.94	0.77	0.85
1.0	0.93	0.77	0.85
10.0	0.93	0.77	0.85
50.0	0.93	0.77	0.84
100.0	0.93	0.77	0.85
500.0	0.93	0.77	0.84
1000.0	0.93	0.77	0.85

xorg

C	PRECISION	RECALL	F1-SCORE
0.0001	0.99	0.81	0.89
0.001	0.99	0.81	0.89
0.01	0.99	0.82	0.89
0.1	0.99	0.82	0.89
1.0	0.98	0.82	0.89
10.0	0.98	0.82	0.89
50.0	0.98	0.82	0.89
100.0	0.98	0.82	0.89
500.0	0.98	0.82	0.89
1000.0	0.98	0.82	0.89

Again, like with SVM, the results for logistic regression with the bag-of-words features are practically the same as without them. The only difference in results is with the “lucene” project where in part 2 the F1-Score consistently dropped all the way down to 0.84 with C = 1.0, but here the F1-Score fluctuates with different C values. C = 0.0001 achieves the best F1-Score across all the projects and as such we choose this value to be optimal.

Support Vector Machine

jackrabbit

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.54	0.7
0.001	1.0	0.54	0.7

0.01	1.0	0.54	0.7
0.1	0.97	0.54	0.69
1.0	0.97	0.54	0.69
10.0	0.93	0.55	0.69
50.0	0.92	0.56	0.69
100.0	0.92	0.56	0.69
1000.0	0.88	0.57	0.69
10000.0	0.84	0.58	0.68

jdt

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.81	0.9
0.001	1.0	0.81	0.9
0.01	1.0	0.81	0.9
0.1	1.0	0.81	0.9
1.0	1.0	0.81	0.9
10.0	1.0	0.81	0.9
50.0	1.0	0.81	0.9
100.0	1.0	0.82	0.9
1000.0	0.99	0.82	0.9
10000.0	0.97	0.83	0.89

lucene

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.77	0.87
0.001	1.0	0.77	0.87
0.01	1.0	0.77	0.87
0.1	1.0	0.77	0.87
1.0	1.0	0.77	0.87
10.0	1.0	0.77	0.87
50.0	0.98	0.78	0.87
100.0	0.93	0.77	0.84
1000.0	0.92	0.77	0.84
10000.0	0.97	0.78	0.87

xorg

C	PRECISION	RECALL	F1-SCORE
0.0001	1.0	0.81	0.9
0.001	1.0	0.81	0.9
0.01	1.0	0.81	0.9
0.1	1.0	0.81	0.9
1.0	0.99	0.81	0.89
10.0	0.99	0.81	0.89
50.0	0.99	0.81	0.89

100.0	0.99	0.81	0.89
1000.0	0.98	0.81	0.89
10000.0	0.97	0.81	0.88

Utilizing the new features with SVM results in practically no difference in F1-Scores compared to the original dataset with the old features. In part 2, the "jackrabbit" results are exactly the same with the F1-Score decreasing as the C value increases. Both with the new features and the old in part 2, the highest F1-Score achieved on "jackrabbit" is 0.7 with C = 0.0001. The results for the other three projects are also the same. Nonetheless, we choose C = 0.0001 as the F1-Score seems to decrease steadily after this point.

Overall, comparing the different classifiers using the new feature set, the best performer is still random forests as it achieves the best (0.73 versus 0.70 of logistic and SVM on "jackrabbit") and most consistent F1-Scores across all the projects and with "jackrabbit" in particular, it performs the best when compared to SVM or logistic regression for example. We would not say the new features made a major difference in performance of the classifiers as can be seen with SVM and logistic regression, but they did make a slight difference when using the random forests and decision tree classifier.

Part 4: Improving our models by resampling

In this section we further improve our classification algorithms by resampling the training data as the number of buggy instances (1) is much less than the number of non-buggy or clean instances (0) – this leads to an imbalanced dataset and can result in skewed results that are not really representative of your actual predictive performance. Since we will be using three different resampling techniques, we will analyze the performance of each on the "jackrabbit" project and then pick the one that results in the best predictive performance. We will then run that resampling technique on all the projects. We apply the following resampling techniques on the training data:

SMOTE: select instances from minority class using the k-nearest neighbours of each instance and generates new instances using the selected instances and their neighbours.

