**FFIT 3162/ FIT COMPUTER SCIENCE PROJECT**

**Final Project Report**

**Project title: Improving Software Testing using Software Fault Prediction Methods when data is highly imbalanced**

**Team:**

FIT3162\_MA\_4

**Team Members:**

* Tah Wen Zhong
* Jason Toh Zhern Wee
* Ethan Hor Sheng Jian

**Project Supervisor:**

Golnoush Abaei

**Table of contents**

[**1. Introduction**](#_qqtq3nh9ra6m) **4**

[**2. Background**](#_sp7z6xldtl2m) **5**

[2.1 Literature Review](#_vx9sajh0mdlx) 5

[2.2 Machine learning models](#_m19wnq635elc) 5

[2.2.1 Naive Bayes](#_93yisc7j95is) 6

[2.2.2 Decision Tree](#_p0z2jey61foz) 6

[2.2.3 Complement Naive Bayes](#_qt7fgqyzox72) 6

[2.2.4 Logistic Regression](#_fme3qhc0m39m) 6

[2.2.5 Multi-Layer Perceptron](#_5e8bcfe6940n) 6

[2.2.6 Random Forest](#_oyi8tj38x33j) 7

[2.2.7 Rotation Forest](#_mvkkh9wv4ncn) 7

[2.2.8 Voting](#_gr37hmezyb7g) 7

[2.3 Preprocessing techniques](#_5vf68eyrrkum) 7

[2.4 K-Fold](#_xrh3e8ftzvuc) 7

[2.5 Undersampling](#_vo7l43ejnww4) 8

[2.6 Feature selection techniques](#_psyjxwmbsdc7) 8

[2.6.1 Correlation-based Feature Selection (CFS)](#_b5q191cohee6) 8

[2.6.2 Recursive Feature Elimination (RFE)](#_wx9hbf4dpk51) 8

[2.7 Related works](#_878j17xywibi) 9

[**3 Outcomes**](#_e4uhij6wzgmg) **13**

[3.1 What has been implemented](#_a0d6gieouvh) 13

[3.2 Results achieved/product delivered](#_y0vznwcrn01) 14

[3.2.1 Analysis 1: Prediction model](#_madmb7jw3495) 17

[3.2.2 Analysis 2: Feature reduction](#_4d34og8) 19

[3.2.3 Analysis 3: Feature selection method](#_rdwvd95v1pwb) 24

[3.2.4 Summary](#_xck3rgo2sqw2) 25

[3.3 How are requirements met](#_ll2u74vtrtxt) 26

[3.3 Justification of decisions made](#_55fya3islmzs) 29

[3.3.1 Prediction models](#_1jc21dt3jx6k) 29

[3.3.2 Datasets](#_bn2vfhzk8lm) 29

[3.3.3 Resolving imbalanced data](#_rntmyfokoaib) 29

[3.3.4 Performance metrics](#_ndrnlpk7u2n1) 30

[3.4 Discussion of Results](#_gmzcw9mozed9) 30

[3.4.1 Proposed method](#_uwz1lqoci50f) 30

[3.4.2 Analysis 4: Performance comparison with other algorithms](#_d8v143xdj4zf) 31

[3.4.3 Results summary](#_6i9mmojc6nw8) 33

[3.5 Limitations of project outcomes](#_b9nox6xv9az) 33

[3.5.1 Product limitations](#_7y1i8mrpgzka) 33

[3.5.2 Results limitations](#_ezlchg5cg7v) 33

[3.6 Improvements & Possible Future Works](#_eqklo4vlcnbq) 34

[3.6.1 Improvements](#_stskt3uhkfai) 34

[3.6.2 Future work](#_yblvmkwceb02) 34

[3.7 Critical discussion on project outcome](#_650slfkgqshd) 35

[**4. Methodology**](#_qc7mvwd66x3l) **36**

[4.1 System architecture](#_r881ov2pmmj0) 36

[4.2 Data preparation](#_7v49s7khiiyv) 37

[4.2.1 Data reading/Preprocessing](#_cwb26mu0ann0) 38

[4.2.2 Undersampling/K-fold cross validation](#_pyng6qf3uqj5) 38

[4.2.3 Feature selection](#_woww0z8yx60o) 39

[4.3 Main algorithm](#_czodh1qqga5m) 40

[4.3.1 Prediction models](#_i6o8g8qah238) 41

[4.3.2 Model evaluation](#_5w8hdshjnqpx) 42

[4.4 User interface](#_a6hn7cy5cz4r) 43

[**5 Software Deliverables**](#_tjrrynwn9if9) **45**

[5.1 Summary of software deliverables](#_qbeqjyxy2pfi) 45

[5.1.1 What is delivered](#_boly3yq66x2t) 45

[5.1.2 Sample Screenshots and Descriptions of Usage](#_sp78nrej4wyb) 45

[5.2 Summary and discussion of handling software qualities](#_faju73xt0umf) 48

[5.2.1 Documentation and Maintainability](#_38bxc2ltlysz) 48

[5.2.2 Security](#_iyqz4mnv6rqd) 48

[5.2.3 Robustness](#_d7ri2sb4zhr6) 48

[5.2.4 Usability](#_shme2f5dgi15) 48

[5.2.5 Scalability](#_1rbhfspqnhso) 49

[5.3 Sample Source Code in Appendix](#_yfoz1mapno3e) 49

[**6 Critical Discussion**](#_y91m45p6c5hw) **50**

[**7 Conclusion**](#_xet4d51wkb1) **52**

[**8. Appendix**](#_f22scei47s7q) **53**

[8.1 Normalization](#_avflgi2okz8g) 53

[8.2 K Fold and Stratified K Fold](#_pux9hylkuhfu) 53

[8.3 IHT](#_35do2hj8x6u5) 54

[8.4 Preprocess](#_xvfvmht18y25) 54

[8.5 Feature Selection](#_mhmbowfzt6of) 55

[8.6 Model Creation](#_2p3dkjn764lr) 56

[8.7 Writing to CSV](#_arjefu1a8840) 56

[8.8 Main Program](#_db4f43tkbqis) 57

[8.9 User Interface](#_4ffp2xt6kct) 58

[**9. References**](#_t7btqh40l7yd) **59**

# 

# 1. Introduction

In an effort to make the majority of softwares defect-free, a considerable amount of budget needs to be allocated towards the software testing phase. As each day goes by, this budget slowly starts to rise, as most software are also growing in terms of size and complexity. This causes an issue for certain companies which are unable to allocate sufficient resources towards testing

To tackle this issue, many researchers have come up with software fault prediction models in hopes that these models can help to detect certain modules in software which have a higher tendency of being defect, so that resources can be allocated more efficiently when testing software (Yucalar et al., 2020). Although this is a feasible plan, the effectiveness of these machine learning models also depends on a few factors, with one such factor being its ability to tackle the issue of data imbalance.

The main objective of this study is to create a software fault prediction model that is efficient in terms of performance. Additionally, this software fault prediction model should also have the ability to solve the issue of data imbalance, as it would help to boost the algorithm in terms of performance when it deals with imbalanced datasets.

# 

# 2. Background

## 2.1 Literature Review

The process of software testing is key to ensuring that the software is defect-free, resulting in a final product which is of good quality. However, as software continues to grow in terms of size and complexity, so does the cost of testing (Tong et al., 2018). To tackle this, many different parties have come up with different versions of software fault prediction models, which help to detect modules of a software which are more fault prone, so that the budget for software testing can be efficiently allocated. Though, the effectiveness of the models is dependent on its efficiency, as well as how other external factors are considered when developing the prediction model. One such external factor to be considered is the issue of dataset imbalance.

According to Le et al. (2018), the issue of dataset imbalances involves a classification for a dataset in which there are one or more classes which have a large number of instances compared to the other classes within the dataset. The class with the large number of instances is referred to as the majority class, while the latter is the minority class. Class imbalance can be found in many real-world datasets, such as protein detection, disease diagnosis and bankruptcy prediction (Le et al., 2018). There are several ways to resolve the issue of data imbalance. Firstly, Boughorbel et al. (2017) has proposed to either rebalance the class distribution of the majority and minority class, or to learn more from the minority class when the data is misclassified. Another approach is from Le at al. (2018), which involves an undersampling method called the Instance Hardness Threshold (IHT).

This literature review mainly aims to analyse these techniques as explored by other researchers in this field, so that it may help us to propose an approachable method for software fault prediction.

## 2.2 Machine learning models

In the field of software fault prediction, it is common for many researchers to make use of prediction models to help with predicting defects in software. According to Yucalar et al. (2020), a prediction model is one that is trained with metrics such as software metrics and fault data from training datasets so that it is able to estimate defects in future software datasets. There are many common prediction models being used today, with the prediction models being split into two categories, base and ensemble. As studied by Yucalar et al. (2020), some examples of base predictors are Naive Bayes, Logistic Regression, Multi-Layer Perceptron, and many more. On the other hand, ensemble prediction models are usually a combination of base prediction models. By combining them, it improves the diversity of decisions made within the model, making the model more effective in being trained (Yucalar et al., 2020). After a considerable amount of research, we have decided to implement these models (as described in the upcoming subsections) in our system.

### 2.2.1 Naive Bayes

According to Ray (2017), the Naive Bayes model is a prediction model which makes use of the Bayes’ Theorem. It works in a way where it assumes that the presence of any feature in a class is unrelated to the other features. This algorithm is common in most studies we have researched, and has been stated by Rana et al. (2015) that it outperforms quite a number of other prediction models on its own.

### 2.2.2 Decision Tree

Based on an article by Xoriant (2017), a decision tree is a form of supervised machine learning which involves data being separated along a parameter. They also further explained that the tree mainly consists of two different entities, which are decision nodes and leaves, with the leaves acting as final outcomes for every decision path. In the case of software fault prediction, the final outcomes for the leaves would be whether or not a software is considered defect.

### 2.2.3 Complement Naive Bayes

Complement Naive Bayes works just like the normal Naive Bayes algorithm, with the exception being that instead of calculating the probability of a piece of data belonging to a certain class, it calculates the probability of the data belonging to all the other classes within the dataset. According to GeeksforGeeks (2020), this algorithm performs well when used on imbalanced datasets, and therefore is crucial for us to include in our program.

### 2.2.4 Logistic Regression

Logistic regression is a statistical prediction model, and is suitable for machine learning problems which involve binary classification. According to Brownlee (2020), the Logistic Regression algorithm is usually represented by an equation, where the input values x are used to predict the output value of y. This would mean that it is a suitable choice to be included in our program, as our prediction models are aimed at software fault prediction.

### 2.2.5 Multi-Layer Perceptron

According to Nicholson (n.d.), a multi-layer perceptron is an artificial neural network that consists of multiple perceptrons, which are essentially linear classifiers. Among the layers of perceptrons within this prediction model, there consists of an input layer, an output layer and a random number of hidden layers. It works in such a way where the input layer first receives input. Then, the hidden layers, which are the computational engine of the model itself, process the input, which is then passed on to the output layer to make a prediction based on the input.

### 2.2.6 Random Forest

A random forest prediction model is an ensemble prediction model which consists of building a large number of decision trees. According to Section (2020), it is used in regression and classification problems, and is trained through an algorithm called bagging, which helps to improve the accuracy of the model itself. They have also stated that this algorithm is beneficial in different ways, as it is efficient at handling missing data, and also tackles the problems of overfitting which is present in decision trees.

### 2.2.7 Rotation Forest

The rotation forest prediction model works similar to the random forest model in terms of building an ensemble of decision trees with the use of random attributes. The difference, as stated by Yucalar et al. (2020), is that it makes use of the Principal Components Analysis transformation to diversify the attribute sets which are passed to the decision trees.

### 2.2.8 Voting

The voting ensemble prediction model, as defined by Yucalar et al. (2020), is a model which takes into account the results made from an ensemble of prediction models, to make a decision or a prediction. According to Brownlee (2020), the aim of this model is to maximize its performance by combining the results it has obtained from the ensemble of prediction models trained within it. This model is useful when the ensemble of models trained within it all have roughly the same performance.

## 2.3 Preprocessing techniques

When it comes to training prediction models, one of the issues that is commonly faced is within the dataset itself. Occurrences such as duplicate data within the dataset may affect the performance of the prediction models themselves. By applying certain preprocessing techniques towards the dataset, such as techniques present in Haque et al. (2016) proposed approach, which are the removal of duplicate data, replacement for missing values and normalization of data, there will be a noticeable improvement in the performance of the prediction models. With that, we have decided to include these techniques into our proposed approach as well.

## 2.4 K-Fold

According to Brownlee (2018), k-fold cross validation is a method in machine learning which is used to get a rough approximation of the skill of a prediction model. To summarize, this procedure splits the dataset used on the prediction model into k number of folds, with one of the folds acting as the test dataset, while the others are used to train the prediction model itself. This technique is very commonly used in the proposed approaches by other researchers, with one good example being the GA-EoC algorithm by Haque et al. (2016). Therefore, we have decided to implement this method in our approach as well, in the hopes that it would benefit the training process of our models.

## 2.5 Undersampling

The process of undersampling data is one which involves reducing the number of data present in the majority class (Yen & Lee, 2006). It is helpful for solving the issue of data imbalance, and there are a few ways of approaching it. One good example of this is from Le et al. (2018), where they perform undersampling through the usage of the Instance Hardness Threshold (IHT), which helps to remove data from the majority class which have a high IHT value. In short, the dataset is rebalanced by removing noisy data from the majority class. Another example of data undersampling is from Haque et al. (2016), in which they use an undersampling technique which separates the datasets into multiple datasets in such a way that each class within the dataset has a balanced dataset produced for it. While both of these approaches are feasible, we have made the decision to include the undersampling technique through the Instance Hardness Threshold (IHT) to help balance the datasets.

## 2.6 Feature selection techniques

In our proposal, we discussed the properties and importance of software metrics for training predictors. For software fault prediction models in particular, static code and process metrics are the most commonly used types (Tong et al., 2018). The extraction of software metrics such as the ones stated before this are often redundant, which may negatively affect the performance of the prediction models that are trained. Also, we highlighted how recent studies such as one by Pandey et al. (2021) had revealed the impact of feature selection towards the performance of prediction software. However, our previous research has minimal coverage on this field. Therefore, we have done additional research on this field to tackle this problem. Our findings will be presented in the two subsections below.

### 2.6.1 Correlation-based Feature Selection (CFS)

Correlation-based Feature Selection (CFS) is a feature selection algorithm that uses a correlation coefficient to measure the linear relationship between two software metrics, with a higher absolute value of the coefficient meaning a stronger relationship between the two metrics (Yin et al., 2013). To make use of this correlation coefficient in machine learning, the software metrics can be ranked according to their correlation scores. Then, only the top ranking software metrics will be selected for training the model. We find this approach for feature selection to be useful, therefore it will be included in the algorithm.

### 2.6.2 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a feature selection algorithm which involves eliminating non-predictive features from the dataset, leaving it with a more optimized selection of predictive features. According to Rtayli and Enneya (2020), they have found several cases where the usage of RFE has improved the performance of the prediction models. One such example would be a decision tree model which made use of the RFE algorithm to improve the selection of relevant features, which resulted in an improvement in the efficiency of an Intrusion Detection System (IDS) (Nkiama et al., 2016, as cited in Rtayli and Enneya, 2020). With this algorithm being evidently efficient in the research we did, it will be included in our proposed algorithm as well.

