FIT3162

Final Project Report

Improving Software Testing Using Software Fault Prediction When Labelled Data Is Not Present

Team:

Charles Tan Wei Wen

Edmund Wu Xiao Cong

Shafkat Ibrahimy

Supervisor:

Dr. Golnoush Abaei

Word Count: 8616

1. Introduction

Due to the increasing size and complexity of modern software, it has become increasingly difficult to conduct software testing efficiently and effectively due to time and resource constraints (Abaei et al., 2013). Without testing, the software is much more prone to faults and errors which results in customer dissatisfaction at best and catastrophic failure at worst. Some examples are the NASA Mars Climate Orbiter in 1999 and the European Space Agency's Ariane 5 Rocket in 1996, all of which could have been avoided with sufficient testing. Both of which could have been avoided with sufficient software testing.

The software testing process contains many processes such as testing, verification and validation, fault tolerance, and fault prediction. The process is also very time-consuming and is a very important part of the software development phase. Good software testing guarantees software quality and saves money by identifying faults early and fixing them before they can cause harm to users. This project focuses on the software fault prediction of the software testing process, Fault prediction is used to identify modules that are most likely to contain errors and is used when time or budget restraints don't allow for full software testing (Abaei et al., 2013).

Our project aims to alleviate the process of software testing by offering up a system that can predict the fault proneness of each module in any given system to allow for better, more thorough software testing. We aim to combine multiple machine learning techniques via the use of an ensemble. The ensemble consists of Fuzzy-C-Means, Hierarchical Clustering, and Self-Organizing Maps and is used as no one method is the best in all cases, so we attempt to make up for the shortcomings of certain algorithms with other algorithms to improve the accuracy of the predictions.

Bringing to life a system that can accurately predict the fault proneness of each system module would be a boon to programmers all over the world as it helps them save time on software testing and would improve software quality across the board and if the system performs well enough it could help in every field. However, not everyone would be able to use the system as it takes some technical knowledge to run it. To address this problem we have encapsulated the ensemble in a simple, user-friendly GUI to allow people who are less tech-savvy to also benefit from the system.

2. Background

As our ensemble makes use of multiple machine learning algorithms, we have reviewed several research papers in the previous semester, the following section is borrowed from our initial project proposal.

2.1 Literature Review

(Li et al., 2020) reviews 49 studies and found that unsupervised methods for fault prediction are on the rise as they do not require labeled data and can produce results comparable to those of supervised methods. With clustering methods seeming to dominate the unsupervised field. These usually start with clustering the dataset into the most appropriate clusters then labeling each cluster as faulty or non-faulty. They also found that certain dataset characteristics have an impact on prediction accuracy but as they were not able to identify what these characteristics are we will not be able to make use of this knowledge. Lastly, they concluded that a single machine learning algorithm will likely be nonoptimal.

(Bai et al., 2020) Proposes multiple k-means clustering algorithms with the local credible constraint to produce multiple clusters with different local-credible labels to solve the non-linearly separable clustering problem. The non-linearly separable clustering problem is when objects in the same clusters are highly similar to each other but those in different clusters are highly distinct from each other. This is an issue as there isn’t a single clustering algorithm that can suitably deal with all clustering tasks. The proposed algorithm aims to tackle this problem by using a clustering ensemble, taking k-means as the base cluster to build a multiple k-means clustering ensemble to simulate a non-linear clustering. To test their algorithm the team on some given datasets to compare the clustering accuracies, the team also found that their method performs better than other clustering ensembles on synthetic datasets and made the conclusion that in the initial cluster by the base k-means clustering, there exist some very good labels which other existing clustering ensembles do not evaluate. Secondly, the proposed algorithm can effectively discover non-linearly separable clusters to improve the accuracy of the k-means algorithm. On real datasets, it also performs better than other clustering ensembles. Lastly, as the proposed algorithm has a certain level of randomness, the team executed it on each dataset 50 times and found that the standard deviation is less than 0.1 on each dataset, thus showing that randomness has a small effect on the algorithm performance. To conclude, the team came up with an ensemble algorithm utilizing the popular k-means clustering algorithm to overcome its performance drop when met with data distributions. The team’s algorithm consists of producing multiple k-means clusterings, evaluating the local credibility of each label, building the relationship between clusters, and generating the final clusters. This algorithm improves the robustness of and quality of k-means and recognizes non-linearly separable clusters. This is further supported by the results of their performance test as shown above.

(Zhang et al., 2018) proposed a low-rank representation-based semi-supervised software defect prediction approach with low-rank representation for constructing a weighted graph. LRR aims to find the lowest rankness representation among all candidates that can express data vectors as linear combinations of the basis atoms in a proper dictionary; it guarantees that the sample coefficients from the same class are highly correlated and suppress noise present in the dataset. The proposed approach aims to address the problem of previous defect labels of the module in datasets are limited, and attempts to address it by using the few labeled defect data together with unlabelled defect data to construct a graph-based semi-supervised learning defect prediction model, by applying LRR in its graph construction phase, it can remove the noise present in the dataset while constructing the data graph. The algorithm first extracts and label defective/defect-free modules from the dataset into instances, then using predefined defect prediction metrics are used as module features, by combining module features and instances into a training corpus, LRR and data vector's neighborhood’s distance are used to construct the relationship graph of training instances. Finally, the predictors are used to predict the label of unlabelled modules. In conclusion, LRRSSDP not only uses labelled modules available in datasets but also unlabelled data to improve its generalization capability to predict the defect proneness of unlabelled modules. This helps to solve the problem of supervised training where the lack of labeled modules hinders its ability to build effective prediction models. If the dataset contains noise, LRRSSDP's performance does not degrade as much as other defect prediction models.

(Boucher & Badri, 2018) compares three techniques, ROC Curves, VARL, and Alves using 6 models with 12 datasets. Threshold-based models with ROC Curves performed the best. These models label classes as fault-prone when their metric value is greater than some threshold values, one advantage of these models is that it is easily implemented by developers and shows why a class is fault-prone by identifying the metric which needs to be checked. Alves, previously untested in the context of fault proneness, was found to be a good option and has the advantage of not needing fault data history (unlike ROC, VARL). Chosen metrics for this study were SLOC, RFC, CBO, and WMC, determined using logistic regression. For each system, one threshold value per code metric was calculated.

(Ying & Ahmed, 2016) Proposes an application of a connectivity-based unsupervised classifier based on spectral clustering to mitigate the problem of data heterogeneity. Data heterogeneity is when data(from another project) is too different to use as training data. This is an issue in defect prediction as newer projects or those with limited history have little data to use for defect prediction, a common solution to this is to reuse classifiers from other projects which may result in data heterogeneity. The team’s suggested method makes use of unsupervised classifiers and is free from the problems caused by data heterogeneity. Their proposed algorithm is as follows. An input of a matrix with rows as software entities and columns as metrics is given to the algorithm, the software metrics are then normalized using their z-score. Next, a weighted adjacency matrix W is created and the Laplacian matrix is calculated using it. The second smallest eigenvector from the eigendecomposition of the laplacian matrix is selected and bipartition is performed on it using zero. Finally, each cluster is labeled as defective or clean. To test the practicality of their proposed method they conducted an experiment and found that general unsupervised classifiers underperform compared to supervised ones but connectivity-based unsupervised classifiers can compete with supervised ones. Their proposed method ranks as one of the top classifiers in the cross-project setting and it is also ranked as a tier 2 classifier in the within-project setting. Upon further investigation, they found their assumption that faulty entities have significantly stronger connections to other faulty entities than non-faulty entities is correct. In conclusion, (Ying and Ahemd, 2016) found that the use of unsupervised methods to avoid the issue of data heterogeneity is a good way to address the issue. Furthermore, their proposed method is capable of achieving good results for cross-project scenarios and is a decent performer in the within-project sphere. Overall they found that distance-based classifiers did not perform as well as connectivity-based ones and that supervised methods perform better in the within-project setting.

(Abaei et al., 2015) HySOM is a semi-supervised algorithm model that makes use of artificial neural networks and self-organizing maps for fault detection and classification of data. It is used to predict labels of the input modules using measurement threshold values which is useful when defect data hasn’t been identified or when the company does not have similar versions of the software they are testing. Combining neural networks and SOM in the model leads to the overall model performing better in improving software development and resource allocation than its individual components alone. The hybrid model uses SOM as the primary clustering algorithm to produce sample data sets which are then used by ANN (artificial neural network) to train the predictive model to classify new-set of unlabeled data as either faulty or non-faulty. In developing the model, input data is split 66-33 into two groups using stratified sampling, a training set for building the model, and a testing set to evaluate the model. Unlabeled training set goes into Phase 1: clustering using SOM, Phase 2: removing dead neurons, then Phase 3: computing neuron’s weights using threshold values (method-level metrics threshold values determined from predictive tools by ISM.). After that, the output weights are given to ANN for classification. Lastly, validation, the ANN model is accepted if the computed error value is reasonable. The test set is then labeled by the accepted model. For evaluation FNR, FPR, and overall error rate are calculated. The performance in terms of overall error rate is comparable with other unsupervised and supervised methods and HySOM can be used without the availability of experts as an automated tool.

(Laradji et al., 2015) proposed a new software defect classification using average probability ensemble learning module where the system incorporates seven classifiers (random forests (RF), gradient boosting (GB), stochastic gradient descent (SGD), weighted SVMs (W-SVMs), logistic regression (LR), multinomial naive Bayes (MNB) and Bernoulli naive Bayes (BNB)). This system is used to demonstrate the positive effect of feature selection on the performance of defect classification and enables the base classifiers to capture different statistical characteristics of given data. It is expected to exhibit stronger robustness to redundant and irrelevant features when combined with efficient feature selection yielding an enhanced ensemble classifier. The preprocessing of proposed models are to split the dataset into 90% training subset and 10% testing subset where for each best next feature, append selected features into a list, the ensemble classifier will be trained on those selected features and the trained classifier will be used to predict defective components of testing subset, its performance will be recorded and the next iteration will append best next feature into said list. For training the proposed model, the same splitting will be done on a dataset of ratio 9:1, the proposed model will be trained using training subset and evaluated with testing subset, store classification performance as p1, select best features and append into a list from training subset using greedy-forward selection method and retrain model, re-test model using testing set and re-iterate steps with next best features appended into the list, store classification performance as p2. For evaluation, greedy selection significantly outperforms Fisher's criterion and Pearson's correlation, obtaining the highest AUC values given the same features selected. For feature classification results, its AUC values also outperform the likes of W-SVMs and RF when given known imbalanced datasets with redundant features.

(Abaei et al., 2013) proposed two experiments, the first where they used Self-Organizing Maps(SOM) for both clustering and classification and the second where a threshold is simply applied to raw data. It should be noted that any labels that would have existed on the data were removed. In the two-stage SOM approach, data is categorized in a small representative map where threshold values were applied for labeling each trained neuron as faulty or non-faulty. The next step is to calculate the chance of "winning" for each neuron. This is done by counting the Number Of Hits(NOH). Neurons with a lower NOH than the threshold are considered to be dead to attempt to improve accuracy. The next step has classification done by SOM on the same training data set. Error rates and other evaluation metrics are calculated for each epoch. In conclusion, (Golnoush, Zahra & Ali, 2013) found that their proposed method of two-stage SOM clustering and classification with threshold outperformed earlier research papers and improved classification of unlabelled program modules in terms of FPR, FNR, and error rate in most cases. Furthermore, the proposed method makes use of thresholds instead of the traditional method of hiring experts with 15 years of experience. This is very impactful as such experts are very difficult to find.

(Bishnu & Bhattacherjee, 2012) proposed a method where using QuadTrees as the initiation of the k-means algorithm can help to improve overall error rates compared to other existing algorithms, assigns appropriate initial clusters by varying the value of threshold parameters, and help eliminate outliers. In the quad tree-based k-means clustering algorithm, the preprocessing is done by recursive decomposition of space using separators parallel to the coordinate axis. At each step, if the number of data points in the current square is less than the threshold, it is marked as white, if the number is higher than the threshold, it is marked as black, otherwise, it is marked as grey as it falls in between the lowest and highest boundary(threshold). Calculate center for each black leaf bucket, for each neighborhood set of the center of black leaf bucket, for each unmarked center, select a center and label it as marked, find the distance of nearest distance to an unmarked neighbor and include them in the neighborhood set of the center of black leaf bucket. For all neighborhood sets of the center of the black leaf bucket, calculate the mean and call it the cluster center. In conclusion, this preprocessing can help to identify suitable initial cluster centers and remove outliers for the k-means algorithm to perform better compared to other algorithms in terms of overall error rates and better FNR and FPR compared to the metrics-based approach. Furthermore, it removes the dependencies of needing to have experts to label clusters as faulty or not.