ADASYN: generates minority data samples using k-nearest neighbours, however, unlike SMOTE, ADASYN uses a density distribution to decide the number of synthetic samples generated for each minority class sample.

Random under sampler (with/without replacement): Under samples the majority class by randomly picking samples with or without replacement.

First, we attempt resampling using SMOTE:

```
Original dataset shape Counter({0: 1856, 1: 407})
Resampled dataset shape Counter({0: 1856, 1: 1856})
Original dataset shape Counter({0: 678, 1: 392})
Resampled dataset shape Counter({0: 678, 1: 678})
Original dataset shape Counter({0: 1611, 1: 309})
Resampled dataset shape Counter({0: 1611, 1: 1611})
Original dataset shape Counter({0: 734, 1: 381})
Resampled dataset shape Counter({0: 734, 1: 734})
Original dataset shape Counter({0: 300, 1: 72})
Resampled dataset shape Counter({0: 300, 1: 300})
Original dataset shape Counter({0: 356, 1: 87})
Resampled dataset shape Counter({0: 356, 1: 356})
```

With SMOTE we are able to obtain an equal amount of buggy and clean instances with the original dataset being highly imbalanced as seen in the above figure. We will try various settings for SMOTE to determine which settings achieve the best predictive performance. In the process, we will also retune our classifiers from part 3.

Decision Tree

SMOTE

Using Gini

K_NEIGHBORS	MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.38	0.59	0.47
2.0	2.0	0.49	0.59	0.54
3.0	5.0	0.72	0.66	0.69
4.0	8.0	0.65	0.59	0.62
5.0	10.0	0.64	0.59	0.62
6.0	20.0	0.73	0.61	0.66
7.0	30.0	0.62	0.6	0.61
8.0	50.0	0.65	0.57	0.61
9.0	80.0	0.68	0.6	0.63
10.0	100.0	0.59	0.6	0.6

Using Entropy

K_NEIGHBORS	MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.38	0.59	0.47
2.0	2.0	0.49	0.59	0.54
3.0	5.0	0.64	0.58	0.61
4.0	8.0	0.62	0.57	0.59
5.0	10.0	0.61	0.6	0.6
6.0	20.0	0.71	0.62	0.66
7.0	30.0	0.68	0.59	0.63
8.0	50.0	0.63	0.59	0.61
9.0	80.0	0.71	0.6	0.65
10.0	100.0	0.58	0.57	0.58

With SMOTE, the results above show that the values fluctuate quite a bit with different values for the k_neighbors parameter in combination with the max_depth. Based on these results, we determine that k_neighbors = 6 is the optimal parameter value for SMOTE and a max_depth of 20 achieves the best F1-Score on the "jackrabbit" project. "entropy" and "gini" provide similar results but "gini" is slightly more consistent which is why we chose it.

ADASYN

Using Gini

K_NEIGHBORS	MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.37	0.59	0.46
2.0	2.0	0.46	0.64	0.53
3.0	5.0	0.59	0.67	0.63
4.0	8.0	0.66	0.61	0.63
5.0	10.0	0.65	0.62	0.64
6.0	20.0	0.69	0.61	0.65
7.0	30.0	0.71	0.62	0.66
8.0	50.0	0.67	0.59	0.63
9.0	80.0	0.65	0.6	0.62
10.0	100.0	0.72	0.61	0.66

Using Entropy

K_NEIGHBORS	MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	1.0	0.37	0.59	0.46
2.0	2.0	0.43	0.64	0.52
3.0	5.0	0.63	0.66	0.65
4.0	8.0	0.64	0.61	0.63
5.0	10.0	0.65	0.64	0.64
6.0	20.0	0.68	0.62	0.65
7.0	30.0	0.72	0.61	0.66
8.0	50.0	0.72	0.6	0.66
9.0	80.0	0.7	0.62	0.66
10.0	100.0	0.72	0.62	0.67

With ADASYN the results are quite different. With SMOTE we had selected the "gini" criterion to be optimal but here it appears that "entropy" performs better, as can be seen in the above results. ADASYN achieves a max F1-Score of 0.67 with k_neighbors = 10 and a decision tree depth of 10. The "gini" criterion is quite close but because the "entropy" results are slightly higher we chose it to be optimal.