## 2.7 Related works

The following table contains a compilation of the research articles that we have read through, as well as the summaries of the contents within them:

**Table 1: Compilation of research articles that are relevant to our topic**

| **Source** | **Technique** | | | **Methods** | **Strength** |
| --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Zhang et al. (2015) | ✔ |  |  | * Preprocessing to tokenize the source code file * Tokens extracted in the preprocessing phase is used to build 6 classifiers * The 6 classifiers will be used to build a composer to determine if a source code file is vulnerable or not | * Prediction models that are built from text features perform more efficiently than models built from code metrics * VulPredictor approach has an impressive training time, faster than most ensemble predictors as well as certain base predictors |
| Le et al. (2018) |  | ✔ |  | * 5-fold-cross-validation * Four sections of the dataset are resampled through undersampling, and are then used to train the prediction model. Then, the trained prediction model is used to predict bankruptcy for the remaining 5th dataset | * Having the ability to manipulate the dataset through undersampling is a great tool. Through making use of a concept called Instance Hardness Threshold (IHT), the dataset is resampled by removing instances of data that are more likely to be misclassified. |
| Yucalar et at. (2020) | ✔ |  |  | * 10-fold-cross validation * Combining base predictors using ensemble predictions | * The ensemble prediction model outperforms the base prediction models by a large margin, obtaining better FM and AUC values. * Ensemble predictors are flexible, the base predictors can be added and removed to obtain different results.. |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **Methods** | **Strength** |
| --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Pandey et at. (2021) | ✔ |  |  | * A compilation of software prediction methods devised from other projects | * The article contains the results of many different projects regarding software predictions which becomes a good source for comparison when evaluating our project * The article includes common components used for software fault prediction as well as a detail explanation for each component |
| Boughorbel et al. (2017) | ✔ |  |  | * In certain cases where the dataset is imbalanced but also small, performing undersampling would not be very desirable. * This article proposes a method which attempts to optimize Matthew’s Correlation Coefficient (MCC), a performance metric defined in terms of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN), to deal with evaluating the performance of a machine learning algorithm on an imbalanced dataset. | * The performance metric created through the optimization of MCC is simple and efficient, therefore requiring less computational power compared to other performance metrics. |
| Tong et al. (2018) | ✔ |  |  | * Uses a deep learning model called Stacked denoising autoencoders (SDAEs) to extract deep representations from the traditional software metrics * Uses two-stage ensemble (TSE) to deal with class imbalance and overfitting problems * 5 Fold Cross Validation | * The two-stage ensemble (TSE) uses ensemble classifiers as base learners and combines them using weighted average probabilities develops an effective and efficient method against overfitting and class imbalance compared to traditional ensemble methods |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **Methods** | **Strength** |
| --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Ali Rana et al. (2015) | ✔ |  |  | * Association mining is applied to get the focus itemsets * Naive Bayes classifier is developed and validated for the datasets * Model is validated for different number of bins to check stability of the preprocessing approach | * The preprocessing phase proposed in the article which segregates itemsets into Defective module and Non-Defective Modules will facilitate the later process much easier and faster as the data has been modified |
| Yap et al. (2014) | ✔ |  |  | * Bagging   Randomly selects a datasets with N samples   * Obtain a learner from the resampled dataset * Use the predictive model to predict the cases * Combine all predicted models into aggregated model * Using a voting approach, the class that has been predicted most often is chosen * Boosting   Similar to Bagging, but instead apply the learner on the training dataset and compute the sum of the weighted errors of all training samples | * The article introduces two new ensemble methods - Bagging and Boosting * Bagging deals with variance and overfitting * Boosting deals with bias and underfitting * Both ensemble methods go hand-to-hand together and are useful to resample data when needed |
| Haque et al. (2016) | ✔ |  |  | * 10 fold cross validation * Genetic algorithm called GA-EoC uses the 10 fold cross validation on the set of training data to evaluate the performance of the of each candidate ensembles | * The article discusses how diverse set of base classifiers can be useful in identifying different characteristics of the training set and these classifiers are referred as heterogeneous ensembles |

**Table 1: Compilation of research articles that are relevant to our topic (continued)**

| **Source** | **Technique** | | | **Methods** | **Strength** |
| --- | --- | --- | --- | --- | --- |
| **Sup** | **Semi** | **Unsup** |
| Yen et al. (2006) |  |  | ✔ | * SBC (under-Sampling Based on Clustering) considers the ratio of the number of majority class to the number of minority class samples in the cluster and selects a suitable number of majority class for undersampling | * The article introduces a new technique for undersampling which is based on clustering * This further widens our techniques when choosing the appropriate sampling method * SBC was proven to not only has high classification accuracy but also fast execution time |
| Rtayli & Enneya (2020) | ✔ |  |  | * A combination the support vector machine prediction model and recursive feature elimination (SVM-RFE) to obtain the best predictive features from a dataset * A technique called Synthetic Minority Oversampling Technique (SMOTE) to deal with data imbalance. | * The article introduces an oversampling technique for dealing with data imbalance * It also has a good feature selection algorithm to learn from . |
| Yin et al. (2013) | ✔ |  |  | * Usage of a variety of feature selection methods to select the best features in a dataset, especially on those datasets which have imbalance data. | * The article provides a lot of insight in terms of feature selection algorithms. |

# 

# 3 Outcomes

## 3.1 What has been implemented

From our research, we were able to identify items which we felt were suitable to be added into our system. These items were listed in our project proposal for FIT3161. Our final product incorporates all the items, as well as the new additions from the current FIT3162. Below are the list items we successfully implemented into our system.

List of techniques

* Missing data handling
* Normalisation
* Undersampling using IHT techniques
* K-fold Cross-validation
* Correlation-based feature selection
* Recursive feature elimination

List of machine learning models (base)

* Multi-Layer Perceptron
* Logistic Regression
* Naïve Bayes
* Complement Naïve Bayes
* Decision Tree

List of machine learning models (ensemble)

* Rotation Forest
* Voting
* Random Forest

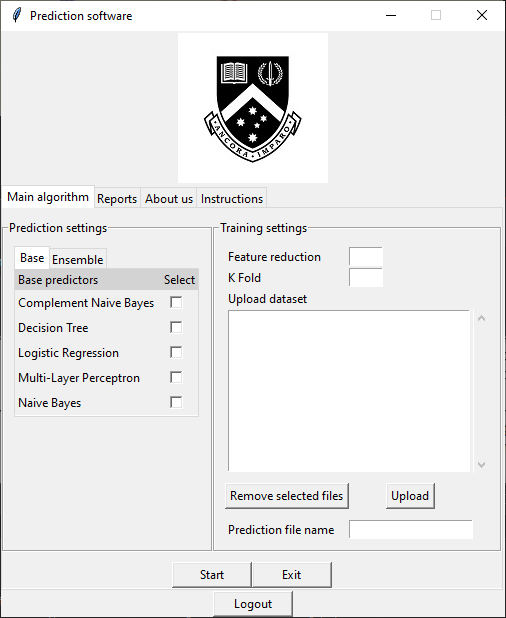
User Interface

* Login Screen
* Main page
* Results page
* Feature Selection Page
* Report

## 

## 3.2 Results achieved/product delivered

The final product of our project is a software which builds and evaluates the performance of various fault prediction models for any given dataset. Multiple datasets can be provided within a single run, and will be processed appropriately using the techniques stated in subsection 3.1. Furthermore, the application provides a user interface that improves usability and eases the evaluation process. A detailed explanation on the implementation of our system’s can be found in Section 4.



**Fig. 1 Main page of the application**

With the use of our application, we were able to conduct a number of experiments to examine the performance of various combinations of models and techniques. Our analysis utilizes datasets from the NASA and PROMISE repository, which we selected based on their features. We studied the properties of each dataset and tabulated the information as shown in the tables below.

**Table 2: Properties of datasets from NASA (Shepherd et al., 2014) repository**

| **Dataset** | **Procedural Metric** | **Object oriented Metric** | **Misc.**  **Metric** | **Metric count** | **Defective** | **Not defective** | **Module count** | **Degree of imbalance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CM1.arff | 25 | 11 | 2 | 38 | 12.84% | 87.16% | 327 | Moderate |
| JM1.arff | 17 | 4 | 0 | 21 | 20.88% | 79.12% | 7720 | Mild |
| KC1.arff | 17 | 4 | 0 | 21 | 25.3% | 74.7% | 1162 | Mild |
| KC3.arff | 25 | 13 | 2 | 40 | 18.56% | 81.44% | 194 | Moderate |
| KC4.arff  (Raw) | 24 | 14 | 2 | 40 | 48.8% | 51.2% | 125 | Normal |
| MC1.arff | 24 | 13 | 2 | 39 | 1.84% | 98.16% | 1952 | High |
| MC2.arff | 24 | 13 | 2 | 39 | 35.48% | 64.52% | 124 | Mild |
| MW1.arff | 24 | 11 | 2 | 37 | 10% | 90% | 250 | Moderate |
| PC1.arff | 24 | 11 | 2 | 37 | 8.1% | 91.9% | 679 | Moderate |
| PC2.arff | 23 | 11 | 2 | 36 | 2.22% | 97.78% | 722 | High |
| PC3.arff | 24 | 11 | 2 | 37 | 12.35% | 87.65% | 1053 | Moderate |
| PC4.arff | 24 | 11 | 2 | 37 | 13.86% | 86.14% | 1270 | Moderate |
| PC5.arff | 23 | 13 | 2 | 38 | 27.04% | 72.96% | 1694 | Mild |

**Table 3: Properties of datasets from Promise (Menzies, 2004) repository**

| **Dataset** | **Procedural Metric** | **Object oriented Metric** | **Misc.**  **Metric** | **Metric count** | **Defective** | **Not defective** | **Module count** | **Degree of imbalance** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| kc1.arff | 17 | 4 | 0 | 21 | 15.46% | 84.54% | 2109 | Moderate |
| cm1.arff | 17 | 4 | 0 | 21 | 9.84% | 90.16% | 498 | Moderate |
| kc2.arff | 17 | 4 | 0 | 21 | 20.5% | 79.5% | 522 | Mild |
| jm1.arff | 17 | 4 | 0 | 21 | 19.35% | 80.65% | 10885 | Moderate |
| pc1.arff | 17 | 4 | 0 | 21 | 19.35% | 80.65% | 1109 | Moderate |

These datasets will be used throughout the analysis on our program, where we conducted various experiments to analyze its impact towards performance. For all these experiments, our focus would mainly be on datasets which are highly imbalanced. As such, our experiments were conducted using datasets that were categorized to have a moderate or high degree of imbalance. The selected datasets are as followed:

* Moderate: CM1, KC3, MW1, PC1, PC3, PC4
* High: MC1, PC2

There are three analyses that were performed on our program. Each of these allowed us to not only evaluate the performance of various methods, but to also study the effects of techniques used towards performance. As such, we believe that our results will be a great contribution towards this field of study. These analyses are included in the following subsections.

### 

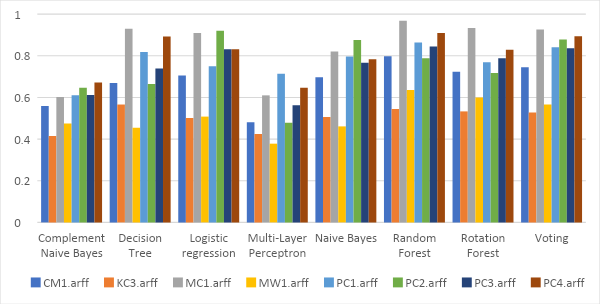
### 3.2.1 Analysis 1: Prediction model

There are several prediction models included within our program, each having different levels of proficiency towards handling imbalanced datasets. To compare the performance of each model, we used our program to obtain the performance result for all models in chart and tabular form. On a side note, no feature selection methods were used as this analysis focuses solely on the effectiveness of each model.

AUC Score

**Table 4: AUC results for the model analysis**

| **Model name** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CM1.arff | 0.559 | 0.669 | 0.705 | 0.481 | 0.697 | 0.798 | 0.723 | 0.745 |
| KC3.arff | 0.415 | 0.566 | 0.501 | 0.424 | 0.506 | 0.545 | 0.533 | 0.528 |
| MC1.arff | 0.602 | 0.93 | 0.909 | 0.61 | 0.82 | 0.968 | 0.933 | 0.926 |
| MW1.arff | 0.475 | 0.455 | 0.508 | 0.378 | 0.461 | 0.636 | 0.6 | 0.566 |
| PC1.arff | 0.611 | 0.818 | 0.75 | 0.714 | 0.796 | 0.864 | 0.769 | 0.841 |
| PC2.arff | 0.646 | 0.665 | 0.92 | 0.479 | 0.876 | 0.788 | 0.717 | 0.878 |
| PC3.arff | 0.612 | 0.739 | 0.831 | 0.563 | 0.766 | 0.845 | 0.788 | 0.836 |
| PC4.arff | 0.672 | 0.893 | 0.831 | 0.646 | 0.783 | 0.909 | 0.829 | 0.894 |
| **Average performance** | 0.574 | 0.716875 | 0.744375 | 0.536875 | 0.713125 | 0.794125 | 0.7365 | 0.77675 |



**Fig. 2 Bar chart displaying the AUC evaluation scores for the model analysis**

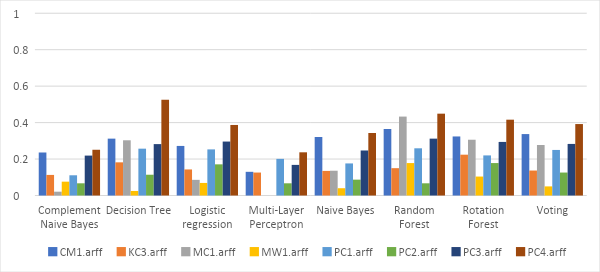
As shown, most models performed outstandingly well in this field. In literature, Naïve Bayes and Logistic Regression are known to be proficient for handling imbalanced datasets. So, the scores for these two predictions were expected to perform greater than other base predictors. If we observe Figure 2, these two base models are shown to achieve scores which outperform others, these scores fall between acceptable and excellent range. If we look at the datasets with the highest degree of imbalance, MC1 and PC2, both models show outstanding performance. Overall, the Logistic regression model is shown to have results for handling imbalanced datasets.

For the ensemble predictors, these models overall perform greater than most base prediction models. Consequently, the voting and random forest models achieved the best performance among all models.

F1-score

**Table 5: F1-score results for the model analysis**

| **Model name** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CM1.arff | 0.236 | 0.312 | 0.272 | 0.13 | 0.321 | 0.365 | 0.324 | 0.337 |
| KC3.arff | 0.113 | 0.182 | 0.143 | 0.126 | 0.135 | 0.15 | 0.224 | 0.137 |
| MC1.arff | 0.021 | 0.303 | 0.086 | 0 | 0.136 | 0.433 | 0.306 | 0.277 |
| MW1.arff | 0.076 | 0.025 | 0.069 | 0 | 0.04 | 0.178 | 0.104 | 0.05 |
| PC1.arff | 0.111 | 0.257 | 0.253 | 0.201 | 0.176 | 0.259 | 0.22 | 0.25 |
| PC2.arff | 0.067 | 0.114 | 0.171 | 0.067 | 0.087 | 0.067 | 0.178 | 0.126 |
| PC3.arff | 0.219 | 0.282 | 0.296 | 0.168 | 0.247 | 0.312 | 0.294 | 0.283 |
| PC4.arff | 0.251 | 0.525 | 0.387 | 0.237 | 0.343 | 0.449 | 0.416 | 0.392 |
| **Average performance** | 0.13675 | 0.25 | 0.209625 | 0.116125 | 0.185625 | 0.276625 | 0.25825 | 0.2315 |



**Fig. 3 Bar chart displaying the F1-score evaluation scores for the model analysis**

For F1-score, none of the models were able to achieve outstanding results, which is a common outcome for this field. An interesting observation was that every model shown to perform poorly for the MW1, which can be observed in 3rd row of Table 5.