(Sammar et al, 2018) proposed an application of software fault prediction using ensemble classifiers and different sets of software metrics to determine the best model for fault prediction. Among the different software metrics, three main sets of metrics are used, namely static code metrics and class level change metrics and the combination of previously mentioned two sets of metrics. The proposed bug prediction models are separated into two stages where the first will be generating the software metrics from the dataset, and the second stage will be to create ensemble classifiers using 4 types of weighted majority voting techniques, using WM, RWM, CWM, and CRWM, and 5 types of single classifiers that consist of NB, decision trees, SVM, LR, and RF. Through running a combination of mentioned ensemble classifiers using a combination of single classifiers in 3 and 5 seats, the team has discovered that throughout the study, CRWM using an ensemble of 5 single classifiers overall has better FM, accuracy, and AUC values compared to single classifiers and common ensemble classifications such as WM and RWM. Overall, they have discovered that the usage of using change metrics resulted in better performance than static metrics while the use of ensemble classifiers has also solved the class-imbalanced datasets by improving their classification accuracy.

2.2 Key Literature (summary)

(Li et al., 2020) has conducted a study where they have reviewed 49 studies and discovered that the number of studies that are using unsupervised learning methods for fault prediction is increasing while acknowledging that a single machine learning algorithm will not be nonoptimal. (Bai et al., 2020) proposed multiple k-means clustering algorithms using a clustering ensemble to simulate a non-linear clustering algorithm. The resulting algorithm can efficiently discover non-linear separable clusters to improve accuracy while randomness does not affect its run to run variance in results. (Zhang et al., 2018) proposed a low rank representation-based semi-supervised software fault prediction algorithm, LRR ensures the correlation of sample coefficients from the same class and thus achieves noise suppression in the dataset. The results from using LRR show the algorithm uses both labeled data and unlabelled data to build its model, resulting in better noise suppression and its generalization capability. (Boucher & Badri, 2018) discovered that among ROC, VARL, and Alves, ROC values performed the best in terms of threshold-based models. (Ying & Ahmed, 2016) proposed a connectivity-based unsupervised classifier based on spectral clustering to resolve the problem of data heterogeneity, the proposed algorithm performs better than unsupervised learning while being able to compete against supervised learning in terms of its performance. (Abaei et al., 2015) proposed an algorithm that uses HySOM that uses ANN and SOM for fault prediction, SOM is first used as a clustering algorithm and ANN is used to train the predictive model, the result is comparable with unsupervised algorithms and HySOM can be used as an automated tool, negating the need of an expert. (Abaei et al., 2013) also proposed an algorithm where SOM is used for clustering and classification while a threshold is simply applied to the raw data for prediction. The result of which showed an improvement towards its FPR, FNR, and error rates in most cases. (Bishnu & Bhattacherjee, 2012) proposed an approach to improve k-means by using quadtrees as an initiation technique, this preprocessing technique resulted in better performance for k-means as suitable initial cluster centers are located and overall FNR and FPR were achieved compared to metrics-based approach.

3. Project Outcomes

3.1 What has been implemented

Throughout the semester the team has successfully implemented the Self-Organizing Maps, Hierarchical Clustering, and Fuzzy-C-Means clustering algorithms for software fault prediction. Each algorithm is capable of being run independently. These algorithms all take an input of a dataset containing no labels (should labels be present in any location other than the final column they will need to be removed beforehand) and produce a container containing the predicted labels and finalize the results in the form of a standardized confusion matrix showing the accuracy of the predictions. They are further integrated into a majority vote-based ensemble which takes their results and conducts a vote to determine the final predictions before outputting the final confusion matrix.

All of the above is neatly packed into a Graphical User Interface that simplifies the usage of the program. The GUI is simple and minimalistic with minimal clutter. The system does not need any user intervention apart from selecting the dataset and pressing run. For more details on implementation please refer to section 4.2

3.2 Results achieved/product delivered

The final product delivered is in the form of the GUI mentioned above, the users will not need to interact with the backend of the system unless they wish to make modifications to the code such as changing the parameters for specific algorithms. In all other cases, all the user has to do is select the input dataset, the algorithm to run, and press the run button to execute the program. Figure 1 below shows the blank state of the GUI.

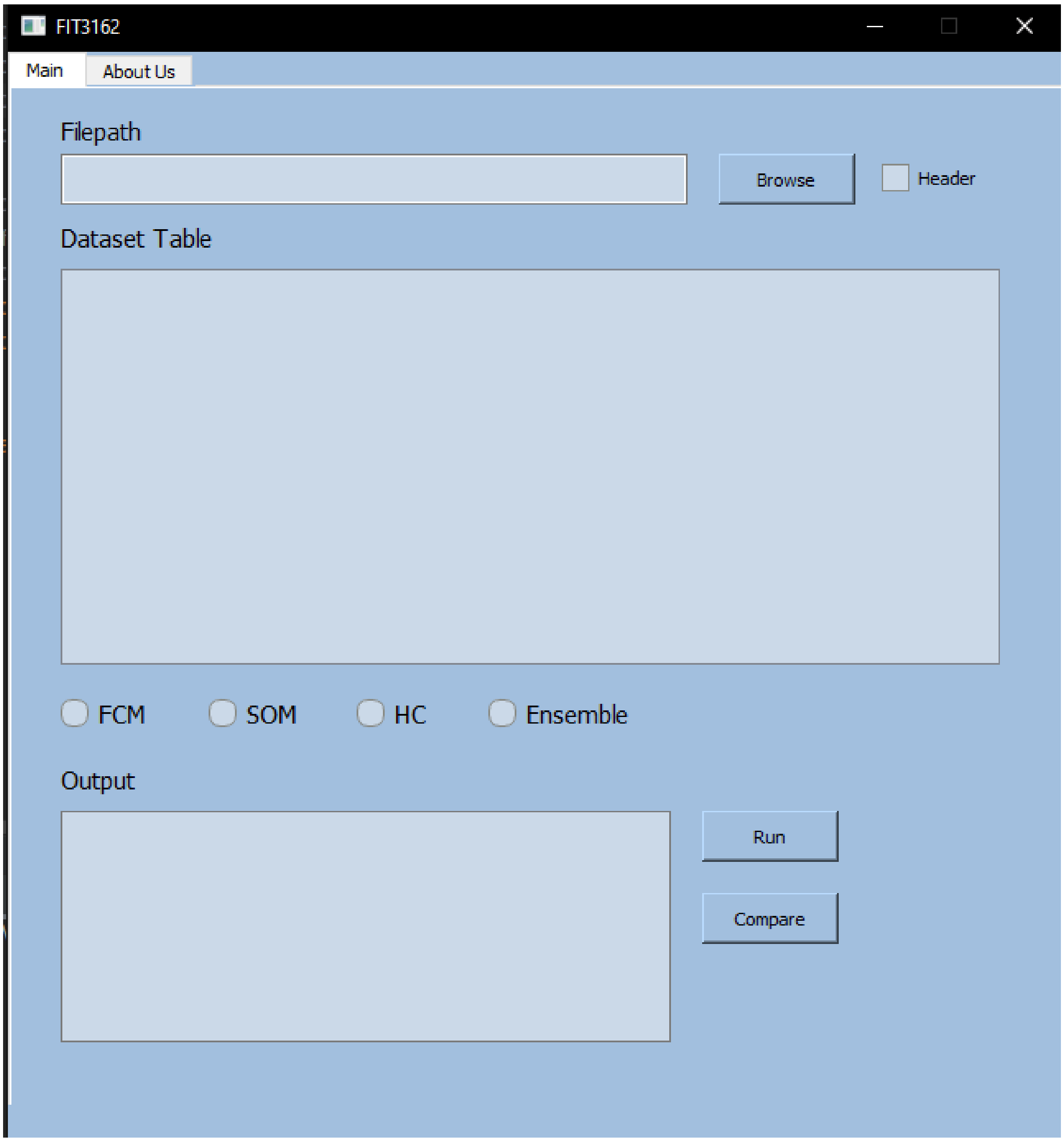


Figure 1. Blank state of the GUI

3.3. How are requirements met

The following table 1 of functional and non-functional requirements is extracted from our project proposal in FIT3161.

Table 1: Product Requirements

ID	Requirement
F1	The fault detection system can predict faulty modules with good accuracy.
F2	The system should complete execution within an acceptable time frame.
F3	The system should identify labels present in datasets and remove them prior to fault detection
F4	The system should be fully autonomous apart from getting input data from the user.
F5	The system should not need assistance from an expert.
NF1	The user interface should be simple and intuitive.
NF2	The system should return the performance of each machine learning algorithm in the ensemble.
NF3	The program should be easy for anyone to learn and use.

The following table 2 describes how each requirement has been met.

Table 2: How Requirements are met

ID	How it was satisfied
F1	The system was tested thoroughly and went through multiple iterations until the best results from our combination of unsupervised machine learning algorithms was achieved. For more information on how it was carried out please refer to section 3.5.
F2	We have removed the slower implementation of FCM we initially had and replaced it with a faster one to reduce processing time. Further description of this replacement is discussed in section 4.2.3.
F3	The system has been coded to remove the data labels present in the dataset by completely removing the last column of data (where the labels are present). This has also been manually checked to be functioning as intended as reported in our test report.
F4	The system has been coded to execute all relevant code as soon as the user selects the dataset through the browse button in the GUI and presses the run button. No further input is needed.
F5	The system has been coded to output 3 files that contain all the relevant data needed for someone to test the predicted faulty modules, namely in the predicted.csv file which contains the predicted labels, and as such, no professional help should be needed.
NF1	The user interface has been designed with minimal clutter and only contains necessary elements. An amateur should be able to use it at first glance without professional help.
NF2	The GUI contains a compare button which when pressed shows the user some graphics of how each algorithm performed. Furthermore, it also allows execution of the specific algorithms in the ensemble should the user wish to compare manually.
NF3	The provided user guide contains simple instructions on the usage of the program for anyone to learn. Furthermore as mentioned in NF1, there is minimal clutter so there should be minimal confusion in terms of usage.

3.4 Justification of decisions made

To allow for easier readability this section will be divided into 3 subsections, each of which will represent a phase in the development of the program.

3.4.1 Phase 1

The initial prototype ensemble consisted of K-Means, SOM, and FCM. However, due to implementation issues (further expanded in section 4.2.1) the team could not make a prototype as we only had 2 out of 3 needed algorithms for the ensemble. Here the team fell victim to the sunk cost fallacy, where we continued to put more time into trying to fix SOM instead of swapping the algorithm. The suggested solution from our project supervisor was to replace SOM with another algorithm. As the team could not properly interpret the output of the initial SOM which ended up taking a lot of development time which produced no results.

3.4.2 Phase 2

To start, SOM was replaced with HC. The swap allowed the team to have a prototype ensemble with all the core functionality implemented. Some of the unaddressed issues at this stage were the slow running time of FCM and the lack of a benchmark for comparison. At this stage, the team decided on first addressing the slow run time of FCM. This was done by replacing the locally hosted FCM with a library as described in section 4.2.3. This was the first iteration of the ensemble, which showed an average accuracy below those of the individual parts. As this did not meet the requirements the team went on to optimize the ensemble.

3.4.3 Phase 3

Phase 3 consisted of mostly optimizations to the results produced by the ensemble. To that end, the team revisited SOM by looking for online libraries that contained it. The reason for abandoning the initial implementation was because it faced the same issues as FCM in terms of runtime and would take more time than is worth it to fix. With SOM available the team could experiment with multiple ensemble types. Namely, the initial ensemble KHF (K-means, Hierarchical Clustering, and Fuzzy-C-Means) and the new ensemble SHF(Self-Organizing Maps, Hierarchical Clustering, and Fuzzy-C-Means). The results from the two were compared and the new ensemble proved to show better performance and was picked as the final ensemble. The comparison of results will be further explored in the next section.

3.5 Discussion of all results

In reaching the final iteration of our ensemble we went over multiple combinations of algorithms, this section will explore the thought process behind the final chosen combination. The initial ensemble consisted of the KHF model(K-means, Hierarchical Clustering, and Fuzzy-C-Means) where the FCM being used is the initial FCM with long runtimes. The confusion matrix values of the initial Ensemble are as shown in table 3 below (rounded to 2 decimal places for easier viewing).

Table 3: Results of initial Ensemble

Ensemble	Accuracy	Precision	Recall	TPR	TNR	FPR	FNR	F1 Score
CM1	90.08	0.00	0.00	0.00	99.56	0.44	100.00	N/A
JM1	80.74	87.50	0.33	0.33	99.99	0.01	99.67	0.66
KC1	85.19	58.67	13.54	13.54	98.26	1.74	86.46	22.00
KC3	89.93	41.18	16.28	16.28	97.58	2.42	83.72	23.33
KC4	52.42	75.00	4.92	4.92	98.41	1.59	95.08	9.23
MC1	99.26	0.00	0.00	0.00	99.98	0.02	100.00	N/A
MC2	70.00	75.00	11.54	11.54	98.15	1.85	88.46	20.00
MW1	86.57	17.14	19.35	19.35	92.18	7.82	80.65	18.18
PC1	93.22	66.67	2.63	2.63	99.90	0.10	97.37	5.06
PC2	99.55	33.33	8.70	8.70	99.93	0.07	91.30	13.79
PC3	89.69	0.00	0.00	0.00	99.93	0.07	100.00	N/A
PC4	86.34	19.44	3.95	3.95	97.73	2.27	96.05	6.57
PC5	97.05	66.67	3.49	3.49	99.95	0.05	96.51	6.63

At a glance, looking at only the accuracy value shows that it achieves good results in most cases with exceptions for KC4 and MC2 which both produce unfavorable results. However when compared to the accuracies achieved by the individual algorithms a major issue is made apparent. The following table 4 illustrates this issue.