Random Under Sampler

Using Gini

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.47	0.58	0.52
2.0	0.48	0.6	0.53
5.0	0.52	0.58	0.55

8.0	0.51	0.58	0.55
10.0	0.52	0.61	0.56
20.0	0.62	0.63	0.62
30.0	0.62	0.63	0.62
50.0	0.62	0.63	0.62
80.0	0.62	0.63	0.62
100.0	0.62	0.63	0.62

Using Entropy

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.47	0.58	0.52
2.0	0.49	0.59	0.54
5.0	0.65	0.67	0.66
8.0	0.64	0.64	0.64
10.0	0.64	0.65	0.64
20.0	0.62	0.64	0.63
30.0	0.62	0.64	0.63
50.0	0.62	0.64	0.63
80.0	0.62	0.64	0.63
100.0	0.62	0.64	0.63

WITH REPLACEMENT

Using Gini

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.4	0.6	0.48
2.0	0.56	0.62	0.59
5.0	0.67	0.65	0.66
8.0	0.55	0.61	0.58
10.0	0.51	0.61	0.56
20.0	0.52	0.6	0.56
30.0	0.52	0.6	0.56
50.0	0.52	0.6	0.56
80.0	0.52	0.6	0.56
100.0	0.52	0.6	0.56

Using Entropy

MAX_DEPTH	PRECISION	RECALL	F1-SCORE
1.0	0.4	0.6	0.48
2.0	0.56	0.63	0.59
5.0	0.5	0.65	0.56
8.0	0.6	0.6	0.6
10.0	0.6	0.61	0.61
20.0	0.59	0.61	0.6

30.0	0.59	0.61	0.6
50.0	0.59	0.61	0.6
80.0	0.59	0.61	0.6
100.0	0.59	0.61	0.6

Using random under sampling without replacement the highest F1-Score achieved is 0.66 with a max_depth of 5 under the "entropy" criterion. With "gini" the highest F1-Score we get is 0.62 which is why we decided on using the "entropy" criterion with the random under sampler without replacement. If we sample with replacement the results are much worse with both "gini" and "entropy" criterions achieving only a max F1-Score of 0.56 with the former and 0.61 with the latter, hence why we chose sampling without replacement.

Random Forests

SMOTE

Using Gini

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.78	0.61	0.69
2.0	0.75	0.6	0.66
3.0	0.79	0.62	0.69
4.0	0.76	0.61	0.68
5.0	0.73	0.61	0.67
6.0	0.76	0.61	0.68
7.0	0.74	0.61	0.67
8.0	0.73	0.62	0.67
9.0	0.75	0.62	0.68
10.0	0.77	0.62	0.69

Using Entropy

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.8	0.63	0.71
2.0	0.73	0.6	0.66
3.0	0.76	0.63	0.69
4.0	0.76	0.62	0.68
5.0	0.76	0.62	0.69
6.0	0.78	0.63	0.7
7.0	0.75	0.62	0.68
8.0	0.77	0.63	0.69
9.0	0.73	0.62	0.67
10.0	0.75	0.63	0.68

When tuning the k_neighbors parameter of SMOTE, we can see from the above results that different values for "k_neighbors" has varying effects on the performance of random forests. With k_neighbors = 1 we get a score of 0.71 for "entropy" and 0.69 for "gini". With k_neighbors = 5, we get a score of 0.69 for "entropy" and 0.67 for "gini". The scores for "entropy" are clearly higher so we choose k_neighbors = 1

as it provides the best F1-Score for the “jackrabbit” dataset. Next, we look at the “n_estimators” parameter.