An interesting observation was that the best scores were achieved mainly by the tree-based models. The decision tree achieved promising results and shows to have the best results when compared with the other base prediction models. The Rotation Forest had the best performance, achieving remarkable scores and showing good consistency.

One thing to highlight would be the performance of the Logistic regression and Voting models. While the scores achieved by these models were not greater than the tree-based models, they are above average and show consistency.

### 3.2.2 Analysis 2: Feature reduction

For this analysis, we will introduce feature selection methods to our experiment. These methods will reduce the metric for the data to fit each model. The idea behind the feature selection methods is that not all metrics are good indicators for faultiness of a software. These algorithms are used to reduce the metrics so that only useful metrics remain which will improve the performance of our program. There are several discoveries that were identified through testing which relates to this topic.

**Correlation-based feature selection**

The correlation-based feature selection is a supervised method which will rank attributes for a given dataset based on its subset’s correlation with the class label. For our algorithm, the user will be able to input the number of features to reduce to from a given dataset. Below are the average results of each model for CFS using the 8 datasets stated previously with varying feature reduction values:

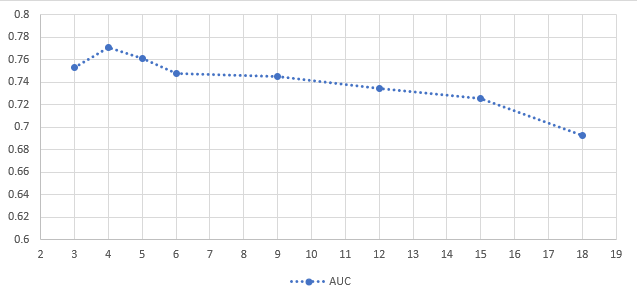
**Table 6: Average AUC results for CFS analysis**

| **Feature reduction** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** | **Average**  **model**  **score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 0.647 | 0.711 | 0.77575 | 0.740875 | 0.8295 | 0.792875 | 0.72825 | 0.800625 | 0.753234 |
| 4 | 0.680625 | 0.73425 | 0.79925 | 0.751375 | 0.828125 | 0.80825 | 0.75125 | 0.813 | 0.770766 |
| 5 | 0.689125 | 0.732125 | 0.791375 | 0.719375 | 0.827875 | 0.804125 | 0.715625 | 0.809625 | 0.761156 |
| 6 | 0.61625 | 0.72375 | 0.780375 | 0.6825 | 0.814 | 0.794125 | 0.754375 | 0.8185 | 0.747984 |
| 9 | 0.63875 | 0.72325 | 0.777625 | 0.623625 | 0.807375 | 0.8135 | 0.765875 | 0.809375 | 0.744922 |
| 12 | 0.70425 | 0.70075 | 0.7595 | 0.591125 | 0.787125 | 0.7875 | 0.73825 | 0.807375 | 0.734484 |
| 15 | 0.685375 | 0.692 | 0.778 | 0.557 | 0.76575 | 0.788625 | 0.728 | 0.80975 | 0.725563 |
| 18 | 0.590875 | 0.663 | 0.7375 | 0.581375 | 0.717875 | 0.7725 | 0.7065 | 0.768625 | 0.692281 |
| **Max** | 0.70425 | 0.73425 | 0.79925 | 0.751375 | 0.8295 | 0.8135 | 0.765875 | 0.8185 |  |
| **Min** | 0.590875 | 0.663 | 0.7375 | 0.557 | 0.717875 | 0.7725 | 0.7065 | 0.768625 |

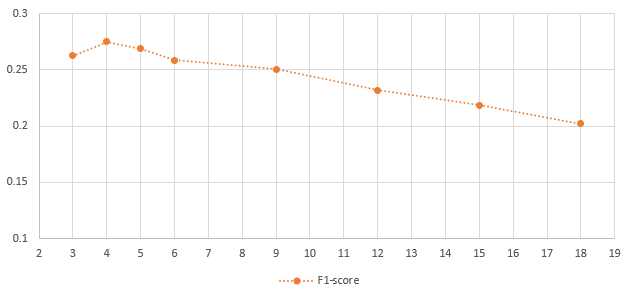
**Table 7: Average F1-score results for CFS analysis**

| **Feature reduction** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** | **Average**  **model**  **score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 0.2075 | 0.272625 | 0.222375 | 0.26225 | 0.26175 | 0.28775 | 0.27775 | 0.308625 | 0.262578 |
| 4 | 0.195375 | 0.291375 | 0.25125 | 0.299875 | 0.23875 | 0.303375 | 0.307125 | 0.312625 | 0.274969 |
| 5 | 0.190875 | 0.287375 | 0.236 | 0.295875 | 0.242625 | 0.312 | 0.2805 | 0.304875 | 0.268766 |
| 6 | 0.1775 | 0.289625 | 0.224625 | 0.229625 | 0.240125 | 0.291 | 0.327625 | 0.29 | 0.258766 |
| 9 | 0.1845 | 0.27025 | 0.234 | 0.214875 | 0.231 | 0.312625 | 0.282875 | 0.2735 | 0.250453 |
| 12 | 0.184 | 0.239 | 0.21625 | 0.19 | 0.213375 | 0.2955 | 0.26725 | 0.248875 | 0.231781 |
| 15 | 0.16725 | 0.233 | 0.22625 | 0.150875 | 0.2165 | 0.273875 | 0.244625 | 0.23725 | 0.218703 |
| 18 | 0.143875 | 0.22575 | 0.196125 | 0.14375 | 0.1905 | 0.265875 | 0.231375 | 0.221 | 0.202281 |
| **Max** | 0.2075 | 0.291375 | 0.25125 | 0.299875 | 0.26175 | 0.312625 | 0.327625 | 0.312625 |  |
| **Min** | 0.143875 | 0.22575 | 0.196125 | 0.14375 | 0.1905 | 0.265875 | 0.231375 | 0.221 |

From the results gained, we can observe that the CFS method works particularly well when the feature reduction value is below 6. Logistic regression along with all ensemble models showed consistent results when the feature reduction value varies. The Decision tree, Multi-Layer Perceptron and Naïve Bayes base predictors have a more visible effect with greater decrease in performance from a reduction value of 6 onwards. The Complement Naïve Bayes shows no visible pattern with random changes between each interval.



**Fig. 4 Line chart displaying the Average model AUC for CFS analysis**



**Fig. 5 Line chart displaying the Average model F1-score for CFS analysis**

From the charts shown, we can see that the performance peaks when the feature reduction value is between 3 to 6. Additionally, any value above 6 is shown to gradually decrease the overall score for both AUC and F1-score.

## 

**Recursive Feature Elimination**

The recursive feature elimination function is a supervised method which uses a recursive approach to remove the least important feature until a set number of features remain. Similar to the CFS method, our program allows us to configure the number of features to reduce. Below are the average results of each model for RFE using the 8 datasets stated previously with varying feature reduction values:

**Table 8: Average AUC results for RFE analysis**

| **Feature reduction** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** | **Average**  **model**  **score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 0.646125 | 0.653625 | 0.698 | 0.66175 | 0.68575 | 0.707625 | 0.676625 | 0.71525 | 0.680594 |
| 6 | 0.7145 | 0.669 | 0.785875 | 0.65825 | 0.73475 | 0.771 | 0.683375 | 0.77775 | 0.724313 |
| 9 | 0.778125 | 0.686875 | 0.777 | 0.735125 | 0.755 | 0.777625 | 0.710375 | 0.785125 | 0.750656 |
| 12 | 0.76925 | 0.731375 | 0.760375 | 0.71275 | 0.74675 | 0.797375 | 0.72175 | 0.783125 | 0.752844 |
| 15 | 0.7875 | 0.714875 | 0.786875 | 0.75475 | 0.7555 | 0.790625 | 0.7265 | 0.782125 | 0.762344 |
| 16 | 0.802 | 0.7205 | 0.77775 | 0.721 | 0.759125 | 0.8035 | 0.7315 | 0.78875 | 0.763016 |
| 17 | 0.78925 | 0.727625 | 0.76675 | 0.722 | 0.751125 | 0.797125 | 0.7275 | 0.77775 | 0.757391 |
| 18 | 0.78775 | 0.70025 | 0.806375 | 0.72525 | 0.746875 | 0.796875 | 0.714125 | 0.80075 | 0.759781 |
| **Max** | 0.802 | 0.731375 | 0.806375 | 0.75475 | 0.759125 | 0.8035 | 0.7315 | 0.80075 |  |
| **Min** | 0.646125 | 0.653625 | 0.698 | 0.65825 | 0.68575 | 0.707625 | 0.676625 | 0.71525 |

**Table 9: Average AUC results for CFS analysis**

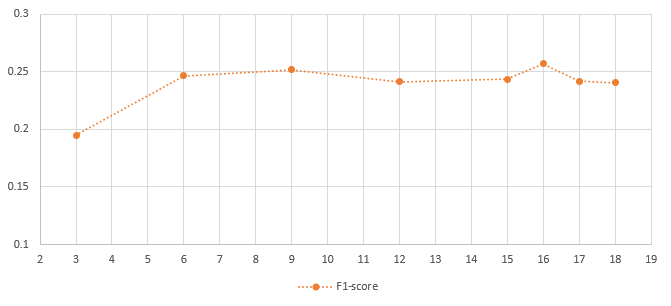
| **Feature reduction** | **Complement Naive Bayes** | **Decision Tree** | **Logistic regression** | **Multi-Layer Perceptron** | **Naive Bayes** | **Random Forest** | **Rotation Forest** | **Voting** | **Average**  **model**  **score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 0.181375 | 0.230375 | 0.113625 | 0.206875 | 0.162125 | 0.250875 | 0.20975 | 0.202625 | 0.194703 |
| 6 | 0.191 | 0.243625 | 0.244625 | 0.25625 | 0.207375 | 0.281125 | 0.276625 | 0.27075 | 0.246422 |
| 9 | 0.217125 | 0.246375 | 0.236 | 0.29125 | 0.214375 | 0.256375 | 0.2705 | 0.281875 | 0.251734 |
| 12 | 0.1855 | 0.260375 | 0.261625 | 0.242875 | 0.201125 | 0.27175 | 0.260375 | 0.245375 | 0.241125 |
| 15 | 0.196375 | 0.2535 | 0.272125 | 0.251625 | 0.202875 | 0.242375 | 0.246625 | 0.281 | 0.243313 |
| 16 | 0.20575 | 0.259375 | 0.28775 | 0.27625 | 0.196625 | 0.275 | 0.2585 | 0.297 | 0.257031 |
| 17 | 0.194375 | 0.2695 | 0.2345 | 0.231875 | 0.194375 | 0.264875 | 0.27825 | 0.267 | 0.241844 |
| 18 | 0.198 | 0.2435 | 0.253375 | 0.252 | 0.198 | 0.255375 | 0.259 | 0.263 | 0.240281 |
| **Max** | 0.217125 | 0.2695 | 0.28775 | 0.29125 | 0.214375 | 0.281125 | 0.27825 | 0.297 |  |
| **Min** | 0.181375 | 0.230375 | 0.113625 | 0.206875 | 0.162125 | 0.242375 | 0.20975 | 0.202625 |

Unlike the CFS method, a greater feature reduction value seems to show better results for all models, as observed in the tables above. Unlike CFS, only ensemble methods show consistent results. While the changes in performance are visible for the base models, there is less value difference between each interval. Hence, the acceptable range for better performance is much wider as compared to CFS.

Chart, line chart

Description automatically generated

**Fig. 6 Line chart displaying the Average model AUC for RFE analysis**



**Fig. 7 Line chart displaying the Average model F1-score for RFE analysis**

From the charts shown, we can see that the performance peak when the feature reduction value is between 15 to 16. Unlike the CFS method, an increase in feature reduction showed greater improvements towards the performance for both evaluation scores.

# 

### 3.2.3 Analysis 3: Feature selection method

This analysis uses the results from analysis 2 to determine the best feature selection methods for each model and finding the performance increase when compared to the results from analysis 1.

**Table 10: Results for comparison between feature selection methods**

| Model Name | CFS | | | | | | RFE | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AUC | | | F1-score | | | AUC | | | F1-score | | |
| Max | Min | Average | Max | Min | Average | Max | Min | Average | Max | Min | Average |
| Complement Naive Bayes | 0.7043 | 0.5909 | 0.6565 | 0.2075 | 0.1439 | 0.1814 | 0.8020 | 0.6461 | 0.7593 | 0.2171 | 0.1814 | 0.1962 |
| Decision Tree | 0.7343 | 0.6630 | 0.7100 | 0.2914 | 0.2258 | 0.2636 | 0.7314 | 0.6536 | 0.7005 | 0.2695 | 0.2304 | 0.2508 |
| Logistic regression | 0.7993 | 0.7375 | 0.7749 | 0.2513 | 0.1961 | 0.2259 | 0.8064 | 0.6980 | 0.7699 | 0.2878 | 0.1136 | 0.2380 |
| Multi-Layer Perceptron | 0.7514 | 0.5570 | 0.6559 | 0.2999 | 0.1438 | 0.2234 | 0.7548 | 0.6583 | 0.7114 | 0.2913 | 0.2069 | 0.2511 |
| Naive Bayes | 0.8295 | 0.7179 | 0.7972 | 0.2618 | 0.1905 | 0.2293 | 0.7591 | 0.6858 | 0.7419 | 0.2144 | 0.1621 | 0.1971 |
| Random Forest | 0.8135 | 0.7725 | 0.7952 | 0.3126 | 0.2659 | 0.2928 | 0.8035 | 0.7076 | 0.7802 | 0.2811 | 0.2424 | 0.2622 |
| Rotation Forest | 0.7659 | 0.7065 | 0.7360 | 0.3276 | 0.2314 | 0.2774 | 0.7315 | 0.6766 | 0.7115 | 0.2783 | 0.2098 | 0.2575 |
| Voting | 0.8185 | 0.7686 | 0.8046 | 0.3126 | 0.2210 | 0.2746 | 0.8008 | 0.7153 | 0.7763 | 0.2970 | 0.2026 | 0.2636 |

The red encoded text indicates the best results between the two selection methods. We determine the best method for each model using the table above by the count of red encoded data. The best feature selection method based on our analysis are as followed:

Correlation-based feature selection

* Decision Tree
* Naive Bayes
* Random Forest
* Rotation Forest
* Voting

Recursive feature elimination

* Complement Naive Bayes
* Logistic regression
* Multi-Layer Perceptron

**Table 11: Performance increase from the base and max score for each model**

| Model Name | AUC | | | F1-score | | |
| --- | --- | --- | --- | --- | --- | --- |
| Base | Max | % Increase | Base | Max | % Increase |
| Complement Naive Bayes | 0.574 | 0.802 | 39.72125 | 0.13675 | 0.217125 | 58.77514 |
| Decision Tree | 0.716875 | 0.73425 | 2.423714 | 0.25 | 0.291375 | 16.55 |
| Logistic regression | 0.744375 | 0.806375 | 8.329135 | 0.209625 | 0.28775 | 37.26893 |
| Multi-Layer Perceptron | 0.536875 | 0.75475 | 40.58207 | 0.116125 | 0.29125 | 150.8073 |
| Naive Bayes | 0.713125 | 0.8295 | 16.31902 | 0.185625 | 0.26175 | 41.0101 |
| Random Forest | 0.794125 | 0.8135 | 2.439792 | 0.276625 | 0.312625 | 13.01401 |
| Rotation Forest | 0.7365 | 0.765875 | 3.988459 | 0.25825 | 0.327625 | 26.8635 |
| Voting | 0.77675 | 0.8185 | 5.37496 | 0.2315 | 0.312625 | 35.0432 |

With the introduction of feature selection methods, the models shown to have great improvements towards the F1-score. There is also an increase in performance for the AUC of all models, with significant improvements for the Multi-Layer Perceptron and Complement Naïve Bayes models. As such, we can conclude that our proposed method has improved the overall performance of all models within the systems, with significant improvements particularly on models that by nature are not suited for handling imbalanced datasets.