Table 4: Results of Ensemble against its parts

	Accuracy (%)			
	Kmeans	Hierarchical	FCM	Ensemble
CM1	90.07936508	90.07936508	90.09900990	90.07936508
JM1	80.73917440	80.73917440	80.74094503	80.73917440
KC1	85.13770180	85.23266857	85.19221642	85.18518519
KC3	89.93435449	90.15317287	89.95633188	89.93435449
KC4	52.41935484	52.41935484	52.80000000	52.41935484
MC1	99.26043317	99.26043317	99.26051130	99.26043317
MC2	70.00000000	70.00000000	70.18633540	70.00000000
MW1	86.56716418	91.04477612	86.60049628	86.56716418
PC1	93.21880651	93.21880651	93.13459801	93.21880651
PC2	99.55261274	99.55261274	99.58847737	99.55261274
PC3	89.62868118	89.69270166	89.69929623	89.69270166
PC4	86.89087165	84.14550446	86.28257888	86.34179822
PC5	97.05557172	97.05557172	97.04992436	97.04975269
Average	86.19108398	86.35339555	86.19928624	86.15697717

As the table shows, the average accuracy when using the ensemble is actually lower than all of the individual parts in the first iteration. This was likely due to the similarities in the results achieved by the algorithms in certain datasets, an example of which is CM1 where K-means and Hierarchical Clustering achieved the same results while FCM achieved a slightly higher accuracy rating, however, due to the nature of the majority voting ensemble, the overall results were brought down as two algorithms voted for the wrong prediction. This led to the team optimizing the results from the individual algorithms and replacing the older FCM with the new FCM based on a library. The results of the second iteration of the KHF model are shown in table 5 below.

Table 5: Results of the second Ensemble

ACCURACY					
Dataset	**K-Means**	**Hierarchical**	**FCM**	**KHF**	From DIFFERENCE table
CM1	90.0990099	90.0990099	90.0990099	90.0990099	0
JM1	80.74094503	80.74094503	80.74094503	80.74094503	0
KC1	85.09729473	85.23967727	85.23967727	85.23967727	0.1423825344
KC3	89.95633188	90.17467249	90.17467249	90.17467249	0.2183406114
KC4	52.8	52.8	52.8	52.8	0
MC1	99.2605113	99.2605113	99.2605113	99.2605113	0
MC2	70.1863354	70.1863354	70.1863354	70.1863354	0
MW1	91.06699752	91.06699752	86.60049628	91.06699752	4.466501241
PC1	93.13459801	93.22493225	93.13459801	93.13459801	-0.09033423668
PC2	99.58847737	99.55269279	99.58847737	99.58847737	0.03578457685
PC3	89.69929623	89.69929623	89.69929623	89.69929623	0
PC4	87.31138546	84.0877915	85.73388203	85.73388203	0.06858710562
PC5	97.05574305	97.05574305	97.05574305	97.05574305	0
Average	86.61514814	86.39912344	86.17797264	86.52154966	
				Average:	0.3724047563
				Sum:	4.841261832

From the table, it can be seen that while the Ensemble performs slightly below K-means on average it improves the accuracy of the predictions in 5 cases and only reduces it slightly in 1 case, on average it has improved accuracy by 0.37% and overall has improved it by 4.84%. To further improve results the team has tested the SHF model(Self-Organizing Maps, Hierarchical Clustering, and Fuzzy-C-Means). The results of the SHF model are shown in table 6 below.

Table 6: Results of SHF ensemble against its parts

ACCURACY					
Dataset	**SOM**	**HC**	**FCM**	**SHF**	From DIFFERENCE table
CM1	90.0990099	90.0990099	90.0990099	90.0990099	0
JM1	80.87883802	80.74094503	80.74094503	80.74094503	-0.137892995
KC1	85.04983389	85.23967727	85.23967727	85.23967727	0.1898433792
KC3	89.95633188	90.17467249	90.17467249	90.17467249	0.2183406114
KC4	53.6	52.8	52.8	52.8	-0.8
MC1	99.27107543	99.2605113	99.2605113	99.2605113	-0.01056412423
MC2	70.1863354	70.1863354	70.1863354	70.1863354	0
MW1	90.07444169	91.06699752	86.60049628	90.07444169	2.481389578
PC1	93.22493225	93.22493225	93.13459801	93.22493225	0.09033423668
PC2	99.28430846	99.55269279	99.58847737	99.55269279	0.2325997495
PC3	89.50735765	89.69929623	89.69929623	89.69929623	0.1919385797
PC4	86.83127572	84.0877915	85.73388203	85.73388203	0.548696845
PC5	97.07319912	97.05574305	97.05574305	97.07319912	0.03491213779
Average	86.54130303	86.39912344	86.17797264	86.45073811	
				Average:	0.2338152306
				Sum:	3.039597998

The following table 7 is used to compare the performances of the two models.

Table 7: Comparison of KHF and SHF

ENSEMBLE MODELS ACCURACY						
				Comparison		
Dataset	KHF	SHF	MAX	SHF comp	KHF comp	SHF-KHF
CM1	90.10	90.10	90.10	0.00	0.00	0.00
JM1	80.74	80.74	80.74	0.00	0.00	0.00
KC1	85.24	85.24	85.24	0.00	0.00	0.00
KC3	90.17	90.17	90.17	0.00	0.00	0.00
KC4	52.80	52.80	52.80	0.00	0.00	0.00
MC1	99.26	99.26	99.26	0.00	0.00	0.00
MC2	70.19	70.19	70.19	0.00	0.00	0.00
MW1	91.07	91.07	91.07	0.00	0.00	0.00
PC1	93.13	93.22	93.22	0.00	-0.09	0.09
PC2	99.59	99.55	99.59	-0.04	0.00	-0.04
PC3	89.70	89.70	89.70	0.00	0.00	0.00
PC4	85.73	85.73	85.73	0.00	0.00	0.00
PC5	97.06	97.07	97.07	0.00	-0.02	0.02

From the comparisons, it can be observed that overall SHF outperforms KHF. Resulting in the SHF model being chosen as the final model.

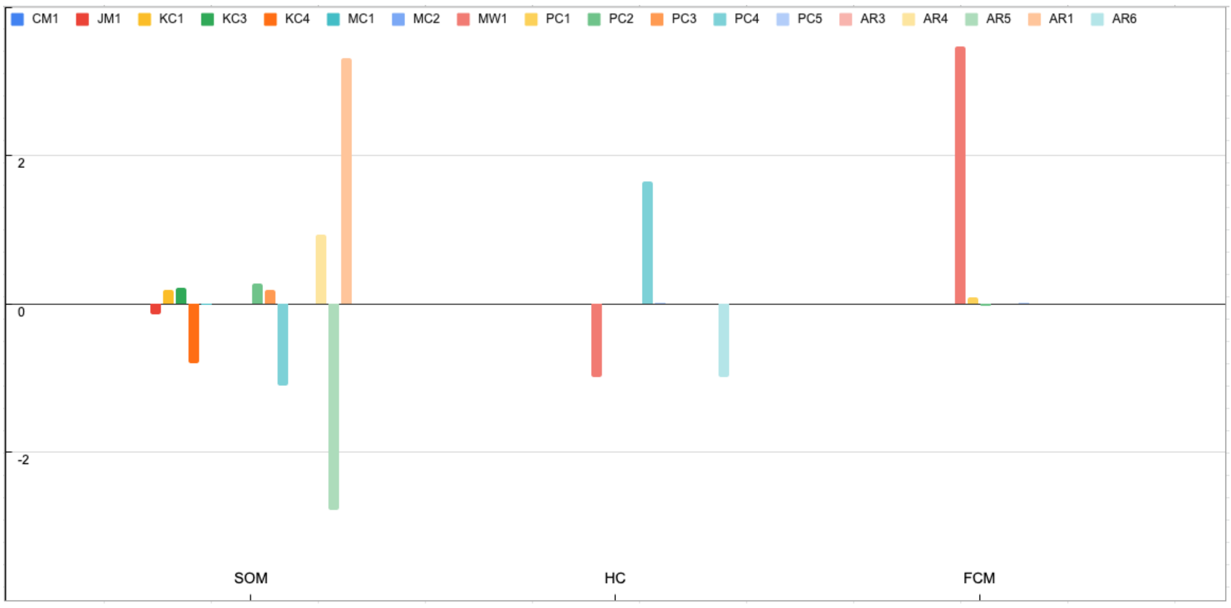


Figure 2: Visualization of SHF against its parts

Figure 2 above is a visualization of our final SHF model compared against SOM, FCM, and Hierarchical Clustering algorithms. This figure's y-axis represents the accuracy difference (where the scale is) between SHF and the other algorithms. The original 13 NASA datasets were used, as well as additional datasets from a Turkish manufacturing company, known as the "Turkish dataset" (AR3, AR4, AR5, AR1, AR6) to show the overall performance. From the figure above, it can be seen that SHF has a net positive increase in performance when considering a large number of datasets, even if it underperforms on certain datasets against some of the algorithms, for example, SHF does worse on KC4 when compared to SOM (and 3 other datasets). However, it outperforms SOM on 6 other datasets, which indicates a better overall performance.

3.6 Limitations of project outcomes

Some limitations of the project include the GUI's lack of batch processing capabilities and csv support which were omitted due to lack of developmental time. Another limitation is the variance of the accuracy of the predicted labels. While the system can reach up to 99% accuracy it can also fall as low as 52%, this is an issue as it is uncertain when a user will get the 99% accurate prediction or the 52% accurate prediction. Which limits the real-world applications of the program. Lastly, the team was limited in the amounts of unsupervised machine learning algorithms as we could only test the few algorithms that the team had implemented. More algorithms could open the door to consistent accuracy predictions which would greatly improve its real-world applications.

3.7 Discussion of possible improvements and future works

As shown above, the performance of our ensemble is not satisfactory as most of the accuracy results are below 95%. One of the possible improvements will be to revise our ensemble algorithm and implement weighted majority voting techniques such as the literature review we have mentioned. Other improvements can be to implement the program into a website where the users will be able to use the program without having to install libraries or frameworks, further improving the usability and allowing it to be far more accessible. Another improvement would be to implement a parser or interpreter to convert code files into a dataset where its input format will be understood by the program to work with, making the program more accessible to users as users do not have to research the type of datasets the program accepts and convert into a format that is readable for the program.

4. Methodology

4.1 Design

The initial design submitted for the ensemble in FIT3161 is as seen in figure 3 below.