Tuning “n_esimators” with “entropy” criterion:

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.68	0.59	0.64
2.0	0.81	0.57	0.67
5.0	0.7	0.59	0.64
8.0	0.77	0.6	0.67
10.0	0.77	0.6	0.68
20.0	0.77	0.6	0.67
30.0	0.8	0.62	0.7
50.0	0.78	0.62	0.69
80.0	0.79	0.63	0.7
100.0	0.8	0.63	0.71

Here, it can be seen above that increasing the “n_estimators” value has a positive impact on the F1-Score, achieving a maximum of 0.71 at n_estimators = 100. Thus, we select n_estimators = 100 as the optimal parameter value. Next, we look at the “max_features” parameter.

Using “entropy”, k_neighbors = 1, and n_estimators = 100 to tune “max_features”:

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.87	0.61	0.71
3.0	0.85	0.62	0.72
5.0	0.83	0.63	0.71
8.0	0.82	0.63	0.71
10.0	0.81	0.62	0.7
12.0	0.81	0.63	0.71
13.0	0.82	0.63	0.71
14.0	0.81	0.63	0.71
15.0	0.8	0.63	0.7
16.0	0.81	0.63	0.71

From the above table, the best value for max_features is 3 as it achieves the highest F1-Score of 0.72 among all the values. Thus, we must consider a maximum of 3 features when splitting a tree in random forests in order to achieve optimal results on the “jackrabbit” dataset.

ADASYN

Tuning “k_neighbors” using “gini”

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.76	0.61	0.68
2.0	0.81	0.63	0.71
3.0	0.77	0.63	0.69
4.0	0.77	0.63	0.69
5.0	0.75	0.62	0.68

6.0	0.76	0.63	0.69
7.0	0.76	0.62	0.68
8.0	0.75	0.63	0.69
9.0	0.74	0.63	0.68
10.0	0.76	0.63	0.69

Tuning “k_neighbors” using “entropy”

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.74	0.62	0.67
2.0	0.77	0.64	0.7
3.0	0.76	0.63	0.69
4.0	0.76	0.63	0.69
5.0	0.74	0.63	0.68
6.0	0.75	0.63	0.68
7.0	0.75	0.64	0.69
8.0	0.75	0.64	0.69
9.0	0.74	0.64	0.69
10.0	0.75	0.64	0.69

The ADASYN results for different values of k_neighbors are quite similar to SMOTE and we achieve the best F1-Score with the “gini” criterion specifically with k_neighbors = 2.

Tuning “n_estimators” with “gini” criterion and k_neighbors = 2:

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.7	0.59	0.64
2.0	0.85	0.58	0.69
5.0	0.76	0.62	0.68
8.0	0.81	0.61	0.7
10.0	0.81	0.61	0.7
20.0	0.81	0.62	0.7
30.0	0.81	0.62	0.7
50.0	0.81	0.62	0.7
80.0	0.8	0.63	0.7
100.0	0.81	0.63	0.71

As seen from the above table, the results are identical to SMOTE when tuning the “n_estimators” parameter using k_neighbors = 2. With both we achieve a maximum F1-Score of 0.71 with n_estimators = 100. We will now use n_estimators = 100 to tune the max_features parameter.

Tuning max_feat with “gini”, k_neighbors = 2, and n_estimators = 100:

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.83	0.62	0.71
3.0	0.82	0.63	0.71
5.0	0.82	0.63	0.71
8.0	0.81	0.63	0.71

10.0	0.81	0.62	0.7
12.0	0.8	0.63	0.71
13.0	0.81	0.63	0.71
14.0	0.8	0.63	0.7
15.0	0.8	0.62	0.7
16.0	0.8	0.63	0.7

Tuning the max_features parameter with ADASYN gives us a maximum F1-Score of 0.71 which is achieved using a max_feat value of 1, 3, 5, 8, 12, and 13. This is slightly lower than SMOTE, where we managed an F1-Score of 0.72 with max_feat = 3.

Random Under Sampler

WITH REPLACEMENT

Using Gini

PRECISION	RECALL	F1-SCORE
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59
0.54	0.65	0.59

Using Entropy

PRECISION	RECALL	F1-SCORE
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**
0.56	**0.65**	**0.6**

WITHOUT REPLACEMENT

PRECISION	RECALL	F1-SCORE
0.64	0.66	0.65

Using Gini

PRECISION	RECALL	F1-SCORE
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66
0.65	0.67	0.66

Using Entropy

0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65
0.64	0.66	0.65

Sampling with replacement does not give as good results as sampling without replacement and when using replacement the better F1-Score are achieved with “entropy” criterion, but without replacement it can be seen that the “gini” criterion gives better results than “entropy”. As the results imply, we adopt sampling without replacement using “gini” criterion with random under sampling method.