### 3.2.4 Summary

From these analyses, we were able to examine various combinations of our program. As a result, this eventually led us to find the best method for fault prediction. This became our proposed method and it has shown promising results for handling imbalanced data. Section 3.4 will explain further on our proposed method, along with a comparative analysis with results from other research papers.

## 3.3 How are requirements met

In FIT3161, we listed several functional and non-functional requirements for our project. We retained all these requirements with a few additions which cover new features we decided to add to our program. These requirements are shown in Table 12 and Table 13.

**Table 12: Functional Requirements**

| **Requirement ID** | **Functional requirements** |
| --- | --- |
| R1 | The program can extract the software metrics within uploaded datasets correctly. |
| R2 | The program should ensure no duplicate and missing data is present after the data undergoes preprocessing |
| R3 | The program performs the correct feature selection method based on user selections |
| R4 | The program can undersample data correctly based on the criterias required |
| R5 | The program can identify the machine learning techniques selected as well as any configuration made by the user |
| R6 | The program can split test and training data appropriately based on the k value used during the k-fold cross validation |
| R7 | The program can build the prediction models appropriately and ensure each model were trained with the same training data |
| R8 | The program can perform all the required computation for every dataset and models selected within one run |
| R9 | The program can compile the results of every component to form the chart and result table |
| R10 | The program can store the results in a designated file path with a filename given by the user |

**Table 13: Non-functional Requirements**

| **Requirement ID** | **Type** | **Functional requirements** |
| --- | --- | --- |
| NR1 | Security | The program will only be accessible to authorized user |
| NR2 | Usability | The user interface should be user-friendly. |

**Table 13: Non-functional Requirements (cont)**

| NR3 | Usability | The program provides helper tools which contains sufficient information to guide the user |
| --- | --- | --- |
| NR4 | Usability | The program divides the result screen content appropriately to ensure the results are readable |

The following tables will explain how our program addresses these requirements.

**Table 14: Solutions for achieving the functional requirements**

| **Requirement ID** | **Status** | **Description** |
| --- | --- | --- |
| R1 | Achieved | Our system utilizes the loadarff function from the scipy library to correctly read any file with a arff type. |
| R2 | Achieved | At the end of preprocessing, the program checks every row within the data to validate and remove any empty data. Whereas the duplicate data will be removed during undersampling. |
| R3 | Achieved | The user interface is implemented to provide a selection menu which will allow users to select the feature selection methods they wish to perform for each dataset. |
| R4 | Achieved | Our system uses the instance hardness threshold method which undersamples the data based on how frequently they are misclassified. |
| R5 | Achieved | We added input sections in the user interface that allows users to provide values which will configure part of the algorithm, along with a selection menu which allows users to select their desired models to be built and evaluated by the system. |
| R6 | Achieved | We implemented our user interface to accept a user input which will determine the k-value that will be used for the k-fold cross validation in the main algorithm. |
| R7 | Achieved | We implemented our main algorithm to work on each training data in a sequential manner. This ensures that every model will be provided the same training data, in the same order. |
| R8 | Achieved | We have built our main algorithm with flexibility in mind. The algorithm will iterate through every dataset and train the appropriate models in each iteration. |
| R9 | Achieved | We implemented our result screen to have two tabs, allowing users to view the results in both tabular and chart form. |
| R10 | Achieved | The main user interface will allow users to input a filename, and the results of our program will be added in a folder called csv file within the main directory of our system. |

**Table 15: Solutions for achieving the non-functional requirements**

| **Requirement ID** | **Status** | **Description** |
| --- | --- | --- |
| NR1 | Achieved | For security, we added a login screen so only authorized users will be able to access the content of our program. |
| NR2 | Achieved | Through the usage of the tkinter package, we were able to create graphical user interfaces which provide interactive screens that are easy to interpret and provide a better user experience. |
| NR3 | Achieved | For all the input sections within our application’s UI, there will be tooltips shown to the user which provides sufficient explanation on various parts of the program to guide users when using our application. |
| NR4 | Achieved | To ensure that the results can be interpreted by the user, both views were organized to provide a pleasant view of our results for the user.  For the table view, the results of each evaluation metrics component are sectioned clearly with proper title arrangements, and encoded texts to improve user interpretation.  For the chart view of our results screen, the charts were split for each evaluation metric. Each metric will have its own chart that the user can choose to view. |

## 3.3 Justification of decisions made

There were a few major decisions made throughout the project lifecycle which are divided to separate subsections. Each subsection will have a description of these decisions, along with our justification for the final choices.

### 3.3.1 Prediction models

Prediction models are one of the major components within our system. Not every model is capable of handling imbalanced datasets, and some learning techniques are not suited for our field. While there are no constraints to the number of models we can add to our application, we felt that our focus and efforts should be directed towards achieving the main objectives. As such, we limited our selection to the appropriate models which have the highest potential in achieving our desired results.

Our final product had 5 base predictors and 3 ensemble predictors. The selection of prediction models was made mainly through research and factors such as availability and potential. Our research materials covered details on the properties of various models which allowed us to learn its strengths and potential. In addition, most of these prediction models were available in libraries such as scikit-learn.

### 3.3.2 Datasets

One of the key parts for our project is to be able to properly evaluate the performance of our fault prediction methods. In result, we require a selection of datasets which will allow us to analyze these methods. For this, we chose datasets from NASA and PROMISE repositories.These datasets are widely used for any research on this field, and were mentioned numerous times in our research articles. As such, these datasets were well suited for our project and also allows us to perform comparative analysis with other research results on this field of study.

### 3.3.3 Resolving imbalanced data

In the field of class imbalance, there are many known techniques that can potentially improve the performance of prediction models through processing the dataset before providing it as input. Strategies as such include data resampling and feature selection methods. However, not all techniques are compatible with one another. In essence, we need to decide on the techniques to include as not all of them will improve the performance on fault prediction. With that in mind, we were able to identify techniques that were used in other studies that are closely related to ours. These techniques include IHT undersampling, recursive feature elimination and correlation based feature selection. We decided on these techniques as they are shown to have the greatest potential towards improving fault prediction as shown from other research articles.

### 3.3.4 Performance metrics

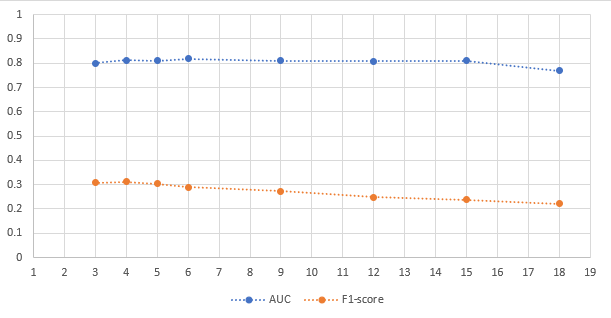
Similar to the datasets, the performance metrics are also an important component for a proper evaluation to be performed. Not every performance metrics will provide good insight on the performance of our models. As such, we chose performance metrics which measure qualities of models that are useful in the field of imbalanced datasets. Our choices of evaluation metrics are AUC, F1-Score, FNR and FPR. as these are commonly known metrics used to study the performance of models in the case of imbalance datasets and will allow us to perform comparative analysis with other research articles.

## 3.4 Discussion of Results

This section will cover details on our proposed method, along with our analysis on our method’s performance when compared to other results from other research articles.

### 3.4.1 Proposed method

From these experiments, we were able to study the nature of various models with various changes. As such, we were also able to identify the model which performs best which we would like to highlight in this section. This model being the Voting ensemble model.



**Fig. 8 Line chart displaying the Average score for Voting model**

From previous analysis, we were able to identify the best feature selection method for this model so this section will be focused only on this combination of model and feature selection. If we observe the chart in Figure 8, the voting ensemble model has shown the most consistent results. Additionally, the Voting model also achieved the second highest score on the average AUC and F1-score among all models. In essence, the Voting model outperforms other models in terms of consistency and achieving scores greater than the majority of the models available. This led us to decide that the Voting method is the best model for our system.

### 3.4.2 Analysis 4: Performance comparison with other algorithms

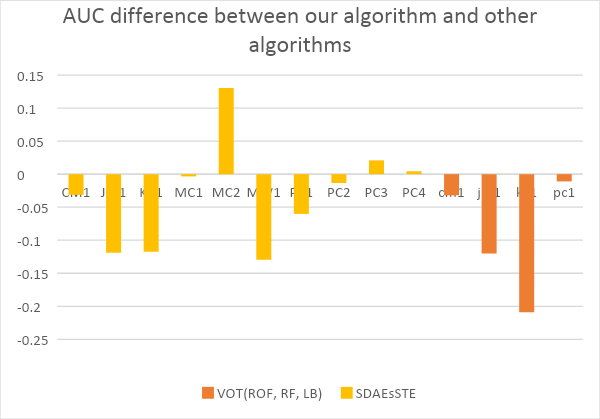
This analysis covers the performance of the best algorithm from our proposed system against other software fault prediction algorithms devised from other research papers.

**Table 16: AUC and F1-score results comparisons with other proposed algorithms**

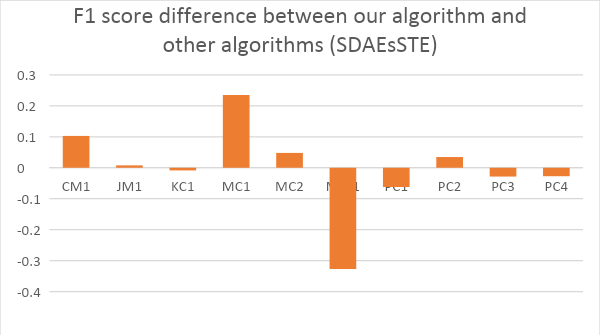
| **Datasets used** | Results | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Our Algorithm | | VOT(ROF, RF, LB)  (Yucalar et al.) | | SDAEsSTE (Tong et al.) | |
| AUC | F1 | AUC | F1 | AUC | F1 |
| CM1 | 0.806 | **0.391** | N/A | N/A | **0.8373** | 0.2882 |
| JM1 | 0.654 | **0.325** | N/A | N/A | **0.7731** | 0.3174 |
| KC1 | 0.625 | 0.386 | N/A | N/A | **0.7426** | **0.3946** |
| MC1 | 0.958 | **0.457** | N/A | N/A | **0.9614** | 0.2217 |
| MC2 | **0.815** | **0.635** | N/A | N/A | 0.6846 | 0.5871 |
| MW1 | 0.73 | 0.08 | N/A | N/A | **0.8597** | **0.4073** |
| PC1 | 0.846 | 0.244 | N/A | N/A | **0.9062** | **0.3062** |
| PC2 | 0.913 | **0.25** | N/A | N/A | **0.9264** | 0.2154 |
| PC3 | **0.859** | 0.326 | N/A | N/A | 0.8381 | **0.3545** |
| PC4 | **0.889** | 0.527 | N/A | N/A | 0.8846 | **0.5544** |
| cm1 | 0.752 | 0.264 | **0.784** | N/A | N/A | N/A |
| jm1 | 0.633 | 0.285 | **0.753** | N/A | N/A | N/A |
| kc1 | 0.636 | 0.291 | **0.845** | N/A | N/A | N/A |
| pc1 | 0.865 | 0.354 | **0.876** | N/A | N/A | N/A |

After evaluating the performances of each of our algorithms, we then selected the best performing algorithm for comparison with other algorithms from other research papers, such as the SDAEsSTE approach devised by Tong et al. The algorithms are compared based on the AUC and F1-scores results which were present on the research paper themselves, and the data is tabulated based on the various datasets which were used on the algorithms. The comparisons can be found in the table at the results section. The table above shows our results when compared against other algorithms from other research papers. To better visualise the results of the analysis, graphs have also been created, and they can be viewed in the next page.

# 



**Fig. 9 Bar chart displaying the differences in performance of our algorithm and other algorithms in different datasets for the AUC performance metric**



**Fig. 10 Bar chart displaying the differences in performance of our algorithm and other algorithms in different datasets**

From the results above, we can see that our algorithm outperforms Tong et al.’s (2018) SDAEsSTE algorithm in certain datasets. On the other hand, our algorithm is not able to outperform Yucalar et al’s algorithm which is a combination of ensemble predictors.

### 

### 3.4.3 Results summary

From this analysis, we were able to determine the optimal configurations for achieving the best results from every model. The experiments performed have revealed valuable information on the nature of each model and its changes with the introduction of feature selection methods. The results also led us to identify the best configuration for our system, that is the Voting model with the CFS feature selection method. Though that is said, our proposed method lacks slightly in performance when compared with the two other algorithms mentioned in this report. Despite this being the case, we have come to the conclusion that although our best algorithm is not the best in terms of performance, it is still a decently viable method to use when a dataset is found to be imbalanced, as parts of our algorithm are built to handle those situations.All in all, we felt that our resulting method was a success.

## 3.5 Limitations of project outcomes

### 3.5.1 Product limitations

Our final product is a complete system, but there are a few limitations, mainly due to complexity and time constraint. Firstly, the system doesn’t allow users to change the hyperparameters of the prediction models. These hyperparameters are fixed internally using values we set. As such, manual changes to the back-end of the system will be required. Next would be datasets handling, our system is only capable of handling datasets that are in arff format. This is because the majority of datasets available are written in this format, but there are other file formats used such as csv. Lastly, the progress of the main algorithm is not shown in the interface. When the main algorithm is executed, there is no indication of its progression rate so the user will not be informed on the remaining time required before completion.

### 3.5.2 Results limitations

For the results, we felt that we could have included more in depth analysis but were restricted due to time constraints and availability of resources. Our current analysis only covers part of the datasets available. There are some datasets which we excluded in our experiments because of time constraints. In addition, our analysis only examines the two main evaluation metrics AUC and F1-score. As a result, the FPR and FNR scores were excluded from our analysis.

## 

## 3.6 Improvements & Possible Future Works

### 3.6.1 Improvements

While our product does meet all the requirements, there is still room for improvements. One major improvement that can be made would be on the flexibility of our program. As of now, there are only a few parts in our program that can be configured by the user. There are a few parts of the program that should be made configurabled hyperparameters of models as well as the inclusion/exclusion of the IHT undersampling. The user interface should allow further configurations to be made as such configurations will allow more concise experiments and may potentially lead us to the discovery of a better fault prediction method than our current proposed method.

Another improvement that can be made is to allow the models used by the voting model to be changeable.The voting ensemble model is the best model, having achieved great results for every experiment we conducted. As such, we view voting to be the model with the most potential towards gaining better results. Currently the voting model consists of 3 models that are set internally, these are Logistic Regression, Random Forest and Naive Bayes. The user should be allowed to change these models to other models for the chance of finding better methods for handling imbalance data.