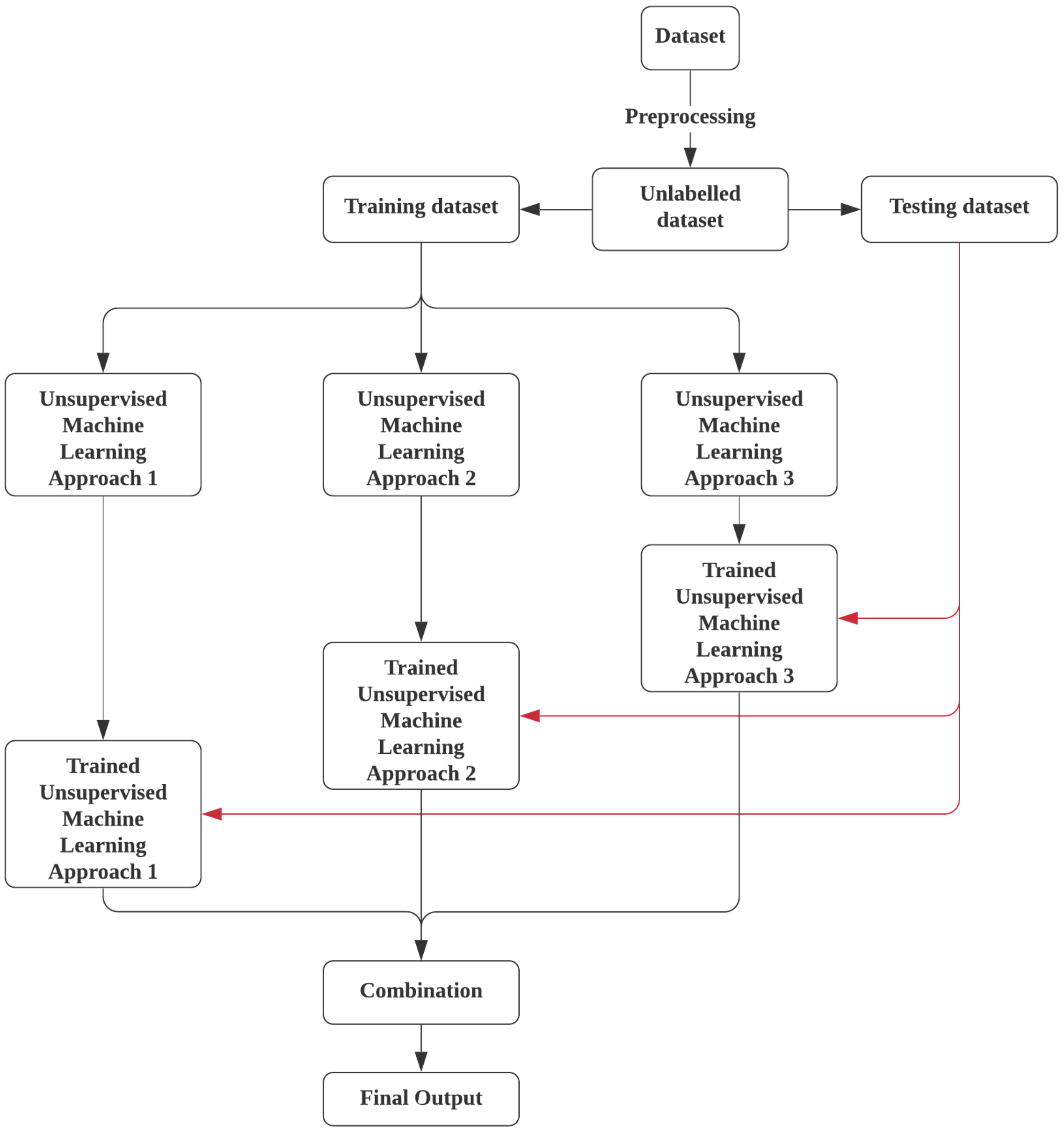

Dataset
Preprocessing
Training dataset
Unlabelled dataset
Testing dataset
Unsupervised Machine Learning Approach 1
Unsupervised Machine Learning Approach 2
Unsupervised Machine Learning Approach 3
Trained Unsupervised Machine Learning Approach 3
Trained Unsupervised Machine Learning Approach 2
Trained Unsupervised Machine Learning Approach 1
Combination
Final Output

Figure 3: Initial Design of the Ensemble

Initially, our design consisted of a bottom-up dataflow with a poorly defined GUI to backend linkage, where the data is given to each module first before they each return a list of predictions to the ensemble. The ensemble would then perform a majority vote to determine the final prediction and output it accordingly. However, the initial design has been updated as seen in figure 4 below

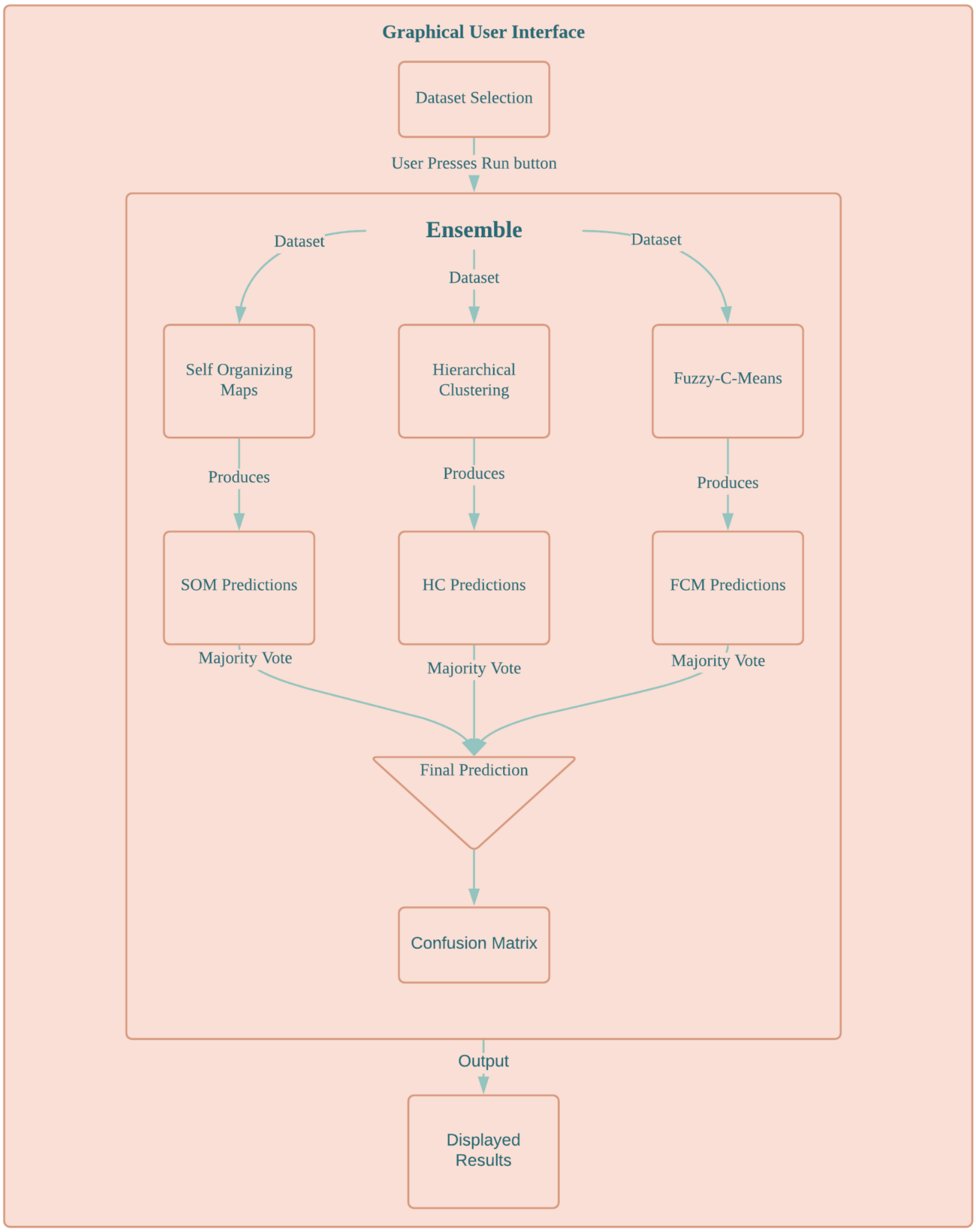


Figure 4: New design of Ensemble

Our new design builds on top of the previous design but changes the flow of data to be from the GUI to the ensemble. In our new design, the user only interacts with the GUI, the user would simply select the input dataset and press run. The ensemble will then receive this data and pass it to each module in the system. In our case, it would pass the dataset to FCM, HC, and SOM. Each algorithm would then complete execution and return a container that contains the predicted labels of each module in the dataset. The ensemble then takes the 3 returned containers and carries out a majority voting to determine the final label before it is returned to the GUI where its Accuracy, Precision, and Recall are outputted for the user to inspect. Three output files are also created, predicted which contains the dataset with the predicted labels, compare which contains the real label against the predicted label, and confusion matrix which contains the confusion matrix values.

Overall, while the design has changed slightly, most of the modules remain the same. The 3 unsupervised techniques remain in the system albeit in different locations. The Ensemble is better defined in the new design and the way the GUI interacts with the backend is also better defined.

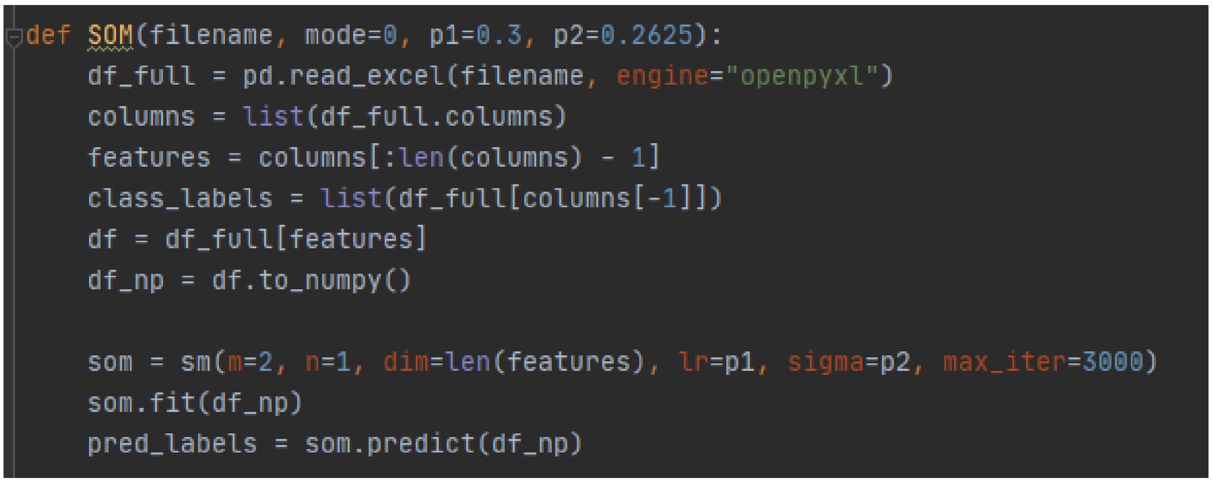
4.2 Implementation and Justification

The project consists of 5 major components, Self-Organizing Maps, Hierarchical Clustering, Fuzzy-C-Means, Ensemble, and the Graphical User Interface

4.2.1 Self-Organizing Maps

SOM was chosen as it was a different type of Unsupervised learning method that uses the artificial neural networks as opposed to clustering; it gives more variation and may produce better results if clustering proves to be ineffective in predicting faulty modules.

To implement SOM we made use of the sklearn_som library which houses the library for SOM. It contains a minimalistic implementation of the Kohonen Self Organizing Map with a planar topology. The parameters used in the creation of grid are defaulted to *(m = 3, n = , dim = 3, lr = 1, sigma = 1, max_iter = 3000, **kwargs)*. This is unsuitable for our project in several ways, the first is the m and n which determines the dimension of the SOM, as we need a 2-dimensional SOM we set m = 2 and n = 1. The next issue was the dim parameter, which is the number of features in the input space, as some datasets may have different amounts of features we set it to be dynamic as dim = len(features). The next parameters of lr which stands for learning rate and sigma which determines the magnitude of change of each weight had to be selected through brute force via the use of a loop. From the brute force testing, we reached the combination of lr = 0.3 and sigma = 0.2625 which was the best combination the team could find. The only dependency this library contains is the NumPy library and it is by far the easiest to use. This simplicity of the package is the main reason for its choice. The other library used is the Pandas library which was used to read the input files and remove the last column of data which contains the labels. The code sample for our implementation of SOM is shown in figure 5 below.



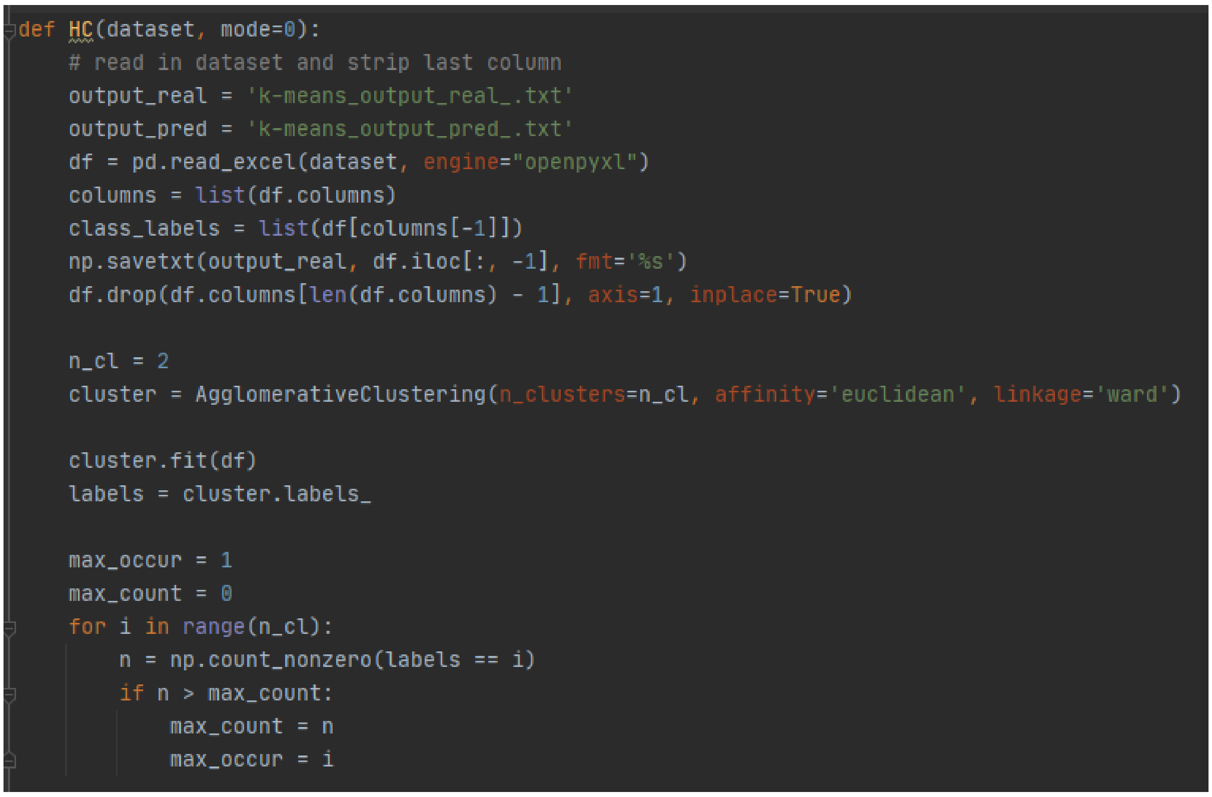
```
def SOM(filename, mode=0, p1=0.3, p2=0.2625):
    df_full = pd.read_excel(filename, engine="openpyxl")
    columns = list(df_full.columns)
    features = columns[:len(columns) - 1]
    class_labels = list(df_full[columns[-1]])
    df = df_full[features]
    df_np = df.to_numpy()

    som = sm(m=2, n=1, dim=len(features), lr=p1, sigma=p2, max_iter=3000)
    som.fit(df_np)
    pred_labels = som.predict(df_np)
```

Figure 5. SOM function in SOM.py

4.2.2 Hierarchical Clustering

Agglomerative Clustering, a variant of HC was chosen as one of the ensembles algorithms as it groups objects into clusters based on their similarities. It begins by treating each object as a single cluster and slowly merges clusters until only one cluster remains, this results in a dendrogram which is a tree-based representation of the objects. This clustering method lets us see if the similarities in data in the dataset imply the same label as some code features such as lines of code could mean nothing even in the event if it's the exact same number.