Tuning “n_estimators” with “gini” criterion and without replacement:

N_ESTIMATORS	PRECISION	RECALL	F1-SCORE
1.0	0.57	0.59	0.58
2.0	0.7	0.56	0.62
5.0	0.59	0.64	0.61
8.0	0.69	0.63	0.66
10.0	0.67	0.63	0.65
20.0	0.67	0.65	0.66
30.0	0.67	0.65	0.66
50.0	0.65	0.65	0.65
80.0	0.65	0.66	0.66
100.0	0.65	0.67	0.66

The above table shows that as the value for “n_estimators” gets larger, the better the F1-Score gets. The best score is achieved at n_estimators = 8 and then again at 20, 30, 80, and 100. As such, any one of these five values will be adequate for random forests. We decide to go with n_estimators = 80.

Tuning “max_features” with “gini” criterion and without replacement:

MAX_FEAT	PRECISION	RECALL	F1-SCORE
1.0	0.67	0.64	0.65
3.0	0.65	0.68	0.67
5.0	0.63	0.67	0.65
8.0	0.66	0.67	0.67
10.0	0.63	0.66	0.65
12.0	0.66	0.67	0.66

13.0	0.67	0.67	0.67
14.0	0.67	0.66	0.66
15.0	0.63	0.67	0.65
16.0	0.64	0.67	0.66

The highest F1-Score is 0.67 and is achieved using max_feat = 3, 8, and 13, meaning when we consider these number of features when splitting the tree, we achieve the best F1-Score. We decide to choose max_feat = 13.

From the three sampling methods, SMOTE performs the best with random forests (achieving an F1-Score of 0.72) with ADASYN providing similar results and random under sampling being the worst performer.

Support Vector Machine

SMOTE

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.49	0.5	0.49
2.0	0.48	0.51	0.49
3.0	0.51	0.51	0.51
4.0	0.46	0.5	0.48
5.0	0.5	0.5	0.5
6.0	0.49	0.51	0.5
7.0	0.5	0.5	0.5
8.0	0.51	0.5	0.5
9.0	0.49	0.51	0.5
10.0	0.48	0.5	0.49

As can be seen, rebalancing the dataset with SMOTE has decreased the F1-Score significantly compared to part (2) and (3). The best F1-Score is achieved with a k_neighbors value of 3 and as such we will choose this value.

Tuning C with k_neighbors = 3:

C	PRECISION	RECALL	F1-SCORE
0.0001	0.53	0.53	0.53
0.001	0.53	0.53	0.53
0.01	0.53	0.53	0.53
0.1	0.53	0.53	0.53
1.0	0.51	0.51	0.51
10.0	0.5	0.51	0.5
50.0	0.52	0.52	0.52
100.0	0.49	0.53	0.51
1000.0	0.48	0.56	0.52
10000.0	0.59	0.55	0.57

For the C parameter, like in part (2) and (3) we tried values ranging from 0.0001 to 10000 and unlike before, this time the different values have a significant impact on the F1-Score. Before the F1-Scores for "jackrabbit" remained relatively the same across all values but now the F1-Scores change slightly as we increase the value of C. The overall F1-Scores are a bit lower than in part (2) and (3) but that is because we have rebalanced the data now and one class is not heavily biased than the other. C = 10000 achieves the highest score of 0.57 so we choose this value as optimal.

ADASYN

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.46	0.5	0.48
2.0	0.36	0.48	0.41
3.0	0.44	0.5	0.47
4.0	0.4	0.51	0.45
5.0	0.35	0.48	0.4
6.0	0.37	0.49	0.42
7.0	0.4	0.49	0.44
8.0	0.41	0.49	0.45
9.0	0.44	0.5	0.47
10.0	0.44	0.5	0.47

The above results indicate that rebalancing with ADASYN is much worse than SMOTE as the F1-Scores from k_neighbors = 1 to 10 are much lower. For example, with k_neighbor = 10 we get an F1-Score of 0.47 but with ADASYN we get a score of 0.49. Here the best score is achieved using k_neighbors = 1 so we pick this value when tuning the C parameter.