### 3.6.2 Future work

For future work, one possibility would be the implementation of an algorithm which can extract features from any given modules. Currently, our program is not capable of directly testing software modules that are provided by users. Our current implementation can only accept inputs in a form of datasets. So for users to test their software, they will need to manually examine and record the software metrics of their modules, and convert these details into a dataset. With that in mind, automating the process of extracting software features will be a great addition to our product as it eliminates the need for datasets to be manually produced by the user. In addition, this algorithm would also be a great contribution as it may well improve the rate of progression for this field of study.

Another possible future work would be the creation of a web and mobile version of our application. There are several advantages to creating these variations of applications, mainly accessibility and performance. Currently for users to use our application, they will need to install many python packages before usage as mentioned in our technical guide. A web and mobile application will eliminate these needs which makes our application more accessible. Additionally, if privacy is not a concern by the user then a web application may allow users to upload their dataset to have its calculations performed in a remote computer set up by us which will be built for handling our main algorithm’s computations. As such, the capacity of their available machines will no longer be of concern as they can utilize machines that we set up to perform the program’s computation for them.

Besides that, another possibility would be the inclusion of other metrics. Throughout the project, our program has only been exposed to two types of metrics, these being static code metrics and object oriented metrics. With that in mind, it is possible for other metric types to measure other important qualities of a software, and potentially serve as better defect indicators. Two metric types that we believe may potentially improve the fault proneness detection of our software are functional and component-based metrics. As of now, we are unsure whether our current system is suited to work on these metrics. However, we believe that this can be achieved with a few modifications to our program and may lead us to the discovery of alternate options for fault proneness detection.

## 3.7 Critical discussion on project outcome

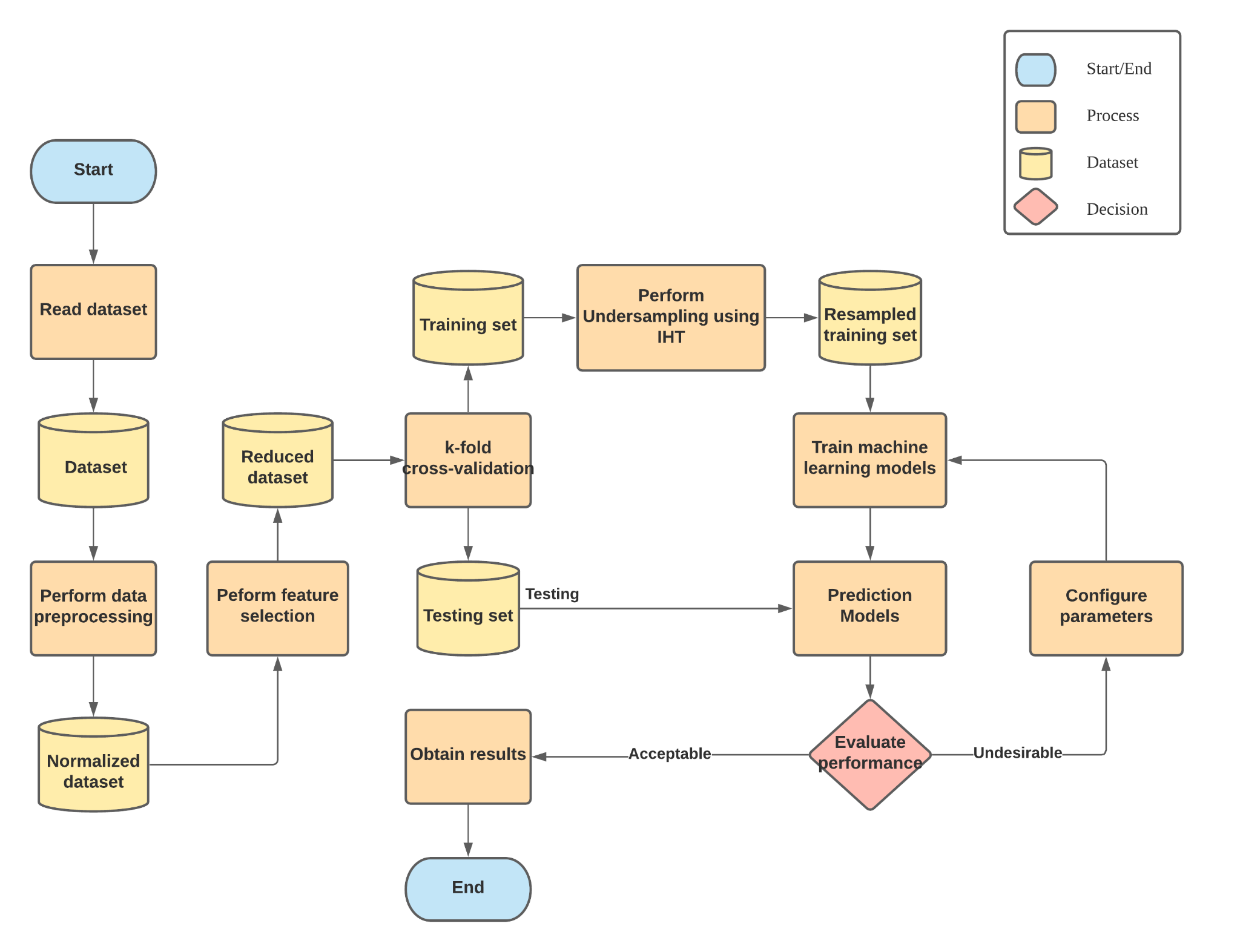
Overall, we felt that our project was a great success. As shown previously, we were able to achieve all our requirements. After performing several evaluations, we began to see the potential performance of the models, which exceeded our initial expectations towards the software we developed. Additionally, we could also see that our best algorithm was outperforming some of the proposed methods from the research papers for certain evaluation metrics. Combined with its ability to handle the issue of data imbalance, this greatly contributes to the reasoning as to why we think this project was successful.

However, we do feel that our software has some room for improvement as well. For example, our voting model, which is our best performing model, could be tuned to use an ensemble of models which have a greater overall performance, so that the performance of the voting model is also increased. Aside from that, we feel that certain parts of the algorithm can be changed to allow more user configuration, so that users can potentially find a better performing algorithm through their configurations. As there was little time remaining to complete the project, we have decided to label these potential improvements as future improvements or limitations.

# 

# 4. Methodology

## 4.1 System architecture



**Fig. 11 System flowchart**

The chart shown in Figure 11 describes our system’s flow, showing the steps, events and interactions between each process in sequential order. As shown, the dataset will undergo several processing before being used to train the models.Our system consists of 3 main components which will be elaborated further in the following sections.

## 

## 4.2 Data preparation

Before the dataset is ready for usage, there are a number of processes required to be performed. Our program promotes flexibility so we allow users to select and configure part of the algorithm. As such, we implemented our functions with parameters that allow the behavior of the algorithm to be configured for preparing the data.

**Pseudocode of data preparation functions**

| **function** preprocess(filename):  extract the dataset  perform Normalisation  remove missing data from the dataset  remove outliers  **return** dataset  **end function**  **function** feature\_selection(dataset, fs\_selections, reduced\_size, k\_fold):  **for** fs **in** fs\_selections **do:**  perform feature selection fs  obtain training-test data split using K-fold  **for** training\_data **in** training-test data split **do**:  perform IHT undersampling to training\_data  add training-test data split to result\_list  **end for**  **end for**  **return** result\_list  **end function** |
| --- |

The pseudocode above describes the algorithm which utilizes the items in this section to prepare the data.The preprocess() method as its name states will extract the data from the given file to perform preprocessing on its content before being passed to the next function. Whereas the feature\_selection() method is used to generate multiple sets of training-test splits to be used to train the models. The number of sets are based on the number of feature selection methods the user chooses. One thing to highlight is that the result contains the training-test splits of all versions of the datasets for each feature selection method.

The individual processes are explained further in the subsections that follow.

### 

### 4.2.1 Data reading/Preprocessing

Datasets will be provided by the user and will have its content extracted appropriately by the program. This was done using the loadarff function from the scipy.io library which reads the file’s content and produces an array containing all the metrics and labels. After this, the labels will be checked as datasets will have different labels due to different labeling conventions. In our system, all the libraries interpret true labels as 1 and false labels as 0. As such, we implemented a code

Before the dataset is used, the extracted data will undergo preprocessing. This includes normalization and the removal of missing data. Table 17 shows the functions within our system which performs the tasks stated.

**Table 17: Data reading/Preprocessing functions**

| **Function name** | **Description** | |
| --- | --- | --- |
| read\_data() | **Parameters:** Filename/File address  The function will search for a file with the name stated in its parameter. If the filename is given then it will search for files within the same directory. Whereas if a file address was given then it will search for the file in the given location. | |
| data\_conversion() | **Parameters:** Array of labels after extraction  The function iterates every label and converts them to appropriate values (0,1) | |
| Normalize() | **Parameters:** Data array  The function will both normalize the data array given and remove any missing data found. | |

### 4.2.2 Undersampling/K-fold cross validation

After the datasets undergo preprocessing, the preprocessed data will be undersampled based on the instance hardness threshold. This method will balance the dataset through removing data based on how frequent the data was misclassified . An implementation of this technique can be found in the imblearn library, which was also used in our system.

For our program to evaluate the performance of every model, the dataset will be partitioned into training/test splits. To do so, we implemented our system to perform K-fold cross validation on the dataset which will produce a number of training/test splits based on the k value provided. The table below will describe the function within our program which performs the undersampling and k-fold cross validation.

**Table 18: Undersampling and K-fold function**

| **Function name** | **Description** | |
| --- | --- | --- |
| IHT() | **Parameters:** Data, cv  The function will filter the dataset based on the IHT value of each data. The cv parameter determines the number folds used by the function for estimating sample instance hardness. | |
| K-fold() | **Parameters:** Data, folds  The function iterates every label and converts them to appropriate values (0,1) | |

### 4.2.3 Feature selection

In our project proposal for FIT3161, we initially designed our system to allow users to manually select the software metrics they want to include or exclude from the dataset. However we later learnt how impractical this approach was as every dataset has a large quantity of software metrics. As such, we decided to include feature selection methods to our system to filter the software metrics so only the important metrics will remain. For our project, we included two feature selection methods which are correlation based feature selection and recursive feature elimination.

An additional note is that our system also allows the option to use all the metrics which is the initial dataset before performing the feature selection.

**Table 19: Feature selection functions**

| **Function name** | **Description** | |
| --- | --- | --- |
| cfs\_algo() | **Parameters:** Dataset, Number of software metrics to reduce to  Performs correlation based feature selection to filter the metrics in a given dataset, the second parameter determines the number of metrics that should remain in the dataset after filtering | |
| rfe\_algo() | **Parameters:** Dataset, Number of software metrics to reduce to  Performs recursive based feature selection to filter the metrics in a given dataset, the second parameter determines the number of metrics that should remain in the dataset after filtering | |

## 

## 4.3 Main algorithm

After the data is processed using the techniques in section 4.2, the data will be ready for building the prediction models with. The main algorithm of our program will use the prepared data to test each model when fed the processed datasets.

**Pseudocode of the main algorithm**

| **function** main\_algorithm(filename, fs\_params, model\_selections):  dataset = preprocess(filename)  fs\_res = feature\_selection(dataset, fs\_params)  **for** training\_test\_splints **in** fs\_res **do:**  **for** train\_data, test\_data, train\_label, test\_label **in** training\_test\_splits **do:**  fit selected models using the train\_data  **for** model **in** selected\_models **do:**  perform predictions on test\_data  evaluate the models based on prediction  add evaluation\_scores to results  **end for**  **end for**  **end for**  **return** results  **end function** |
| --- |

The pseudocode above describes the main algorithm within our program. It was designed to be capable of configuring parts of its computation based on the user’s input. The main\_algorithm() includes the two functions shown in section 4.2 to perform the required computations on the dataset given and produce the sets of training/test splits. The function’s last parameter is used to determine the models to build. The results returned is an array of evaluation scores, with the dataset and feature selection used indicated in the first column. The result will also be used to produce the csv output file.

The following subsections will provide details on the models within our program and functions used for calculating the evaluation metrics.

### 

### 4.3.1 Prediction models

In our program, we included various machine learning models to be built and tested in our system. The base models are all available in the sklearn library which we used to create individual functions. Below are lists of models added into our program, categorized by type:

List of machine learning models (base)

1. Multi-Layer Perceptron
2. Logistic Regression
3. Naïve Bayes
4. Complement Naïve Bayes
5. Decision Tree

List of machine learning models (ensemble)

1. Rotation Forest
2. Voting
3. Random Forest

### 4.3.2 Model evaluation

Initially, we only included two evaluation metrics that were stated in our project proposal for FIT3161. However, we later decided to add two new evaluation metrics. So our final product includes a total of 4 evaluation metrics, these being AUC score, F1-score, False Positive Rate and False Negative Rate. The sklearn library has implementations which allows us to retrieve these scores. For the FPR and FNR in particular, we obtained them using a confusion matrix and performing the required calculations using the TP, TN, FP and FN values. Below shows a table which describes the functions in our program that are used to obtain these values.

**Table 20: Evaluation methods**

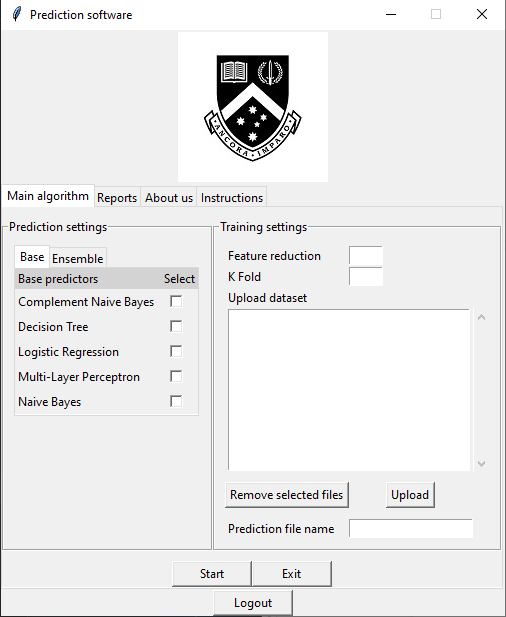
| **Function name** | **Description** | |
| --- | --- | --- |
| auc\_roc\_model() | **Parameters:** Model, test split samples, test split labels  Performs predictions using the test split samples and obtained the AUC score based on the predicted labels and the correct labels provided. | |
| f1\_model() | **Parameters:** Model, test split samples, test split labels  Performs predictions using the test split samples and obtained the F1-score based on the predicted labels and the correct labels provided. | |
| confusion\_matrix\_model() | **Parameters:** Model, test split samples, test split labels  First, the confusion matrix will be used to map the values based on the predicted labels against the correct labels. Them=n calculations are then performed using Equations (1) and (2) to obtain and return the FPR and FNR values  FPR = FP/TN+FP (1)  FNR = FN/FN+TP (2) | |

## 

## 4.4 User interface

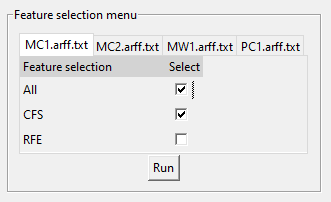
Tkinter is a standard GUI library that we used to build all the user interfaces within our system. There are several interfaces within our program, each having several tabs or sections which allow users to easily navigate through every part of our application. We implemented our application using a modular approach, so a module class is created for each tab, sections and pages in our application.