Much like SOM, Numpy and Pandas are used in the implementation of HC and are used for dataset manipulation. The library used for HC is the sklearn.cluster which contains the functions of the Agglomerative Clustering. The default parameters of which are (*n_clusters=2*, *, *affinity='euclidean'*, *memory=None*, *connectivity=None*, *compute_full_tree='auto'*, *linkage='ward'*, *distance_threshold=None*, *compute_distances=False*). These settings are a perfect fit for our project as we need to end up with 2 clusters in the end as indicated by n_clusters = 2. The other relevant parameters are affinity which is the metric used to compute the linkage and linkage which determine the distance to use between sets of observations which is used to merge pairs of clusters that minimize this criterion. The best combination of affinity and linkage the team has found is the default as it has an affinity of euclidean which is the team's preferred method and a linkage of the ward which minimizes the variance of the clusters being merged. The code sample for our implementation of HC is shown in figure 6 below.



```python
def HC(dataset, mode=0):
    # read in dataset and strip last column
    output_real = 'k-means_output_real_.txt'
    output_pred = 'k-means_output_pred_.txt'
    df = pd.read_excel(dataset, engine="openpyxl")
    columns = list(df.columns)
    class_labels = list(df[columns[-1]])
    np.savetxt(output_real, df.iloc[:, -1], fmt='%s')
    df.drop(df.columns[len(df.columns) - 1], axis=1, inplace=True)

    n_cl = 2
    cluster = AgglomerativeClustering(n_clusters=n_cl, affinity='euclidean', linkage='ward')

    cluster.fit(df)
    labels = cluster.labels_

    max_occur = 1
    max_count = 0
    for i in range(n_cl):
        n = np.count_nonzero(labels == i)
        if n > max_count:
            max_count = n
            max_occur = i
```

Figure 6. HC function in HC.py

4.2.3 Fuzzy-C-Means

FCM was used as one of the algorithms in the ensemble. The reason why FCM was used was because of its soft-clustering that yields better results compared to hard-clustering. The non-binary cluster membership for each data point can yield better prediction results as noise and outliers in datasets are handled more efficiently as each iteration refines the membership values of the data points to cluster centers.

The fcmeans library is used for FCM implementation, initially, we made use of a locally hosted FCM implementation but it was found to be too slow in terms of runtime and caused our software to take too long to execute. The fcmeans library was used as a replacement due to the runtime issues, in terms of performance it gives the same predictions but is much faster in execution. The main function for the fcmeans library is the FCM() function which is described to only take the number of clusters as input, as such the only thing the team modified was the n_clusters value which we set as 2 as a module is either faulty or non-faulty. As always, the NumPy and Pandas libraries are used to manipulate the input dataset. The code sample for our implementation of FCM is shown in figure 7 below.

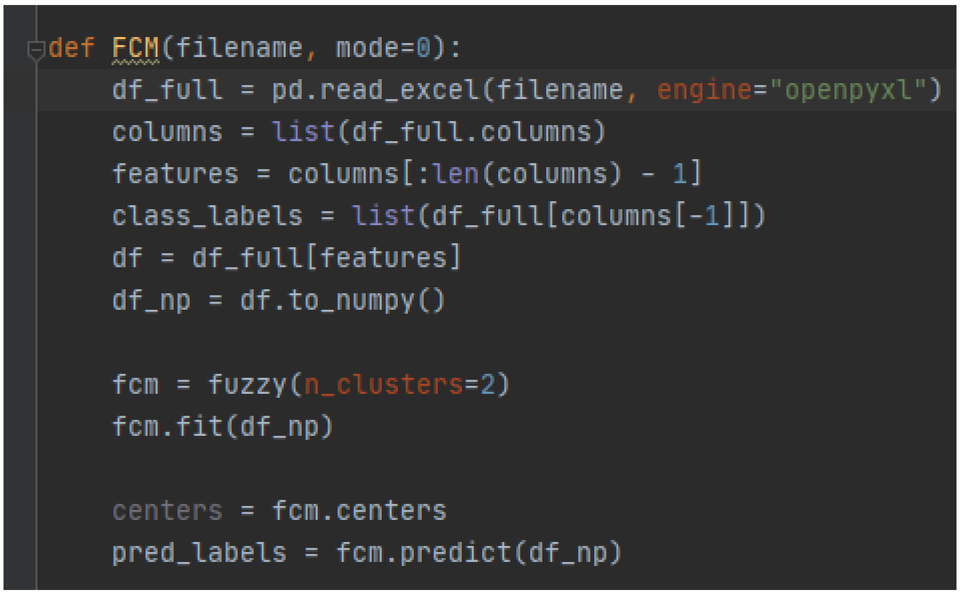
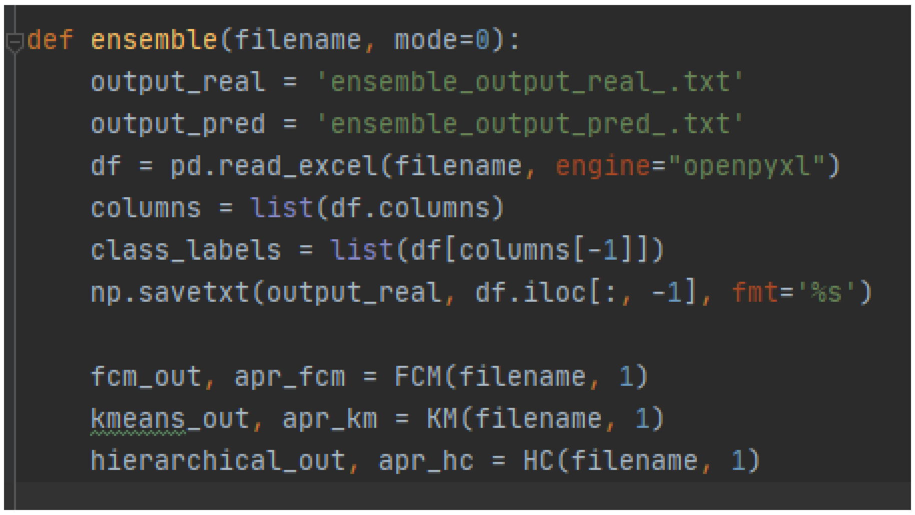

```python
def FCM(filename, mode=0):
    df_full = pd.read_excel(filename, engine="openpyxl")
    columns = list(df_full.columns)
    features = columns[:len(columns) - 1]
    class_labels = list(df_full[columns[-1]])
    df = df_full[features]
    df_np = df.to_numpy()

    fcm = fuzzy(n_clusters=2)
    fcm.fit(df_np)

    centers = fcm.centers
    pred_labels = fcm.predict(df_np)
```

Figure 7. FCM function in FCM.py

4.2.4 Ensemble

The ensemble is our main focus and while there are many forms of an ensemble, we chose the simple method of majority voting where the ensemble aggregates vote from the previously mentioned prediction models of SOM, HC, and FCM to produce potentially better results. The reason why ensemble was used was that the ensemble can combine the advantages of different algorithms and produce a prediction model that has better accuracy and precision, the overall runtime will be increased as the number of algorithms in the ensemble increases. Furthermore, the ensemble can sometimes produce worse results due to the nature of the majority vote where sometimes two weaker algorithms can bring down the performance of a stronger algorithm. This means that the ensemble the team implemented is by no means the best and future interactions will most likely improve the performance and accuracy even further.

As mentioned before, the ensemble makes use of SOM, HC, and FCM in the ensemble. So the previous modules are all implemented as separate files for easy importing with the ensemble, the ensemble imports all 3 modules and calls them all with the imputed dataset from the user, which then results in the algorithms returning their individual predictions. These predictions are then counted through a majority vote where the final prediction is decided. Lastly, we also included code that calculates the confusion matrix of the ensemble, and it is coded to work with all the three modules above so individual algorithms can be monitored to see which is the worst performer. The code sample for our implementation of the ensemble is shown in figure 8 below.



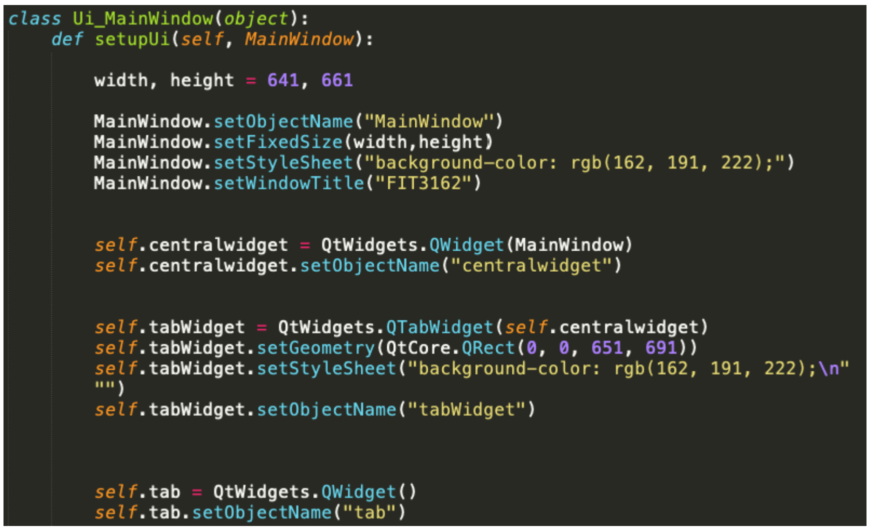
```python
def ensemble(filename, mode=0):
    output_real = 'ensemble_output_real_.txt'
    output_pred = 'ensemble_output_pred_.txt'
    df = pd.read_excel(filename, engine="openpyxl")
    columns = list(df.columns)
    class_labels = list(df[columns[-1]])
    np.savetxt(output_real, df.iloc[:, -1], fmt='%s')

    fcm_out, apr_fcm = FCM(filename, 1)
    kmeans_out, apr_km = KM(filename, 1)
    hierarchical_out, apr_hc = HC(filename, 1)
```

Figure 8. Ensemble function in ENS.py

4.2.5 Graphical User Interface

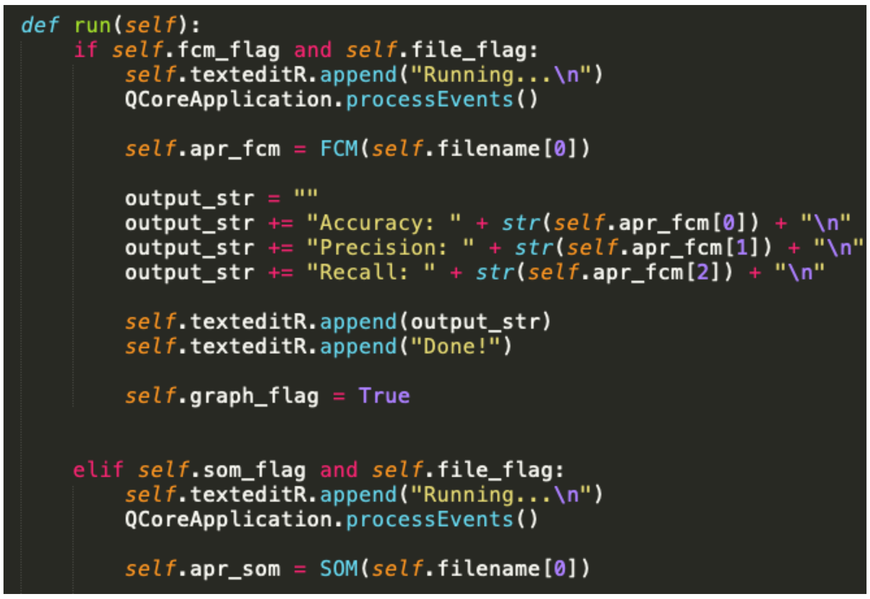
The inclusion of a GUI improves the usability of the system greatly, but acting as an interface between the user and the backend, it helps to greatly simplify the process of selecting the input dataset and feeding it to the backend. Furthermore, it allows for graphical representations of the results which help to illustrate the performance of each algorithm and the ensemble.