Tuning C with k_neighbors = 1:

C	PRECISION	RECALL	F1-SCORE
0.0001	0.0	nan	nan
0.001	0.0	nan	nan
0.01	0.28	0.51	0.36
0.1	0.37	0.5	0.42
1.0	0.46	0.5	0.48
10.0	0.5	0.52	0.51
50.0	0.48	0.54	0.51
100.0	0.48	0.54	0.51
1000.0	0.41	0.59	0.48
10000.0	0.46	0.58	0.52

The k_neighbors value seems to be too low as we get "nan" on C = 0.0001 and 0.001 so we choose k_neighbors = 3 instead:

C	PRECISION	RECALL	F1-SCORE
0.0001	0.5	0.55	0.52
0.001	0.15	0.54	0.24
0.01	0.37	0.5	0.42
0.1	0.37	0.5	0.42

1.0	0.44	0.5	0.47
10.0	0.45	0.51	0.48
50.0	0.41	0.54	0.47
100.0	0.43	0.54	0.48
1000.0	0.43	0.58	0.49
10000.0	0.47	0.57	0.51

With k_neighbors = 3 the best C values appears to be 0.0001 as it achieves the highest F1-Score of 0.52. Again, unlike before in parts 2 and 3 where we saw little change with different C values, here we do get differing results for varying values for C. Compared to SMOTE, however, the results are still slightly worse. Thus, we choose C = 0.0001 as optimal for ADASYN.

Random Under Sampler

WITH REPLACEMENT

C	PRECISION	RECALL	F1-SCORE
0.0001	0.33	0.5	0.39
0.001	0.33	0.5	0.39
0.01	0.33	0.5	0.39
0.1	0.33	0.5	0.39
1.0	0.36	0.49	0.42
10.0	0.43	0.5	0.46
50.0	0.43	0.5	0.46
100.0	0.43	0.5	0.46
1000.0	0.4	0.54	0.46
10000.0	0.4	0.58	0.47

WITHOUT REPLACEMENT

C	PRECISION	RECALL	F1-SCORE
0.0001	0.53	0.53	0.53
0.001	0.53	0.53	0.53
0.01	0.53	0.53	0.53
0.1	0.53	0.53	0.53
1.0	0.55	0.52	0.54
10.0	0.5	0.52	0.51
50.0	0.54	0.53	0.54
100.0	0.54	0.53	0.54
1000.0	0.47	0.56	0.51
10000.0	0.52	0.57	0.55

As the above results show, sampling without replacement achieves significantly better F1-Scores across all C values. In both cases, as the C value gets higher, the F1-Scores also get better.

For SVM, out of SMOTE, ADASYN, and random under sampler, SMOTE performs the best as it achieves and F1-Score of 0.57 with C = 10000 but sampling without replacement is quite close with 0.55 with C = 10000.

Logistic Regression

SMOTE

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.68	0.61	0.64
2.0	0.7	0.6	0.65
3.0	0.61	0.59	0.6
4.0	0.64	0.6	0.62
5.0	0.62	0.59	0.61
6.0	0.65	0.6	0.62
7.0	0.63	0.59	0.61
8.0	0.65	0.6	0.63
9.0	0.68	0.6	0.64
10.0	0.7	0.6	0.65

With k_neighbors = 2 and k_neighbors = 10 we get the highest F1-Scores of 0.65. It seems that it is best to choose either a low or high k_neighbors value as in the middle the F1-Scores drop slightly going as low as 0.60 with k_neighbors = 3.

Tuning C with k_neighbors = 10:

C	PRECISION	RECALL	F1-SCORE
0.0001	0.6	0.61	0.6
0.001	0.62	0.61	0.62
0.01	0.63	0.61	0.62
0.1	0.64	0.6	0.62
1.0	0.7	0.6	0.65
10.0	0.66	0.59	0.63
50.0	0.67	0.6	0.63
100.0	0.67	0.59	0.63
500.0	0.67	0.6	0.63
1000.0	0.67	0.59	0.63

Here there is no direct relationship with the value of C and the F1-Score as we get a maximum score of 0.65 with C = 1.0 but drop to 0.63 with C = 10 and onwards. However, compared to logistic regression with SMOTE, these results are still much better. The best parameter value is thus C = 1.0 with an F1-Score of 0.65.