The main.psx function is the main executable in our program which starts our application and handles the interface to show the user. It imports all the interface classes and controls the order and structure of which interfaces are presented.A detailed description of every interface can be found in our User Guides. So, we will proceed with explaining how the interface and our system’s main algorithm interconnects.



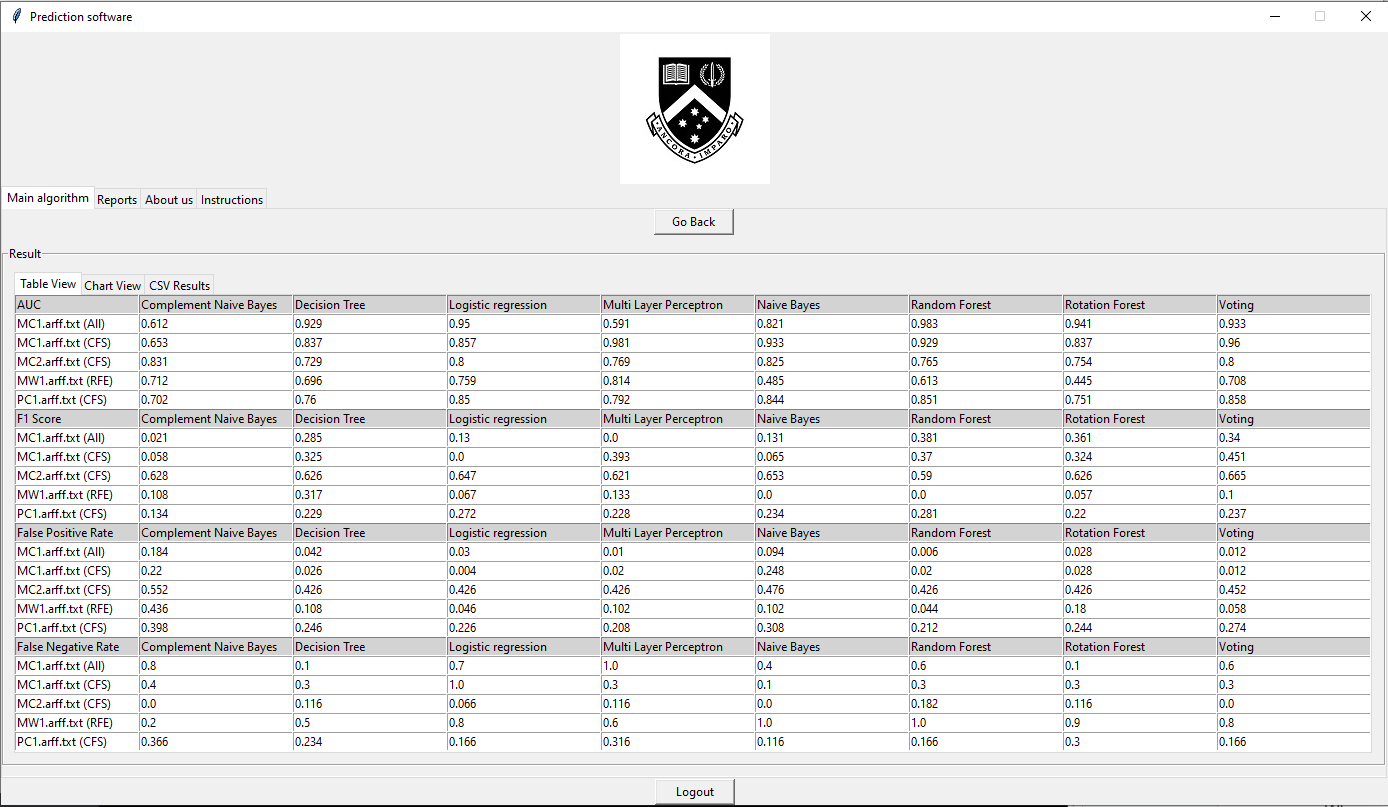
**Fig. 12 Configurations menu**

Firstly, any of the models selected by the user will be the models that are built in the main program. This is an optimization technique where we only build selected models instead of all the models to save time and space. Next, the feature reduction entered by the user will be the number of features to be reduced during the preprocessing phase. The K fold will be the number of folds to split the training and testing dataset. Datasets are the raw information to be fitted into our models and a new model will be built upon it. The more datasets uploaded which means more data to be fitted into the model, the better the accuracy will be.



**Fig. 13 Feature selection menu**

The second screen provided to the user will allow the users to select the feature selection methods they want to perform for each dataset. These selections will determine the parameter set for the feature selection function shown in section 4.2.



**Fig. 14 Table result**

Once the user runs the program, the main algorithm will be executed with the configurations set by the user. The results gained from the main algorithm will be used by the user interface to produce the table and chart views within the result screen. Figure 14 shows a sample of this table view. Additionally, the filename input the user set will be used as the csv output file’s name that can be found in the csv\_results folder in our main program folder.

# 

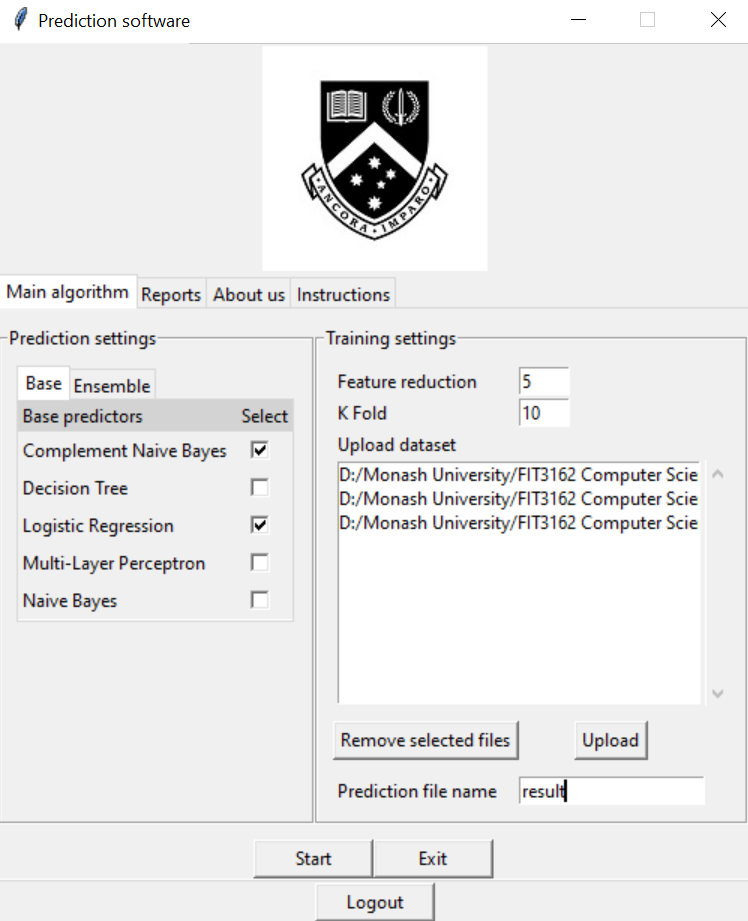
# 5 Software Deliverables

## 5.1 Summary of software deliverables

### 5.1.1 What is delivered

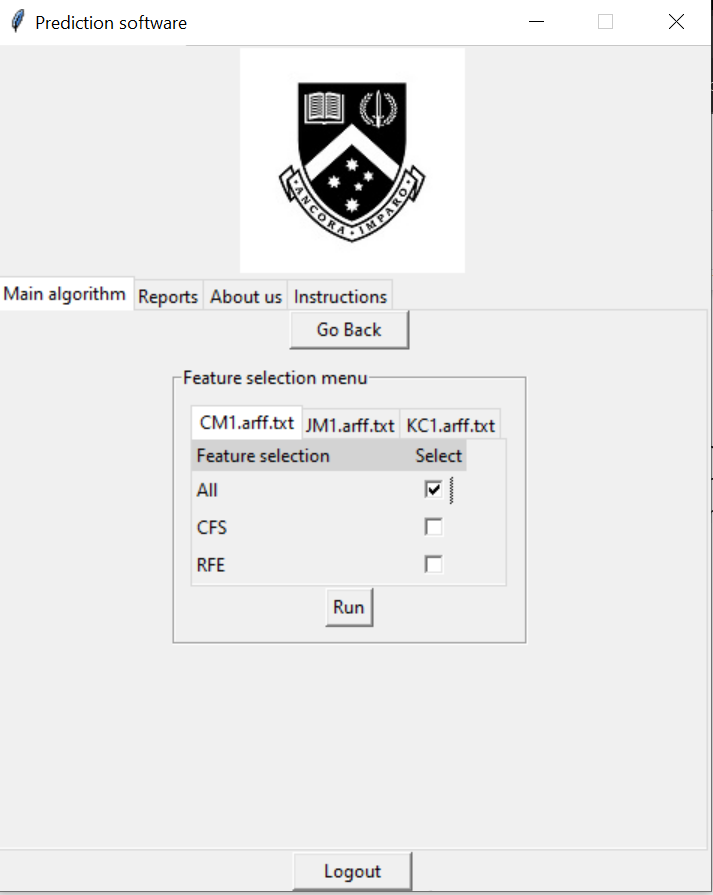
The final product for our project is a software application that can predict the fault proneness of software modules using machine learning algorithms. The source code for our program is stored in our Github repository. Inside the repository, you will find multiple folders relating to the project. All the code which builds the application is stored inside the Programs folder. The Testing folder contains the results and the code for unit testing, integration testing, system testing and user acceptance testing.

### 5.1.2 Sample Screenshots and Descriptions of Usage



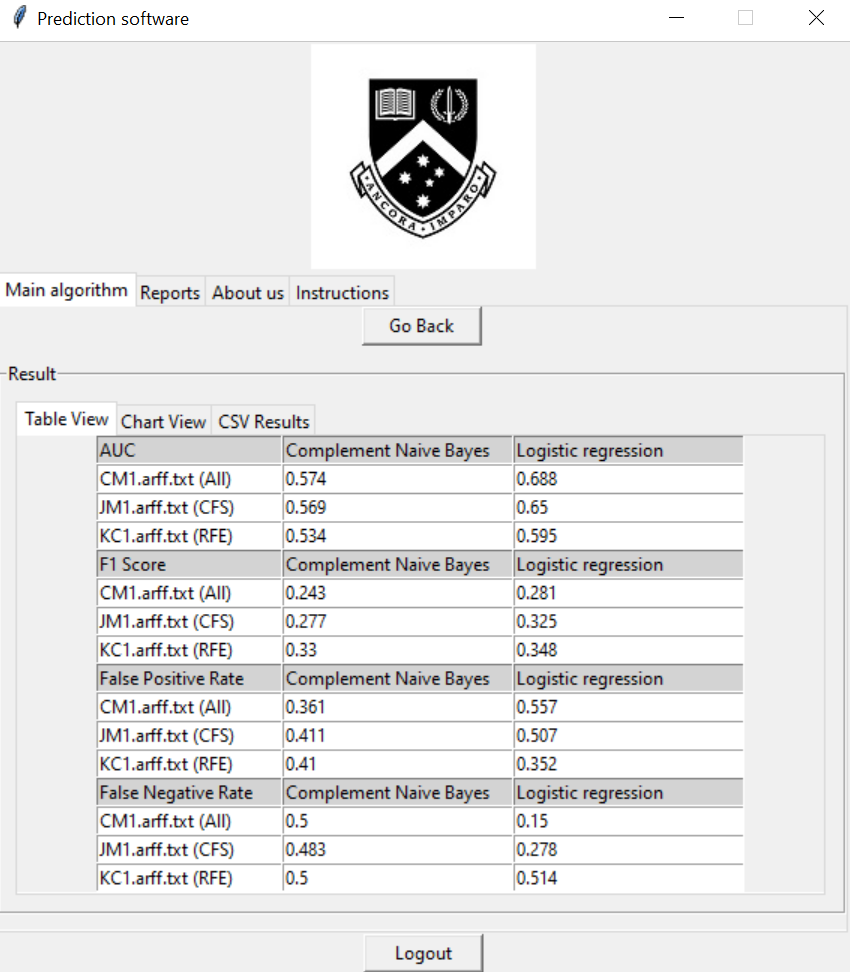
**Fig. 15 Home Screen Page**

This is the application’s home screen page where users are able to select the models as well as entering the values for feature selection and k fold. There is also an option to upload the dataset which will be processed and fed into the models. At least one model needs to be selected before the program can be run.

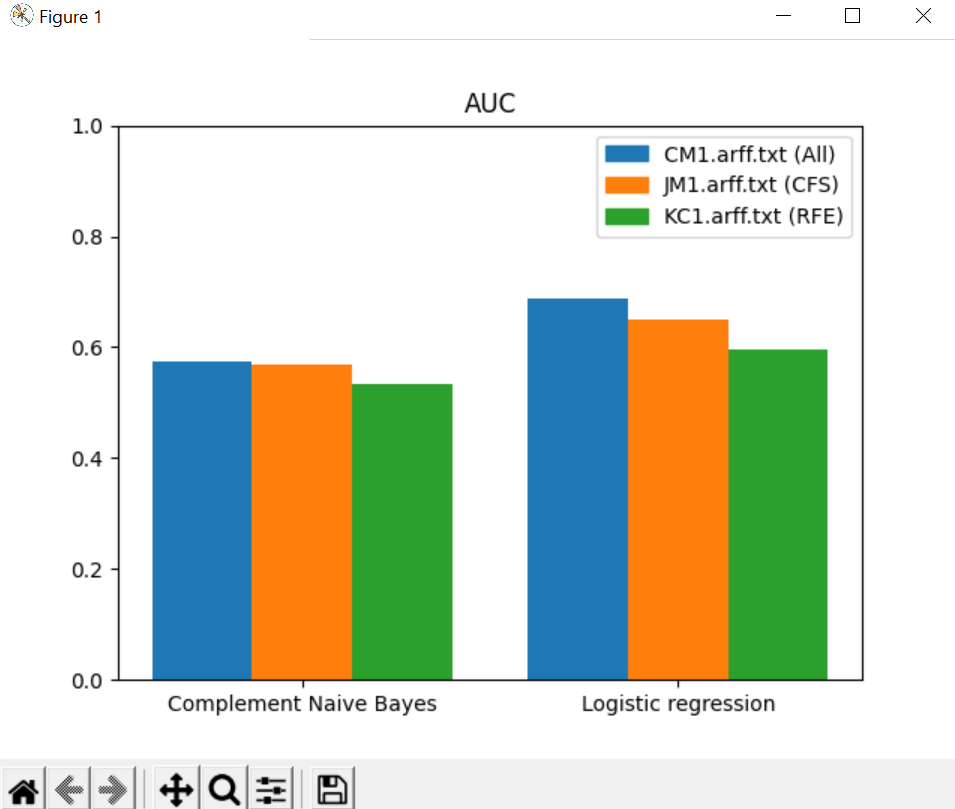


**Fig. 16 Feature selection menu**

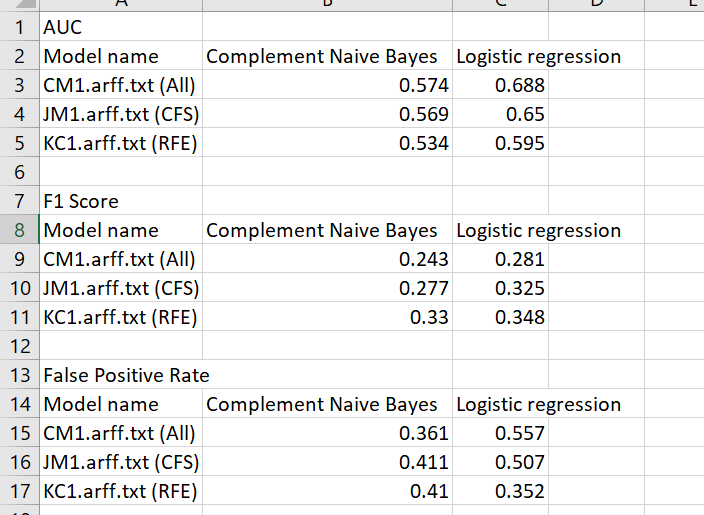
The feature selection menu is where users will be given to select their preferred feature selection method. Users are given the option to select all features or use CFS or RFE feature selection method. At least one tick must be selected for each dataset the user has uploaded.



**Fig. 17 Table View of results**



**Fig. 18 Chart View of results**

**

**Fig. 19 CSV form**

After selecting the features, the main program will begin the computation using the parameters defined by the user.

Once the processing is completed, the results will be displayed in 3 forms.

The tabulated view will show the exact results of the computation and will be ordered by each evaluation metric. A graph view is provided for visualisation of the results and allows users to identify patterns or make comparisons easily. Lastly, the results of the computation will be saved to a csv file for future reference.

## 5.2 Summary and discussion of handling software qualities

### 5.2.1 Documentation and Maintainability

Our team has thoroughly documented all the functions by specifying the purpose along with the parameters and return output of each function. Any complex code structures that may not seem obvious or require understanding are also commented to allow other users to know the meaning behind it. As time passes by, our team as the developers will forget the reason for writing those specific chunks of code and having comments will benefit us greatly. However, our team also believes that having too many comments is considered a code smell as some code that is obvious at first glance will not be commented on. By only commenting on complex code blocks, the depth will be more concise instead of full of comments.

In terms of maintainability, our team has successfully managed to modularize the code into functions and sub functions. Any repeatable part of the code is segregated and placed into its own function. This ensures that the core parts are not easily susceptible to modification and it allows for extensibility. Another advantage of modularizing code is that it makes debugging much easier since the same part of code is only appearing in one place instead of many places.

### 5.2.2 Security

In our program, all user inputs are thoroughly sanitised and reject any invalid types of input. The program does not store user’s information so there is no risk of exposing private information when any user uses our program. When uploading datasets, only two types of formats are accepted which are csv and arff files. This prevents any malicious scripting files from being accepted into our program which may potentially cause damage to it.

### 5.2.3 Robustness

All inputs that are invalid or out of the accepted range will cause an error message in the form of a pop up text box to appear. This ensures that the functions inside will not incur any error due to faulty parameters. Also, our program is able to detect if any dataset uploaded contains the required software metrics and labels. This helps to remove empty files or files with nonsense content from passing through.

The program is built upon using the fail-safe principle where if any fault were to occur, the system will respond in such a way that minimizes harm. We implemented the principle by introducing error handling techniques to combat all types of possible errors that may occur.

### 5.2.4 Usability

Our program uses a Graphical User Interface to make it more user friendly as well as making the program easier and comfortable to interact with. Not only that, informative tooltips are presented in sections where users are required to type. We also included an instructions tab in the program in case users are in need of guidance. However, we recommend users to read the user guides as it provides a step-by-step walkthrough of the program. When a user enters an invalid input, error messages will pop out to inform of the error in layman’s terms as well as providing a solution to the error.

### 5.2.5 Scalability

In terms of scalability, our program is not designed to handle extremely large datasets due to the number of lines it needs to read. As every line needs to be read and processed, it’s computation time will be longer, the larger the dataset is. Given that our code is modularized, it is able to integrate into larger scale programs as the functions are easily extensible and can be reused.

## 5.3 Sample Source Code in Appendix

Sample source code relating to the project can be found in the Appendix.

[Sample Source Code](#_f22scei47s7q)

# 

# 6 Critical Discussion

6.1 Discussions on the software developed for the project

During the initial phase of the project, we felt there was nothing much needed to be changed, and that we just had to follow what was written in our project proposal for Semester 1. However, as we started to develop our program, we began to understand more about how the final software should come about, and in turn this led us to slightly deviate from certain plans made in the project proposal.

To elaborate, we did feel that our initial plan for the project which was laid out in our proposal was already sufficient. Despite that, the development phase gave us an eye opener to certain improvements that could be made as we were creating the final software. For instance, our initial idea for having software metrics selected by users was impractical for larger datasets, which led us to adding implementations such as the correlation-based feature selection (CFS) and recursive feature elimination (RFE) methods to our software, which helps automate the process of selecting the best software metrics for the user.

Additionally, us beginning early development of the software during the semester break gave us more time to think about potential improvements as well. By doing this, it led us to think about the flexibility of the software itself, in which more configuration which can be done by the user would also mean a greater chance to obtain better evaluation results for the prediction models. This became evident when we were documenting the evaluation report, as we found that the configuration parameters for feature reduction did indeed change the overall performance of the models.

All in all, while there were certain changes to certain implementations in the final software, it did not necessarily hinder the accomplishment of fulfilling all the software deliverables specified in our project proposal.

6.2 Analysis on Project Success

Overall, we felt that this project was a huge success as we were able to achieve great results from our project and were able to meet all our objectives. In between the development phase, we had a good view on what tasks we should complete to ensure we were on track for the project completion. Although we did deviate from the plans highlighted in the project proposal by implementing the improvements stated in the previous section, we were still able to progress at a steady rate whilst fulfilling the software deliverables stated in the project proposal. Here are the few reasons as to why we were able to do so.

Firstly, we feel that we had good project management, as each of us adhered to the Agile Methodology. With its iterative and incremental nature, we were able to implement the additional improvements to the software (as stated in 7.1) without it affecting the overall progress towards completion. With each weekly meeting held acting as a sprint review, we were able to receive constructive feedback from our supervisor each week which helped us to understand our tasks better, improving the way we implement parts of the program. Additionally, with the meeting minutes made, we were able to easily track our progress each week and recall any events that occurred in previous weeks.

Aside from that, we also feel that we had good countermeasures for certain risks that occurred for this project. To explain more on this, we actually did decide to begin development early in order to curb any potential future risks that would halt our progress for the project. This became especially important when we had assignments clashing with our weekly scheduled tasks, rendering us unable to complete certain tasks within the given time frame. Since we started development early, we were able to be ahead of schedule by roughly two weeks. As such, we were comfortable with reducing the tasks for certain weeks as we were confident that we could still complete our project by the designated deadline.

To conclude, these are some of the strong factors that contributed to our project’s success, and we will continue to keep these factors in mind should it be needed for future projects to come.

# 

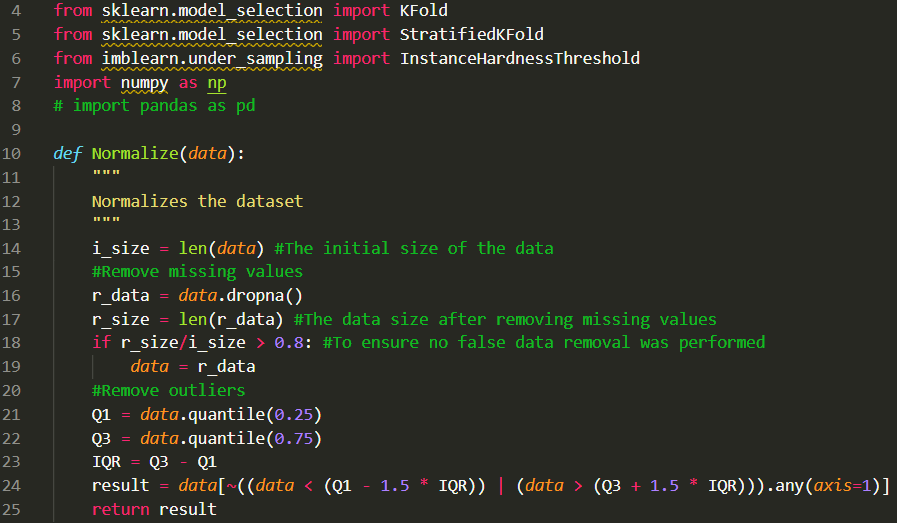
# 7 Conclusion

In conclusion, with softwares becoming larger and more complex each day, it becomes increasingly crucial for us to make sure that the budgets for testing these softwares are lightened. With the ample amount of research done on our project topic, we have managed to produce a software that is capable of building software fault prediction models which can potentially help to curb the costs of the testing phase by detecting software modules which are more likely to be defective. On top of that, our software also contains the appropriate algorithms which help to tackle the issue of data imbalance within datasets, which helps to boost the performance of the prediction models.

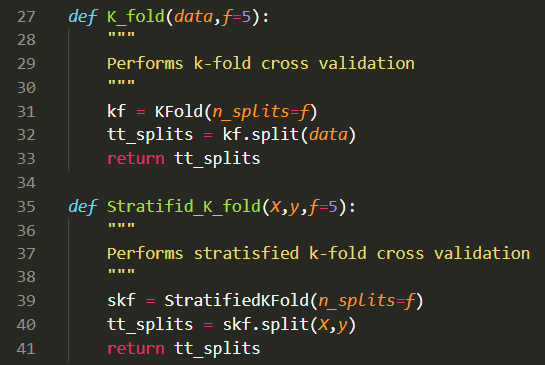
However, we are aware that our software has some room for improvement, such as the inclusion of the progress bar, as well as allowing more configurations available to increase the flexibility of our main algorithm. Despite this, we still feel that the final product of this project satisfies the objectives specified in the project topic, which makes us satisfied with our overall efforts.

# 8. Appendix

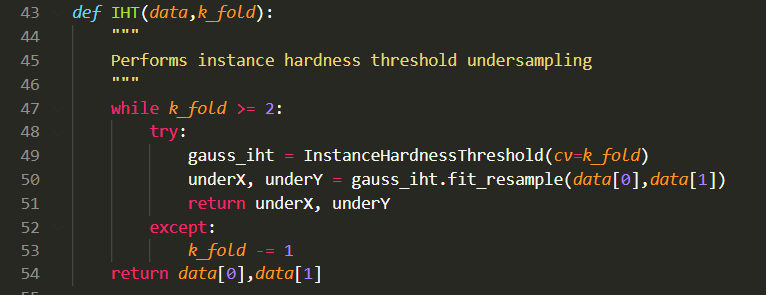
## 8.1 Normalization



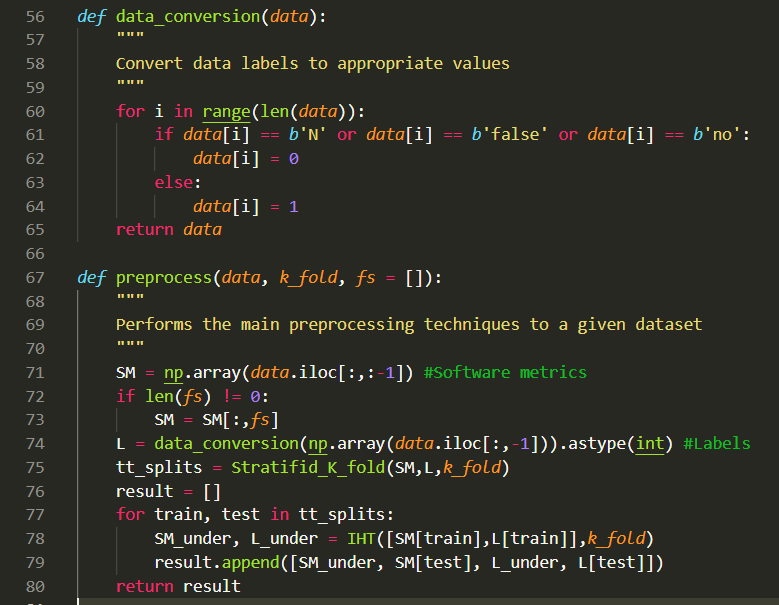
## 8.2 K Fold and Stratified K Fold



## 8.3 IHT

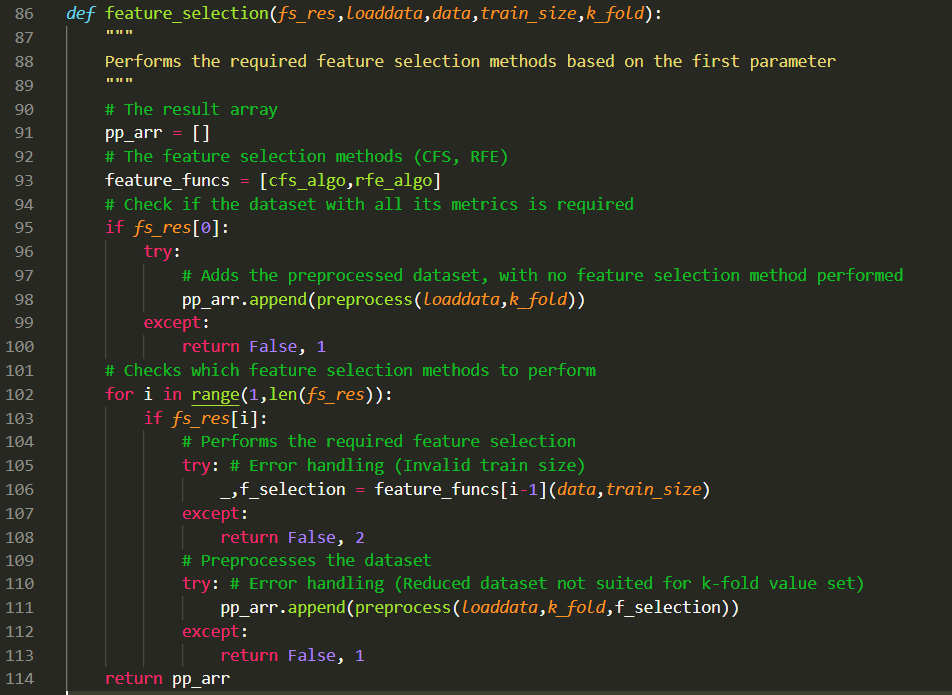


## 8.4 Preprocess

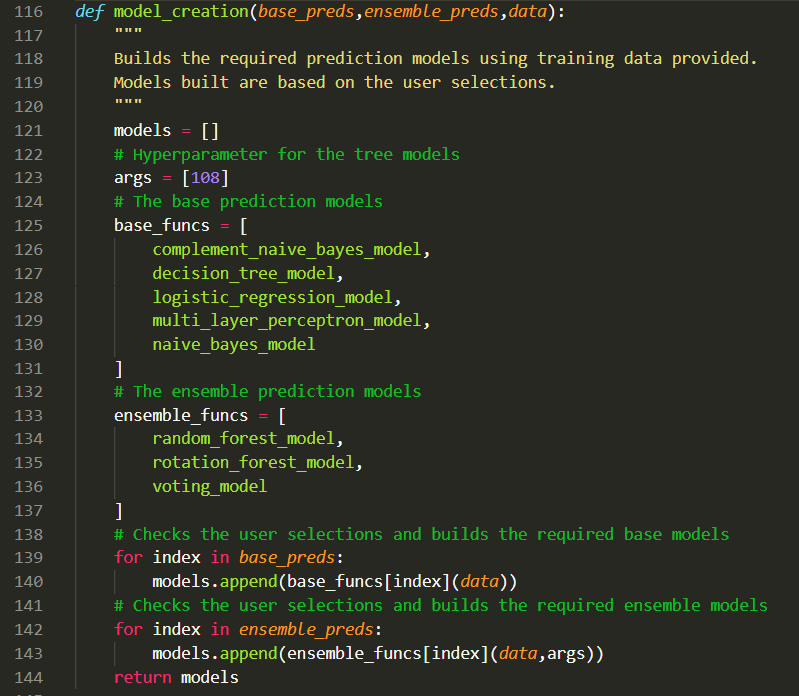


## 

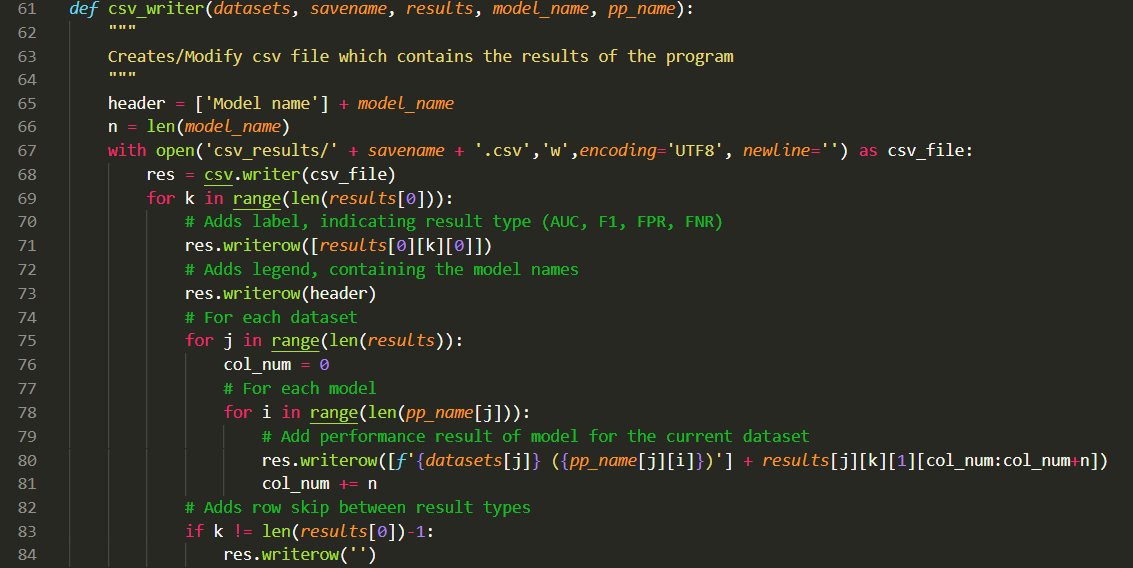
## 8.5 Feature Selection



## 8.6 Model Creation

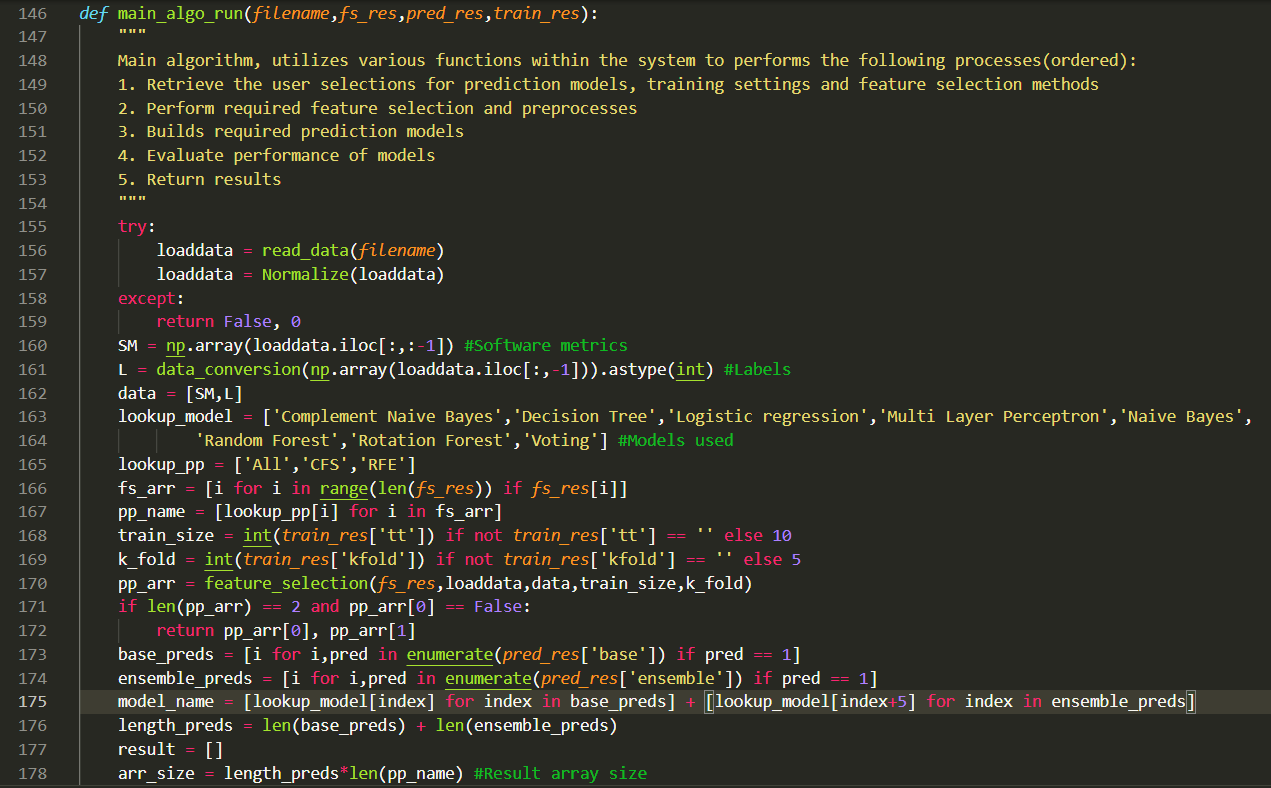


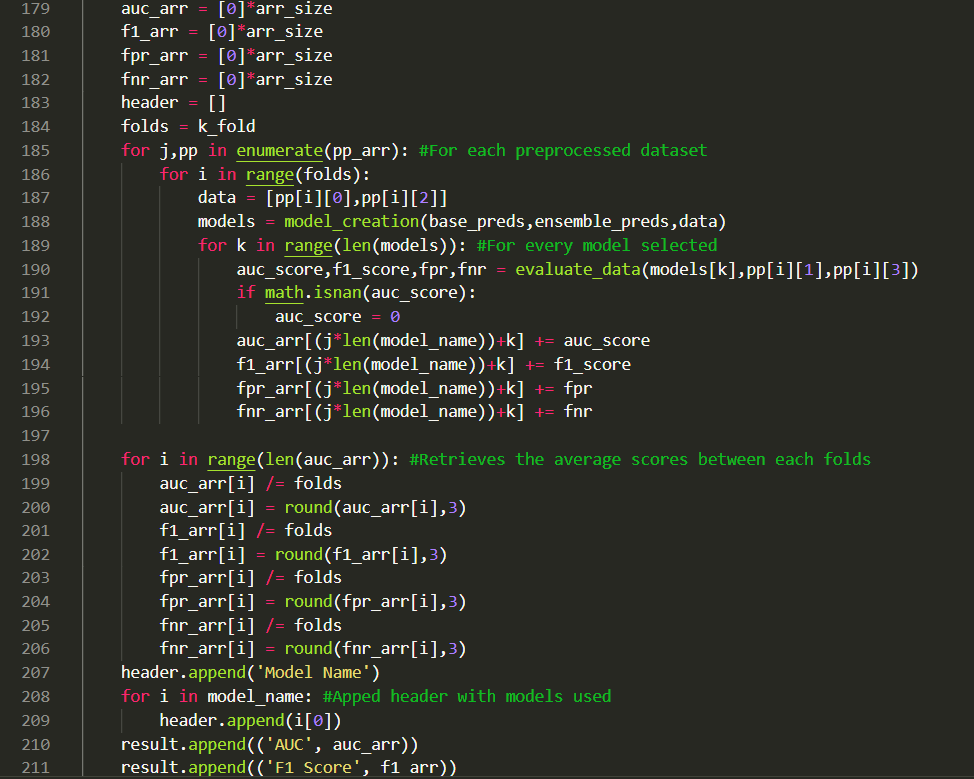
## 8.7 Writing to CSV

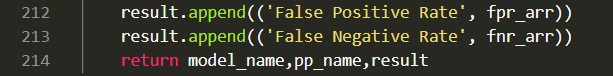


# 

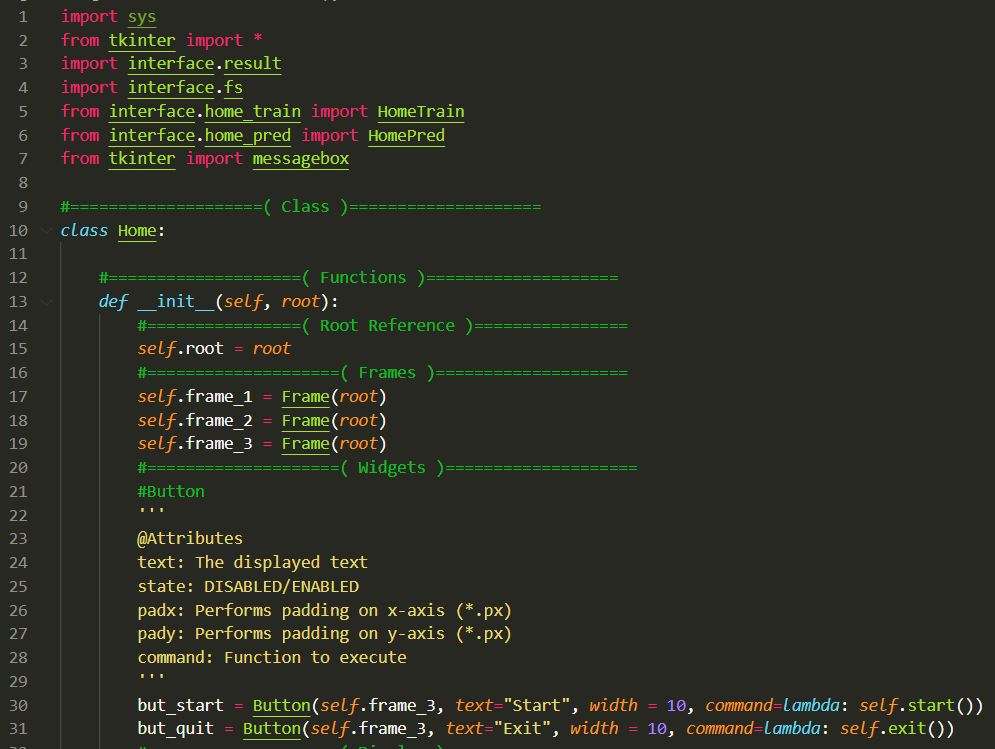
## 8.8 Main Program



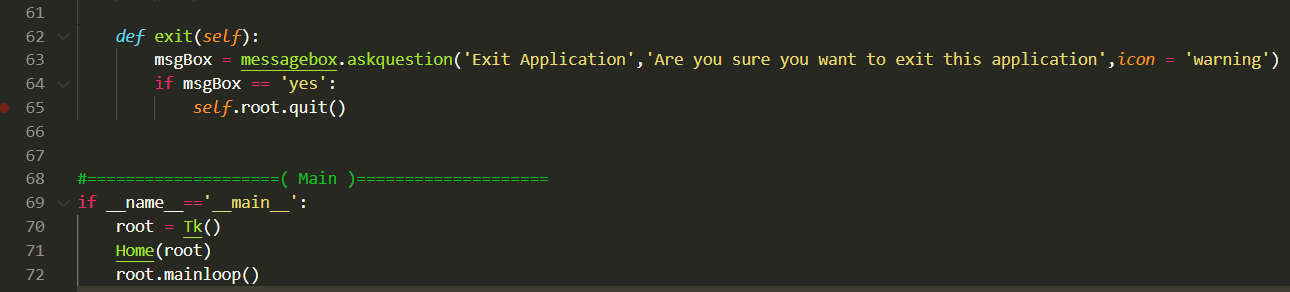




## 8.9 User Interface







# 9. References

Brownlee, J. (2016). Logistic Regression for Machine Learning.

Retrieved 20 October 2021, from

<https://machinelearningmastery.com/logistic-regression-for-machine-learning/>

Brownlee, J. (2018). A Gentle Introduction to k-fold Cross-Validation.

Retrieved 20 October 2021, from

<https://machinelearningmastery.com/k-fold-cross-validation/>

Brownlee, J. (2020). How to Develop Voting Ensembles With Python.

Retrieved 20 October 2021, from

<https://machinelearningmastery.com/voting-ensembles-with-python/>

Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data

using Matthews Correlation Coefficient metric. *PloS one*, *12*(6), e0177678.

GeeksforGeeks (2020). Complement Naive Bayes (CNB) Algorithm.

Retrieved 20 October 2021, from

<https://www.geeksforgeeks.org/complement-naive-bayes-cnb-algorithm/>

Haque, M. N., Noman, N., Beretta, R., & Moscato, P. (2016). Heterogeneous ensemble

combination search using genetic algorithms for class imbalanced data classification. *PloS one*, *11*(1), e0146116.

Le, T., Le Son, H., Vo, M. T., Lee, M. Y., & Baik, S. W. (2018). A cluster-based boosting

algorithm for bankruptcy prediction in a highly imbalanced dataset. *Symmetry*, *10*(7), 250.

Nicholson, C. (n.d.). A Beginner’s Guide to Multilayer Perceptrons (MLP).

Retrieved 20 October 2021, from

<https://wiki.pathmind.com/multilayer-perceptron>

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for

software fault prediction: A survey. *Expert Systems with Applications*, 114595.

Rana, Z. A., Mian, M. A., & Shamail, S. (2015). Improving Recall of software defect

prediction models using association mining. *Knowledge-Based Systems*, *90*, 1-13.

Ray, S. (2017). 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. Retrieved 20 October 2021, from <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>

Rtayli, N., & Enneya, N. (2020). Enhanced credit card fraud detection based on SVM-recursive feature elimination and hyper-parameters optimization. *Journal of Information Security and Applications*, *55*, 102596..

Section (2020). Introduction to Random Forest in Machine Learning.

Retrieved 20 October 2021, from

<https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/>

Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising

autoencoders and two-stage ensemble learning. *Information and Software Technology*, *96*, 94-111.

Xoriant (2017). Decision Trees for Classification: A Machine Learning Algorithm.

Retrieved 20 October 2021, from <https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html>

Yap, B. W., Abd Rani, K., Abd Rahman, H. A., Fong, S., Khairudin, Z., & Abdullah, N. N.

(2014). An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)* (pp. 13-22). Springer, Singapore.

Yen, S. J., & Lee, Y. S. (2006). Under-sampling approaches for improving prediction of the

minority class in an imbalanced dataset. In *Intelligent Control and Automation* (pp. 731-740). Springer, Berlin, Heidelberg.

Yin, L., Ge, Y., Xiao, K., Wang, X., & Quan, X. (2013). Feature selection for high-dimensional imbalanced data. *Neurocomputing*, *105*, 3-11

Yucalar, F., Ozcift, A., Borandag, E., & Kilinc, D. (2020). Multiple-classifiers in software

quality engineering: Combining predictors to improve software fault prediction ability. *Engineering Science and Technology, an International Journal*, *23*(4), 938-950.

Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., & Li, S. (2015, December). Combining software

metrics and text features for vulnerable file prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)* (pp. 40-49). IEEE.