The GUI interface was built primarily using PyQt5 libraries. It can be split into two sections, one is the setup of the UI Main Window which contains details about buttons, texts, labels, and their styling. The second section is to do with the functions bound to those buttons, namely the main functions that we have are: browsing for a file (restricted to .xlsx), toggle switches for the clustering algorithms (i.e radio buttons), a graph compare which uses the matplotlib library, and the run button. The GUI also has a section to view the loaded dataset table. The functions are implemented by using triggers, which is receiving a click, it then checks that certain conditions have been met (for example, before a run can be executed an algorithm and dataset needs to be selected). The gui.py imports the functions of the clustering algorithm from HC.py, FCM.py, SOM.py, and ENS.py and executes them depending on which triggers (radio buttons) are selected. The following figures 9 and 10 are snippets of the two main sections mentioned.



```python
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):

        width, height = 641, 661

        MainWindow.setObjectName("MainWindow")
        MainWindow.setFixedSize(width,height)
        MainWindow.setStyleSheet("background-color: rgb(162, 191, 222);")
        MainWindow.setWindowTitle("FIT3162")


        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")


        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setGeometry(QtCore.QRect(0, 0, 651, 691))
        self.tabWidget.setStyleSheet("background-color: rgb(162, 191, 222);\n"
"")
        self.tabWidget.setObjectName("tabWidget")



        self.tab = QtWidgets.QWidget()
        self.tab.setObjectName("tab")
```

Figure 9. Main Window UI snippet from gui.py



```python
def run(self):
    if self.fcm_flag and self.file_flag:
        self.texteditR.append("Running...\n")
        QCoreApplication.processEvents()

        self.apr_fcm = FCM(self.filename[0])

        output_str = ""
        output_str += "Accuracy: " + str(self.apr_fcm[0]) + "\n"
        output_str += "Precision: " + str(self.apr_fcm[1]) + "\n"
        output_str += "Recall: " + str(self.apr_fcm[2]) + "\n"

        self.texteditR.append(output_str)
        self.texteditR.append("Done!")

        self.graph_flag = True


    elif self.som_flag and self.file_flag:
        self.texteditR.append("Running...\n")
        QCoreApplication.processEvents()

        self.apr_som = SOM(self.filename[0])
```

Figure 10. Run function snippet from gui.py

5. Software Deliverables

5.1 Deliverables

5.1.1 What is delivered

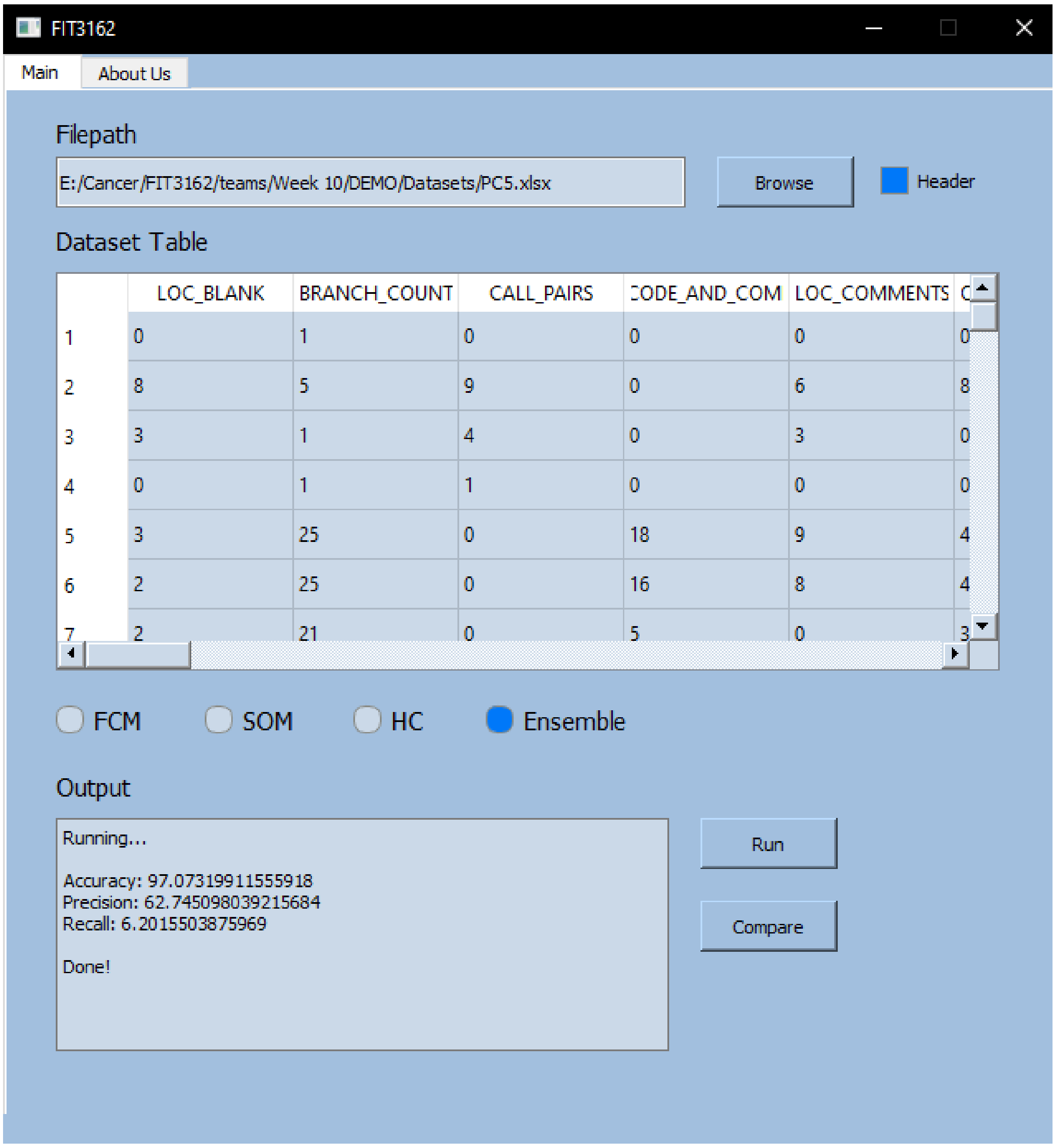
For our project, the deliverables are the source code that contains all of the algorithms used for our ensemble inside the folder we have uploaded.

Inside the folder, there are 5 files that contain the source code for each algorithm used in our project and the 13 cleaned datasets used for the testing. For each file, its usage is described in the table below.

Filename	Description
ENS.py	Contains functions for ensemble
FCM.py	Contains functions for Fuzzy-c means algorithm
gui.py	Python file for running the GUI for ensemble
HC.py	Contains functions for hierarchical clustering
SOM.py	Contains functions for Self-organizing maps
13 Datasets	The datasets used for testing and training

5.1.2 Sample screenshots and description of usage

Screenshots below are the sample usage of the program where the user will interact and obtain results from the ensemble. The following screenshots (figures 10 and 11) demonstrate the user running the ensemble with a dataset that the user inputs.



E:/Cancer/FIT3162/teams/Week 10/DEMO/Datasets/PC5.xlsx

Header [✓]

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	CODE_AND_COM	LOC_COMMENTS	C
1	0	1	0	0	0	0
2	8	5	9	0	6	8
3	3	1	4	0	3	0
4	0	1	1	0	0	0
5	3	25	0	18	9	4
6	2	25	0	16	8	4
7	2	21	0	5	0	3

FCM SOM HC [✓ Ensemble]

```
Running...

Accuracy: 97.07319911555918
Precision: 62.745098039215684
Recall: 6.2015503875969

Done!
```

Figure 10. Screenshot of the program running the ensemble on dataset PC5

After the ensemble has finished executing, the user can compare the results of each algorithm run in the ensemble about their Accuracy, Precision, and Recall. The screenshot below shows the output for the compare function.

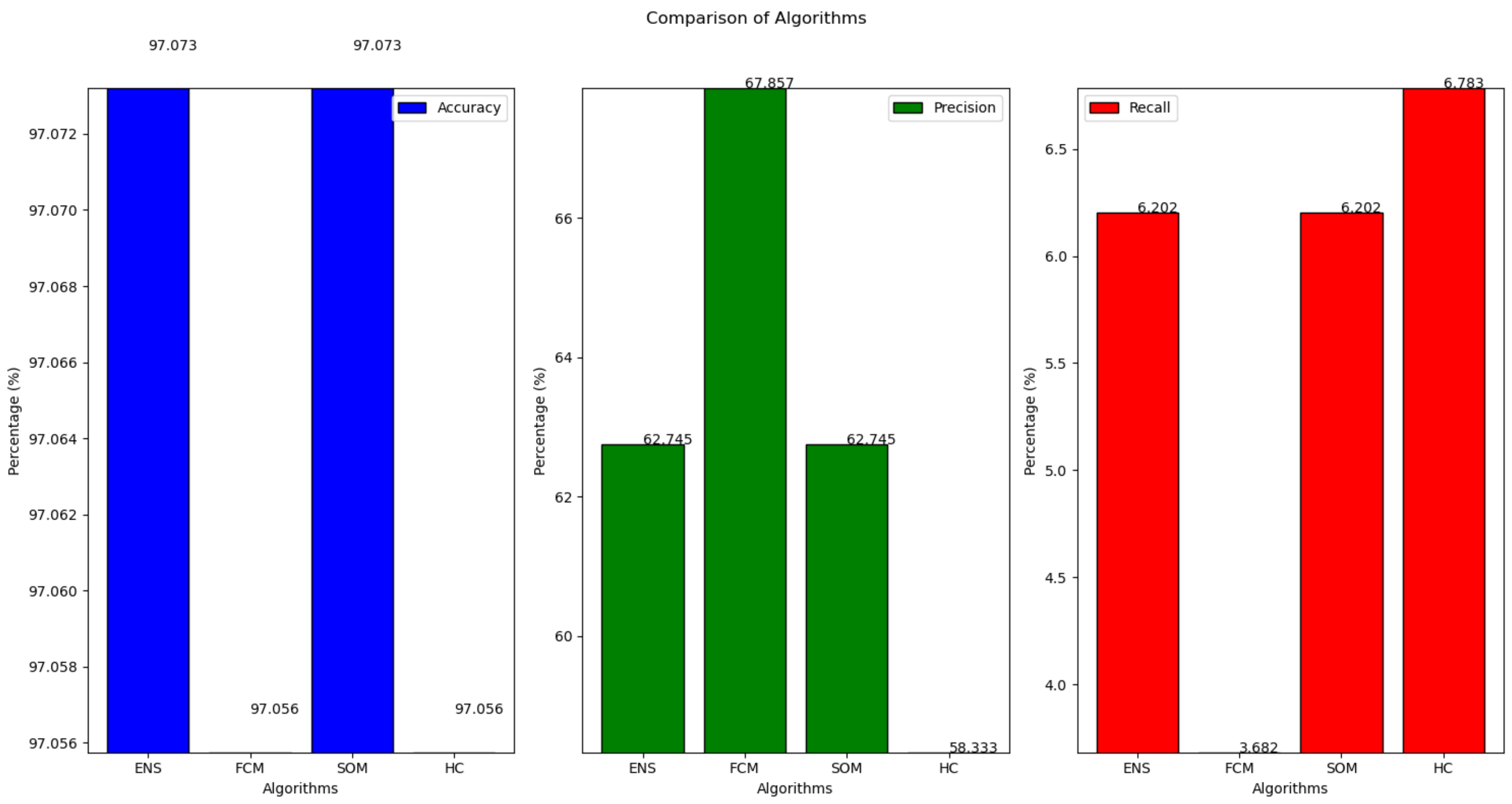


Figure 11. Screenshot of the compare function between each individual algorithm and the ensemble

5.2 Software Qualities

The following sections will be dedicated to discussing the 5 software qualities that we have done in relation to our project.

5.2.1 Robustness

Software robustness is determined by how the algorithm performs when variations are present during testing compared to the dataset we used during the development phase. Such an example of robustness in our program can be shown by the consistency of accuracy of the trained model and the amount of noise present in the dataset.

However, there exist some limitations to our program where the lack of pre-processing of noise present in the raw dataset directly affects the accuracy of the trained model. Due to this limitation, if any noisy dataset is fed into the ensemble, it will be difficult for the trained model to have a high accuracy and precision score.

5.2.2 Security

The security aspect in software development is important as doing so will ensure there aren't any bugs or exploits that allow hackers to inject malicious code or leak personal information. Though, for our project, the security aspect does not apply as the program that we have built does not modify or require users to input any sensitive information, since our program is only requesting datasets as our only input and the datasets requested does not include any personal information that induces a privacy leak towards any entity. Thus, our program does not require any form of security such as encryption or data encapsulation as the data needed from the user for the program to function is only datasets that contain data of a software system that do not contain any identifiable data that can be used against any entity.

5.2.3 Useability

For useability, we have created a GUI which acts as an interface between the user and our program. This GUI allows the user to feed the dataset into the program with a familiar user interface and provides the end result to our user. The GUI improves the usability of our program as it is very easy to interact with the program as opposed to using the command line where some users without a technical background will get confused and need a manual to use the program.

The GUI has also allowed the user to select which algorithms they wanted to train the model. Each radio button that corresponds to an algorithm provides a clear representation of what they meant and users can run the program multiple times by selecting the run button. The result of the model trained and its statistics are shown clearly in a textbox, this provides a very straightforward presentation of the result and does not complicate the process for the user to view its result.

5.2.4 Scalability

In terms of scalability, our program is very open-ended to increasing the number of algorithms in the ensemble. For example, we could add 2 more unsupervised machine learning algorithms into the ensemble and this would likely increase the accuracy of the predictions, however, we would see a trade-off in terms of execution time as it would increase as the number of algorithms increases.

5.2.5 Documentation and Maintainability

Throughout the development process of the program, we have ensured that documentation is done thoroughly by using meaningful function and variable names, while important functions are given docstrings to thoroughly explain what the function does, its input parameters, and its expected output. Each module is also separated into one algorithm per file to ensure the source code is easy and quick to locate.

To allow maintainability throughout the project, we have also aggregated each algorithm to have a unified output format that is easy to follow and parse for the ensemble to compare functions against each other and to debug if necessary.

Our program, which consists of 6 modules, is separated evenly in terms of its responsibility such as GUI or ensemble, is to encapsulate the complex algorithm per module, and modification such as adding or removing an algorithm from the ensemble can be done easily by just commenting out the import line in the ensemble module. This makes it easier for the ensemble to call any algorithms in any of the modules present in the directory without having to understand the underlying logic of the algorithm or implementation.

5.3 Documented Sample Code

Code that is used to perform prediction using SOM and ensemble is included in appendix 2, 3, and 4.

6. Critical Analysis

6.1 Project Execution

Throughout the project, there were a lot of changes based on the knowledge obtained as we researched further and tried to implement libraries we were at first not familiar with. In our initial project design, we were met with the lack of publicly available datasets and stuck with the ones provided in the PROMISE repository. Consulting with our supervisor, we have changed to using the full dataset instead of the previous proposal of splitting the datasets.

However, during implementation, we found that the algorithms, in particular, k-means is failing to perform clustering due to inconsistency of raw data, it surprises us as we have expected that the datasets should not be failing such simple expectation of having irregular data types, the result of which is there exist some data points that was recorded as a question mark instead of numerical. Therefore, we have decided to clean the dataset by replacing all the missing values with ‘0’s and manually review the datasets to ensure that it's consistent and does not contain anything other than float or double data types.

As for the methodology for implementing the ensemble, there are no major changes from the project proposal. As we have originally proposed, we are using three unsupervised learning methods, namely SOM, k-means, and FCM as the initial prototype for the ensemble. The implementation for k-means and FCM was successful but the implementation of SOM prove to be difficult as the source code’s useful output is a graphical representation of the map itself and the weights are aggregated into dimensions that do not match the original dataset, thus after discussing with our supervisor, it was decided to replace SOM for HC. During the implementation of mentioned algorithms, we have also standardized the format of their outputs to allow for easy confusion matrix calculation to evaluate the performance of each algorithm, as well as the ensemble.

Even though different algorithms were implemented as compared to the initial proposal, the ensemble was kept the same as the voting system using different algorithms’ input. We have examined the validity of the ensemble and determined that the ensemble is working as expected with results that have the same or higher performance than any single algorithm. Since we have determined the algorithm integrated into the ensemble without any issues, we have turned back in an attempt to fix SOM by finding another implementation or library that performs the same task. Fortunately, we found out that there is a library for SOM namely sklearn_som, and since now SOM is working, we have decided to swap out k-means in favor of SOM for the ensemble.

Finally, as the backend of the system was done, the team focused on implementing a GUI using PyQt5 in time for the demo presentation and overall presentation to the user.

6.2 Deviation From Proposal

Compared to the initial design that we proposed in the project proposal and the implementation that we have done in this semester, we initially intended to split the dataset into two parts, training which consists of 90% of the original dataset, and testing which consist of 10% of the original dataset. The testing dataset would have been the performance evaluation metrics of each prediction and the combination of which will go through a majority voting process, the results of which will be the final output. This approach was infeasible due to the lack of datasets as the team could only identify 13 datasets to use for testing and training. To that end, it was decided that the test data would also be the training data, while there's a risk of overfitting it was not observed in the results we acquired. Furthermore, initially, it was mentioned that data labels would be removed to simulate unlabelled data. This was consistent with the new design but the labels were used after the algorithms have predicted the labels of each module to calculate the confusion matrix so the team can evaluate the performance of the model.

The initial design also showed that the 3 unsupervised models would be run first before the ensemble, this is slightly changed in the final design as it made little sense to execute each model before the ensemble. The old design would have had each model receive the dataset from the user through the GUI before they passed their results to the ensemble. This was changed to have the ensemble receive the data directly from the GUI when the user selects a dataset then calls the relevant algorithms. This change allows us to only run the necessary modules of code as we left the design very open-ended to being expanded on in terms of the number of algorithms in the ensemble. For example, our current ensemble has 3 algorithms but offers the option to run each algorithm separately. Using our old design it would not be possible to run a module individually as the algorithms would be executed at the same time which would cause extra waiting time during execution.

Lastly, we included some output files for the ensemble when execution finishes, we had no form of outputs in the initial design which only showed the results in terms of accuracy, etc. The three output files allow the user to easily evaluate the results themselves and check which modules have been flagged as potentially faulty. The GUI was also updated in the visual aspect but their functions remain the same.

6.3 Reflection

Throughout the development of the project, there were many issues that arose from a lack of understanding of the relevant material, the most prominent was the initial SOM implementation, which the team could not properly interpret. The output of the SOM was unlike any of the other algorithms and caused issues in trying to convert them to match the standardized output format. The issue was ongoing for two weeks of development and was an overall poor utilization of human resources. Looking back, it was clear that the correct course of action the team should have undertaken was to look elsewhere for a different solution instead of spending all that time on trying to fix a single algorithm that was simply replaceable, furthermore, the algorithm also did not meet the team's runtime requirements so it should have been scrapped way earlier. The resolution the team reached was to replace SOM with HC as suggested by our supervisor but the wasted time would come back to push back future tasks.

Another issue is the poor definition of tasks in the initial project proposal. The team had not planned ahead in terms of the confusion matrix calculations, it was only touched upon after the supervisor mentioned it to us, which caused further delays in future tasks. But this issue was a smaller one as the team came up with a standardized method of confusion matrix calculation that could be used for all the algorithms. However, it is still worth noting that it should have been in the initial project proposal as it is the method by which the performance of the program is evaluated.

Overall, the team made decisions based on the tasks and challenges that appeared during the developmental phase and managed to overcome all of the setbacks to deliver the final product on time, while there were hiccups along the way we believe that we solved most issues within a timely manner and stayed mostly on track. In the future, the team will be more experienced and can be more prepared for such obstacles should they be encountered again.

7. Conclusion

In conclusion, we have successfully developed our program for software fault prediction using unsupervised learning methods, in which we are able to predict faulty modules using the dataset containing the statistics of a given codebase. We have created a GUI that allows users to input the dataset with or without labels, allowing the user to choose any of the available algorithms or the ensemble. After obtaining all the necessary options, the program will execute according to its parameters and produce a result containing the statistics of the trained model, namely Accuracy, Precision, and Recall.

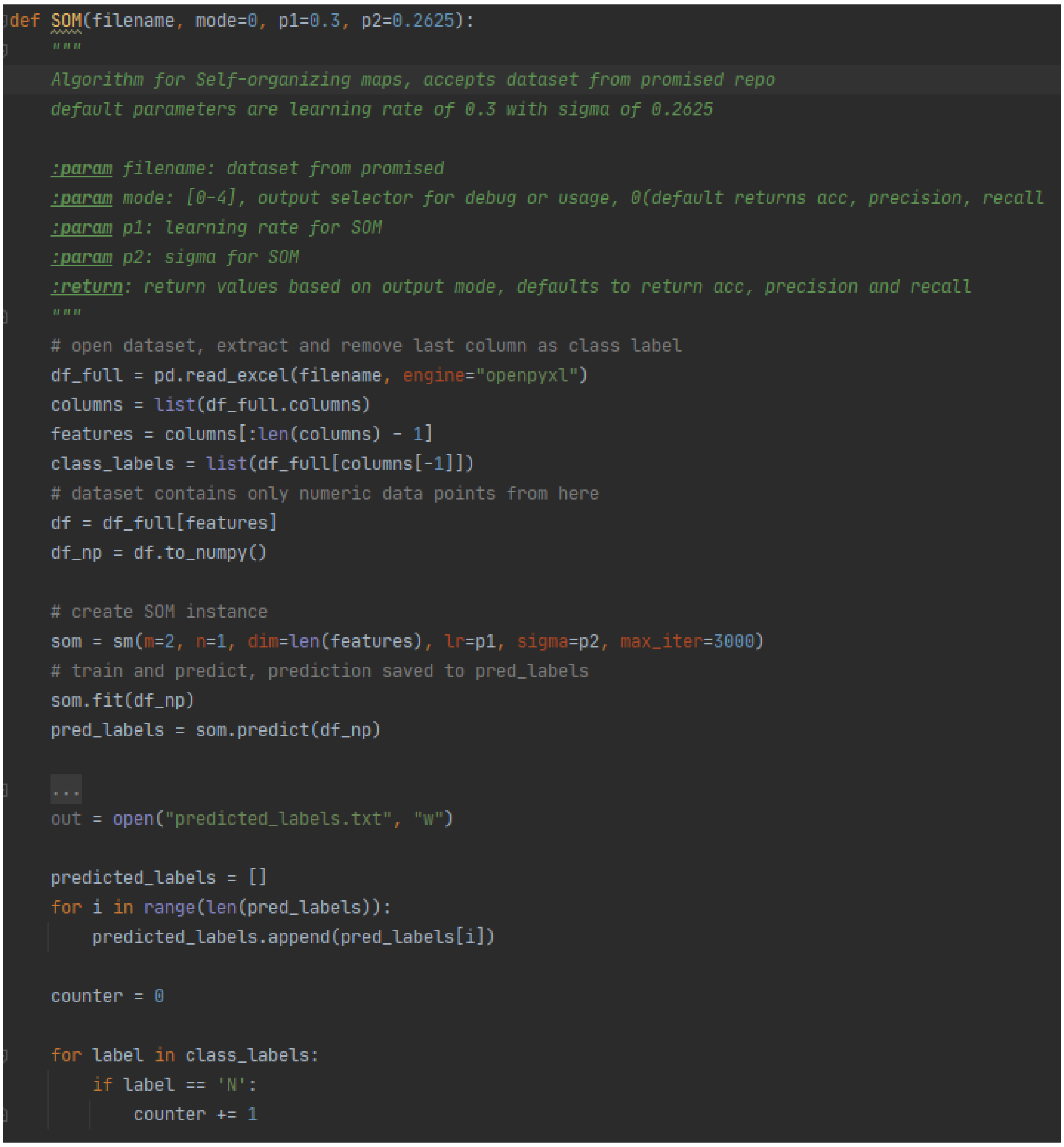
Overall, the program is able to produce a result within a reasonable time no longer than a few minutes. However, the lack of pre-processing of the noisy datasets and the lack of options for users to customize the parameters of any specific algorithms paved the way for future works that the program can use to improve further. This includes implementing a pre-processing method to minimize the impact of noisy data and an additional textbox to pass custom parameters to algorithms instead of using the predefined parameters that we have coded in.

Throughout this project, we have adapted the Agile methodology where we conduct meetings each week to discuss and work on the prototype, continuously refining and adding new algorithms into the prototype until all features mentioned in the project proposal are implemented. The application of unsupervised learning such as SOM and FCM has shown us how these complex algorithms work using clustering to build a prediction model. Our models have performed quite well, averaging 98% on clean datasets and 85% to 90% on average using datasets with low noise. Hence, using unsupervised learning training methods, we are able to successfully create a working system to predict faulty software modules when labeled data is not present.

8. Appendix

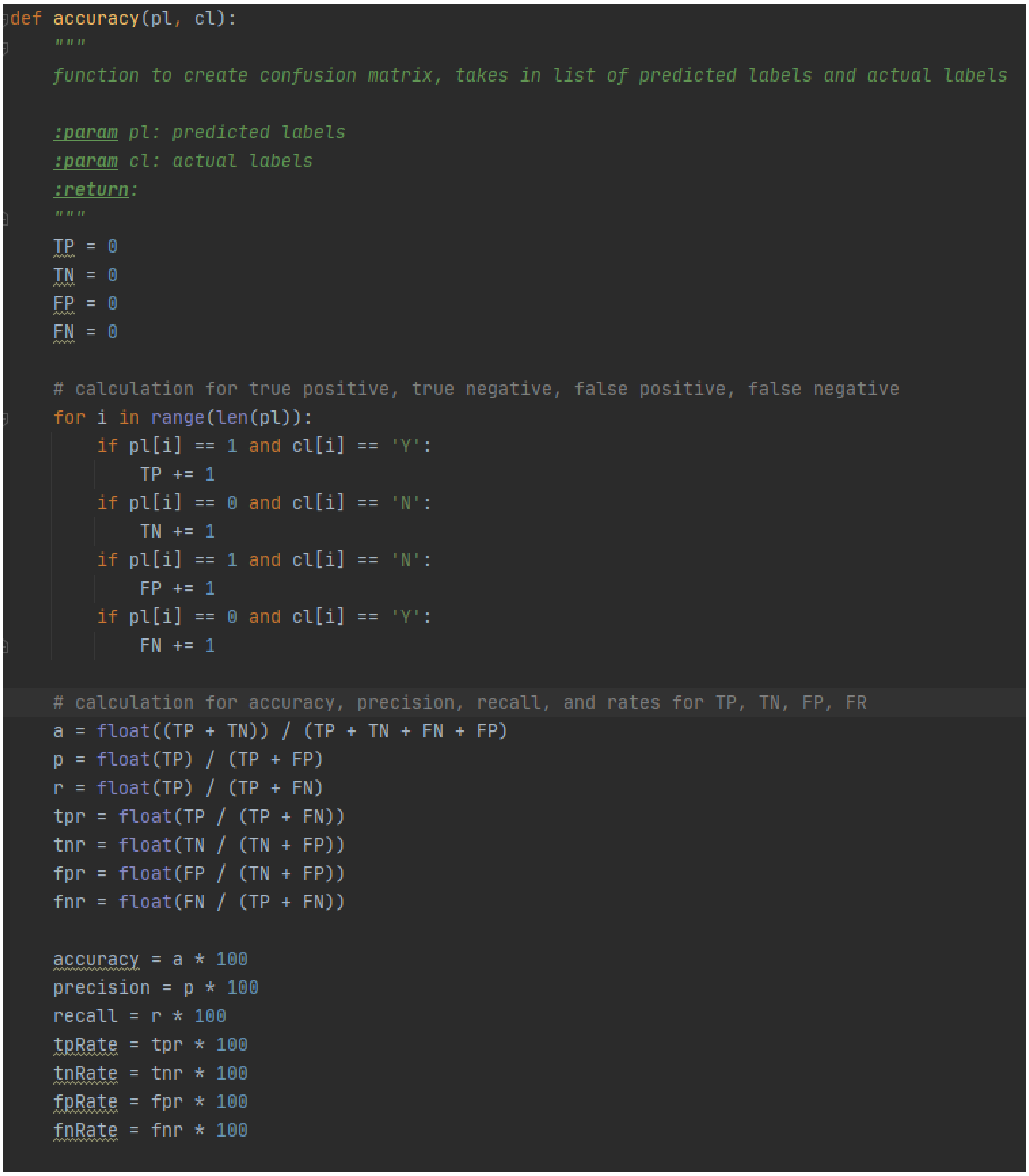
Member	**Contribution**
Xiao Cong Wu	35%
Charles Tan Wei Wen	35%
Shafkat Ibrahimy	30%

Appendix 1: Team Member' Contribution



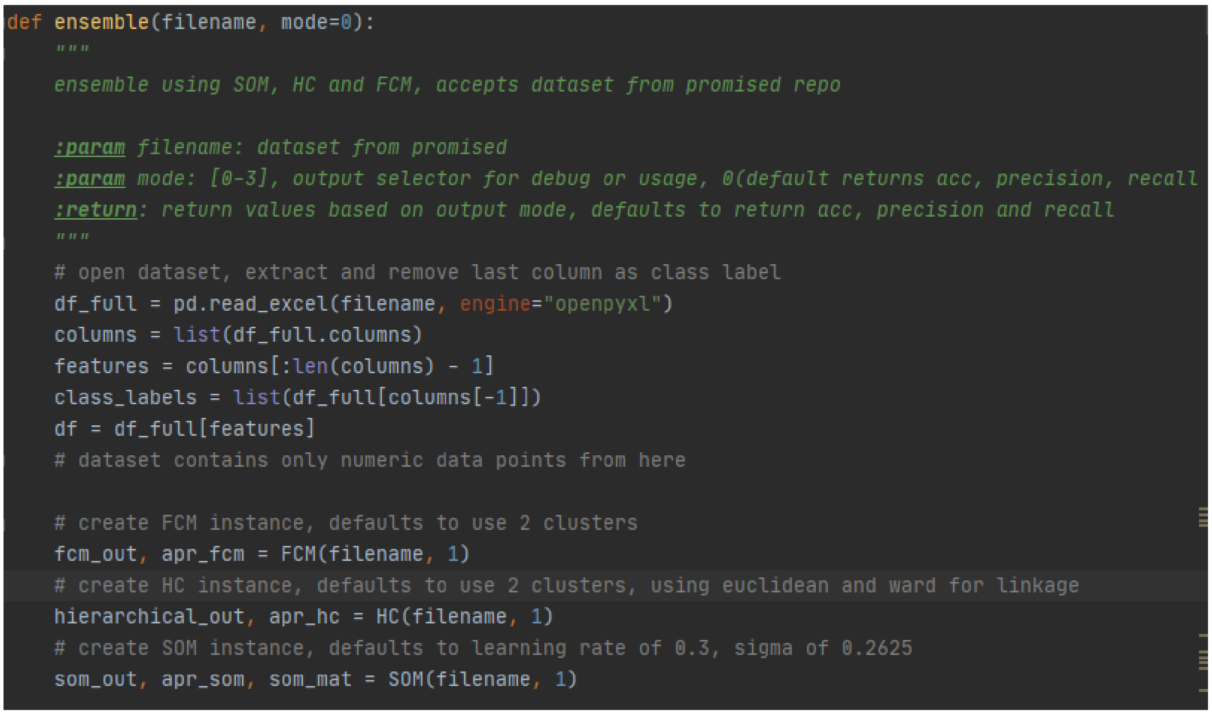
```
def SOM(filename, mode=0, p1=0.3, p2=0.2625):
    """
    Algorithm for Self-organizing maps, accepts dataset from promised repo
    default parameters are learning rate of 0.3 with sigma of 0.2625

    :param filename: dataset from promised
    :param mode: [0-4], output selector for debug or usage, 0(default returns acc, precision, recall
    :param p1: learning rate for SOM
    :param p2: sigma for SOM
    :return: return values based on output mode, defaults to return acc, precision and recall
    """
    # open dataset, extract and remove last column as class label
    df_full = pd.read_excel(filename, engine="openpyxl")
    columns = list(df_full.columns)
    features = columns[:len(columns) - 1]
    class_labels = list(df_full[columns[-1]])
    # dataset contains only numeric data points from here
    df = df_full[features]
    df_np = df.to_numpy()

    # create SOM instance
    som = sm(m=2, n=1, dim=len(features), lr=p1, sigma=p2, max_iter=3000)
    # train and predict, prediction saved to pred_labels
    som.fit(df_np)
    pred_labels = som.predict(df_np)

    ...
    out = open("predicted_labels.txt", "w")

    predicted_labels = []
    for i in range(len(pred_labels)):
        predicted_labels.append(pred_labels[i])

    counter = 0

    for label in class_labels:
        if label == 'N':
            counter += 1
```

Appendix 2: SOM Code



```python
def accuracy(pl, cl):
    """
    function to create confusion matrix, takes in list of predicted labels and actual labels

    :param pl: predicted labels
    :param cl: actual labels
    :return:
    """
    TP = 0
    TN = 0
    FP = 0
    FN = 0

    # calculation for true positive, true negative, false positive, false negative
    for i in range(len(pl)):
        if pl[i] == 1 and cl[i] == 'Y':
            TP += 1
        if pl[i] == 0 and cl[i] == 'N':
            TN += 1
        if pl[i] == 1 and cl[i] == 'N':
            FP += 1
        if pl[i] == 0 and cl[i] == 'Y':
            FN += 1

    # calculation for accuracy, precision, recall, and rates for TP, TN, FP, FR
    a = float((TP + TN)) / (TP + TN + FN + FP)
    p = float(TP) / (TP + FP)
    r = float(TP) / (TP + FN)
    tpr = float(TP / (TP + FN))
    tnr = float(TN / (TN + FP))
    fpr = float(FP / (TN + FP))
    fnr = float(FN / (TP + FN))

    accuracy = a * 100
    precision = p * 100
    recall = r * 100
    tpRate = tpr * 100
    tnRate = tnr * 100
    fpRate = fpr * 100
    fnRate = fnr * 100
```

Appendix 3: Confusion Matrix code



```python
def ensemble(filename, mode=0):
    """
    ensemble using SOM, HC and FCM, accepts dataset from promised repo

    :param filename: dataset from promised
    :param mode: [0-3], output selector for debug or usage, 0(default returns acc, precision, recall
    :return: return values based on output mode, defaults to return acc, precision and recall
    """
    # open dataset, extract and remove last column as class label
    df_full = pd.read_excel(filename, engine="openpyxl")
    columns = list(df_full.columns)
    features = columns[:len(columns) - 1]
    class_labels = list(df_full[columns[-1]])
    df = df_full[features]
    # dataset contains only numeric data points from here

    # create FCM instance, defaults to use 2 clusters
    fcm_out, apr_fcm = FCM(filename, 1)
    # create HC instance, defaults to use 2 clusters, using euclidean and ward for linkage
    hierarchical_out, apr_hc = HC(filename, 1)
    # create SOM instance, defaults to learning rate of 0.3, sigma of 0.2625
    som_out, apr_som, som_mat = SOM(filename, 1)
```

Appendix 4. ensemble code calling SOM, HC, and FCM

9. References

Sammar.M, Mustafa.Y, Nagwa.E, Mohamed,S(2018). Software bug prediction using weighted majority voting techniques, Alexandria Engineering Journal, from https://doi.org/10.1016/j.aej.2018.01.003.

University of Ottawa. (n.d.). *PROMISE SOFTWARE ENGINEERING REPOSITORY*. Retrieved 27 May 2021, from http://promise.site.uottawa.ca/SERepository/datasets-page.html

Lutins, E. (2021). *Ensemble Methods in Machine Learning: What are They and Why Use Them?*. Towards Data Science. Retrieved 27 May 2021, from https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f.

Dabbura, I. (2021). *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*. Towards Data Science. Retrieved 27 May 2021, from https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

Self-Organizing Map - an overview | ScienceDirect Topics. Sciencedirect.com. (2021). Retrieved 27 May 2021, from https://www.sciencedirect.com/topics/engineering/self-organizing-map.

Chatterjee, M. (2021). *What is Spectral Clustering and how its work?*. GreatLearning Blog: Free Resources what Matters to shape your Career!. Retrieved 27 May 2021, from https://www.mygreatlearning.com/blog/introduction-to-spectral-clustering/.

Li, N., Shepperd, M., & Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. *Information And Software Technology*, *122*, 106287. https://doi.org/10.1016/j.infsof.2020.106287

Bai, L., Liang, J., & Cao, F. (2020). A multiple k-means clustering ensemble algorithms to find nonlinearly separable clusters. *Information Fusion*, *61*, 36-47. https://doi.org/10.1016/j.inffus.2020.03.009

Zhang, Z., Jing, X., & Wu, F. (2018). Low‐rank representation for semi‐supervised software defect prediction. *IET Software*, *12*(6), 527-535. https://doi.org/10.1049/iet-sen.2017.0198

Boucher, A., & Badri, M. (2018). Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information And Software Technology*, *96*, 38-67. https://doi.org/10.1016/j.infsof.2017.11.005

Zhang, F., Zheng, Q., Zou, Y., & Hassan, A. (2016). Cross-project defect prediction using a connectivity-based unsupervised classifier. *Proceedings Of The 38Th International Conference On Software Engineering*. https://doi.org/10.1145/2884781.2884839

Laradji, I., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information And Software Technology*, *58*, 388-402. https://doi.org/10.1016/j.infsof.2014.07.005

Abaei, G., Selamat, A., & Fujita, H. (2015). An empirical study based on a semi-supervised hybrid self-organizing map for software fault prediction. *Knowledge-Based Systems*, *74*, 28-39. https://doi.org/10.1016/j.knosys.2014.10.017

Abaei, G., Rezaei, Z., & Selamat, A. (2013). Fault prediction by utilizing a self-organizing Map and Threshold. *2013 IEEE International Conference On Control System, Computing And Engineering*. https://doi.org/10.1109/iccsce.2013.6720010

Bishnu, P., & Bhattacherjee, V. (2012). Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm. *IEEE Transactions On Knowledge And Data Engineering*, *24*(6), 1146-1150. https://doi.org/10.1109/tkde.2011.163