ADASYN

K_NEIGHBORS	PRECISION	RECALL	F1-SCORE
1.0	0.59	0.6	0.59
2.0	0.59	0.61	0.6
3.0	0.58	0.6	0.59
4.0	0.6	0.6	0.6
5.0	0.6	0.6	0.6
6.0	0.58	0.6	0.59
7.0	0.59	0.6	0.6
8.0	0.6	0.61	0.6

9.0	0.59	0.6	0.6
10.0	0.61	0.6	0.6

For ADASYN, the F1-Score does not change much with the different k_neighbors values only achieving a high of 0.6 with k_neighbors = 2 and also with 4, 5, 7, 8, 9, and 10. With ADASYN, we also decide to choose k_neighbors = 10.

Tuning C with k_neighbors = 10:

C	PRECISION	RECALL	F1-SCORE
0.0001	0.53	0.63	0.57
0.001	0.54	0.62	0.57
0.01	0.55	0.61	0.58
0.1	0.56	0.6	0.58
1.0	0.61	0.6	0.6
10.0	0.6	0.59	0.6
50.0	0.61	0.6	0.61
100.0	0.6	0.6	0.6
500.0	0.61	0.6	0.61
1000.0	0.6	0.6	0.6

Compared with SMOTE, the ADASYN results with k_neighbors = 10 are slightly worse only achieving a high of 0.61 versus 0.65 with SMOTE. With ADASYN, C = 50 and C = 500 perform best achieving an F1-Score of 0.61. Thus, with ADASYN, the optimal C value is 1.0.

Random Under Sampler

WITH REPLACEMENT

C	PRECISION	RECALL	F1-SCORE
0.0001	0.57	0.6	0.58
0.001	0.58	0.61	0.6
0.01	0.59	0.61	0.6
0.1	0.58	0.61	0.59
1.0	0.59	0.62	0.6
10.0	0.58	0.62	0.6
50.0	0.59	0.62	0.6
100.0	0.58	0.62	0.6
500.0	0.58	0.62	0.6
1000.0	0.58	0.61	0.6

WITHOUT REPLACEMENT

C	PRECISION	RECALL	F1-SCORE
0.0001	0.54	0.61	0.57
0.001	0.63	0.63	0.63
0.01	0.62	0.63	0.62
0.1	0.55	0.61	0.58

1.0	0.57	0.61	0.59
10.0	0.6	0.61	0.6
50.0	0.57	0.61	0.59
100.0	0.57	0.61	0.59
500.0	0.57	0.61	0.59
1000.0	0.62	0.62	0.62

Again, as with SVM, sampling without replacement achieves slightly better (a max F1-Score of 0.63) results than sampling with replacement (a max F1-Score of 0.60). The best results achieved for sampling without replacement are with C = 0.001 which obtains an F1-Score of 0.63.

Overall, just as with SVM, the best resampling method for logistic regression is SMOTE, as it achieves an F1-Score of 0.65 with k_neighbors = 10, the highest of the three methods.

In conclusion, many improvements have been made to the F1-Scores of the various classifiers using bag-of-words features and resampling with SMOTE, ADASYN, and random under-sampling with/without replacement. It turned out that the new features we created had little impact on most of the classifiers besides random forests where we saw a slight increase in predictive performance. Then we employed resampling strategies like SMOTE, ADASYN, and random under-sampling to further improve results. Overall, random forests remains the best classifier on this dataset particularly on the “jackrabbit” project where it beats out SVM, logistic, and decision tree, achieving an F1-Score of 0.73 from part (3) on the original feature set. With SMOTE, random forests achieves a slightly less F1-Score of 0.72.

References

[1] https://stackoverflow.com/questions/46756606/what-does-splitter-attribute-in-sklearns-decisiontreeclassifier-do

[2] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier